THE POWER TO KNOW.

SAS® LASR™ Analytic Server 2.1

Administration Guide

SAS® Documentation

The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2013. *SAS® LASR™ Analytic Server 2.1: Administration Guide*. Cary, NC: SAS Institute Inc.

SAS® LASR™ Analytic Server 2.1: Administration Guide

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

July 2013

SAS provides a complete selection of books and electronic products to help customers use SAS® software to its fullest potential. For more information about our e-books, e-learning products, CDs, and hard-copy books, visit **support.sas.com/bookstore** or call 1-800-727-3228.

Contents

What's New In SAS LASR Analytic Server

Overview

SAS LASR Analytic Server 2.1 includes the following changes:

- Enhancements to the LASR procedure
- Enhancements to the VASMP procedure
- Enhancements to the SAS LASR Analytic Server engine

Enhancements to the LASR Procedure

The LASR procedure is enhanced with options that enable greater control over memory use by the SAS LASR Analytic Server.

- The READAHEAD option can be used when loading tables from HDFS. This option causes the server to read blocks from disk more aggressively. The initial load requires more time than without the option, but the server performs the first analysis of the table more quickly.
- The TABLEMEM= option can limit the amount of physical memory used by the server. You can specify the percentage of memory that can be allocated before the server rejects requests to add tables or append data. If the percentage is exceeded, either by the server or any other processes on the machine, adding tables or appending rows fails.
- The EXTERNALMEM= option specifies the percentage of memory that can be allocated before the server stops transferring data to external processes such as the SAS High-Performance Analytics procedures. Because the procedures copy the data from the server (and consume memory to do so), you might want to limit the amount of memory that can be used.

The SIGNER= option for the procedure now supports the HTTPS protocol.

Enhancements to the VASMP Procedure

The following enhancements have been made to the VASMP procedure:

- The SERVERPARM statement has been enhanced to support the HOST= and PORT= options. This enables you to set server parameters before loading a table to memory. You can also set the TABLEMEM= and EXTERNALMEM= options for setting memory limits.
- The SIGNER= option for the procedure now supports the HTTPS protocol.

Enhancements to the SAS LASR Analytic Server Engine

The following enhancements have been made to the SAS LASR Analytic Server engine:

- When you load a table to memory, the server adds a default format to a variable (BEST for numeric and $ for character variables). The engine is displaying the "forced" format. If you specify the NODEFAULTFORMAT option, the format enforced by the server is not used in the SAS session.
- Computed temporary column expressions for the character data type are now supported.
- The SIGNER= option now supports the HTTPS protocol.

Chapter 1

Introduction to the SAS LASR Analytic Server

What is SAS LASR Analytic Server?

The SAS LASR Analytic Server is an analytic platform that provides a secure, multi-user environment for concurrent access to data that is loaded into memory. The server can take advantage of a distributed computing environment by distributing data and the workload among multiple machines and performing massively parallel processing. The server can also be deployed on a single machine where the workload and data volumes do not demand a distributed computing environment.

The server handles both big data and smaller sets of data, and it is designed with a high-performance, multi-threaded, analytic code. The server processes client requests at extraordinarily high speeds due to the combination of hardware and software that is designed for rapid access to tables in memory. By loading tables into memory for analytic processing, the SAS LASR Analytic Server enables business analysts to explore data and discover relationships in data at the speed of RAM.

The architecture was originally designed for optimal performance in a distributed computing environment. A distributed SAS LASR Analytic Server runs on multiple machines. A typical distributed configuration is to use a series of blades as a cluster. Each blade contains both local storage and large amounts of memory. In this analytic environment, many gigabytes of RAM per blade is common. Local storage is used to store large data sets in distributed form. Data is loaded into memory and made available so that clients can quickly access that data.

For distributed deployments, having local storage available on machines is critical in order to store large data sets in a distributed form. The SAS LASR Analytic Server supports the Hadoop Distributed File System (HDFS) as a co-located data provider. HDFS is used because the server can read from and write to HDFS in parallel. In addition, HDFS provides replication for data redundancy. HDFS stores data as blocks in distributed form on the blades and the replication provides failover capabilities.

In a distributed deployment, the server also supports some third-party vendor databases as co-located data providers. Teradata Data Warehouse Appliance and Greenplum Data Computing Appliance are massively parallel processing database appliances. You can install the SAS LASR Analytic Server software on each of the machines in either appliance. The server can read in parallel from the local data on each machine.

For the SAS LASR Analytic Server 1.6 release (concurrent with the SAS Visual Analytics 6.1 release) the server supports a non-distributed deployment. A non-distributed SAS LASR Analytic Server can perform the same in-memory analytic operations as a distributed deployment server. A non-distributed deployment does not support SAS High-Performance Deployment of Hadoop or third-party vendor appliances as co-located data providers.

How Does the SAS LASR Analytic Server Work?

Distributed SAS LASR Analytic Server

The SAS LASR Analytic Server provides a client/server environment where the client connects to the server, sends requests to the server, and receives results back from the server. The server-side environment is a distributed computing environment. A typical deployment is to use a series of blades in a cluster. In addition to using a homogeneous

hardware profile, the software installation is also homogeneous. The same operating system is used throughout and the same SAS software is installed on each blade that is used for the server. In order for the software on each blade to share the workload and still act as a single server, the SAS software that is installed on each blade implements the Message Passing Interface (MPI). The MPI implementation is used to enable communication between the blades.

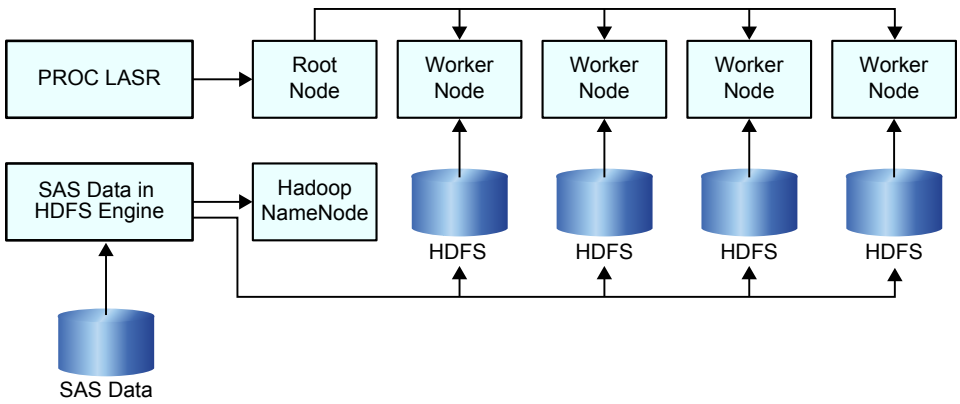
After a client connection is authenticated, the server performs the operations requested by the client. Any request (for example, a request for summary statistics) that is authorized will execute. After the server completes the request, there is no trace of the request. Every client request is executed in parallel at extraordinarily high speeds, and client communication with the server is practically instantaneous and seamless.

There are two ways to load data into a distributed server:

- **load data from tables and data sets.** You can start a server instance and directly load tables into the server by using the SAS LASR Analytic Server engine or the LASR procedure from a SAS session that has a network connection to the cluster. Any data source that can be accessed with a SAS engine can be loaded into memory. The data is transferred to the *root node* and the root node distributes the data to the *worker nodes*. You can also append rows to an in-memory table with the SAS LASR Analytic Server engine.
- **load tables from a co-located data provider.**
  - Tables can be read from the Hadoop Distributed File System (HDFS) that is provided by SAS High-Performance Deployment of Hadoop. You can use the SAS Data in HDFS engine to add tables to HDFS. When a table is added to HDFS, it is divided into blocks that are distributed across the machines in the cluster. The server software is designed to read data in parallel from HDFS. When used to read data from HDFS, the LASR procedure causes the worker nodes to read the blocks of data that are local to the machine.
  - Tables can also be read from a third-party vendor database. For distributed databases like Teradata and Greenplum, the SAS LASR Analytic Server can access the local data on each machine that is used for the database.

The following figure shows the relationship of the root node, the worker nodes, and how they interact when working with large data sets in HDFS. As described in the previous list, the LASR procedure communicates with the root node and the root node directs the worker nodes to read data in parallel from HDFS. The figure also indicates how the SAS Data in HDFS engine is used to transfer data to HDFS.

Figure 1.1 *Relationship of PROC LASR and the SAS Data in HDFS Engine*



Note: The preceding figure shows the distributed architecture of SAS High-Performance Deployment of Hadoop. For deployments that use a third-party vendor database, the architecture is also distributed, but different procedures and software components are used for distributing and reading the data.

After the data is loaded into memory on the server, it resides in memory until the table is unloaded or the server terminates. After the table is in memory, client applications that are authorized to access the table can send requests to the server and receive the results from the server.

In memory tables can be saved. You can use the SAS LASR Analytic Server engine to save an in-memory table as a SAS data set or as any other output that a SAS engine can use. For large tables, saving to HDFS is supported with the LASR procedure and the SAS Data in HDFS engine.

Non-Distributed SAS LASR Analytic Server

Most of the features that are available with a distributed deployment also apply to the non-distributed deployment too. Any limitations are related to the reduced functionality of using a single-machine rather than a distributed computing environment.

In a non-distributed deployment, the server acts in a client/server fashion where the client sends requests to the server and receives results back. The server performs the analytic operations on the tables that are loaded in to memory. As a result, the processing times are very fast and the results are delivered almost instantaneously.

You can load tables to a non-distributed server with the SAS LASR Analytic Server engine. Any data source that SAS can access can be used for input and the SAS LASR Analytic Server engine can store the data as an in-memory table. The engine also supports appending data.

You can save in-memory tables by using the SAS LASR Analytic Server engine. The tables can be saved as a SAS data set or as any other output that a SAS engine can use.

Benefits to Using the Hadoop Distributed File System

Loading data from disk to memory is efficient when the SAS LASR Analytic Server is co-located with a distributed data provider. The Hadoop Distributed File System (HDFS) provided by SAS High-Performance Deployment of Hadoop acts as a co-located data provider. HDFS offers some key benefits:

- **Parallel I/O.** The SAS LASR Analytic Server can read data in parallel at very impressive rates from a co-located data provider.
- **Data redundancy.** By default, two copies of the data are stored in HDFS. If a machine in the cluster becomes unavailable or fails, the SAS LASR Analytic Server instance on another machine in the cluster retrieves the data from a redundant block and loads the data into memory.
- **Homogeneous block distribution.** HDFS stores files in blocks. The SAS implementation enables a homogeneous block distribution that results in balanced memory utilization across the SAS LASR Analytic Server and reduces execution time.

Components of the SAS LASR Analytic Server

About the Components

The following sections identify some software components and interactions for SAS LASR Analytic Server.

Root Node

When the SAS client initiates contact with the *grid host* to start a SAS LASR Analytic Server instance, the SAS software on that machine takes on the role of distributing and coordinating the workload. This role is in contrast to a worker node. This term applies to a distributed SAS LASR Analytic Server only.

Worker Nodes

This is the role of the software that receives the workload from the root node. When a table is loaded into memory, the root node distributes the data to the worker nodes and they load the data into memory. If you are using a co-located data provider, each worker node reads the portion of the data that is local to the machine. The data is loaded into memory and requests that are sent to root node are distributed to the worker nodes. The worker nodes perform the analytic tasks on the data that is loaded in memory on the machine and then return the results to the root node. This term applies to a distributed SAS LASR Analytic Server only.

In-Memory Tables

SAS LASR Analytic Server performs analytics on tables that are in-memory only. Typically, large tables are read from a co-located data provider by worker nodes. The tables are loaded quickly because each worker node is able read a portion of the data from local storage. Once the portion of the table is in memory on each worker node, the server instance is able to perform the analytic operations that are requested by the client. The analytic tasks that are performed by the worker nodes are done on the in-memory data only.

Signature Files

SAS LASR Analytic Server uses two types of signature files, server signature files and table signature files. These files are used as a security mechanism for server management and for access to data in a server. When a server instance is started, a directory is specified on the PATH= option to the LASR procedure. The specified directory must exist on the machine that is specified as GRIDHOST= environment variable.

In order to start a server, the user must have Write access to the directory in order to be able to create the server signature file. In order stop a server, the user must have Read access to the server signature file so that it can be removed from the directory.

In order to load and unload tables on a server, the user must have Read access to the server signature file in order to interact with the server. Write permission to the directory

is needed to create the table signature file when loading a table and to delete the table signature file when unloading the table.

Server Description Files

Note: Most administrators prefer to use the PORT= option in the LASR procedure rather than use server description files.

If you specify a filename in the CREATE= option in the LASR procedure, then you start a SAS LASR Analytic Server instance, the LASR procedure creates two files:

- a server description file
- a server signature file (described in the previous section)

The server description file contains information such as the host names of the machines that are used by the server instance and signature file information.

In the LASR procedure, the server description file is specified with the CREATE= option. The server description file is created on the SAS client machine that invoked PROC LASR.

Administering the SAS LASR Analytic Server

Administering a Distributed Server

Basic administration of distributed SAS LASR Analytic Server can be performed with the LASR procedure from a SAS session. Server instances are started and stopped with the LASR procedure. The LASR procedure can be used to load and unload tables from memory though the SAS LASR Analytic Server engine also provides that ability. It is also possible to use a DETAILS= option with the LASR procedure to retrieve information about the server instance and tables that are resident in memory.

The SAS Data in HDFS engine is used to add and delete tables from the Hadoop Distributed File System (HDFS). You can use the DATASETS procedure with the engine to display information about tables that are stored in HDFS.

The HPDS2 procedure has a specific purpose for use with SAS LASR Analytic Server. In this deployment, the procedure is used to distribute data to the machines in an appliance. After the data are distributed, the SAS LASR Analytic Server can read the data in parallel from each of the machines in the appliance.

The LASR and HPDS2 procedures are described in this document. The SAS Data in HDFS engine is also described in this document.

Administering a Non-Distributed Server

A non-distributed SAS LASR Analytic Server runs on a single machine. A non-distributed server is started and stopped with the SAS LASR Analytic Server engine. A server is started with the STARTSERVER= option in the LIBNAME statement. The server is stopped when one of the following occurs:

- The libref is cleared (for example, **`libname lasrsvr clear;`**).
- The SAS program and session that started the server ends. You can use the SERVERWAIT statement in the VASMP procedure to keep the SAS program (and the server) running.

- The server receives a termination request from the SERVERTERM statement in the VASMP procedure.

A non-distributed deployment does not include a distributed computing environment. As a result, a non-distributed server does not support a co-located data provider. Tables are loaded and unloaded from memory with the SAS LASR Analytic Server engine only.

Common Administration Features

As described in the previous sections, the different architecture for distributed and non-distributed servers requires different methods for starting, stopping, and managing tables with servers. However, the VASMP procedure works with distributed and non-distributed servers to provide administrators with information about server instances. The statements that provide information that can be of interest to administrators are as follows:

- SERVERINFO
- TABLEINFO

Administrators might also be interested in the SERVERPARM statement. You can use this statement to adjust the number of requests that are processed concurrently. You might reduce the number of concurrent requests if the number of concurrent users causes the server to consume too many sockets from the operating system.

Features Available in SAS Visual Analytics Administrator

The SAS Visual Analytics Administrator is a web application that provides an intuitive graphical interface for server management. You can use the application to start and stop server instances, as well as load and unload tables from the servers. Once a server is started, you can view information about libraries and tables that are associated with the server. The application also indicates whether a table is in-memory or whether it is unloaded.

For deployments that use SAS High-Performance Deployment of Hadoop, an HDFS content explorer enables you to browse the tables that are stored in HDFS. The content explorer also enables adding tables to HDFS from registered tables. Once tables are stored in HDFS, you can to load them into memory in a server instance. Because SAS uses a special file format for the data that is stored in HDFS, the HDFS content explorer also provides information about the columns, row count, and block distribution.

Understanding Server Run Time

By default, servers are started and run indefinitely. However, in order to conserve the hardware resources in a distributed computing environment, server instances can be configured to exit after a period of inactivity. This feature applies to distributed SAS LASR Analytic Server only. You specify the inactivity duration with the LIFETIME= option when you start the server.

When the LIFETIME= option is used, each time a server is accessed, such as to view data or perform an analysis, the run time for the server is reset to zero. Each second that a server is unused, the run timer increments to count the number of inactive seconds. If the run timer reaches the maximum run time, the server exits. All the previously used hardware resources become available to the remaining server instances.

Distributing Data

SAS High-Performance Deployment of Hadoop

SAS provides SAS High-Performance Deployment of Hadoop as a co-located data provider. The SAS LASR Analytic Server software and the SAS High-Performance Deployment of Hadoop software are installed on the same blades in the cluster. The SAS Data in HDFS engine can be used to distribute data to HDFS.

For more information, see "Using the SAS Data in HDFS Engine" on page 81.

PROC HPDS2 for Big Data

For deployments that use Greenplum or Teradata, the HPDS2 procedure can be used to distribute large data sets to the machines in the appliance. The procedure provides an easy-to-use and efficient method for transferring large data sets.

For deployments that use Greenplum, the procedure is more efficient than using a DATA step with the SAS/ACCESS Interface to Greenplum and is an alternative to using the gpfdist utility.

The SAS/ACCESS Interface for the database must be configured on the client machine. It is important to distribute the data as evenly as possible so that the SAS LASR Analytic Server has an even workload when the data is read into memory.

For more information, see the Chapter 6, "HPDS2 Procedure," on page 55.

Bulkload for Teradata

The SAS/ACCESS Interface to Teradata supports a bulk loading feature. With this feature, a DATA step is as efficient at transferring data as the HPDS2 procedure.

The following code sample shows a LIBNAME statement and two DATA steps for adding tables to Teradata.

```
libname tdlib teradata
    server="dbc.example.com"
    database=hps
    user=dbuser
    password=dbpass
    bulkload=yes; 1

data tdlib.order_fact;
  set work.order_fact;
run;

data tdlib.product_dim (dbtype=(partno='int') 2
    dbcreate_table_opts='primary index(partno)'); 3
  set work.product_dim;
run;

data tdlib.salecode(dbtype=(_day='int' fpop='varchar(2)')
    bulkload=yes
    dbcreate_table_opts='primary index(_day,fpop)'); 4
    set work.salecode;
run;

data tdlib.automation(bulkload=yes
```

```
    dbcommit=1000000 5
    dbcreate_table_opts='unique primary index(obsnum)'); 6
    set automation;
    obsnum = _n_;
run;
```

1 Specify the BULKLOAD=YES option. This option is shown as a LIBNAME option but you can specify it as a data set option.

2 Specify a data type of **int** for the variable named partno.

3 Specify to use the variable named partno as the distribution key for the table.

4 Specify to use the variables that are named _day and fpop as a distribution key for the table that is named salecode.

5 Specify the DBCOMMIT= option when you are loading many rows. This option interacts with the BULKLOAD= option to perform checkpointing. Checkpointing provides known synchronization points if a failure occurs during the loading process.

6 Specify the **UNIQUE** keyword in the table options to indicate that the primary key is unique. This keyword can improve table loading performance.

Smaller Data Sets

You can use a DATA step to add smaller data sets to Greenplum or Teradata. Transferring small data sets does not need to be especially efficient. The SAS/ACCESS Interface for the database must be configured on the client machine.

The following code sample shows a LIBNAME statement and DATA steps for adding tables to Greenplum.

```
libname gplib greenplm server="grid001.example.com"
    database=hps
    schema=public
    user=dbuser
    password=dbpass;

data gplib.automation(distributed_by='distributed randomly'); 1
    set work.automation;
run;

data gplib.results(dbtype=(rep='int') 2
    distributed_by='distributed by (rep)') 3;
    set work.results;
run;

data gplib.salecode(dbtype=(day='int' fpop='varchar(2)') 4
    distributed_by='distributed by day,fpop'); 5
    set work.salecode;
run;
```

1 Specify a random distribution of the data. This data set option is for the SAS/ACCESS Interface to Greenplum.

2 Specify a data type of **int** for the variable named rep.

3 Specify to use the variable named rep as the distribution key for the table that is named results.

4 Specify a data type of **int** for the variable named day and a data type of **varchar(2)** for the variable named fpop.

5 Specify to use the combination of variables day and fpop as the distribution key for the table that is named salecode.

The following code sample shows a LIBNAME statement and a DATA step for adding a table to Teradata.

```
libname tdlib teradata server="dbc.example.com"
    database=hps
    user=dbuser
    password=dbpass;

data tdlib.parts_dim;
    set work.parts_dim;
run;
```

For Teradata, the SAS statements are very similar to the syntax for bulk loading. For more information, see "Bulkload for Teradata" on page 8.

See Also

SAS/ACCESS for Relational Databases: Reference

Memory Management

About Physical and Virtual Memory

The amount of memory on a machine is the physical memory. The amount of memory that can be used by an application can be larger, because the operating system can provide virtual memory. Virtual memory makes the machine appear to have more memory available than there actually is, by sharing physical memory between applications when they need it and by using disk space as memory.

When memory is not used and other applications need to allocate memory, the operating system pages out the memory that is not currently needed to support the other applications. When the paged-out memory is needed again, some other memory needs to be paged out. Paging means to write some of the contents of memory onto a disk.

Paging does affect performance, but some amount of paging is acceptable. Using virtual memory enables you to access tables that exceed the amount of physical memory on the machine. So long as the time to write pages to the disk and read them from the disk is short, the server performance is good.

One advantage of SASHDAT tables that are read from HDFS is that the server performs the most efficient paging of memory.

How Does the Server Use Memory for Tables?

When you load a table to memory with the SAS LASR Analytic Server engine, the server allocates physical memory to store the rows of data. This applies to both distributed and non-distributed servers.

When a distributed server loads a table from HDFS to memory with the LASR procedure, the server defers reading the rows of data into physical memory. You can direct the server to perform an aggressive memory allocation scheme at load time with the READAHEAD option for the PROC LASR statement.

Note: When a distributed server loads a table from either the Greenplum Data Computing Appliance or the Teradata Data Warehouse Appliance, physical memory is allocated for the rows of data. This is true even when the data provider is co-located.

How Else Does the Server Use Memory?

Physical memory is used when the server performs analytic operations such as summarizing a table. The amount of memory that a particular operation requires typically depends on the cardinality of the data. In most cases, the cardinality of the data is not known until the analysis is requested. When the server performs in-memory analytics, the following characteristics affect the amount of physical memory that is used:

- Operations that use group-by variables can use more memory than operations that do not. The amount of memory that is required is not known without knowing the number of group-by variable combinations that are in the data.
- The memory utilization pattern on the worker nodes can change drastically depending on the distribution of the data across the worker nodes. The distribution of the data affects the size of intermediate result sets that are merged across the network.

Some requests, especially with high-cardinality variables, can generate large result sets. To enable interactive near-real-time work with high cardinality problems, the server allocates memory for data structures that speed performance. The following list identifies some of these uses:

- The performance for traversing and querying a decision tree is best when the tree is stored in the server.
- Paging through group-by results when you have a million groups is best done by storing the group-by structure in a temporary table in the server. The temporary table is then used to look up groups for the next page of results to deliver to the client.

Managing Memory

The following list identifies some of the options that SAS provides for managing memory:

- You can use the TABLEMEM= option to specify a threshold for physical memory utilization.
- You can use the EXERNALMEM= option to specify a threshold for memory utilization for SAS High-Performance Analytics procedures.

By default, whenever the amount of physical memory in use rises above 75% of the total memory available on a node of a distributed server, loading tables, adding tables (including temporary ones), appending rows, or any other operation that consumes memory for storing data fails. You can specify the threshold when you start a server with the TABLEMEM= option in the PROC LASR statement or alter it for a running server with the SERVERPARM statement in the VASMP procedure. By default, TABLEMEM=75 (%).

Note: The memory that is consumed by tables loaded from HDFS do not count toward the TABLEMEM= limit.

Be aware that the TABLEMEM= option does not specify the percentage of memory that can be filled with tables. The memory consumption is measured across all processes of a machine.

A separate memory setting can be applied to processes that extract data from a server on a worker node. SAS High-Performance Analytics procedures can do this. If you set the EXTERNALMEM= option in the PROC LASR statement or through the SERVERPARM statement in the VASMP procedure, then you are specifying the threshold of total memory (expressed as a percentage) at which the server stops sending data to the high-performance analytics procedure.

See Also

- "TABLEMEM=*pct*" on page 26
- "EXTERNALMEM=*pct*" on page 23

Data Partitioning and Ordering

Overview of Partitioning

By default, partitioning is not used and data are distributed in a round-robin algorithm. This applies to SAS Data in HDFS engine as well as SAS LASR Analytic Server. In general, this works well so that each machine in a distributed server has an even workload.

However, there are some data access patterns that can take advantage of partitioning. When a table is partitioned in a distributed server, all of the rows that match the partition key are on a single machine. If the data access pattern matches the partitioning (for example, analyzing data by Customer_ID partitioning the data by Customer_ID), then the server can direct the work to just the one machine. This can speed up analytic processing because the server knows where the data are.

However, if the data access pattern does not match the partitioning, processing times might slow. This might be due to the uneven distribution of data that can cause the server to wait on the most heavily loaded machine.

Note: You can partition tables in non-distributed SAS LASR Analytic Server. However, all the partitions are kept on the single machine because there is no distributed computing environment.

Understanding Partition Keys

Partition keys in SASHDAT files and in-memory tables are constructed based on the formatted values of the partition variables. The formatted values are derived using internationalization and localization rules. (All formatted values in the server follow the internationalization and localization rules.)

All observations that compare equal in the (concatenated) formatted key belong to the same partition. This enables you to partition based on numeric variables. For example, you can partition based on binning formats or date and time variables use date and time formats.

A multi-variable partition still has a single value for the key. If you partition according to three variables, the server constructs a single character key based on the three

variables. The formatted values of the three variables appear in the order in which the variables were specified in the PARTITION= data set option. For example, partitioning a table by the character variable REGION and the numeric variable DATE, where DATE is formatted with a MONNAME3. format:

```
data hdfslib.sales(partition=(region date) replace=yes);
   format date monname3.;
   set work.sales;
run;
```

The partition keys might resemble EastJan, NorthJan, NorthFeb, WestMar, and so on. It is important to remember that partition keys are created only for the variable combinations that occur in the data. It is also important to understand that the partition key is not a sorting of Date (formatted as MONNAME3.) within Region. For information about ordering, see "Ordering within Partitions" on page 13.

If the formats for the partition keys are user-defined, they are transferred to the LASR Analytic Server when the table is loaded to memory. Be aware that if you use user-defined formats to partition a SASHDAT file, the definition of the user-defined format is not stored in the SASHDAT file. Only the name of the user-defined format is stored in the SASHDAT file. When you load the SASHDAT file to a server, you need to provide the XML definition of the user-defined format to the server. You can do this with the FMTLIBXML= option to the LASR procedure at server start-up or with the PROC LASR ADD request.

Ordering within Partitions

Ordering of records within a partition is implemented in the SAS Data in HDFS engine and the SAS LASR Analytic Server. You can order within a partition by one or more variables and the organization is hierarchical—that is ordering by A and B implies that the levels of A vary slower than those of B (B is ordered within A).

Ordering requires partitioning. The sort order of character variables uses national language collation and is sensitive to locale. The ordering is based on the raw values of the order-by variables. This is in contrast to the formation of partition keys, which is based on formatted values.

When a table that is partitioned and ordered in HDFS is loaded into memory on the server, the partitioning and ordering is maintained. You can append to in-memory tables that are partitioned and ordered. This does, however, require a re-ordering of the observations after the observations are transferred to the server.

SAS LASR Analytic Server Logging

Understanding Logging

Logging is an optional feature that can be enabled when a server instance is started with the LASR procedure. In order to conserve disk space, the default behavior for the server is to delete log files when the server exits. You can override this behavior with the KEEPLOG suboption to the LOGGING option when you start the server. You can also override this behavior with a suboption to the STOP option when you stop the server.

The server writes logs files on the grid host machine. The default directory for log files is **`/tmp`**. You can specify a different directory in the LOGGING option when you start

the server instance. The log filename is the same as server signature file with a .log suffix (for example, LASR.924998214.28622.saslasr.log).

See Also

- LOGGING option for the LASR procedure on page 24
- "Example 2: Starting a Server with Logging Options" on page 31
- "Starting and Stopping Non-Distributed Servers" on page 17

What is Logged?

When a server is started with the LOGGING option, the server opens the log file immediately, but does not generate a log record to indicate that the server started. As clients like SAS Visual Analytics Explorer make requests to the server for data, the server writes a log record.

The server writes a log record when a request is received and completed by the server. The server does not write log records for activities that do not contact the server (for example, ending the SAS session).

A user that is configured with passwordless SSH to access the machines in the cluster, but who is not authorized to use a server instance is denied access. The denial is logged with the message **`You do not have sufficient authorization to add tables to this LASR Analytic Server`**. However, if a user is not configured correctly to access the machines in the cluster, communication with the server is prevented by the operating system. The request does not reach the server. In this second case, the server does not write a log record because the server does not receive the request.

Log Record Format

The following file content shows an example of three log records. Line breaks are added for readability. Each record is written on a single line and fields are separated by commas. Each field is a name-value pair.

File 1.1 *Sample Log File Records*

```
ID=1,PID=28622,SASTime=1658782485.36,Time=Tue Jul 24 20:54:45 2012,User=sasdemo,
Host=grid001,LASRServer=/tmp/LASR.924998214.28622.saslasr,Port=56925,
RawCmd=action=ClassLevels name=DEPT.GRP1.PRDSALE "NlsenCoding=62",
ExeCmd=action=ClassLevels name=DEPT.GRP1.PRDSALE "NlsenCoding=62",JnlMsg=,
StatusMsg=Command successfully completed.,RunTime=        2.17

ID=2,PID=28622,SASTime=1658782593.09,Time=Tue Jul 24 20:56:33 2012,User=sasdemo,
Host=grid001,LASRServer=/tmp/LASR.924998214.28622.saslasr,Port=56925,
RawCmd=action=BoxPlot name=DEPT.GRP1.PRDSALE,
ExeCmd=action=BoxPlot name=DEPT.GRP1.PRDSALE,JnlMsg=,
StatusMsg=Command successfully completed.,RunTime=        0.12

ID=3,PID=28622,SASTime=1658825361.76,Time=Wed Jul 25 08:49:21 2012,User=sasdemo,
Host=grid001,LASRServer=/tmp/LASR.924998214.28622.saslasr,Port=56925,
RawCmd=action=APPEND_TABLE     ,ExeCmd=action=APPEND_TABLE     ,JnlMsg=,
StatusMsg=Command successfully completed.,RunTime=        0.09
```

Table 1.1 Log Record Fields

Field Name	Description
ID	specifies a unique identifier for the action.
PID	specifies the operating system process identifier for the server.
SASTime	specifies the local time of execution in SAS datetime format.
Time	specifies the local time of execution as a date and time string.
User	specifies the user ID that started the server.
Host	specifies the host name of the grid host machine.
LASRServer	specifies the server signature file.
Port	specifies the network port number on which the server listens.
RawCmd	specifies the request that is received by the server.
ExeCmd	specifies the command that the server executes. This value can include default substitutions or adjustments to the RawCmd (for example, completion of variable lists).
JnlMsg	specifies an error message that is buffered in a journal object.
StatusMsg	specifies the status completion message.
RunTime	specifies the processing duration (in seconds).

The server uses a journal object to buffer messages that can be localized. The format for the JnlMsg value is **`n-m:text`**.

n
is an integer that specifies the message is the **n**th in the journal.

m
is an integer that specifies the message severity.

text
is a text string that specifies the error.

Sample JnlMsg Values

```
JnlMsg=1-4:ERROR: The variable c1 in table WORK.EMPTY must be
numeric for this analysis.
```

```
JnlMsg=2-4:ERROR: You do not have sufficient authorization to add
tables to this LASR Analytic Server.
```

Chapter 2

Non-Distributed SAS LASR Analytic Server

About Non-Distributed SAS LASR Analytic Server

In a non-distributed deployment, the SAS LASR Analytic Server runs on a single machine. All of the in-memory analytic features that are available for the distributed deployment are also available for the non-distributed server.

One key difference has to do with reading and writing data. Because the server does not use a distributed computing environment, the server cannot be co-located with a data provider. The server does not read data in parallel and does not write SASHDAT files to HDFS.

Starting and Stopping Non-Distributed Servers

Starting Servers

Non-distributed servers are started and stopped with the SAS LASR Analytic Server engine. Starting a server requires the STARTSERVER= LIBNAME option.

To start a server:

Example Code 2.1 *Starting a Non-Distributed Server*

```
libname server1 sasiola
    startserver=( 2
        path="c:\temp"
        keeplog maxlogsize=20 2
    )
    host=localhost 3
    port=10010 4
    tag='hps';
```

1 The STARTSERVER option indicates to start a server. For information about the options, see "STARTSERVER =(*non-distributed-server-options*)" on page 70.

2 The KEEPLOG option implies the LOGGING option and prevents the server from removing the log file when the server exits. The MAXLOGSIZE= option specifies to use up to 20 MB for the log file before the file is rolled over.

3 The HOST= specification is optional.

4 If you do not specify a PORT= value, then the server starts on a random port and sets the LASRPORT macro variable to the network port number.

Submitting the previous LIBNAME statement from a SAS session starts a server and the server remains running as long as the SAS session remains running. In a batch environment where you want to start a server for client/server use by other users, follow the LIBNAME statement with the following VASMP procedure statements:

Example Code 2.2 *SERVERWAIT Statement for the VASMP Procedure*

```
proc vasmp;
    serverwait port=10010; 1
quit;
```

1 The SERVERWAIT statement causes the server to continue running and wait for a termination request.

When non-distributed SAS LASR Analytic Server is used in a metadata environment like SAS Visual Analytics, the SIGNER= option enables the server to enforce the permissions that are set in metadata. The values for the HOST= and PORT= options must match the host name and network port number that are specified for the server in metadata.

```
libname server1 sasiola startserver=(path="/tmp")
    host="server.example.com" port=10010 tag='hps'
    signer="http://server.example.com/SASLASRAuthorization";
```

For information about using SAS LASR Analytic Server in a metadata environment, see *SAS Visual Analytics: Administration Guide*.

Stopping Servers

Stopping a server is performed by clearing the libref that was used to start the server (if you start the server from a SAS session and keep the session running) or with the SERVERTERM statement.

To stop a server from the same SAS session that started it:

Example Code 2.3 *Stopping a Non-Distributed Server with the LIBNAME CLEAR Option*

```
libname server1 clear;
```

To stop a server from a different SAS session, use the SERVERTERM statement:

Example Code 2.4 *SERVERTERM Statement for the VASMP Procedure*

```
proc vasmp;
    serverterm host="server.example.com" port=10010;
quit;
```

Note: Exiting the SAS session that started the server also terminates the server because all librefs are automatically cleared at the end of a SAS session.

Loading and Unloading Tables for Non-Distributed Servers

Tables are loaded into memory in a non-distributed server with the SAS LASR Analytic Server engine. A DATA step can be used. The following example demonstrates loading the Prdsale table into memory after starting a server on port 10010.

To load a table to memory:

Example Code 2.5 *Loading a Table to Memory for Non-Distributed Servers*

```
libname server1 startserver port=10010 tag='hps';

data server1.prdsale;
    set sashelp.prdsale;
run;
```

You can unload a table from memory with the DATASETS procedure:

Example Code 2.6 *Unloading a Table with the DATASETS Procedure*

```
proc datasets lib=server1;
    delete prdsale;
quit;
```

Chapter 3

LASR Procedure

Overview: LASR Procedure

What Does the LASR Procedure Do?

The LASR procedure is used to start, stop, and load and unload tables from the SAS LASR Analytic Server. The LASR procedure can also be used to save in-memory tables to HDFS.

Data Sources

The LASR procedure can transfer data from any data source that SAS can read and load it into memory on the SAS LASR Analytic Server. However, the LASR procedure can also be used to make the server read data from a co-located data provider. The HDFS that is part of SAS High-Performance Deployment of Hadoop provides a co-located data provider. Some third-party vendor databases can also act as co-located data providers.

Two examples of third-party vendor databases are the Greenplum Data Computing Appliance (DCA) and Teradata Data Warehouse Appliance. When the data is co-located, each machine that is used by the server instance reads the portion of the data that is local. Because the read is local and because the machines read in parallel, very large tables are read quickly.

In order to use a third-party vendor database as a co-located data provider, the client machine must be configured with the native database client software and the SAS/ACCESS Interface software for the database. The database is identified in a LIBNAME statement. The LASR procedure then uses the SERVER= information from the LIBNAME statement and the host name information in the PERFORMANCE statement to determine whether the data is co-located. If the host information is the same, then the data is read in parallel.

Syntax: LASR Procedure

PROC LASR *server-options*;
PERFORMANCE *performance-options*;
REMOVE *table-specification*;
SAVE *table-specification* / *save-options*;

Statement	Task	Example
PROC LASR	Start a server.	Ex. 1
PROC LASR	Start a server with logging.	Ex. 2
PROC LASR	Using the SAS Data in HDFS engine.	Ex. 3
PROC LASR	Load a table from Teradata to memory	Ex. 4
PROC LASR	Load a table from Greenplum to memory	Ex. 5
REMOVE	Unload a table from memory.	Ex. 6
PROC LASR	Stop a server.	Ex. 7
PROC LASR	Working with user-defined formats.	Ex. 8
PROC LASR	Working with user-defined formats and the FMTLIBXML= option.	Ex. 9
SAVE	Save a table to HDFS.	Ex. 10

PROC LASR Statement

Controls the SAS LASR Analytic Server.

Syntax

PROC LASR *server-options*;

Server Options

These options control how the server starts, stops, and operates with data.

ADD
specifies to load a table to the SAS LASR Analytic Server. The data to load is identified by the DATA= option or the HDFS= option.

You can also add tables to memory with the SAS LASR Analytic Server engine. An important difference between using the LASR procedure and the engine is that the procedure has the ability to load data in parallel from a co-located data provider like SAS High-Performance Deployment of Hadoop.

CONCURRENT=*maximum-requests*
specifies the number of concurrent requests that can execute in the server. This option does not reject connections or requests that exceed *maximum-requests*. When *maximum-requests* is reached, the additional requests are queued and then processed in first-in-first-out order.

After the server is running, you can adjust this value in a SERVERPARM statement with the VASMP procedure.

Alias NACTIONS=

Default 20

CREATE <="*server-description-file*">
specifies to start a server. The optional *server-description-file* argument specifies the fully qualified path to a server description file. Enclose the value in quotation marks. The fully qualified path is limited to 200 characters. The server description file is assigned to the LASRLAST macro variable.

If you do not specify a server description file, then you can use the PORT= option to specify the network port number. In either case, the LASRPORT macro variable is updated with the network port number that the server uses for communication.

DATA=*libref.member-name*
specifies the data to load into the SAS LASR Analytic Server.

DETAILS= TABLES | ALL
specifies the information to return. Use TABLES to retrieve the table names, NLS encoding, row count, owner, and the table load time. The ALL value provides the previous information and adds the MPI rank and host name for each machine in the server.

The information always includes the performance information. This information includes the host name for the grid host, the grid installation location, and the number of machines in the server.

EXTERNALMEM=*pct*
specifies the percentage of memory that can be allocated before the server stops transferring data to external processes such as external actions and the SAS High-Performance Analytics procedures. If the percentage is exceeded, the server stops transferring data.

Default 75

FMTLIBXML
specifies the file reference for a format stream. For more information, see "Example 8: Working with User-Defined Formats" on page 35.

FORCE
specifies that a server should be started even if the server description file specified in the CREATE= option already exists. The procedure attempts to stop the server process that is described in the existing server description file and then the file is overwritten with the details for the new server.

Restriction Use this option with the CREATE= option only.

HDFS(*HDFS-options*)
specifies the parameters for the SASHDAT file to load from HDFS.

TIP Instead of specifying the HDFS option and parameters, you can use the ADD= option with a SAS Data in HDFS engine library.

FILE=
specifies the fully qualified path to the SASHDAT file. Enclose the value in quotation marks. The filename is converted to lowercase and the SASHDAT file in HDFS must be named in lowercase.

Alias PATH=

LABEL=
specifies the description to assign to the table. This value is used to override the label that was associated with the data set before it was stored in HDFS. If this option is not specified, then the label that was associated with the data set is used. Enclose the value in quotation marks.

DIRECT
specifies that the data is loaded directly from HDFS into memory. This option provides a significant performance improvement. With this option, the user account ID that is used to start the server process is used to create the table signature file.

Alias HADOOP=

LIFETIME=*maximum-runtime*<(*active-time*)>
specifies the duration of the server process, in seconds. If you do not specify this option, the server runs indefinitely.

maximum-runtime
When the *maximum-runtime* is specified without an *active-time* value, the server exits after *maximum-runtime* seconds.

active-time
When the *maximum-runtime* and *active-time* values are specified, the server runs for *maximum-runtime* seconds and then starts a run timer with an inactivity time-out of *active-time* seconds. When the server is contacted with a request, the run timer is reset to zero. Each second that the server is unused, the run timer increments to count the number of inactive seconds. If the run timer reaches the *active-time*, the server exits.

LOGGING <(*log-options*)>
The log file is named lasr.log.

CLF
: specifies to use the common log format for log files. This format is a standardized text file format that is frequently analyzed by web analysis software. Specifying this option implies the LOGGING option.

KEEPLOG
: specifies to keep the log files when the server exits instead of deleting them. By default, the log files are removed when the server exits.If you did not specify this option when the server was started, you can specify it as an option to the STOP option.

MAXFILESIZE=
: specifies the maximum log file size, in megabytes, for a log file. When the log file reaches the specified size, a new log file is created and named with a sequentially assigned index number (for example, `.log.1`). The default value is 100 megabytes.

  TIP Do not include an MB or M suffix when you specify the size.

MAXROLLNUM=
: specifies the maximum number of log files to create. When the maximum has been reached, the server begins to overwrite existing log files. The oldest log file is overwritten first. The default value is 10.

OSENCODING
: specifies that the log file is produced with the operating system encoding of the SAS LASR Analytic Server root node. This option is useful when the server is run in a different encoding than the operating system, but you want a log file that is readable in the server operating system.

PATH='*log-file-directory*'
: specifies the fully qualified path to the directory to use for server log files. The default value is `/tmp`.

MERGELIMIT=*n*
: specifies that when the number of unique values in a numeric GROUPBY variable exceeds *n*, the variable is automatically binned and the GROUPBY structure is determined based on the binned values of the variable, rather than the unique formatted values.

  For example, if you specify MERGELIMIT=500, any numeric GROUPBY variable with more than 500 unique formatted values is binned. Instead of returning results for more than 500 groups, the results are returned for the bins. You can specify the number of bins with the MERGEBINS= option.

NOCLASS
: specifies that all character variables are not to be treated implicitly as classification variables. Without this option, all character variables are implicitly treated as classification variables. The performance for loading tables is improved when this option is used.

  Interaction You must specify the NOCLASS option in order to use the APPEND data set option of the SAS LASR Analytic Server engine.

PATH="*signature-file-path*"
: specifies the directory to use for storing the server and table signature files. The specified directory must exist on the machine that is specified in the GRIDHOST= environment variable.

PORT=*integer*
specifies the network port number to use for communicating with the server. You can specify a port number with the CREATE option to start a server on the specified port.

Interaction Do not specify the PORT= option in the LASR procedure statement with a LASRSERVER= option in the PERFORMANCE statement.

READAHEAD
specifies for the server to be more aggressive in reading memory pages during the mapping phase when tables are loaded from HDFS. Loading the table takes more time with this option, but the first access of the table is faster.

Engine SAS Data in HDFS engine

SERVERPERMISSIONS=*mode*
specifies the permission setting for accessing the server instance. The mode value is specified as an integer value such as 755. The mode corresponds to the mode values that are used for UNIX file access permissions.

Alias SERVERPERM=

Range 600 to 777

Interaction You can use this option with the CREATE option when you start a server.

SIGNER="*authorization-web-service-uri*"
specifies the URI for the SAS LASR Authorization web service. The web service is provided by the SAS Visual Analytics software. For more information, see *SAS Visual Analytics: Administration Guide*.

Example SIGNER="https://server.example.com/SASLASRAuthorization"

STOP <(*stop-options*)>
terminates a SAS LASR Analytic Server. The server instance is specified in the LASRSERVER= option that identifies a server description file, or it is determined from the LASRLAST macro variable. Once the server instance receives a request to stop, the server does not accept new connections.

IMMEDIATE
specifies to stop the server without waiting for current requests to complete. Without this option, termination requests are queued and can be queued behind a long-running request.

Alias NOW

KEEPLOG
specifies to keep log files that are created with the LOGGING option.

Alias TERM

TABLEMEM=*pct*
specifies the percentage of memory that can be allocated before the server rejects requests to add tables or append data. If the percentage is exceeded, adding a table or appending rows to tables fails. These operations continue to fail until the percentage is reset or the memory usage on the server drops below the threshold.

This option has no effect for non-distributed servers. For non-distributed servers, the memory limits can be controlled with the MEMSIZE system option.

Note: The specified *pct* value does not specify the percentage of memory allocated to in-memory tables. It is the percentage of all memory used by the entire machine that—if exceeded—prevents further addition of data to the server. The memory used is not measured at the process or user level, it is computed for the entire machine. In other words, if operating system processes allocate a lot of memory, then loading tables into the server might fail. The threshold is not affected by memory that is associated with SASHDAT tables that are loaded from HDFS.

Alias MEMLOAD=

Default 75

TABLEPERMISSIONS=*mode*
specifies the permission setting for accessing a table. The mode value is specified as an integer value such as 755. The mode corresponds to the mode values that are used for UNIX file access permissions.

Alias TABLEPERM=

Range 600 to 777

Interaction You can use this option with the ADD option when you load a table to memory.

VERBOSE
specifies to request additional information about starting a server or connecting to a server in the SAS log. This information can be helpful to diagnose environment configuration issues.

Alias GRIDMSG

PERFORMANCE Statement

The PERFORMANCE statement defines performance parameters for multithreaded and distributed computing.

Examples: "Example 4: Load a Table from Teradata to Memory" on page 32
"Example 6: Unload a Table from Memory" on page 34

Syntax

PERFORMANCE *performance-options*;

Performance Statement Options

COMMIT=
specifies that periodic updates are written to the SAS log when observations are sent from the client to the server instance. Whenever the number of observations sent exceeds an integer multiple of the COMMIT= size, a message is written to the SAS

log. The message indicates the actual number of observations distributed and not an integer multiple of the COMMIT= size.

DATASERVER=
specifies the host to use for a database connection. This option is used in Teradata deployments so that the LASR procedure compares this host name with the host name that is specified in the SERVER= option in the LIBNAME statement. If you do not specify the DATASERVER= option, the host to use for the database connection is determined from the GRIDDATASERVER= environment variable.

HOST=
specifies the grid host to use for the server instance. Enclose the host name in quotation marks. If you do not specify the HOST= option, it is determined from the GRIDHOST= environment variable.

Alias GRIDHOST=

INSTALL=
specifies the path to the TKGrid software on the grid host. If you do not specify this option, it is determined from the GRIDINSTALLLOC= environment variable.

Alias INSTALLOC=

LASRSERVER=
specifies the server to use. Provide the fully qualified path to the server description file.

Alias LASR=

NODES=
specifies the number of machines in the cluster to use for the server instance. Specify ALL to calculate the number automatically.

Alias NNODES=

Restriction This option has no effect when you use a third-party vendor database as a co-located data provider and you specify the CREATE= and DATA= options in the PROC LASR statement. When you use a third-party vendor database as a co-located data provider, you must use all of the machines to read data from the database.

NTHREADS=
specifies the number of threads for analytic computations and overrides the SAS system option THREADS | NOTHREADS. By default, the server uses one thread for each CPU core that is available on each machine in the cluster. Use this option to throttle the number of CPU cores that are used on each machine.

The maximum number of concurrent threads is controlled by the SAS software license.

Note: The SAS system options THREADS | NOTHREADS apply to the client machine that issues the PROC LASR statement. They do not apply to the machines in the cluster.

TIMEOUT=
specifies the time in seconds for the LASR procedure to wait for a connection to the grid host and establish a connection back to the client. The default value is 120 seconds. If jobs are submitted through workload management tools that might suspend access to the grid host for a longer period, you might want to increase the value.

REMOVE Statement

The REMOVE statement is used to unload a table from memory.

Syntax

REMOVE *table-specification*;

Required Argument

table-specification
specifies the table to unload from memory. For a table that was loaded from a SAS library, the table specification is the same libref.*member-name* that was used to load the table. For a table that was loaded from HDFS, the table specification is the same as the HDFS path to the table, but is delimited with periods (.) instead of slashes (/). For a table that was loaded from the / directory in HDFS, the table specification is HADOOP.*TABLENAME*.

SAVE Statement

The SAVE statement is used to save an in-memory table to HDFS.

Syntax

SAVE *table-specification* / *save-options*;

Required Arguments

table-specification
specifies the table that is in memory. For a table that was loaded from a SAS library with the procedure, the table specification is the same libref.*member-name* that was used to load the table. For a table that was loaded from HDFS, the table specification is the same as the HDFS path to the table, but is delimited with periods (.) instead of slashes (/). For a table that was loaded from the / directory in HDFS, the table specification is HADOOP.*TABLENAME*.

save-options
specifies the options for saving the file in HDFS.

BLOCKSIZE=
specifies the block size to use for distributing the data set. Suffix values are B (bytes), K (kilobytes), M (megabytes), and G (gigabytes). The default block size is 32M.

Alias BLOCK=

COPIES=*n*
specifies the number of replications to make for the data set (beyond the original blocks). The default value is 1.

FULLPATH
specifies that the value for the PATH= option specifies the full path for the file, including the filename.

PATH='*HDFS-path*'
specifies the directory in HDFS in which to store the SASHDAT file. The value is case sensitive. The filename for the SASHDAT file that is stored in the path is always lowercase.

Note: If the PATH= option is not specified, the server attempts to save the table in the **`/user/userid`** directory. The *userid* is the user ID that started the server instance.

REPLACE
specifies that the SASHDAT file should be overwritten if it already exists.

Examples: LASR Procedure

Example 1: Start a Server

Details

This PROC LASR example demonstrates starting a server instance on network port number 10010. Once the server instance is started, the LASRPORT macro variable in the SAS session is set.

Program

```
option set=GRIDHOST="grid001.example.com"; 1
option set=GRIDINSTALLLOC="/opt/TKGrid";

proc lasr create port=10010 2
    path="/tmp" noclass;

    performance nodes=all;
run;
```

Program Description

1. The GRIDHOST= and GRIDINSTALLLOC= environment variables are used to identify the machine to connect to and the location of the SAS High-Performance Analytics components.
2. The CREATE option is required and the PORT= option specifies the network port number to use.

Example 2: Starting a Server with Logging Options

Details

This PROC LASR example demonstrates how to start a server instance and specify logging options.

Program

```
option set=GRIDHOST="grid001.example.com";
option set=GRIDINSTALLLOC="/opt/TKGrid";

proc lasr
    create port=10010
    path="/tmp"
    noclass
    logging(path="/opt/logs" maxfilesize=5 keeplog clf);

    performance nodes=all;
run;
```

Program Description

The logging statement modifies the default logging behavior. Log files are written to **/opt/logs** instead of the default directory, **/tmp**. The log files are rolled over when they reach five megabytes. The KEEPLOG option is used to keep the log files when the server exits rather than delete them.

Example 3: Using the SAS Data in HDFS Engine

Details

The LASR procedure can load tables to memory from HDFS with the SAS Data in HDFS engine. This use is similar to using the HDFS option with the procedure, but has the advantage that you can FORMAT statements and data set options.

Program

```
option set=GRIDHOST="grid001.example.com"; 1
option set=GRIDINSTALLLOC="/opt/TKGrid";

libname grp1 sashdat path="/dept/grp1"; 2

proc lasr create port=10010 noclass;
    performance nodes=all;
run;

proc lasr add data=grp1.sales2012 port=10010;
    format predict $dollar20. 3
           actual $dollar20.;
```

```
run;
```

Program Description

1. The GRIDHOST= and GRIDINSTALLLOC= environment variables are used by the LASR procedure and the GRIDHOST= option is also used by the LIBNAME statement.
2. The SAS Data in HDFS engine uses the GRIDHOST= environment variable to determine the host name for the NameNode. The PATH= option is used to specify the directory in HDFS.
3. The FORMAT statement is used to override the format name in HDFS for the variable.

If the table in HDFS has variables that are associated with user-defined formats, then you must have the user-defined formats available in the format catalog search order.

Example 4: Load a Table from Teradata to Memory

Details

This PROC LASR example demonstrates how to load a table to memory from Teradata. The native database client for Teradata and SAS/ACCESS Interface to Teradata must be installed and configured on the client machine.

```
libname tdlib teradata server="dbccop1.example.com" 1
    database=hps user=dbc password=dbcpass;

proc lasr create port=10010
    data=tdlib.sometable
    path="/tmp";

    performance host="tms.example.com" 2
        install="/opt/TKGrid"
        dataserver="dbccop1.example.com"; 3

run;

proc lasr add data=tdlib.tabletwo (label = "Table description") 4
    port=10010;
    format revenue dollar20.2
           units comma9; 5
run;
```

Program Description

1. The SERVER= option in the LIBNAME statement specifies the host name for the Teradata database.
2. The HOST= option in the PERFORMANCE statement specifies the host name of the Teradata Management Server (TMS).
3. The DATASERVER= option in the PERFORMANCE statement specifies the same host name for the Teradata database that is used in the LIBNAME statement.

4. The input data set option, LABEL=, associates the description with the data in the server instance. This option causes a warning in the SAS log because the SAS/ACCESS Interface to Teradata does not support data set labels.
5. SAS formats are applied with the FORMAT statement. Specifying the variable formats is useful for DBMS tables because database systems do not store formats.

Example 5: Load a Table from Greenplum to Memory

Details

This PROC LASR example demonstrates how to load a table to memory from Greenplum. The ODBC drivers and SAS/ACCESS Interface to Greenplum must be installed and configured on the client machine.

```
libname gplib greenplm server="mdw.example.com" 1
    database=hps user=dbuser password=dbpass;

proc lasr create port=10010
    data=gplib.sometable
    path="/tmp";

    performance host="mdw.example.com" 2
        install = "/opt/TKGrid";

run;

proc lasr add data=gplib.tabletwo (label = "Table description") 3
    port=10010;
    format y x1-x15 5.4
           dt date9.; 4
run;
```

Program Description

1. The SERVER= option in the LIBNAME statement specifies the host name for the Greenplum database.
2. The HOST= option in the PERFORMANCE statement specifies the host name of the Greenplum master host.
3. The input data set option, LABEL=, associates the description with the data in the server instance. This option causes a warning in the SAS log because the SAS/ACCESS Interface to Greenplum does not support data set labels.
4. SAS formats are applied with the FORMAT statement. Specifying the variable formats is useful for DBMS tables because database systems do not store formats.

Example 6: Unload a Table from Memory

Details

This PROC LASR example demonstrates how to unload tables from memory. The first REMOVE statement applies to tables that were loaded from HDFS. The second REMOVE statement is typical for tables that are loaded from SAS libraries.

Program

```
libname finance "/data/finance/2011/";

proc lasr port=10010;
    remove user.sales.2011.q4; 1
    remove finance.trans; 2
    performance host="grid001.example.com"
        install="/opt/TKGrid";
run;
```

Program Description

1. This REMOVE statement specifies a table that was loaded from HDFS.
2. The libref and member name for a SAS data set are specified in this REMOVE statement example.

Example 7: Stopping a Server

Details

This PROC LASR example demonstrates stopping a server instance.

Program

```
option set=GRIDHOST="grid001.example.com";
option set=GRIDINSTALLLOC="/opt/TKGrid";

proc lasr term port=10010;
run;
```

Program Description

The server instance listening on port 10010 is stopped.

Example 8: Working with User-Defined Formats

Details

By default, when user-defined formats are used with the server, the LASR procedure automatically uses these formats. The formats must be available in the format catalog search order. You can use the FMTSEARCH= system option to specify the format catalog search order. The LASR procedure converts the formats to an XML representation and transfers them to the server with the data.

Program

```
proc format library=myfmts;
     value YesNo        1='Yes'        0='No';
     value checkThis    1='ThisisOne'  2='ThisisTwo';
     value $cityChar    1='Portage'    2='Kinston';
run;

options fmtsearch=(myfmts); 1

proc lasr add data=orsdm.profit_company_product_year port=10010;
    format city $cityChar.; 2

    performance host="grid001.example.com"
        install="/opt/TKGrid"
        nodes=ALL;
run;
```

Program Description

1. The user-defined formats are available to the LASR procedure because they are added to the format catalog search order.
2. When the $cityChar. format is applied to the city variable, the LASR procedure converts the formats to XML, and transfers the format information and the data to the server.

Example 9: Working with User-Defined Formats and the FMTLIBXML= Option

Details

As explained in the previous example, the LASR procedure can use any format so long as the format is in the format catalog search order. The procedure automatically converts the format information to XML and transfers it to the server with the data. However, if the same formats are used many times, it is more efficient to convert the formats to XML manually and use the FMTLIBXML= option.

You can use the FORMAT procedure to write formats to an XML fileref. Then, you can reference the fileref in the FMTLIBXML= option each time you use the LASR

procedure to load tables. This improves performance because the conversion to XML occurs once rather than each time LASR procedure transfers the data.

Formats are created with the FORMAT procedure. The following SAS statements show a simple example of creating a format and using the XML fileref in the LASR procedure.

Program

```
proc format library=gendrfmt;
     value $gender   'M'='Male'        'F'='Female';
run;

options fmtsearch=(gendrfmt); 1

filename fmtxml 'genderfmt.xml';
libname  fmtxml XML92 xmltype=sasfmt tagset=tagsets.XMLsuv;

proc format library=gendrfmt cntlout=fmtxml.allfmts; 2
run;

proc lasr add data=sashelp.class fmtlibxml=fmtxml; 3

    format sex $gender.; 4

    performance host="grid001.example.com"
        install="/opt/TKGrid"
        nodes=ALL;
run;
```

Program Description

1. The user-defined formats are available to the LASR procedure because they are added to the format catalog search order.
2. An XML stream for the formats in the file genderfmt.xml is associated with the file reference fmtxml. The formats are converted to XML and stored in the file.
3. The file reference fmtxml is used with the FMTLIBXML= option in the PROC LASR statement. For subsequent uses of the LASR procedure, using the FMTLIBXML= option to reference the fileref is efficient because the formats are already converted to XML.
4. The $gender. format information is transferred to the server in an XML stream and associated with the variable that is named sex. However, the format must be available to the SAS session that runs the LASR procedure.

Example 10: Saving a Table to HDFS

Details

The server can save in-memory tables to HDFS. Use the SAVE statement to provide a table specification and the save options.

```
option set=GRIDHOST="grid001.example.com";
option set=GRIDINSTALLLOC="/opt/TKGrid";
```

```
proc lasr port=10010;
    save sales.sales2012 / path="/dept/grp1/" copies=1 blocksize=32m; 1
    save sales.avg2012 / fullpath path="/dept/grp1/avg/y2012" copies=1; 2
run;
```

Program Description

1. The table that is named sales2012 is saved to HDFS as /dept/grp1/sales2012.sashdat.
2. The table that is named avg2012 is saved to HDFS as /dept/grp1/avg/y2012.sashdat. The FULLPATH option is used to rename the file.

Chapter 4

VASMP Procedure

Overview: VASMP Procedure

What Does the VASMP Procedure Do?

The VASMP procedure is used to list in-memory tables and perform administration of Non-distributed SAS LASR Analytic Server instances.

Syntax: VASMP Procedure

PROC VASMP <*options*>;
QUIT;
SERVERINFO <*option*>;
SERVERPARM <*option*>;
SERVERTERM <*options*>;
SERVERWAIT <*options*>;
TABLEINFO </ *options*>;

PROC VASMP Statement

in a SAS LASR Analytic Server instance.

Syntax

PROC VASMP <*options*>;

Optional Arguments

DATA=*libref.member-name*
specifies the table to access from memory. The libref must be assigned from a SAS LASR Analytic Server engine LIBNAME statement.

IMMEDIATE
specifies that the procedure executes one statement at a time rather than accumulating statements in RUN blocks.

Alias SINGLESTEP

NOPRINT
This option suppresses the generation of ODS tables and other printed output in the VASMP procedure.

NOTIMINGMSG
When an action completes successfully, the VASMP procedure generates a SAS log message that contains the execution time of the request. Specify this option to suppress the message.

Alias NOTIME

QUIT Statement

The QUIT statement is used to end the procedure execution. When the procedure reaches the QUIT statement, all resources allocated by the procedure are released. You can no longer execute procedure statements without invoking the procedure again. However, the connection to the server is not lost, because that connection was made through the SAS LASR Analytic Server engine. As a result, any subsequent invocation of the procedure that uses the same libref executes almost instantaneously because the engine is already connected to the server.

Interaction: Using a DATA step or another procedure step is equivalent to issuing a QUIT statement. If there is an error during the procedure execution, it is also equivalent to issuing a QUIT statement.

Syntax

QUIT;

SERVERINFO Statement

The SERVERINFO statement returns information about the SAS LASR Analytic Server.

Syntax

SERVERINFO </ *option*>;

SERVERINFO Statement Options

HOST="*host-name*"
specifies the host name for the SAS LASR Analytic Server. Use this option with the PORT= option.

NORANKS
specifies to omit the list of host names for the worker nodes. This option reduces the output of the SERVERINFO option considerably for large environments.

PORT=*number*
specifies the port number for the SAS LASR Analytic Server. If you do not specify a PORT= value, then behavior of the SERVERINFO statement depends on whether an in-memory table is active. If there is no active table, then the procedure attempts to connect to the server using the LASRPORT macro variable. If a table is active, the information is gathered for the server that is implied by the libref of the active table.

SERVERPARM Statement

The SERVERPARM statement enables you to change some global settings for the server if you have sufficient authorization. The user account that starts the server has privileges to modify server parameters.

Syntax

SERVERPARM <*options*>;

SERVERPARM Statement Options

CONCURRENT=*number*
specifies the number of concurrent requests that can execute in the server. Once the threshold is met, the requests are queued and then executed as the currently running requests complete.

Alias	NACTIONS=
Default	20

EXTERNALMEM=*pct*
specifies the percentage of memory that can be allocated before the server stops transferring data to external processes such as external actions and the SAS High-Performance Analytics procedures. If the percentage is exceeded, the server stops transferring data.

Default 75

HADOOPHOME="*path*"
specifies the path for the HADOOP_HOME environment variable. Changing this variable is useful for migrating SASHDAT files from one Hadoop installation to another.

Setting the HADOOP_HOME environment variable is a server-wide change. All requests, by all users, for reading files from HDFS and saving files, use the specified HADOOP_HOME. This can cause unexpected results if users are not aware of the change.

Note: If you are using this option to migrate SASHDAT files, then consider starting a server for that exclusive purpose.

Alias HADOOP=

HOST="*host-name*"
specifies the host name for the SAS LASR Analytic Server. Use this option with the PORT= option.

PORT=*number*
specifies the port number for the SAS LASR Analytic Server. If you do not specify a PORT= value, then behavior of the SERVERPARM statement depends on whether an in-memory table is active. If there is no active table, then the procedure attempts to connect to the server using the LASRPORT macro variable. If a table is active, the information is gathered for the server that is implied by the libref of the active table.

TABLEMEM=*pct*
specifies the percentage of memory that can be allocated before the server rejects requests to add tables or append data. If the percentage is exceeded, adding a table or appending rows to tables fails. These operations continue to fail until the percentage is reset or the memory usage on the server drops below the threshold.

This option has no effect for non-distributed servers. For non-distributed servers, the memory limits can be controlled with the MEMSIZE system option.

Note: The specified *pct* value does not specify the percentage of memory allocated to in-memory tables. It is the percentage of all memory used by the entire machine that—if exceeded—prevents further addition of data to the server. The memory used is not measured at the process or user level, it is computed for the entire machine. In other words, if operating system processes allocate a lot of memory, then loading tables into the server might fail. The threshold is not affected by memory that is associated with SASHDAT tables that are loaded from HDFS.

Alias MEMLOAD=

Default 75

SERVERTERM Statement

The SERVERTERM statement sends a termination request to the server that is identified through the statement options. You must have sufficient authorization for this request to succeed.

Syntax

SERVERTERM <*options*>;

SERVERTERM Statement Options

HOST="*host-name*"
specifies the host name for the SAS LASR Analytic Server. Use this option with the PORT= option.

PORT=*number*
specifies the port number for the SAS LASR Analytic Server.

SERVERWAIT Statement

The SERVERWAIT statement suspends execution of the VASMP procedure until the server that it uses receives a termination request. This is useful for starting a non-distributed server from a batch program. This statement suspends the SAS session in which it is executed until the server stops or until an interrupt signal is received.

Syntax

SERVERWAIT <*options*>;

SERVERWAIT Statement Options

HOST="*host-name*"
specifies the host name for the SAS LASR Analytic Server. Use this option with the PORT= option.

PORT=*number*
specifies the port number for the SAS LASR Analytic Server.

TABLEINFO Statement

The TABLEINFO statement is used to return information about an in-memory table. This information includes the table name, label, number of rows and column, owner, encoding, and the time of table creation. If no table is in use, then information is returned for the in-memory tables for the server specified in the HOST= and PORT= options.

Syntax

TABLEINFO </ *options*>;

TABLEINFO Statement Options

HOST="*host-name*"
specifies the host name for the SAS LASR Analytic Server. Use this option with the PORT= option.

PORT=*number*
specifies the port number for the SAS LASR Analytic Server. If you do not specify a PORT= value, then behavior of the TABLEINFO statement depends on whether an in-memory table is active. If there is no active table, then the procedure attempts to connect to the server using the LASRPORT macro variable. If a table is active, the information is gathered for the server that is implied by the libref of the active table.

Example: Copying Tables from One Hadoop Installation to Another

Details

This example does not apply to Non-distributed SAS LASR Analytic Server. It might be necessary to work with more than one Hadoop installation so that you can copy SASHDAT files from one Hadoop installation to a newer version. The SAS LASR Analytic Server must be co-located with both Hadoop installations and both versions of Hadoop must be running.

Note: Using the HADOOPHOME= option to switch between Hadoop installations is a server-wide change. If users access the server while the setting is being switched, they might accidentally access the older Hadoop installation. Consider starting a server for the exclusive use of copying files.

Program

```
proc lasr create port=12636 serverpermissions=700; 1
    performance host="grid001.example.com" install="/opt/TKGrid" nodes=all;
run;

libname private sasiola host="grid001.example.com" port=12636 tag='hps';

data private.iris; set sashelp.iris; run;  /* a table must be active */

proc VASMP data=private.iris; 2
    serverparm hadoophome="/olderhadoop/path"; 3
quit;

proc lasr add hdfs(path="/dept/sales/y2011" direct) port=12636; 4
    performance host="grid001.example.com";
run;

proc VASMP data=private.y2011(tag="dept.sales"); 5
    serverparm hadoophome="/newerhadoop/path"; 6
run;
    save path="/dept/sales/"; 7
quit;
```

Program Description

1. Starting a server with SERVERPERMISSIONS=700 creates a single-user server. This is not required but can be used to prevent users from accessing the server while the HADOOP_HOME value is changed and accidentally accessing older or incorrect data.

2. You must have an active table. You can specify an active table with the DATA= option. Any table, such as the Iris data set can be used.
3. Use the SERVERPARM statement to specify the path to the older Hadoop installation with the HADOOPHOME= option. Specify the same path that is returned for the HADOOP_HOME environment variable for the older installation. Example: /hadoop/hadoop-0.21.
4. You must specify the DIRECT option. This statement loads table y2011 into memory from the /dept/sales directory in HDFS.
5. The TAG= option must be used to specify the in-memory table. The server tag matches the HDFS path to the table, but the slashes are replaced with periods (.). If the table was loaded from /, then specify TAG=HADOOP.
6. Use the SERVERPARM statement to specify the path to the newer Hadoop installation. Example: /hadoop-0.23/hadoop-0.23.1.
7. The SAVE statement writes the y2011 table to HDFS in the /dept/sales directory. The HDFS directory is in the newer Hadoop installation.

Chapter 5

OLIPHANT Procedure

Overview: OLIPHANT Procedure

What about the SAS Data in HDFS Engine?

The SAS Data in HDFS engine replaces the functionality provided by the OLIPHANT procedure. For more information, see "Using the SAS Data in HDFS Engine" on page 81.

What Does the OLIPHANT Procedure Do?

The OLIPHANT procedure is used to add, delete, and manage SASHDAT files that are stored in the Hadoop Distributed File System (HDFS). The procedure is used to add data sets from SAS libraries into HDFS. Once the data is in HDFS, it is stored as a SASHDAT file. The filename for the SASHDAT file is always lowercase. The procedure is also used to remove SASHDAT files from HDFS. For the data in SASHDAT files, the procedure can provide information about the data such as file size, block size, column count, row count, and so on.

Understanding How SAS LASR Analytic Server Uses HDFS

The SAS LASR Analytic Server reads data in parallel from the SASHDAT files that are added to HDFS.

Concepts: OLIPHANT Procedure

Adding Big Data

The best performance for reading data into memory on the SAS LASR Analytic Server occurs when the server is co-located with the distributed data and the data is distributed evenly. The OLIPHANT procedure distributes the data such that parallel read performance by the SAS LASR Analytic Server is maximized. In addition, the distribution also ensures an even workload for query activity performed by the SAS LASR Analytic Server.

In order to produce an even distribution of data, it is important to understand that Hadoop stores data in blocks and that any block that contains data occupies the full size of the block on disk. The default block size is 32 megabytes and blocks are padded to reach the block size after the data is written. The data is distributed among the machines in the cluster in round-robin fashion. In order to maximize disk space, you can specify a block size that minimizes the padding.

It is important to know the size of the input data set such as the row count and the length of a row. This information, along with the number of machines in the cluster, can be used to set a block size that distributes the blocks evenly on the machines in the cluster and uses the space in the blocks efficiently.

For example, if the input data set is approximately 25 million rows with a row length of 1300 bytes, then the data set is approximately 30 gigabytes. If the hardware is a cluster of 16 machines, with 15 used to provide HDFS storage, then storing 2 gigabytes on each machine is optimal. In this case, a BLOCKSIZE= setting of 32 megabytes or 64 megabytes would fill the overwhelming majority of blocks with data and reduce the space that is wasted by padding.

Adding Small Data

If the amount of data to add is not very large, then distributing it evenly can lead to poor block space utilization because at least one block is used on each machine in the cluster. However, the blocks might be mostly padding and contain little data. In these cases, the INNAMEONLY option can be used. This option sends the data to the Hadoop NameNode only. The blocks are distributed according to the default strategy used by Hadoop. The distribution is likely to be unbalanced, but the performance is not reduced because the data set is not large.

Syntax: OLIPHANT Procedure

PROC OLIPHANT HOST=*root-node* INSTALL='*grid-install-path*'
<PATH='*HDFS-path*'> <LOGUPDATE> <INNAMEONLY>;

ADD *libref.member-name* PATH='*HDFS-path*'<BLOCKSIZE=*size*><COPIES=*n*>
<REPLACE>;

REMOVE *SASHDAT-file* PATH='*HDFS-path*';

DETAILS PATH='*HDFS-path*' <FILE=*SASHDAT-file*>
<ALL | COLUMN | RECURSIVE | ROWCOUNT>;

Statement	Task	Example
ADD	Add a data set.	Ex. 1
REMOVE	Remove a data set.	Ex. 1
DETAILS	Query data set metadata.	Ex. 2

PROC OLIPHANT Statement

Enables adding, removing, and managing SASHDAT files in Hadoop Distributed File System (HDFS).

Syntax

PROC OLIPHANT HOST=*root-node* INSTALL='*grid-install-path*'
<PATH='*HDFS-path*'> <LOGUPDATE> <INNAMEONLY>

Required Arguments

HOST=

specifies the host name or IP address of the grid host. This is the machine that is running the Hadoop NameNode that is provided by SAS High-Performance Deployment of Hadoop. If you do not specify the HOST= option, it is determined from the GRIDHOST= environment variable.

Alias NAMENODE=

INSTALL=

specifies the path to the TKGrid software on the grid host. If you do not specify this option, it is determined from the GRIDINSTALLLOC= environment variable.

Alias INSTALLLOC=

Oliphant Options

PATH=
specifies the directory in HDFS to use. This value can be overridden with a PATH= option on an ADD, REMOVE, or DETAILS statement.

Alias OUTDIR=

LOGUPDATE
provides progress messages in the SAS log about the data transfer to the grid host. The data transfer size is not necessarily the same as the block size that is used to form blocks in HDFS. The data transfer size is selected to optimize network throughput.

Alias LOGNOTE

INNAMEONLY
specifies that data identified in an ADD statement should be sent as a single block to the Hadoop NameNode for distribution. This option is appropriate for smaller data sets.

Restriction The BLOCKSIZE= option is ignored.

ADD Statement

Adds a data set to HDFS as a SASHDAT file.

Example: "Example 1: Adding and Removing Files in HDFS" on page 52

Syntax

ADD *libref.member-name <add-statement-options>*;

Add Statement Options

BLOCKSIZE=
specifies the block size to use for distributing the data set. Suffix values are B (bytes), K (kilobytes), M (megabytes), and G (gigabytes). The default block size is 32M.

Alias BLOCK=

COPIES=
specifies the number of replications to make for the data set (beyond the original blocks). The default value is 2 when the INNAMEONLY option is specified and otherwise is 1. Replicated blocks are used to provide fault tolerance for HDFS. If a machine in the cluster becomes unavailable, then the blocks needed for the SASHDAT file can be retrieved from replications on other machines.

Alias COPY=

(*input-data-set-options*)
specifies any data set options to apply to the input data set.

Typically, you specify a description for the data set with the LABEL= option. The LABEL= option assigns the description to the SASHDAT file when the data set is stored in HDFS. The LABEL= option is used to override the label that is associated with the data set. Enclose the options in parentheses.

PATH='*HDFS-path*'
specifies the directory in HDFS in which to store the SASHDAT file. The value is case sensitive. The filename for the SASHDAT file that is stored in the path is always lowercase.

Alias OUTDIR=

REPLACE
specifies that the SASHDAT file should be overwritten if it already exists.

Alias OVERWRITE

VARS=(<*variables*>)
specifies the variables from the input data set to include in the SASHDAT file that is stored to HDFS. The default action is to include all the variables from the input data set.

REMOVE Statement

Removes a SASHDAT file from HDFS.

Example: "Example 1: Adding and Removing Files in HDFS" on page 52

Syntax

REMOVE *SASHDAT-file* PATH='*HDFS-path*';

Required Arguments

SASHDAT-file
specifies the name of the file to remove. Do not specify a fully qualified HDFS path. Do not enclose the value in quotation marks. Do not include the SASHDAT filename suffix. The name is converted to lowercase and the filename of the SASHDAT file in HDFS must also be in lowercase.

PATH='*HDFS-path*'
specifies the HDFS directory.

Alias OUTDIR=

DETAILS Statement

Queries information about the data in a SASHDAT file.

Example: "Example 2: Querying File Details from HDFS" on page 53

Syntax

DETAILS <*details-statement-options*>;

Details Statement Options

ALL
includes the number of rows for each SASHDAT file in the SAS output.

Alias ALLFILES

COLUMN
includes the column attributes for the specified SASHDAT file in the SAS output.

Alias COLUMNINFO

FILE=*SASHDAT-file*
specifies the name of the SASHDAT file to use. Do not specify a fully qualified HDFS path. Do not enclose the value in quotation marks. Do not include the SASHDAT filename suffix. The name is converted to lowercase and the filename of the SASHDAT file in HDFS must also be in lowercase.

Alias TABLE=

PATH='*HDFS-path*'
specify the fully qualified HDFS directory name.

Alias OUTDIR=

RECURSIVE
when FILE= is not specified, the details are reported for all SASHDAT files that are found in the path and child directories.

ROWCOUNT
includes the number of observations in the specified SASHDAT file.

Examples: OLIPHANT Procedure

Example 1: Adding and Removing Files in HDFS

Details

This PROC OLIPHANT example demonstrates adding and removing data sets to HDFS. One data set is added and a different SASHDAT file is removed.

Program

```
libname hrdata "/data/hr/2011";

proc oliphant host="grid001.example.com" install="/opt/TKGrid"; 1
```

```
   add hrdata.emps blocksize=16M path="/sasdata/2011/" replace; 2

   add (label='Bonuses for 2011') hrdata.bonus path="/sasdata/2011"; 3
   remove salary path="/sasdata/2011"; 4
run;
```

Program Description

1. The PROC OLIPHANT statement uses the HOST= and INSTALL= options to identify the SAS High-Performance Deployment of Hadoop cluster to use.
2. The ADD statement copies the EMPS data set to the HDFS path. The data set is distributed in blocks of 16 megabytes each. If an emps.sashdat file for the EMPS data set already exists, it is replaced.
3. This ADD statement includes a LABEL= option for the input data set.
4. The REMOVE statement deletes the salary.sashdat file from the HDFS path.

Example 2: Querying File Details from HDFS

Details

This PROC OLIPHANT example demonstrates how to query the details of SASHDAT files.

Program

```
proc oliphant host="grid001.example.com" install="/opt/TKGrid"; 1
   details path="/sasdata/2011/" recursive; 2

   details file=emps path="/sasdata/2011/" column; 3
run;
```

Program Description

1. The PROC OLIPHANT statement uses the HOST= and INSTALL= options to identify the SAS High-Performance Deployment of Hadoop to use.
2. The table information details for all SASHDAT files in the /sasdata/2011 directory and any subdirectories are displayed.
3. The column information for the emps.sashdat file is displayed.

Chapter 6

HPDS2 Procedure

Overview: HPDS2 Procedure

What Does the HPDS2 Procedure Do?

The HPDS2 procedure enables you to submit DS2 language statements from a SAS session to a SAS High-Performance Analytics cluster for parallel execution. The procedure verifies the syntactic correctness of the DS2 source on the client machine before it submits the source to the cluster for execution. The data that is created by the DS2 DATA statement can be generated in either of the following ways: it can be written in parallel to the cluster data store or it can be returned to the client machine and directed to any data store that is supported by SAS.

The syntax of DS2 is similar to that of the DATA step, but it does not include several key statements such as INPUT and MERGE. In addition, when you use DS2 in a SAS High-Performance Analytics environment, the SET statement of the DS2 procedure is limited to a single input stream. The use of BY processing within the SET statement is also not supported. Therefore, many of the traditional DATA step data preparation features are not available in the HPDS2 procedure. The procedure is most useful when significant amounts of computationally intensive, row-independent logic must be applied to the data. The DSTRANS procedure converts a DATA step to DS2 and in the process identifies DATA step syntax that is not compatible with PROC HPDS2.

For more information about the DSTRANS procedure and the DS2 language, see the *SAS DS2 Language Reference*. This document is available at **http://support.sas.com/documentation/solutions/ds2/DS2Ref.pdf**.

HPDS2 Procedure Features

The HPDS2 procedure enables the parallel execution of DS2 code in a distributed computing environment. The following list summarizes the basic features of the HPDS2 procedure:

- provides the ability to execute DS2 code in parallel
- enables DS2 code to be executed on a local client machine or on the SAS High-Performance Analytics cluster
- enables control of the level of parallelism per execution node and the number of machines to use
- performs a syntax check of the DS2 code on the client machine before sending it to the cluster for execution
- manages data transfer to the location of execution and the transfer back to the client machine as needed

Client and Cluster Execution Modes

The HPDS2 procedure controls the execution of DS2 code in two ways. You can control both the number of threads for each machine and also the number of machines to use. The threading provided by the HPDS2 procedure operates outside the syntax of the language. This is in contrast to the THREADS PACKAGE DS2 syntax, which provides single-machine scalability as part of the DS2 syntax.

Alternatively, the HPDS2 procedure can be executed on the SAS High-Performance Analytics cluster. In this case, one or more copies of the DS2 program are executed in parallel on each machine in the cluster.

Execution has two variations in a cluster environment:

- In the client-data (local-data) model of execution, the input data is not stored on the SAS High-Performance Analytics cluster. The data is distributed to the cluster during the execution of the HPDS2 procedure.
- In the co-located data provider model of execution, the data source is the database on the cluster. The data is stored in the distributed database, and the DS2 program that is running on each machine can read and write the data in parallel during the execution of the procedure. Instead of data being transferred across the network and possibly back to the client machine, data is passed locally between the processes on each machine of the cluster. In general, especially with large data sets, you can achieve the best HPDS2 performance if execution is with a co-located data provider.

By default, the number of instances of the DS2 program that are executed in parallel on a given machine (that is, on a client machine or on a cluster machine) is determined by the HPDS2 procedure, based on the number of CPUs (cores) that are available on the machine. The default is to execute one instance of the DS2 program in one dedicated thread for each CPU. You can change the default with the NTHREADS= option in the PERFORMANCE statement. For example, if NTHREADS=*n* is specified, then the HPDS2 procedure runs *n* instances of the DS2 program in parallel on each machine.

Parallel Execution of DS2 Code

An important characteristic of multithreaded or distributed applications is that they might produce nondeterministic or unpredictable results. The behavior of a DS2 program running in parallel is influenced by a number of factors: the pattern of data distribution that is used, the execution mode that is chosen, the number of compute nodes and threads that are used, and so on. The HPDS2 procedure does not determine whether the DS2 code that is submitted produces meaningful and reproducible results. The procedure executes the DS2 code that is provided on each of the units of work, whether these units are multiple threads on a single machine or multiple threads on separate machines. Each instance of the DS2 program operates on a subset of the data. The results that each unit of work produces are then gathered, without further aggregation, into the output data set.

Because the DS2 code instances are executed in parallel, you must consider the DS2 language elements that are included in the DS2 code block of an HPDS2 procedure. Not all DS2 language elements can be meaningfully used in multithreaded or distributed applications. For example, lagging or retaining of variables can imply the ordering of observations. A deterministic order of observations does not exist in distributed applications, and enforcing data order might have a negative impact on performance.

You can achieve optimal performance when the input data is stored in the distributed database and the SAS High-Performance Analytics software is installed on the same machines. With the data distributed in this manner, the different instances of the DS2 code running on the machines can read the input data and write the output data in parallel from the local database management system (DBMS).

Limitations

DS2 Packages

DS2 packages are collections of variables and methods that can be used in DS2 programs and threads. The HPDS2 procedure does not support DS2 packages. Use of the PACKAGE and ENDPACKAGE constructs within an HPDS2 procedure results in an error. Similarly, you cannot reference existing packages within an HPDS2 procedure.

PERFORMANCE Statement Options

The maximum allowed value for the CPUCOUNT= option in the PERFORMANCE statement is 256. However, setting CPUCOUNT= to high values, including values that substantially exceed the actual number of available CPUs, can result in unpredictable errors. These errors might include DS2 program instances that unexpectedly produce no

observations in the output data set. Specifying CPUCOUNT=ACTUAL sets CPUCOUNT to the number of physical processors that are available.

Setting the NTHREADS= option in the PERFORMANCE statement to very high values can cause out-of-memory errors. For example, errors have been generated when NTHREADS=100.

Data Input and Output

If an input data set is specified, then you must include a SET DS2GTF.in statement in the METHOD RUN() statement. If either the SET DS2GTF.in or the SET DS2GTF.out driver reference is missing, then the SAS session stops responding. The following list identifies some additional I/O limitations for the HPDS2 procedure:

- BY groups within the SET statement are not supported.
- Nested SQL within the SET statement is not supported.
- The OVERWRITE= option is not supported.
- The PUT statement does not write any data to the client log.
- Dropping the only variable in a one-variable input data set might cause SAS to stop responding or might result in an exception.

Data Types and Declarations

The following list identifies the limitations to the HPDS2 procedure for data types and declarations:

- The REAL, TINYINT, NCHAR, TIMESTAMP, DATE, and TIME data types are not supported. If you declare any of these data types within an HPDS2 procedure, then an error is displayed.
- User-defined formats are not supported.
- Formats, informats, and labels that are specified in the HAVING clause of a DECLARE statement are ignored.
- Delimited identifiers (for example, **`dcl double "a%& b"`**) are not supported.
- No warning or error messages are generated when assignments that involve out-of-bounds arrays are used.

Error Messages

Incorrect source line numbers are reported when there is an error in an HPDS2 procedure. In addition, the order of error messages that are displayed is reversed for PROC HPDS2 from the order of error messages that is generated for DS2.

Syntax: HPDS2 Procedure

```
PROC HPDS2 options;
    PERFORMANCE performance-options;
    DATA DS2GTF.out;
        DS2 statements;
        METHOD RUN();
            SET DS2GTF.in;
        END;
    ENDDATA;
    RUN;
    RUN CANCEL;
    QUIT;
```

Statement	Task	Example
PROC HPDS2	Distribute data to Greenplum.	Ex. 1
PROC HPDS2	Distribute data to Teradata.	Ex. 2

PROC HPDS2 Statement

The PROC HPDS2 statement invokes the procedure.

Syntax

PROC HPDS2 *hpds2-options*;

HPDS2 Statement Options

These options control the input data and output data for the procedure.

DATA=*libref.member-name*
specifies the SAS data set or database table to be used by PROC HPDS2. If this option is not specified, then the default value is the most recently created data set.

Alias IN=

OUTPUT=*libref.member-name*
specifies the SAS data set or database table to create with the HPDS2 procedure.

Alias OUT=

DATA Statement Statement

The DATA statement indicates the beginning of the DS2 code block. The code block terminates with the ENDDATA statement.

Syntax

DATA DS2GTF.out;

Details

You must include a reference to the DS2 Grid Table Function driver (DS2GTF.out) as part of the DATA statement. If you specify an input data set in the PROC HPDS2 statement, then you should include a RUN() method in the DS2 code block. The first statement after the METHOD RUN() statement must be the SET DS2GTF.in statement for this case. DS2GTF.out and DS2GTF.in refer to the input and output data sets, respectively.

ENDDATA Statement Statement

The DATA statement indicates the beginning of the DS2 code block. The code block terminates with the ENDDATA statement.

Syntax

ENDDATA;

Details

The ENDDATA statement terminates the DS2 code block. The statements between the DATA and ENDDATA statement are submitted to the cluster for execution. Specify the DS2 run, init, and term methods between the DATA and ENDDATA statements.

PERFORMANCE Statement

The PERFORMANCE statement defines performance parameters for multithreaded and distributed computing. It also passes variables about the distributed computing environment, and requests detailed results about the performance characteristics of a SAS High-Performance Analytics procedure.

Syntax

PERFORMANCE *performance-options*;

Performance Statement Options

You must understand the difference between the NODES= and NTHREADS= options to use these options effectively. The NODES= option specifies the number of separate machines in the cluster to use for executing the DS2 code. The NTHREADS= option specifies how many independent instances of the DS2 program to run in parallel on each

machine. If the data is located on the cluster, then specify NODES=ALL. Setting NODES=0 causes the DS2 code to execute on the client machine only. Setting the NTHREADS= option to a value that is greater than the CPU count on each machine is not likely to improve overall throughput.

COMMIT=
specifies that periodic updates are written to the SAS log when observations are sent from the client to the server instance. Whenever the number of observations sent exceeds an integer multiple of the COMMIT= size, a message is written to the SAS log. The message indicates the actual number of observations that were distributed and not an integer multiple of the COMMIT= size.

HOST=
specifies the name of the appliance host in single quotation marks or double quotation marks. If the HOST= option is specified, it overrides the value of the GRIDHOST environment variable. For Greenplum deployments, specify the host name of the master host.

Alias GRIDHOST=

INSTALL=
specifies the path to the TKGrid software on the grid host. If you do not specify this option, it is determined from the GRIDINSTALLLOC= environment variable.

Alias INSTALLOC=

NODES=
specifies the number of machines in the cluster to use for the server instance. Specify ALL to calculate the number automatically.

Alias NNODES=

NTHREADS=
specifies the number of threads for analytic computations and overrides the SAS system options THREADS | NOTHREADS. By default, the server uses one thread for each CPU core that is available on each machine in the cluster. Use this option to throttle the number of CPU cores that are used on each machine.

Note: The SAS system options THREADS | NOTHREADS apply to the client machine that issues the procedure statement. They do not apply to the machines in the cluster.

TIMEOUT=
specifies the time, in seconds, for the procedure to wait for a connection to the grid host and to establish a connection back to the client. The default value is 120 seconds. If jobs are submitted through workload management tools that might suspend access to the grid host for a longer period, you might want to increase the value.

QUIT Statement Statement

The QUIT statement stops the procedure. The HPDS2 procedure statements that have not been submitted with a RUN statement are terminated.

Syntax

QUIT;

RUN Statement Statement

The RUN statement submits the preceding HPDS2 procedure statements for execution. The procedure requires the RUN statement to submit the statements. SAS reads the program statements that are associated with one task until it reaches a RUN statement.

Syntax

RUN;

RUN CANCEL Statement

The RUN CANCEL statement cancels the preceding HPDS2 procedure statements. RUN CANCEL is useful if you enter a typographical error.

Syntax

RUN CANCEL;

Examples: HPDS2 Procedure

Example 1: Distribute Data to Greenplum

Details

This example shows how to use the HPDS2 procedure to copy a data set to Greenplum. The distribution algorithm in the first code block uses a random distribution. The second code block specifies a column name to use for distributing rows.

Program

```
libname source "/data/marketing/2012";

libname target greenplm
    server = "grid001.example.com"
    user = dbuser
    password = dbpass
    schema = public
    database = template1
    dbcommit=1000000;

proc hpds2 data = source.mktdata
```

```
    out = target.mktdata (distributed_by = 'distributed randomly'); 1

    performance host = "grid001.example.com"
        install = "/opt/TKGrid";;

    data DS2GTF.out;
        method run();
            set DS2GTF.in;
        end;
    enddata;
run;

proc hpds2 data = source.mkdata2
    out = target.mkdata2 (dbtype=(id='int')
        distributed_by='distributed by (id)'); 2

    performance host = "grid001.example.com"
        install = "/opt/TKGrid";

    data DS2GTF.out;
        method run();
            set DS2GTF.in;
        end;
    enddata;
run;
```

Program Description

1. The rows of data from the input data set are distributed randomly to Greenplum.
2. The id column in the input data set is identified as being an integer data type. The rows of data are distributed based on the value of the id column.

Example 2: Distribute Data to Teradata

Details

This example shows how to use the HPDS2 procedure to copy a data set to Teradata. The first example does not specify a distribution key. The second code block specifies a column name to use for distributing rows and creates a new column based on the log value of another column.

Program

```
libname source "/data/sales";

libname sales teradata
    server = "dbc.example.com"
    user = dbuser
    password = dbpass
    database = hps
    bulkload=yes; 1

proc hpds2 data = source.sale2011
```

```
    out = target.sale2011;

    performance host = "grid001.example.com"
        install = "/opt/TKGrid"
        dataserver = "dbc.example.com";

    data DS2GTF.out;
        dcl double log_x5; 2
        method run();
            set DS2GTF.in;
            log_x5 = log(x5);
        end;
    enddata;
run;

proc hpds2 data = source.mktdata
    out = target.mktdata (dbtype=(region='int')
          bulkload=yes
          dbcreate_table_opts='primary index(region)'); 3

    performance host = "grid001.example.com"
        install = "/opt/TKGrid"
        dataserver = "dbc.example.com";

    data DS2GTF.out;
        method run();
            set DS2GTF.in;
        end;
    enddata;
run;
```

Program Description

1. The BULKLOAD= option is specified in the LIBNAME statement.
2. The data type for the column to create is declared. The values in the column are created with the LOG function in the RUN() method.
3. The region column in the input data set is identified as being an integer data type. The column is used as a distribution key.

Chapter 7

Using the SAS LASR Analytic Server Engine

What Does the SAS LASR Analytic Server Engine Do?

The SAS LASR Analytic Server engine is used to add, remove, and access tables in a SAS LASR Analytic Server instance.

Typically, the tables that are loaded in memory are very large on a SAS LASR Analytic Server instance. The engine makes it possible to access a table and use procedures like the UNIVARIATE procedure. However, in this case, the entire table is transferred from the server instance to the SAS session and then the procedure is executed on the data. If the table is large, the data volume can overwhelm the SAS session.

The best performance for accessing the data through the engine is with a SAS High-Performance Analytics procedure. These procedures are designed to operate in a distributed computing environment and can read data in parallel from a SAS LASR Analytic Server instance.

Understanding How the SAS LASR Analytic Server Engine Works

An engine is a component of SAS software that reads from or writes to a file. The SAS LASR Analytic Server engine provides Read and Write access for data and metadata information such as variable attributes. Each engine enables SAS to access files that are in a particular format. There are several types of SAS engines.

You use the SAS LASR Analytic Server engine like other SAS data access engines. That is, you execute a LIBNAME statement to assign a libref and to specify the engine. You then use that libref throughout the SAS session where a libref is valid to access a SAS LASR Analytic Server instance.

Understanding Server Tags

What is a Server Tag?

A server tag is a text string that is associated with a table that is loaded into memory on a SAS LASR Analytic Server instance. The server tag is specified in the LIBNAME statement or as a data set option. The server tag and the table name are used together to match the name used for tables in the SAS LASR Analytic Server.

Why Use a Server Tag?

The following list identifies some reasons for specifying a server tag:

- You must use a server tag in a LIBNAME statement or as a data set option to access tables that are loaded from HDFS.
- Different users can load tables with the same name, such as Forecast, into a server instance. You use a server tag and the Forecast table name to specify which table to access.
- Tables that are loaded into memory with the LASR procedure (but not from HDFS) use the libref as the server tag. In order to access these tables, you must specify the server tag.
- When you load a table into memory from HDFS with the LASR procedure, the table is assigned a server tag. The server tag represents the directory path from which the SASHDAT file was loaded. You need to use that server tag to access the table.

See Also

- "Example 4: Accessing Tables Loaded with a DATA Step" on page 73
- "Example 5: Accessing Tables Loaded with the LASR Procedure" on page 74
- "Example 6: Accessing Tables That Are Loaded from HDFS" on page 75

Comparing the SAS LASR Analytic Server Engine with the LASR Procedure

The engine and the LASR procedure are similar in that you can use them to load tables to memory in a SAS LASR Analytic Server instance. You can also use the engine and the procedure to unload tables from memory.

You can use the engine with the APPEND data set option to add data to an existing table. The procedure cannot modify the data.

You cannot use the engine to load tables into memory from HDFS. Only the LASR procedure can be used to load tables into memory from HDFS.

You can use the LASR procedure to save in-memory tables to HDFS. The procedure writes the data in parallel because the server instance uses SAS High-Performance Deployment of Hadoop as a co-located data provider.

You can use the engine to supply a libref to SAS procedures or DATA steps. However, be aware that if you use the engine as an input data source, the data volume can be large. Large data volumes can overwhelm the SAS session.

What is Required to Use the SAS LASR Analytic Server Engine?

To use the SAS LASR Analytic Server engine, the following are required:

- access to the machines in the cluster where a SAS LASR Analytic Server is running. A server instance is started with the LASR procedure.
- an operating system user ID that is configured for passwordless secure shell (SSH) on the machines in the cluster

The requirement for passwordless SSH is not unique to using the engine. Passwordless SSH is used throughout SAS High-Performance Analytics. The SAS High-Performance Computing Management Console can be used to simplify configuring users for passwordless SSH.

What is Supported?

The following list identifies some usage notes:

- The engine does not support views or BY-group processing.
- You cannot add or drop columns from a table using a DATA step or a SAS procedure.
- You cannot replace or overwrite tables in memory. You must unload the table and then load the new table.
- You cannot use the APPEND procedure. However, you can use an APPEND data set option to achieve the same result.
- Loading tables into memory from HDFS is performed with the LASR procedure. You cannot load tables into memory from HDFS with the engine.

Chapter 8

LIBNAME Statement for the SAS LASR Analytic Server Engine

Dictionary

LIBNAME Statement Syntax

associates a SAS libref with tables on a SAS LASR Analytic Server.

Valid in: Anywhere

Category: Data Access

Syntax

LIBNAME *libref* **SASIOLA** <LASR="*server-description-file*">
<HOST="*grid-host*"> <PORT=*number*>
<TAG=*server-tag*> <FORMATEXPORT=DATA | NONE | ALL>
<STARTSERVER <=(*non-distributed-server-options*)>>
<SIGNER="*authorization-web-service-uri*">;

Required Arguments

libref
is a valid SAS name that serves as a shortcut name to associate with the tables on the SAS LASR Analytic Server. The name must conform to the rules for SAS names. A libref cannot exceed eight characters.

SASIOLA
is the engine name for the SAS LASR Analytic Server engine.

Optional Arguments

FORMATEXPORT= DATA | NONE | ALL
specifies how the engine interacts with user-defined formats when tables are added to the server instance. The default value is FORMATEXPORT=DATA. This option

can be overridden in a data set option. This option has no effect for input data sets (data sets that are transferred from the server instance to the SAS client).

DATA
specifies that the definition of all user-defined formats associated with variables written to the server instance are transferred to the server. You can then use those formats when you access the table (from a client such as SAS Visual Analytics). The user-defined formats are transferred to the server only once. The formats are not transferred as XML streams on subsequent requests to the server.

NONE
specifies that user-defined formats are not transferred to the server.

ALL
specifies that all formats in the format catalog search path are converted and transferred to the server with the table. This option is useful if the catalog search path contains user-defined formats that are not associated with variables in the table, but you might want to use later. Considerable resources can be required to generate the XML representation of the formats for deployments that have large catalogs or a deep search path.

HOST="*grid-host*"
specifies the grid host that has a running server instance. Enclose the host name in quotation marks. If you do not specify the HOST= option, it is determined from the GRIDHOST= environment variable.

Alias SERVER=

Interaction If the LASR= option is specified, then the host name specified in the HOST= option is ignored.

LASR="*server-description-file*"
specifies the server to use. Provide the fully qualified path to the server description file.

Interaction If you specify the server description file to use, then you do not need to specify the HOST= or PORT= options.

PORT=*number*
specifies the port number to use for connecting to the running server instance. If you use the PORT= option when you start a non-distributed server instance, then use this option to specify the network port number for the server.

Interaction The LASR procedure stores the port number of the last server instance that is started in the LASRPORT macro variable. You can specify PORT=&LASRPORT to use the macro variable.

SIGNER="*authorization-web-service-uri*"
specifies the URI for the SAS LASR Authorization web service. The web service is provided by the SAS Visual Analytics 6.1 software. For information about implementing row-level security, see *SAS Visual Analytics: Administration Guide*.

Example SIGNER="https://server.example.com/SASLASRAuthorization"

STARTSERVER <=(*non-distributed-server-options*)>
specifies to start a non-distributed server instance. Options are specified as name and value pairs. Separate each option with a space. The following options are available:

CLF
: specifies to use the common log format for log files. This format is a standardized text file format that is frequently analyzed by web analysis software. Specifying this option implies the LOGGING option.

KEEPLOG
: specifies to keep the log files when the server exits instead of deleting them. By default, the log files are removed when the server exits. Specifying this option implies the LOGGING option.

LOGGING
: specifies to enabling logging of server actions. The log file is stored with the signature files in the directory that is specified in the PATH= option. The log file is named in the pattern **`LASR.`*`timestamp`*`.0.saslasr.log`**.

MAXLOGSIZE=*n*
: specifies the maximum log file size, in megabytes, for a log file. When the log file reaches the specified size, the log file is rolled over and renamed with a sequentially assigned index number (for example, **`.log.1`**). The default value is 100 megabytes. Specifying this option implies the LOGGING option.

  TIP Do not include an MB or M suffix when you specify the size.

MAXLOGROLL=*n*
: specifies the maximum number of log files to create. When the maximum has been reached, the server begins to overwrite existing log files. The oldest log file is overwritten first. The default value is 10. Specifying this option implies the LOGGING option.

MERGELIMIT=*n*
: specifies the limit for merging large result sets into smaller groups. The MERGEBINS= option specifies the size of the group. If MERGEBINS= is not specified, then *n* is the bin limit.

MERGEBINS=*b*
: specifies the number of bins that numeric variables are binned into when MERGELIMIT=*n* is reached.

NTHREADS=*n*
: specifies the number of threads to use for the server. By default, *n* equals the number of CPU cores on the machine.

PATH="*signature-file-path*"
: specifies the directory to use for storing the server and table signature files. The specified directory must already exist.

  If you do not specify a value for PATH=, the signature files are stored in the default utility file directory of the SAS session.

PERMISSION=*mode*
: specifies the permission setting for accessing the server instance. The mode value is specified as an integer value such as 755. The mode corresponds to the mode values that are used for UNIX file access permissions.

  Alias PERM=

  Range 600 to 777

Alias START=

TAG=*server-tag*
specifies the tag to use for identifying the tables in the server instance. The value for *server-tag* cannot exceed 128 characters in length.

Examples

Example 1: Submitting a LIBNAME Statement Using the Defaults

Program

The following example shows the code for starting a server with the LASR procedure and then connecting to the same server with a LIBNAME statement:

```
option set=GRIDHOST="grid001.example.com"; 1
option set GRIDINSTALLLOC="/opt/TKGrid";

proc lasr
    create port=10010
    path="/tmp" noclass;

    performance nodes=all;
run;

libname salessvr sasiola; 2
```

```
NOTE: No tag was specified in the LIBNAME statement. The default tag (WORK) is
      used to name and identify tables in the LASR Analytic Server. You can
      specify a tag as a data set option.
NOTE: Libref SALESSVR was successfully assigned as follows:
      Engine:        SASIOLA
      Physical Name: SAS LASR Analytic Server engine on host
      'grid001.example.com', port 10010
```

Program Description

1. The grid host is specified in the GRIDHOST environment variable.
2. The default LIBNAME statement does not include the LASR=, HOST=, or PORT= options. The LIBNAME statement uses host name from the GRIDHOST environment variable and the LASRPORT macro variable and connect to server instance.

Example 2: Submitting a LIBNAME Statement Using the LASR= Option

The following example shows a LIBNAME statement that uses the LASR= option to specify the server instance to use:

```
proc lasr
    create="/tmp/hrsvr" 1
    path="/opt/VADP/var/hr"
    noclass;

    performance host="grid001.example.com" install="/opt/TKGrid" nodes=all; 2
run;

libname hrsvr sasiola lasr="/tmp/hrsvr"; 3
```

Program Description

1. A server instance is started with the CREATE= option. The server description file is /tmp/hrsvr.
2. The HOST= option is specified in the PERFORMANCE statement rather than specifying the GRIDHOST environment variable.
3. The LASR= option specifies the server description file that was created when the server instance started.

Example 3: Submitting a LIBNAME Statement Using the HOST= and PORT= Options

The following example shows the code for starting a server with the LASR procedure and then submitting a LIBNAME statement to use the same server by specifying the HOST= and PORT= options.

```
proc lasr
    create port=10010
    path="/tmp"
    noclass;

    performance host="grid001.example.com" install="/opt/TKGrid" nodes=all;
run;
```

```
NOTE: The LASR procedure is executing in the distributed computing environment
      with 7 worker nodes.
NOTE: The server started on 'grid001.example.com' port 10010. 1
NOTE: The LASR Analytic Server port '12637' has been assigned to the macro
      variable "LASRPORT".
```

```
libname hrdata sasiola host="grid001.example.com" port=10010 tag='hr'; 2
```

```
NOTE: Libref hrdata was successfully assigned as follows:
      Engine:        SASIOLA
      Physical Name: SAS LASR Analytic Server engine on host
      'grid001.example.com', port 10010
```

Program Description

1. When a server instance is started, the SAS log indicates the port number for the server instance.
2. The PORT= option in the LIBNAME statement references the port number. The value for the PORT= option can also be specified as PORT=&LASRPORT to use the port number for the most recently started server instance.

Example 4: Accessing Tables Loaded with a DATA Step

The following example shows how to use the engine without a server tag in a DATA step.

```
libname sales sasiola port=10010;

data sales.prdsale;
    set sashelp.prdsale;
run;
```

```
proc datasets lib=sales;
quit;

* a server tag is not needed to access the data ;
proc print data=sales.prdsale(obs=5);
run;
```

When no server tag is specified, a default server tag that is named WORK is used.

Output 8.1 *DATASETS Procedure Output Showing the WORK Server Tag*

Directory	
Libref	SALES
Engine	SASIOLA
Physical Name	SAS LASR Analytic Server engine on host 'grid001.example.com', port 10010

#	Name	Member Type	Number of Rows	Number of Columns	Last Modified	Server Tag	Data Encoding	Owner
1	PRDSALE	DATA	1440	10	29Nov12:20:53:13	WORK	latin1	sasdemo

Example 5: Accessing Tables Loaded with the LASR Procedure

When tables are loaded to memory on a server instance with the LASR procedure, the libref that is used with the procedure is set as the server tag. The following example shows how to add a table to a server instance and then access the table with a LIBNAME statement that includes a server tag.

```
proc lasr port=10010 add data=sashelp.prdsale noclass;
run;

libname lasr2 sasiola tag=sashelp;

proc datasets lib=lasr2;
run;

* a server tag is not needed to access the data              ;
* because a server tag is specified in the LIBNAME statement ;
proc print data=lasr2.prdsale(obs=5);
run;
```

By default, the libref is used as the server tag. The following display shows **sashelp** used as the server tag.

Output 8.2 *DATASETS Procedure Output Showing the SASHELP Server Tag*

Directory	
Libref	LASR2
Engine	SASIOLA
Physical Name	SAS LASR Analytic Server engine on host 'grid001.example.com', port 10010

#	Name	Member Type	Number of Rows	Number of Columns	Last Modified	Server Tag	Data Encoding	Owner
1	PRDSALE	DATA	1440	10	29Nov12:21:34:03	SASHELP	latin1	sasdemo

Example 6: Accessing Tables That Are Loaded from HDFS

When tables are loaded into memory on the server instance with the LASR procedure and the SAS Data in HDFS engine, the server tag is related to the HDFS directory name. The server tag is the same as the HDFS path to the SASHDAT file, but is delimited with periods (.) instead of slashes (/).

The following example shows how to add a table to a server instance from HDFS and then access the table with a LIBNAME statement that includes a server tag.

```
libname sales sashdat path="/dept/sales";

proc lasr port=10010 add data=sales.sales2012 noclass;
run;

libname lasr3 sasiola tag="dept.sales";

proc datasets lib=lasr3;
run;

* access the data with the "dept.sales" server tag;
proc print data=lasr3.sales2012(obs=5);
run;
```

Output 8.3 *DATASETS Procedure Output Showing the DEPT.SALES Server Tag*

Directory	
Libref	LASR3
Engine	SASIOLA
Physical Name	SAS LASR Analytic Server engine on host 'grid001.example.com', port 10010

#	Name	Member Type	Number of Rows	Number of Columns	Last Modified	Server Tag	Data Encoding	Owner
1	SALES2012	DATA	452459	39	19Jul12:21:34:0	DEPT.SALES	la in1	sasdemo

Example 7: Loading a Table and Partitioning

Partitioning a table as it is loaded to memory can be a powerful feature for reducing processing times. For more information, see "Data Partitioning and Ordering" on page 12.

```
libname lasrlib sasiola host="grid001.example.com" port=10010 tag="sales";

data lasrlib.prdsale(partition=(country region) orderby=(descending year)); 1
    set sashelp.prdsale;
run;
```

Program Description

The Prdsale table is distributed to the machines in the cluster according to the PARTITION= data set option. The rows are distributed according to the unique combinations of the formatted values for the variables Country and Region. In addition, the ORDERBY= option is used to sort the rows in each partition by Year, in descending order.

Chapter 9

Data Set Options for the SAS LASR Analytic Server Engine

Dictionary

APPEND Data Set Option

specifies to append the data to an existing table in the server instance.

Valid in: DATA Step

Default: NO

Interaction: You must use the NOCLASS option if you load the initial table with the LASR procedure.

Syntax

APPEND

Details

By default, the SAS LASR Analytic Server engine does not permit appending observations to tables. The APPEND data set option can be used to permit adding observations to an existing table with a DATA step.

Example Code 9.1 *Using the APPEND Data Set Option*

```
proc lasr add data=grp1.sales noclass port=10010;
run;

libname grp1lasr host="grid001.example.com" port=10010 tag=grp1;
```

```
data grp1lasr.sales (append);
    set yr2012.sales (keep=date location amount);
run;
```

As shown in the preceding example, the APPEND data set option can be used to add observations to an existing table. The KEEP= option on the input data set specifies the variables from the input data to append. Any variables for which the input data set does not append data are set to missing. You cannot add new variables to the table.

The example also shows how to load the initial table to memory with the LASR procedure. The NOCLASS option must be specified if you use the LASR procedure. As an alternative, you can load the initial table to memory with the SAS LASR Analytic Server engine.

FORMATEXPORT= Data Set Option

specifies how the engine interacts with user-defined formats when tables are added to the server instance.

Syntax

FORMATEXPORT=DATA | NONE | ALL

Details

This option is used to override the FORMATEXPORT= option for the LIBNAME statement.

See Also

FORMATEXPORT= option in the LIBNAME statement

HASH Data Set Option

specifies that when partitioning data, the distribution of partitions is not determined by a tree, but by a hashing algorithm. As a result, the distribution of the partitions is not as evenly balanced, but it is effective when working with high-cardinality partition keys (in the order of millions of partitions).

Syntax

PARTITION=(*variable-list*) **HASH**

Example

```
data lasrlib.transactions(partition=(cust_id year) hash);
    set somelib.sometable;
run;
```

ORDERBY= Data Set Option

specifies the variables by which to order the data within a partition.

Example: "Example 7: Loading a Table and Partitioning" on page 75

Syntax

ORDERBY=(*variable-list*)

ORDERBY=(*variable-name* <DESCENDING> *variable-name*)

Details

The variable names in the *variable-list* are separated by spaces.

The ordering is hierarchical. For example, ORDERBY=(A B) specifies ordering by the values of variable B within the ordered values of variable A. The specified variables must exist and cannot be specified as partitioning variables. The order is determined based on the raw value of the variables and uses locale-sensitive collation for character variables. By default, values are arranged in ascending order. You can specify descending order by preceding the variable name in the *variable-list* with the keyword DESCENDING.

Example

The following code sample orders the data in the partitions by Year in ascending order and then by Quarter in descending order.

```
data lasrlib.prdsale (partition=(country region)
                          orderby=(year descending quarter));
    set sashelp.prdsale;
run;
```

PARTITION= Data Set Option

specifies the list of partitioning variables to use for partitioning the table.

Example: "Example 7: Loading a Table and Partitioning" on page 75

Syntax

PARTITION=(*variable-list*)

Details

Partitioning is available only when you create tables. User-defined format definitions for partitioning variables are always transferred to the server, regardless of the FORMATEXPORT= option.

Partitioning by a variable that does not exist in the output table is an error. Partitioning by a variable listed in the ORDERBY= option is also an error. Partition keys are derived based on the formatted values in the order of the variable names in the *variable-list*.

Be aware that the key construction is not hierarchical. That is, PARTITION=(A B) specifies that any unique combination of formatted values for variables A and B defines a partition.

PERM= Data Set Option

specify the permission setting for the table in the server.

Alias: PERMISSION=

Syntax

PERM=*mode*

Details

The *mode* is specified as an integer (for example, PERM=755). The value is converted by the engine to a umask. If no permission is specified, the access permissions for the table are set according to the umask of user that loads the table.

TAG= Data Set Option

specifies the tag to use for identifying the tables in the server instance.

Syntax

TAG='*server-tag*'

Details

If no TAG= option is specified as a data set option, then the TAG= option from the LIBNAME statement is used. If the LIBNAME statement does not specify the TAG= option, then the name of the libref is used as the server tag.

UCA Data Set Option

specifies that you want to use Unicode Collation Algorithms (UCA) to determine the ordering of character variables in the ORDERBY= option.

Syntax

PARTITION=(*key*) ORDERBY=(*variable-list*) **UCA**

Chapter 10

Using the SAS Data in HDFS Engine

What Does the SAS Data in HDFS Engine Do?

The SAS Data in HDFS engine is used to distribute data in the Hadoop Distributed File System (HDFS) that is provided by SAS High-Performance Deployment of Hadoop. The engine enables you to distribute the data in a format that is designed for high-performance analytics. The block redundancy and distributed computing provided by SAS High-Performance Deployment of Hadoop is complemented by the block structure that is created with the engine.

The engine is designed to distribute data in HDFS only. Because the data volumes in HDFS are typically very large, the engine is not designed to read from HDFS and transfer data back to the SAS client. For example, consider the case of reading several terabytes of data from a distributed computing environment, transferring that data back to a SAS session, and then using the UNIVARIATE or REG procedures on such a large volume of data. In contrast, the SAS High-Performance Analytics procedures are designed to operate in a distributed computing environment and to read data in parallel from a co-located data provider like SAS High-Performance Deployment of Hadoop.

Understanding How the SAS Data in HDFS Engine Works

An engine is a component of SAS software that reads from or writes to a file. The SAS Data in HDFS engine is write-only for data and read-write for metadata information such as variable attributes. Each engine enables SAS to access files that are in a particular format. There are several types of SAS engines.

You use the SAS Data in HDFS engine like other SAS data access engines. That is, you execute a LIBNAME statement to assign a libref and to specify the engine. You then use that libref throughout the SAS session where a libref is valid to transfer data to the

Hadoop Distributed File System (HDFS) or to retrieve information about a table in HDFS.

What is Required to Use the SAS Data in HDFS Engine?

To use the SAS Data in HDFS engine, the following are required:

- access to the machines in the cluster where SAS High-Performance Deployment of Hadoop is installed and running
- an operating system user ID that is configured for passwordless secure shell (SSH) on the machines in the cluster

The requirement for passwordless SSH is not unique to using the engine. Passwordless SSH is used throughout SAS High-Performance Analytics. The SAS High-Performance Computing Management Console can be used to simplify configuring users for passwordless SSH.

What is Supported?

The SAS Data in HDFS engine is used with SAS High-Performance Deployment of Hadoop only.

The engine is designed as a write-only engine for transferring data to HDFS. However, SAS High-Performance Analytics procedures are designed to read data in parallel from a co-located data provider. The LASR procedure, and other procedures such as HPREG and HPLOGISTIC, can read data from HDFS with the engine. The HPDS2 procedure is designed to read data and write data in parallel. The HPDS2 procedure can be used with the engine to read data from HDFS and create new tables in HDFS.

Whenever a SAS High-Performance Analytics procedure is used to create data in HDFS, the procedure creates the data with a default block size of 8 megabytes. This size can be overridden with the BLOCKSIZE= data set option.

The engine does not support views.

Chapter 11

LIBNAME Statement for the SAS Data in HDFS Engine

Dictionary

LIBNAME Statement Syntax

Associates a SAS libref with SASHDAT tables stored in HDFS.

Valid in: Anywhere

Category: Data Access

Syntax

LIBNAME *libref* **SASHDAT**
<HOST="*grid-host*"> <INSTALL="*grid-install-location*">
<PATH="*HDFS-path*"> <COPIES=*n* > <INNAMEONLY>

Required Arguments

libref
is a valid SAS name that serves as a shortcut name to associate with the SASHDAT tables that are stored in the Hadoop Distributed File System (HDFS). The name must conform to the rules for SAS names. A libref cannot exceed eight characters.

SASHDAT
is the engine name for the SAS Data in HDFS engine.

Optional Arguments

COPIES=*n*
specifies the number of replications to make for the data set (beyond the original blocks). The default value is 2 when the INNAMEONLY option is specified and otherwise is 1. Replicated blocks are used to provide fault tolerance for HDFS. If a machine in the cluster becomes unavailable, then the blocks needed for the SASHDAT file can be retrieved from replications on other machines. If you specify

COPIES=0, then the original blocks are distributed, but no replications are made and there is no fault tolerance for the data.

HOST="*grid-host*"
specifies the grid host that has a running Hadoop NameNode. Enclose the host name in quotation marks. If you do not specify the HOST= option, it is determined from the GRIDHOST= environment variable.

INNAMEONLY
specifies that when data is added to HDFS, that it should be sent as a single block to the Hadoop NameNode for distribution. This option is appropriate for smaller data sets.

Alias NODIST

INSTALL="*grid-install-location*"
specifies the path to the TKGrid software on the grid host. If you do not specify this option, it is determined from the GRIDINSTALLLOC= environment variable.

PATH="*HDFS-path*"
specifies the fully qualified path to the HDFS directory to use for SASHDAT files. You do not need to specify this option in the LIBNAME statement because it can be specified as a data set option.

Examples

Example 1: Submitting a LIBNAME Statement Using the Defaults

Program

The following example shows the code for connecting to a Hadoop NameNode with a LIBNAME statement:

```
option set=GRIDHOST="grid001.example.com"; 1
option set GRIDINSTALLLOC="/opt/TKGrid";

libname hdfs sashdat; 2
```

```
NOTE: Libref HDFS was successfully assigned as follows:
      Engine:        SASHDAT
      Physical Name: grid001.example.com
```

Program Description

1. The host name for the Hadoop NameNode is specified in the GRIDHOST environment variable.
2. The LIBNAME statement uses host name from the GRIDHOST environment variable and the path to TKGrid from the GRIDINSTALLLOC environment variable. The PATH= and COPIES= options can be specified as data set options.

Example 2: Submitting a LIBNAME Statement Using the HOST=, INSTALL=, and PATH= Options

The following example shows the code for submitting a LIBNAME statement with the HOST=, INSTALL=, and PATH= options.

```
libname hdfs sashdat host="grid001.example.com" install="/opt/TKGrid"
    path="/user/sasdemo";
```

```
NOTE: Libref HDFS was successfully assigned as follows:
      Engine:        SASHDAT
      Physical Name: Directory '/user/sasdemo' of HDFS cluster on host
                     grid001.example.com
```

Example 3: Adding Tables to HDFS

The following code sample demonstrates the LIBNAME statement and the REPLACE= and BLOCKSIZE= data set options. The LABEL= data set option is common to many engines.

```
libname arch "/data/archive";
libname hdfs sashdat host="grid001.example.com" install="/opt/TKGrid"
    path="/dept";

data hdfs.allyears(label="Sales records for previous years"
                   replace=yes blocksize=32m);
  set arch.sales2012
      arch.sales2011
      ...
  ;
run;
```

Example 4: Adding a Table to HDFS with Partitioning

The following code sample demonstrates the PARTITION= and ORDERBY= data set options. The rows are partitioned according to the unique combinations of the formatted values for the Year and Month variables. Within each partition, the rows are sorted by descending values of the Prodtype variable. For more information, see "Data Partitioning and Ordering" on page 12.

```
libname hdfs sashdat host="grid001.example.com" install="/opt/TKGrid"
    path="/dept";

data hdfs.prdsale(partition=(year month) orderby=(descending prodtype));
    set sashelp.prdsale;
run;
```

Example 5: Removing Tables from HDFS

Removing tables from HDFS can be performed with the DATASETS procedure.

```
libname hdfs sashdat host="grid001.example.com" install="/opt/TKGrid"
    path="/dept";

proc datasets lib=hdfs;
    delete allyears;
run;
```

```
NOTE: Deleting HDFS.ALLYEARS (memtype=DATA).
```

Example 6: Creating a SASHDAT File from Another SASHDAT File

The following example shows copying a data set from HDFS, adding a calculated variable, and then writing the data to HDFS in the same library. The BLOCKSIZE= data set option is used to override the default 8-megabyte block size that is created by SAS

High-Performance Analytics procedures. The COPIES=0 data set option is used to specify that no redundant blocks are created for the output SASHDAT file.

```
libname hdfs sashdat host="grid001.example.com" install="/opt/TKGrid"
    path="/dept";

proc hpds2
    in = hdfs.allyears(where=(region=212)) 1
    out = hdfs.avgsales(blocksize=32m copies=0); 2

    data DS2GTF.out;
        dcl double avgsales;
        method run();
        set DS2GTF.in;
            avgsales = avg(month1-month12);
        end;
    enddata;
run;
```

1 The WHERE clause is used to subset the data in the input SASHDAT file.

2 The BLOCKSIZE= and COPIES= options are used to override the default values.

Example 7: Working with CSV Files

The comma-separated value (CSV) file format is a popular format for files stored in HDFS. The SAS Data in HDFS engine can read these files in parallel. The engine does not write CSV files.

List the Variables in a CSV File

The following example shows how to access a CSV file in HDFS and use the CONTENTS procedure to list the variables in the file. For this example, the first line in the CSV file lists the variables names. The GETNAMES data set option is used to read them from the first line in the file.

```
libname csvfiles sashdat host="grid001.example.com" install="/opt/TKGrid"
    path="/user/sasdemo/csv";

proc contents data=csvfiles.rep(filetype=csv getnames);
run;
```

Output 11.1 *List the Variables in a CSV File with the CONTENTS Procedure*

The SAS System

The CONTENTS Procedure

Data Set Name	/user/sasdemo/csv/rep.csv	**Observations**	.
Member Type	DATA	**Variables**	6
Engine	SASHDAT	**Indexes**	0
Created	Tuesday, July 03, 2012 08:53:36 AM	**Observation Length**	208
Last Modified	Thursday, June 28, 2012 02:48:41 PM	**Deleted Observations**	0
Protection		**Compressed**	NO
Data Set Type		**Sorted**	NO
Label			
Data Representation	SOLARIS_X86_64, LINUX_X86_64, ALPHA_TRU64, LINUX_IA64		
Encoding	Default		

Alphabetic List of Variables and Attributes

#	Variable	Type	Len
1	alias	Char	13
6	enddt	Char	10
3	path	Char	140
4	set	Char	1
5	startdt	Char	10
2	id	Char	7

Convert a CSV File to SASHDAT

The engine is not designed to transfer data from HDFS to a SAS client. As a consequence, the contents of a CSV file can be accessed only by a SAS High-Performance Analytics procedure that runs on the same cluster that is used for HDFS. The SAS High-Performance Analytics procedures can read the data because the procedures are designed to read data in parallel from a co-located data provider.

The following code sample shows how to convert a CSV file to a SASHDAT file with the HPDS2 procedure.

```
option set=GRIDHOST="grid001.example.com"; 1
option set=GRIDINSTALLLOC="/opt/TKGrid";

libname csvfiles sashdat path="/user/sasdemo/csv";

proc hpds2 in=csvfiles.rep(filetype=csv getnames) 2
    out=csvfiles.rephdat(path="/user/sasdemo" copies=0 blocksize=32m); 3

    data DS2GTF.out;
        method run();
            set DS2GTF.in;
        end;
    enddata;
run;
```

1 The values for the GRIDHOST and GRIDINSTALLLOC environment variables are read by the SAS Data in HDFS engine in the LIBNAME statement and by the HPDS2 procedure.

2 The FILETYPE=CSV data set option enables the engine to read the CSV file. The GETNAMES data set option is used to read the variable names from the first line in the CSV file.

3 The PATH= data set option is used to store the output as **`/user/sasdemo/rephdat.sashdat`**. The COPIES=0 data set option is used to specify that no redundant blocks are created for the rephdat.sashdat file.

Chapter 12

Data Set Options for the SAS Data in HDFS Engine

Dictionary

BLOCKSIZE= Data Set Option

specifies the block size to use for distributing the data set.

Valid in: DATA Step

Default: 2 megabytes

Example: “Example 6: Creating a SASHDAT File from Another SASHDAT File” on page 85

Syntax

BLOCKSIZE=

Details

By default, the SAS Data in HDFS engine distributes data in 2-megabyte blocks or the length of a record, which ever is greater. You can override this value by specifying the block size to use. Suffix values are B (bytes), K (kilobytes), M (megabytes), and G

(gigabytes). The actual block size is slightly larger than the value that you specify. This occurs for any of the following reasons:

- to reach the record length. This occurs if the specified size is less than the record length.
- to align on a 512-byte boundary.
- to include a metadata header in HDFS for the SASHDAT file.

The following code shows an example of specifying the BLOCKSIZE= option.

Example Code 12.1 *Using the BLOCKSIZE= Data Set Option*

```
data hdfs.sales (blocksize=48M);
    set yr2012.sales;
run;
```

COLUMNS= Data Set Option

specifies the variable names and types for a CSV file.

Alias: COLS=

Applies to: Reading CSV files

Syntax

COLUMNS=(*column-specification* < …*column-specification*>);

Required Argument

column-specification
is a name-value pair that specifies the column name and data type. For numeric data, specify **double** as the data type. For character data, specify **'char(*length*)'**.

Default Any variables that are not named are assigned the name VAR*n*.

Example
```
columns=(station='char(4)' obsdate='char(18)' tempf=double precip=double)
```

Details

Numeric variables use eight bytes. For character variables, if the byte length is not specified, then the default action is to use eight bytes. If the variable in the CSV file uses fewer bytes than the specified length, then the variable is padded with spaces up to the specified length. If the variable in the CSV file uses more bytes than the specified length, then the variable is truncated to the specified length.

If the variable name is not specified, then the variable is named automatically. Automatically named variables are named **VAR*n***, starting at 1. If the data type is not specified and cannot be determined, the variable is assigned as **char(8)**.

TIP Do not use a comma between each column specification. Enclose **'char(*n*)'** in quotation marks.

COPIES= Data Set Option

specifies the number of replications to make for the data set (beyond the original blocks).

Default: 1

Syntax

COPIES=*n*

Details

The default value is 1. This default value creates one copy of each block, in addition to the original block. When the INNAMEONLY option is specified, the default is 2. Replicated blocks are used to provide fault tolerance for HDFS. If a machine in the cluster becomes unavailable, then the blocks needed for the SASHDAT file can be retrieved from replications on other machines.

You can specify COPIES=0 to avoid creating redundant blocks for the SASHDAT file. This option can be useful to preserve storage space when you have redundancy for the source data.

FILETYPE= Data Set Option

specifies whether to access a comma-separated value (CSV) file instead of a SASHDAT file.

Applies to: Reading CSV files

Syntax

FILETYPE=CSV

Details

The SAS Data in HDFS engine can be used to read CSV files. The engine does not write CSV files. Specify this option to use the file as input for a SAS High-Performance Analytics procedure or the SAS LASR Analytic Server.

The filename for CSV files in HDFS can be upper, mixed, or lower case. If more than one file in the directory has the same name (but with different casing), the engine does not read the file because the file reference is ambiguous.

See Also

- COLUMNS= data set option
- GETNAMES data set option
- GUESSROWS= data set option

GETNAMES Data Set Option

specifies to read variable names from the first line in the CSV file.

Applies to: Reading CSV files

Syntax

GETNAMES

Details

Specify this option if the first line of a CSV file contains the variable names for the file. Alternatively, you can specify the variable names in the COLUMNS=data set option, or you can use the default names that are provided by the SAS Data in HDFS engine.

GETOBS Data Set Option

specifies to retrieve the number of observations in SASHDAT files.

Syntax

GETOBS

Details

By default, the SAS Data in HDFS engine does not compute the number of observations in a SASHDAT file. This improves performance for SASHDAT files that are distributed among a large number of blocks, or for HDFS directories that have a large number of SASHDAT files. When you specify this option, the engine calculates the number of observations in a SASHDAT file.

```
ods select attributes;

proc datasets library=hdfs;
    contents data=sales2012(getobs);
run;
```

The SAS System

The DATASETS Procedure

Data Set Name	/user/sasdemo/sales2012.sashdat	**Observations**	100000
Member Type	DATA	**Variables**	88
Engine	SASHDAT	**Indexes**	0
Created	Thursday, June 28, 2012 09:47:45 AM	**Observation Length**	704
Last Modified	Wednesday, June 27, 2012 04:06:40 PM	**Deleted Observations**	0
Protection		**Compressed**	NO
Data Set Type		**Sorted**	NO
Label			
Data Representation	SOLARIS_X86_64, LINUX_X86_64, ALPHA_TRU64, LINUX_IA64		
Encoding	wlatin1 Western (Windows)		

GUESSROWS= Data Set Option

specifies the number of lines in CSV file to scan for determining variable types and lengths.

Default: 20

Applies to: Reading CSV files

Syntax

GUESSROWS=*n*

Details

The SAS Data in HDFS engine scans the specified number of lines from the CSV file to determine the variable types and lengths. If the GETNAMES data set option is specified, then the engine begins scanning lines from the second line in the file.

HASH Data Set Option

specifies that when partitioning data, the distribution of partitions is not determined by a tree, but by a hashing algorithm. As a result, the distribution of the partitions is not as evenly balanced, but it is effective when working with high-cardinality partition keys (in the order of millions of partitions).

Syntax

PARTITION=(*variable-list*) **HASH**

Example

```
data hdfs.transactions(partition=(cust_id year) hash);
    set somelib.sometable;
run;
```

LOGUPDATE Data Set Option

provides progress messages in the SAS log about the data transfer to the grid host.

Syntax

LOGUPDATE

Details

The data transfer size is not necessarily the same as the block size that is used to form blocks in HDFS. The data transfer size is selected to optimize network throughput. A message in the SAS log does not mean that a block was written to HDFS. The message indicates the transfer progress only.

```
data hdfs.sales2012(logupdate);
    set saleslib.sales2012;
run;
```

```
NOTE: 4096 kBytes (5191 records) have been transmitted (1.91 MB/sec).
NOTE: 8192 kBytes (10382 records) have been transmitted (3.65 MB/sec).
NOTE: 12288 kBytes (15573 records) have been transmitted (5.19 MB/sec).
NOTE: 16384 kBytes (20764 records) have been transmitted (6.15 MB/sec).
NOTE: 20480 kBytes (25955 records) have been transmitted ( 7.3 MB/sec).
NOTE: 24576 kBytes (31146 records) have been transmitted (8.16 MB/sec).
NOTE: 28672 kBytes (36337 records) have been transmitted (8.83 MB/sec).
NOTE: 32768 kBytes (41528 records) have been transmitted (9.73 MB/sec).
NOTE: 36864 kBytes (46719 records) have been transmitted (10.3 MB/sec).
NOTE: 40960 kBytes (51910 records) have been transmitted (10.8 MB/sec).
NOTE: 45056 kBytes (57101 records) have been transmitted (11.6 MB/sec).
NOTE: 49152 kBytes (62292 records) have been transmitted (  12 MB/sec).
NOTE: 53248 kBytes (67483 records) have been transmitted (12.4 MB/sec).
NOTE: 57344 kBytes (72674 records) have been transmitted (12.9 MB/sec).
NOTE: 61440 kBytes (77865 records) have been transmitted (13.2 MB/sec).
NOTE: 65536 kBytes (83056 records) have been transmitted (13.5 MB/sec).
NOTE: 69632 kBytes (88247 records) have been transmitted (13.9 MB/sec).
NOTE: 73728 kBytes (93438 records) have been transmitted (14.1 MB/sec).
NOTE: 77824 kBytes (98629 records) have been transmitted (14.3 MB/sec).
NOTE: There were 100000 observations read from the data set SALESLIB.YEAR2012.
NOTE: The data set /user/sasdemo/sales2012 has 100000 observations and 86
variables.
NOTE: 78906 kBytes (100000 records) have been transmitted (14.3 MB/sec).
```

ORDERBY= Data Set Option

specifies the variables by which to order the data within a partition.

Example: “Example 4: Adding a Table to HDFS with Partitioning” on page 85

Syntax

ORDERBY=(*variable-list*)

ORDERBY=(*variable-name* <DESCENDING> *variable-name*)

Details

The variable names in the *variable-list* are separated by spaces.

The ordering is hierarchical. For example, ORDERBY=(A B) specifies ordering by the values of variable B within the ordered values of variable A. The specified variables must exist and cannot be specified as partitioning variables. The order is determined based on the raw value of the variables and uses locale-sensitive collation for character variables. By default, values are arranged in ascending order. You can specify descending order by preceding the variable name in the *variable-list* with the keyword DESCENDING.

Example

The following code sample orders the data in the partitions by Year in ascending order and then by Quarter in descending order.

```
data hdfs.prdsale (partition=(country region)
                   orderby=(year descending quarter));
    set sashelp.prdsale;
run;
```

PARTITION= Data Set Option

specifies the list of partitioning variables to use for partitioning the table.

Interaction: If you specify the PARTITION= option and the BLOCKSIZE= option, but the block size is less than the calculated size that is needed for a block, the operation fails and the table is not added to HDFS. If you do not specify a block size, the size is calculated to accommodate the largest partition.

Example: "Example 4: Adding a Table to HDFS with Partitioning" on page 85

Syntax

PARTITION=(*variable-list*)

Details

Partitioning is available only when you add tables to HDFS. If you partition the table when you add it to HDFS, it becomes a partitioned in-memory table when you load it to SAS LASR Analytic Server. If you also specify the ORDERBY= option, then the ordering is preserved when the table is loaded to memory too.

Partition keys are derived based on the formatted values in the order of the variable names in the *variable-list*. All of the rows with the same partition key are stored in a single block. This ensures that all the data for a partition is loaded into memory on a single machine in the cluster. The blocks are replicated according to the default replication factor or the value that you specify for the COPIES= option.

If user-defined formats are used, then the format name is stored with the table, but not the format. The format for the variable must be available to the SAS LASR Analytic Server when the table is loaded into memory. This can be done by having the format in the format catalog search path for the SAS session.

Be aware that the key construction is not hierarchical. That is, PARTITION=(A B) specifies that any unique combination of formatted values for variables A and B defines a partition.

Partitioning by a variable that does not exist in the output table is an error. Partitioning by a variable listed in the ORDERBY= option is also an error.

PATH= Data Set Option

specifies the fully qualified path to the HDFS directory to use for SASHDAT files.

Syntax

PATH='*HDFS-path*'

Details

This option overrides the PATH= option specified in the LIBNAME statement.

PERM= Data Set Option

specifies how the engine sets the file access permissions on the SASHDAT file.

Alias: PERMISSION=

Syntax

PERM=*mode*

Details

The *mode* value is specified as an integer value such as 755. The mode corresponds to the mode values that are used for UNIX file access permissions.

REPLACE= Data Set Option

specifies whether to overwrite an existing SASHDAT file.

Syntax

REPLACE=YES | NO

Details

By default, the SAS Data in HDFS engine does not replace SASHDAT files. Specify REPLACE=YES as a data set option to replace a SASHDAT file by overwriting it.

UCA Data Set Option

specifies that you want to use Unicode Collation Algorithms (UCA) to determine the ordering of character variables in the ORDERBY= option.

Syntax

PARTITION=(*key*) ORDERBY=(*variable-list*) **UCA**

Glossary

Apache Hadoop
a framework that allows for the distributed processing of large data sets across clusters of computers using a simple programming model.

BY-group processing
the process of using the BY statement to process observations that are ordered, grouped, or indexed according to the values of one or more variables. Many SAS procedures and the DATA step support BY-group processing. For example, you can use BY-group processing with the PRINT procedure to print separate reports for different groups of observations in a single SAS data set.

co-located data provider
a distributed data source, such as SAS Visual Analytics Hadoop or a third-party vendor database, that has SAS High-Performance Analytics software installed on the same machines. The SAS software on each machine processes the data that is local to the machine or that the data source makes available as the result of a query.

grid host
the machine to which the SAS client makes an initial connection in a SAS High-Performance Analytics application.

Hadoop Distributed File System
a framework for managing files as blocks of equal size, which are replicated across the machines in a Hadoop cluster to provide fault tolerance.

HDFS
See Hadoop Distributed File System

Message Passing Interface
is a message-passing library interface specification. SAS High-Performance Analytics applications implement MPI for use in high-performance computing environments.

MPI
See Message Passing Interface

root node
in a SAS High-Performance Analytics application, the role of the software that distributes and coordinates the workload of the worker nodes. In most deployments

the root node runs on the machine that is identified as the grid host. SAS High-Performance Analytics applications assign the highest MPI rank to the root node.

SASHDAT file
the data format used for tables that are added to HDFS by SAS. SASHDAT files are read in parallel by the server.

server description file
a file that is created by a SAS client when the LASR procedure executes to create a server. The file contains information about the machines that are used by the server. It also contains the name of the server signature file that controls access to the server.

signature file
small files that are created by the server to control access to the server and to the tables loaded in the server. There is one server signature file for each server instance. There is one table signature file for each table that is loaded into memory on a server instance.

worker node
in a SAS High-Performance Analytics application, the role of the software that receives the workload from the root node.

Index