SAS® 9.2 Scalable Performance Data Engine

Reference

SAS® Documentation

The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2009. *SAS® 9.2 Scalable Performance Data Engine: Reference*. Cary, NC: SAS Institute Inc.

SAS® 9.2 Scalable Performance Data Engine: Reference

ISBN 978-1-59994-707-5

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

1st electronic book, February 2009
2nd electronic book May 2010

1st printing, February 2009

Contents

What’s New

Overview

The SAS 9.2 Scalable Performance Data (SPD) Engine has the following enhancements:

- □ new and enhanced data set options
- □ new and enhanced LIBNAME statement options
- □ new and enhanced system options

These enhancements enable the SPD Engine to perform rapid processing of very large data sets that are stored in partitions across multiple disk volumes.

SPD Engine Data Set Options

COMPRESS=
The COMPRESS= data set option now supports binary compression with the CHAR and BINARY options.

ENCRYPT=
The new ENCRYPT= data set option specifies whether to encrypt an output SPD Engine data set.

IDXBY=
The new IDXBY= data set option specifies whether to use indexes when processing BY statements in the SPD Engine. In some cases, this option can enhance performance.

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

PARTSIZE=
The PARTSIZE= data set option now expresses the partition size value in gigabytes, terabytes, or the default megabytes. The maximum number of megabytes has increased to 8,796,093,022,207.

SPD Engine LIBNAME Statement Options

ACCESS=READONLY
The ACCESS=READONLY LIBNAME statement option determines the access level of the data source. Using this option prevents writing to the data source.

IDXBY=
The new IDXBY= LIBNAME statement option specifies whether to use indexes when processing BY statements in the SPD Engine. In some cases, this option can enhance performance.

PARTSIZE=
The PARTSIZE= LIBNAME statement option now expresses the partition size value in gigabytes, terabytes, or the default megabytes. The maximum number of megabytes has increased to 8,796,093,022,207.

SPD Engine System Options

COMPRESS=
The COMPRESS= system option now supports binary compression with the CHAR and BINARY options.

MINPARTSIZE=
The MINPARTSIZE= system option now has a default of 16 megabytes, instead of 0 megabytes.

SPD Engine Compatability

A new section has been added with information for upgrading from SAS 9.1.3 to SAS 9.2. See “SPD Engine Compatability”.

PART 1

Usage

CHAPTER

1

Overview: The SPD Engine

Introduction to the SPD Engine

The SPD Engine is designed for high-performance data delivery. It enables rapid access to SAS data for processing by the application. 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/output functions.

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

Although it is not intended to replace the default Base SAS engine, the SPD Engine is a high-speed alternative for processing very large data sets. It reads and writes data sets that contain millions of observations. For example, this includes data sets that expand beyond the 2-gigabyte size limit imposed by some operating systems and data sets that SAS analytic software and procedures must process faster.

The SPD Engine performance is boosted in the following ways:

□ support for gigabytes of data

□ scalability on symmetric multiprocessor (SMP) computers

□ parallel WHERE selections

□ parallel loads

□ parallel index creation

□ parallel I/O data delivery to applications

□ automatic sorting on BY statements

The SPD Engine runs on UNIX, Windows, z/OS (zFS file system only), and OpenVMS on HP Integrity Servers (ODS-5 file systems only).

Note: Be sure to visit the Scalability and Performance focus area at **http://support.sas.com/rnd/scalability** for more information about scalability in SAS 9.2. For system requirements, visit the Install Center at **http://support.sas.com/documentation/installcenter**. △

SPD Engine Compatibility

If you upgrade from SAS 9.1.3 to SAS 9.2, you do not need to migrate your data sets if you remain 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*.

Visit the Migration focus area at **http://support.sas.com/rnd/migration** for migration information.

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

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, two index files are created for each index. Each of these four types of files is called an SPD Engine *component file* and each has an identifying file extension.

In addition, each of these components can consist of one or more physical files so that the component can span volumes but can be referenced as one logical file. For example, the SPD Engine can create many physical files containing data, but reference 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. However, you have little or no control over the size of the metadata or index partitions.
2. The data component files are created in a cyclical fashion across all defined paths. The metadata and index components are created in a single path until that path is full, and then the next path is used.

Metadata Component Files

The 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.

Index Component Files

If the file is indexed, the SPD Engine creates two index component files for each index. Each of these files contains a particular view of the index, so both exist for each data set.

□ 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 by you 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 on 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 collection of files that reside in one or more directories. However, 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 on 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 and the ODS-5 file system for OpenVMS on HP Integrity Servers. This means that some functionality might be slightly different on these platforms. For example, for z/OS, the user must have a home directory on zFS.

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 81.

Temporary Storage of Interim 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 '/mydata' 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[1]	yes
Catalogs	no	yes
Views	no	yes
MDDBs	no	yes
Integrity constraints	no	yes
Data set generations	no	yes
CEDA	no	yes
Audit trail	no	yes
NLS transcoding	no	yes
Number of observations that can be counted	2^{63}-1 (on all hosts)	2^{31}-1 (on 32-bit hosts) 2^{63}-1 (on 64-bit hosts)

Feature	SPD Engine	Default Base SAS Engine
COMPRESS=	YES | NO | CHAR | BINARY (only if the file is not encrypted)	YES | NO | CHAR | BINARY
ENCRYPT=	cannot be used with COMPRESS=	can be used with COMPRESS=
Encryption	data files only	yes (all files)
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
Functions and call routines	yes, with some exceptions[2]	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

[1] In WHERE processing, user-defined formats and informats are passed to the supervisor for handling. Therefore, they are not processed in parallel.

[2] In WHERE processing, functions and call routines introduced in SAS 9 or later 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 on 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, then only that user can access it.

Features That Enhance I/O Performance

Overview of I/O Performance Enhancements

The SPD Engine has several new 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 optimal process 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 which can choose the best method to use to evaluate WHERE expressions that use indexes to optimize evaluation.

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. 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.

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.

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:

□ Chapter 3, “SPD Engine LIBNAME Statement Options,” on page 25

□ Chapter 4, “SPD Engine Data Set Options,” on page 41

□ Chapter 5, “SPD Engine System Options,” on page 75

CHAPTER 2

Creating and Loading SPD Engine Files

Introduction for Creating and Loading SPD Engine Files

This section provides details on 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 I/O 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 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

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

All metadata component files must begin in the primary path, even if they span devices. In addition, specifying separate paths for the data and index components provides further performance gains. You specify different paths because the I/O load is distributed across disk drives. Separating the data and index components 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. The code uses the "DATAPATH= LIBNAME Statement Option" on page 29 and "INDEXPATH= LIBNAME Statement Option" on page 32 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 is stored on disk1, which is the primary path. The data is on disk2 and disk3, and the indexes 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 option other than the primary path. △

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.

You cannot change the pathnames of the files. When you specify the DATAPATH=, INDEXPATH=, METAPATH=, or primary path LIBNAME options, make sure that the identical paths that were used when the data set was created are used every time you access the data sets. The names of these locations are stored internally in the data set. If you change any part of the pathname, the SPD Engine might not be able to find the data set or might damage the data set. △

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 the “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, so the primary path might be large enough to contain all the metadata files for the library.

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 indexes are single or composite indexes. When an index component file grows beyond the space available in the current file path, a new component file is created in the next path.

Data component files can be quite 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 partitions is specified in the “PARTSIZE= LIBNAME Statement Option” on page 34. Data files are the only component files for which you can specify a partition size.

Storage of the Metadata Component Files

Much of the information that the SPD Engine needs to efficiently read and write partitioned data is stored in the metadata component. The SPD Engine must be able to rapidly access that metadata. By design, the SPD Engine expects every data set’s metadata component to begin in the primary path. These metadata component files can overflow into other paths (specified in the “METAPATH= LIBNAME Statement Option” on page 33), but they must always begin in the primary path. This concept is very important to understand because it directly affects whether you can add data sets (with their associated metadata files) to the library.

For example, a new data set for a library is created and the space in the primary path is full. The SPD Engine cannot begin the metadata component file in that primary path as required. 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. You cannot use the METAPATH= option to create space for a new data set's first metadata partition. METAPATH= only specifies overflow space for a metadata component that begins in the primary path, but has expanded to fill the space reserved in the primary path. Your metadata component might grow to exceed the file size or library space limitations. To ensure you have space in the primary path for additional data sets, specify an overflow path for metadata in the METAPATH= option when you first create the library.

You can specify additional space at a later time for data and index component files, even if you specified separate paths in the initial LIBNAME statement.

Example: Initial Set of Paths

In the following example, the LIBNAME statement specifies the MYLIB directory for the primary path. By default, this path is used to store initial metadata partitions. Other devices and directories are specified to store the data and index partitions.

```
libname myref spde 'mylib'
   datapath=('/mydisk30/siteuser')
   indexpath=('/mydisk31/siteuser');
```

Example: Adding Subsequent Paths

Later, if more space is needed (for example, for appending large amounts of data), additional devices are added for the data and indexes, as in the following example:

```
libname myref spde 'mylib'
   datapath=('/mydisk30/siteuser' '/mydisk32/siteuser' '/mydisk33/siteuser')
   indexpath=('/mydisk31/siteuser' '/mydisk34/siteuser');
```

Storage of the Index Component Files

Index component files are stored based on overflow space. Several file paths are specified with the INDEXPATH= option. Index files are started in the first available space, and then overflow to the next file path when the previous space is filled. Unlike metadata components, index component files do not have to begin in the primary path.

Storage of the Data Partitions

The data component partitions are the only files for which you can specify the size. Partitioned data can be processed in threads easily, thereby taking full advantage of multiple CPUs on your computer.

The partition size for the data component is fixed and it is set at the time 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 size 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 60.)

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 software continues to cycle among the file paths, as many times as needed, until all data partitions for the data set are stored. The path selected 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**.

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**.

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

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 has integrity constraints, then the integrity constraints are dropped when the file is created in the SPD Engine format. The following chart of file characteristics indicates whether the characteristic can be retained or dropped, or if it results in an error when converted.

Table 2.1 Conversion Results for Base SAS File Characteristics

Base SAS File Characteristic	Conversion Result
Indexes	Rebuilt in SPD Engine (in parallel if ASYNCINDEX=YES)
Default Base SAS engine COMPRESS=YES |CHAR |BINARY *	Converts with compression if the data file is not encrypted

Base SAS File Characteristic	Conversion Result
Default Base SAS engine ENCRYPT=YES *	Converts with encryption
Integrity constraints	Dropped without ERROR
Audit file	Dropped without ERROR
Generations file	Dropped without ERROR

* If the default Base SAS engine file 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 as shown in the following example. 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 the 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 and both index 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 'old_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* with the OUT= option, or PROC SQL with the CREATE TABLE= option.

* except PROC MIGRATE

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

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. △

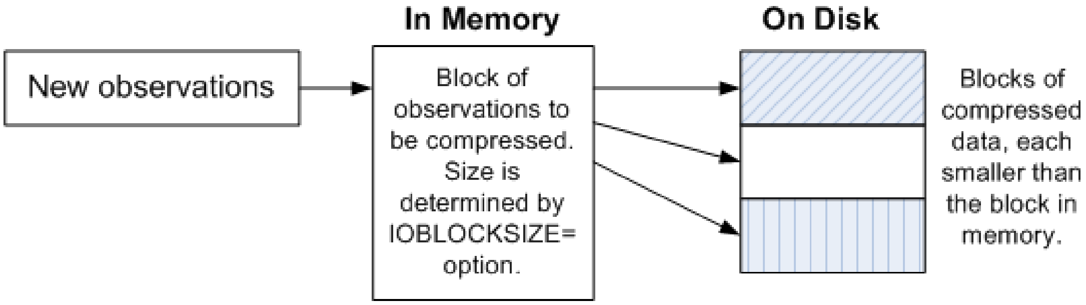
```
libname cardata spde '/report_area';

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
;
```

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 migrating a default Base SAS engine data set that is both compressed and encrypted, the encryption is retained, but the compression is dropped.

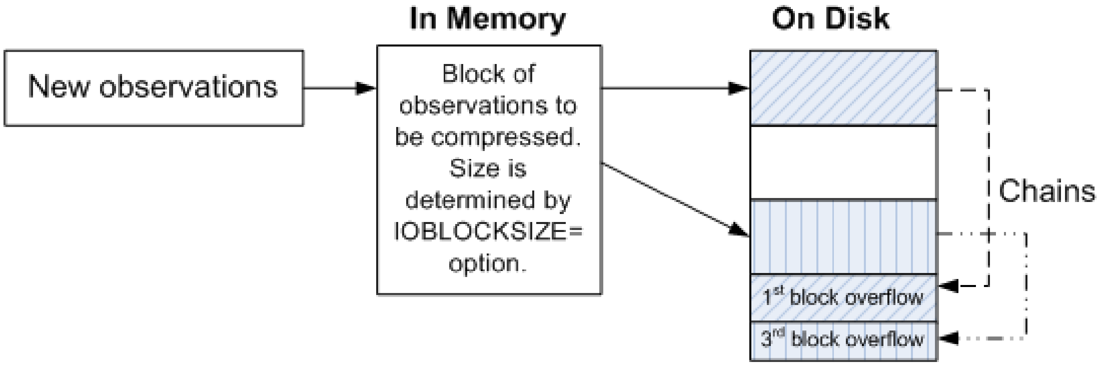
Once a compressed data set is created, you cannot change its block size. The compressed blocks are stored linearly, with no spaces between the blocks. The following figure illustrates how the blocks are stored on the disk:

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:

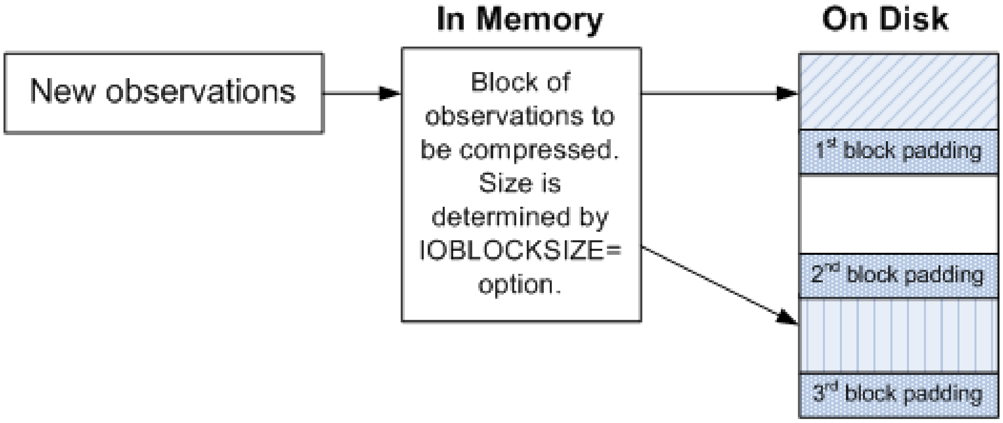
Figure 2.2 Compressed Blocks with Overflow


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, setting IOBLOCKSIZE= to the block size appropriate for the output data set.

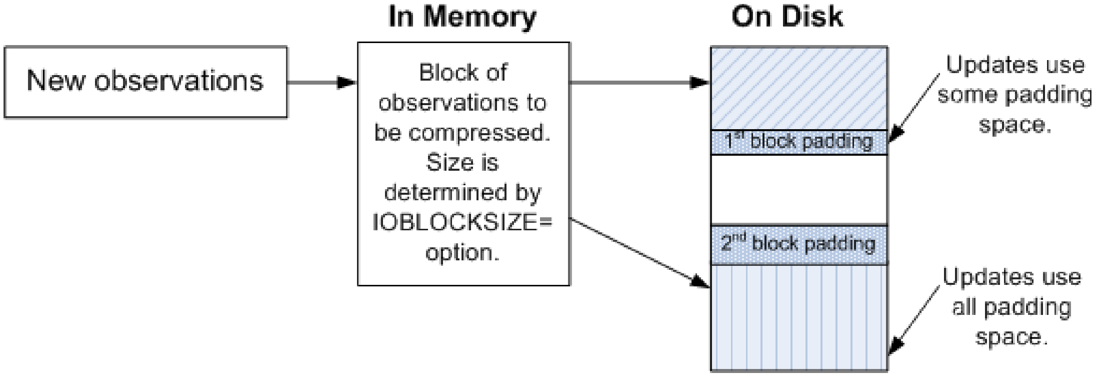
When the data set is expected to be updated frequently, it is recommended that you use PADCOMPRESS=. SPD Engine creates a padded space for each block, instead of creating new block fragments. The following figure illustrates how each block has padded space for updates:

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, instead of creating new block fragments. 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:

```
-          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.

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. *

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

Table 2.2 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

* 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.

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

Efficient Indexing in the SPD Engine

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

When an SPD Engine data set is encrypted, only the data component files are encrypted. None of the other files are encrypted, such as the metadata and index files.

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 42.

Use this option with the DATA step INDEX= option, with PROC DATASETS INDEX CREATE commands, or in the PROC APPEND statement when creating an SPD Engine data set from a default Base SAS engine data set that has an index. Each method enables all of the declared indexes to be populated from a single scan of the data set.

Note: If you create an SPD Engine data set from a default Base SAS engine data set that is encrypted and that has an index, the index is not encrypted in the SPD Engine data set. For more information, see “Converting Default Base SAS Engine Data Sets to SPD Engine Data Sets” on page 15. △

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 81 to allocate disk space and “SPDEINDEXSORTSIZE= System Option” on page 79 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.

PART 2

Reference

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*’ <*options*> ;

libref
: a name that is up to eight characters long and that conforms to the rules for SAS names. You cannot specify TEMP as a libref for an SPD Engine library unless TEMP is not used as an environment variable.

’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.

Note: You cannot change the names of the locations of the files. When you specify the DATAPATH=, INDEXPATH=, METAPATH=, or primary path LIBNAME options, make sure the identical paths that were used when the data set was created are used every time you access the data sets. The names of these locations are stored internally in the data set. △

options
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.

ACCESS= LIBNAME Statement Option

Determines the access level of the data source.

Default: none

Syntax

ACCESS=READONLY

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

Corresponding data set option: BYSORT=

Affected by data set option: BYNOEQUALS=

Default: YES

Syntax

BYSORT=YES | NO

YES
specifies to automatically sort the data based on the BY variables whenever a BY statement is encountered, instead of invoking PROC SORT before a BY statement.

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 81.

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 on the specified BY variables. This result can be from a previous sort using PROC SORT, or from the data set having been created in BY variable order. 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. Therefore, you set BYSORT=NO in the LIBNAME statement and subsequently a BY statement is encountered. An error occurs unless your data has been sorted (either by previously using PROC SORT or because it was created in sorted order). Set BYSORT=YES in the DATA or PROC step, for input or update 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.
```

Examples

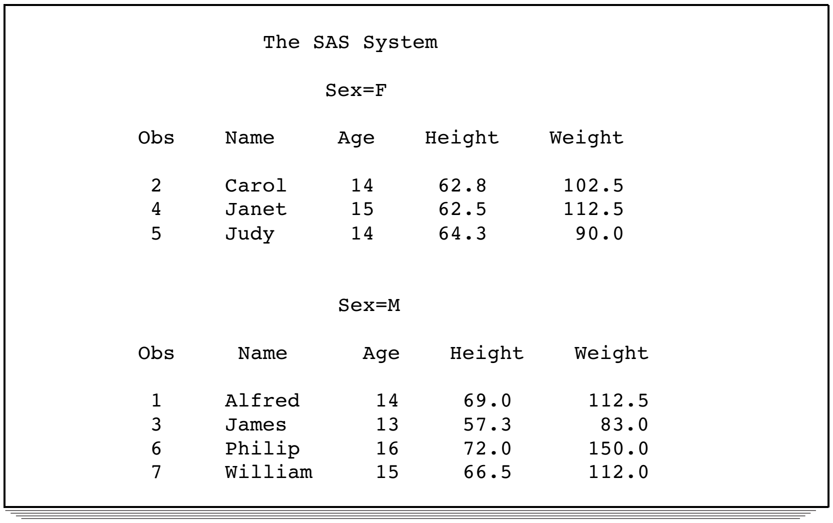
Example 1: Group Formatting with BYSORT=YES by Default

```
libname growth spde 'D:\SchoolAge';
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 'D:\SchoolAge' 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.
```

DATAPATH= LIBNAME Statement Option

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

Affected by: PARTSIZE= data set or LIBNAME option

Default: the primary path specified in the LIBNAME statement

Syntax

DATAPATH=(*'path1' 'path2'...*)

'pathn'

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.

You cannot change the pathnames of the files. When you specify the DATAPATH=, INDEXPATH=, METAPATH=, or primary path LIBNAME options, make sure that the identical paths that were used when the data set was created are used every time you access the data sets. The names of these locations are stored internally in the data set. If you change any part of the pathname, the SPD Engine might not be able to find the data set or might damage the data set. △

Example

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.

Used with: STARTOBS=

Corresponding data set option: ENDOBS=

Default: the last observation in the data set

Restriction: use ENDOBS= with input data sets only

Restriction: cannot be used with OBS= system and data set option or FIRSTOBS= system and data set option

Syntax

ENDOBS=*n*

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 Chapter 4, "SPD Engine Data Set Options," on page 41 for information about using the ENDOBS= data set option in WHERE processing.)

Example

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 'D:\SchoolAge' 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;
```

The output is shown:

```
                        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 indexes when processing BY statements in the SPD Engine.

Corresponding data set option: IDXBY=

Affected by data set option: BYSORT=

Default: YES

Syntax

IDXBY=YES | NO

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.
  ```

Examples

Example 1: Using the IDXBY=NO LIBNAME Option

```
libname permdata spde ’c:/sales’ 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 ’c:\sales’ 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

Syntax

INDEXPATH=(*’path1’*)<*’path2’...*>

'pathn'

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

The SPD Engine creates the two index component files in the locations specified. If there are multiple pathnames specified, the first path is randomly selected. When there is not enough space in the path, the index component files overflow into the next file path specified, and so on.

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.

You cannot change the pathnames of the files. When you specify the DATAPATH=, INDEXPATH=, METAPATH=, or primary path LIBNAME options, make sure that the identical paths that were used when the data set was created are used every time you access the data sets. The names of these locations are stored internally in the data set. If you change any part of the pathname, the SPD Engine might not be able to find the data set or might damage the data set. △

Example

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.

METAPATH= LIBNAME Statement Option

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

Default: the primary path specified in the LIBNAME statement

Syntax

METAPATH=(*'path1'*)<*'path2'...*>

'pathn'

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

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

Details

The METAPATH= option is specified for space that is exclusively overflow space for the metadata component file. The metadata component file for each data set must begin in the primary path, and overflow occurs to 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.

You cannot change the pathnames of the files. When you specify the DATAPATH=, INDEXPATH=, METAPATH=, or primary path LIBNAME options, make sure that the identical paths that were used when the data set was created are used every time you access the data sets. The names of these locations are stored internally in the data set. If you change any part of the pathname, the SPD Engine might not be able to find the data set or might damage the data set. △

Example

The following example creates overflow metadata file partitions as needed using the path /DISK1/METAFLOW1.

When /METADISK/METADATA is full, the metadata overflows to /DISK1/METAFLOW1.

```
libname mylib spde '/metadisk/metadata'
    datapath=('/disk1/dataflow1' '/disk2/dataflow2')
   metapath=('/disk1/metaflow1');
```

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 the SPD Engine data set is created. This size is a fixed size. This specification applies only to the data component files.

Corresponding data set option: PARTSIZE=

Used with: MINPARTSIZE= system option

Affected by LIBNAME option: DATAPATH=

Default: 128 MB

Syntax

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

***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. In SAS 9.2, 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 before 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. If you have a table with an observation length greater than 65K, you might find that the PARTSIZE= 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 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

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. △

Example

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.

Used with: ENDOBS=

Corresponding data set option: STARTOBS=

Default: the first observation in the data set

Restriction: use STARTOBS= with input data sets only

Restriction: cannot be used with OBS= system and data set option or FIRSTOBS= system and data set option

Syntax

STARTOBS=*n*

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. △

(Refer to Chapter 4, “SPD Engine Data Set Options,” on page 41 for information about using the STARTOBS= data set option in WHERE processing.)

Example

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 ’D:\SchoolAge’ 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 3.2 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 directory.

Default: NO

Syntax

TEMP=YES | NO

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 of its files contained are deleted at the end of the session.

You can use TEMP= with the SAS option USER= to create a temporary directory to store interim 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

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 table has its metadata file stored in 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;
```

SPD Engine LIBNAME Statement Options List

"ACCESS= LIBNAME Statement Option" on page 26
specifies that data sets can be read, but not updated or created.

"BYSORT= LIBNAME Statement Option" on page 26
specifies for the SPD Engine to perform an automatic sort when it encounters a BY statement.

"DATAPATH= LIBNAME Statement Option" on page 29
specifies a list of paths in which to store data partitions (.dpf) for an SPD Engine data set.

"ENDOBS= LIBNAME Statement Option" on page 30
specifies the end observation number in a user-defined range of observations to be processed.

"IDXBY= LIBNAME Statement Option" on page 31
specifies whether to use an index when processing BY statements in the SPD Engine.

"INDEXPATH= LIBNAME Statement Option" on page 32
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.

"METAPATH= LIBNAME Statement Option" on page 33
specifies a list of overflow paths in which to store metadata (.mdf) component files for an SPD Engine data set.

"PARTSIZE= LIBNAME Statement Option" on page 34
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.

"STARTOBS= LIBNAME Statement Option" on page 36
specifies the starting observation number in a user-defined range of observations to be processed.

"TEMP= LIBNAME Statement Option" on page 38
specifies to store the library in a temporary subdirectory of the primary directory.

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 if a data set option is used subsequent to a LIBNAME option of the same name, the value of the data set option takes precedence.

Syntax

(*option1=value1* <(option2=value2>...)

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

ASYNCINDEX= Data Set Option

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

Valid in: DATA step and PROC step

Default: NO

Syntax

ASYNCINDEX=YES | NO

YES
creates the indexes in parallel (asynchronously).

NO
creates one index at a time (synchronously).

Details

The SPD Engine can create multiple indexes for a data set at the same time. 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 creating one index at a time, the default for this option is NO. Parallel creation requires additional utility work space and additional memory, which might not be available. If the index creation fails due to insufficient resources, you can do one of the following:

- □ set the SAS system option to MEMSIZE=0*
- □ increase the size of the utility file space using the SPDEUTILLOC= system option

You increase the memory space that is used for index sorting using the SPDEINDEXSORTSIZE= system option. If you specify to create indexes in parallel, specify a large-enough space using the SPDEUTILLOC= system option.

Example

The DATASETS procedure has the flexibility to use 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 PatientNo PatientClass;
   run;
```

* for OpenVMS on HP Integrity Servers, increase the paging file quota (PGFLQUO); for z/OS, increase the REGION size.

```
      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 with: BYSORT=YES data set option

Default: NO

Syntax

BYNOEQUALS=YES | NO

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 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 Engines 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)

Condition:	Data Set Order Preserved?
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=tempdata.housreps(bynoequals=yes);
   by state;
   where state in ('CA' 'TX');
run;
```

The output is shown:

Output 4.1 BYNOEQUALS=YES

```
                          With BYNOEQUALS=YES
                               State=CA

        Obs    Representative                  District
         26    Berman, Howard L.               26th
         55    Calvert, Ken                    43d
         60    Capps, Lois                     22d
         76    Cardoza, Dennis                 18th
         22    Becerra, Xavier                 30th
          9    Baca, Joe                       42d
         80    Cox, Christopher                47th
         38    Bono, Mary                      44th
         89    Cunningham, Randy "Duke"        50th


                              State=TX

        Obs    Representative                  District

         87    Culberson, John Abney           7th
         20    Barton, Joe                     6th
         75    Combest, Larry                  19th
         36    Bonilla, Henry                  23d
          8    Armey, Richard K.               26th
         23    Bentsen, Ken                    25th
         44    Brady, Kevin                    8th
```

Example 2: BYNOEQUALS=NO

The following example shows the output with BYNOEQUALS=NO:

```
title 'With BYNOEQUALS=NO';
proc print data=tempdata.housreps(bynoequals=no);
   by state;
   where state in ('CA' 'TX');
run;
```

The output is shown:

Output 4.2 BYNOEQUALS=NO

```
                         With BYNOEQUALS=NO

                               State=CA

       Obs    Representative                   District

         9    Baca, Joe                        42d
        22    Becerra, Xavier                  30th
        26    Berman, Howard L.                26th
        38    Bono, Mary                       44th
        55    Calvert, Ken                     43d
        60    Capps, Lois                      22d
        76    Cardoza, Dennis                  18th
        80    Cox, Christopher                 47th
        89    Cunningham, Randy "Duke"         50th


                               State=TX

       Obs    Representative                   District

         8    Armey, Richard K.                26th
        20    Barton, Joe                      6th
        23    Bentsen, Ken                     25th
        36    Bonilla, Henry                   23d
        44    Brady, Kevin                     8th
        75    Combest, Larry                   19th
        87    Culberson, John Abney            7th
```

BYSORT= Data Set Option

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

Valid in: DATA step and PROC step

Affects data set option: BYNOEQUALS=

Default: YES

Syntax

BYSORT=YES | NO

YES
specifies to automatically sort the data based on the BY variables whenever a BY statement is encountered, instead of invoking PROC SORT before a BY statement.

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 81.

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 on the specified BY variables. This result can be from a previous sort using PROC SORT, or from the data set having been created in BY variable order. 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. Therefore, you set BYSORT=NO in the LIBNAME statement and subsequently a BY statement is encountered. An error occurs unless your data has been sorted (either by previously using PROC SORT or because it was created in sorted order). Set BYSORT=YES in the DATA or PROC step, for input or update 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.
  ```

Examples

Example 1: Group Formatting with BYSORT=YES by Default

```
libname growth spde 'D:\SchoolAge';
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 'D:\SchoolAge';
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

Restriction: Cannot be used with ENCRYPT=YES or ENCRYPT=RC4

Related data set options: "IOBLOCKSIZE= Data Set Option" on page 56 and "PADCOMPRESS= Data Set Option" on page 60

Default: NO

Syntax

COMPRESS= NO | YES | CHAR | BINARY

NO
performs no data set compression.

YES | CHAR
performs the Run Length Compression (SPDSRLC2) on the data set.

BINARY
performs the Ross Data Compression (SPDSRDC) on the data set.

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 56 when you create the data set. To add padding to the newly compressed blocks, specify “PADCOMPRESS= Data Set Option” on page 60 when creating or updating the data set. For more information, see “Compressing SPD Engine Data Sets” on page 17.

Examples

Example 1: COMPRESS=BINARY

Here is PROC CONTENTS output from the SPD Engine with COMPRESS=BINARY.

```
                                 The SAS System                       1

                             The CONTENTS Procedure

Data Set Name        URBANLIB.HOUSING       Observations           96754
Member Type          DATA                   Variables              56
Engine               SPDE                   Indexes                0
Created              Tuesday, October 02,   Observation Length     448
                     2007 01:16:21 PM
Last Modified        Tuesday, October 02,   Deleted Observations   0
                     2007 01:16:21 PM
Protection                                  Compressed             BINARY
Data Set Type                               Point to Observations  YES
Label                                       Sorted                 NO
Data Representation  WINDOWS_32
Encoding             wlatin1  Western (Windows)


                       Engine/Host Dependent Information

                   Blocking Factor (obs/block)     146
                   Disk Compression Name           SPDSRDC
                   Data Partsize                   134249920
                   -          Compressed Info      -
                   Number of compressed blocks     1326
                   Raw data blocksize              32704
                   Number of blocks with overflow  0
                   Max overflow chain length       0
                   Block number for max chain      0
                   Min overflow area               0
                   Max overflow area               0
```

Example 2: COMPRESS=CHAR

Here is PROC CONTENTS output from the SPD Engine with COMPRESS=CHAR.

```
                                The SAS System                         1

                             The CONTENTS Procedure

Data Set Name        SALESLIB.COSTS         Observations           459568
Member Type          DATA                   Variables              21
Engine               SPDE                   Indexes                0
Created              Tuesday, October 02,   Observation Length     112
                     2007 01:26:30 PM
Last Modified        Tuesday, October 02,   Deleted Observations   0
                     2007 01:26:30 PM
Protection                                  Compressed             CHAR
Data Set Type                               Point to Observations  YES
Label                                       Sorted                 NO
Data Representation  WINDOWS_32
Encoding             wlatin1  Western (Windows)


                       Engine/Host Dependent Information

                   Blocking Factor (obs/block)     585
                   Disk Compression Name           SPDSRLC2
                   Data Partsize                   134249920
                   -          Compressed Info      -
                   Number of compressed blocks     1574
                   Raw data blocksize              32704
                   Number of blocks with overflow  0
                   Max overflow chain length       0
                   Block number for max chain      0
                   Min overflow area               0
                   Max overflow area               0
```

ENCRYPT= Data Set Option

Specifies whether to encrypt an output SPD Engine data set.

Valid in: DATA step and PROC steps

Restriction: use with output data sets only

Restriction: ENCRYPT=YES cannot be used with COMPRESS=

Syntax

ENCRYPT=YES | NO

Syntax Description

YES

encrypts the file. 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 when ENCRYPT=YES. If you forget the passwords, you cannot reset it without assistance from SAS. The process is time-consuming and resource-intensive. △

NO

does not encrypt the file.

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 data set option and the COMPRESS= data set or LIBNAME option, the following error is generated:

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

When you copy a Base SAS data set that is compressed and encrypted to an SPD Engine data set, the compression is dropped. SAS retains the security of the data set instead of the compression.

Example 1: Using the ENCRYPT=YES Option

The following example encrypts the data set:

```
libname depta spde '/datasecret';

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;
```

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

Used with: STARTOBS= data set option

Default: the last observation in the data set

Restriction: use ENDOBS= with input data sets only

Restriction: cannot be used with OBS= system and data set option or FIRSTOBS= system and data set option

Syntax

ENDOBS=*n*

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= 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, except when it is specified in a WHERE expression.

Using ENDOBS= in 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. △

Examples

Example 1: ENDOBS= with SPD Engine

A data set is created and processed by the SPD Engine with ENDOBS=5 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 'c:\temp';

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;
```

The output is shown:

Output 4.4 Four Observations Printed

```
                 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 'c:\temp';

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 4.5 Five Observations Printed

```
                  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 BY statements in the SPD Engine.

Valid in: DATA step and PROC step

Default: YES

Syntax

IDXBY=YES | NO

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.
```

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 to use indexes when processing WHERE expressions in the SPD Engine.

Valid in: DATA step and PROC step

Default: YES

Restriction: WHERENOINDEX= option cannot be used with IDXWHERE=NO option

Syntax

IDXWHERE=YES | NO

YES
uses indexes when processing WHERE expressions.

NO
ignores indexes when processing WHERE expressions.

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 supports four WHERE expression evaluation strategies. For more information, see "SPDEWHEVAL= System Option" on page 83. Strategies 1, 3, and 4 use available indexes and execute the indexed part of the WHERE expression. Evaluation strategy 2 executes the non-indexed part of the WHERE expression.

The first example shows that evaluation strategy 2 is used in the WHERE expression because IDXWHERE=NO was specified. The second example shows that evaluation strategy 1 was used because IDXWHERE=YES was specified.

Examples: WHINIT Log Output (MSGLEVEL=I)

Output 4.6 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';
```

Output 4.7 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';
```

CAUTION:

Do not arbitrarily suppress index use when using both WHERE and BY statements in combination. When you use both a WHERE expression to filter the observations and a BY expression to order the observations, the filtered observations qualified by the WHERE expression 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 for WHERE processing greatly improves the filtering performance feeding into the sort step. △

IOBLOCKSIZE= Data Set Option

Specifies the size in bytes of a block of observations to be compressed.

Valid in: DATA step and PROC step

Affects data set options: "COMPRESS= Data Set Option" on page 47 and "PADCOMPRESS= Data Set Option" on page 60

Default: 32,768 bytes

Syntax

IOBLOCKSIZE=*n*

n

is the size in bytes of a block of observations to be compressed.

Details

The SPD Engine uses blocks in memory to collect the observations to be compressed before they are written to a data component file. IOBLOCKSIZE= specifies the size of these blocks. (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).

Once a compressed data set is created, you cannot change its block size. To resize the block, you must copy the data set to a new data set, setting IOBLOCKSIZE= to the appropriate block size for the output data set.

The default value and the IOBLOCKSIZE= smallest value is 32,768 bytes. You specify an IOBLOCKSIZE= value that complements the data to be accessed. Access to data that is randomly distributed favors a smaller block size, such as 32,768 bytes, because accessing smaller blocks is faster than accessing larger blocks. In contrast, access to data that is uniformly or sequentially distributed or that requires a full data set scan should have a large block size, such as 131,072 bytes.

If the data set is to be updated frequently, use the “PADCOMPRESS= Data Set Option” on page 60 to enable expansion between the compressed blocks on the disk.

If a default Base SAS engine data set is copied to the SPD Engine, you do not have to specify COMPRESS= or IOBLOCKSIZE= unless you want to specify a block size other than the default size because compression is retained.

Note: To add padded space to compressed data sets without changing the compression block size, see PADCOMPRESS= option“PADCOMPRESS= Data Set Option” on page 60. △

Example

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


/*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

Syntax

LISTFILES=YES | NO

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 and the CONTENTS procedure to list the complete pathnames of all of the component files of an SPD Engine data set.

Example

```
proc contents data=hrdept.names (listfiles=yes);
```

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

Output 4.8 LISTFILES=YES

```
                          The SAS System                         1

                      The CONTENTS Procedure

Data Set Name        HRDEPT.NAMES    Observations          1249854
Member Type          DATA            Variables             11
Engine               SPDE            Indexes               1
Created              Tuesday,        Observation Length    120
                     October 02,
                     2007 01:46:23
                     PM
Last Modified        Tuesday,        Deleted Observations  0
                     October 02,
                     2007 01:47:21
                     PM
Protection                           Compressed            NO
Data Set Type                        Sorted                NO
Label
Data Representation  WINDOWS_32
Encoding             wlatin1
                     Western
                     (Windows)

                Engine/Host Dependent Information

        Blocking Factor (obs/block)              546
        Data Partsize                            16776960
        -   Alphabetic List of Index Info        -
        Index                                    Id
        KeyValue (Min)                           64587
        KeyValue (Max)                           64587
        Number of discrete values                1
        -   Metadata Files                       -
        d:\hrdir1\names.mdf.0.0.0.spds9          -
        -   Data Files                           -
        d:\hrdird\names.dpf.d__sasmb.0.2.spds9   -
        d:\hrdird\names.dpf.d__sasmb.1.2.spds9   -
        d:\hrdird\names.dpf.d__sasmb.2.2.spds9   -
        d:\hrdird\names.dpf.d__sasmb.3.2.spds9   -
        d:\hrdird\names.dpf.d__sasmb.4.2.spds9   -
        d:\hrdird\names.dpf.d__sasmb.5.2.spds9   -
        d:\hrdird\names.dpf.d__sasmb.6.2.spds9   -
        d:\hrdird\names.dpf.d__sasmb.7.2.spds9   -
        d:\hrdird\names.dpf.d__sasmb.8.2.spds9   -
        -   Index Files                          -
        d:\hrdiri\names.idxid.d__sasmb.0.2.spds9 -
        d:\hrdiri\names.hbxid.d__sasmb.0.2.spds9 -

           Alphabetic List of Variables and Attributes

                  #    Variable      Type    Len

                  7    Address01     Char     27
                  8    Address02     Char      9
                  9    City          Char     12
                  3    CostCenter    Num       8
                  5    FirstName     Char      8
                  2    Id            Num       8

          Alphabetic List of Variables and Attributes

                  #    Variable      Type    Len

                  4    LastName      Char     18
                  6    MiddleName    Char      9
                 10    State         Char      2
                 11    ZipCode       Char     10
                  1    i             Num       8

             Alphabetic List of Indexes and Attributes

                                       # of
                                     Unique
                       #    Index    Values

                       1    Id            1
```

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

Related to data set option: "COMPRESS= Data Set Option" on page 47 and "IOBLOCKSIZE= Data Set Option" on page 56

Default: 0

Syntax

PADCOMPRESS= *n*

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 with: MINPARTSIZE= system option

Corresponding LIBNAME option: PARTSIZE=

Affected by LIBNAME option: DATAPATH=

Default: 128M

Syntax

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

***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 9.2, 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 before 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= 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 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. △

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

Restriction: use STARTOBS= with input data sets only

Restriction: cannot be used with OBS= system and data set option or FIRSTOBS= system and data set option

Syntax

STARTOBS=*n*

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=.

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, except when it 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.

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 'c:\temp';

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 4.9 Six Observations Printed

```
                  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 specified. PROC PRINT prints five observations, which are all of the observations qualified by the WHERE expression, starting with the fifth qualified observation. FIRSTOBS= is not supported in the SPD Engine.

```
libname growth v9 'c:\temp';

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 4.10 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.

Valid in: PROC SQL

Affects the data set option: UNIQUESAVE=

Default: NO

Syntax

SYNCADD=YES | NO

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

NO
processes multiple observations at a time (asynchronously).

Details

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

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

When SYNCADD=NO, observations are added in blocks (pipelining), which is usually faster. If you are adding observations to a data set with a unique index and the SPD Engine encounters an observation with a duplicate index value, 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

In the following example, two data sets, UQ01A and UQ01B, are created. On UQ01A, PROC SQL creates a unique composite index, and then inserts new values into the data set with SYNCADD=NO (inserting blocks of data). Duplicate values are stored in a separate file because UNIQUESAVE= is set to YES.

Then, PROC SQL creates a unique composite index on UQ01B and inserts new values with SYNCADD=YES. PROC SQL stops when duplicate values are encountered and restores the data set. Even though UNIQUESAVE=YES, it is ignored. The SAS log is shown:

```
1097  libname userfile spde ’c:\temp’;
NOTE: Libref SPDS USERFILE was successfully assigned as follows:
      Engine:        SPD Engine
      Physical Name: d3727.na.sas.com:528c:\temp\
1098
1099  data uq01a uq01b;
1100    input z $ 1-20 x y;
1101    list;
1102    datalines;

RULE:----+----1----+----2----+----3----+----4----+----5----+----6----+----7
1103        one                  1 10
1104        two                  2 20
1105        three                3 30
1106        four                 4 40
1107        five                 5 50
NOTE: The data set USER.UQ01A has 5 observations and 3 variables.
NOTE: The data set USER.UQ01B has 5 observations and 3 variables.
NOTE: DATA statement used (Total process time):
      real time           0.51 seconds
      cpu time            0.06 seconds



1108  ;
1109
1110
1111  proc sql    sortseq=ascii exec noerrorstop;
1112  create unique index comp
1113    on uq01a  (x, y);
NOTE: Composite index comp has been defined.
1114  insert into uq01a(syncadd=no,uniquesave=yes)
1115    values(’rollback1’, -80, -80)
1116    values(’rollback2’,-90, -90)
1117    values(’nonunique’, 2, 20)
1118   ;
NOTE: 3 observations were inserted into USER.UQ01A.

WARNING: Duplicate values not allowed on index comp for file USER.UQ01A.
         (Occurred 1 times.)
NOTE: Duplicate records have been stored in file USER._D2DAAF7.
NOTE: PROCEDURE SQL used (Total process time):
      real time           0.99 seconds
      cpu time            0.05 seconds



1119  proc sql    sortseq=ascii exec noerrorstop;
1120  create unique index comp
1121    on uq01b  (x, y);
NOTE: Composite index comp has been defined.
1122  insert into uq01b(syncadd=yes,uniquesave=yes)
1123    set z=’rollback3’, x=-60, y=-60
1124    set z=’rollback4’, x=-70, y=-70
1125    set z=’nonunique’, x=2, y=20;
ERROR: Duplicate values not allowed on index comp for file UQ01B.
```

```
NOTE: Deleting the successful inserts before error noted above to restore
      data set to a consistent state.
1126
NOTE: PROCEDURE SQL used (Total process time):
      real time           0.26 seconds
      cpu time            0.17 seconds


1127  proc compare data=uq01a compare=uq01b;run;

NOTE: There were 7 observations read from the data set USER.UQ01A.
NOTE: There were 5 observations read from the data set USER.UQ01B.
NOTE: PROCEDURE COMPARE used (Total process time):
      real time           0.51 seconds
      cpu time            0.05 seconds
```

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

Affected by system option: SPDEMAXTHREADS=

Default: the value of the SPDEMAXTHREADS= system option, if set. Otherwise, the default is two times the number of CPUs on your computer

Syntax

THREADNUM=*n*

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.

SPDEMAXTHREADS=, a configurable system option, imposes an upper limit on the consumption of system resources and, therefore, 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

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 nonunique 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

Affected by the data set option: SYNCADD=NO

Used with: SPDSUSDS automatic macro variable

Default: NO

Syntax

UNIQUESAVE=YES | NO

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. △

Examples

In the following example, two data sets with unique indexes on the variable NAME are created, and then appended together using PROC APPEND with UNIQUESAVE=YES. The SAS log is shown:

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

RULE:----+----1----+----2----+----3----+----4----+----5----+----6----+
5          Jill 4344
6          Jack 5589
7          Jim 8888
8          Sam 3334
NOTE: The data set EMPLOYEE.EMP1 has 4 observations and 2 variables.
NOTE: DATA statement used (Total process time):
      real time           9.98 seconds
      cpu time            1.28 seconds


9    run;

10   data employee.emp2 (index=(name/unique));
11             input name $ exten;
12             list; datalines;

RULE:----+----1----+----2----+----3----+----4----+----5----+----6----+
13                  Jack 4443
14                  Ann 8438
15                  Sam 3334
16                  Susan 5321
17                  Donna 3332
NOTE: The data set EMPLOYEE.EMP2 has 5 observations and 2 variables.
NOTE: DATA statement used (Total process time):
      real time           0.04 seconds
      cpu time            0.04 seconds


18             run;
```

```
19   proc append data=employee.emp2 base=employee.emp1
20                  (syncadd=no uniquesave=yes);
21             run;

NOTE: Appending EMPLOYEE.EMP2 to EMPLOYEE.EMP1.
NOTE: There were 5 observations read from the data set EMPLOYEE.EMP2.
NOTE: 3 observations added.
NOTE: The data set EMPLOYEE.EMP1 has 7 observations and 2 variables.
WARNING: Duplicate values not allowed on index name for file
         EMPLOYEE.EMP1. (Occurred 2 times.)
NOTE: Duplicate records have been stored in file EMPLOYEE._D3596FF.
NOTE: PROCEDURE APPEND used (Total process time):
      real time           6.25 seconds
      cpu time            1.26 seconds


22   proc print data=employee.emp1;
23             title 'Listing of Final Data Set';
24             run;

NOTE: There were 7 observations read from the data set EMPLOYEE.EMP1.
NOTE: PROCEDURE PRINT used (Total process time):
      real time           2.09 seconds
      cpu time            0.40 seconds


25
26   proc print data=&spdsusds;
27             title 'Listing of Rejected observations';
28             run;

NOTE: There were 2 observations read from the data set EMPLOYEE._D3596FF.
NOTE: PROCEDURE PRINT used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds
```

Output 4.11 UNIQUESAVE=YES

```
                  Listing of Final Data Set

                    Obs    name    exten
                     1     Jill     4344
                     2     Jack     5589
                     3     Jim      8888
                     4     Sam      3334
                     5     Ann      8438
                     6     Susan    5321
                     7     Donna    3332

               Listing of Rejected observations

               Obs    name    exten    XXX00000
                1     Jack     4443      name
                2     Sam      3334      name
```

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

Syntax

WHERENOINDEX=(*name1 name2...*)

(*name1 name2...*)
a list of index names to exclude from the WHERE planner.

Example

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. △

SPD Engine Data Set Options List

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

"BYNOEQUALS= Data Set Option" on page 43
specifies the index output order of data set observations that have identical values for the BY variable.

"BYSORT= Data Set Option" on page 45
specifies for the SPD Engine to perform an automatic sort when it encounters a BY statement.

"COMPRESS= Data Set Option" on page 47
compresses SPD Engine data sets on disk.

Restriction: Compression and encryption are mutually exclusive in SPD Engine.

"ENCRYPT= Data Set Option" on page 49
Encrypts data files.

Restriction: Compression and encryption are mutually exclusive in SPD Engine.

"ENDOBS= Data Set Option" on page 50
specifies the end observation number in a user-defined range of observations to be processed.

"IDXBY= Data Set Option" on page 53
specifies whether to use an index when processing BY statements in the SPD Engine.

"IDXWHERE= Data Set Option" on page 54
specifies to use indexes when processing WHERE expressions in the SPD Engine.

"IOBLOCKSIZE= Data Set Option" on page 56
specifies the size of a block of observations to be compressed.

"LISTFILES= Data Set Option" on page 57
specifies whether the CONTENTS procedure lists the complete pathnames of all of the component files in an SPD Engine data set.

"PADCOMPRESS= Data Set Option" on page 60
specifies the number of bytes to add to compressed blocks in a data set opened for OUTPUT or UPDATE.

"PARTSIZE= Data Set Option" on page 60
specifies the maximum partition size of the data component files. PARTSIZE= is also a LIBNAME option.

"STARTOBS= Data Set Option" on page 62
specifies the starting observation number in a user-defined range of observations to be processed.

"SYNCADD= Data Set Option" on page 65
specifies to process one observation at a time or a block of observations at a time.

"THREADNUM= Data Set Option" on page 67
specifies the maximum number of threads to use for the SPD Engine processing.

"UNIQUESAVE= Data Set Option" on page 68
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= Data Set Option" on page 71
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

Restriction: 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.

ENCRYPT=
encrypts data files

Restriction: Compression and encryption are mutually exclusive in SPD Engine.

SAS Data Set Options Not Supported by the SPD Engine

□ BUFNO=
□ BUFSIZE=
□ DLDMGACTION=
□ ENCODING=
□ FIRSTOBS=
□ GENMAX=
□ GENNUM=
□ IDXNAME=
□ OUTREP=
□ POINTOBS=
□ REUSE=
□ TOBSNO=

CHAPTER 5

SPD Engine System Options

Introduction to SPD Engine System Options

SAS system options are instructions that affect your SAS session. They control the way that 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 *option1 <...option-n>*;

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.

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

Restriction: cannot be used with ENCRYPT=YES or ENCRYPT=RC4

Default: NO

Syntax

COMPRESS= NO | YES | CHAR | BINARY

NO
performs no data set compression.

YES | CHAR
performs the Run Length Compression (SPDSRLC2) on the data set.

BINARY
performs the Ross Data Compression (SPDSRDC) on the data set.

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 56 when you create the data set. To add padding to the newly compressed blocks, specify “PADCOMPRESS= Data Set Option” on page 60 when creating or updating the data set. For more information, see “Compressing SPD Engine Data Sets” on page 17.

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

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

Default: 75

Syntax

MAXSEGRATIO=*n*

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 you are querying responds to shifting the index-segment ratio. △

Examples

Example 1: MAXSEGRATIO=65 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: MAXSEGRATIO=0 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: MAXSEGRATIO=100 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 size that the data component partitions can be. The value is specified when the SPD Engine data set is created.

Valid in: configuration file, SAS invocation

Related to: PARTSIZE= data set and LIBNAME option

Default: 16M

Syntax

MINPARTSIZE=*n* | *n*K | *n*M | *n*G

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. In SAS 9.2, 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 before 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.

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

Affected by data set option: MEMSIZE=

Default: 32M

Syntax

SPDEINDEXSORTSIZE=*n* | *n*K | *n*M | *n*G

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 SPDEINDEXSORTSIZE= option specifies the maximum amount of memory that can be used when sorting values for creating an index. When indexes are created in parallel (because ASYNCINDEX=YES), the value you specify in SPDEINDEXSORTSIZE= is divided among all of the concurrent index creation threads.

Perform one of the following if the index creation fails due to insufficient memory:

- □ restart SAS with the SAS system option MEMSIZE=0*
- □ increase the size of the utility file space using the SPDEUTILLOC= system option

You increase the memory space that is used for index sorting using the SPDEINDEXSORTSIZE= system option. If you specify to create indexes in parallel, specify a large-enough space using the SPDEUTILLOC= system option.

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

Default: 0

Syntax

SPDEMAXTHREADS=*n*

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 two times the number of CPUs on your computer.

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

* for OpenVMS on HP Integrity Servers, increase the paging file quota (PGFLQUO); for z/OS, increase the REGION size.

Default: 32M

Syntax

SPDESORTSIZE=*n* | *n*K | *n*M | *n*G

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 you specify for SPDESORTSIZE= should be multiplied by the number of processes that are in parallel. This total should be less than the physical memory available to your process. 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*
- □ increase the size of the utility file space using the SPDEUTILLOC= system option

You increase the memory that is used for index sorting using the SPDEINDEXSORTSIZE= system 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. △

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

See: The SAS Companion for your operating system details how to specify system options.

Syntax

SPDEUTILLOC=*location* | *(location-1 ...location-n)*

* for OpenVMS on HP Integrity Servers, increase the paging file quota (PGFLQUO); for z/OS, increase the REGION size.

location
is an existing directory where the utility files are created.

(location-1 ...location-n)
a series of existing directories where the utility files are created.

Note: *Location* can be enclosed in single or double quotation marks. Quotation marks are required if *location* contains embedded blanks. △

Operating Environment Information: The SAS Companion for your operating system details how to specify system options. △

Details

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. If the SPDEUTILLOC= system option or the UTILLOC= SAS system option is not specified, and the SPD Engine cannot locate the SAS WORK directory (or does not have Write permission to it), the location for temporary utility file storage is defined by each operating environment. 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
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
OpenVMS on HP Integrity Servers	UTILLOC= SAS system option, if specified	WORK= SAS system option, if specified, and ODS-5 directory[1]	sys$scratch:

[1] If the WORK= SAS system option does not specify an ODS-5 directory, and if the SAS session was started with an ODS-5 file specification of SASROOT, the utility files will be created in the SASROOT directory. Otherwise, there will be no default location, and the LIBNAME assignment will fail.

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

Default: COST

Syntax

SPDEWHEVAL=COST | EVAL1 | EVAL3EVAL4

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 underutilized 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 in which all of the key variables have a simple index and no condition tests for non-equality. Multi-threading might be used to retrieve the observations.

Details

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 underutilized.

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 you are querying responds to rules-based WHERE planning. △

SPD Engine System Options List

"COMPRESS= System Option" on page 76
specifies to compress SPD Engine data sets on disk as they are being created.

"MAXSEGRATIO= System Option" on page 77
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= System Option" on page 78
specifies the minimum partition size to use when creating SPD Engine data sets.

"SPDEINDEXSORTSIZE= System Option" on page 79
specifies the memory space size of the sorting utility can use when sorting values for creating an index.

"SPDEMAXTHREADS= System Option" on page 80
specifies the maximum number of threads that the SPD Engine can spawn for I/O processing.

"SPDESORTSIZE= System Option" on page 80
specifies the memory space size needed for sorting operations used by the SPD Engine.

"SPDEUTILLOC= System Option" on page 81
specifies one or more file system locations in which the SPD Engine can temporarily store utility files.

"SPDEWHEVAL= System Option" on page 83
specifies the process to determine which observations meet the condition or conditions of a WHERE expression.

SAS System Options That Behave Differently with SPD Engine

MSGLEVEL=
the value I enables WHINIT planner output

MSGLEVEL=I
produces WHERE optimization information in the SAS log

COMPRESS=
cannot perform user-defined compression

DLDMGACTION=
does not affect the SPD Engine. If an SPD Engine data set is damaged, it must be restored from a system backup file.

FIRSTOBS=
cannot be used in the SPD Engine

PART 3

Appendix

APPENDIX 1

Recommended Reading

Recommended Reading

Here is the recommended reading list for this title:

□ *Base SAS Procedures Guide*

□ *SAS Language Reference: Concepts*

□ *SAS Language Reference: Dictionary*

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

SAS Publishing Sales
SAS Campus Drive
Cary, NC 27513
Telephone: 1-800-727-3228
Fax: 1-919-531-9439
E-mail: **sasbook@sas.com**
Web address: **support.sas.com/bookstore**

Customers outside the United States and Canada, please contact your local SAS office for assistance.

Glossary

block (of data)
a group of observations in a data set. If an application is thread-enabled, it can read, write, and process the observations faster when they are delivered as a block than when 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 of the data component of an SPD Engine data set. See also partition, partitioned data set.

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 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.

multi-threading
See threading.

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 uses multiple CPUs to process independent threads of an application's computations. 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, partitioned data set.

partitioned data set
in the SPD Engine, a data set whose data is stored in multiple physical files (partitions) so that it can span storage devices. One or more partitions can be read in parallel by using threads. This method improves the speed of I/O and processing for very large data sets. See also parallel processing, partition, thread.

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.

process
a functional unit of a program or task. In a thread-enabled operating system, a process can consist of a single thread, or it can contain many threads that collectively perform a complex function. See also thread, thread-enabled operating system.

RAID (redundant array of independent disks)
a type of storage system that consists of many disks and which implements interleaved storage techniques that were developed at the University of California at Berkeley. 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 various levels of redundancy and storage capability. A RAID provides large amounts of data storage inexpensively. Also, because the same data is stored in different places, I/O operations can overlap, which can result in improved performance. See also redundancy.

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 RAID (redundant array of independent disks).

SASROOT
a term that represents the name of the directory or folder in which SAS is installed at your site or on your computer.

scalability
the ability of a software application to function well with little degradation in performance despite changes in the volume of computations or operations that it performs and despite changes in the computing environment. 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, server scalability, SMP (symmetric multiprocessing).

Scalable Performance Data Engine
See SPD Engine.

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. See also server scalability, SMP (symmetric multiprocessing), time to solution.

server scalability
the ability of a server to take advantage of SMP hardware and threaded processing 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 SMP (symmetric multiprocessing), threaded processing.

SMP
a hardware and software architecture that can improve the speed of I/O and processing. An SMP computer has multiple CPUs and a thread-enabled operating system. An SMP computer is usually configured with multiple controllers and with multiple disk drives per controller.

spawn
to start a process or a process thread such as a light-weight process thread (LWPT). See also thread.

SPD Engine
a SAS engine that is able to deliver data to applications rapidly because it organizes the data into a streamlined file format. The SPD Engine also reads and writes partitioned data sets, which enable it to use multiple CPUs to perform parallel I/O functions. See also parallel I/O.

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
See SMP.

thread
a single path of execution of a process in a single CPU, or a basic unit of program execution in a thread-enabled operating system. In an SMP environment, which uses multiple CPUs, multiple threads can be spawned and processed simultaneously. Regardless of whether there is one CPU or many, each thread is an independent flow of control that is scheduled by the operating system. See also SMP (symmetric multiprocessing), thread-enabled operating system, threading.

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 to increase its speed. 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.

threaded processing
processing that is performed in multiple threads on multiple CPUs to improve the speed of CPU-bound applications. See also CPU-bound application.

threading
a high-performance method of data I/O or data processing in which the I/O or processing is divided into multiple threads that are executed in parallel. In the boss-worker model of threading, the same code for the I/O or calculation process is executed simultaneously in separate threads on multiple CPUs. In the pipeline model, a process is divided into steps, which are then executed simultaneously in separate threads on multiple CPUs. See also parallel I/O, parallel processing, SMP (symmetric multiprocessing).

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
a type of SAS expression that specifies a condition for selecting observations for processing by a DATA step or a PROC step. WHERE expressions can contain special operators that are not available in other SAS expressions. WHERE expressions can appear in a WHERE statement, a WHERE= data set option, a WHERE clause, or a WHERE command. See also compound WHERE expression.

Index

Your Turn

We welcome your feedback.

- □ If you have comments about this book, please send them to **yourturn@sas.com**. Include the full title and page numbers (if applicable).
- □ If you have comments about the software, please send them to **suggest@sas.com**.