THE POWER TO KNOW.

# SAS® 9.4 Scalable Performance Data Engine: Reference, Fourth Edition

SAS® Documentation

The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2016. *SAS® 9.4 Scalable Performance Data Engine: Reference, Fourth Edition*. Cary, NC: SAS Institute Inc.

**SAS® 9.4 Scalable Performance Data Engine: Reference, Fourth Edition**

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

November 2016

9.4-P1:engspde

# Contents

# About This Book

## 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:

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

# What’s New in SAS 9.4 Scalable Performance Data Engine

## Overview

The following are new or enhanced options for 9.4:

- ALIGN= data set option
- COMPRESS= LIBNAME statement option
- ENCRYPT=AES (Advanced Encryption Standard) data set option
- ENCRYPTKEY= data set option
- IOBLOCKSIZE= LIBNAME statement option
- SPDEFILECACHE system option
- In the first maintenance release for SAS 9.4, SPD Engine does not support DLDMGACTION=NOINDEX, but does support ABORT, FAIL, PROMPT, and REPAIR.
- In the third maintenance release for SAS 9.4, a new section named “Accessing SPD Engine Files on Another Host” on page 3 was added.

## SPD Engine System Option

A new SPD Engine system option enables you to do the following:

- The SPDEFILECACHE system option enables or disables caching of opened SPD Engine data files. For more information, see “SPDEFILECACHE System Option” on page 99.

## SPD Engine Data Set Options

New and enhanced SPD Engine data set options enable you to do the following:

- The new ALIGN= data set option specifies variable alignment. For more information, see “ALIGN= Data Set Option” on page 50.

- The ENCRYPT= data set option has been enhanced to include the AES algorithm for stronger security. For more information, see "Encrypting SPD Engine Data Sets" on page 21.
- The new ENCRYPTKEY= data set option specifies the key value for AES encryption. For more information, see "ENCRYPTKEY= Data Set Option" on page 65.

## SPD Engine LIBNAME Statement Options

The new LIBNAME statement options enable you to do the following:

- In the second maintenance release for SAS 9.4, the new IOBLOCKSIZE= LIBNAME statement option enables you to specify the size in bytes of a block of observations to be used in an I/O operation. For more information, see "IOBLOCKSIZE= LIBNAME Statement Option" on page 39.
- In the second maintenance release for SAS 9.4, the COMPRESS= LIBNAME statement option enables you to compress an SPD Engine data set on disk as it is being created.

# *Chapter 1*
# Overview: The SPD Engine

# Introduction to the SPD Engine

The SPD Engine is designed for high-performance data delivery. It enables an application rapid access to SAS data for processing. The SPD Engine delivers data to applications rapidly because it organizes the data into a streamlined file format that takes advantage of multiple CPUs to perform parallel input functions.

The SPD Engine uses *threads* to read blocks of data very rapidly and in parallel. Tasks are performed in conjunction with an operating system that enables threads to execute on any of the computer’s available CPUs. Although threaded Read tasks are an important part of the SPD Engine functionality, the real power of the SPD Engine comes from how it structures SAS data. The SPD Engine organizes data into a file format that includes partitioning the data. This data structure permits threads, running in parallel, to perform Read tasks efficiently.

The SPD Engine is a high-speed alternative to the default Base SAS engine for processing very large data sets. It reads data sets that contain billions of observations. For example, this includes data sets that expand beyond the size limit imposed by some operating systems and data sets that SAS analytic software and procedures must process faster.

The SPD Engine boosts performance in the following ways:

- support for hundreds of gigabytes of data
- scalability on symmetric multiprocessor (SMP) computers and massively parallel processor (MPP) computers
- parallel WHERE selections
- parallel loads
- parallel index creation
- parallel data delivery to applications
- automatic sorting on BY statements

The SPD Engine runs on UNIX, Windows, and z/OS (zFS file system only).

*Note:* Be sure to visit the Scalability and Performance Community focus area at http://support.sas.com/rnd/scalability for more information about scalability. For system requirements, visit the Install Center at http://support.sas.com/documentation/installcenter.

# SPD Engine Compatibility

## Upgrading SAS 9

If you upgrade from a previous release of SAS 9, you do not need to migrate your data sets if you stay in the same operating environment. If you upgrade across hosts, such as from a 32-bit to a 64-bit Windows operating environment, you must migrate your data sets. To migrate your data sets, use CIMPORT, COPY, or CPORT. For more information, see the *Base SAS Procedures Guide*.

*Note:* The MIGRATE procedure does not support the SPD Engine.

### *Accessing SPD Engine Files on Another Host*

The SPD Engine cannot share data sets across host families, but it can share data sets within a host family. For example, the engine can access (Read and Update) SPD Engine files on another host as long as both hosts have the same data representation.

The SPD Engine runs on the following hosts:

HP-UX IPF or HP-UX on Itanium
Linux on x64 (x86-64)
MVS_32
AIX 64-bit or AIX on Power
Solaris 64-bit64 or Solaris on SPARC
Solaris 10 on x64 (x86-64)
Windows 32-bit
Windows 64-bit

Based on their data representation, data sets can be separated into the following host families:

- big endian that runs on the following:
  - HP-UX IPF or HP-UX on Itanium
  - AIX 64-bit or AIX on Power
  - Solaris 64-bit or Solaris on SPARC
- little endian that runs on the following:
  - Linux on x64 (x86-64)
  - Solaris 10 on x64 (x86-64)
- z/OS that runs on z/OS 32-bit
- Windows that runs on Windows 32-bit
- Windows x64 that runs on Windows 64-bit

## Using the SMP Computer

The SPD Engine exploits a hardware and software architecture known as symmetric multiprocessing. An SMP computer has multiple central processing units (CPUs) and an operating system that supports threads. An SMP computer is usually configured with multiple controllers and multiple disk drives per controller. When the SPD Engine reads a data file, it launches one or more threads for each CPU; these threads then read data in parallel from multiple disk drives, driven by one or more controllers per CPU. The SPD Engine running on an SMP computer provides the capability to read and deliver much more data to an application in a given elapsed time.

Reading a data set with an SMP computer that has 5 CPUs and 10 disk drives could be as much as 5 times faster than I/O on a single-CPU computer. In addition to threaded I/O, an SMP computer enables threading of application processes (for example, threaded sorting in the SORT procedure in SAS 9.1 or later).

The exact number of CPUs on an SMP computer varies by manufacturer and model. The operating system of the computer is also specialized; it must be capable of scheduling code segments so that they execute in parallel. If the operating system kernel is threaded, performance is further enhanced because it prevents contention between the executing threads.

As threads run on the SMP computer, managed by a threaded operating system, the available CPUs work together. The synergy between the CPUs and threads enables the software to scale processing performance. The scalability, in turn, significantly increases overall processing speed for tasks such as creating data sets, appending data, and querying the data by using WHERE statements.

# Organizing SAS Data Using the SPD Engine

## *How the SPD Engine Organizes SAS Data*

Because the SPD Engine organizes data for high-performance processing, an SPD Engine data set is physically different from a default Base SAS engine data set. The default Base SAS engine stores data in a single data file that contains both data and data descriptors for the file (metadata). The SPD Engine creates separate files for the data and data descriptors. In addition, if the data set is indexed, the SPD Engine creates two index files for each index. Each of these four files is called an SPD Engine *component file*, and each has an identifier embedded in the filename.

The metadata component is a single physical file, but it can occupy multiple physical files, and each file has .mdf embedded in the filename. The data component is one or more physical files, and each file has .dpf embedded in the filename. If the index component exists because indexes have been defined, each index has two physical files:

- one file with .hbx embedded in the filename
- one file with .idx embedded in the filename

Each of these component files can consist of one or more physical files so that the component can span volumes, but be referenced as one logical file. For example, the SPD Engine can create many physical files containing data, but it references the files containing data as a single data component in an SPD Engine data set.

The *metadata* and *index* components differ from the *data* component in two ways:

1. You can specify a fixed-length partition size for data component files using the PARTSIZE= option. You cannot specify the partition size for the metadata or index components.
2. The data component files are created in a cyclical fashion across all defined paths. The metadata and index component files are created in a single defined path until that path is full, and then the next defined path is used.

## *Metadata Component Files*

A SPD Engine data set stores the descriptive metadata in a file with the file extension .mdf. Usually, an SPD Engine data set has only one .mdf file. Its .mdf file includes the pathnames of all of its other component files.

## *Index Component Files*

If the file is indexed, the SPD Engine creates two index files for each index. Each of these files contains a particular view of the index.

- The index file with the .hbx file extension contains the global index.
- The index file with the .idx file extension contains the segment index.

### Data Component Files

The data component of an SPD Engine data set can be several files (partitions) per path or device, rather than just one. Each of these partitions is a fixed length, specified when you create the SPD Engine data set.

Specifying a partition size for the data component files enables you to tune the performance of your applications. The partitions are the threadable units, that is, each partition (file) is read in one thread. Chapter 2, “Creating and Loading SPD Engine Files,” on page 11 provides details about how the SPD Engine stores data, metadata, and indexes.

## Comparing the Default Base SAS Engine and the SPD Engine

### Overview of Comparisons

Default Base SAS engine data sets and SPD Engine data sets have many similarities. They both store data in a SAS library, which is a logical collection of files. Because the SPD Engine data libraries can span devices and file systems, the SPD Engine is ideal for use with very large data sets. Also, the SPD Engine enables you to specify separate directories, or devices, for each component in the LIBNAME statement. Chapter 2, “Creating and Loading SPD Engine Files,” on page 11 provides details about designing and setting up the SPD Engine data libraries.

### The SPD Engine Libraries and File Systems

An SPD Engine library can contain data files, metadata files, and index files. The SPD Engine does not support catalogs, SAS views, MDDBs, or other utility (byte) files.

The SPD Engine uses the zFS file system for z/OS.

### Utility File Workspace

Utility files are generated during the SPD Engine operations that need extra space (for example, when creating parallel indexes or when sorting very large files). Default locations exist for all platforms but, if you have large amounts of data to process, the default location might not be large enough. The SPD Engine system option SPDEUTILLOC= lets you specify a set of file locations in which to store utility scratch files. For more information, see “SPDEUTILLOC= System Option” on page 102.

### Storing Temporary Data Sets

To create a library to store interim data sets, specify the SPD Engine option TEMP= in the LIBNAME statement. If you want current applications to refer to these interim files using one-level names, specify the library on the USER= system option.

The following example code creates a user libref for interim data sets. It is deleted at the end of the session.

```
libname user spde 'SAS-library' temp=yes;
data a; x=1;
run;
proc print data=a;
```

The USER= option can be set in the configuration file so that applications that reference interim data sets with one-level names can run in the SPD Engine.

### Differences between the Default Base SAS Engine Data Sets and the SPD Engine Data Sets

The following chart compares the SPD Engine capabilities to default Base SAS engine capabilities.

***Table 1.1*** *Comparing the Default Base SAS Engine Data Sets and the SPD Engine Data Sets*

| Feature | SPD Engine | Default Base SAS Engine |
|---|---|---|
| Partitioned data sets | yes | no |
| Parallel WHERE optimization | yes | no |
| Lowest locking level | member | record |
| Concurrent access from multiple SAS sessions on a given data set | READ (INPUT Open mode) | READ and WRITE (all Open modes) |
| Remote computing via SAS/CONNECT | no | yes |
| Data transfer via SAS/CONNECT | no | yes |
| RLS (Remote Library Services) via SAS/CONNECT | no | yes |
| Available via SAS/CONNECT | no | yes |
| Support in SAS/SHARE | no | yes |
| Automatic sort for SAS BY processing (sort a temporary copy of the data to support BY processing) | yes | no |
| User-defined formats and informats | yes, except in WHERE* | yes |
| Catalogs | no | yes |
| Views | no | yes |
| MDDBs | no | yes |

| Feature | SPD Engine | Default Base SAS Engine |
|---|---|---|
| Integrity constraints | no | yes |
| Data set generations | no | yes |
| CEDA | no | yes |
| Audit trail | no | yes |
| NLS transcoding | no | yes |
| COMPRESS= | YES \| NO \| CHAR \| BINARY (only if the file is not encrypted) | YES \| NO \| CHAR \| BINARY |
| DLCREATEDIR | no | yes |
| ENCRYPT= | cannot be used with COMPRESS= | can be used with COMPRESS= |
| encryption | yes | yes |
| AES encryption | data and index files only | yes |
| FIRSTOBS= system option and data set option | no | yes |
| OBS= system option and data set option | yes, if used without ENDOBS= or STARTOBS= SPD Engine options | yes |
| EXTENDOBSCOUNTER= system option and data set option | no | yes |
| Extended attributes | no | yes |
| Functions and call routines | yes, with some exceptions. | yes |
| Move table via OS utilities to a different directory or folder | no | yes |
| Observations returned in physical order | no, if BY or WHERE is present | yes |
| DLDMGACTION= system option and data set option | yes, with ABORT \| FAIL \| PROMPT \| REPAIR, but not with NOINDEX | yes |

* In WHERE processing, functions and call routines are passed to the supervisor for handling. Therefore, they are not processed in parallel.

## Interoperability of the Default Base SAS Engine and the SPD Engine Data Sets

Default Base SAS engine data sets must be converted to the SPD Engine format so that the SPD Engine can access them. You can convert the default Base SAS engine data sets easily using the COPY procedure, the APPEND procedure, or a DATA step. (PROC MIGRATE cannot be used.) In addition, most of your existing SAS programs can run on the SPD Engine files with little modification other than to the LIBNAME statement. Chapter 2, “Creating and Loading SPD Engine Files,” on page 11 provides details about converting default Base SAS engine data sets to the SPD Engine format.

## Sharing the SPD Engine Files

The SPD Engine supports member-level locking, which means that multiple users can have the same SPD Engine data set open for INPUT (read-only). However, if an SPD Engine data set has been opened for update or for index creation, then only that user can access it.

## Features That Enhance I/O Performance

### *Overview of I/O Performance Enhancements*

The SPD Engine has several features that enhance I/O performance. These features can dramatically increase the performance of I/O bound applications, in which large amounts of data must be delivered to the application for processing.

### *Multiple Directory Paths*

You can specify multiple directory paths and devices for each component type because the SPD Engine can reference multiple physical files across volumes as a single logical file. For very large data sets, this feature circumvents any file size limits that the operating system might impose.

### *Physical Separation of the Data File and the Associated Indexes*

Because each component file type can be stored in a different location, file dependencies are not a concern when deciding where to store the component files. Only cost, performance, and availability of disk space need to be considered.

### *WHERE Optimization*

The SPD Engine automatically determines the best method to use to evaluate observations for qualifying criteria specified in a WHERE statement. WHERE statement efficiency depends on such factors as whether the variables in the expression are indexed. A WHERE evaluation planner is included in the SPD Engine. It can choose the best method to use to optimize evaluation of WHERE expressions that use indexes.

# Features That Boost Processing Performance

### *Automatic Sort Capabilities*

The SPD Engine's automatic sort capabilities save time and resources for SAS applications that process large data sets. With the SPD Engine, you do not need to invoke the SORT procedure before you submit a SAS statement with a BY clause. When the SPD Engine encounters a BY clause and the data is not already sorted or indexed on the BY variable, the SPD Engine automatically sorts the data without affecting the permanent data set or producing a new output data set.

### *Queries Using Indexes*

Large data sets can be indexed to maximize performance. Indexes permit rapid WHERE expression evaluations for indexed variables. The SPD Engine takes advantage of multiple CPUs to search the index component file efficiently.

*Note:* You cannot create an index or composite index on a variable if the variable name contains any of the following special characters (even with the VALIDMEMNAME=EXTEND option):

```
" * | \ : / < > ? - .
```

### *Parallel Index Creation*

In addition, the SPD Engine supports parallel index creation so that indexing large data sets is not time-consuming. The SPD Engine decomposes data set Append or Insert operations into a set of steps that can be performed in parallel. The level of parallelism depends on the number of indexes present in the data set. The more indexes you have, the greater the exploitation of parallelism during index creation. However, index creation requires utility file space and memory resources.

*Note:* You cannot create an index or composite index on a variable if the variable name contains any of the following special characters (even with the VALIDMEMNAME=EXTEND option):

```
" * | \ : / < > ? - .
```

# The SPD Engine Options

The SPD Engine works with many default Base SAS engine options. In addition, there are options that are used only with the SPD Engine that enable you to further manage the SPD Engine libraries and processing. See:

- SPD Engine Data Set Options on page 47
- SPD Engine LIBNAME Statement Options on page 27
- SPD Engine System Options on page 93

# *Chapter 2*
# Creating and Loading SPD Engine Files

# Introduction for Creating and Loading SPD Engine Files

This section provides details about allocating SPD Engine libraries and creating and loading SPD Engine data and indexes. Performance considerations related to these tasks are also discussed.

# Allocating the Library Space

## *How to Allocate the Library Space*

To realize performance gains through SPD Engine's partitioned data read and threading capabilities, the SPD Engine libraries must be properly configured and managed. Optimally, a SAS system administrator performs these tasks.

An SPD Engine data set requires a file system with enough space to store the various component files. Often that file system includes multiple directories for these components. Usually, a single directory path (part of a file system) is constrained by a volume limit for the file system as a whole. This limit is the maximum amount of disk space configured for the file system to use.

Within this maximum disk space, you must allocate enough space for all of the SPD Engine component files. Understanding how each component file is handled is critical to estimating the amount of storage that you need in each library.

## *Configuring Space for All Components in a Single Path*

In the simplest SPD Engine library configuration, all of the SPD Engine component files (data files, metadata files, and index files) can reside in a single path called the *primary path*. The primary path is the default path specification in the LIBNAME statement. The following LIBNAME statement sets up the primary file system for the MyLib library:

```
libname mylib spde '/disk1/spdedata';
```

Because there are no other path options specified, all component files are created in this primary path. Storing all component file types in the primary path is simple and works for very small data sets. It does not take advantage of the performance boost that storing components separately can achieve, nor does it take advantage of multiple CPUs.

*Note:* The SPD Engine requires complete pathnames to be specified.

## *Configuring Separate Library Space for Each Component File Type*

Most sites use the SPD Engine to manage very large amounts of data, which can have thousands of variables and some of them indexed. At these sites, separate storage paths are usually defined for the various component types. In addition, using disk striping and RAID (Redundant Array of Independent Disks) can be very efficient. For more information, see "SPD Engine Disk I/O Setup" in Scalability and Performance at http://support.sas.com/rnd/scalability/spde/setup.html.

The metadata component files for all data sets in a library must reside in the primary path.

In addition, specifying separate paths for the data component files and index component files provides performance gains. You specify separate paths because the Read load is distributed across disk drives. Separating the data and index component files helps prevent disk contention and increases the level of *parallelism* that can be achieved, especially in complex WHERE evaluations. The following example code specifies a primary path for the metadata component files. The code uses the DATAPATH= option on page 34 and the INDEXPATH= option on page 38 to specify additional, separate paths for the data and index component files:

```
libname all_users spde '/disk1/metadata'
   datapath= ('/disk2/userdata' '/disk3/userdata')
   indexpath= ('/disk4/userindexes' '/disk5/userindexes');
```

The metadata component files are stored on disk1, which is the primary path. The data component files are on disk2 and disk3, and the index component files are on disk4 and disk5. For all path specifications, you must specify the complete pathname.

***CAUTION:***

**The primary path must be unique for each library.** If two librefs are created with the same primary path but with differences in the other paths, data can be lost. You cannot use NFS in any path other than the primary path.

*Note:* If you are planning to store data in locally mounted drives and to access the data from a remote computer, use the remote pathname when you specify the LIBNAME. If **`/data01`** and **`/data02`** are locally mounted drives on the localA computer, use the pathnames **`/nfs/localA/data01`** and **`/nfs/localA/data02`** in the LIBNAME statement.

### *Anticipating the Space for Each Component File*

To properly configure the SPD Engine library space, you need to understand the relative sizes of the SPD Engine component files. The following information provides a general overview. For more information, see “SPD Engine Disk I/O Setup” in Scalability and Performance at http://support.sas.com/rnd/scalability/spde/setup.html.

Metadata component files are relatively small files, but the primary path that you specify must be large enough to contain all of the metadata component files for the library. Metadata component files cannot grow beyond the space available in the path.

Index component files (both .idx and .hbx) can be medium to large, depending on the number of distinct values in each index and whether the index is a single or composite index. When an index component file grows beyond the space available in the current file path, a new index component file is created in the next path.

Data component files can be numerous, depending on the amount of data and the partition size specified for the data set. Each data partition is stored as a separate data component file. The size of the data partition is specified in the PARTSIZE= LIBNAME statement option on page 41 or in the PARTSIZE= data set option on page 78. Data component files are the only component files for which you can specify a partition size.

## Storage of the Metadata Component Files

### Metadata Component Files

The metadata component file for an SPD Engine data set stores the descriptive information about the data set and the pathnames to its constituent data component files and index component files. This concept is very important to understand because it directly affects whether you can add data sets (with their associated metadata component files) to the library.

The metadata component files for all of the data sets in a library must reside in the same location specified in the primary path. In effect, the files in the primary path act like a directory to the entire library. When an SPD Engine data set is accessed, the SPD Engine first opens the data set's metadata component file to determine its attributes and to determine whether it can access all of its other component files. If a new data set for a library is created, and the space in the primary path is full, the SPD Engine cannot begin creating the metadata component file in that path, and the Create operation fails with an appropriate error message. To successfully create a new data set in this case, you must either free space in the primary path or assign a new library and copy some or all of the data sets to the new library. Data component files and index component files do not have that limitation. You can specify additional space at a later time for data component files and index component files.

Certain actions cause metadata component files to grow to exceed the file size or space limitations. In that case, the SPD Engine creates another partition of the metadata component file to accommodate the overflow. New metadata partitions can reside in the primary path or in the paths specified in the "METAPATH= LIBNAME Statement Option" on page 40.. You cannot use the METAPATH= option to create space for a new data set's first metadata partition. The METAPATH= option specifies space for only metadata component files beyond the first one.

### Storage of the Index Component Files

An index component file is stored based on overflow space. When an index component file grows to exceed the file size or space limitations, the SPD Engine creates another partition of the index component file to accommodate the overflow. When several file paths are specified with the INDEXPATH= option, index component files are created in the first available space, and then they overflow to the next path when the previous path is filled. Unlike metadata component files, index component files do not have to be in the primary path.

### Storage of the Data Component Files

The data component files are the only files for which you can specify the partition size. Partitioned data can be processed in threads easily, taking full advantage of multiple CPUs on your computer. The partition size for the data component file is fixed. It is set when the data set is created. The default is 128 megabytes, but you can specify a different partition size using the PARTSIZE= option. Performance depends on appropriate partition sizes. You are responsible for knowing the sizes and uses of the data. SPD Engine data sets can be created with a partition size that results in a balanced number of observations. (For more information, see "PARTSIZE= Data Set Option" on page 78 and "PARTSIZE= LIBNAME Statement Option" on page 41. Many data partitions can be created in each data path for a given data set. The SPD Engine uses the file paths that you specify with the DATAPATH= option to distribute partitions in a cyclic fashion. The SPD Engine creates the first data partition in one of the specified paths, the second partition in the next path, and so on. The SPD Engine continues to cycle through the file paths as many times as needed until all data partitions for the data set are stored. The file path for the first partition is selected at random. Assume that you specify the following in your LIBNAME statement:

```
datapath=('/data1' '/data2')
```

The SPD Engine stores the first partition in /DATA1, the second partition in /DATA2, the third partition in /DATA1, and so on. Cyclical distribution of the data partitions creates disk striping, which can be highly efficient. Disk striping is discussed in detail in "SPD Engine Disk I/O Setup" in Scalability and Performance at http://support.sas.com/rnd/scalability/spde/setup.html.

### Initial Set of Paths

In the following example, the LIBNAME statement specifies the MyLib directory for the primary path. This path is used to store the metadata partitions. Other devices and directories are specified to store the data and index partitions.

```
libname myref spde 'Mylib'
   datapath=('/mydisk30' '/mydisk31')
   indexpath=('/mydisk36');
```

Assuming that all of the data sets created in the MyLib library were large enough to have several data partitions, they will all have their metadata in MyLib, their data in **/mydisk30** and **/mydisk31**, and any indexes in **/mydisk36**. This specifically means that the metadata component files for those data sets include those pathnames.

### Adding Subsequent Paths

Later, if more space is needed (for example, for appending more data), additional devices can be added for the data and index partitions, as in the following example:

```
libname myref spde 'Mylib'
   datapath=('/mydisk30' '/mydisk31' '/mydisk32')
   indexpath=('/mydisk36' '/mydisk37');
```

All of the data sets created with the MyLib library will have their metadata in MyLib, their data in one or more of the three paths, and any indexes in **/mydisk36** or **/mydisk37**. If data was appended to an existing data set, the new data goes in one or more of the three paths and the metadata component file is updated accordingly.

If one or more of the data or index partitions do not have much free space, you can exclude them in the LIBNAME statement the next time you specify it:

```
libname myref spde 'Mylib'
   datapath=('/mydisk31' '/mydisk32' '/mydisk33')
   indexpath=('/mydisk37' '/mydisk38');
```

The SPD Engine is still able to access data sets that use the excluded paths because the data sets' metadata includes all of the used paths.

### Omitting Paths

If you need to read only the data sets in a library, then because all of the necessary path information is already in the metadata component files, you can specify the LIBNAME statement without the extra DATAPATH= and INDEXPATH= options:

```
libname myref spde 'Mylib';
```

### Renaming, Copying, or Moving Component Files

***CAUTION:***
**Do not rename, copy, or move an SPD Engine data set or its component files using operating system commands.**

You should always use the COPY procedure to copy SPD Engine data sets from one location to another or the DATASETS procedure to rename or delete SPD Engine data sets.

## Efficiency Using Disk Striping and Large Disk Arrays

Your system might have a file creation utility that enables you to override the file system limitations and create file systems (volumes) greater than the space on a single disk. You can use this utility to allocate SPD Engine libraries that span multiple disk devices, such as RAID. RAID configurations use a technique called disk striping that can significantly enhance I/O. For more information about disk striping and RAID, see “SPD Engine Disk I/O Setup” in Scalability and Performance at http://support.sas.com/rnd/scalability/spde/setup.html.

*Note:* If you are using Hadoop Distributed File System (HDFS) for storage, see *SAS SPD Engine: Storing Data in the Hadoop Distributed File System*.

## Converting Default Base SAS Engine Data Sets to SPD Engine Data Sets

### Using the COPY and APPEND Procedures

You can convert existing default Base SAS engine data sets to SPD Engine data sets using the following methods:

- PROC COPY
- PROC APPEND

Some limitations apply. If your default Base SAS engine data set has integrity constraints, then the integrity constraints are dropped when the data set is created in the SPD Engine format. The following chart of file characteristics indicates whether the characteristic can be retained or dropped or if conversion results in an error.

***Table 2.1*** *Conversion Results for Base SAS Engine Data Set Characteristics*

| Base SAS Engine Data Set Characteristic | Conversion Result |
|---|---|
| Indexes | Rebuilt in SPD Engine (in parallel if ASYNCINDEX=YES) |

| Base SAS Engine Data Set Characteristic | Conversion Result |
|---|---|
| Default Base SAS engine COMPRESS=YES \| CHAR \| BINARY * | Converts with compression if the data set is not encrypted |
| Default Base SAS engine ENCRYPT=YES * | Converts with encryption |
| Integrity constraints | Dropped without Error |
| Audit file | Dropped with Warning |
| Generations file | Dropped with Warning |
| Extended attributes | Dropped with Warning |

* If the default Base SAS engine data set has both compression and encryption, the compression is dropped but the encryption is retained. SAS retains the security of the data set instead of the compression.

### Converting Default Base SAS Engine Data Sets Using PROC COPY

To create an SPD Engine data set from an existing default Base SAS engine data set, you can simply use the COPY procedure. The PROC COPY statement copies the default Base SAS engine-formatted data set Local.Racquets to a new SPD Engine-formatted data set Sport.Racquets:

```
libname sport spde 'conversion_area';

proc copy in=local out=sport;
   select racquets;
run;
```

Even though the indexes on the default Base SAS engine data set are automatically regenerated as SPD Engine indexes (both .hdx and .idx files), they are not created in parallel because the data set option ASYNCINDEX=NO is the default.

If an SPD Engine data set is encrypted, only the data component files are encrypted. The metadata component files and both index component files are not encrypted.

### Converting Default Base SAS Engine Data Sets Using PROC APPEND

Use the APPEND procedure when you need to specify data set options for a new SPD Engine data set.

The following example creates an SPD Engine data set from a default Base SAS engine data set using PROC APPEND. The ASYNCINDEX=YES data set option specifies to build the indexes in parallel. The PARTSIZE= option specifies to create partitions of 100 megabytes.

```
libname spdelib spde 'new_data';
libname somelib 'old_data';
proc append base=spdelib.cars (asyncindex=yes partsize=100)
  data=somelib.cars;
run;
```

## Creating and Loading New SPD Engine Data Sets

To create a new SPD Engine data set, you can use the DATA step, any PROC statement[1] with the OUT= option, or PROC SQL with the CREATE TABLE= option.

The following example uses the DATA step to create a new SPD Engine data set, CARDATA.OLD_AUTOS in the report_area directory.

```
libname cardata spde '/report_area';

data cardata.old_autos(compress=no encrypt=yes pw=secret);
   input year $4. @6 manufacturer $12. @18 model $12. @31 body_style $5. @37
   engine_liters @42 transmission_type $1. @45 exterior_color
   $10. @55 mileage @62 condition;

datalines;

1966 Ford        Mustang      conv  3.5  M  white     143000 2
1967 Chevrolet   Corvair      sedan 2.2  M  burgundy   70000 3
1975 Volkswagen  Beetle       2door 1.8  M  yellow     80000 4
1987 BMW         325is        2door 2.5  A  black     110000 3
1962 Nash        Metropolitan conv  1.3  M  red       125000 3
;
run;
```

*Note:* Encryption and compression are mutually exclusive in SPD Engine. You can use the ENCRYPT= option only when you are creating an SPD Engine data file that is not compressed. You cannot create an SPD Engine data set with both encryption and compression.

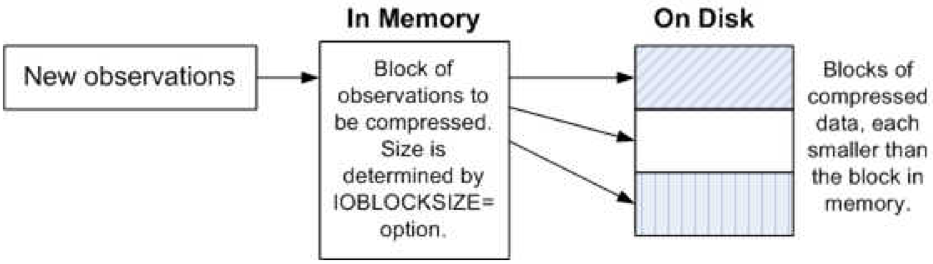
## Compressing SPD Engine Data Sets

When COMPRESS=YES | BINARY | CHAR, the SPD Engine compresses, by blocks, the data component file as it is created. The SPD Engine does not support user-specified compression. In addition, if you are copying a default Base SAS engine data set that is both compressed and encrypted, the encryption is retained, but the compression is dropped.

Once a compressed data set is created, you cannot change its block size. The compressed blocks are stored linearly, with no spaces between the blocks. The following figure illustrates how the blocks are stored on the disk:

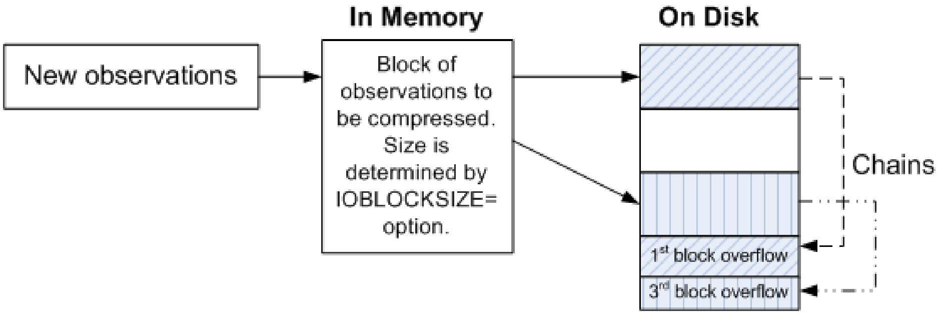
[1] except PROC MIGRATE

***Figure 2.1*** *Compressed Blocks on the Disk*


In Memory
On Disk
New observations
Block of observations to be compressed. Size is determined by IOBLOCKSIZE= option.
Blocks of compressed data, each smaller than the block in memory.

If updates to the data set after compression require more space than what is available in a block, SPD Engine creates a new block fragment to hold the overflow. If further updates again cause overflows, new block fragments are created, forming a chain. The following figure illustrates how the updates create a chain of blocks on the disk:

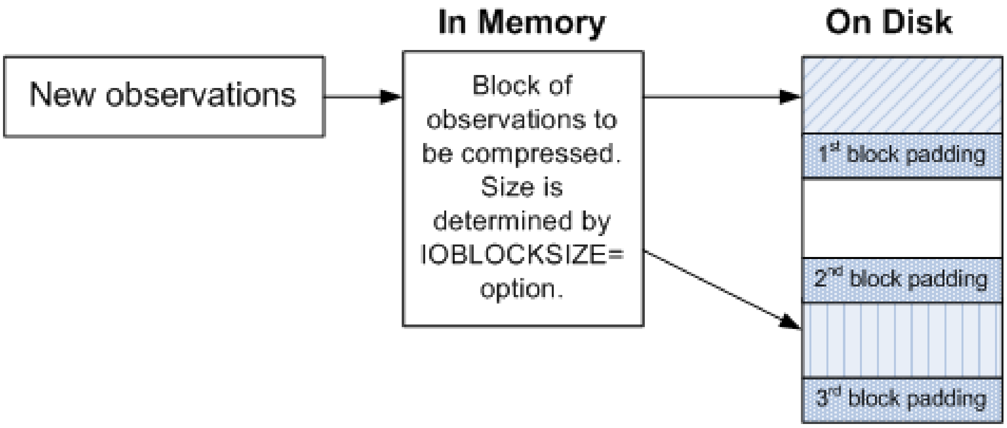
***Figure 2.2*** *Compressed Blocks with Overflow*


In Memory
On Disk
New observations
Block of observations to be compressed. Size is determined by IOBLOCKSIZE= option.
Chains
1st block overflow
3rd block overflow

Performance is affected if the chains get too long. To remove the chains and resize the block, you must copy the data set to a new data set. Specify IOBLOCKSIZE= on page 73 to the block size appropriate for the output data set.

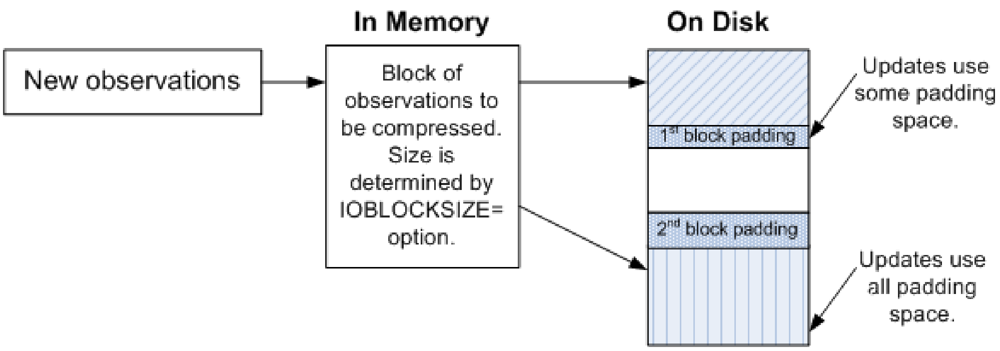
When the data set is expected to be updated frequently, it is recommended that you use PADCOMPRESS= on page 77. SPD Engine creates a padded space for each block, instead of creating new block fragments. The following figure illustrates how each block has padded space for updates:

***Figure 2.3*** *Compressed Padded Blocks*



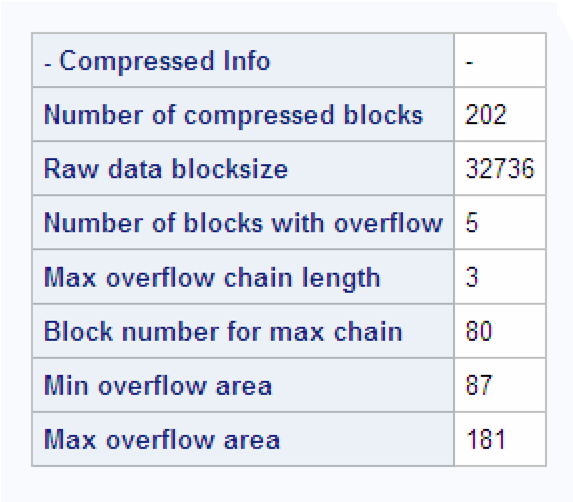
If updates to the data set after compression require more space than what is available in a block, SPD Engine uses the padded space for each block. New block fragments are not created. The following figure illustrates how the updates decrease the padded space:

***Figure 2.4*** *Compressed Padded Blocks with Updates*


In Memory
On Disk
New observations
Block of observations to be compressed. Size is determined by IOBLOCKSIZE= option.
1st block padding
2nd block padding
Updates use some padding space.
Updates use all padding space.

The CONTENTS procedure prints information about the compression. The following example explains the compressed info fields in the CONTENTS procedure output:

***Output 2.1*** *CONTENTS Procedure Compressed Info Output*



| - Compressed Info | - |
|---|---|
| Number of compressed blocks | 202 |
| Raw data blocksize | 32736 |
| Number of blocks with overflow | 5 |
| Max overflow chain length | 3 |
| Block number for max chain | 80 |
| Min overflow area | 87 |
| Max overflow area | 181 |

Number of compressed blocks
: number of compressed blocks that are required to store data.

Raw data blocksize
: compressed block size in bytes calculated from the size specified in the IOBLOCKSIZE= data set option.

Number of blocks with overflow
: number of compressed blocks that needed more space. When data is updated and the compressed new block is larger than the compressed old block, an overflow block fragment is created.

Max overflow chain length
: largest number of overflows for a single block. For example, the maximum overflow chain length would be 2 if a compressed block was updated and became larger, and then updated again to a larger size.

Block number for max chain
: number of the block containing the largest number of overflow blocks.

Min overflow area
: minimum amount of disk space that an overflow requires.

Max overflow area
: maximum amount of disk space that an overflow requires.

# Encrypting SPD Engine Data Sets

## *SPD Engine Encryption Overview*

*Encryption* is the transformation of intelligible data (plain text) into an unintelligible form (cipher text) by a mathematical process. The cipher text is translated back into plain text when you apply the appropriate password or ENCRYPTKEY that is necessary for decrypting (unlocking) the cipher text.

Encryption helps protect information on-disk and in-transit:

- *Over-the-wire* encryption protects SAS data while in transit.
- *On-disk* encryption protects data at rest.

There are two types of algorithms that SAS uses to encrypt SPD Engine data sets at rest:

SAS Proprietary
: provided within Base SAS software. This algorithm provides a medium level of security. You use the ENCRYPT=YES data set option to invoke this encryption.

AES (Advanced Encryption Standard)
: is a *block cipher* that encrypts data in blocks of 128 bits by using a 256-bit key. You use SAS/SECURE software, which is included with default Base SAS software. You use the ENCRYPT=AES data set option to invoke this encryption.

***Table 2.2*** *SPD Engine Encryption Features*

| Features | ENCRYPT=YES | ENCRYPT=AES |
|---|---|---|
| License required | No | No |
| Encryption level | Medium | High |
| Algorithm supported | within Base SAS software | AES |
| Installation required | No (part of Base SAS software) | No (in SAS/SECURE, which is included with Base SAS software) |
| Operating environments supported | UNIX<br>Windows<br>z/OS | UNIX<br>Windows<br>z/OS |
| SAS version support | 8 and later | 9.4 and later |

### SAS Proprietary Algorithm

SAS Proprietary uses a 32-bit fixed encoding and is appropriate only for preventing accidental exposure of information. SAS Proprietary is licensed with Base SAS software and is available in all deployments.

### AES Algorithm

The AES algorithm is a block cipher that encrypts data in blocks of 128 bits by using a 256-bit key. It uses SAS/SECURE software, which is included with Base SAS software. For more information about SAS/SECURE, see *Encryption in SAS*.

*Note:* AES encryption is not supported for OpenVMS on 64-bit Itanium.

AES encryption, which provides enhanced encryption for SPD Engine data sets, is available in SAS 9.4 and later. The AES algorithm produces a stronger encryption using SAS/SECURE. If you want an encrypted SPD Engine data set, you must use the ENCRYPTKEY= data set option with ENCRYPT=AES when you create the SPD Engine data set.

*Note:* You cannot change the ENCRYPTKEY= value on an AES-encrypted SPD Engine data set without re-creating the data set.

The following rules apply to AES encryption on SPD Engine data sets:

- You must use the ENCRYPTKEY= data set option when creating a data set with AES encryption.
- To copy an AES-encrypted SPD Engine data set, the output engine must support AES encryption. Otherwise, the data set is not copied.
- Releases before SAS 9.4 cannot use an AES-encrypted SPD Engine data set.
- If the SPD Engine data sets are AES-encrypted, all associated index files are also AES-encrypted. Metadata files are not AES-encrypted.

For more information, see “ENCRYPT= Data Set Option” on page 62 and “ENCRYPTKEY= Data Set Option” on page 65 .

## SPD Engine Component File Naming Conventions

When you create an SPD Engine data set, many component files can also be created. SPD Engine component files are stored with the following naming conventions:

```
filename.mdf.0.p#.v#.spds9
filename.dpf.fuid.p#.v#.spds9
filename.idxsuffix.fuid.p#.v#.spds9
filename.hbxsuffix.fuid.p#.v#.spds9
```

*filename*
valid SAS filename.

mdf
identifies the metadata component file.

dpf
identifies the partitioned data component files.

*p#*
is the partition number.

*v#*
is the version number. [1]

*fuid*
is the unique file ID, which is a hexadecimal equivalent of the primary (metadata) path.

idx*suffix*
identifies the segmented view of an index, where *suffix* is the name of the index.

hbx*suffix*
identifies the global view of an index, where *suffix* is the name of the index.

spds9
denotes a SAS 9 SPD Engine component file.

Table 2.2 shows the data set component files that are created when you use this LIBNAME statement and DATA step:

```
libname sample spde '/DATA01/SAS-library'
   datapath=('/DATA01/mydir' '/DATA02/mydir')
   indexpath=('/IDX1/mydir');
data sample.mine(index=(ssn));
   do i=1 to 100000;
   ssn=ranuni(0);
   end;
run;
```

***Table 2.3*** *Data Set Component Files*

| | |
|---|---|
| mine.mdf.0.0.0.spds9 | metadata component file |
| mine.dpf.000032a6.0.1.spds9 | data file partition #1 |
| mine.dpf.000032a6.1.1.spds9 | data file partition #2 |
| mine.dpf.000032a6.*n*-1.1.spds9 | data file partition #*n* |
| mine.dpf.000032a6.*n*.1.spds9 | data file partition #*n*+1 |
| mine.hbxssn.000032a6.0.1.spds9 | global index data set for variable SSN |
| mine.idxssn.000032a6.0.1.spds9 | segmented index data set for variable SSN |

[1] The version number increases only when the data set is updated, that is, when the data set is opened in UPDATE mode. Operations such as PROC SORT that replace the data set reset the version number to one, instead of incrementing it.

# Efficient Indexing in the SPD Engine

## *Parallel Indexing*

Indexes can improve the performance of WHERE expression processing and BY expression processing. The SPD Engine enables the rapid creation and update of indexes because it can process them in parallel.

The SPD Engine's indexes are especially appropriate for data sets of varying sizes and data distributions. These indexes contain both a segmented view and a global view of indexed variables' values. This feature enables the SPD Engine to optimally support both of the following queries:

- queries that require global data views, such as BY expression processing
- queries that require segmented views, such as parallel processing of WHERE expressions

## *Parallel Index Creation*

You can create indexes on your SPD Engine data in parallel, asynchronously. To enable asynchronous parallel index creation, use the "ASYNCINDEX= Data Set Option" on page 53.

Use this option with the DATA step INDEX= option and with the PROC DATASETS MODIFY statement when creating a data set that has several indexes. Either method enables all of the declared indexes to be populated from a single scan of the data set.

The following example shows indexes created in parallel using the DATA step. A simple index is created on variable X and a composite index is created on variables A and B.

```
data foo.mine(index=(x y=(a b)) asyncindex=yes);
   x=1;
   a="Doe";
   b=20;
run;
```

To create multiple indexes in parallel, you must allocate enough utility disk space to create all of the key sorts at the same time. You must also allocate enough memory space. Use the SPDEUTILLOC= system option on page 102 to allocate disk space and SPDEINDEXSORTSIZE system option on page 100 in the configuration file or at invocation to allocate additional memory.

The DATASETS procedure has the flexibility to enable batched parallel index creation by using multiple MODIFY groups. Instead of creating all of the indexes at once, which would require a significant amount of space, you can create the indexes in groups as shown in the following example:

```
proc datasets lib=main;
   modify patients(asyncindex=yes);
      index create number;
      index create class;
   run;
   modify patients(asyncindex=yes)'
      index create lastname firstname;
   run;
   modify patients(asyncindex=yes);
      index create fullname=(lastname firstname);
      index create class_sex=(class sex);
   run;
quit;
```

Indexes Number and Class are created in parallel, indexes LastName and FirstName are created in parallel, and indexes FullName and Class_Sex are created in parallel.

### *Parallel Index Updates*

The SPD Engine also supports parallel index updating during data set Append operations. Multiple threads enable updates of the data store and index files. The SPD Engine decomposes a data set Append or Insert operation into a set of steps that can be performed in parallel. The level of parallelism attained depends on the number of indexes in the data set. As with parallel index creation, this operation uses memory and disk space for the key sorts that are part of the index append processing. Use system options SPDEINDEXSORTSIZE= to allocate memory and SPDEUTILLOC= to allocate disk space.

*Note:* The ASYNCINDEX option is not valid for parallel index updates.

## Backing Up SPD Engine Files

When you back up an SPD Engine data set, remember the following requirements:

- Ensure that all of the files that make up the data set are backed up together, at the same time, even if they reside on different disks or file systems.
- Do not back up the data set while any files are being updated.
- After each backup, run a test to verify that the backup was a success.

## Storing SPD Engine Data in HDFS

The SPD Engine can read, write, and update data in HDFS. Storing SPD Engine data in HDFS provides a low-cost alternative to storing big data. You can use the SPD Engine with standard SAS applications to retrieve data for analysis, perform administrative functions, and update data.

*Chapter 3*

# SPD Engine LIBNAME Statement Options

## Introduction to the SPD Engine LIBNAME Statement

This section contains reference information for all LIBNAME options that are valid for the SPD Engine LIBNAME statement.

Some of these LIBNAME options are also data set options. As in the default Base SAS engine, data set options take precedence over corresponding LIBNAME options if both options are set.

## Syntax

LIBNAME *libref* SPDE '*full-primary-path*' <*option(s)*> ;

*libref*
a name that is up to eight characters long and that conforms to the rules for SAS names.

*'full-primary-path'*
the complete pathname of the primary path for the SPD Engine library. The name must be recognized by the operating environment. Enclose the name in single or double quotation marks. Unless the DATAPATH= and INDEXPATH= options are specified, the index and data components are stored in the same location. The primary path must be unique for each library. Librefs that are different but reference the same primary path are interpreted to be the same library and can result in lost data.

*option(s)*
one or more SPD Engine LIBNAME statement options.

*Operating Environment Information*: A valid library specification and its syntax are specific to your operating environment. For details, see the SAS documentation for your operating environment.

## SPD Engine LIBNAME Statement Options List

ACCESS=READONLY
specifies that data sets can be read, but not updated or created.

BYSORT=
specifies the SPD Engine to perform an automatic sort when it encounters a BY statement. BYSORT= is also a data set option.

COMPRESS=
specifies to compress SPD Engine data sets on disk as they are being created. COMPRESS= is also a data set option.

DATAPATH=
specifies a list of paths in which to store data partitions (.dpf) for an SPD Engine data set.

ENDOBS=
specifies the end observation number in a user-defined range of observations to be processed. ENDOBS= is also a data set option.

IDXBY=
specifies whether to use an index when processing a BY statement in the SPD Engine. IDXBY= is also a data set option.

INDEXPATH=
specifies a path or list of paths in which to store the two types of index component files (.hbx and .idx) associated with an SPD Engine data set.

IOBLOCKSIZE=
specifies the size in bytes of a block of observations to be used in an I/O operation. IOBLOCKSIZE= is also a data set option.

METAPATH=
specifies a list of paths in which to store overflow metadata (.mdf) component files for an SPD Engine data set.

PARTSIZE=
specifies the maximum size that the data component partitions can be. The value is specified when the SPD Engine data set is created. This size is a fixed size. This specification applies only to the data component files. PARTSIZE= is also a data set option.

STARTOBS=
: specifies the starting observation number in a user-defined range of observations to be processed. STARTOBS= is also a data set option.

TEMP=
: specifies to store the library in a temporary subdirectory of the primary path.

# Dictionary

## ACCESS= LIBNAME Statement Option

Determines the access level of the data source.

**Default:** none

**Engine:** SPD Engine only

### Syntax

**ACCESS=**READONLY

#### *Required Argument*

**READONLY**
: specifies that data sets can be read, but not updated or created.

### Details

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

## BYSORT= LIBNAME Statement Option

Specifies the SPD Engine to perform an automatic sort when it encounters a BY statement.

**Default:** YES

**Interaction:** "BYNOEQUALS= Data Set Option" on page 54

**Engine:** SPD Engine only

### Syntax

**BYSORT=**YES | NO

#### *Required Arguments*

**YES**
: specifies to automatically sort the data based on the BY variables when a BY statement is encountered instead of sorting the data ahead of time.

**NO**
specifies not to sort the data based on the BY variables. Specifying NO means that the data must already be sorted before the BY statement. Indexes are not used.

## Details

DATA or PROC step processing using the default Base SAS engine requires that if there is no index or if the observations are not in order, the data set must be sorted before a BY statement is issued. In contrast, by default, the SPD Engine sorts the data returned to the application if the observations are not in order. Unlike PROC SORT, which creates a new sorted data set, the SPD Engine's automatic sort does not change the permanent data set and does not create a new data set. However, utility file space is used. For more information, see “SPDEUTILLOC= System Option” on page 102.

The default is BYSORT=YES. A BYSORT=YES argument enables the automatic sort, which outputs the observations in BY group order. If the data set option BYNOEQUALS=YES, then the observations within a group might be output in a different order from the order in the data set. Set BYNOEQUALS=NO to retain data set order.

The BYSORT=NO argument means that the data must already be sorted before the BY statement. Sorting can be from a previous PROC SORT or from the data set having been created in BY variable order. An error occurs if the data set is not sorted. When BYSORT=NO, grouped data is delivered to the application in data set order. Indexes are not used to retrieve the observations in BY variable order. The data set option BYNOEQUALS= has no effect when BYSORT=NO.

If you specify the BYSORT= option in the LIBNAME statement, it can be overridden by specifying BYSORT= in the PROC or DATA steps. Set BYSORT=YES in the DATA or PROC step, for input opens, to override BYSORT=NO in the LIBNAME statement. The point is that BYSORT=NO instructs the engine to do nothing to sort the data.

When you use the BYSORT=YES and the IDXWHERE= data set options, the following messages are written to the SAS log if you set the MSGLEVEL=I SAS system option:

- If IDXWHERE=YES and there is an index on the BY variable, the index is used to order the rows of the table. The following message is written to the SAS log:

  ```
  Note: BY ordering was produced by using an index for table tablename.
  ```

- If IDXWHERE=NO or IDXWHERE=YES and there is no index on the BY variable, SPD Engine performs an automatic sort to order the rows of the table. The following message is written to the SAS log:

  ```
  Note: BY ordering was produced by performing an automatic
  sort on table tablename.
  ```

## Comparisons

The BYSORT= data set option overrides the BYSORT= LIBNAME statement option.

## Examples

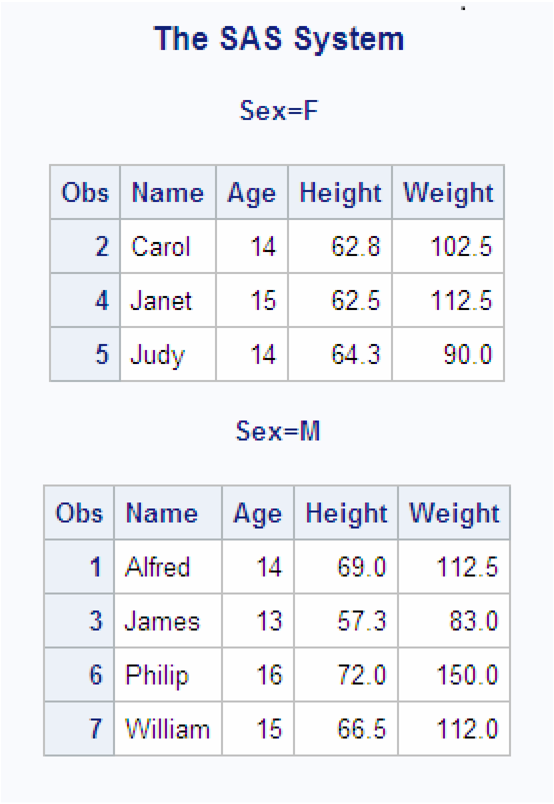
### Example 1: Group Formatting with BYSORT=YES by Default

```
libname growth spde 'SAS-library';
data growth.teens;
   input Name $ Sex $ Age Height Weight;
datalines;
Alfred M 14 69.0 112.5
```

```
Carol F 14 62.8 102.5
James M 13 57.3 83.0
Janet F 15 62.5 112.5
Judy F 14 64.3 90.0
Philip M 16 72.0 150.0
William M 15 66.5 112.0
;
proc print data=growth.teens; by sex;
run;
```

Even though the data was not sorted using PROC SORT, no error occurred because BYSORT=YES is the default. The output is shown:

***Output 3.1*** *Group Formatting with BYSORT=YES by Default*



The SAS System

Sex=F

| Obs | Name | Age | Height | Weight |
|---|---|---|---|---|
| 2 | Carol | 14 | 62.8 | 102.5 |
| 4 | Janet | 15 | 62.5 | 112.5 |
| 5 | Judy | 14 | 64.3 | 90.0 |

Sex=M

| Obs | Name | Age | Height | Weight |
|---|---|---|---|---|
| 1 | Alfred | 14 | 69.0 | 112.5 |
| 3 | James | 13 | 57.3 | 83.0 |
| 6 | Philip | 16 | 72.0 | 150.0 |
| 7 | William | 15 | 66.5 | 112.0 |

## *Example 2: Using BYSORT=NO in the LIBNAME Statement*

In the following example, SAS returns an error because BYSORT=YES was not specified on the DATA or PROC steps to override the BYSORT=NO specification in the LIBNAME statement. Whenever automatic sorting is suppressed (BYSORT=NO), the data must be sorted on the BY variable before the BY statement (for example, by using PROC SORT).

```
libname growth spde 'SAS-library' bysort=no;
proc print data=growth.teens;
by sex;
run;
```

```
ERROR: Data set GROWTH.TEENS is not sorted in ascending sequence.
       The current by-group has Sex = M and the next by-group has Sex = F.
NOTE: The SAS System stopped processing this step because of errors.
```

## COMPRESS= LIBNAME Statement Option

Specifies to compress an SPD Engine data set on disk as it is being created.

**Default:** NO

**Restriction:** Cannot be used with ENCRYPT=YES or ENCRYPT=AES

**Interactions:** "IOBLOCKSIZE= LIBNAME Statement Option" on page 39
"PADCOMPRESS= Data Set Option" on page 77

**Engine:** SPD Engine only

### Syntax

**COMPRESS=**NO | YES | CHAR | BINARY

#### Required Arguments

**NO**
performs no data set compression.

**YES | CHAR**
specifies that data in an SPD Engine data set be compressed in blocks by using RLE (run-length encoding). RLE compresses data by reducing repeated runs of the same character (including a blank space) to two-byte or three-byte representations.

**BINARY**
specifies that data in an SPD Engine data set be compressed in blocks by using RDC (Ross Data Compression). RDC combines RLE and sliding window compression to compress the file by representing repeated byte patterns more efficiently.

*Note:* This method is highly effective for compressing medium to large (several hundred bytes or larger) blocks of binary data (character and numeric variables).

### Details

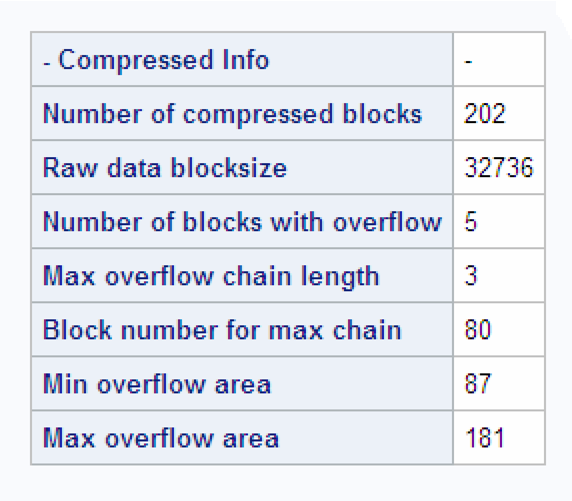
When you specify COMPRESS=YES | BINARY | CHAR, the SPD Engine compresses, by blocks, the data component file as it is created. To specify the size of the compressed blocks, use the "IOBLOCKSIZE= Data Set Option" on page 73 when you create the data set. To add padding to the newly compressed blocks, specify "PADCOMPRESS= Data Set Option" on page 77 when creating or updating the data set. For more information, see "Compressing SPD Engine Data Sets" on page 18.

The SPD Engine does not support user-specified compression. If you are migrating a default Base SAS engine data set that is both compressed and encrypted, the encryption is retained, but the compression is dropped.

The CONTENTS procedure identifies the compress setting. If the data set is compressed, PROC CONTENTS prints information about the compression. The following example explains the Compressed Info fields in the CONTENTS procedure output:

In general, COMPRESS=CHAR provides good compression when single bytes repeat; COMPRESS=BINARY provides good compression when strings of bytes repeat. At the same time, it is more costly to look for strings of bytes that repeat, than to look for single bytes that repeat. For examples, see "Example 1: COMPRESS=CHAR" on page 62 and "Example 2: COMPRESS=BINARY" on page 62.

***Output 3.2*** *PROC CONTENTS Compressed Section*



| - Compressed Info | - |
|---|---|
| Number of compressed blocks | 202 |
| Raw data blocksize | 32736 |
| Number of blocks with overflow | 5 |
| Max overflow chain length | 3 |
| Block number for max chain | 80 |
| Min overflow area | 87 |
| Max overflow area | 181 |

Number of compressed blocks
number of compressed blocks that are required to store data.

Raw data blocksize
compressed block size in bytes calculated from the size specified in the IOBLOCKSIZE= data set option. It is the largest multiple of the observation length that gets in the block size.

Number of blocks with overflow
number of compressed blocks that needed more space. When data is updated and the compressed new block is larger than the compressed old block, an overflow block fragment is created.

Max overflow chain length
largest number of overflows for a single block. For example, the maximum overflow chain length would be 2 if a compressed block was updated and became larger, and then updated again to a larger size.

Block number for max chain
number of the block containing the largest number of overflow blocks.

Min overflow area
minimum amount of disk space that an overflow requires.

Max overflow area
maximum amount of disk space that an overflow requires.

Accessing compressed files usually requires more processing time. The files have to be decompressed before reading them and, if updating, they have to be compressed again when written to disk.

## Comparisons

The COMPRESS= LIBNAME statement option overrides the COMPRESS= system option.

The COMPRESS= data set option overrides the COMPRESS= LIBNAME statement option.

## DATAPATH= LIBNAME Statement Option

Specifies a list of paths in which to store data partitions (.dpf) for an SPD Engine data set.

**Default:** The primary path specified in the LIBNAME statement

**Interactions:** "PARTSIZE= LIBNAME Statement Option" on page 41
"PARTSIZE= Data Set Option" on page 78

**Engine:** SPD Engine Only

### Syntax

**DATAPATH=**(*'path1'* <*'path2'*>…)

#### *Required Argument*

***'path'***
is a complete pathname in single or double quotation marks within parentheses. Separate multiple arguments with spaces.

*Note:* The pathnames specified in the DATAPATH= option must be unique for each library. Librefs that are different but reference the same pathnames can result in lost data.

*Note:* If your data is in the zFS file system, only one path specification is required. The zFS system automatically spreads the partitions across multiple logical volumes.

### Details

The SPD Engine creates as many partitions as needed to store all the data. The size of the partitions is set using the PARTSIZE= option, and partitions are created in the paths specified using the DATAPATH= option in a cyclic fashion.

*Note:* If you are planning to store data in locally mounted drives and access the data from a remote computer, use the remote pathname when you specify the LIBNAME. For example, if **`/data01`** and **`/data02`** are locally mounted drives on the localA computer, use the pathnames **`/nfs/localA/data01`** and **`/nfs/localA/data02`** in the LIBNAME statement.

### Example: DATAPATH= for First Partition

The path for the first partition is randomly selected and then continues in a cyclical fashion:

```
libname mylib spde '/metadisk/metadata'
   datapath=('/disk1/dataflow1' '/disk2/dataflow2' '/disk3/dataflow3');
```

For example, if /disk2/dataflow2 is randomly selected as the first path, the first partition is located there. The second partition is located in /disk3/dataflow3, the third partition is located in /disk1/dataflow1, and so on.

## ENDOBS= LIBNAME Statement Option

Specifies the end observation number in a user-defined range of observations to be processed.

**Default:** The last observation in the data set

**Restrictions:** Use ENDOBS= with input data sets only

Cannot be used with the OBS= system or data set option or the FIRSTOBS= system or data set option

**Interactions:** "ENDOBS= Data Set Option" on page 67

"STARTOBS= LIBNAME Statement Option" on page 43

"STARTOBS= Data Set Option" on page 80

**Engine:** SPD Engine only

### Syntax

**ENDOBS=***n*

#### *Required Argument*

***n***

is the number of the end observation.

### Details

By default, the SPD Engine processes all of the observations in the entire data set unless you specify a range of observations with the STARTOBS= and ENDOBS= options. If the STARTOBS= option is used without the ENDOBS= option, the implied value of ENDOBS= is the end of the data set. When both options are used together, the value of ENDOBS= must be greater than the value of STARTOBS=.

In contrast to the default Base SAS engine option FIRSTOBS=, the STARTOBS= and ENDOBS= SPD Engine system options can be used in the LIBNAME statement.

*Note:* The OBS= system option and the OBS= data set option cannot be used with STARTOBS= or ENDOBS= data set or LIBNAME options.

(See SPD Engine Data Set Options on page 47 for information about using the ENDOBS= data set option in WHERE processing.)

### Comparisons

The ENDOBS= data set option overrides the ENDOBS= LIBNAME statement option.

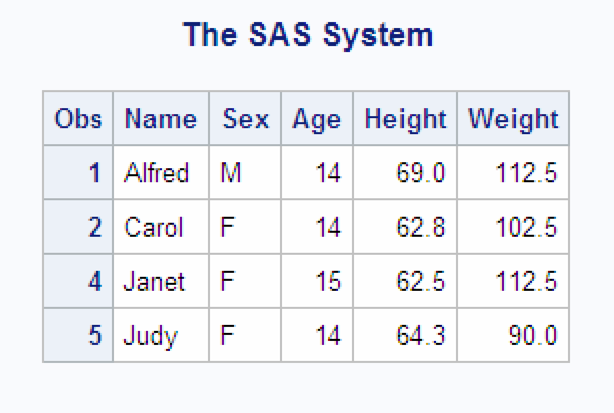
### Example: Using the ENDOBS= LIBNAME Statement Option

The following example shows that the STARTOBS= and ENDOBS= options subset the data before the WHERE clause executes. The example prints the four observations that were qualified by the WHERE expression (age >13 in PROC PRINT). The four observations are out of the five observations that were processed from the input data set:

```
libname growth spde 'SAS-library' endobs=5;
```

```
data growth.teens;
   input Name $ Sex $ Age Height Weight;
datalines;
Alfred M 14 69.0 112.5
Carol F 14 62.8 102.5
James M 13 57.3 83.0
Janet F 15 62.5 112.5
Judy F 14 64.3 90.0
Philip M 16 72.0 150.0
William M 15 66.5 112.0
;
proc print data=growth.teens;
   where age >13;
run;
```

***Output 3.3*** *ENDSOBS=*



The SAS System

| Obs | Name | Sex | Age | Height | Weight |
|---|---|---|---|---|---|
| 1 | Alfred | M | 14 | 69.0 | 112.5 |
| 2 | Carol | F | 14 | 62.8 | 102.5 |
| 4 | Janet | F | 15 | 62.5 | 112.5 |
| 5 | Judy | F | 14 | 64.3 | 90.0 |

## IDXBY= LIBNAME Statement Option

Specifies whether to use an index when processing a BY statement in the SPD Engine.

**Default:** YES

**Interactions:** “BYSORT= LIBNAME Statement Option” on page 29
“BYSORT= Data Set Option” on page 57

**Engine:** SPD Engine only

### Syntax

**IDXBY=**YES | NO

### *Required Arguments*

**YES**
uses an index when processing indexed variables in a BY statement.

*Note:* If the BY statement specifies more than one variable or the DESCENDING option, then the index is not used, even if IDXBY=YES.

**NO**

does not use an index when processing indexed variables in a BY statement.

**Note** IDXBY=NO performs an automatic sort when processing a BY statement.

## Details

When you use the IDXBY= LIBNAME option, make sure that you use BYSORT=YES option and that the BY variable is indexed.

In some cases, you might get better performance from the SPD Engine if you automatically sort the data. To use the automatic sort, BYSORT=YES must be set and you should specify IDXBY=NO.

Set the SAS system option MSGLEVEL=I so that the BY processing information is written to the SAS log. When you use the IDXBY= LIBNAME option and the BYSORT=YES option, the following messages are written to the SAS log:

- If IDXBY=YES and there is an index on the BY variable, the index is used to order the rows of the table. The following message is written to the SAS log:

  ```
  NOTE: BY ordering was produced by using an index for
        table tablename.
  ```

- If you use IDXBY=NO, the following message is written to the SAS log:

  ```
  NOTE: BY ordering was produced by performing an automatic sort
        on table tablename.
  ```

## Comparisons

The IDXBY= data set option overrides the IDXBY= LIBNAME statement option.

## Examples

### *Example 1: Using the IDXBY=NO LIBNAME Option*

```
libname permdata spde 'SAS-library' idxby=no;
options msglevel=i;
    proc means data=permdata.customer;
       var sales;
   by state;
    run;
```

The following message is written to the SAS log:

```
NOTE: BY ordering was produced by performing an automatic sort
      on table PERMDATA.customer.
NOTE: There were 100 observations read from the data
      set PERMDATA.CUSTOMER.
```

### *Example 2: Using the IDXBY=YES LIBNAME Option*

The following example uses IDXBY=YES:

```
libname permdata spde 'SAS-library' idxby=yes;
options msglevel=i;
    proc means data=permdata.customer;
    var sales;
    by state;
```

```
run;
```

The following message is written to the SAS log:

```
NOTE: BY ordering was produced by using an index for table
      PERMDATA.customer.
NOTE: There were 2981 observations read from the data set
      PERMDATA.CUSTOMER.
```

## INDEXPATH= LIBNAME Statement Option

Specifies a path or list of paths in which to store the two types of index component files (.hbx and .idx) associated with an SPD Engine data set.

**Default:** The primary path specified in the LIBNAME statement

**Engine:** SPD Engine only

### Syntax

**INDEXPATH=**(*'path1'* <*'path2'*…>)

#### *Required Argument*

***'path'***

is a complete pathname, in single or double quotation marks within parentheses. Separate multiple arguments with spaces.

*Note:* The pathnames specified in the INDEXPATH= option must be unique for each library. Librefs that are different but reference the same pathnames can result in lost data.

### Details

Unlike metadata component files, index component files do not have to be in the primary path. For more information, see "Storage of the Index Component Files" on page 14.

The INDEXPATH= option enables index I/O to be moved to another physical path or device. This enhances performance. For more information, see "Features That Enhance I/O Performance" on page 8.

The SPD Engine creates two index component files in the locations specified. If there are multiple paths specified with the INDEXPATH= option, the first path is randomly selected. If multiple paths are specified, index component files are created in the first path, and then they overflow to the next path when the first path is filled.

*Note:* If you are planning to store data in locally mounted drives and access the data from a remote computer, use the remote pathname when you specify the LIBNAME. For example, if **/data01** and **/data02** are locally mounted drives on the localA computer, use the pathnames **/nfs/localA/data01** and **/nfs/localA/data02** in the LIBNAME statement.

### Example: Creating Index Component Files

The following example creates index component files that span the paths /disk1/idxflow1, /disk2/idxflow2, and /disk3/idxflow3.

```
libname mylib spde '/metadisk/metadata'
                   datapath= ('/disk1/dataflow1' '/disk2/dataflow2'
                                '/disk3/dataflow3')
                   indexpath=('/disk1/idxflow1' '/disk2/idxflow2'
                                '/disk3/idxflow3' );
```

The path for the first index component files is randomly selected. SAS puts the index component files in the first location until that location is full, and then continues in a cyclical fashion. For example, if /disk2/idxflow2 is randomly selected, the first index component files are located there. When that location is full, the index component files overflow to /disk3/idxflow3, and then to /disk1/idxflow1.

## IOBLOCKSIZE= LIBNAME Statement Option

Specifies the size in bytes of a block of observations to be used in an I/O operation.

**Default:** 1,048,576 bytes (1 megabyte)

**Range:** The minimum block size is 32,768 bytes. The maximum block size is half the size of the SPD Engine data partition file.

**Engine:** SPD Engine only

**Tip:** When reading a data set, the block size can significantly affect performance. When retrieving a large percentage of the data, a larger block size improves performance. However, when retrieving a subset of the data such as with WHERE processing, a smaller block size performs better.

### Syntax

**IOBLOCKSIZE=***n*

#### *Required Argument*

***n***

is the size in bytes of a block of observations.

### Details

The I/O block size determines the amount of data that is physically transferred together in an I/O operation. The larger the block size, the less I/O. The SPD Engine uses blocks in memory to collect the observations to be written to or read from a data component file. The IOBLOCKSIZE= option specifies the size of the block. (The actual size is computed to accommodate the largest number of observations that fit in the specified size of *n* bytes. Therefore, the actual size is a multiple of the observation length.)

The block size affects I/O operations for compressed, uncompressed, and encrypted data sets. However, the effects are different and depend on the I/O operation.

- For a compressed data set, the block size determines how many observations are compressed together, which determines the amount of data that is physically transferred for both Read and Write operations. The block size is a permanent attribute of the file. To specify a different block size, you must copy the data set to a new data set, and then specify a new block size for the output file. For a compressed data set, a larger block size can improve performance for both Read and Write operations.
- For an encrypted data set, the block size is a permanent attribute of the file.

- For an uncompressed data set, the block size determines the size of the blocks that are used to read the data from disk to memory. The block size has no affect when writing data to disk. For an uncompressed data set, the block size is not a permanent attribute of the file. That is, you can specify a different block size based on the Read operation that you are performing. For example, reading data that is randomly distributed or reading a subset of the data calls for a smaller block size because accessing smaller blocks is faster than accessing larger blocks. In contrast, reading data that is uniformly or sequentially distributed or that requires a full data set scan works better with a larger block size.

### Comparisons

The IOBLOCKSIZE= data set option overrides the IOBLOCKSIZE= LIBNAME statement option.

### Example: Using IOBLOCKSIZE=

```
/*IOBLOCKSIZE set to 64K */
libname employees spde 'SAS-library' ioblocksize=65536;

/*IOBLOCKSIZE set to 512M */
libname sales spde 'SAS-library' ioblocksize= 524288;
```

## METAPATH= LIBNAME Statement Option

Specifies a list of paths in which to store overflow metadata (.mdf) component files for an SPD Engine data set.

**Engine:** SPD Engine only

### Syntax

**METAPATH=**(*'path1'* <*'path2'*…>)

#### *Required Argument*

***'path'***
is a complete pathname in single or double quotation marks within parentheses. Separate multiple arguments with spaces.

### Details

The metadata component files for all of the data sets in a library must reside in the same location specified in the primary path. If a new data set for a library is created, and the space in the primary path is full, the SPD Engine cannot begin creating the metadata component file in the primary path. The Create operation fails with an appropriate error message. For more information, see "Storage of the Metadata Component Files" on page 14.

The METAPATH= option specifies space that is exclusively overflow space for metadata component files. The metadata component file for each data set must reside in the primary path. Overflow populates the METAPATH= location when the primary path is full.

*Note:* If you are planning to store data in locally mounted drives and access the data from a remote computer, use the remote pathname when you specify the LIBNAME. If `/data01` and `/data02` are locally mounted drives on the localA computer, use the pathnames `/nfs/localA/data01` and `/nfs/localA/data02` in the LIBNAME statement.

## PARTSIZE= LIBNAME Statement Option

Specifies the maximum size (in megabytes, gigabytes, or terabytes) that the data component partitions can be. The value is specified when an SPD Engine data set is created. This size is a fixed size. This specification applies only to the data component files.

**Default:** 128 MB

**Interactions:** "DATAPATH= LIBNAME Statement Option" on page 34
"MINPARTSIZE= System Option" on page 99

**Engine:** SPD Engine only

### Syntax

**PARTSIZE=***n* | *n*M | *n*G | *n*T

#### *Required Argument*

***n* | *n*M | *n*G | *n*T**
is the size of the partition in megabytes, gigabytes, or terabytes. If *n* is specified without M, G, or T, the default is megabytes. PARTSIZE=128 is the same as PARTSIZE=128M. The maximum value is 8,796,093,022,207 megabytes.

**Restriction** This restriction applies only to 32-bit hosts with the following operating systems: z/OS, Linux SLES 9 x86, and the Windows family. If you create a data set with a partition size greater than or equal to 2 gigabytes, you cannot open the data set with any version of SPD Engine prior to SAS 9.2. The following error message is written to the SAS log: **`ERROR: Unable to open data file because its data representation differs from the SAS session data representation.`**

### Details

SPD Engine data must be stored in multiple partitions for it to be subsequently processed in parallel. Specifying PARTSIZE= forces the software to partition SPD Engine data files at the specified size. The actual size of the partition is computed to accommodate the maximum number of observations that fit in the specified size of *n* megabytes, gigabytes, or terabytes.

By splitting (partitioning) the data portion of an SPD Engine data set into fixed-sized files, the software can introduce a high degree of scalability for some operations. The SPD Engine can spawn threads in parallel (for example, up to one thread per partition for WHERE evaluations). Separate data partitions also enable the SPD Engine to process the data without the overhead of file access contention between the threads. Because each partition is one file, the trade-off for a small partition size is that an increased number of files (for example, UNIX i-nodes) are required to store the observations.

Scalability limitations using PARTSIZE= depend on how you configure and spread the file systems specified in the DATAPATH= option across striped volumes. (You should spread each individual volume's striping configuration across multiple disk controllers or SCSI channels in the disk storage array.) The goal for the configuration is to maximize parallelism during data retrieval. For information about disk striping, see “I/O Setup and Validation” under “SPD Engine” in Scalability and Performance at http://support.sas.com/rnd/scalability.

The PARTSIZE= specification is limited by the SPD Engine system option MINPARTSIZE=, which is usually set and maintained by the system administrator. MINPARTSIZE= ensures that an inexperienced user does not arbitrarily create small partitions, thereby generating a large number of files.

The partition size determines a unit of work for many of the parallel operations that require full data set scans. But, more partitions does not always mean faster processing. The trade-offs involve balancing the increased number of physical files (partitions) required to store the data set against the amount of work that can be done in parallel by having more partitions. More partitions means more open files to process the data set, but a smaller number of observations in each partition. A general rule is to have 10 or fewer partitions per data path and 3 to 4 partitions per CPU.

To determine an adequate partition size for a new SPD Engine data set, you should be aware of the following:

- the types of applications that run against the data
- how much data you have
- how many CPUs are available to the applications
- which disks are available for storing the partitions
- the relationships of these disks to the CPUs

For example, if each CPU controls only one disk, then an appropriate partition size would be one in which each disk contains approximately the same amount of data. If each CPU controls two disks, then an appropriate partition size would be one in which the load is balanced. Each CPU does approximately the same amount of work.

*Note:* The PARTSIZE= value for a data set cannot be changed after a data set is created. To change PARTSIZE=, you must re-create the data set and specify a different PARTSIZE= value in the LIBNAME statement or on the new (output) data set.

## Comparisons

The PARTSIZE= data set option overrides the PARTSIZE= LIBNAME statement option.

## Example: Specifying the Partition Size

When you specify the partition size in the LIBNAME statement, you have to select a size that is appropriate for most of the data sets stored in that library. For example, suppose you have an 8-disk configuration. The smallest data set has 20 gigabytes of data, the largest has 50 gigabytes of data, and the remaining data sets have 36 gigabytes of data each. A partition size of 1250M is optimal for a 36-gigabyte data set (four partitions per disk). The 20-gigabyte data set uses two partitions per disk, and the 50-gigabyte data set uses five partitions per disk.

```
libname sales spde '/primdisk' partsize=1250M
datapath=('/disk01' '/disk02' '/disk03' '/disk04'
'/disk05' '/disk06' '/disk07' '/disk08');
```

## STARTOBS= LIBNAME Statement Option

Specifies the starting observation number in a user-defined range of observations to be processed.

**Default:** The first observation in the data set

**Restrictions:** Use STARTOBS= with input data sets only

Cannot be used with the OBS= system or data set option or with the FIRSTOBS= system or data set option

**Interactions:** “STARTOBS= Data Set Option” on page 80

“ENDOBS= LIBNAME Statement Option” on page 35

“ENDOBS= Data Set Option” on page 67

**Engine:** SPD Engine only

### Syntax

**STARTOBS=***n*

#### *Required Argument*

***n***

is the number of the starting observation.

### Details

By default, the SPD Engine processes all of the observations in the entire data set unless you specify a range of observations with the STARTOBS= and ENDOBS= options. If the ENDOBS= option is used without the STARTOBS= option, the implied value of STARTOBS= is 1. When both options are used together, the value of STARTOBS= must be less than the value of ENDOBS=.

In contrast to the default Base SAS engine option FIRSTOBS=, the STARTOBS= and ENDOBS= SPD Engine options can be used in the LIBNAME statement.

*Note:* FIRSTOBS= default Base SAS engine option is not supported by the SPD Engine. The OBS= system option and the OBS= data set option cannot be used with STARTOBS= or ENDOBS= data set or LIBNAME options.

(See SPD Engine Data Set Options on page 47 for information about using the STARTOBS= data set option in WHERE processing.)

### Comparisons

The STARTOBS= data set option overrides the STARTOBS= LIBNAME statement option.

### Example: Using the WHERE Expression

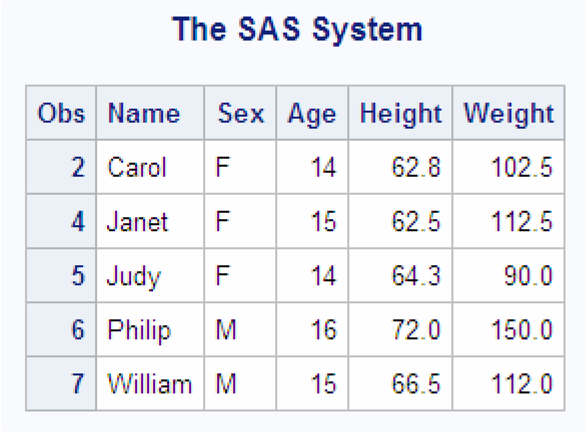
The following example prints the five observations that were qualified by the WHERE expression (age >13 in PROC PRINT). The five observations are out of the six observations that were processed, starting with the second observation in the data set:

```
libname growth spde 'SAS-library' startobs=2;
data growth.teens;
```

```
   input Name $ Sex $ Age Height Weight;
datalines;
Alfred M 14 69.0 112.5
Carol F 14 62.8 102.5
James M 13 57.3 83.0
Janet F 15 62.5 112.5
Judy F 14 64.3 90.0
Philip M 16 72.0 150.0
William M 15 66.5 112.0
;
proc print data=growth.teens;
   where age >13;
run;
```

The output is shown:

***Output 3.4*** *STARTOBS=*



The SAS System

| Obs | Name | Sex | Age | Height | Weight |
|---|---|---|---|---|---|
| 2 | Carol | F | 14 | 62.8 | 102.5 |
| 4 | Janet | F | 15 | 62.5 | 112.5 |
| 5 | Judy | F | 14 | 64.3 | 90.0 |
| 6 | Philip | M | 16 | 72.0 | 150.0 |
| 7 | William | M | 15 | 66.5 | 112.0 |

## TEMP= LIBNAME Statement Option

Specifies to store the library in a temporary subdirectory of the primary path.

**Default:** NO

**Engine:** SPD Engine only

### Syntax

**TEMP=**YES | NO

#### *Required Arguments*

**YES**
specifies to create the temporary subdirectory.

**NO**
specifies not to create a temporary subdirectory.

## Details

The TEMP= option creates a temporary subdirectory of the primary directory that was named in the LIBNAME statement. The subdirectory and all component files are deleted at the end of the session.

You can use TEMP= with the SAS option USER= to store temporary data sets that can be referenced with a single-level name.

*Note:* When using the SIGNON statement in SAS/CONNECT software, the INHERITLIB= option cannot refer to an SPD Engine library that was defined with the TEMP= option.

## Example: Creating a Temporary Library

The following example illustrates two features:

- the use of the TEMP= LIBNAME option to create a temporary library
- the use of the USER= system option to enable the use of single-level table names for SPD Engine tables

A directory is created under MyData. The MasterCopy data set has its metadata file stored in a subdirectory of MyData. The data and index for MasterCopy are created in the locations specified in the DATAPATH= and INDEXPATH= options.

```
libname perm <masterdata>
libname mywork spde 'mydata'
   datapath=('/data01/mypath' '/data02/mypath' '/data03/mypath' '/data04/mypath')
   indexpath=('index/mypath') TEMP=YES;
option user=mywork;
data mastercopy (index=(lastname));
   set perm.customer;
   where region='W';
run;
```

# *Chapter 4*
# SPD Engine Data Set Options

## Introduction to SPD Engine Data Set Options

Specifying data set options for the SPD Engine is the same as specifying data set options for the default Base SAS engine or SAS/ACCESS engines. This section provides details about data set options that are used only with the SPD Engine. The default Base SAS engine data set options that affect the SPD Engine are also listed.

When using the options, remember that the value of the data set option overrides the value of its corresponding LIBNAME option.

## Syntax

(*option-1*=*value-1* <(*option-2*=*value-2*>...)

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

## SPD Engine Data Set Options List

ALIGN=
: specifies variable alignment.

ASYNCINDEX=
: specifies to create indexes in parallel when creating multiple indexes on an SPD Engine data set.

BYNOEQUALS=
: specifies the index output order of data set observations that have identical values for the BY variable.

BYSORT=
: specifies the SPD Engine to perform an automatic sort when it encounters a BY statement. BYSORT= is also a LIBNAME statement option.

COMPRESS=
: specifies to compress SPD Engine data sets on disk as they are being created. COMPRESS= is also a LIBNAME statement option.

*Note:* Compression and encryption are mutually exclusive in the SPD Engine.

ENCRYPT=
: specifies whether to encrypt an output SPD Engine data set.

*Note:* Compression and encryption are mutually exclusive in the SPD Engine.

ENCRYPTKEY=
: specifies the key value for AES encryption.

ENDOBS=
: specifies the end observation number in a user-defined range of observations to be processed. ENDOBS= is also a LIBNAME statement option.

IDXBY=
: specifies whether to use an index when processing a BY statement in the SPD Engine. IDXBY= is also a LIBNAME statement option.

IDXWHERE=
: specifies whether to use an index when processing a WHERE expression in the SPD Engine.

IOBLOCKSIZE=
: specifies the size in bytes of a block of observations to be compressed. IOBLOCKSIZE= is also a LIBNAME statement option.

LISTFILES=
: specifies whether the CONTENTS procedure lists the complete pathnames of all of the component files in an SPD Engine data set.

PADCOMPRESS=
specifies the number of bytes to add to compressed blocks in a data set opened for OUTPUT or UPDATE.

PARTSIZE=
specifies the maximum size that the data component partitions can be. PARTSIZE= is also a LIBNAME statement option.

STARTOBS=
specifies the starting observation number in a user-defined range of observations to be processed. STARTOBS= is also a LIBNAME statement option.

SYNCADD=
specifies to process one observation at a time or a block of observations at a time.

THREADNUM=
specifies the maximum number of threads to use for SPD Engine processing.

UNIQUESAVE=
specifies to save (in a separate file) any observations that were rejected because of nonunique key values during an append or insert to a data set with unique indexes when SYNCADD=NO.

WHERENOINDEX=
specifies a list of indexes to exclude when making WHERE expression evaluations.

## SAS Data Set Options That Behave Differently with the SPD Engine Than with the Default Base SAS Engine

CNTLLEV=
only the value MEM is accepted

COMPRESS=
no user-supplied values are accepted

***CAUTION:***

**Compression and encryption are mutually exclusive in the SPD Engine.** If you are copying a default Base SAS engine data set to an SPD Engine data set and the data set is compressed and encrypted, the compression is dropped. You cannot create an SPD Engine data set with both encryption and compression.

DLDMGACTION=
does not support DLDMGACTION=NOINDEX, but does support ABORT, FAIL, PROMPT, and REPAIR.

ENCRYPT=
encrypts data files

***CAUTION:***

**Compression and encryption are mutually exclusive in SPD Engine.**

## SAS Data Set Options Not Supported by the SPD Engine

- BUFNO=
- BUFSIZE=
- ENCODING=
- EXTENDOBSCOUNTER=
- FIRSTOBS=
- GENMAX=
- GENNUM=
- IDXNAME=
- OUTREP=
- POINTOBS=
- REUSE=
- TOBSNO=

## Dictionary

### ALIGN= Data Set Option

specifies variable alignment.

| | |
|---|---|
| **Valid in:** | DATA step and PROC step |
| **Default:** | YES |
| **Restriction:** | Use only with SPD Server |
| **Engine:** | SPD Engine only |

#### Syntax

**ALIGN=**YES | NO

##### *Required Arguments*

**YES**
enables variable alignment.

**NO**
disables variable alignment to allow the SPD Engine data set to be compatible with the SPD Server.

## Details

### Variable Alignment

Base SAS imposes proper numeric data alignment on an observation on disk by the careful arrangement of the variables within the observation. Like the default Base SAS engine, the SPD Engine ensures that all of the numeric values are grouped together at the beginning of the observation. It also ensures that the total observation length is an even multiple of 8-bytes by adding extra-padding bytes at the end of the observation when necessary. The SAS memory system ensures that all allocated memory starts on an 8-byte boundary and that all of the numeric values are 8 bytes long. Grouping the numeric values together at the beginning of the observation ensures that they will all be properly aligned and can be used directly from the memory.

An observation in an SPD Engine data set is read from disk into memory. All of the bytes are read at once, and their relative alignment is maintained in memory. For maximum performance when accessing that observation data, all numeric data values need to be properly aligned. Therefore, the need to move the values to a new location is avoided. The normal behavior of the SPD engine is to ensure that all of the numeric values in an observation are aligned on an 8-byte boundary when written to disk. This allows the SPD engine to retrieve the observation data from disk and store it in memory. The SPD Engine application uses the observation data directly from that location without the need to move it.

### Using the SPD Server

The SPD Server does not support the data alignment feature. Data sets created by the SPD Engine that are used by the SPD Server must be created with data alignment disabled. Using the ALIGN=NO option causes the data stored in the data set to not be aligned. As a result, the data must be moved from its original memory location that it was read into and into a different memory location that is aligned.

The purpose of the ALIGN=NO data set option is to specify that no data alignment is done by the SPD Engine. This is not useful in most situations, but it is necessary when using the SPD Server. The SPD Server does not support variable alignment capability. In some situations, the SPD Server will refuse to operate on a data set that has aligned variables.

***CAUTION:***

> **Do not use the ALIGN=NO option unless the data set is destined for the SPD Server.** Unaligned variable data will cause a significant decrease in performance when processed by Base SAS.

## Examples

### Example 1: Data Set with Variable Alignment

The first data set shows the default behavior—it has aligned variables. Note that the observation length is not the sum of the variable lengths. The observation length has been rounded up to be an even multiple of 8. Also, the variables have been rearranged so that the numeric variables come first in the observation.

***Log 4.1*** *Data Set with Variable Alignment*

```
data testdata.size;
   length text $10   width 8    chars 8;
   text='Zero';      width=1;   chars=4;  output;
   text='Ten';       width=2;   chars=3;  output;
   text='Twenty';    width=2;   chars=6;  output;
run;

NOTE: The data set TESTDATA.SIZE has 3 observations and 3 variables.

proc sql;
   select obslen
   from dictionary.tables
   where memname="SIZE";


   Observation
        Length
   -----------
            32

   select  varnum, name, npos, length
   from dictionary.columns
   where memname="SIZE"
   order by npos;

     Column
     Number                                      Column    Column
   in Table  Column Name                       Position    Length
   ---------------------------------------------------------------
          2  width                                    0         8
          3  chars                                    8         8
          1  text                                    16        10

quit;

/* ------------------------------------------------------------------- */

data testdata.size (align=no);
   length text $10   width 8    chars 8;
   text='Zero';      width=1;   chars=4;  output;
   text='Ten';       width=2;   chars=3;  output;
   text='Twenty';    width=2;   chars=6;  output;
run;

NOTE: The data set TESTDATA.SIZE has 3 observations and 3 variables.
```

### *Example 2: Data Set without Variable Alignment*

The second data set is identical except that the variables are not aligned. Note that the observation length is just the sum of the variable lengths. The observation length is not rounded to an even multiple of 8-bytes. The variables appear in the order in which they were encountered in the DATA step in the LENGTH statement. The variables were not rearranged for alignment.

***Log 4.2** Data Set without Variable Alignment*

```
data testdata.size (align=no);
   length text $10   width 8    chars 8;
   text='Zero';      width=1;   chars=4;  output;
   text='Ten';       width=2;   chars=3;  output;
   text='Twenty';    width=2;   chars=6;  output;
run;

NOTE: The data set TESTDATA.SIZE has 3 observations and 3 variables.

proc sql;
   select obslen
   from dictionary.tables
   where memname="SIZE";

   Observation
        Length
   -----------
            26

   select varnum, name, npos, length
   from dictionary.columns
   where memname="SIZE"
   order by npos;


     Column
     Number                                      Column    Column
   in Table  Column Name                       Position    Length
   --------------------------------------------------------------
          1  text                                     0        10
          2  width                                   10         8
          3  chars                                   18         8
quit;

NOTE: The data set TESTDATA.SIZE has 3 observations and 3 variables.
```

## ASYNCINDEX= Data Set Option

Specifies to create indexes in parallel when creating multiple indexes on an SPD Engine data set.

**Valid in:** DATASETS procedure or with the INDEX data set option

**Default:** NO

**Engine:** SPD Engine only

### Syntax

**ASYNCINDEX=**YES | NO

#### *Required Arguments*

**YES**
creates the indexes in parallel (asynchronously).

**NO**
creates one index at a time (synchronously).

## Details

The SPD Engine can create multiple indexes with a single scan of a data set. The SPD Engine spawns a single thread for each index created, and then processes the threads simultaneously. Although creating indexes in parallel is much faster than scanning the data set for each index, the default for this option is NO because parallel index creation requires extra utility space to store the sorting files and requires additional memory. If index creation fails due to insufficient resources, you can do one or both of the following:

- Increase the size of the utility file space using the SPDEUTILLOC= system option.
- Set the SAS system option to MEMSIZE=0[1] and increase the utility space that is used for index sorting using the SPDEINDEXSORTSIZE= system option.

## Example: Creating Indexes in Groups

The DATASETS procedure has the flexibility to use batched parallel index creation using multiple MODIFY groups. Instead of creating all of the indexes at once, which would require a significant amount of space, you can create the indexes in groups as shown in the following example. Indexes PatientNo and PatientClass are created together as are the indexes LastName and FirstName. The other indexes are created serially.

```
proc datasets lib=main;
   modify patients(asyncindex=yes);
      index create PatientNo PatientClass;
   run;
   modify patients(asyncindex=yes);
      index create LastName FirstName;
   run;
   modify patients(asyncindex=no);
      index create FullName=(LastName FirstName)
        ClassSex=(PatientClass PatientSex);
   run;
quit;
```

# BYNOEQUALS= Data Set Option

Specifies whether the output order of data set observations that have identical values for the BY variable is guaranteed to be in the data set order.

| | |
|---|---|
| **Valid in:** | DATA step and PROC step |
| **Used by:** | BYSORT=YES data set option |
| **Default:** | NO |
| **Engine:** | SPD Engine only |

## Syntax

**BYNOEQUALS=**YES | NO

[1] For z/OS, increase the REGION size.

### Required Arguments

**YES**
does not guarantee that the output order of data set observations that have identical values for the BY variable is in data set order.

**NO**
guarantees that the output order of data set observations that have identical values for the BY variable is in data set order.

## Details

When a group of observations that have identical values for the BY statement is output, the order of the observations in the output is the same as the data set order. This happens because the default is BYNOEQUALS=NO. By specifying YES, the processing time is decreased, but the observations are not guaranteed to be output in the data set order.

The data set or LIBNAME option BYSORT= must be YES (the default) because the BYNOEQUALS= option has no effect when BYSORT=NO.

The following table shows when the SPD Engine preserves physical order in the output:

***Table 4.1*** *SPD Engine Preserves Physical Order*

| Condition: | Data Set Order Preserved? |
|---|---|
| If BY is present | YES (BYNOEQUALS=NO and BYSORT=YES by default) |
| If BY is present and BYNOEQUALS=YES | NO |
| If BY is present and BYSORT=NO | YES (because no automatic sort occurs) |
| If neither BY nor WHERE is present | YES |
| If WHERE is present | NO |

## Examples

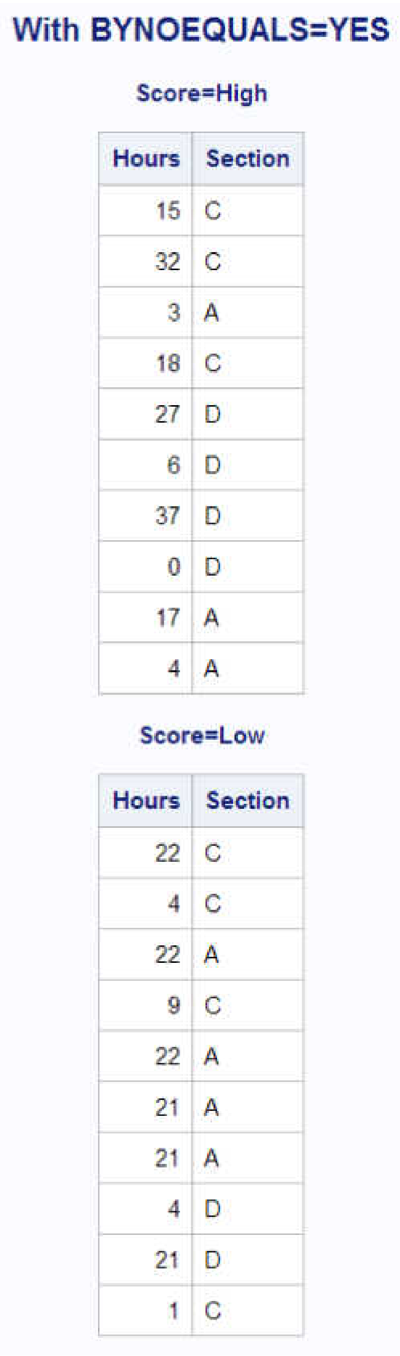
### Example 1: BYNOEQUALS=YES

In the following example, the observations that have identical BY values on the key variable are output in unpredictable order because BYNOEQUALS=YES:

```
title 'With BYNOEQUALS=YES';
proc print data=labs.performance(bynoequals=yes) noobs;
   by score;
run;
```

The output is shown:

***Output 4.1*** *BYNOEQUALS=YES*



With BYNOEQUALS=YES

Score=High

| Hours | Section |
|---|---|
| 15 | C |
| 32 | C |
| 3 | A |
| 18 | C |
| 27 | D |
| 6 | D |
| 37 | D |
| 0 | D |
| 17 | A |
| 4 | A |

Score=Low

| Hours | Section |
|---|---|
| 22 | C |
| 4 | C |
| 22 | A |
| 9 | C |
| 22 | A |
| 21 | A |
| 21 | A |
| 4 | D |
| 21 | D |
| 1 | C |

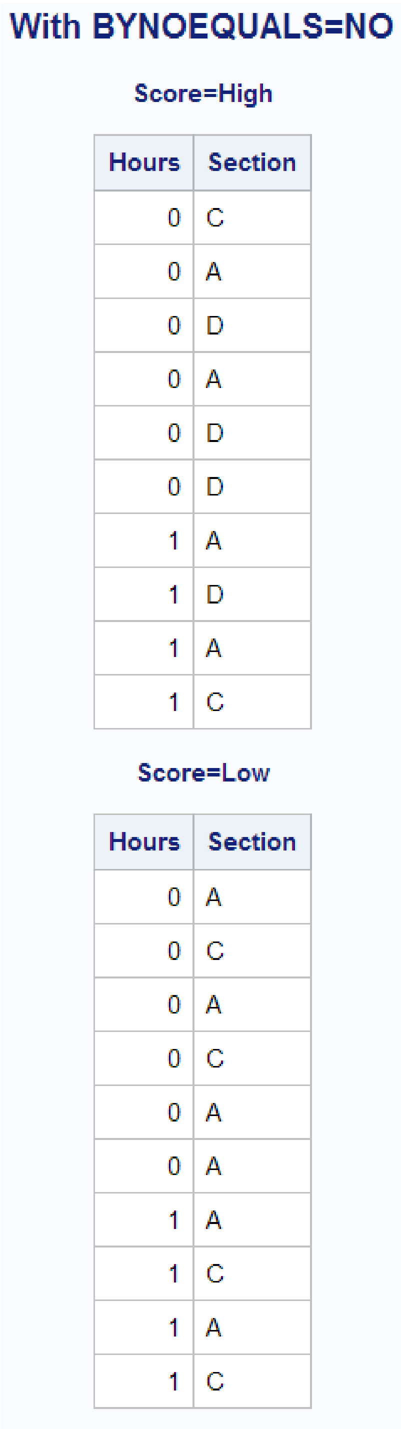
## *Example 2: BYNOEQUALS=NO*

The following example shows the output with BYNOEQUALS=NO:

```
title 'With BYNOEQUALS=NO;
proc print data=labs.performance(bynoequals=no) noobs;
   by score;
run;
```

The output is shown:

***Output 4.2*** *BYNOEQUALS=NO*



**With BYNOEQUALS=NO**

**Score=High**

| Hours | Section |
|---|---|
| 0 | C |
| 0 | A |
| 0 | D |
| 0 | A |
| 0 | D |
| 0 | D |
| 1 | A |
| 1 | D |
| 1 | A |
| 1 | C |

**Score=Low**

| Hours | Section |
|---|---|
| 0 | A |
| 0 | C |
| 0 | A |
| 0 | C |
| 0 | A |
| 0 | A |
| 1 | A |
| 1 | C |
| 1 | A |
| 1 | C |

## BYSORT= Data Set Option

Specifies the SPD Engine to perform an automatic sort when it encounters a BY statement.

**Valid in:** DATA step and PROC step

**Default:** YES

**Interaction:** "BYNOEQUALS= Data Set Option" on page 54

**Engine:** SPD Engine only

## Syntax

**BYSORT**=YES | NO

### *Required Arguments*

**YES**
specifies to automatically sort the data based on the BY variables when a BY statement is encountered instead of sorting the data ahead of time.

**NO**
specifies not to sort the data based on the BY variables. Specifying NO means that the data must already be sorted before the BY statement. Indexes are not used.

## Details

DATA or PROC step processing using the default Base SAS engine requires that if there is no index or if the observations are not in order, the data set must be sorted before a BY statement is issued. In contrast, by default, the SPD Engine sorts the data returned to the application if the observations are not in order. Unlike PROC SORT, which creates a new sorted data set, the SPD Engine's automatic sort does not change the permanent data set and does not create a new data set. However, utility file space is used. For more information, see "SPDEUTILLOC= System Option" on page 102.

The default is BYSORT=YES. A BYSORT=YES argument enables the automatic sort, which outputs the observations in BY group order. If the data set option BYNOEQUALS=YES, then the observations within a group might be output in a different order from the order in the data set. Set BYNOEQUALS=NO to retain data set order.

The BYSORT=NO argument means that the data must already be sorted before the BY statement. Sorting can be from a previous PROC SORT or from the data set having been created in BY variable order. An error occurs if the data set is not sorted. When BYSORT=NO, grouped data is delivered to the application in data set order. Indexes are not used to retrieve the observations in BY variable order. The data set option BYNOEQUALS= has no effect when BYSORT=NO.

If you specify the BYSORT= option in the LIBNAME statement, it can be overridden by specifying BYSORT= in the PROC or DATA steps. Set BYSORT=YES in the DATA or PROC step, for input opens, to override BYSORT=NO in the LIBNAME statement. The point is that BYSORT=NO instructs the engine to do nothing to sort the data.

When you use the BYSORT=YES and the IDXWHERE= data set options, the following messages are written to the SAS log if you set the MSGLEVEL=I SAS system option:

- If IDXWHERE=YES and there is an index on the BY variable, the index is used to order the rows of the table. The following message is written to the SAS log:

  ```
  Note: BY ordering was produced by using an index for table tablename.
  ```

- If IDXWHERE=NO or IDXWHERE=YES and there is no index on the BY variable, SPD Engine performs an automatic sort to order the rows of the table. The following message is written to the SAS log:

  ```
  Note: BY ordering was produced by performing an automatic sort on
        table tablename.
  ```

## Comparisons

The BYSORT= data set option overrides the BYSORT= LIBNAME statement option.

## Examples

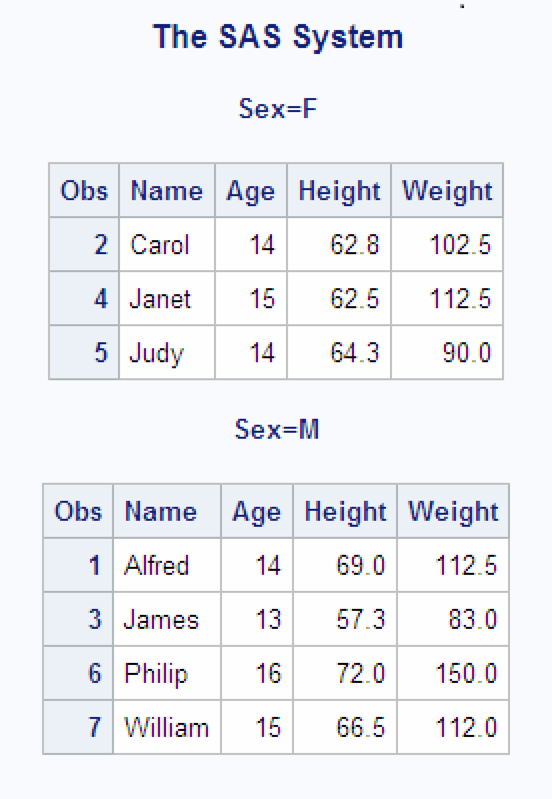
### *Example 1: Group Formatting with BYSORT=YES by Default*

```
libname growth spde 'SAS-library';
data growth.teens;
   input Name $ Sex $ Age Height Weight;
datalines;
Alfred M 14 69.0 112.5
Carol F 14 62.8 102.5
James M 13 57.3 83.0
Janet F 15 62.5 112.5
Judy F 14 64.3 90.0
Philip M 16 72.0 150.0
William M 15 66.5 112.0
;
proc print data=growth.teens; by sex;
run;
```

Even though the data was not sorted using PROC SORT, no error occurred because BYSORT=YES is the default.

The output is shown:

***Output 4.3*** *Group Formatting with BYSORT=YES by Default*



**The SAS System**

Sex=F

| Obs | Name | Age | Height | Weight |
|---|---|---|---|---|
| 2 | Carol | 14 | 62.8 | 102.5 |
| 4 | Janet | 15 | 62.5 | 112.5 |
| 5 | Judy | 14 | 64.3 | 90.0 |

Sex=M

| Obs | Name | Age | Height | Weight |
|---|---|---|---|---|
| 1 | Alfred | 14 | 69.0 | 112.5 |
| 3 | James | 13 | 57.3 | 83.0 |
| 6 | Philip | 16 | 72.0 | 150.0 |
| 7 | William | 15 | 66.5 | 112.0 |

### *Example 2: BYSORT=NO*

With BYSORT=NO in the PROC PRINT statement, SAS returns an error whenever automatic sorting is suppressed (BYSORT=NO). The data must be sorted on the BY variable before the BY statement (for example, by using PROC SORT).

```
libname growth spde 'SAS-library';
proc print data=growth.teens (bysort=no);
by sex;
run;
ERROR: Data set GROWTH.TEENS is not sorted in ascending sequence.
       The current BY-group has Sex = M and the next BY-group has Sex = F.
NOTE: The SAS System stopped processing this step because of errors.
```

## COMPRESS= Data Set Option

Specifies to compress SPD Engine data sets on disk as they are being created.

| | |
|---|---|
| **Valid in:** | DATA step and PROC step |
| **Default:** | NO |
| **Restriction:** | Cannot be used with ENCRYPT=YES or ENCRYPT=AES |
| **Interactions:** | “IOBLOCKSIZE= Data Set Option” on page 73<br>“PADCOMPRESS= Data Set Option” on page 77 |
| **Engine:** | SPD Engine only |

### Syntax

**COMPRESS=**NO | YES | CHAR | BINARY

### *Required Arguments*

**NO**
performs no data set compression.

**YES | CHAR**
specifies that data in an SPD Engine data set be compressed in blocks by using RLE (run-length encoding). RLE compresses data by reducing repeated runs of the same character (including a blank space) to two-byte or three-byte representations.

**BINARY**
specifies that data in an SPD Engine data set be compressed in blocks by using RDC (Ross Data Compression). RDC combines RLE and sliding window compression to compress the file by representing repeated byte patterns more efficiently.

*Note:* This method is highly effective for compressing medium to large (several hundred bytes or larger) blocks of binary data (character and numeric variables).

### Details

When you specify COMPRESS=YES | BINARY | CHAR, the SPD Engine compresses, by blocks, the data component file as it is created. To specify the size of the compressed blocks, use the “IOBLOCKSIZE= Data Set Option” on page 73 when you create the data set. To add padding to the newly compressed blocks, specify “PADCOMPRESS=

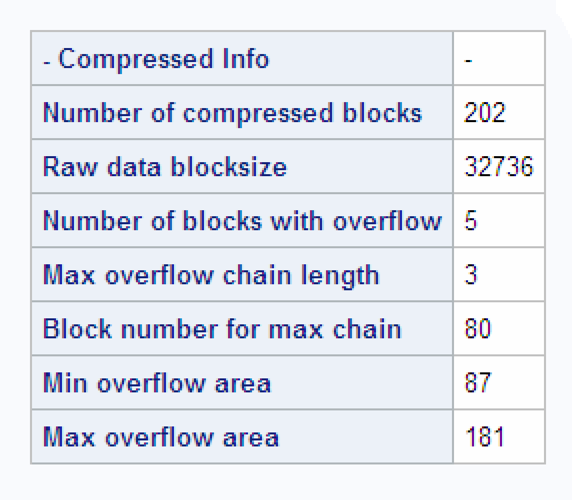
Data Set Option" on page 77 when creating or updating the data set. For more information, see "Compressing SPD Engine Data Sets" on page 18.

The SPD Engine does not support user-specified compression. If you are migrating a default Base SAS engine data set that is both compressed and encrypted, the encryption is retained, but the compression is dropped.

The CONTENTS procedure identifies the compress setting. If the data set is compressed, PROC CONTENTS prints information about the compression. The following example explains the Compressed Info fields in the CONTENTS procedure output:

In general, COMPRESS=CHAR provides good compression when single bytes repeat; COMPRESS=BINARY provides good compression when strings of bytes repeat. At the same time, it is more costly to look for strings of bytes that repeat, than to look for single bytes that repeat. For examples, see "Example 1: COMPRESS=CHAR" on page 62 and "Example 2: COMPRESS=BINARY" on page 62.

***Output 4.4*** *PROC CONTENTS Compressed Section*



| - Compressed Info | - |
|---|---|
| Number of compressed blocks | 202 |
| Raw data blocksize | 32736 |
| Number of blocks with overflow | 5 |
| Max overflow chain length | 3 |
| Block number for max chain | 80 |
| Min overflow area | 87 |
| Max overflow area | 181 |

Number of compressed blocks
: number of compressed blocks that are required to store data.

Raw data blocksize
: compressed block size in bytes calculated from the size specified in the IOBLOCKSIZE= data set option. It is the largest multiple of the observation length that gets in the block size.

Number of blocks with overflow
: number of compressed blocks that needed more space. When data is updated and the compressed new block is larger than the compressed old block, an overflow block fragment is created.

Max overflow chain length
: largest number of overflows for a single block. For example, the maximum overflow chain length would be 2 if a compressed block was updated and became larger, and then updated again to a larger size.

Block number for max chain
: number of the block containing the largest number of overflow blocks.

Min overflow area
: minimum amount of disk space that an overflow requires.

Max overflow area
: maximum amount of disk space that an overflow requires.

Accessing compressed files usually requires more processing time. The files have to be decompressed before reading them and, if updating, they have to be compressed again when written to disk.

## Comparisons

The COMPRESS= data set option overrides the COMPRESS= LIBNAME statement option and the COMPRESS= system option.

## Examples

### *Example 1: COMPRESS=CHAR*

```
data mylib.CharRepeats(compress=char);
   length ca $ 200;
   do i=1 to 100000;
      ca='aaaaaaaaaaaaaaaaaaaaaa';
      cb='bbbbbbbbbbbbbbbbbbbbbb';
      cc='cccccccccccccccccccccc';
      output;
   end;
run;
```

The following message is written to the SAS log:

```
NOTE: Compressing data set MYLIB.CHARREPEATS decreased size by 88.55 percent.
      Compressed is 45 pages; un-compressed would require 393 pages.
```

### *Example 2: COMPRESS=BINARY*

```
data mylib.StringRepeats(compress=binary);
   length cabcd $ 200;
   do i=1 to 1000000;
      cabcd='abcdabcdabcdabcdabcdabcdabcdabcd';
      cefgh='efghefghefghefghefghefghefghefgh';
      cijkl='ijklijklijklijklijklijklijklijkl';
      output;
   end;
run;
```

The following message is written to the SAS log:

```
NOTE: Compressing data set MYLIB.STRINGREPEATS decreased size by 70.27 percent.
      Compressed is 1239 pages; un-compressed would require 4167 pages.
```

# ENCRYPT= Data Set Option

Specifies whether to encrypt an output SPD Engine data set.

**Valid in:** DATA step and PROC step

**Default:** NO

**Restrictions:** Use only with output data sets

ENCRYPT=YES or ENCRYPT=AES cannot be used with COMPRESS=

## Syntax

**ENCRYPT=** AES | NO | YES

### *Syntax Description*

**AES**

encrypts the data set using the AES (Advanced Encryption Standard) algorithm. AES provides stronger encryption using SAS/SECURE software, which is included with Base SAS software. You must use the ENCRYPTKEY= data set option when using ENCRYPT=AES. For more information, see "ENCRYPTKEY= Data Set Option" on page 65.

CAUTION **Record all ENCRYPTKEY= key values if you specify ENCRYPT=AES.** If you forget the ENCRYPTKEY= key value, you lose your data. SAS cannot assist you in recovering the ENCRYPTKEY= key value. The following note is written to the log:

```
Note: If you lose or forget the ENCRYPTKEY= key value, there will
be no way to open the file or recover the data.
```

**NO**

does not encrypt the data set.

**YES**

encrypts the data set using the SAS Proprietary algorithm. This encryption method uses passwords that are stored in the data set. At a minimum, you must specify the READ= or the PW= data set option at the same time that you specify ENCRYPT=YES. Because the encryption method uses passwords, you cannot change any password on an encrypted data set without re-creating the data set.

***CAUTION:***

**Record all passwords if you specify ENCRYPT=YES.** If you forget a password, you cannot reset it without assistance from SAS. The process is time-consuming and resource-intensive.

## Details

Encryption and compression are mutually exclusive in SPD Engine.

You cannot create an SPD Engine data set with both encryption and compression. If you use ENCRYPT=YES or ENCRYPT=AES and the COMPRESS= data set or LIBNAME option, the following error is generated:

```
ERROR: The data set was not created because compression and
       encryption cannot both be specified.
```

You cannot copy a Base SAS data set that is both compressed and encrypted to an SPD Engine library.

When using ENCRYPT=YES, the following rules apply:

- To copy an encrypted data set, the output engine must support encryption. Otherwise, the data set is not copied.

- If the data set is encrypted, all associated index files and metadata files are also encrypted.
- Encryption requires approximately the same amount of CPU resources as compression.
- You cannot use PROC CPORT on SAS Proprietary-encrypted data sets.

When using ENCRYPT=AES, the following rules apply:

- You must use the ENCRYPTKEY= data set option when creating a data set with AES encryption.
- To copy an AES-encrypted data set, the output engine must support AES encryption. Otherwise, the data set is not copied.
- If the data set is AES-encrypted, all associated index files are also AES-encrypted.
- Releases before SAS 9.4 cannot use an AES-encrypted data set.
- You use SAS/SECURE software, which is included with Base SAS software, to use AES encryption.

You cannot change the ENCRYPTKEY= key value on an AES-encrypted data set without re-creating the data set.

## Examples

### *Example 1: Using ENCRYPT=YES Option*

The following example uses the SAS Proprietary algorithm:

```
libname depta spde 'SAS-library';
data salary(encrypt=yes read=green);
 input name $ yrsal bonuspct;
datalines;
Muriel    34567  3.2
Bjorn     74644  2.5
Freda     38755  4.1
Benny     29855  3.5
Agnetha   70998  4.1
;
```

To use this data set, specify the Read password:

```
proc contents data=salary(read=green);
run;
```

### *Example 2: Using ENCRYPT=AES Option*

The following example uses the AES algorithm:

```
data salary(encrypt=aes encryptkey=green);
   input name $ yrsal bonuspct;
   datalines;
Muriel    34567  3.2
Bjorn     74644  2.5
Freda     38755  4.1
Benny     29855  3.5
Agnetha   70998  4.1
```

To use this data set, specify the ENCRYPTKEY= key value:

```
proc contents data=salary(encryptkey=green);
run;
```

### *Example 3: Copying AES-Encrypted Data Sets*

The following are two examples of using ENCRYPTKEY= data set options and the COPY procedure:

```
PROC COPY IN=inlib OUT=outlib ENCRYPTKEY=secret;
    SELECT abc (ENCRYPTKEY=secreta) DEF(ENCRYPTKEY=secretb)…



PROC COPY IN=inlib OUT=outlib;
    SELECT abc (ENCRYPTKEY=secreta) DEF(ENCRYPTKEY=secretb)…
```

## ENCRYPTKEY= Data Set Option

Specifies a key value for AES encryption.

**Valid in:** DATA step and PROC step

**Range:** 1 to 64 bytes

**Restrictions:** Use with SAS 9.4 or later only

Use only with AES-encrypted data sets

## Syntax

**ENCRYPTKEY=***key-value*

### *Syntax Description*

***key-value***

assigns an encrypt key value. You must use the ENCRYPTKEY= data set option with ENCRYPT=AES. The key value can be up to 64-bytes long. You are able to create an ENCRYPTKEY= key value with or without quotation marks using the following rules:

no quotation marks

- alphanumeric characters and underscores only
- up to 64-bytes
- uppercase and lowercase letters
- must start with a letter
- no blank spaces
- is not case sensitive

```
encryptkey=key-value
encryptkey=key-value1
```

single quotation marks

- alphanumeric, special, and DBCS characters

- up to 64-bytes
- uppercase and lowercase letters
- is case sensitive

```
encryptkey='key-value'
encryptkey='1234*#mykey'
```

double quotation marks

- alphanumeric, special, and DBCS characters
- up to 64-bytes
- uppercase and lowercase letters
- enables macro resolution
- is case sensitive

```
encryptkey="key-value"
encryptkey="1234*#mykey"
%let mykey=abcdefghi12;
encryptkey=&key-value
```

When the ENCRYPTKEY= key value uses DBCS characters, the 64-byte limit applies to the character string after it has been transcoded to UTF-8 encoding. You can use the following DATA step to calculate the length in bytes of a key value in DBCS:

```
data _null_;
    key=length(unicodec('key-value','UTF8'));
    put 'key length=' key;
run;
```

**Note** You cannot change the ENCRYPTKEY= key value on an AES-encrypted data set without re-creating the data set.

## Details

***CAUTION:***

**You must remember the ENCRYPTKEY= key value.** If you forget the ENCRYPTKEY= key value, you lose your data. SAS cannot assist you in recovering the ENCRYPTKEY= key value.

You must use the ENCRYPTKEY= data set option when creating or accessing an SPD Engine data set with AES encryption.

The ENCRYPTKEY= data set option does not protect the data set from deletion or replacement. Encrypted data sets can be deleted using any of the following scenarios without having to specify a key value:

- the KILL option in PROC DATASETS
- the DROP statement in PROC SQL
- the DELETE procedure

The ENCRYPTKEY= data set option prevents access only to the contents of the data set. To protect the data set from deletion or replacement, the data set must also contain an ALTER= password.

You must specify the ENCRYPTKEY= key value when you copy AES-encrypted data sets. The key value follows the data set name in the SELECT statement. The following example uses the SELECT statement:

```
copy in=OldLib out=NewLib;
    select salary(encryptkey=key-value);
run;
```

When working with data sets protected by the ENCRYPTKEY= key value in the DATASETS procedure, you can specify the key value in the AGE, APPEND, CONTENTS, and MODIFY statements. The ENCRYPTKEY= data set option can be specified either in parentheses after the name of the SAS data set or after a forward slash.

It is possible to use a macro variable as the ENCRYPTKEY= key value. To use a macro variable, you must use double quotation marks. The following code defines a macro variable:

```
%let secret=MyValue;
```

The following code uses the macro variable as the ENCRYPTKEY= key value:

```
data my.dsname(encrypt=aes encryptkey="&secret");
```

When you specify a macro variable as the ENCRYPTKEY= key value, you must enclose the macro variable in double quotation marks. If you do not use the double quotation marks, unpredictable results can occur.

## Example: Using ENCRYPTKEY= Data Set Option

This example uses the ENCRYPT=AES option:

```
data spdelib.salary(encrypt=aes encryptkey=green);
   input name $ yrsal bonuspct;
   datalines;
Muriel    34567  3.2
Bjorn     74644  2.5
Freda     38755  4.1
Benny     29855  3.5
Agnetha   70998  4.1
```

To use this data set, specify the ENCRYPTKEY= key value:

```
proc contents data=spdelib.salary(encryptkey=green);
run;
```

# ENDOBS= Data Set Option

Specifies the end observation number in a user-defined range of observations to be processed.

| | |
|---|---|
| **Valid in:** | DATA step and PROC step |
| **Default:** | The last observation in the data set |
| **Restrictions:** | Use ENDOBS= with input data sets only |
| | Cannot be used with the OBS= system or data set option or the FIRSTOBS= system and data set option |
| **Interactions:** | “ENDOBS= LIBNAME Statement Option” on page 35 |
| | “STARTOBS= LIBNAME Statement Option” on page 43 |

"STARTOBS= Data Set Option" on page 80

**Engine:** SPD Engine only

## Syntax

**ENDOBS=***n*

### Required Argument

***n***

is the number of the end observation.

## Details

### Specifying a Range of Observations

By default, the SPD Engine processes all of the observations in the entire data set unless you specify a range of observations with the STARTOBS= or ENDOBS= options. If the STARTOBS= option is used without the ENDOBS= option, the implied value of ENDOBS= is the end of the data set. When both options are used together, the value of ENDOBS= must be greater than the value of STARTOBS=.

The ENDOBS= data set option in the SPD Engine works the same way as the OBS= data set option in the default Base SAS engine. The only difference is when ENDOBS= is specified in a WHERE expression.

### Using ENDOBS= with a WHERE Expression

When ENDOBS= is used in a WHERE expression, the ENDOBS= value represents the last observation to process, rather than the number of observations to return. The following examples show the difference.

*Note:* The OBS= system option and the OBS= data set option cannot be used with STARTOBS= or ENDOBS= data set or LIBNAME options.

## Comparisons

The ENDOBS= data set option overrides the ENDOBS= LIBNAME statement option.

## Examples

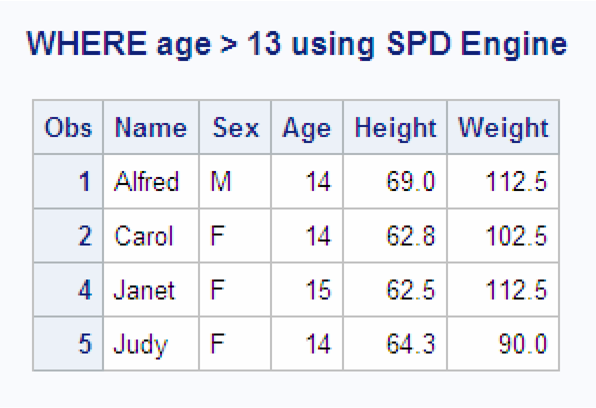
### Example 1: Using the ENDOBS= Data Set Option

A data set is created and processed by the SPD Engine with ENDOBS=5 option specified. The WHERE expression is applied to the data set ending with observation number 5. The PRINT procedure prints four observations, which are the observations qualified by the WHERE expression.

```
libname growth spde 'SAS-library';
data growth.teens;
   input Name $ Sex $ Age Height Weight;
   list;
datalines;
Alfred M 14 69.0 112.5
Carol F 14 62.8 102.5
```

```
James M 13 57.3 83.0
Janet F 15 62.5 112.5
Judy F 14 64.3 90.0
Philip M 16 72.0 150.0
Zeke M 14 71.1 105.0
Alice F 14 65.1 91.0
William M 15 66.5 112.0
;
proc print data=growth.teens (endobs=5);
   where age >13;
   title 'WHERE age > 13 using SPD Engine';
run;
```

***Output 4.5*** *ENDOBS=*



WHERE age > 13 using SPD Engine

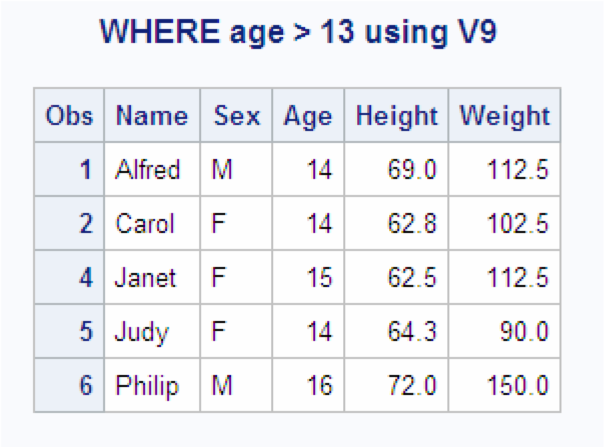
| Obs | Name | Sex | Age | Height | Weight |
|---|---|---|---|---|---|
| 1 | Alfred | M | 14 | 69.0 | 112.5 |
| 2 | Carol | F | 14 | 62.8 | 102.5 |
| 4 | Janet | F | 15 | 62.5 | 112.5 |
| 5 | Judy | F | 14 | 64.3 | 90.0 |

### *Example 2: OBS= with SPD Engine*

The same data set is processed with OBS=5 specified. PROC PRINT prints five observations, which are all of the observations qualified by the WHERE expression, ending with the fifth qualified observation.

```
libname growth spde 'SAS-library';
data growth.teens;
   input Name $ Sex $ Age Height Weight;
   list;
datalines;
Alfred M 14 69.0 112.5
Carol F 14 62.8 102.5
James M 13 57.3 83.0
Janet F 15 62.5 112.5
Judy F 14 64.3 90.0
Philip M 16 72.0 150.0
Zeke M 14 71.1 105.1
Alice F 14 65.1 91.0
William M 15 66.5 112.0
;
proc print data=growth.teens (obs=5);
   where age >13;
   title 'WHERE age > 13 using V9';
run;
```

***Output 4.6*** *OBS=*



WHERE age > 13 using V9

| Obs | Name | Sex | Age | Height | Weight |
|---|---|---|---|---|---|
| 1 | Alfred | M | 14 | 69.0 | 112.5 |
| 2 | Carol | F | 14 | 62.8 | 102.5 |
| 4 | Janet | F | 15 | 62.5 | 112.5 |
| 5 | Judy | F | 14 | 64.3 | 90.0 |
| 6 | Philip | M | 16 | 72.0 | 150.0 |

## IDXBY= Data Set Option

Specifies whether to use an index when processing a BY statement in the SPD Engine.

**Valid in:** DATA step and PROC step

**Default:** YES

**Engine:** SPD Engine only

### Syntax

**IDXBY**=YES | NO

#### *Required Arguments*

**YES**

uses an index when processing indexed variables in a BY statement.

*Note:* If the BY statement specifies more than one variable or the DESCENDING option, then the index is not used, even if IDXBY=YES.

**NO**

does not use an index when processing indexed variables in a BY statement.

Note IDXBY=NO performs an automatic sort when processing a BY statement.

### Details

When you use the IDXBY= data set option, make sure that you use the BYSORT=YES option and that the BY variable is indexed.

In some cases, you might get better performance from the SPD Engine if you automatically sort the data. To use the automatic sort, BYSORT=YES must be set and you should specify IDXBY=NO.

Set the SAS system option MSGLEVEL=I so that the BY processing information is written to the SAS log. When you use the IDXBY= data set option and the BYSORT=YES option, the following messages are written to the SAS log:

- If IDXBY=YES and there is an index on the BY variable, the index is used to order the rows of the table. The following message is written to the SAS log:

  ```
  NOTE: BY ordering was produced by using an index for
        table tablename.
  ```

- If IDXBY=NO, the following message is written to the SAS log:

  ```
  NOTE: BY ordering was produced by performing an automatic sort
        on table tablename.
  ```

## Comparisons

The IDXBY= data set option overrides the IDXBY= LIBNAME statement option.

## Examples

### *Example 1: Using the IDXBY=NO Data Set Option*

```
options msglevel=i;
proc means data=permdata.customer(IDXBY=no);
   by sales;
   by state;
run;
```

The following message is written to the SAS log:

```
NOTE: BY ordering was produced by performing an automatic sort
      on table PERMDATA.customer.
NOTE: There were 2981 observations read from the data set
      PERMDATA.CUSTOMER.
```

### *Example 2: Using the IDXBY=YES Data Set Option*

```
proc means data=permdata.customer(IDXBY=yes);
   var sales;
   by state;
run;
```

The following message is written to the SAS log:

```
NOTE: BY ordering was produced by using an index for table
      PERMDATA.customer.
NOTE: There were 2981 observations read from the data set
      PERMDATA.CUSTOMER.
```

# IDXWHERE= Data Set Option

Specifies whether to use an index when processing a WHERE expression in the SPD Engine.

| | |
|---|---|
| **Valid in:** | DATA step and PROC step |
| **Default:** | YES |
| **Restriction:** | WHERENOINDEX= option cannot be used with IDXWHERE=NO option |
| **Engine:** | SPD Engine only |

## Syntax

**IDXWHERE=**YES | NO

### *Required Arguments*

**YES**

uses an index when processing a WHERE expression.

**NO**

ignores an index when processing a WHERE expression.

| | |
|---|---|
| Restriction | You cannot use the IDXWHERE=NO option and the WHERENOINDEX= option together. |

## Details

IDXWHERE= is used with the SPD Engine's WHERE expression planning software called WHINIT. WHINIT tests the performance of index use with WHERE processing in various applications. Set the SAS system option MSGLEVEL=I so that the WHERE processing information is output to the SAS log.

When you use the IDXWHERE= data set option and the BYSORT=YES option, the following messages are written to the SAS log:

- If IDXWHERE=YES and there is an index on the BY variable, the index is used to order the rows of the table. The following message is written to the SAS log:

  ```
  Note: BY ordering was produced by using an index for
  table tablename.
  ```

- If IDXWHERE=NO or IDXWHERE=YES and there is no index on the BY variable, SPD Engine performs an automatic sort to order the rows of the table. The following message is written to the SAS log:

  ```
  Note: BY ordering was produced by performing an
  automatic sort on table tablename.
  ```

The SPD Engine uses WHINIT, a rules-based WHERE expression planner, to select the most appropriate evaluation strategy for a query. The SAS system option MSGLEVEL=I surfaces WHINIT messages to the SAS log that can help you determine whether one or more indexes are used in a query. For more details about WHINIT, see "SPDEWHEVAL= System Option" on page 104.

*Note: Do not arbitrarily suppress index use when using WHERE and BY statements together.* You use a WHERE statement to filter the observations and a BY statement to order the observations. The filtered observations qualified by the WHERE statement are fed directly into a sort step as part of the parallel WHERE expression evaluation. The final, ordered observation set is produced as the result. Index use in WHERE processing greatly improves the filtering and feeding performance into the sort step.

## Examples

### *Example 1: Using WHINIT Log Output with IDXWHERE=NO*

This example shows that evaluation strategy 2 is used in the WHERE expression because IDXWHERE=NO was specified.

***Log 4.3*** *IDXWHERE=NO*

```
34   options msglevel=i;
35   proc means data=permdata.customer(idxwhere=no);
36      var sales;
37      where state="CA";
38   run;
whinit: WHERE (sstate='CA')
whinit returns: ALL EVAL2
NOTE: There were 2981 observations read from the data set
      PERMDATA.CUSTOMER. WHERE state='CA';
```

### *Example 2: Using WHINIT Log Output with IDXWHERE=YES*

This example shows that evaluation strategy 1 was used because IDXWHERE=YES was specified.

***Log 4.4*** *IDXWHERE=YES*

```
39   proc means data=permdata.customer(idxwhere=yes);
40      var sales;
41      where state="CA";
42   run;
whinit: WHERE (sstate='CA')
 --
whinit: SBM-INDEX STATE uses 45% of segs (WITHIN maxsegratio 75%)
whinit returns: ALL EVAL1(w/SEGLIST)
NOTE: There were 2981 observations read from the data set
      PERMDATA.CUSTOMER. WHERE state='CA';
```

## IOBLOCKSIZE= Data Set Option

Specifies the size in bytes of a block of observations to be used in an I/O operation.

**Valid in:** DATA step and PROC step

**Default:** 1,048,576 bytes (1 megabyte)

**Range:** The minimum block size is 32,768 bytes. The maximum block size is half the size of the SPD Engine data partition file.

**Restriction:** When reading a data set, the block size can significantly affect performance. When retrieving a large percentage of the data, a larger block size improves performance. However, when retrieving a subset of the data such as with WHERE processing, a smaller block size performs better.

**Engine:** SPD Engine only

### Syntax

**IOBLOCKSIZE=***n*

### *Required Argument*

***n***

is the size in bytes of a block of observations.

## Details

The I/O block size determines the amount of data that is physically transferred together in an I/O operation. The larger the block size, the less I/O. The SPD Engine uses blocks in memory to collect the observations to be written to or read from a data component file. The IOBLOCKSIZE= data set option specifies the size of the block. (The actual size is computed to accommodate the largest number of observations that fit in the specified size of *n* bytes. Therefore, the actual size is a multiple of the observation length.)

The block size affects I/O operations for compressed, uncompressed, and encrypted data sets. However, the effects are different and depend on the I/O operation.

- For a compressed data set, the block size determines how many observations are compressed together, which determines the amount of data that is physically transferred for both Read and Write operations. The block size is a permanent attribute of the file. To specify a different block size, you must copy the data set to a new data set, and then specify a new block size for the output file. For a compressed data set, a larger block size can improve performance for both Read and Write operations.
- For an encrypted data set, the block size is a permanent attribute of the file.
- For an uncompressed data set, the block size determines the size of the blocks that are used to read the data from disk to memory. The block size has no affect when writing data to disk. For an uncompressed data set, the block size is not a permanent attribute of the file. That is, you can specify a different block size based on the Read operation that you are performing. For example, reading data that is randomly distributed or reading a subset of the data calls for a smaller block size because accessing smaller blocks is faster than accessing larger blocks. In contrast, reading data that is uniformly or sequentially distributed or that requires a full data set scan works better with a larger block size.

## Comparisons

The IOBLOCKSIZE= data set option overrides the IOBLOCKSIZE= LIBNAME statement option.

## Example: Using IOBLOCKSIZE=

```
/*IOBLOCKSIZE set to 64K */
data sport.maillist(ioblocksize=65536);
/*IOBLOCKSIZE set to 32K */
data sport.maillist(ioblocksize=32768 compress=yes);
```

# LISTFILES= Data Set Option

Specifies whether the CONTENTS procedure lists the complete pathnames of all of the component files of an SPD Engine data set.

**Valid in:** PROC CONTENTS only

**Default:** NO

**Engine:** SPD Engine only

## Syntax

**LISTFILES=**YES | NO

### *Required Arguments*

**YES**
lists the complete pathnames of all of the component files of an SPD Engine data set.

**NO**
does not list the pathnames.

## Details

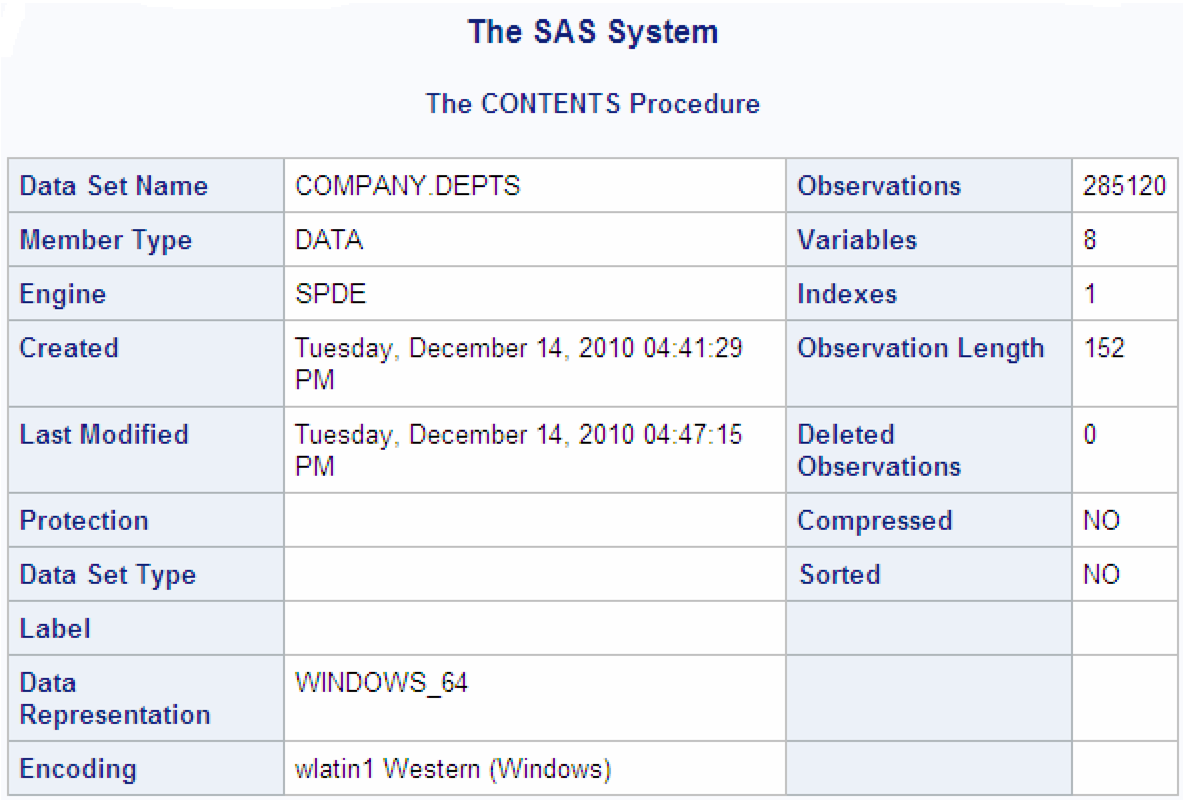
The LISTFILES= data set option is used only with the SPD Engine. The CONTENTS procedure is used to list the complete pathnames of all of the component files of an SPD Engine data set.

## Example: LISTFILES Option

```
proc contents data=company.depts (listfiles=yes);
```

The following CONTENTS procedure output shows the complete pathnames of all of the component files:

***Output 4.7*** *CONTENTS Procedure—Output Section 1*

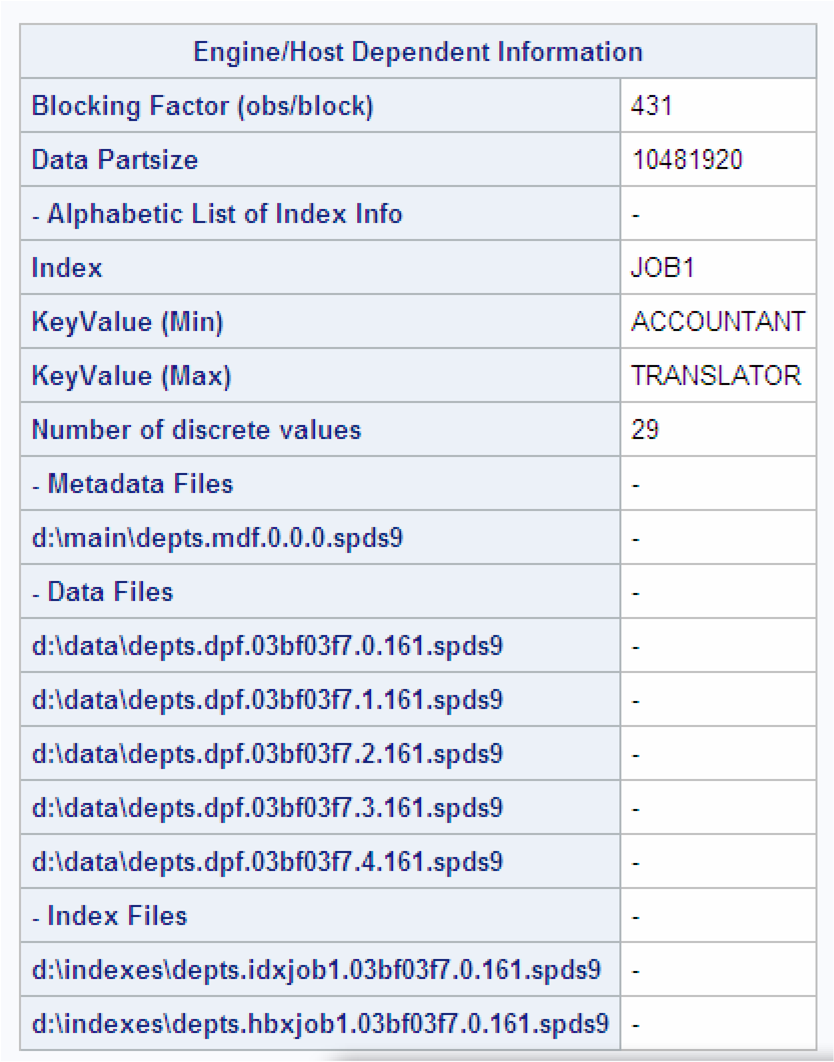


The SAS System

The CONTENTS Procedure

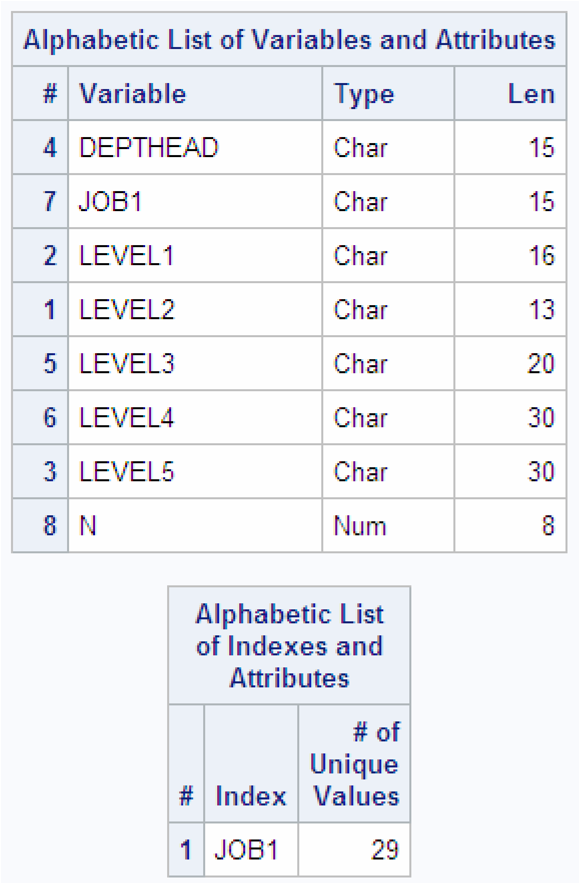
| | | | |
|---|---|---|---|
| Data Set Name | COMPANY.DEPTS | Observations | 285120 |
| Member Type | DATA | Variables | 8 |
| Engine | SPDE | Indexes | 1 |
| Created | Tuesday, December 14, 2010 04:41:29 PM | Observation Length | 152 |
| Last Modified | Tuesday, December 14, 2010 04:47:15 PM | Deleted Observations | 0 |
| Protection | | Compressed | NO |
| Data Set Type | | Sorted | NO |
| Label | | | |
| Data Representation | WINDOWS_64 | | |
| Encoding | wlatin1 Western (Windows) | | |

***Output 4.8*** *CONTENTS Procedure—Output Section 2*



| Engine/Host Dependent Information | |
|---|---|
| Blocking Factor (obs/block) | 431 |
| Data Partsize | 10481920 |
| - Alphabetic List of Index Info | - |
| Index | JOB1 |
| KeyValue (Min) | ACCOUNTANT |
| KeyValue (Max) | TRANSLATOR |
| Number of discrete values | 29 |
| - Metadata Files | - |
| d:\main\depts.mdf.0.0.0.spds9 | - |
| - Data Files | - |
| d:\data\depts.dpf.03bf03f7.0.161.spds9 | - |
| d:\data\depts.dpf.03bf03f7.1.161.spds9 | - |
| d:\data\depts.dpf.03bf03f7.2.161.spds9 | - |
| d:\data\depts.dpf.03bf03f7.3.161.spds9 | - |
| d:\data\depts.dpf.03bf03f7.4.161.spds9 | - |
| - Index Files | - |
| d:\indexes\depts.idxjob1.03bf03f7.0.161.spds9 | - |
| d:\indexes\depts.hbxjob1.03bf03f7.0.161.spds9 | - |

***Output 4.9*** *CONTENTS Procedure—Output Section 3*



| Alphabetic List of Variables and Attributes | | | |
|---|---|---|---|
| # | Variable | Type | Len |
| 4 | DEPTHEAD | Char | 15 |
| 7 | JOB1 | Char | 15 |
| 2 | LEVEL1 | Char | 16 |
| 1 | LEVEL2 | Char | 13 |
| 5 | LEVEL3 | Char | 20 |
| 6 | LEVEL4 | Char | 30 |
| 3 | LEVEL5 | Char | 30 |
| 8 | N | Num | 8 |

| Alphabetic List of Indexes and Attributes | | |
|---|---|---|
| # | Index | # of Unique Values |
| 1 | JOB1 | 29 |

## PADCOMPRESS= Data Set Option

Specifies the number of bytes to add to compressed blocks in a data set opened for OUTPUT or UPDATE.

| | |
|---|---|
| **Valid in:** | DATA step and PROC step |
| **Default:** | 0 |
| **Interactions:** | "COMPRESS= Data Set Option" on page 60<br>"IOBLOCKSIZE= Data Set Option" on page 73 |
| **Engine:** | SPD Engine only |

### Syntax

**PADCOMPRESS=***n*

### *Required Argument*

***n***

is the number of bytes to add.

## Details

Compressed SPD Engine data sets occupy blocks of space on the disk. The size of a block is derived from the IOBLOCKSIZE= data set option specified when the data set is created. When the data set is updated, a new block fragment might need to be created to hold the update. More updates might then create new fragments, which, in turn, increases the number of I/O operations needed to read a data set.

By increasing the block padding in certain situations where many updates to the data set are expected, fragmentation can be kept to a minimum. However, adding padding can waste space if you do not update the data set.

You must weigh the cost of padding all compression blocks against the cost of possible fragmentation of some compression blocks.

Specifying the PADCOMPRESS= data set option when you create or update a data set adds space to all of the blocks as they are written back to the disk. The PADCOMPRESS= setting is not retained in the data set's metadata.

# PARTSIZE= Data Set Option

Specifies the maximum size (in megabytes, gigabytes, or terabytes) that the data component partitions can be. The value is specified when an SPD Engine data set is created. This size is a fixed size. This specification applies only to the data component files.

| | |
|---|---|
| **Valid in:** | DATA step and PROC step |
| **Used by:** | MINPARTSIZE= system option |
| **Default:** | 128 MB |
| **Interaction:** | “DATAPATH= LIBNAME Statement Option” on page 34 |
| **Engine:** | SPD Engine only |

## Syntax

**PARTSIZE=***n* | *n*M | *n*G | *n*T

### *Required Argument*

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

is the size of the partition in megabytes, gigabytes, or terabytes. If *n* is specified without M, G, or T, the default is megabytes. For example, PARTSIZE=128 is the same as PARTSIZE=128M. The maximum value is 8,796,093,022,207 megabytes.

**Restriction** This restriction applies only to 32-bit hosts with the following operating systems: z/OS, Linux SLES 9 x86, and the Windows family. In SAS 9.3, if you create a data set with a partition size greater than or equal to 2 gigabytes, you cannot open the data set with any version of SPD Engine prior to SAS 9.2. The following error message is written to the SAS log: **`ERROR: Unable to open data file because its data representation differs from the SAS session data representation.`**

## Details

Multiple partitions are necessary to read the data in parallel. The option PARTSIZE= forces the software to partition SPD Engine data files at the specified size. The actual size of the partition is computed to accommodate the maximum number of observations that fit in the specified size of *n* megabytes, gigabytes, or terabytes. If you have a table with an observation length greater than 65K, you might find that the PARTSIZE= that you specify and the actual partition size do not match. To get these numbers to match, specify a PARTSIZE= that is a multiple of 32 and the observation length.

By splitting (partitioning) the data portion of an SPD Engine data set into fixed-sized files, the software can introduce a high degree of scalability for some operations. The SPD Engine can spawn threads in parallel (for example, up to one thread per partition for WHERE evaluations). Separate data partitions also enable the SPD Engine to process the data without the overhead of file access contention between the threads. Because each partition is one file, the trade-off for a small partition size is that an increased number of files (for example, UNIX i-nodes) are required to store the observations.

Scalability limitations using PARTSIZE= depend on how you configure and spread the file systems specified in the DATAPATH= option across striped volumes. (You should spread each individual volume's striping configuration across multiple disk controllers or SCSI channels in the disk storage array.) The goal for the configuration, at the hardware level, is to maximize parallelism during data retrieval. For information about disk striping, see “I/O Setup and Validation” under “SPD Engine” in Scalability and Performance at http://support.sas.com/rnd/scalability.

The PARTSIZE= specification is limited by the SPD Engine system option MINPARTSIZE=, which is usually maintained by the system administrator. MINPARTSIZE= ensures that an inexperienced user does not arbitrarily create small partitions, thereby generating a large number of data files.

The partition size determines a unit of work for many of the parallel operations that require full data set scans. But, more partitions does not always mean faster processing. The trade-offs involve balancing the increased number of physical files (partitions) required to store the data set against the amount of work that can be done in parallel by having more partitions. More partitions means more open files to process the data set, but a smaller number of observations in each partition. A general rule is to have 10 or fewer partitions per data path, and 3 to 4 partitions per CPU. (Some operating systems have a limit on the number of open files that you can use.)

To determine an adequate partition size for a new SPD Engine data set, you should be aware of the following:

- the types of applications that run against the data
- how much data you have
- how many CPUs are available to the applications
- which disks are available for storing the partitions
- the relationships of these disks to the CPUs

For example, if each CPU controls only one disk, then an appropriate partition size would be one in which each disk contains approximately the same amount of data. If each CPU controls two disks, then an appropriate partition size would be one in which the load is balanced. Each CPU does approximately the same amount of work.

*Note:* The PARTSIZE= value for a data set cannot be changed after a data set is created. To change PARTSIZE=, you must re-create the data set and specify a different PARTSIZE= value in the LIBNAME statement or on the new (output) data set.

## Comparisons

The PARTSIZE= data set option overrides the PARTSIZE= LIBNAME statement option.

## Example: Using PROC SQL

You have 100 gigabytes of data and 8 disks, so you can store 12.5 gigabytes per disk. Optimally, you want 3 to 4 partitions per disk. A partition size of 3.125 gigabytes is appropriate. So, you can specify PARTSIZE=3200M.

```
data salecent.sw (partsize=3200m);
```

Using the same amount of data, you anticipate the amount of data doubles within a year. You can either specify the same PARTSIZE= and have about 7 partitions per disk, or you can increase PARTSIZE= to 5000M and have 5 partitions per disk.

# STARTOBS= Data Set Option

Specifies the starting observation number in a user-defined range of observations to be processed.

**Valid in:** DATA step and PROC step

**Default:** The first observation in the data set

**Restrictions:** Use STARTOBS= with input data sets only

Cannot be used with the OBS= system or data set option or with the FIRSTOBS= system and data set option

**Interactions:** “STARTOBS= LIBNAME Statement Option” on page 43

“ENDOBS= LIBNAME Statement Option” on page 35

“ENDOBS= Data Set Option” on page 67

**Engine:** SPD Engine only

## Syntax

**STARTOBS=***n*

### *Required Argument*

***n***

is the number of the starting observation.

## Details

### *Specifying a Range of Observations*

By default, the SPD Engine processes all of the observations in the entire data set unless you specify a range of observations with the STARTOBS= and ENDOBS= options. If the ENDOBS= option is used without the STARTOBS= option, the implied value of STARTOBS= is 1. When both options are used together, the value of STARTOBS= must be less than the value of ENDOBS=.

The STARTOBS= data set option in the SPD Engine works the same way as the FIRSTOBS= SAS data set option in the default Base SAS engine. The only difference is when STARTOBS= is specified in a WHERE expression.

*Note:* The FIRSTOBS= SAS data set option is not supported by the SPD Engine. The OBS= system option and the OBS= data set option cannot be used with the STARTOBS= or ENDOBS= data set or LIBNAME options.

### Using STARTOBS= with a WHERE Expression

When STARTOBS= is used in a WHERE expression, the STARTOBS= value represents the first observation on which to apply the WHERE expression. Compare this value to the default Base SAS engine data set option FIRSTOBS=, which specifies the starting observation number within the subset of data qualified by the WHERE expression.

## Comparisons

The STARTOBS= data set option overrides the STARTOBS= LIBNAME statement option.

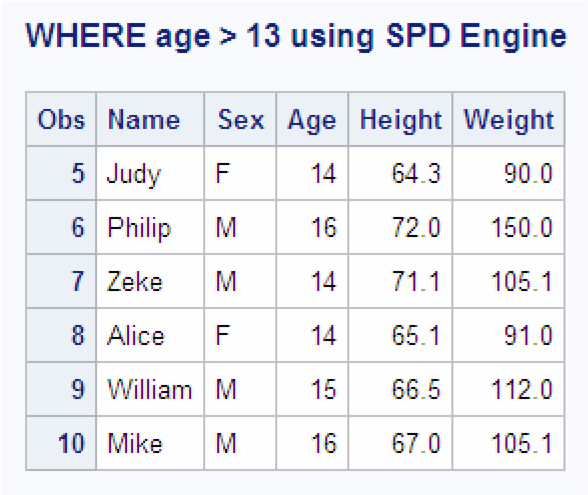
## Examples

### Example 1: STARTOBS= with SPD Engine

A data set is created and processed by the SPD Engine with STARTOBS=5 specified. The WHERE expression is applied to the data set, beginning with observation number 5. The PRINT procedure prints six observations, which are the observations qualified by the WHERE expression.

```
libname growth spde 'SAS-library';
data growth.teens;
   input Name $ Sex $ Age Height Weight;
   list;
datalines;
Alfred M 14 69.0 112.5
Carol F 14 62.8 102.5
James M 13 57.3 83.0
Janet F 15 62.5 112.5
Judy F 14 64.3 90.0
Philip M 16 72.0 150.0
Zeke M 14 71.1 105.1
Alice F 14 65.1 91.0
William M 15 66.5 112.0
Mike M 16 67.0 105.1
;
proc print data=growth.teens (startobs=5);
   where age >13;
   title 'WHERE age>13 using SPD Engine';
run;
```

***Output 4.10*** *STARTOBS=*



WHERE age > 13 using SPD Engine

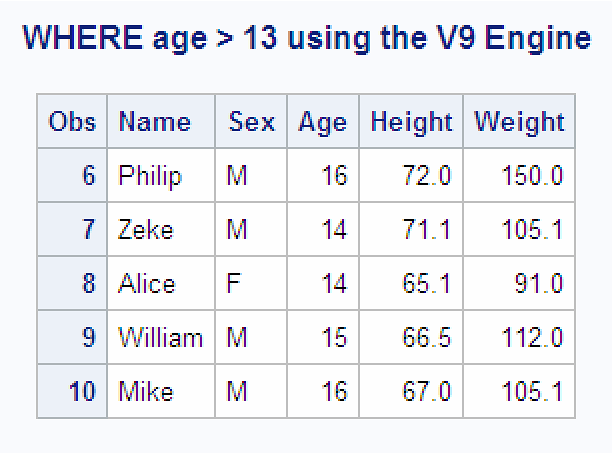
| Obs | Name | Sex | Age | Height | Weight |
|---|---|---|---|---|---|
| 5 | Judy | F | 14 | 64.3 | 90.0 |
| 6 | Philip | M | 16 | 72.0 | 150.0 |
| 7 | Zeke | M | 14 | 71.1 | 105.1 |
| 8 | Alice | F | 14 | 65.1 | 91.0 |
| 9 | William | M | 15 | 66.5 | 112.0 |
| 10 | Mike | M | 16 | 67.0 | 105.1 |

### Example 2: FIRSTOBS= with the Default Base SAS Engine

The same data set is processed by the default Base SAS engine with FIRSTOBS=5 option specified. PROC PRINT prints five observations, which are all of the observations qualified by the WHERE expression, starting with the fifth qualified observation. FIRSTOBS= option is not supported in the SPD Engine.

```
libname growth v9 'SAS-library';
data growth.teens;
   input Name $ Sex $ Age Height Weight;
   list;
datalines;
Alfred M 14 69.0 112.5
Carol F 14 62.8 102.5
James M 13 57.3 83.0
Janet F 15 62.5 112.5
Judy F 14 64.3 90.0
Philip M 16 72.0 150.0
Zeke M 14 71.1 105.1
Alice F 14 65.1 91.0
William M 15 66.5 112.0
Mike M 16 67.0 105.1
;
proc print data=growth.teens (firstobs=5);
   where age >13;
   title 'WHERE age>13 using the V9 Engine';
run;
```

***Output 4.11*** *Five Observations Printed*



WHERE age > 13 using the V9 Engine

| Obs | Name | Sex | Age | Height | Weight |
|---|---|---|---|---|---|
| 6 | Philip | M | 16 | 72.0 | 150.0 |
| 7 | Zeke | M | 14 | 71.1 | 105.1 |
| 8 | Alice | F | 14 | 65.1 | 91.0 |
| 9 | William | M | 15 | 66.5 | 112.0 |
| 10 | Mike | M | 16 | 67.0 | 105.1 |

## SYNCADD= Data Set Option

Specifies to process one observation at a time or multiple observations at a time when adding observations.

| | |
|---|---|
| **Valid in:** | PROC SQL |
| **Default:** | NO |
| **Interaction:** | UNIQUESAVE= |
| **Engine:** | SPD Engine only |

### Syntax

**SYNCADD=**YES | NO

#### *Required Arguments*

**YES**
processes a single observation at a time (synchronously).

**NO**
processes multiple observations at a time (asynchronously).

### Details

With SYNCADD=YES, observations are processed one at a time. With PROC SQL, if you are inserting observations into a data set with a unique index, and the SPD Engine encounters an observation with a non-unique value, the following occurs:

- the add operation stops
- all transactions just added are backed out
- the original data set on disk is unchanged

Adding observations with SYNCADD=NO is obviously much faster. However, when inserting a few observations into a data set with a unique index using PROC SQL, the SPD Engine can back out all the observations if one duplicate value is found. Specifically, the following occurs:

- the SPD Engine rejects the observation
- the SPD Engine continues processing
- a status code is issued only at the end of the Append or Insert operation

To save the rejected observations in a separate data set, set the UNIQUESAVE= data set option to YES.

## Example: Inserting Observations with Duplicate Values into a Data Set with a Unique Index

In the following example, two identical data sets, WITH_NO and WITH_YES, are created. Both have a unique index.

PROC SQL is used to insert three new observations, one of which has duplicate values. The SYNCADD=YES option is used. PROC SQL stops when the duplicate values are encountered and restores the data set.

PROC SQL is used again to insert these three new observations (as before). In this case, the SYNCADD=NO option is used. The observation with duplicate values is rejected. The SAS log is shown:

***Log 4.5*** *Inserting Observations*

```
1    libname addlib spde 'c:\temp';
NOTE: Libref ADDLIB was successfully assigned as follows:
      Engine:        SPDE
      Physical Name: c:\temp\
2
3    data addlib.with_no(index=(x /unique))
4         addlib.with_yes(index=(x /unique)) ;
5      input z $ 1-20 x y;
6      list;
7      datalines;

RULE:      ----+----1----+----2----+----3----+----4----+----5----+----6----
+----7----+----8----+
8          one                  1 10
9          two                  2 20
10         three                3 30
11         four                 4 40
12         five                 5 50
NOTE: The data set ADDLIB.WITH_NO has 5 observations and 3 variables.
NOTE: The data set ADDLIB.WITH_YES has 5 observations and 3 variables.

13   run;
14
15   proc sql;
16      insert into addlib.with_yes(syncadd=yes)
17       values('six_yes',   6, 60 )
18       values('seven_yes', 2, 70 )
19       values('eight_yes', 8, 80 )
20   ;
ERROR: Duplicate values not allowed on index x for file WITH_YES.
NOTE: This insert failed while attempting to add data from VALUES clause 2 to
the data set.
NOTE: Deleting the successful inserts before error noted above to restore table
to a consistent
      state.
21   quit;
NOTE: The SAS System stopped processing this step because of errors.
22


23   proc sql;
24      insert into addlib.with_no(syncadd=no)
25       values('six_no',   6, 60 )
26       values('seven_no', 2, 70 )
27       values('eight_no', 8, 80 )
28   ;
NOTE: 3 rows were inserted into ADDLIB.WITH_NO.

WARNING: Duplicate values not allowed on index x for file WITH_NO, 1
observations rejected.
29   quit;

30
31   proc compare data=addlib.with_no compare=addlib.with_yes;
32   run;

NOTE: There were 7 observations read from the data set ADDLIB.WITH_NO.
NOTE: There were 5 observations read from the data set ADDLIB.WITH_YES.
```

# THREADNUM= Data Set Option

Specifies the maximum number of I/O threads the SPD Engine can spawn for processing an SPD Engine data set.

|  |  |
|---|---|
| **Valid in:** | DATA step and PROC step |
| **Default:** | The value of the SPDEMAXTHREADS= system option, if set; otherwise, the default is two times the number of CPUs on your computer |
| **Interaction:** | "SPDEMAXTHREADS= System Option" on page 101 |
| **Engine:** | SPD Engine only |

## Syntax

**THREADNUM=***n*

### *Required Argument*

***n***

specifies the number of threads.

## Details

THREADNUM= enables you to specify the maximum number of I/O threads that the SPD Engine spawns for processing an SPD Engine data set. The THREADNUM= value applies to any of the following SPD Engine I/O processing:

- WHERE expression processing
- parallel index creation
- I/O requested by thread-enabled applications

Adjusting THREADNUM= enables the system administrator to adjust the level of CPU resources the SPD Engine can use for any process. For example, in a 64-bit processor system, setting THREADNUM=4 limits the process to, at most, four CPUs, thereby enabling greater throughput for other users or applications.

When THREADNUM= is greater than 1, parallel processing is likely to occur. Therefore, physical order might not be retained in the output.

You can also use this option to explore scalability for WHERE expression evaluations.

The SPDEMAXTHREADS= system option imposes an upper limit on the consumption of system resources and constrains the THREADNUM= value.

*Note:* The SAS system option NOTHREADS does not affect the SPD Engine.

*Note:* Setting THREADNUM=1 means that no parallel processing occurs, which is behavior consistent with the default Base SAS engine.

## Example: Using %MACRO

The SPD Engine system option SPDEMAXTHREADS= is set to 128 for the session. A SAS macro shows the effects of parallelism in the following example:

```
%macro dotest(maxthr);
```

```
%do nthr=1 %to &maxthr;
data _null_;
set spde cen.precs(threadnum= &nthr);
   where occup= '022'
   and state in('37','03','06','36');
run;
%mend dotest;
```

## UNIQUESAVE= Data Set Option

Specifies to save observations with non-unique key values (the rejected observations) to a separate data set when appending or inserting observations to data sets with unique indexes.

| | |
|---|---|
| **Valid in:** | PROC APPEND and PROC SQL |
| **Used by:** | SPDSUSDS automatic macro variable |
| **Default:** | NO |
| **Interaction:** | "SYNCADD= Data Set Option" on page 83 |
| **Engine:** | SPD Engine only |

### Syntax

**UNIQUESAVE=**YES | NO

#### *Required Arguments*

**YES**
if SYNCADD=NO, writes rejected observations to a separate, system-created data set, which can be accessed by a reference to the macro variable SPDSUSDS.

**NO**
does not write rejected observations to a separate data set.

### Details

Use UNIQUESAVE=YES when you are adding observations to a data set with unique indexes and the data set option SYNCADD=NO is set.

SYNCADD=NO specifies that an Append or Insert operation should process observations in blocks (pipelining), instead of one at a time. Duplicate index values are detected only after all the observations are applied to a data set. With UNIQUESAVE=YES, the rejected observations are saved to a separate data set whose name is stored in the SPD Engine macro variable SPDSUSDS. You can specify the macro variable in place of the data set name to identify the rejected observations.

*Note:* When SYNCADD=YES, the UNIQUESAVE= option is ignored. For more information see the SYNCADD= data set option.

### Example: Using the UNIQUESAVE= Option with the APPEND Procedure

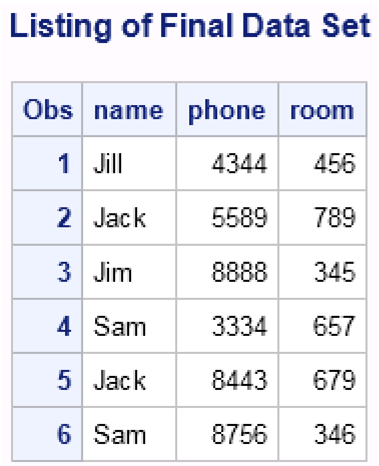
In the following example, a data set with two unique indexes is created and another data set with duplicate values is then appended to the first one. Because the

UNIQUESAVE=YES option is specified, a data set containing the rejected observations is created. That data set includes a variable identifying the variable that had the duplicate value. The SAS log is shown.

***Log 4.6*** *UNIQUESAVE= Option*

```
1    libname employee spde 'c:\temp';
NOTE: Libref EMPLOYEE was successfully assigned as follows:
      Engine:        SPDE
      Physical Name: c:\temp\
2
3    data employee.emp1 (index=(phone/unique room/unique));
4       input name $ phone room;
5       list;
6       datalines;

RULE:      ----+----1----+----2----+----3----+----4----+----5----+----6----
+----7----+----8----+
7          Jill 4344 456
8          Jack 5589 789
9          Jim  8888 345
10         Sam  3334 657
NOTE: The data set EMPLOYEE.EMP1 has 4 observations and 3 variables.

11   run;
12
13   data employee.emp2;
14      input name $ phone room;
15      list;
16      datalines;

RULE:      ----+----1----+----2----+----3----+----4----+----5----+----6----
+----7----+----8----+
17         Jack  8443 679
18         Ann   3334 987
19         Sam   8756 346
20         Susan 5321 456
NOTE: The data set EMPLOYEE.EMP2 has 4 observations and 3 variables.

21   run;
22
23   proc append base=employee.emp1(syncadd=no uniquesave=yes)
NOTE: Writing HTML Body file: sashtml.htm
24               data=employee.emp2;
25   run;

NOTE: Appending EMPLOYEE.EMP2 to EMPLOYEE.EMP1.
NOTE: There were 4 observations read from the data set EMPLOYEE.EMP2.
NOTE: 2 observations added.
NOTE: The data set EMPLOYEE.EMP1 has 6 observations and 3 variables.
WARNING: Duplicate values not allowed on index phone for file EMP1, 1
observations rejected.
WARNING: Duplicate values not allowed on index room for file EMP1, 1
observations rejected.
NOTE: Duplicate records have been stored in file
EMPLOYEE._SPDEDUP048604700067A9F340C7E3E6.

26
27   proc print data=employee.emp1;
28      title 'Listing of Final Data Set';
29   run;

NOTE: There were 6 observations read from the data set EMPLOYEE.EMP1.

30
31   proc print data=&spdsusds
32      title 'Listing of Rejected observations';
33   run;

NOTE: There were 2 observations read from the data set
      EMPLOYEE._SPDEDUP048604700067A9F340C7E3E6.
```

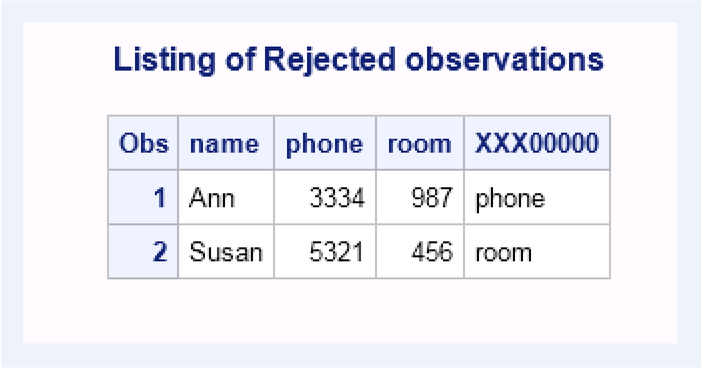
***Output 4.12*** *UNIQUESAVE=YES*



Listing of Final Data Set

| Obs | name | phone | room |
|---|---|---|---|
| 1 | Jill | 4344 | 456 |
| 2 | Jack | 5589 | 789 |
| 3 | Jim | 8888 | 345 |
| 4 | Sam | 3334 | 657 |
| 5 | Jack | 8443 | 679 |
| 6 | Sam | 8756 | 346 |

***Output 4.13*** *Rejected Observations*



Listing of Rejected observations

| Obs | name | phone | room | XXX00000 |
|---|---|---|---|---|
| 1 | Ann | 3334 | 987 | phone |
| 2 | Susan | 5321 | 456 | room |

## WHERENOINDEX= Data Set Option

Specifies a list of indexes to exclude when making WHERE expression evaluations.

**Valid in:** DATA step and PROC step

**Default:** Blank

**Restriction:** Cannot be used with IDXWHERE=NO data set option

**Engine:** SPD Engine only

### Syntax

**WHERENOINDEX=**(*name(s)*)

### *Required Argument*

***(name(s))***
a list of index names to exclude from the WHERE planner.

## Example: Excluding Indexes

The data set PRECS is defined with indexes:

```
proc datasets lib=spde cen
   modify precs;
   index create stser=(state serialno) occind=(occup industry) hour89;
quit;
```

When evaluating the next query, the SPD Engine does not use the indexes for either the STATE or HOUR89 variables.

In this case, the AND combination of the conditions for the OCCUP and INDUSTRY variables produce a very small yield. Few observations satisfy the conditions. To avoid the extra index I/O (computer time) that the query requires for a full-indexed evaluation, use the following SAS code:

```
proc sql;
   create data set hr80spde
   as select state, age, sex, hour89, industry, occup from spde cen.precs
     (wherenoindex=(stser hour89))
   where occup='022'
   and state in('37','03','06','36')
   and industry='012'
   and hour89 > 40;
 quit;
```

*Note:* Specify the index names in the WHERENOINDEX list, not the variable names. In the previous example, both the composite index for the STATE variable, STSER, and the simple index, HOUR89, are excluded.

*Chapter 5*

# SPD Engine System Options

## Introduction to SPD Engine System Options

SAS system options are instructions that affect your SAS session. They control how SAS performs operations, such as SAS system initialization, hardware and software interfacing, and the input, processing, and output of jobs and SAS files. The SPD Engine system options work the same way as SAS system options. This section discusses system options that are used only with the SPD Engine, and Base SAS system options that behave differently with the SPD Engine.

## Syntax

OPTIONS *option-1 <option-2 ...>*;

*option*
specifies one or more SPD Engine system options that you want to change.

The following example specifies the SPD Engine system option MAXSEGRATIO=:

```
options maxsegratio=50;
```

*Operating Environment Information:* On the command line or in a configuration file, the syntax is specific to your operating environment. For details, see the SAS documentation for your operating environment.

## SPD Engine System Options List

COMPRESS=
: specifies to compress SPD Engine data sets on disk as they are being created.

MAXSEGRATIO=
: controls what percentage of index segments to identify as candidate segments before processing the WHERE expression. This occurs when evaluating a WHERE expression that contains indexed variables.

MINPARTSIZE=
: specifies the minimum partition size to use when creating SPD Engine data sets.

SPDFILECACHE | NOSPDEFILECACHE
: enables or disables caching of opened SPD Engine data files.

SPDEINDEXSORTSIZE=
: specifies the memory space size that the sorting utility can use when sorting values for creating an index.

SPDEMAXTHREADS=
: specifies the maximum number of threads that the SPD Engine can spawn for I/O processing.

SPDESORTSIZE=
: specifies the memory space size that is needed for sorting operations used by the SPD Engine.

SPDEUTILLOC=
: specifies one or more file system locations in which the SPD Engine can temporarily store utility files.

SPDEWHEVAL=
: specifies the process to determine which observations meet the condition or conditions of a WHERE expression.

## SAS System Options That Behave Differently with the SPD Engine Than with the Default Base SAS Engine

MSGLEVEL=I
: produces WHERE optimization information in the SAS log.

COMPRESS=
: cannot perform user-defined compression.

DLDMGACTION=
: does not support DLDMGACTION=NOINDEX, but does support ABORT, FAIL, PROMPT, and REPAIR.

DLCREATEDIR
: does not work with the SPD Engine.

ERRORS=MAX
: sets the maximum number of observations to 2147483647 for which SAS can issue error messages.

FIRSTOBS=
: cannot be used in the SPD Engine.

SORTPGM=
: using the BEST option can cause performance issues.

VALIDMEMNAME=
: has the following restrictions on member name when you use VALIDMEMNAME=EXTEND:

- a member name cannot have a period, such as *class.group*.
- a member name cannot start with $, such as*$class*.

VALIDVARNAME=
: cannot create an index or composite index on a variable if the variable name contains any of the following special characters:

```
" * | \ : / < > ? -
```

# Dictionary

## COMPRESS= System Option

Specifies to compress SPD Engine data sets on disk as they are being created.

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

**Category:** System administration: Performance

**Default:** NO

**Restriction:** Cannot be used with ENCRYPT=YES or ENCRYPT=AES

**Interactions:** “IOBLOCKSIZE= Data Set Option” on page 73
“PADCOMPRESS= Data Set Option” on page 77

**Engine:** SPD Engine only

### Syntax

**COMPRESS=**NO | YES | CHAR | BINARY

### *Required Arguments*

**NO**
: performs no data set compression.

**YES | CHAR**
specifies that data in an SPD Engine data set be compressed in blocks by using RLE (run-length encoding). RLE compresses data by reducing repeated runs of the same character (including a blank space) to two-byte or three-byte representations.

**BINARY**
specifies that data in an SPD Engine data set be compressed in blocks by using RDC (Ross Data Compression). RDC combines RLE and sliding window compression to compress the file by representing repeated byte patterns more efficiently.

*Note:* This method is highly effective for compressing medium to large (several hundred bytes or larger) blocks of binary data (character and numeric variables).

## Details

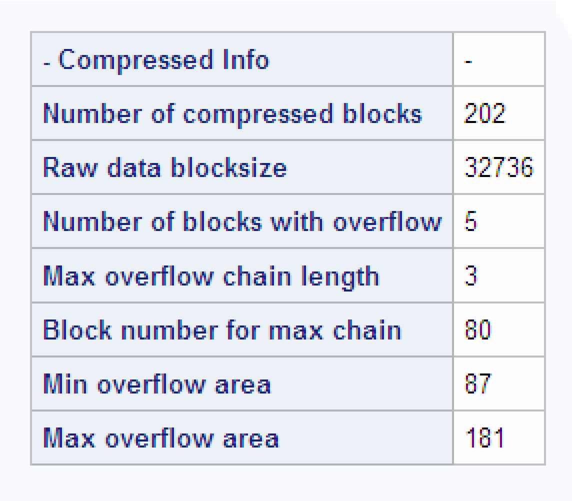
When you specify COMPRESS=YES | BINARY | CHAR, the SPD Engine compresses, by blocks, the data component file as it is created. To specify the size of the compressed blocks, use the "IOBLOCKSIZE= Data Set Option" on page 73 when you create the data set. To add padding to the newly compressed blocks, specify "PADCOMPRESS= Data Set Option" on page 77 when creating or updating the data set. For more information, see "Compressing SPD Engine Data Sets" on page 18.

The SPD Engine does not support user-specified compression. If you are migrating a default Base SAS engine data set that is both compressed and encrypted, the encryption is retained, but the compression is dropped.

The CONTENTS procedure identifies the compress setting. If the data set is compressed, PROC CONTENTS prints information about the compression. The following example explains the Compressed Info fields in the CONTENTS procedure output:

In general, COMPRESS=CHAR provides good compression when single bytes repeat; COMPRESS=BINARY provides good compression when strings of bytes repeat. At the same time, it is more costly to look for strings of bytes that repeat, than to look for single bytes that repeat. For examples, see "Example 1: COMPRESS=CHAR" on page 62 and "Example 2: COMPRESS=BINARY" on page 62.

***Output 5.1*** *PROC CONTENTS Compressed Section*



| | |
|---|---|
| - Compressed Info | - |
| Number of compressed blocks | 202 |
| Raw data blocksize | 32736 |
| Number of blocks with overflow | 5 |
| Max overflow chain length | 3 |
| Block number for max chain | 80 |
| Min overflow area | 87 |
| Max overflow area | 181 |

Number of compressed blocks
number of compressed blocks that are required to store data.

Raw data blocksize
: compressed block size in bytes calculated from the size specified in the IOBLOCKSIZE= data set option. It is the largest multiple of the observation length that gets in the block size.

Number of blocks with overflow
: number of compressed blocks that needed more space. When data is updated and the compressed new block is larger than the compressed old block, an overflow block fragment is created.

Max overflow chain length
: largest number of overflows for a single block. For example, the maximum overflow chain length would be 2 if a compressed block was updated and became larger, and then updated again to a larger size.

Block number for max chain
: number of the block containing the largest number of overflow blocks.

Min overflow area
: minimum amount of disk space that an overflow requires.

Max overflow area
: maximum amount of disk space that an overflow requires.

Accessing compressed files usually requires more processing time. The files have to be decompressed before reading them and, if updating, they have to be compressed again when written to disk.

### Comparisons

The COMPRESS= system option is overridden by the COMPRESS= LIBNAME statement option and the COMPRESS= data set option.

## MAXSEGRATIO= System Option

Controls what percentage of index segments to identify as candidate segments before processing the WHERE expression. This occurs when evaluating a WHERE expression that contains indexed variables.

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

**Category:** System administration: Performance

**Default:** 75

**Engine:** SPD Engine only

### Syntax

**MAXSEGRATIO=***n*

#### *Required Argument*

***n***

specifies an upper limit for the percentage of index segments that the SPD Engine identifies as containing the value referenced in the WHERE expression. The default is 75, which specifies for the SPD Engine to do the following:

- use the index to identify segments that contain the particular WHERE expression value

- stop identifying candidate segments when more than 75% of all segments are found to contain the value

The range of valid values is integers between 0 and 100. If n=0, the SPD Engine does not try to identify candidate segments, but instead applies the WHERE expression to all segments. If n=100, the SPD Engine checks 100% of the segments to identify candidate segments, and then applies the WHERE expression only to those candidate segments.

## Details

For WHERE queries on indexed variables, the SPD Engine determines the number of index segments that contain one or more variable values that match one or more of the conditions in the WHERE expression. Often, a substantial performance gain can be realized if the WHERE expression is applied only to the segments that contain observations satisfying the WHERE expression.

The SPD Engine uses the value of MAXSEGRATIO= to determine at what point the cost of applying the WHERE expression to every segment would be less than the cost of continuing to identify candidate segments. When the calculated ratio exceeds the ratio specified in MAXSEGRATIO=, the SPD Engine stops identifying candidate segments and applies the WHERE expression to all segments.

*Note:* For a few tables, 75% might not be the optimal setting. To determine a better setting, run a performance benchmark, adjust the percentage, and rerun the performance benchmark. Comparing results shows you how the specific data population that you are querying responds to shifting the index-segment ratio.

## Examples

### *Example 1: Identifying Index Segments*

The following example causes the SPD Engine to begin identifying index segments that might satisfy the WHERE expression until the percentage of identified segments, compared to the total number of segments, exceeds 65. If the percentage exceeds 65, the SPD Engine stops identifying candidate segments and applies the WHERE expression to all segments:

```
 options maxsegratio=65;
```

### *Example 2: Applying the WHERE Expression to All Segments*

The following example causes the SPD Engine to apply the WHERE expression to all segments without first identifying any candidate segments:

```
options maxsegratio=0;
```

### *Example 3: Not Stopping Until All Index Segments Are Evaluated*

The following example causes the SPD Engine to begin identifying index segments and to not stop until it has evaluated all segments. Then, the WHERE expression is applied to all candidate segments that were identified:

```
options maxsegratio=100;
```

## MINPARTSIZE= System Option

Specifies the minimum partition size to use when creating SPD Engine data sets.

|  |  |
|---|---|
| **Valid in:** | Configuration file, SAS invocation |
| **Category:** | System administration: Performance |
| **Default:** | 16M |
| **Engine:** | SPD Engine only |

### Syntax

**MINPARTSIZE=***n* | *n*K | *n*M | *n*G

#### *Required Argument*

***n***

is the size of the partition in bytes, kilobytes, megabytes, or gigabytes. The maximum value for the minimum partition size is 2GB–1 or 2047 megabytes.

**Restriction** This restriction applies only to 32-bit hosts with the following operating systems: z/OS, Linux SLES 9 x86, and the Windows family. If you create a data set with a partition size greater than or equal to 2 gigabytes, you cannot open the data set with any version of SPD Engine prior to SAS 9.2. The following error message is written to the SAS log: **`ERROR: Unable to open data file because its data representation differs from the SAS session data representation.`**

### Details

Specifying MINPARTSIZE= sets a lower limit for the partition size that can be specified with the PARTSIZE= option. The MINPARTSIZE= specification could affect whether the partitions are created with approximately the same number of observations. A small partition size means more open files during processing. Your operating system might have a limit on the number of open files used.

## SPDEFILECACHE System Option

Enables or disables caching of opened SPD Engine data files.

|  |  |
|---|---|
| **Valid in:** | Configuration file, SAS invocation |
| **Category:** | System administration: Performance |
| **Default:** | NOSPDEFILECACHE |
| **Engine:** | SPD Engine only |

### Syntax

**SPDEFILECACHE | NOSPDEFILECACHE**

### Actions

**SPDEFILECACHE**
Enables caching of opened SPD Engine data files.

**NOSPDEFILECACHE**
Disables caching of opened SPD Engine data files.

## Details

Use the SPDEFILECACHE system option to enable or disable file caching of opened SPD Engine data files. Caching SPD Engine data files can improve performance by reducing the need to open and close files. If SPD Engine data files are used in a shared environment, caching files can result in performance degradation. This degradation happens when a user must wait on a file that another user has opened but is no longer accessing.

# SPDEINDEXSORTSIZE= System Option

Specifies the memory space size that the sorting utility can use when sorting values for creating an index.

| | |
|---|---|
| **Valid in:** | Configuration file, SAS invocation, OPTIONS statement, Systems Options window |
| **Category:** | System administration: Performance |
| **Default:** | 32M |
| **Engine:** | SPD Engine only |

## Syntax

**SPDEINDEXSORTSIZE=***n* | *n*K | *n*M | *n*G

### Required Argument

***n***
is the size of memory space in bytes, kilobytes, megabytes, or gigabytes. If *n*=0, the sort utility uses its default. The valid value range is from 1,048,576 to 10,736,369,664 bytes.

## Details

The SPD Engine can create multiple indexes with a single scan of a data set. The SPD Engine spawns a single thread for each index created, and then processes the threads simultaneously. Although creating indexes in parallel is much faster than scanning the data set for each index, the default for this option is NO because parallel index creation requires extra utility space to store the sorting files and requires additional memory. If index creation fails due to insufficient resources, you can do one or both the following:

- Increase the size of the utility file space using the SPDEUTILLOC= system option.
- Set the SAS system option to MEMSIZE=0[1]and increase the utility space that is used for index sorting using the SPDEINDEXSORTSIZE= system option.

The maximum SPDEINDEXSORTSIZE= value is 10 GB, but this value cannot be honored on host systems that are limited to 2 GB. On host systems that have a 64-bit

[1] For z/OS, increase the REGION size.

LONG data type, SPD Engine honors values greater that 2 GB. On hosts systems that have a 32-bit LONG data type, SPD Engine honors only the memory used up to 2 GB. The SPDEINDEXSORTSIZE option value can be set to a larger value, but the larger value is not honored.

*Note:* You receive a warning in the SAS log when the larger value is not honored and used.

## SPDEMAXTHREADS= System Option

Specifies the maximum number of threads that the SPD Engine can spawn for I/O processing.

| | |
|---|---|
| **Valid in:** | Configuration file, SAS invocation |
| **Category:** | System administration: Performance |
| **Default:** | 0 |
| **Engine:** | SPD Engine only |

### Syntax

**SPDEMAXTHREADS=***n*

#### *Required Argument*

***n***

is the maximum number of threads the SPD Engine can spawn. The range of valid values is 0 to 65,536. The default is zero, which means that the SPD Engine uses the value of THREADNUM= if set. Otherwise, the SPD Engine sets the number of threads to spawn to be equivalent to the number of CPUs.

### Details

Specifying SPDEMAXTHREADS= sets an upper limit on the number of threads to spawn for the SPD Engine processing, which includes the following:

- WHERE expression processing
- parallel index creation
- any I/O processing requested by thread-enabled applications such as SAS thread-enabled procedures

SPDEMAXTHREADS= constrains the THREADNUM= data set option.

## SPDESORTSIZE= System Option

Specifies the memory space size that is needed for sorting operations used by the SPD Engine.

| | |
|---|---|
| **Valid in:** | Configuration file, SAS invocation, OPTIONS statement, System Options window |
| **Category:** | System administration: Performance |
| **Default:** | 32M |
| **Engine:** | SPD Engine only |

## Syntax

**SPDESORTSIZE=***n* | *n*K | *n*M | *n*G

### *Required Argument*

***n***

is the size of memory space in bytes, kilobytes, megabytes, or gigabytes. If *n*=0, the sort utility uses its default. The valid value range is from 1,048,576 to 10,736,369,664 bytes.

## Details

The SPD Engine can perform an automatic sort in parallel. The sort size that you specify for SPDESORTSIZE= should be multiplied by the number of processes that are in parallel. This total for sort size should be less than the physical memory available to your process. The proper specification of SPDESORTSIZE= can improve performance by restricting the swapping of memory that is controlled by the operating environment.

Perform one of the following if the sort process needs more memory than you specified:

- restart SAS with the SAS system option MEMSIZE=0 (For z/OS, increase the REGION size.)
- increase the size of the utility file space using the SPDEUTILLOC= system option

You increase the memory that is used when sorting values for creating an index using the SPDEINDEXSORTSIZE= option. If you specify to create indexes in parallel, specify a large-enough space using the SPDEUTILLOC= system option.

*Note:* The SORTSIZE= option documented for the default Base SAS engine affects PROC SORT operations. The SPDESORTSIZE= specification affects sorting operations specific to the SPD Engine.

The maximum SPDESORTSIZE= value is 10 GB, but this value cannot be honored on host systems that are limited to 2 GB. On host systems that have a 64-bit LONG data type, SPD Engine honors values greater that 2 GB. On host systems that have a 32-bit LONG data type, SPD Engine honors only the memory used up to 2 GB. The SPDESORTSIZE option value can be set to a larger value, but the larger value is not honored.

*Note:* You receive a warning in the SAS log when the larger value is not honored and used.

# SPDEUTILLOC= System Option

Specifies one or more file system locations in which the SPD Engine can temporarily store utility files.

| | |
|---|---|
| **Valid in:** | Configuration file, SAS invocation |
| **Category:** | System administration: Performance |
| **Engine:** | SPD Engine only |
| **See:** | The SAS Companion for your operating system details how to specify system options |

## Syntax

**-SPDEUTILLOC** *directory | ("directory-1" "directory-2"…)*

### Required Arguments

***location***

is an existing directory where the utility files are created.

***("directory-1" "directory-2" ...)***

is a series of existing directories where the utility files are created. You can use single or double quotation marks.

Note *location* can be enclosed in single or double quotation marks. Quotation marks are required if *location* contains embedded blanks.

## Details

*Operating Environment Information*

The SAS Companion for your operating system details how to specify system options.

The SPD Engine creates temporary utility files during certain processing, such as automatic sorting and creating indexes. To successfully complete the process, you must have enough space to store the utility files. The SPDEUTILLOC= system option enables you to specify an adequate amount of space for processing. However, for OpenVMS on HP Integrity Servers, the libraries must be ODS-5 files. When multiple directories are specified in the SPDEUTILLOC= system option, the directory for the first utility file is randomly selected when processing starts. The selection continues in a cyclical fashion to the other directories. Utility files are temporary and are removed after processing is completed.

*Note:* To avoid syntax errors, specify multiple directories in the configuration file.

SAS recommends that you always specify the SPDEUTILLOC= option or the UTILLOC= option to ensure that you have enough space for processes that create utility files. The location for temporary utility file storage is defined by each operating environment if the following scenarios occur:

- the SPDEUTILLOC= system option or the UTILLOC= SAS system option is not specified
- the SPD Engine cannot locate the SAS Work directory (or does not have Write permission to it)

The following table shows the default utility file locations:

***Table 5.1*** *Default Utility File Locations*

| Operating Environment | Default Location 1 | Default Location 2 | Default Location 3 |
|---|---|---|---|
| | | | |
| UNIX | UTILLOC= SAS system option, if specified | SAS Work library | /tmp |

| Operating Environment | Default Location 1 | Default Location 2 | Default Location 3 |
|---|---|---|---|
| | | | |
| Windows | UTILLOC= SAS system option, if specified | SAS Work library | location specified by the TEMP= environment variable |
| z/OS | UTILLOC= SAS system option, if specified | SAS Work library | /tmp |

## SPDEWHEVAL= System Option

Specifies the process to determine which observations meet the condition or conditions of a WHERE expression.

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

**Category:** System administration: Performance

**Default:** COST

**Engine:** SPD Engine only

### Syntax

**SPDEWHEVAL=**COST | EVAL1 | EVAL3EVAL4

#### *Required Arguments*

**COST**
specifies that the SPD Engine decides which evaluation strategy to use to optimize the WHERE expression. This process calculates the number of threads to be used, which minimizes the overhead of spawning underused threads. This is the default.

**EVAL1**
is a multi-threaded index evaluation strategy that can quickly determine the rows that satisfy the WHERE expression using multiple threads. The number of threads that are spawned to retrieve the observations is equal to the THREADNUM= value.

**EVAL3EVAL4**
is a single-threaded index evaluation strategy that is used for a simple or compound WHERE expression. All the key variables have a simple index and no condition tests for non-equality. Multi-threading might be used to retrieve the observations.

### Details

The SPD Engine uses WHINIT, a rules-based WHERE expression planner, to select the most appropriate evaluation strategy for a query. The SAS system option MSGLEVEL=I surfaces WHINIT messages to the SAS log that can help you determine whether one or more indexes are used in a query.

COST, the default setting for SPDEWHEVAL=, analyzes the WHERE expression and any available indexes. Based on the analysis, the SPD Engine chooses an evaluation strategy to optimize the WHERE expression. The evaluation strategy can be EVAL1, EVAL3, EVAL4, or a strategy that sequentially reads the data if no indexes are available. Or, if the analysis shows that using the index or indexes cannot improve processing time.

COST optimizes the number of threads to use for processing the WHERE expression. COST determines and spawns the number of threads that can be efficiently used. Based on the value of THREADNUM=, COST can save significant processing time by not spawning threads that are underused.

COST is the recommended value for SPDEWHEVAL=, unless the WHERE expression exactly meets one of the other evaluation strategy criterion. It is strongly recommended that benchmark tests be used to determine whether a value other than COST is more efficient.

EVAL1 might be more efficient if the WHERE expression is complex and there are multiple indexes for the variables. EVAL1 spawns multiple threads to determine which segments meet the conditions of the WHERE expression. Multiple threads can also be used to retrieve the observations.

*Note:* In a few situations, COST might not perform the best. To determine whether changing the value to EVAL1 or EVAL3EVAL4 can produce better performance, run a performance benchmark, change the value, and re-run the performance benchmark. Comparing results shows you how the specific data population that you are querying responds to rules-based WHERE planning.

# Recommended Reading

Here is the recommended reading list for this title:

- *Base SAS Procedures Guide*
- *SAS Data Set Options: Reference*
- *SAS Language Reference: Concepts*
- *SAS System Options: Reference*
- *The Little SAS Book: A Primer*

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

# Glossary

**block**

a group of observations in a data set. By using blocks, thread-enabled applications can read, write, and process the observations faster than if they are delivered as individual observations.

**compound WHERE expression**

a WHERE expression that contains more than one operator, as in WHERE X=1 and Y>3. *See also* WHERE expression.

**controller**

a computer component that manages the interaction between the computer and a peripheral device such as a disk or a RAID. For example, a controller manages data I/O between a CPU and a disk drive. A computer can contain many controllers. A single CPU can command more than one controller, and a single controller can command multiple disks.

**CPU-bound application**

an application whose performance is constrained by the speed at which computations can be performed on the data. Multiple CPUs and threading technology can alleviate this problem.

**data partition**

a physical file that contains data and which is part of a collection of physical files that comprise the data component of a SAS Scalable Performance Data Engine data set. *See also* partition.

**I/O-bound application**

an application whose performance is constrained by the speed at which data can be delivered for processing. Multiple CPUs, partitioned I/O, threading technology, RAID (redundant array of independent disks) technology, or a combination of these can alleviate this problem.

**light-weight process thread**

a single-threaded subprocess that is created and controlled independently, usually with operating system calls. Multiple light-weight process threads can be active at one time on symmetric multiprocessing (SMP) hardware or in thread-enabled operating systems.

### parallel I/O

a method of input and output that takes advantage of multiple CPUs and multiple controllers, with multiple disks per controller to read or write data in independent threads.

### parallel processing

a method of processing that divides a large job into multiple smaller jobs that can be executed simultaneously on multiple CPUs. *See also* threading.

### partition

part or all of a logical file that spans devices or directories. In the SPD Engine, a partition is one physical file. Data files, index files, and metadata files can all be partitioned, resulting in data partitions, index partitions, and metadata partitions, respectively. Partitioning a file can improve performance for very large data sets. *See also* data partition.

### primary path

the location in which SPD Engine metadata files are stored. The other SPD Engine component files (data files and index files) are stored in separate storage paths in order to take advantage of the performance boost of multiple CPUs.

### RAID

*See* redundant array of independent disks.

### redundancy

a characteristic of computing systems in which multiple interchangeable components are provided in order to minimize the effects of failures, errors, or both. For example, if data is stored redundantly (in a RAID, for example), then if one disk is lost, the data is still available on another disk. *See also* redundant array of independent disks.

### redundant array of independent disks (RAID)

a type of interleaved storage system that comprises multiple disks to store large amounts of data inexpensively. RAIDs can have several levels. For example, a level-0 RAID combines two or more hard drives into one logical disk drive. Various RAID levels provide different amounts of redundancy and storage capability. Also, because the same data is stored in different places, I/O operations can overlap, which can result in improved performance. *See also* redundancy.

### SAS Scalable Performance Data Engine (SPD Engine)

a SAS engine that organizes data into a streamlined file format, enabling rapid delivery of data to applications. *See also* parallel I/O, parallel processing.

### sasroot

a representation of the name for the directory or folder in which SAS is installed at a site or a computer.

### scalability

the ability of a software application to function well and with minimal loss of performance, despite changing computing environments, and despite changes in the volume of computations, users, or data. Scalable software is able to take full advantage of increases in computing capability such as those that are provided by the use of SMP hardware and threaded processing. *See also* scalable software.

### scalable software

software that responds to increased computing capability on SMP hardware in the expected way. For example, if the number of CPUs is increased, the time to solution for a CPU-bound problem decreases by a proportionate amount. And if the throughput of the I/O system is increased, the time to solution for an I/O-bound problem decreases by a proportionate amount.

**server scalability**

the ability of a server to take advantage of SMP hardware and threaded processing in order to process multiple client requests simultaneously. That is, the increase in computing capacity that SMP hardware provides increases proportionately the number of transactions that can be processed per unit of time. *See also* symmetric multiprocessing.

**SMP**

*See* symmetric multiprocessing.

**sort indicator**

an attribute of a data file that indicates whether a data set is sorted, how it was sorted, and whether the sort was validated. Specifically, the sort indicator attribute indicates the following information: 1) the BY variable(s) that were used in the sort; 2) the character set that was used for the character variables; 3) the collating sequence of character variables that was used; 4) whether the sort information has been validated. This attribute is stored in the data file descriptor information. Any SAS procedure that requires data to be sorted as a part of its process uses the sort indicator.

**spawn**

to start a process or a process thread such as a light-weight process thread (LWPT). *See also* thread.

**SPD Engine**

*See* SAS Scalable Performance Data Engine.

**SPD Engine data file**

the data component of an SPD Engine data set. In contrast to SAS data files, SPD Engine data files contain only data; they do not contain metadata. The SPD Engine does not support data views. *See also* SPD Engine data set.

**SPD Engine data set**

a data set created by the SPD Engine that has up to four component files: one for data, one for metadata, and two for any indexes. The minimum number of component files is two: data and metadata. Data is separated from the metadata for SPD Engine file organization.

**symmetric multiprocessing (SMP)**

a type of hardware and software architecture that can improve the speed of I/O and processing. An SMP machine has multiple CPUs and a thread-enabled operating system. An SMP machine is usually configured with multiple controllers and with multiple disk drives per controller.

**thread**

the smallest unit of processing that can be scheduled by an operating system.

**thread-enabled operating system**

an operating system that can coordinate symmetric access by multiple CPUs to a shared main memory space. This coordinated access enables threads from the same process to share data very efficiently.

**thread-enabled procedure**

a SAS procedure that supports threaded I/O or threaded processing.

**threaded I/O**

I/O that is performed by multiple threads in order to increase its speed. In order for threaded I/O to improve performance significantly, the application that is performing the I/O must be capable of processing the data rapidly as well. *See also* I/O-bound application, thread.

**threaded processing**

processing that is performed in multiple threads in order to improve the speed of CPU-bound applications. *See also* CPU-bound application, symmetric multiprocessing.

**threading**

a high-performance technology for either data processing or data I/O in which a task is divided into threads that are executed concurrently on multiple cores on one or more CPUs.

**time to solution**

the elapsed time that is required for completing a task. Time-to- solution measurements are used to compare the performance of software applications in different computing environments. In other words, they can be used to measure scalability. *See also* scalability.

**WHERE expression**

is a syntax string within a WHERE clause that defines the criteria for selecting observations. For example, in a membership database, the expression "WHERE member_type=Senior" returns all senior members. *See also* compound WHERE expression.

# Index