SAS® LASR™ Analytic Server 2.2 Reference Guide
Gain Greater Insight into Your SAS® Software with SAS Books.

Discover all that you need on your journey to knowledge and empowerment.
# Contents

*What’s New In SAS LASR Analytic Server* ............................................. vii

**Chapter 1 • Introduction to the SAS LASR Analytic Server** .................. 1
  What is SAS LASR Analytic Server? ................................................. 2
  How Does the SAS LASR Analytic Server Work? .................................. 2
  Benefits of Using the Hadoop Distributed File System .......................... 4
  Components of the SAS LASR Analytic Server ................................... 5
  Administering the SAS LASR Analytic Server .................................... 6
  Memory Management .......................................................................... 11
  Data Partitioning and Ordering ....................................................... 13
  SAS LASR Analytic Server Logging ..................................................... 14

**Chapter 2 • Non-Distributed SAS LASR Analytic Server** .................... 19
  About Non-Distributed SAS LASR Analytic Server .............................. 19
  Starting and Stopping Non-Distributed Servers ................................. 19
  Loading and Unloading Tables for Non-Distributed Servers .................. 21

**Chapter 3 • LASR Procedure** ............................................................. 23
  Overview: LASR Procedure ............................................................... 23
  Syntax: LASR Procedure .................................................................. 24
  Examples: LASR Procedure ............................................................... 32

**Chapter 4 • IMSTAT Procedure** ......................................................... 41
  Overview: IMSTAT Procedure ............................................................ 42
  Concepts: IMSTAT Procedure ............................................................ 42
  Syntax: IMSTAT Procedure ............................................................... 46
  Examples: IMSTAT Procedure ............................................................ 75

**Chapter 5 • IMXFER Procedure** .......................................................... 89
  Overview: IMXFER Procedure ............................................................ 89
  Syntax: IMXFER Procedure ............................................................... 89
  Examples: IMXFER Procedure ............................................................ 93

**Chapter 6 • OLIPHANT Procedure** .................................................... 95
  Overview: OLIPHANT Procedure ...................................................... 95
  Concepts: OLIPHANT Procedure ...................................................... 96
  Syntax: OLIPHANT Procedure .......................................................... 97
  Examples: OLIPHANT Procedure ...................................................... 100

**Chapter 7 • VASMP Procedure** .......................................................... 103
  Overview: VASMP Procedure ........................................................... 103
  Syntax: VASMP Procedure .............................................................. 103
  Example: Copying Tables from One Hadoop Installation to Another ....... 108

**Chapter 8 • Using the SAS LASR Analytic Server Engine** ..................... 111
  What Does the SAS LASR Analytic Server Engine Do? ....................... 111
  Understanding How the SAS LASR Analytic Server Engine Works .......... 111
  Understanding Server Tags .................................................................. 112
  Comparing the SAS LASR Analytic Server Engine with the LASR Procedure . 112
<table>
<thead>
<tr>
<th>Content</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>What is Required to Use the SAS LASR Analytic Server Engine?</td>
<td>113</td>
</tr>
<tr>
<td>What is Supported?</td>
<td>113</td>
</tr>
<tr>
<td><strong>Chapter 9 • LIBNAME Statement for the SAS LASR Analytic Server Engine</strong></td>
<td>115</td>
</tr>
<tr>
<td>Dictionary</td>
<td>115</td>
</tr>
<tr>
<td><strong>Chapter 10 • Data Set Options for the SAS LASR Analytic Server Engine</strong></td>
<td>125</td>
</tr>
<tr>
<td>Dictionary</td>
<td>125</td>
</tr>
<tr>
<td><strong>Chapter 11 • Using the SAS Data in HDFS Engine</strong></td>
<td>131</td>
</tr>
<tr>
<td>What Does the SAS Data in HDFS Engine Do?</td>
<td>131</td>
</tr>
<tr>
<td>Understanding How the SAS Data in HDFS Engine Works</td>
<td>131</td>
</tr>
<tr>
<td>What is Required to Use the SAS Data in HDFS Engine?</td>
<td>132</td>
</tr>
<tr>
<td>What is Supported?</td>
<td>132</td>
</tr>
<tr>
<td><strong>Chapter 12 • LIBNAME Statement for the SAS Data in HDFS Engine</strong></td>
<td>133</td>
</tr>
<tr>
<td>Dictionary</td>
<td>133</td>
</tr>
<tr>
<td><strong>Chapter 13 • Data Set Options for the SAS Data in HDFS Engine</strong></td>
<td>139</td>
</tr>
<tr>
<td>Dictionary</td>
<td>139</td>
</tr>
<tr>
<td><strong>Chapter 14 • Programming with SAS LASR Analytic Server</strong></td>
<td>149</td>
</tr>
<tr>
<td>About Programming</td>
<td>149</td>
</tr>
<tr>
<td>DATA Step Programming for Scoring In SAS LASR Analytic Server</td>
<td>149</td>
</tr>
<tr>
<td><strong>Appendix 1 • Removing a Machine from the Cluster</strong></td>
<td>155</td>
</tr>
<tr>
<td>About Removing a Machine from the Cluster</td>
<td>155</td>
</tr>
<tr>
<td>Which Servers Can I Leave Running When I Remove a Machine?</td>
<td>155</td>
</tr>
<tr>
<td>Remove the Host Name from the grid.hosts File</td>
<td>156</td>
</tr>
<tr>
<td>Remove the Host Name from the Hadoop Slaves File</td>
<td>156</td>
</tr>
<tr>
<td>Restart Servers</td>
<td>156</td>
</tr>
<tr>
<td><strong>Appendix 2 • Adding Machines to the Cluster</strong></td>
<td>157</td>
</tr>
<tr>
<td>About Adding Machines to the Cluster</td>
<td>157</td>
</tr>
<tr>
<td>Which Servers Can I Leave Running When I Add a Machine?</td>
<td>158</td>
</tr>
<tr>
<td>Configure System Settings</td>
<td>158</td>
</tr>
<tr>
<td>Add Host Names to Gridhosts</td>
<td>159</td>
</tr>
<tr>
<td>Propagate Operating System User IDs</td>
<td>160</td>
</tr>
<tr>
<td>Configure SAS High-Performance Deployment of Hadoop</td>
<td>161</td>
</tr>
<tr>
<td>Configure SAS High-Performance Analytics Infrastructure</td>
<td>164</td>
</tr>
<tr>
<td>Restart SAS LASR Analytic Server Monitor</td>
<td>165</td>
</tr>
<tr>
<td>Restart Servers and Redistribute HDFS Blocks</td>
<td>165</td>
</tr>
<tr>
<td>View Explorations and Reports</td>
<td>165</td>
</tr>
<tr>
<td><strong>Glossary</strong></td>
<td>167</td>
</tr>
<tr>
<td><strong>Index</strong></td>
<td>169</td>
</tr>
</tbody>
</table>
What’s New In SAS LASR Analytic Server

Overview

SAS LASR Analytic Server 2.2 includes the following changes:

• Enhancements to the IMSTAT procedure
• The IMXFER procedure
• Enhancements to the SAS LASR Analytic Server engine
• Enhancements to the SAS Data in HDFS engine
• Limited DATA step programs can run in memory for scoring.
• Documentation enhancements

Enhancements to the IMSTAT Procedure

The IMSTAT procedure has the following enhancements:

• The SCHEMA statement can be used to join in-memory tables.
• The SET statement can be used to append entire in-memory tables to another in-memory table.
• The LIFETIME and PURGETEMPTABLES statements can be used to manage (and conserve) memory.

IMXFER Procedure

The IMXFER procedure is a new procedure that enables you to transfer in-memory tables from a distributed SAS LASR Analytic Server to another. The servers can be on the same cluster or different clusters. The number of machines in each cluster can be different.

The procedure works with distributed servers only.
Enhancements to the SAS LASR Analytic Server engine

The engine has the following enhancements:

• The VERBOSE= option is added. This option can be used to request more details for error messages related to connection problems.

• The AUTOCOMMIT= option is added. This option enables you to control how the server commits rows during append operations.

• The TEMPNAMES= option is enhanced to enable you to specify whether a variable is a numeric or character data type. For character data types, you can also specify the variable length.

Enhancements to the SAS Data in HDFS engine

The VERBOSE= option is added. This option is identical to the SAS LASR Analytic Server engine and can be used to help diagnose connection problems.

Support for DATA Step Programming for Scoring in SAS LASR Analytic Server

The DATA step is enhanced to support running in-memory for scoring. The DSACCEL=ANY SAS system option must be set. A subset of the DATA step syntax can be passed through to the server for parallel execution. Information about how to enable the feature and the limitations are provided in Chapter 14, “Programming with SAS LASR Analytic Server.”

Documentation Enhancements

The language element reference information for the HPDS2 procedure is removed. The reference information is available in the Base SAS Procedures Guide: High-Performance Procedures.

SAS LASR Analytic Server: Reference Guide is a new document that covers administration and programming information for SAS LASR Analytic Server.

The SAS LASR Analytic Server: Administration Guide is not published for this release.
Chapter 1
Introduction to the SAS LASR Analytic Server

What is SAS LASR Analytic Server? ............................... 2
How Does the SAS LASR Analytic Server Work? .................. 2
  Distributed SAS LASR Analytic Server .......................... 2
  Non-Distributed SAS LASR Analytic Server ....................... 4
Benefits of Using the Hadoop Distributed File System ............ 4
Components of the SAS LASR Analytic Server .................... 5
  About the Components ......................................... 5
  Root Node ...................................................... 5
  Worker Nodes .................................................. 5
  In-Memory Tables .............................................. 5
  Signature Files ................................................ 5
  Server Description Files ....................................... 6
Administering the SAS LASR Analytic Server ...................... 6
  Administering a Distributed Server ............................. 6
  Administering a Non-Distributed Server ......................... 6
  Common Administration Features ................................ 7
  Features Available in SAS Visual Analytics Administrator ....... 7
  Understanding Server Run Time ................................ 7
  Distributing Data .............................................. 8
Memory Management .................................................... 11
  About Physical and Virtual Memory ............................ 11
  How Does the Server Use Memory for Tables? .................. 11
  How Else Does the Server Use Memory? ........................ 12
  Managing Memory ............................................... 12
Data Partitioning and Ordering ........................................ 13
  Overview of Partitioning ....................................... 13
  Understanding Partition Keys .................................. 13
  Ordering within Partitions ..................................... 14
SAS LASR Analytic Server Logging ..................................... 14
  Understanding Logging ........................................ 14
  What is Logged? ............................................... 15
  Log Record Format .......................................... 15
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 of 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 a 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 lasrsrvr 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 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 deployments 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 131.

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.

The following code sample shows a LIBNAME statement and an example of the HPDS2 procedure for adding tables to Greenplum.

```sas
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";
```

Chapter 1 • Introduction to the SAS LASR Analytic Server
data DS2GTF.out;
  method run();
    set DS2GTF.in;
  end;
enddata;
run;

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.


**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;

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

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

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

    data tdlib.automation(bulkload=yes
      dbcommit=1000000
      dbcreate_table_opts='unique primary index(obsnum)');
    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.
Specify to use the variable named partno as the distribution key for the table.

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

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.

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.

```sas
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.

```sas
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 9.

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 EXTERNALMEM= 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, adding tables (including temporary ones), appending rows, or any other operation that consumes memory for storing data fails.

If the machine has already crossed the threshold, your requests to add data are immediately rejected. If you attempt to add a table and the server crosses the threshold as the data is added, the server removes the table you attempted to add and frees the memory. Similarly, if you attempt to append rows and the server crosses the threshold during the request, the entire append request fails. The table remains as it was before the append was attempted.

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 28
- “EXTERNALMEM=pct” on page 25

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 deployments. 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:

```sas
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 14.

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. However, this does 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 26
- “Example 2: Starting a Server with Logging Options” on page 33
- “Starting and Stopping Non-Distributed Servers” on page 19

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

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

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 nth 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 SASHIDAT 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=(
      path="c:\temp"
      keeplog=yes maxlogsize=20
   )
host=localhost
port=10010
   tag='hps';
```
1. The \texttt{STARTSERVER=} option indicates to start a server. For information about the options, see “\texttt{STARTSERVER= YES | NOSTARTSERVER =\text{(non-distributed-server-options)}}” on page 117.

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

3. The \texttt{HOST=} specification is optional.

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

Submitting the previous \texttt{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 \texttt{LIBNAME} statement with the following \texttt{VASMP} procedure statements:

\textbf{Example Code 2.2 \textit{SERVERWAIT Statement for the VASMP Procedure}}

\begin{verbatim}
proc vasmp;
  serverwait port=10010;
quit;
\end{verbatim}

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

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

\begin{verbatim}
libname server1 sasiola startserver=(path="/tmp")
  host="server.example.com" port=10010 tag='hps'
  signer="http://server.example.com/SASLASRAuthorization";
\end{verbatim}

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

If you want to use a script for starting a server, then include the \texttt{STARTSERVER=} \texttt{LIBNAME} option and the \texttt{SERVERWAIT} statement for the \texttt{VASMP} procedure in the program. Start one server only and do not include additional SAS statements after the \texttt{QUIT} statement for the \texttt{VASMP} procedure. If additional statements are included, they can prevent the SAS session from terminating (after receiving a \texttt{SERVERTERM} request). This can prevent the SAS session from freeing memory resources that were used by the server. It is best to restrict the program to starting the server only.

\textbf{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 \texttt{SERVERTERM} statement.

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

\textbf{Example Code 2.3 \textit{Stopping a Non-Distributed Server with the LIBNAME CLEAR Option}}

\begin{verbatim}
libname server1 clear;
\end{verbatim}
To stop a server from a different SAS session, use the SERVERTERM statement:

**Example Code 2.4  SERVERTERM Statement for the VASMP Procedure**

```sas
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**

```sas
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**

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

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

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

### 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 37.

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.

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 timeout 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

**TABLEPERMISIONS=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 34
- “Example 6: Unload a Table from Memory” on page 36

**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.

**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.

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

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

**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.

```
BLOCK=
```

**Alias**

```
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

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

proc lasr create port=10010 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

```sas
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 use FORMAT statements and data set options.

Program

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

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

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

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

proc lasr add data=grp1.sales2013(where=(region="West")) port=10010; 4
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.
4. The WHERE clause subsets the Sales2013 table. Only the rows with Region equal to "West" are read into memory. The WHERE clause is useful for subsetting data, but it does not take advantage of the memory efficiencies that are normally used with SASHDAT tables.

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 \texttt{SERVER=} option in the \texttt{LIBNAME} statement specifies the host name for the Teradata database.

2. The \texttt{HOST=} option in the \texttt{PERFORMANCE} statement specifies the host name of the Teradata Management Server (TMS).

3. The \texttt{DATASERVER=} option in the \texttt{PERFORMANCE} statement specifies the same host name for the Teradata database that is used in the \texttt{LIBNAME} statement.

4. The input data set option, \texttt{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 \texttt{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.

```sas
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 \texttt{SERVER=} option in the \texttt{LIBNAME} statement specifies the host name for the Greenplum database.

2. The \texttt{HOST=} option in the \texttt{PERFORMANCE} statement specifies the host name of the Greenplum master host.

3. The input data set option, \texttt{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 \texttt{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;  
    remove finance.trans;  
    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

```plaintext
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);

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

filename fmtxml 'genderfmt.xml';
libname fmtxml XML92 xmltype=sasfmt tagset=tagsets.XMLsuv;
proc format library=gendrfmt cntlout=fmtxml.allfmts;
run;
proc lasr add data=sashelp.class fmtlibxml=fmtxml;
   format sex $gender.;
   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
IMSTAT Procedure

Overview: IMSTAT Procedure .................................................. 42
What Does the IMSTAT Procedure Do? .................................. 42

Concepts: IMSTAT Procedure .................................................. 42
Partitioned Tables ................................................................. 42
RUN-Group Processing ......................................................... 43
WHERE Clause Processing ..................................................... 43
Temporary Tables ............................................................... 43
Creating Temporary Tables with GROUPBY Variables .................. 45
Creating Temporary Variables ................................................ 45

Syntax: IMSTAT Procedure ..................................................... 46
PROC IMSTAT Statement ....................................................... 47
BALANCE Statement ............................................................ 48
COLUMNINFO Statement ..................................................... 49
COMPUTE Statement .......................................................... 50
DELETEROWS Statement ...................................................... 50
DISTRIBUTIONINFO Statement ............................................. 51
DROPTABLE Statement ......................................................... 52
FETCH Statement ............................................................... 52
FREE Statement ............................................................... 54
IMPORTCUBE Statement ...................................................... 54
LIFETIME Statement .......................................................... 55
NUMROWS Statement ........................................................ 56
PARTITION Statement ........................................................ 56
PARTITIONINFO Statement .................................................. 57
PROMOTE Statement .......................................................... 58
PURGETEMPTABLES Statement ............................................... 59
REPLAY Statement ............................................................. 59
SAVE Statement ............................................................... 60
SCHEMA Statement ............................................................ 61
SCORE Statement ............................................................. 64
SERVERINFO Statement ...................................................... 67
SERVERPARM Statement ...................................................... 68
SERVERTERM Statement ...................................................... 69
SERVERWAIT Statement ....................................................... 69
SET Statement ................................................................. 70
STORE Statement ............................................................. 71
TABLE Statement ............................................................ 72
TABLEINFO Statement ....................................................... 72
UPDATE Statement ............................................................ 73
QUIT Statement ............................................................... 75
Examples: IMSTAT Procedure

Example 1: Partitioning a Table into a Temporary Table .......................... 75
Example 2: Promoting Temporary Tables to Regular Tables .................... 77
Example 3: Rebalancing a Table ............................................................ 78
Example 4: Deleting Rows and Saving a Table to HDFS ......................... 81
Example 5: Creating a Star Schema ..................................................... 82
Example 6: Appending Tables ............................................................. 84
Example 7: Appending a Non-Partitioned Table to a Partitioned Table ......... 85
Example 8: Storing Temporary Variables ............................................. 86

Overview: IMSTAT Procedure

What Does the IMSTAT Procedure Do?

The IMSTAT procedure is used to manage in-memory tables and SAS LASR Analytic Server instances.

Concepts: IMSTAT Procedure

Partitioned Tables

A SAS LASR Analytic Server table can be partitioned, and it can also be ordered within each partition. A partition is a collection of observations that share the same key value and are located on the same worker node. The key value can be constructed from one or more variables. The partitioning keys are created according to the formatted values of the specified variables. Partitioning of data is an important tool in managing distributed data sources. The process of partitioning can consume computing and network resources as well as create greater imbalance in the data compared to a round-robin distribution. However, partitioned data can be accessed more quickly and subsequent procedure statements can execute more quickly.

You can achieve a partitioned table as follows:

• load a table with the SAS LASR Analytic Server engine using the PARTITION= data set option
• load a SASHDAT file that has been previously partitioned into HDFS by using the PARTITION= data set option of the SAS Data in HDFS engine
• re-partition data from one table into a temporary table (you can then make it a regular table with the PROMOTE statement)
• create a temporary table with a GROUPBY= option. The temporary table is partitioned by the formatted values of the GROUPBY= variables.

For more information, see “Data Partitioning and Ordering” on page 13.
**RUN-Group Processing**

The IMSTAT procedure supports RUN-group processing. RUN-group processing enables you to submit RUN groups without ending the procedure. This feature is particularly useful for running SAS interactively. You can start the procedure with the PROC IMSTAT statement and then execute statements like TABLE and PROMOTE. Each statement runs when it reaches a RUN statement.

To use RUN-group processing, you start the procedure and then submit multiple RUN-groups. A RUN-group is a group of statements that contains at least one action statement and ends with a RUN statement. As long as you do not terminate the procedure, it remains active and you do not need to resubmit the PROC statement.

To end RUN-group processing, submit a RUN CANCEL statement. Statements that have not been submitted are terminated. To stop the procedure, submit a QUIT statement. Statements that have not been submitted are terminated as well.

**WHERE Clause Processing**

There are two important features for WHERE clause processing that are related to the IMSTAT procedure. The first is that the WHERE clause is applied to the data by the server. When you use a WHERE clause to subset data, the subsetting is performed by the server and only the rows that meet the WHERE clause criteria are affected by subsequent operations.

The second important feature for WHERE clause processing is related to the RUN-group processing that the IMSTAT procedure supports. You can modify the WHERE clause between statements.

For an example of the IMSTAT procedure with a WHERE clause, see “Example 4: Deleting Rows and Saving a Table to HDFS” on page 81.

**Temporary Tables**

A temporary table is an in-memory table that contains the result set of a procedure statement. Instead of transferring the results to the client SAS session, the results remain in the server and only the name of the temporary table is transferred to the client. You can then use other procedure statements with the temporary table.

Temporary tables are supported for the SUMMARY, CROSSTAB, DISTINCT, and PERCENTILE statements for partitioned data. For non-partitioned data, you can also generate temporary tables with the SUMMARY and CROSSTAB statements, provided that you request a GROUPBY analysis.

The following DATA step shows how to create a partitioned table on the variables country and region.

```sas
data lasr.prdsale(partition=(country region));
set sashelp.prdsale;
run;
```

The following statements generate a summary analysis for variables actual and predict in each of the partitions.

```sas
proc imstat;
  table lasr.prdsale;
  summary actual predict / partition;
run;
```
The output for the previous statements is as follows:

```sas
The SAS System
The IMSTAT Procedure

Summary Statistics for Table WORK.PROSALE

<table>
<thead>
<tr>
<th>COUNTRY</th>
<th>REGION</th>
<th>Column</th>
<th>Min</th>
<th>Max</th>
<th>N</th>
<th>Sum</th>
<th>Mean</th>
<th>Std Dev</th>
<th>Std Error</th>
<th>Coefficient of Variation</th>
<th>Number Missing</th>
</tr>
</thead>
<tbody>
<tr>
<td>CANADA</td>
<td>EAST</td>
<td>ACTUAL</td>
<td>5.000</td>
<td>999.00</td>
<td>240</td>
<td>127485</td>
<td>531.19</td>
<td>281.61</td>
<td>18.1711</td>
<td>53.1616</td>
<td>0</td>
</tr>
<tr>
<td>CANADA</td>
<td>EAST</td>
<td>PREDICT</td>
<td>0</td>
<td>986.00</td>
<td>240</td>
<td>120846</td>
<td>562.69</td>
<td>284.29</td>
<td>18.3610</td>
<td>66.8540</td>
<td>0</td>
</tr>
<tr>
<td>CANADA</td>
<td>WEST</td>
<td>ACTUAL</td>
<td>3.000</td>
<td>1000.00</td>
<td>240</td>
<td>119505</td>
<td>497.94</td>
<td>296.53</td>
<td>19.1409</td>
<td>59.5507</td>
<td>0</td>
</tr>
<tr>
<td>CANADA</td>
<td>WEST</td>
<td>PREDICT</td>
<td>6.000</td>
<td>1000.00</td>
<td>240</td>
<td>112373</td>
<td>466.22</td>
<td>275.96</td>
<td>17.9103</td>
<td>69.5443</td>
<td>0</td>
</tr>
<tr>
<td>GERMANY</td>
<td>EAST</td>
<td>ACTUAL</td>
<td>13.000</td>
<td>1030.00</td>
<td>240</td>
<td>124547</td>
<td>518.95</td>
<td>287.71</td>
<td>18.5114</td>
<td>66.4406</td>
<td>0</td>
</tr>
<tr>
<td>GERMANY</td>
<td>EAST</td>
<td>PREDICT</td>
<td>4.000</td>
<td>993.00</td>
<td>240</td>
<td>117579</td>
<td>489.91</td>
<td>292.43</td>
<td>18.8762</td>
<td>69.6901</td>
<td>0</td>
</tr>
<tr>
<td>GERMANY</td>
<td>WEST</td>
<td>ACTUAL</td>
<td>3.000</td>
<td>996.00</td>
<td>240</td>
<td>121451</td>
<td>566.05</td>
<td>289.17</td>
<td>18.6663</td>
<td>57.1429</td>
<td>0</td>
</tr>
<tr>
<td>GERMANY</td>
<td>WEST</td>
<td>PREDICT</td>
<td>0</td>
<td>981.00</td>
<td>240</td>
<td>113975</td>
<td>474.50</td>
<td>280.49</td>
<td>18.1066</td>
<td>69.0637</td>
<td>0</td>
</tr>
<tr>
<td>U.S.A.</td>
<td>EAST</td>
<td>ACTUAL</td>
<td>4.000</td>
<td>984.00</td>
<td>240</td>
<td>118229</td>
<td>492.62</td>
<td>292.26</td>
<td>18.2200</td>
<td>67.2883</td>
<td>0</td>
</tr>
<tr>
<td>U.S.A.</td>
<td>EAST</td>
<td>PREDICT</td>
<td>1.000</td>
<td>1003.00</td>
<td>240</td>
<td>126587</td>
<td>502.45</td>
<td>301.35</td>
<td>19.4518</td>
<td>59.5957</td>
<td>0</td>
</tr>
<tr>
<td>U.S.A.</td>
<td>WEST</td>
<td>ACTUAL</td>
<td>6.000</td>
<td>984.00</td>
<td>240</td>
<td>119120</td>
<td>496.33</td>
<td>286.67</td>
<td>18.4401</td>
<td>67.6867</td>
<td>0</td>
</tr>
<tr>
<td>U.S.A.</td>
<td>WEST</td>
<td>PREDICT</td>
<td>22.000</td>
<td>999.00</td>
<td>240</td>
<td>121135</td>
<td>504.73</td>
<td>280.10</td>
<td>18.0801</td>
<td>65.4944</td>
<td>0</td>
</tr>
</tbody>
</table>
```

As an alternative, you can leave the result set in an in-memory table by adding the TEMPTABLE option to the SUMMARY statement:

```sas
summary actual predict / partition temptable;
run;
```

The previous SAS statements generate the following output in the SAS session.

```sas
The SAS System
The IMSTAT Procedure

Temporary Table Information for Table WORK.PROSALE

<table>
<thead>
<tr>
<th>Statement</th>
<th>SUMMARY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Temporary Table</td>
<td>_T_TEMP7387_10F2B178</td>
</tr>
</tbody>
</table>
```

The temporary table is assigned a name by the server. When the IMSTAT procedure ends, any temporary tables created during the procedure run are removed from the server. Since the generated name is not predictable, the procedure assigns the name of the most recently generated temporary table to the _TEMPLAST_ macro variable.

You can use the TABLE statement to switch the active table to the temporary table and perform analyses. Make sure that the statement that generated the temporary table is separated from the next statement with a RUN statement. Otherwise, you receive an error that the table specified in the TABLE statement does not exist. The temporary table does not exist at parse time, it is created at run time when the statement is executed.

The following statements retrieve information about the temporary table, the formatted values for (up to) the first twenty rows, and perform a summarization:

```sas
table lasr._templast_
tableinfo;
columninfo;
fetch / from=1 to=20 format;
summary;
quit;
```
Creating Temporary Tables with GROUPBY Variables

If the input table is not partitioned, you can still use temporary tables with the SUMMARY and CROSSTAB statements, provided that you perform a group-by analysis. The temporary table that is created by the server is automatically partitioned by the group-by variables. This potentially involves a redistribution of the groups in the result set to transfer all the result records for a particular group to the same worker node.

Creating Temporary Variables

You can use temporary numeric variables in a table. For example, if a table has variables that are named x1, x2, and x3, you can calculate the summary statistics for variable d1 = x1 + x2 / 3*x3. One way is to declare d1 as a temporary variable of the table (with data set options for the input table). You can use the temporary variables (temporary for the duration of each statement) with DATA step expression scripts.

```
proc imstat data=lasrlib.table1(array=(d,1));
   summary d1 x1-x3 / tempnames=d1 tempexpress="d1 = x1 + x2 / 3*x3;";
run;
   summary d1 / tempnames=d1 tempexpress="d1 = mean(x1, x2);";
quit;
```

Because the temporary variable exists for the duration of the statement, it can be reused in subsequent statements. The second SUMMARY statement uses the d1 variable with a different expression.
Syntax: IMSTAT Procedure

**PROC IMSTAT** &lt;options&gt;;
  **BALANCE** &lt;/options&gt;;
  **COLUMNINFO** &lt;/options&gt;;
  **COMPUTE** column-name "SAS-statements" &lt;/ option&gt;;
  **DELETEROWS** &lt;/options&gt;;
  **DISTRIBUTIONINFO** &lt;/ option&gt;;
  **DROPTABLE** &lt;libref.member-name&gt;;
  **FETCH** &lt;variable-list&gt; &lt;/ options&gt;;
  **FREE** resource-specification;
  **IMPORTCUBE** &lt;options&gt;;
  **LIFETIME** time-specification &lt;/ MODE= ABSOLUTE | LASTUSE &gt;;
  **NUMROWS**;
  **PARTITION** variable-list &lt;/ options&gt;;
  **PARTITIONINFO** &lt;/ options&gt;;
  **PROMOTE** member-name &lt;/ options&gt;;
  **PURGETEMPTABLES** &lt;options&gt;;
  **REPLAY** table-list;
  **SAVE** &lt;options&gt;;
  **SCHEMA** dim-specification1 &lt;dim-specification2 ...&gt; &lt;/ options&gt;;
  **SCORE** CODE= file-reference &lt;options&gt;;
  **SERVERINFO** &lt;option&gt;;
  **SERVERPARM** &lt;options&gt;;
  **SERVERTERM** &lt;options&gt;;
  **SERVERWAIT** &lt;options&gt;;
  **SET** set-specification1 &lt;set-specification2 ...&gt; &lt;/ options&gt;;
  **STORE** table-name(row &lt; ,&gt; column) = macro-variable-name;
  **TABLE** &lt;libref.member-name&gt;;
  **TABLEINFO** &lt;/ options&gt;;
  **UPDATE** variable1=value1 &lt;variable2=value2 ...&gt; &lt;/options&gt;;
**QUIT**;

<table>
<thead>
<tr>
<th>Statement</th>
<th>Task</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>BALANCE</td>
<td>Rebalancing a table</td>
<td>Ex. 3</td>
</tr>
<tr>
<td>DELETEROWS</td>
<td>Deleting rows and saving a table to HDFS</td>
<td>Ex. 4</td>
</tr>
<tr>
<td>PARTITION</td>
<td>Partitioning a table into a temporary table</td>
<td>Ex. 1</td>
</tr>
<tr>
<td>PROMOTE</td>
<td>Promoting temporary tables to regular tables</td>
<td>Ex. 2</td>
</tr>
<tr>
<td>SCHEMA</td>
<td>Creating a star schema</td>
<td>Ex. 5</td>
</tr>
</tbody>
</table>
PROC IMSTAT Statement

Manages in-memory tables in a SAS LASR Analytic Server instance.

Syntax

PROC IMSTAT <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.

FMTLIBXML=file-reference
  specifies the file reference for a format stream. For more information, see “FMTLIBXML” in the LASR procedure.

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

  Alias SINGLESTEP

NODATE
  specifies to suppress the display of time and date-dependent information in results from the TABLEINFO statement.

NOPREPARSE
  specifies to prevent the procedure from pre-parsing and pre-generating code for temporary expressions, scoring programs, and other user-written SAS statements.

  When this option is specified, the user-written statements are sent to the server "as-is" and then the server attempts to generate code from it. If the server detects problems with the code, the error messages might not to be as detailed as the messages that are generated by SAS client. If you are debugging your user-written program, then you might want to pre-parse and pre-generate code in the procedure. However, if your SAS statements compile and run as you want them to, then you can specify this option to avoid the work of parsing and generating code on the SAS client.

  When you specify this option in the PROC IMSTAT statement, the option applies to all statements that can generate code. You can also exclude specific statements from pre-parsing by using the NOPREPARSE option in statements that allow temporary columns or the SCORE statement.

  Alias NOPREP

NOPRINT
  This option suppresses the generation of ODS tables and other printed output in the IMSTAT procedure. You can use this option to suppress printed output during
execution of the actions, and then use the REPLAY statement to print the tables at a later point in the procedure execution.

**NOTIMINGMSG**

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

*Alias* NOTIME

**PGMMSG**

specifies to capture messages associated with user-provided SAS statements on the server in a temporary table. Messages are produced when parsing errors occur, when code generation fails, or by PUT statements in a SAS program.

You can use this option as a debugging feature for SAS code that you submit through temporary column expressions. The macro variable _PGMMSG_ is used in the IMSTAT procedure to capture the name of the table. The _TEMPLAST_ macro variable is also updated in case this temporary table is the most recently created temporary table.

*Alias* PROGMST

**SIGNER="authorization-web-service-uri"**

specifies the URI for the SAS LASR Authorization web service. For more information, see *SAS Visual Analytics: Administration Guide*.

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

**TEMPTABLEINFO**

specifies to add additional information for temporary tables to the ODS table that is created on the SAS client. The information includes the time at which the temporary table was created in the server, the number of rows, and the number of columns.

*Alias* TEMPINFO

**UCA**

specifies that you want to use Unicode Collation Algorithms (UCA) to determine the ordering of character variables in the GROUPBY= operations and other operations that depend on the order of formatted values.

*Alias* UCACOLLATION

---

**BALANCE Statement**

The BALANCE statement creates a temporary table from the active table and re-balances it so that the number of rows on the worker nodes are balanced as evenly as possible. The rows are balanced within ±1 row of each other.

*Example:* “Example 3: Rebalancing a Table” on page 78

**Syntax**

BALANCE;
Without Arguments
The re-balancing removes any observations marked as deleted or marked for purging in
the active table. A WHERE clause is observed when the data are re-balanced.

One case for re-balancing is if the data distribution for a table has become uneven due to
block movement within the Hadoop Distributed File System. This can occur when nodes
fail in Hadoop or Hadoop processes have exited on some nodes. Another situation where
re-balancing is useful is when a partitioned table has uneven distribution across the
worker nodes due to uneven sizes of the partition. This can affect the performance of all
actions running in the LASR Analytic Server since typically the nodes with the most
records determine the overall performance.

Re-balancing of a table removes partition and ordering information from the table.
The BALANCE statement can be used with non-distributed servers as well. However, it
is less important because all records of a table reside on the same machine. It might be
useful, however, to derive from a partitioned table a new table subject to a WHERE
clause and that has deleted records removed and is not partitioned.

COLUMNINFO Statement
The COLUMNINFO statement is used to return information for all the columns in an in-memory table.

Syntax
COLUMNINFO <options>;

COLUMNINFO Statement Options
SAVE=table-name
saves the result table so that you can use it in other IMSTAT procedure statements
like STORE, REPLAY, and FREE. The value for table-name must be unique within
the scope of the procedure execution. The name of a table that has been freed with
the FREE statement can be used again in subsequent SAVE= options.

SETSIZE
requests that the server estimate the size of the result set. The procedure does not
create a result table if the SETSIZE option is specified. Instead, the procedure reports
the number of rows that are returned by the request and the expected memory
consumption for the result set (in KB). If you specify the SETSIZE option, the SAS
log includes the number of observations and the estimated result set size. See the
following log sample:

NOTE: The LASR Analytic Server action request for the STATEMENT
statement would return 17 rows and approximately
3.641 kBytes of data.

The typical use of the SETSIZE option is to get an estimate of the size of the result
set in situations where you are unsure whether the SAS session can handle a large
result set. Be aware that in order to determine the size of the result set, the server has
to perform the work as if you were receiving the actual result set. Requesting the
estimated size of the result set does consume resources on the server. The estimated
number of KB is very close to the actual memory consumption of the result set. It
might not be immediately obvious how this size relates to the displayed table, since
many tables contain hidden columns. In addition, some elements of the result set
might not be converted to tabular output by the procedure.
COMPUTE Statement

The COMPUTE statement adds a permanent computed column to an in-memory table.

Syntax

COMPUTE column-name file-reference < / option > ;
COMPUTE column-name "SAS-statements" < / option > ;

Required Argument

column-name

specifies the name to use for the computed column. The name cannot already be in use in the table.

COMPUTE Statement Options

NOPREPARSE

specifies to prevent the procedure from pre-parsing and pre-generating code for temporary expressions, scoring programs, and other user-written SAS statements.

When this option is specified, the user-written statements are sent to the server "as-is" and then the server attempts to generate code from it. If the server detects problems with the code, the error messages might not to be as detailed as the messages that are generated by SAS client. If you are debugging your user-written program, then you might want to pre-parse and pre-generate code in the procedure. However, if your SAS statements compile and run as you want them to, then you can specify this option to avoid the work of parsing and generating code on the SAS client.

When you specify this option in the PROC IMSTAT statement, the option applies to all statements that can generate code. You can also exclude specific statements from pre-parsing by using the NOPREPARSE option in statements that allow temporary columns or the SCORE statement.

Alias

NOPREP

DELETEROWS Statement

The DELETEROWS statement is used to mark rows as deleted, to undelete rows, and to purge rows from an in-memory table. Rows that are marked for deletion or purging are not included in the calculations performed by the server.

Interaction:

If a WHERE clause is active, the rows that meet the criteria are marked for deletion or purging. When no WHERE clause is active, a delete request marks all rows for deletion (they can be undeleted), but the PURGE option removes the rows that are already marked for deletion rather than removing all rows.

Example:

"Example 4: Deleting Rows and Saving a Table to HDFS" on page 81
**Syntax**

DELETEROWS /* options */;

**DELETEROWS Statement Options**

**PURGE**

Specifies to remove from memory the rows that are marked for deletion. The memory that was used by the rows is freed. The purged rows cannot be undeleted. One use case for purging rows is to remove older records from an in-memory table after new records were appended to the table.

If a WHERE clause is active, the rows that meet the criteria are purged. If no WHERE clause is active, then the PURGE request removes the rows that are already marked for deletion. It does not remove all the rows in the table. This was implemented to prevent accidentally removing all rows from a table.

**SAVE=table-name**

Saves the result table so that you can use it in other IMSTAT procedure statements like STORE, REPLAY, and FREE. The value for table-name must be unique within the scope of the procedure execution. The name of a table that has been freed with the FREE statement can be used again in subsequent SAVE= options.

**TEMPEXPRESS="SAS-expressions"**

**TEMPEXPRESS=file-reference**

Specifies either a quoted string that contains the SAS expression that defines the temporary variables or a file reference to an external file with the SAS statements.

Alias **TE**=

**TEMPNAMES=variable-name**

**TEMPNAMES=(variable-list)**

Specifies the list of temporary variables for the request. Each temporary variable must be defined through SAS statements that you supply with the TEMPEXPRESS= option.

Alias **TN**=

**UNDELETE**

Specifies to clear the deletion mark from rows.

If a WHERE clause is active, only the rows that meet the criteria have the deletion mark cleared. A row that has been marked for purging from the table cannot be undeleted.

---

**DISTRIBUTIONINFO Statement**

The DISTRIBUTIONINFO statement returns the number of partitions and the number of records on each worker node in the SAS LASR Analytic Server instance. This information provides the approximate distribution characteristics of the data across the worker nodes. If you want more details about the data distribution, then use the PARTITIONINFO statement.

**Syntax**

DISTRIBUTIONINFO /* option */;
**DISTRIBUTIONINFO Statement Options**

**SAVE=table-name**

Saves the result table so that you can use it in other IMSTAT procedure statements like STORE, REPLAY, and FREE. The value for `table-name` must be unique within the scope of the procedure execution. The name of a table that has been freed with the FREE statement can be used again in subsequent SAVE= options.

**DROPTABLE Statement**

The DROPTABLE statement is used to remove an in-memory table from a server. You must specify a table as an active table with the DATA= procedure option or in a TABLE statement before you can use the DROPTABLE statement. Once a table is active, you can specify that table, another table, or a temporary table.

**Syntax**

```
DROPTABLE <libref.member-name>;
```

**Optional Argument**

*libref.member-name*

specifies the name of the in-memory table to remove from the server. If you do not specify the table, then the active table is dropped.

**FETCH Statement**

The FETCH statement is used to retrieve rows from an in-memory table. You can use the FETCH statement to retrieve calculated columns that are calculated according to a script as part of the request. The columns that are calculated this way do not persist beyond the time it takes to execute in the server.

**Syntax**

```
FETCH <variable-list> </options>;
```

**Optional Argument**

*variable-list*

specifies the numeric and character variables to retrieve.

**FETCH Statement Options**

**ARRAYSTART=n**

specifies the starting element of an array when the record of an in-memory table represents a variable array. This is the case, for example, when a pricing cube from SAS High-Performance Risk is loaded into a server. There might then be 10,000 columns for a variable. Specifying the ARRAYSTART= and ARRAYLENGTH= options enables you to page through the data more conveniently.
ARRAYLENGTH=k
specifies the length of the array to fetched when the record of an in-memory table
represents a variable array. Use this option with the ARRAYSTART= option.

FORMAT
specifies that you want to retrieve formatted values. By default, the FETCH
statement retrieves the raw values. Be aware that when you retrieve raw values and
you create an output data set, the variable information such as formats and labels are
transferred to the output data set.

FROM=first-index
specifies the index of the first row to retrieve (inclusive). The value for first-index is
1-based.

Default FROM=1

Interaction The value for FROM= is applied after the evaluation of a WHERE
clause.

OUT=libref.member-name
specifies the name of the data set to store the result set of the FETCH statement.

SAVE=table-name
saves the result table so that you can use it in other IMSTAT procedure statements
like STORE, REPLAY, and FREE. The value for table-name must be unique within
the scope of the procedure execution. The name of a table that has been freed with
the FREE statement can be used again in subsequent SAVE= options.

SETSIZE
requests that the server estimate the size of the result set. The procedure does not
create a result table if the SETSIZE option is specified. Instead, the procedure reports
the number of rows that are returned by the request and the expected memory
consumption for the result set (in KB). If you specify the SETSIZE option, the SAS
log includes the number of observations and the estimated result set size. See the
following log sample:

NOTE: The LASR Analytic Server action request for the STATEMENT
statement would return 17 rows and approximately
3.641 kBytes of data.

The typical use of the SETSIZE option is to get an estimate of the size of the result
set in situations where you are unsure whether the SAS session can handle a large
result set. Be aware that in order to determine the size of the result set, the server has
to perform the work as if you were receiving the actual result set. Requesting the
estimated size of the result set does consume resources on the server. The estimated
number of KB is very close to the actual memory consumption of the result set. It
might not be immediately obvious how this size relates to the displayed table, since
many tables contain hidden columns. In addition, some elements of the result set
might not be converted to tabular output by the procedure.

TEMPEXPRESS="SAS-expressions"

specifies either a quoted string that contains the SAS expression that defines the
temporary variables or a file reference to an external file with the SAS statements.
TEMPNAMES=variable-name
TEMPNAMES=(variable-list)
specifies the list of temporary variables for the request. Each temporary variable
must be defined through SAS statements that you supply with the TEMPEXPRESS=
option.

Alias TN=

TO=last-index
specifies the index of the last row to retrieve (inclusive). The value for last-index is
1-based.

Default The default value is FROM=first-index + 19.

Interaction The value for TO= is applied after the evaluation of a WHERE clause.

FREE Statement
The FREE statement is used to release resources from STORE statements or SAVE= options. If you save
a result table (one table or sets of tables) with the SAVE= option in any of the analytic statements of the
IMSTAT procedure, then you can release the resources with the FREE statement. Once a table has been
freed, you can reuse the table name. While a saved table exists, you cannot create a table by the same
name.

Syntax
FREE table-list;
FREE _ALL_;
FREE TABLE=one-table;
FREE MACRO=macro-variable-name;

FREE Statement Options
table-list
specifies a list of tables to release.

_ALL_
specifies to release the resources for all the tables that were saved throughout the
procedure execution.

TABLE=one-table
specifies the name of the table to release.

MACRO=macro-variable-name
specifies the name of a macro variable to release.

IMPORTCUBE Statement
The IMPORTCUBE statement is used to read a cube descriptor file for a SAS High-Performance Risk
pricing cube. The server converts the pricing information that is in the cube into an in-memory table.
Syntax

IMPORTCUBE <options>;

**IMPORTCUBE Statement Options**

**CUBE="path-to-descriptor-file"**
specifies the SAS High-Performance Risk descriptor file for the pricing cube. The procedure opens the descriptor file and examines the contents. If the file is not a cube descriptor file for a SAS High-Performance Risk cube or the version of the file does not match the ISTAT procedure version, the procedure writes an error to the SAS log and stops processing statements.

**HOST="host-name"**
specifies the machine to use for exporting the SAS High-Performance Risk cube. If you do not specify the host, the procedure examines the GRIDHOST environment variable. You can use the PORT= and HOST= options to specify the server to use. The procedure does not determine the host and port information for the IMPORTCUBE statement from the currently active table.

**PORT=number**
specifies the port on which the server listens for connections. You can combine the PORT= and HOST= options to specify the server to use. If you do not specify the PORT= option the procedure uses the port number that is stored in the LASRPORT macro variable. The procedure does not determine the host and port information for the IMPORTCUBE statement from the currently active table.

**TABLENAME="table-name"**
specifies the name of the in-memory table to use for the imported cube. A table by the same name must not already exist in the server.

**LIFETIME Statement**

The LIFETIME statement enables you to associate an expiration date with an in-memory table. You must have sufficient authorization, which is equivalent to the authorization to drop the table. The server checks periodically if any tables have expired and drops the expired ones. This frees all resources associated with those tables.

**Syntax**

LIFETIME time-specification </ MODE= ABSOLUTE | LASTUSE >;

**Required Argument**

**time-specification**
specifies the duration (in seconds) that the table is to remain in memory. The minimum value is 1 second.
LIFETIME Statement Options

MODE=ABSOLUTE | LASTUSE
specifies how to use the time-specification. If MODE=ABSOLUTE is specified, then the server drops the table after the specified number of seconds. If MODE=LASTUSE is specified, then the server drops the table the specified number of seconds after the last successful access to the table.

Default ABSOLUTE

NUMROWS Statement

The NUMROWS statement identifies how many rows satisfy a selection criterion. The selection observes the WHERE clause and records marked for deletion or purging.

Syntax

NUMROWS;

PARTITION Statement

The PARTITION statement is used to generate a temporary table from the active table and partition it according to the statement options. This re-partitioning of tables is supported for distributed and non-distributed SAS LASR Analytic Server.

Examples:
“Example 1: Partitioning a Table into a Temporary Table” on page 75
“Example 7: Appending a Non-Partitioned Table to a Partitioned Table” on page 85

Syntax

PARTITION variable-list / options;

PARTITION Statement Options

DESCENDING=(variable-list)
DESCENDING=variable-name
specifies the variables in the ORDERBY= list to use with a descending sort order. Specifying the DESCENDING= option by itself has no effect on ordering within a partition. The option is specified in addition to the ORDERBY= option.

Alias DESC=

Example

The following statement requests partitioning of the active table by variables A and B, and ordering within the partition by ascending value of column C and descending value of column D:

partition a b / orderby=(c d) descending=d;

FORMATS=(format-specification<,..>)
specifies the formats for the PARTITIONBY= variables. If you do not specify the FORMATS= option, the default format is applied for that variable. The format of a
partitioning variable is important because the equality of the partition keys is determined from the formatted values.

Enclose each format specification in quotation marks and separate each format specification with a comma.

**NOMISSING**

specifies that missing values are excluded in the determination of partition keys. By default, observations with missing values are included.

Alias NOMISS

**ORDERBY**=(variable-list)

specifies the variable or variables to use for ordering the observations within a partition. By default, the sort order for the ORDERBY= variables is ascending in raw values and follows location and collation rules. If you want to order some ORDERBY= variables in descending sort order, then specify the variable names in the DESCENDING= option (in addition to listing them in the ORDERBY= option).

The ORDERBY= variables are transferred automatically to the partitioned temporary table, whether you list them in the VARS= option or not.

**SAVE**=table-name

saves the result table so that you can use it in other IMSTAT procedure statements like STORE, REPLAY, and FREE. The value for table-name must be unique within the scope of the procedure execution. The name of a table that has been freed with the FREE statement can be used again in subsequent SAVE= options.

**TEMPEXPRESS**="SAS-expressions"

specifies either a quoted string that contains the SAS expression that defines the temporary variables or a file reference to an external file with the SAS statements.

Alias TE=

**TEMPNAMES**=variable-name

specifies the list of temporary variables for the request. Each temporary variable must be defined through SAS statements that you supply with the TEMPEXPRESS= option.

Alias TN=

**VARIABLES**=(variable-list)

specifies the variable or variables to include in the temporary table in addition to the partitioning variables. If you do not specify the VARS= option, then all the variables are transferred from the active table. Temporary calculated columns are also transferred to the temporary table.

Alias VARS=

---

**PARTITIONINFO Statement**

The PARTITIONINFO statement produces very detailed information about the partitions of the data for each node of the server. Be aware that depending on the number of partitions, the result table can be very
large. If you use the statement on non-partitioned data, it reports the number of bytes and records for each node. These results are similar to the result set you get from a DISTRIBUTIONINFO statement.

Syntax

PARTITIONINFO <\ options>;  

**PARTITIONINFO Statement Options**

SAVE=table-name

saves the result table so that you can use it in other IMSTAT procedure statements like STORE, REPLAY, and FREE. The value for table-name must be unique within the scope of the procedure execution. The name of a table that has been freed with the FREE statement can be used again in subsequent SAVE= options.

SETSIZE

requests that the server estimate the size of the result set. The procedure does not create a result table if the SETSIZE option is specified. Instead, the procedure reports the number of rows that are returned by the request and the expected memory consumption for the result set (in KB). If you specify the SETSIZE option, the SAS log includes the number of observations and the estimated result set size. See the following log sample:

```
NOTE: The LASR Analytic Server action request for the STATEMENT  
statement would return 17 rows and approximately 3.641 kBytes of data.
```

The typical use of the SETSIZE option is to get an estimate of the size of the result set in situations where you are unsure whether the SAS session can handle a large result set. Be aware that in order to determine the size of the result set, the server has to perform the work as if you were receiving the actual result set. Requesting the estimated size of the result set does consume resources on the server. The estimated number of KB is very close to the actual memory consumption of the result set. It might not be immediately obvious how this size relates to the displayed table, since many tables contain hidden columns. In addition, some elements of the result set might not be converted to tabular output by the procedure.

**PROMOTE Statement**

The PROMOTE statement is used to change a temporary table to a regular in-memory table. The currently active table must be a temporary table and the table identified with the member-name parameter must not already exist in the server. Promoting a temporary table requires authorization to add a table to the server. You can also specify a tag for the in-memory table with the TAG= option.

Example:  
“Example 5: Creating a Star Schema” on page 82

Syntax

PROMOTE member-name <\ options>;

**Required Argument**

member-name

specifies the name to use for the table that is promoted.
**PROMOTE Statement Options**

**PERM=mode**

specifies the access permission for the newly created table as an integer. 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. You must specify a value that preserves Read and Write permission for your user ID.

**Alias** PERMISSION=

**Range** 600 to 777

**TAG='server-tag'**

specifies the tag to use for naming the table. If no TAG= option is specified, 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.

---

**PURGETEMPTABLES Statement**

The PURGETEMPTABLES removes all temporary tables from a server. The action requires server-level authorization because it removes temporary tables created by all users. To execute this command successfully, you must have the same authorization that is required to terminate the server.

**Syntax**

```
PURGETEMPTABLES <options>;
```

**Without Arguments**

The server to use is identified from the active table. If you do not have an active table, then you can connect to a specific server with the HOST= and PORT= options.

**PURGETEMPTABLES Statement Options**

**HOST='host-name'**

specifies the host name for the server. Use this option with the PORT= option.

**PORT=number**

specifies the port number for the server.

---

**REPLAY Statement**

The REPLAY statement enables you to display a previously saved result table or set of tables. The REPLAY statement displays the saved result tables regardless of the NOPRINT option. This enables you to suppress output generation with the NOPRINT option and to then display the tables that you want in a different order.

**Syntax**

```
REPLAY table-list;
```
Optional Argument

table-list
  specifies the saved result tables to display.

Example: Display Result Tables in a Different Order

The following SAS statements suppress the display of output with the NOPRINT option. Then, the tables are displayed in the reverse order.

```sas
proc imstat data=sales.prdsale noprint;
  fetch country region actual / save=salestab from=1 to=5;
  fetch predict actual / save=predicttab from=1 to=10;
  replay predicttab salestab;
quit;
```

SAVE Statement

The SAVE statement enables you to save an in-memory table to HDFS. The table must be the active table. You specify the active table with the DATA= option in the IMSTAT procedure or with the TABLE statement.

Example:  "Example 4: Deleting Rows and Saving a Table to HDFS" on page 81

Syntax

SAVE <options>;

Optional Arguments

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

Interaction
  If the in-memory table is partitioned, the BLOCKSIZE= specification is ignored. The server determines the block size based on the size of the partitions.

COPIES=n
  specifies the number of replications to make for the data set (beyond the original blocks). The default value is 1. You can specify COPIES=0 if you do not need replications for redundancy.

FULLPATH
  specifies that the value for the PATH= option specifies the full path for the file, including the filename (without the SASHDAT extension). The filename portion of the quoted string is expected to be in lowercase characters.

PATH="HDFS-path"
  specifies the directory in HDFS in which to store the table as a 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.

SAVE=table-name
saves the result table so that you can use it in other IMSTAT procedure statements like STORE, REPLAY, and FREE. The value for table-name must be unique within the scope of the procedure execution. The name of a table that has been freed with the FREE statement can be used again in subsequent SAVE= options.

SCHEMA Statement
The SCHEMA statement is used to define a simple star schema in the server from a single fact table and one or more dimension tables.

Example:  “Example 5: Creating a Star Schema” on page 82

Syntax
SCHEMA dim-specification1 <dim-specification2 …> </options> ;

Required Argument
dim-specification
specifies how to use the dimension table with the fact table. You must specify the variables to use as keys for the fact table (fact-key) and the dimension table (dim-key). The variables do not need to have the same name, but they do need to have the same type.

dim-table-name (fact-key = dim-key < / <PREFIX = dim-prefix> <TAG=server-tag> <, variable-list>>)

dim-table-name
specifies the name of the dimension table.

fact-key
specifies the variable name in the fact table to use.

dim-key
specifies the variable name from the dimension table to use.

PREFIX=dim-prefix
specifies a prefix to use for naming variables in the schema. If you do not specify PREFIX=, then up to the first sixteen characters of the dim-table-name are used as the dimension prefix for naming the variables in the schema.

 Alias NAME=

TAG=server-tag
specifies the server tag to use for identifying the dimension table.

variable-list
specifies the variables from the dimension table to join with the fact table. By default, all variables except the dimension key are transferred from the dimension table. The dimension key is never transferred because a corresponding value is available through the fact-key.
**SCHEMA Statement Options**

**MODE=VIEW | TABLE**

specifies whether the rows of the schema are materialized when the statement executes in the server. The default is MODE=VIEW and implies that the server resolves the relations in the tables but defers the resolution (formation) of the rows until the view is accessed. If you specify MODE=TABLE, then the table is resolved (flattened) when the statement executes. A view consumes much fewer resources (almost none), but data access is slower compared to a flattened table.

Default: **VIEW**

**SAVE=table-name**

saves the result table so that you can use it in other IMSTAT procedure statements like STORE, REPLAY, and FREE. The value for `table-name` must be unique within the scope of the procedure execution. The name of a table that has been freed with the FREE statement can be used again in subsequent SAVE= options.

**TEMPEXPRESS="SAS-expressions"**

**TEMPEXPRESS=file-reference**

specifies either a quoted string that contains the SAS expression that defines the temporary variables or a file reference to an external file with the SAS statements.

**Alias TE=**

**TEMPNAMES=variable-name**

**TEMPNAMES=(variable-list)**

specifies the list of temporary variables for the request. Each temporary variable must be defined through SAS statements that you supply with the TEMPEXPRESS= option.

**Alias TN=**

**Details**

**Assumptions**

The server makes the following assumptions with regard to fact and dimension tables:

- Dimension tables do not have repeat values for the dimension keys. If a key value appears multiple times, the first record that matches the key is used.
- The relation between the fact table and a dimension table is expressed by one pair of keys. That is, one variable in the fact table defines the relation to one variable in the dimension table.
- The variable names of keys in the fact table and dimension tables do not have to be the same.
- The look ups for the keys are performed based on raw values with fuzzing for doubles.
- If the fact table or a dimension table contains pre-levelized CLASS variables, the class-variable signature is removed when the schema is created.
- Partitioning and order-by information is preserved when the schema is created. However, only partitioning of the fact table is taken into consideration and the resulting table or view is partitioned by the same variables as the fact table.
- The relations are resolved when the schema is created. This strategy makes passes through the data more efficient.
**About Views and Tables**

When the SCHEMA statement executes, a temporary table is generated and the name of that temporary table is returned to the client as the result set. You use the &__TEMPLAST__ macro variable to refer to the star schema.

By default, the server creates a view from the schema definition. The temporary table then has columns for all variables in the schema. The relations have been resolved, but the rows of the view have not been formed.

You can request that row resolution takes place when the temporary table is formed. The result is a flattened temporary table where the rows of the schema are materialized.

There are advantages and disadvantages to using views and flattened tables. The primary consideration is whether there is enough memory for the data volume. The following list identifies some of the considerations:

- A flattened table consumes memory when the statement executes. If the memory requirement of the fully joined schema exceeds the capacity of the machine, the statement fails. For example, if you intend to work with relations that expand to seven terabytes of memory, then you cannot flatten the table unless you have that much memory on your system.

- If a flattened table can be held in memory, data access is faster because it is a regular in-memory table.

- A view does not consume memory until it is accessed. At that time, the table is never materialized fully in memory. The joined rows are formed only when a buffer is needed. This enables you to work with views that exceed the memory capacity of the system.

- The performance difference between resolving a view at run time and accessing a flattened table is difficult to quantify. It depends, for example, on the number of columns to resolve and the data access pattern. A request that passes through the data multiple times suffers a greater performance hit (compared to a flat table) than a single-pass request.

Some operations are not supported with views (but are supported with materialized schemas):

- You cannot append tables or rows to a view.
- You cannot perform row updates of the view.
- You cannot re-partition a view.
- You cannot use a view in another schema.

If a view is based on a partitioned fact table and you want to change the partition key, then re-partition the fact table and re-create the view with another SCHEMA statement.

A view is static. For example, if you append rows to the fact table, the append operation succeeds and every new access to the fact table can use the appended rows. However, the view is not affected by the addition of rows to the fact table. The view resolves to the state of the fact table when the view was formed.

If you want a schema to change with appends and updates, then you can materialize it and then append or update the flattened table. Likewise, you can append or update the fact table and dimension tables, drop the view, and re-create it.

Using a view as the fact table or as a dimension table in a SCHEMA statement is not supported.
The SCORE statement applies scoring rules to an in-memory table. The results can take different forms. They can be displayed as tables in the SAS session, output data sets on the client, or temporary tables in the server. The most common use of the SCORE statement is to execute DATA step code on some or all rows of an in-memory table and to produce corresponding output records.

**SCORE Statement**

The SCORE statement applies scoring rules to an in-memory table. The results can take different forms. They can be displayed as tables in the SAS session, output data sets on the client, or temporary tables in the server. The most common use of the SCORE statement is to execute DATA step code on some or all rows of an in-memory table and to produce corresponding output records.

**Syntax**

```
SCORE CODE=file-reference <options>;
```

**Required Argument**

**CODE=file-reference**

specifies a file reference to the SAS program that performs the scoring.

Alias **PGM=**

**SCORE Statement Options**

**KEEP=(variable-list)**

specifies one or more variables that you want to transfer from the input data to the scoring results. You can use _ALL_ for all variables, _NUMERIC_ for all numeric variables, and other valid variable list names. If this option is not specified, then no variables are transferred from the input data (the table that is being scored), unless they are assigned values in the scoring code.

Alias **TABLEVARS=**

**NOPREPARSE**

specifies to prevent pre-parsing and pre-generating the program code that is referenced in the CODE= option. If you know the code is correct, you can specify this option to save resources. The code is always parsed by the server, but you might get more detailed error messages when the procedure parses the code rather than the server. The server assumes that the code is correct. If the code fails to compile, the server indicates that it could not parse the code, but not where the error occurred.

Alias **NOPREP**

**OUT=libref.member-name**

specifies the name of an output data set in which to store the scoring results. If the result set contains variables that match those in the input data set, then format information is transferred to the output data set. The OUT= option and the TEMPTABLE option are mutually exclusive. If you specify the OUT= option, a temporary table is not created in the server.

Alias **DATA=**

**PARTITION <=partition-key>**

specifies to take advantage of partitioning for tables that are partitioned. When this option is specified, the scoring code is executed in the order of the partitions. If the data are also ordered within the partition, the observations are processed in that
order. If the scoring code uses the reserved symbols `__first_in_partition` or `__last_in_partition`, then the data are also processed in partitioned order. Although the observations are processed in a specific order, the execution occurs in concurrent threads (in parallel). Different threads are assigned to work on different partitions.

If you do not specify the optional `partition-key`, then the analysis is performed for all partitions. If you do specify a `partition-key`, then the analysis is performed for the partitions that match the specified key value only. You can use the PARTITIONINFO statement to retrieve the valid `partition-key` values for a table.

You can specify a `partition-key` in two ways. You can supply a single quoted string that is passed to the server, or you can specify the elements of a composite key separated by commas. For example, if you partition a table by variables GENDER and AGE, with formats $1 and BEST12, respectively, then the composite partition key has a length of 13. You can specify the partition for the 11 year-old females as follows:

```plaintext
score / partition="F          11"; /* passed directly to the server */
score / partition="F","11";        /* composed by the procedure */
```

If you choose the second format, the procedure composes a key based on formatting information from the server.

**Alias**

**PART**=

**Interaction**

This option is effective when used with partitioned in-memory tables only.

**SAVE=table-name**

saves the result table so that you can use it in other IMSTAT procedure statements like STORE, REPLAY, and FREE. The value for `table-name` must be unique within the scope of the procedure execution. The name of a table that has been freed with the FREE statement can be used again in subsequent SAVE= options.

**SETSIZE**

requests that the server estimate the size of the result set. The procedure does not create a result table if the SETSIZE option is specified. Instead, the procedure reports the number of rows that are returned by the request and the expected memory consumption for the result set (in KB). If you specify the SETSIZE option, the SAS log includes the number of observations and the estimated result set size. See the following log sample:

```plaintext
NOTE: The LASR Analytic Server action request for the STATEMENT
    statement would return 17 rows and approximately
    3.641 kBytes of data.
```

The typical use of the SETSIZE option is to get an estimate of the size of the result set in situations where you are unsure whether the SAS session can handle a large result set. Be aware that in order to determine the size of the result set, the server has to perform the work as if you were receiving the actual result set. Requesting the estimated size of the result set does consume resources on the server. The estimated number of KB is very close to the actual memory consumption of the result set. It might not be immediately obvious how this size relates to the displayed table, since many tables contain hidden columns. In addition, some elements of the result set might not be converted to tabular output by the procedure.

**SYMBOLS=(symbol-list)**

specifies one or more symbols that are calculated in the scoring code that you want to transfer as columns to the scoring results. If the SYMBOLS= option is not
specified, then all symbols that are assigned values in the program—and that are not just placeholders for intermediate calculations—are transferred to the results. If you use a large program with many assignments, you might want to use the SYMBOLS= option to limit the columns in the results.

Alias  SYM=

TEMPTABLE  
generates an in-memory temporary table from the result set. The IMSTAT procedure displays the name of the table and stores it in the _TEMPLAST_ macro variable, provided that the statement executed successfully.

When the IMSTAT procedure exits, all temporary tables created during the IMSTAT session are removed. Temporary tables are not displayed on a TABLEINFO request, unless the temporary table is the active table for the request.

Details  
To help manage how output is generated, options in the SCORE statement can be brought to bear together with special syntax elements in the scoring code. For example, the PARTITION<=> option can be used to specify that the scoring code is executed separately for each partition or for a specific partition of the data only. If you want to control precisely which observations are used to generate output records, you can use the __lasr_output symbol in your SAS program. When this symbol is set to 1, the row is output. You can also use the __first_in_partition and __last_in_partition variables to programmatically determine the first and last observation in a partition.

The following SAS code is an example:

__lasr_output = 0;
if __first_in_partition then do;  
  totalmsrp = msrp;
  minmsrp = msrp;
  numCars = 1;
end; else do;
  totalmsrp + msrp;
  numCars + 1;
  if (msrp < minmsrp) then minmsrp = msrp;
end;
orgdrive = Origin || drivetrain;  
mpgdiff = mpg_highway - mpg_city;
if __last_in_partition then  
  __lasr_output = 1;

1 For the first observation within a partition, three variables are initialized. The minimum MSRP, the total MSRP, and the number of records in the partition are then computed.

2 The variable ORGDRIVE is obtained by concatenating the strings of the ORIGIN and DRIVETRAIN variables

3 When the last record within the partition is reached, the __lasr_output automatic variable is set to 1, this is used to add of the current record to the result set.

The execution of the SCORE code observes the active WHERE clause in the IMSTAT run block—in other words, the scoring code is executed only for those observations that meet the WHERE condition if a WHERE clause is active.
The following example loads the SASHELP.CARS data set partitioned by the TYPE variable, and executes the previous code sample.

```sas
data lasrlib.cars(partition=(type));
   set sashelp.cars;
run;

filename fref '/path/to/scorepgm.sas';

proc imstat;
   table lasrlib.cars;
   score pgm=fref / partition;
run;
```

The PARTITION option in the SCORE statement requests the server to execute the code separately for each partition of the data. Because the code outputs one observation per partition and there are six unique values of the TYPE variable in the SASHELP.CARS data set, the scoring results show six rows:

<table>
<thead>
<tr>
<th>totalmsrp</th>
<th>minmsrp</th>
<th>numCars</th>
<th>orgdrive</th>
<th>mpgdiff</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>59760</td>
<td>19110</td>
<td>3</td>
<td>Asia</td>
<td>-8</td>
<td>Hybrid</td>
</tr>
<tr>
<td>2087415</td>
<td>17163</td>
<td>60</td>
<td>EuropeAll</td>
<td>5</td>
<td>SUV</td>
</tr>
<tr>
<td>7800688</td>
<td>10280</td>
<td>262</td>
<td>EuropeFront</td>
<td>7</td>
<td>Sedan</td>
</tr>
<tr>
<td>2615966</td>
<td>18345</td>
<td>49</td>
<td>Asia</td>
<td>6</td>
<td>Sports</td>
</tr>
<tr>
<td>598593</td>
<td>12800</td>
<td>24</td>
<td>Asia</td>
<td>3</td>
<td>Truck</td>
</tr>
<tr>
<td>865216</td>
<td>11905</td>
<td>30</td>
<td>EuropeAll</td>
<td>7</td>
<td>Wagon</td>
</tr>
</tbody>
</table>

The SCORE statement does not support the temporary expressions that are available in other IMSTAT statements. This is because you can compute all necessary temporary variables in the scoring code.

**SERVERINFO Statement**

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

**Syntax**

```sas
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.

**SAVE=table-name**
saves the result table so that you can use it in other IMSTAT procedure statements
like STORE, REPLAY, and FREE. The value for *table-name* must be unique within
the scope of the procedure execution. The name of a table that has been freed with
the FREE statement can be used again in subsequent SAVE= options.

---

**SERVERPARAM Statement**
The SERVERPARAM 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**

```plaintext
SERVERPARAM <options>;
```

**SERVERPARAM 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: N_ACTIONS=

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.

<table>
<thead>
<tr>
<th>Alias</th>
<th>MEMLOAD=</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>75</td>
</tr>
</tbody>
</table>

**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 IMSTAT procedure until the server that it uses
receives a termination request.
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.

SET Statement

The SET statement is used to append in-memory tables to each other. The result of the operation is not a
    temporary table, but the appending of rows from the secondary tables to the active table.

Examples:  "Example 6: Appending Tables" on page 84
            "Example 7: Appending a Non-Partitioned Table to a Partitioned Table" on page 85

Syntax

SET set-specification1 <set-specification2 ...> </options>;

Required Argument

set-specification
    specifies the table to append to the active table and options. You can list multiple set-
    specifications. A table can be used in more than one set-specification, and you can
    specify the active table in a set-specification.

    table-name <(TAG='server-tag')>
    table-name
        specifies the table to append to the active table.

    TAG='server-tag'
        specifies the server tag to use for identifying the table to append.

SET Statement Options

DROP
    specifies that the secondary tables (the tables specified in the set-specifications) are
    dropped from the server after the statement executes successfully. If the active table
    is listed in a set-specification, it is not dropped.

NOPARTITION
    specifies to append the secondary tables and undo the partitioning of the active table.
    If the active table is partitioned, and you append partitioned tables to it, then the
    server rejects the request unless all the tables have the same partitioning variables, in
    the same order, and have the same key length. When this option is specified, the
    active table is no longer partitioned if the SET statement succeeds.

Alias    NOPART
WHERE WITH ALL

specifies to apply the WHERE clause to the active table, in addition to the secondary tables to append. By default, the rows of the secondary tables that are appended to the active table are filtered according to the WHERE clause. Rows marked for deletion or purging are not appended to the main table. By default, the WHERE clause does not filter rows of the active table. If you want the WHERE clause to apply to the active table, specify this option.

Alias ALL WHERE

STORE Statement

The STORE statement enables you to assign the contents of previously saved tables to macro variables. You can reuse the results from one statement as input for subsequent statements in the same IMSTAT procedure.

Syntax

STORE table-name (row <,> column) = macro-variable-name;
STORE table-name < [table-number] > (row <,> column) = macro-variable-name;

Required Arguments

macro-variable-name
specifies the name of a macro variable to use for storing the value.

table-name
specifies the saved result table to use.

row

column

specifies the row number and column number to access. These values are identified by the row and column of the table as it appears in the default output or with the REPLAY statement. Be aware that hidden columns that might appear in an output data set when an ODS table is converted to a SAS data set are not counted.

Optional Argument

[table-number]

specifies the table to use for accessing multi-part tables. In the following example, the HISTOGRAM statement generates one histogram table for the Cylinders variable and a second for the EngineSize variable. The two histogram tables are stored in the temporary HistTab table. In order to access the second histogram table, the [2] is used. If you do not specify a table-number, the first table is used.

Example

```sas
proc imstat data=mylasr.cars(tag=sashelp);
  histogram Cylinders EngineSize / save=HistTab;
  store HistTab[2](2,6) = Engsz_Pct;
quit;
%put &Engsz_Pct;
```
TABLE Statement
The TABLE statement is used to specify the in-memory table to use for subsequent IMSTAT procedure statements. You can use this statement to switch between different in-memory tables in a single run of the IMSTAT procedure.

Example: “Example 5: Creating a Star Schema” on page 82

Syntax
TABLE <libref.member-name>;

Optional Argument

libref.member-name
specifies the libref for the SAS LASR Analytic Server and the table name.

If you do not specify the libref and member-name, the procedure closes the table that is currently open.

Example
A common use for the TABLE statement is to reference a temporary table with the libref.&_TEMPLAST_ macro variable. Temporary tables are in-memory tables that are created with the results of a statement that supports the TEMPTABLE option.

proc imstat data=lasrlib.sales2012;
  partition customerid;
run;

table lasrlib.&_templast_;
run;

/*
 * More statements for the partitioned table.
 * The PROMOTE statement can be used to convert the temporary table to a regular table.
 */
quit;

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.

**SAVE=table-name**
saves the result table so that you can use it in other IMSTAT procedure statements like STORE, REPLAY, and FREE. The value for table-name must be unique within the scope of the procedure execution. The name of a table that has been freed with the FREE statement can be used again in subsequent SAVE= options.

---

**UPDATE Statement**

The UPDATE statement performs rowwise updates of the data in an in-memory table.

**Syntax**

```
UPDATE variable1=value1 <variable2=value2 ...> </options>
UPDATE DATA=libref.member-name </options>
```

**Required Arguments**

**variable**
specifies the name of the variable to update.

**value**
specifies the value to assign to the variable.

**libref.member-name**
specifies the libref and table name of a SAS data set to use for updating the in-memory table. The data set must contain the variables and values that you want to update. You can specify a _WHERE_ variable in the data set to apply as a filter to the particular set of update values. This clause in the data set augments the overall WHERE clause, if one is specified.

**UPDATE Statement Options**

**CODE=file-reference**
specifies a file reference to a SAS program to use for the row update (an update script). You can combine the specification of a SAS program through the CODE= option with the name-value pair specification or the DATA= specification for bulk updates. The updates that are specified in the name-value pair and DATE= specifications are performed first and then the update script executes on the modified row to produce the update.

**Alias PGM=**
NOPREPARSE
specifies to prevent the procedure from pre-parsing and pre-generating code for temporary expressions, scoring programs, and other user-written SAS statements.

When this option is specified, the user-written statements are sent to the server "as-is" and then the server attempts to generate code from it. If the server detects problems with the code, the error messages might not be as detailed as the messages that are generated by SAS client. If you are debugging your user-written program, then you might want to pre-parse and pre-generate code in the procedure. However, if your SAS statements compile and run as you want them to, then you can specify this option to avoid the work of parsing and generating code on the SAS client.

When you specify this option in the PROC IMSTAT statement, the option applies to all statements that can generate code. You can also exclude specific statements from pre-parsing by using the NOPREPARSE option in statements that allow temporary columns or the SCORE statement.

Alias NOPREP

SAVE=table-name
saves the result table so that you can use it in other IMSTAT procedure statements like STORE, REPLAY, and FREE. The value for table-name must be unique within the scope of the procedure execution. The name of a table that has been freed with the FREE statement can be used again in subsequent SAVE= options.

TEMPEXPRESS="SAS-expressions"

TEMPEXPRESS=file-reference
specifies either a quoted string that contains the SAS expression that defines the temporary variables or a file reference to an external file with the SAS statements.

Alias TE=

TEMPNAMES=variable-name
TEMPNAMES=(variable-list)
specifies the list of temporary variables for the request. Each temporary variable must be defined through SAS statements that you supply with the TEMPEXPRESS= option.

Alias TN=

Details

It is common to use the UPDATE statement with a WHERE clause. The clause filters the rows to which the updates are applied. If you are unsure about the number of rows that can be updated, use the NUMROWS statement to determine how many rows would be affected by the rowwise update.

You can update the values of ORDERBY variables, but you cannot update the value of variables that are used for constructing partition keys.

You cannot update the values of permanent computed variables. Their values are determined by the SAS program that originally defined them.
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;

Examples: IMSTAT Procedure

Example 1: Partitioning a Table into a Temporary Table

Details
This PROC IMSTAT example demonstrates partitioning a table as it is loaded to memory and then saving it to a temporary table with different partitioning variables.

Program
libname example sasiola host="grid001.example.com" port=10010 tag='hps';
data example.prdsale(partition=(country region));
   set sashelp.prdsale;
runc;
proc imstat data=example.prdsale;
   partitioninfo;
   summary actual predict / partition;         
run;
/* partition the active table, example.prdsale, by region and prodtype */
   partition region prodtype;         
rund;
   table example.&_templast_;         
rund;
   partitioninfo;
   summary actual predict / partition;    
quit;
Program Description

1. The Prdsale data set is loaded into memory and partitioned by the unique combinations of the formatted values for the Country and Region variables.

2. The procedure examines the partitioning of the table and requests a summarization of the Actual and Predict variables by the partition values (unique combinations of Country and Region).

3. In order to accommodate a different data access pattern, the table is partitioned by unique combinations of the Region and Prodtype variables. The table is stored in a temporary table and the name is assigned to the _TEMPLAST_ macro variable.

4. The TABLE statement references the _TEMPLAST_ macro variable and sets the temporary table as the active table. All statements that follow use the temporary table.

5. As with the previous SUMMARY statement, the partitioning is examined and the summary is requested for the Actual and Predict variables by the unique combinations of the Region and Prodtype variables.

Output

Output 4.1  Partitions for Prdsale When Partitioned by Country and Region

<table>
<thead>
<tr>
<th>LASR Node</th>
<th>Partition Number</th>
<th>Partition Key</th>
<th>Size In kBytes</th>
<th>Number of Records</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>CANADA EAST</td>
<td>28 125</td>
<td>240</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>CANADA WEST</td>
<td>28 125</td>
<td>240</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>GERMANY EAST</td>
<td>28 125</td>
<td>240</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>GERMANY WEST</td>
<td>28 125</td>
<td>240</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>U.S.A. EAST</td>
<td>28 125</td>
<td>240</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>U.S.A. WEST</td>
<td>28 125</td>
<td>240</td>
</tr>
</tbody>
</table>

Output 4.2  Summary Statistics for Prdsale When Partitioned by Country and Region

<table>
<thead>
<tr>
<th>COUNTRY</th>
<th>REGION</th>
<th>Column</th>
<th>Min</th>
<th>Max</th>
<th>N</th>
<th>Sum</th>
<th>Mean</th>
<th>Std Dev.</th>
<th>Std Error</th>
<th>Coefficient of Variation</th>
<th>Number Missing</th>
</tr>
</thead>
<tbody>
<tr>
<td>CANADA</td>
<td>EAST</td>
<td>ACTUAL</td>
<td>5.0000</td>
<td>999.00</td>
<td>240</td>
<td>127485</td>
<td>531.19</td>
<td>251.61</td>
<td>18.1731</td>
<td>53.0160</td>
<td>D</td>
</tr>
<tr>
<td>CANADA</td>
<td>EAST</td>
<td>PREDICT</td>
<td>0.9050</td>
<td>993.00</td>
<td>240</td>
<td>120646</td>
<td>503.69</td>
<td>284.29</td>
<td>18.3610</td>
<td>56.6540</td>
<td>D</td>
</tr>
<tr>
<td>CANADA</td>
<td>WEST</td>
<td>ACTUAL</td>
<td>3.0000</td>
<td>1000.00</td>
<td>240</td>
<td>116665</td>
<td>497.94</td>
<td>286.53</td>
<td>19.1436</td>
<td>59.6070</td>
<td>D</td>
</tr>
<tr>
<td>CANADA</td>
<td>WEST</td>
<td>PREDICT</td>
<td>6.0000</td>
<td>1000.00</td>
<td>240</td>
<td>112373</td>
<td>468.22</td>
<td>275.99</td>
<td>17.8150</td>
<td>58.9443</td>
<td>D</td>
</tr>
<tr>
<td>GERMANY</td>
<td>EAST</td>
<td>ACTUAL</td>
<td>13.0000</td>
<td>1000.00</td>
<td>240</td>
<td>124547</td>
<td>518.95</td>
<td>287.71</td>
<td>18.5714</td>
<td>55.4405</td>
<td>D</td>
</tr>
<tr>
<td>GERMANY</td>
<td>EAST</td>
<td>PREDICT</td>
<td>4.0000</td>
<td>993.00</td>
<td>240</td>
<td>117579</td>
<td>489.91</td>
<td>292.43</td>
<td>19.8752</td>
<td>59.6901</td>
<td>D</td>
</tr>
<tr>
<td>GERMANY</td>
<td>WEST</td>
<td>ACTUAL</td>
<td>3.0000</td>
<td>996.00</td>
<td>240</td>
<td>121451</td>
<td>506.05</td>
<td>289.17</td>
<td>19.6658</td>
<td>57.1429</td>
<td>D</td>
</tr>
<tr>
<td>GERMANY</td>
<td>WEST</td>
<td>PREDICT</td>
<td>0.9810</td>
<td>993.00</td>
<td>240</td>
<td>113975</td>
<td>474.50</td>
<td>280.49</td>
<td>19.1056</td>
<td>59.0937</td>
<td>D</td>
</tr>
<tr>
<td>U.S.A.</td>
<td>EAST</td>
<td>ACTUAL</td>
<td>4.0000</td>
<td>984.00</td>
<td>240</td>
<td>118229</td>
<td>492.62</td>
<td>292.26</td>
<td>19.2200</td>
<td>57.2963</td>
<td>D</td>
</tr>
<tr>
<td>U.S.A.</td>
<td>EAST</td>
<td>PREDICT</td>
<td>1.0000</td>
<td>1000.00</td>
<td>240</td>
<td>120687</td>
<td>502.45</td>
<td>301.35</td>
<td>19.4516</td>
<td>59.9757</td>
<td>D</td>
</tr>
<tr>
<td>U.S.A.</td>
<td>WEST</td>
<td>ACTUAL</td>
<td>6.0000</td>
<td>984.00</td>
<td>240</td>
<td>119120</td>
<td>486.33</td>
<td>285.67</td>
<td>18.4401</td>
<td>57.5507</td>
<td>D</td>
</tr>
<tr>
<td>U.S.A.</td>
<td>WEST</td>
<td>PREDICT</td>
<td>22.0000</td>
<td>999.00</td>
<td>240</td>
<td>121135</td>
<td>504.73</td>
<td>280.10</td>
<td>18.0801</td>
<td>55.4944</td>
<td>D</td>
</tr>
</tbody>
</table>
Example 2: Promoting Temporary Tables to Regular Tables

Details

The SUMMARY, SCORE, CROSSTAB, PERCENTILE, and DISTINCT statements offer a TEMPTABLE option. When you specify the TEMPTABLE option, the results of the statement are written to a temporary table. The PARTITION statement also results in a temporary table. If you want to keep the table, you can use the PROMOTE statement to convert the table from being a temporary table to a regular table. Once you do this, other users can access the data.

Program

```
libname example sasiola host="grid001.unx.sas.com" port=10010 tag='hps';

data example.prdsale; set sashelp.prdsale; run;

proc imstat data=example.prdsale; 
  summary / groupby=(country) temptable;  
run; 
  table example.&_templast_;  
run; 
  promote sum_by_country;  
run; 
  table example.sum_by_country;  
run; 
  fetch / format to=10;  
quit;
```
Program Description

1. The TEMPTABLE option stores the results of the SUMMARY statement to a temporary table. If this table is not promoted before the QUIT statement, it is removed from memory.

2. The TABLE statement references the _TEMPLAST_ macro variable and sets the temporary table as the active table. All statements that follow use the temporary table.

3. The PROMOTE statement converts the temporary table to a regular table with the name Sum_By_Country. The table is associated with the current library through the libref, example. The SAS log also includes a note that indicates how to specify the libref and table name.

4. The TABLE statement makes the table the active table explicitly by specifying the libref and table name. The Sum_By_Country table is not removed from memory when the IMSTAT procedure terminates.

5. All the subsequent statements that follow the TABLE statement use the newly promoted table.

The example does not show the use of SAS LASR Analytic Server engine server tags. You can use server tags with the PROMOTE statement as show in the following code sample.

```sas
proc imstat data=example.prdsale;
  summary / groupby=(region) temptable;
run;

table example.&_templast_; run;
promote sum_by_region / tag="sales";
run;
table example.sum_by_country(tag="sales"); run;
quit;
```

As shown in the previous example, the TAG= option is used in the PROMOTE statement. To access the table, the TABLE statement uses the TAG= data set option.

As shown in the following sample, the SAS log indicates the libref, table name, and server tag to use for accessing the table.

Log 4.1  SAS Log for the PROMOTE Statement with the TAG= Option

```
NOTE: The temporary table _T_BE5C2602_45A0DCB8 was successfully promoted to the LASR Analytic Server table WORK.SUM_BY_COUNTRY. You can access this table with

   the TABLE statement as table EXAMPLE.sum_by_country(tag='sales').
```

Example 3: Rebalancing a Table

Details

It might be beneficial to rebalance the rows of a table if the data access patterns do not take advantage of partitioning or if the HDFS block distribution becomes uneven.
Program

```
libname example sasiola host="grid001.example.com" port=10010 tag='hps';

proc imstat immediate;
  table example.table1;
  distributioninfo; 1
  balance;
  droptable; 2
  table example.&_templast_; 3
  promote table1; 4
  table example.table1;
  distributioninfo; 5
  /* save path="/hps" replace; */ 6
quit;
```

Program Description

1. The DISTRIBUTIONINFO statement displays the number of rows from Table1 on each machine in the cluster.

2. The DROPTABLE statement is used to drop the active table, Table1.

3. The BALANCE statement rebalanced Table1 into a temporary table. The TABLE statement is used with the &_TEMPLAST_ macro variable to access the temporary table.

4. The PROMOTE statement changes the temporary table into a regular in-memory table with the original table name, Table1.

5. After setting the Table1 as the active table with the TABLE statement, the DISTRIBUTIONINFO statement displays the nearly homogenous distribution of rows.

6. The SAVE statement can be used to save the table back to HDFS with the homogeneous block distribution.
**Output**

The following output shows the partial display for the first DISTRIBUTIONINFO statement. One machine has zero rows and another machine has approximately twice the number of rows.

**Output 4.5  Uneven Row Distribution**

<table>
<thead>
<tr>
<th>LASR Node</th>
<th>Number of Partitions</th>
<th>Number of Records</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>166971</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>165984</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>166984</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>165984</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>165984</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>165984</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>331955</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>165984</td>
</tr>
</tbody>
</table>

The following output shows the homogenous distribution of rows after the BALANCE statement is used.

**Output 4.6  Homogenous Row Distribution**

<table>
<thead>
<tr>
<th>LASR Node</th>
<th>Number of Partitions</th>
<th>Number of Records</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>165977</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>165977</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>165977</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>165977</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>165977</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>165977</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>165977</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>165977</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>165977</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>165977</td>
</tr>
</tbody>
</table>
Example 4: Deleting Rows and Saving a Table to HDFS

Details
The server can delete rows from in-memory tables and also save tables to HDFS. The following example demonstrates using WHERE clause processing across RUN-group boundaries to copy a subset of an in-memory table to HDFS and then delete the subset from memory.

Program
```
libname example sasiola host="grid001.example.com" port=10010 tag='hps';

data example.prdsale; set sashelp.prdsale; run;

proc imstat data=example.prdsale;
   where year=1994 and quarter=1;
   save path="/dept/sales/y1994q1" copies=1 fullpath;
run;
   deleterows / purge;
run;
   where;
   summary actual;
run;
```

Program Description
1. Once the WHERE clause is specified, it applies to the statements that follow it. It also crosses RUN boundaries.
2. The SAVE statement is subject to the WHERE clause. As a result, the records from the Prdsale table that meet the WHERE clause are saved to /dept/sales/y1994q1.sashdat. The FULLPATH option is used to specify the table name instead of using the name of the active table. This is particularly useful when saving temporary tables.
3. The DELETEROWS statement is also subject to the WHERE clause. The records that were just saved to HDFS are now deleted and purged from memory. (The DELETEROWS statement without the PURGE option would mark the records for deletion and exclude them from being used in calculations, but it does not free the memory resources.)
4. The WHERE clause is cleared and the SUMMARY statement that follows is performed against all the remaining records in the Prdsale table.

This pattern of using a WHERE clause to subset an in-memory table, save the records to HDFS, and then delete them can be combined with the APPEND data set option of the SAS LASR Analytic Server engine. You can create a sliding window for keeping months or years of data in memory for analysis, yet keeping it up-to-date by appending the most recent records.
Example 5: Creating a Star Schema

Details
The following example demonstrates using the SCHEMA statement to join dimension tables with a fact table.

Program

```
libname example sasiola host="grid001.example.com" port=10010 tag='hps';

proc imstat;
  table example.mailorder;  
    schema catalog  (catCode=CatCode)
    products (pcode=pcode )
    customers(custnum=custnum);  
run;

  table example.&_templast_
run;
  columninfo;
quit;
```

Program Description

1. Once the WHERE clause is specified, it applies to the statements that follow it. It also crosses RUN boundaries.

2. The SAVE statement is subject to the WHERE clause. As a result, the records from the Prdsale table that meet the WHERE clause are saved to /dept/sales/y1994q1.sashdat. The FULLPATH option is used to specify the table name instead of using the name of the active table. This is particularly useful when saving temporary tables.

3. The DELETEROWS statement is also subject to the WHERE clause. The records that were just saved to HDFS are now deleted and purged from memory. (The
DELETEROWS statement without the PURGE option would mark the records for deletion and exclude them from being used in calculations, but it does not free the memory resources.

4. The WHERE clause is cleared and the SUMMARY statement that follows is performed against all the remaining records in the Prdsale table.

Output

The following output shows the temporary table name and how the dimension table names are used as prefixes for the column names.

```
<table>
<thead>
<tr>
<th>Temporary Table Information for Table HPS.MAILORDER</th>
</tr>
</thead>
<tbody>
<tr>
<td>Statement</td>
</tr>
<tr>
<td>Temporary Table</td>
</tr>
<tr>
<td>Table Type</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>Column Information for Table _T.BAD4C8F8_7F10C904F48</th>
</tr>
</thead>
<tbody>
<tr>
<td>Id</td>
</tr>
<tr>
<td>----</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>9</td>
</tr>
<tr>
<td>10</td>
</tr>
<tr>
<td>11</td>
</tr>
<tr>
<td>12</td>
</tr>
<tr>
<td>13</td>
</tr>
<tr>
<td>14</td>
</tr>
<tr>
<td>15</td>
</tr>
<tr>
<td>16</td>
</tr>
<tr>
<td>17</td>
</tr>
<tr>
<td>18</td>
</tr>
<tr>
<td>19</td>
</tr>
<tr>
<td>20</td>
</tr>
<tr>
<td>21</td>
</tr>
<tr>
<td>22</td>
</tr>
</tbody>
</table>
```
Example 6: Appending Tables

Details

The following example demonstrates using the SET statement to append tables with the active table.

Program

```sas
libname example sasiola host="grid001.example.com" port=10010 tag='hps';
libname hdfs sashdat host="grid001.example.com" install="/opt/TKGrid" path="/hps";

proc lasr add data=hdfs.january port=10010;
    performance host="grid001.example.com" nodes=all;
run;

proc lasr add data=hdfs.february port=10010;
    performance host="grid001.example.com" nodes=all;
run;

data example.march;
    set otherlib.march;
run;

proc imstat;
    table example.january;
    set february / drop;
    set march;
run;

    save path="/hps/qtr1" copies=1 replace fullpath;
quit;
```

Program Description

1. The value for the TAG= option in the SAS LASR Analytic Server LIBNAME statement matches the PATH= value for the SAS Data in HDFS engine LIBNAME statement.
2. The first table, January, is loaded to memory from HDFS.
3. The second table, February, is loaded to memory from HDFS. The tables are still independent in-memory tables.
4. The third table, March, is loaded from another library into the server with the SAS LASR Analytic Server engine.
5. The first table, January, is set as the active table.
6. The second table, February, is appended to the active table. The DROP option specifies to remove the February table from memory as soon as the SET statement completes.
7. The third table, March, is appended to the active table. This table remains in memory.
8. The active table, January, now has the February and March tables appended to it. The SAVE statement saves the table to HDFS with the name Qtr1.

---

**Example 7: Appending a Non-Partitioned Table to a Partitioned Table**

**Details**

The following example demonstrates how to append a table that is not partitioned to an in-memory table that is partitioned. The SET statement is used to append the table.

*Note:* As an alternative, if the table to append is not already in memory, you can append the rows to the partitioned in-memory table with the SAS LASR Analytic Server engine. For more information, see “APPEND= Data Set Option” on page 125.

**Program**

```sas
libname example sasiola host="grid001.example.com" port=10010 tag='hps';
libname hdfs sashdat host="grid001.example.com" install="/opt/TKGrid" path="/hps";

proc lasr add data=hdfs.transactions(partition=(customerid)) port=10010;
  performance host="grid001.example.com" nodes=all;
run;

proc lasr add data=hdfs.recenttrans(partition=(dateid)) port=10010;
  performance host="grid001.example.com" nodes=all;
run;

proc imstat;
  table example.recenttrans;  
  partition customerid;
run;
  table example.transactions;  
  set &_templast_ / drop;
quit;
```

**Program Description**

1. The value for the TAG= option in the SAS LASR Analytic Server LIBNAME statement matches the PATH= value for the SAS Data in HDFS engine LIBNAME statement.
2. The first table, Transactions, is loaded to memory from HDFS. The table is partitioned by values of the CustomerId variable.
3. The second table, RecentTrans, is loaded to memory from HDFS. The table is partitioned by values of the DateId variable.
4. The second table, RecentTrans, is set as the active table and then partitioned into a temporary table with the PARTITION statement. The temporary table is partitioned by values of the CustomerId variable.
5. The first table, Transactions, is set as the active table.
6. The temporary table is appended to the active table. The DROP option specifies to remove the temporary table from memory as soon as the SET statement completes.
Example 8: Storing Temporary Variables

Details

Many statements offer a SAVE= option that is used to save the result table of the statement for use in other IMSTAT procedure statements. You can use the STORE statement to assign a value from the saved result table to a macro variable.

Program

```
libname example sasiola host="grid001.unx.sas.com" port=10010 tag='hps';

data example.prdsale(partition=(country region)); set sashelp.prdsale; run;

proc imstat data=example.prdsale immediate noprint;
   percentile actual / partition="U.S.A.", "EAST" save=tab1;
   run;
   percentile actual / partition="CANADA", "EAST" save=tab2;
   run;

   store tab1(3,5) = us_SeventyFivePct;
   run;
   store tab2(3,5) = ca_SeventyFivePct;
   run;
   %put %sysfunc(&us_SeventyFivePct - &ca_SeventyFivePct);
   replay tab2;
   run;

   free tab1 tab2;
   free macro=us_SeventyFivePct;
   free macro=ca_SeventyFivePct;
quit;
```

Program Description

1. The NOPRINT option suppresses displaying the results tables.
2. The results tables for the PERCENTILE statements are saved to temporary tables.
3. The STORE statements access the results tables and store the value from the fifth column in the third row (the 75th percentile) to macro variables.
4. The REPLAY statement displays the results table for the second PERCENTILE statement.
5. The FREE statements releases the memory used by results tables and the macro variables.
Log Output

The SAS log describes how the values are stored to the macro variables.

```
store tab1(3,5) = us_SeventyFivePct;
NOTE: The numeric value 746.5 from row 3, column 5 of table tab1 has been stored
     in the macro variable us_SeventyFivePct.
run;

store tab2(3,5) = ca_SeventyFivePct;
NOTE: The numeric value 759.5 from row 3, column 5 of table tab2 has been stored
     in the macro variable ca_SeventyFivePct.
```
Chapter 5
IMXFER Procedure

Overview: IMXFER Procedure

What Does the IMXFER Procedure Do?

The IMXFER procedure is used to transfer in-memory tables between two distributed SAS LASR Analytic Server instances. The procedure takes advantage of network topology and parallelism as much as possible.

The IMXFER procedure cannot be used with a non-distributed SAS LASR Analytic Server.

Syntax: IMXFER Procedure

PROC IMXFER <options>;
    SERVER server-name <HOST=host-name> <PORT=number>;
    TABLE export-server-name export-table-name
        import-server-name <import-table-name> </options>;
QUIT;

Examples: IMXFER Procedure

Example 1: Copying Tables from One Server to Another
Example 2: Copying Tables from One Cluster to Another
PROC IMXFER Statement
Transfers an in-memory table.

Syntax
PROC IMXFER <options>;

Optional Arguments

HOSTONLY
specifies to transfer the tables through the root nodes on the two clusters. With this option, the data are collected by the exporting root node before sending them to the importing root node. The importing root node then distributes the data to its worker nodes before passing the data to the importing server instance.

Specify this option if you know in advance that the worker nodes of the server instances do not have network communication with each other. (Even if you do not specify this option when this network topology exists, the procedure detects the lack of communication, and routes the data this way automatically.) Specify the option so that time is not lost trying to establish network connections between the clusters.

Alias NOWORKER

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

Alias SINGLESTEP

LASRERROR
specifies that the procedure terminate when an error message is received from one of the servers.

If you do not specify this option, the IMXFER procedure attempts to continue interactive processing of programming statements. For example, if you receive an error that a table with the same name already exists in the importing server instance, you might prefer to change the name and continue rather than end the procedure.

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

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

Alias NOTIME

TIMEOUT=n
specifies the time in seconds that the worker nodes of the exporting server waits for network connections. When this interval of time has passed, the data transfer occurs through the root nodes only.
SERVER Statement

The SERVER statement is used to specify a server instance to use in a transfer. In the statement, you assign a logical name to the server and you use that name subsequently to refer to the particular server instance. There is no limit to the number of SERVER statements. You can establish connections to more than two servers with the IMXFER procedure.

Syntax

SERVER server-name <HOST=host-name> <PORT=number>;

Required Argument

server-name

specifies the name to use for referring to the server instance. The name is used in the TABLE statement to identify the exporting server and the importing server.

SERVER Statement Options

HOST="host-name"

specifies the host name for the SAS LASR Analytic Server. If this option is not specified, then the GRIDHOST environment variable is used.

PORT=number

specifies the port number for the SAS LASR Analytic Server.

Default 10616

TABLE Statement

The TABLE statement is used to specify the table to export from one server and import to another server.

Syntax

TABLE export-server-name export-table-name import-server-name <import-table-name> </options>;

Required Arguments

export-server-name

specifies the name to use for the server instance that is exporting the table.

export-table-name

specifies the in-memory table to export. The name is specified as server-tag.member-name.

import-server-name

specifies the name to use for the server instance that is importing the table.
Optional Argument

import-table-name
specifies the name to use for the imported table.

If you do not specify a name, then the IMXFER procedure attempts to create a table with the same name as the exported table. If a table with the same name already exists in the importing server, then the transfer fails.

TABLE Statement Options

DELETED= INCLUDE | INC | EXCLUDE
specifies how rows that are marked for deletion are handled in the transfer. By default, DELETED=EXCLUDE, which implies that any row that has a deletion mark is not transferred.

If you specify DELETED=INCLUDE, the IMXFER procedure instructs the server to ignore the deletion marks. Any rows that are marked for purging are not transferred, regardless of the DELETED= option.

Default EXCLUDE

FILTER="where-expression"
specifies the WHERE clause to apply to the exported table. Only rows that meet the conditions of the WHERE expression are transferred.

Alias WHERE=

PARTITION= NO | REMOVE | YES
specifies how to handle partitioning (and ordering within the partitions) when a partitioned table is transferred. By default, PARTITION=YES, and implies that a partitioned table is transferred to the importing server and remains partitioned and ordered by the same variables. When the servers have different numbers of worker nodes, there is no guarantee that partitions end up on the same nodes. However, it is guaranteed that partitions appear together on a node in the importing server.

Partitioning incurs some overhead and if you transfer a table from a smaller to a larger number of nodes, you might not want to apply partitioning. (Removing the partitioning spreads the data out more evenly in the importing server.) Or, you might not want to maintain partitioning on transfer if the transfer is for archival purposes. In that case, specify PARTITION=NO or PARTITION=REMOVE. This transfers the table to the importing server without the partitioning information.

Default YES

PERMISSION=mode
specifies the permission setting for accessing the imported 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 PERM=

Range 600 to 777
QUIT Statement

This statement terminates the IMXFER procedure. When the QUIT statement is reached, all resources allocated by the procedure are released and all connections to servers are terminated.

**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;

---

**Examples: IMXFER Procedure**

**Example 1: Copying Tables from One Server to Another**

**Details**

It might be necessary to copy tables from one SAS LASR Analytic Server instance to another.

**Program**

```sas
proc imxfer;
  server s1 host="grid001.example.com" port=10031;
  server s2 host="grid001.example.com" port=10010;
  table s1 public.fact_table s2;  // 1
  quit;
```

**Program Description**

1. The first SERVER statement creates a reference to the server that is listening on port 10031. The second SERVER statement creates a reference to a server on the same host, but listening on port 10010.

2. The TABLE statement transfers the Hps.Fact_Table table from the server that is listening on port 10031 to the server that is listening on port 10010. Because no `import-table-name` is specified, the table uses the name Hps.Fact_Table on the importing server.
Example 2: Copying Tables from One Cluster to Another

Details
It might be necessary to copy tables from one SAS LASR Analytic Server instance on one cluster to a server that is running on a different cluster. The clusters can have different numbers of machines.

When the number of machines is not the same, the IMXFER procedure automatically redistributes the rows of the table to provide the most even distribution possible. In most cases, equalizing the data distribution equalizes the work load and provides the best performance. By default, partitioned tables remain partitioned on the importing server. For more information, see the PARTITION= on page 92 option for the TABLE statement.

Program

```
proc imxfer;
  server s1 host="grid001.example.com" port=10031; 1
  server s2 host="cluster2.example.com" port=10010;

  table s1 public.inventory s2 hps.inventory; 2
  quit;

/* access the transferred table */
libname cluster2 sasiola host="cluster2.example.com" port=10010 tag="hps"; 3

proc imstat;
  table cluster2.inventory;
run;

  distributioninfo; 4
  quit;
```

Program Description

1. The first SERVER statement creates a reference to the server that is listening on port 10031. The second SERVER statement creates a reference to a server on a different cluster that is listening on port 10010.

2. The TABLE statement transfers the Public.Inventory table from the server that is listening on port 10031 to the server on the other cluster. The table is renamed to Hps.Inventory on the importing server.

3. To access the transferred table, the LIBNAME statement must use the value "hps" as the server tag.

4. The DISTRIBUTIONINFO statement for the IMSTAT procedure displays the number of rows that are used on each machine in the second cluster.
Chapter 6
OLIPHANT Procedure

Overview: OLIPHANT Procedure ........................................... 95
What about the SAS Data in HDFS Engine? .............................. 95
What Does the OLIPHANT Procedure Do? .............................. 95
Understanding How SAS LASR Analytic Server Uses HDFS .......... 96

Concepts: OLIPHANT Procedure ........................................... 96
Adding Big Data ................................................................. 96
Adding Small Data .............................................................. 96

Syntax: OLIPHANT Procedure ............................................... 97
PROC OLIPHANT Statement ................................................. 97
ADD Statement ................................................................. 98
REMOVE Statement ............................................................ 99
DETAILS Statement ............................................................ 99

Examples: OLIPHANT Procedure ........................................... 100
Example 1: Adding and Removing Files in HDFS ....................... 100
Example 2: Querying File Details from HDFS ............................ 101

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 131.

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 100

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 100

**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 101
Syntax

DETAILS <details-statement-options>;

Details Statement Options

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

Alias ALLFILES

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

Alias COLUMNINFO

FILE= SASDAT-file
specifies the name of the SASDAT file to use. Do not specify a fully qualified HDFS path. Do not enclose the value in quotation marks. Do not include the SASDAT filename suffix. The name is converted to lowercase and the filename of the SASDAT 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 SASDAT files that are found in the path and child directories.

ROWCOUNT
includes the number of observations in the specified SASDAT 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 SASDAT file is removed.

Program

libname hrdata */data/hr/2011; 

proc oliphant host=“grid001.example.com” install=/opt/TKGrid;
add hrdata.emps blocksize=16M path="/sasdata/2011/" replace;

add (label='Bonuses for 2011') hrdata.bonus path="/sasdata/2011";
remove salary path="/sasdata/2011";
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";
    details path="/sasdata/2011/* recursive;

details file=emps path="/sasdata/2011/" column;
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.
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>;
   SERVERINFO <option>;
   SERVERPARM <option>;
   SERVERTERM <options>;
   SERVERWAIT <options>;
   TABLEINFO <!options>;
QUIT;

Example: Copying Tables from One Hadoop Installation to Another
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

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.

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.
Example: Copying Tables from One Hadoop Installation to Another

Details
This example does not apply to a 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

```sas
proc lasr create port=12636 serverpermissions=700;  
   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;  
   serverparm hadoophome="/olderhadoop/path";  
quit;

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

proc VASMP data=private.y2011(tag="dept.sales");  
   serverparm hadoophome="/newerhadoop/path";
run;
   save path="/dept/sales/";  
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 8
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 120
- “Example 5: Accessing Tables Loaded with the LASR Procedure” on page 121
- “Example 6: Accessing Tables That Are Loaded from HDFS” on page 121

---

**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 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.
• The engine guarantees the data order for a particular configuration of worker nodes. If you load a table in to a server and you retrieve the data three times, the order of the data is the same. However, if you start another server and load the same table into a different number of worker nodes, then the order in which you retrieve the data is different. However, it is reproducible within fetches from a single server.
• Any order-dependent operation, such as the LAG or DIF functions, cannot rely on stability of results beyond that which can be guaranteed by the distribution model of the data.
Chapter 9
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>
   <NODEFAULTFORMAT= YES | NO>
   <STARTSERVER <=(non-distributed-server-options)>
   <SIGNER= "authorization-web-service-uri">
   <VERBOSE= YES | NO>;

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.

**NODEFAULTFORMAT= YES | NO**
specifies whether a default format that is applied to a variable is reported by the engine.

If you do not specify a format for a variable when you add a table to the server, the engine adds a default format. The server applies BEST. for numeric variables and $. for character variables.

The engine displays this "forced" format in procedures that list variable attributes, such as the CONTENTS and DATASETS procedures. If you specify NODEFAULTFORMAT=YES, then the display of the "forced" format is suppressed.

*Note:* This setting does not control whether formats are applied to a variable.
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 software. For more information, see SAS Visual Analytics: Administration Guide.

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

STARTSERVER= YES | NO
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:

AFFINITY= YES | NO
requests that the concurrently executing threads of the server are associated with specific CPUs. When thread affinity is set to YES, a thread does not bounce between CPUs.

Default NO

CLF= YES | NO
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= YES | NO
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= YES | NO
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=\textit{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 \textit{n} is the bin limit.

MERGEBINS=\textit{b}

specifies the number of bins that numeric variables are binned into when
MERGELIMIT=\textit{n} is reached.

NOHOSTCHECK = YES | NO

specifies that the server does not check that the host name specified in the
HOST= option is the local host. This option can be useful with unusual network
configurations.

Interaction When the SIGNER= option is also specified, the host name that is
specified in the HOST= option is sent to the SAS LASR
Authorization Service.

NTHREADS=\textit{n}

specifies the number of threads to use for the server. By default, \textit{n} equals the
number of CPU cores on the machine.

PATH="\textit{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=\textit{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 \textbf{PERM=}

Range \textbf{600 to 777}

Alias \textbf{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.

VERBOSE= YES | NO

specifies whether the engine accepts and reports extra messages from TKGrid.
Specifying VERBOSE=YES can help diagnose problems with passwordless SSH
setups, grid install locations, and so on.

The following message in the SAS log shows an example of a problem with
passwordless SSH configuration.

\textbf{ERROR: Failed to load the SAS LASR Analytic Server access extension}
in the distributed computing environment.

Server refused our key from:
/home/sasdemo/.ssh/id_rsa

\textbf{Timeout waiting for Grid connection.}
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:

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

proc lasr
   create port=10010
   path="/tmp" noclass;
   performance nodes=all;
run;

libname salessvr sasiola;
```

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:

```sas
proc lasr
   create="/tmp/hrsvr"
   path="/opt/VADP/var/hr"
   noclass;
   performance host="grid001.example.com" install="/opt/TKGrid" nodes=all;
run;

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

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.

```sas
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.

NOTE: The LASR Analytic Server port '12637' has been assigned to the macro variable "LASRPORT".

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

NOTE: Libref hrdata was successfully assigned as follows:
Engine: SASSOLA
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.

```sas
libname sales sasiola port=10010;

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

```sas
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 9.1  DATASETS Procedure Output Showing the WORK Server Tag

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 9.2  DATASETS Procedure Output Showing the SASHELP Server Tag

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.

```sas
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 9.3  DATASETS Procedure Output Showing the DEPT.SALES Server Tag

<table>
<thead>
<tr>
<th># Name</th>
<th>Member Type</th>
<th>Number of Rows</th>
<th>Number of Columns</th>
<th>Last Modified</th>
<th>Server Tag</th>
<th>Data Encoding</th>
<th>Owner</th>
</tr>
</thead>
<tbody>
<tr>
<td>SALES2012</td>
<td>DATA</td>
<td>452459</td>
<td>39</td>
<td>19Jul12:21:34:0</td>
<td>DEPT.SALES</td>
<td>SAS</td>
<td>sasdemo</td>
</tr>
</tbody>
</table>

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 13.

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

data lasrlib.prdsale(partition=(country region) orderby=(descending year));
  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.

Example 8: Creating an Empty Table
Creating an empty table can be useful to seed the column information for a table that you later append to. The following statements create an empty table with two numeric and two character variables:

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

data lasrlib.empty;
  length c1 $15;
```
length c2 $12;
x=1;
y=1;
c1="";
c2="";
delete;
run;

Program Description

The Empty table is added to the server.
Chapter 10
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.

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

Syntax

APPEND=YES | NO

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.

Example Code 10.1 Using the APPEND= Data Set Option

```java
proc lasr add data=grpl.sales noclass port=10010;
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.

**ARRAY= Data Set Option**

requests that the name space for the table on the SAS session is extended with names that are derived from a temporary numeric array.

**Syntax**

```plaintext
ARRAY=(array-name, n)
```

**Details**

The following example shows how to specify a temporary numeric array with variables named Temp1–Temp4:

```plaintext
proc imstat;
  table lasrlib.sales (array=(temp,4));
```

The variables in the temporary numeric array do not exist in the in-memory table, but the SAS session assumes that they are there. Using temporary names this way can be useful when your SAS program refers to calculated temporary columns that do not exist when table is opened for input. For example, this option can enable you to retrieve the results for calculated columns with the FETCH statement of the IMSTAT procedure.

This option is used for numeric variables only. If you want to refer to specific temporary variable names, you can also use the TEMPNAMES= option. The TEMPNAMES= option enables you to specify the variable type, and in the case of character variables, the length of the variable.

**AUTOCOMMIT= Data Set Option**

specifies how rows are committed to an in-memory table during append operations.

**Syntax**

```plaintext
AUTOCOMMIT= nR
AUTOCOMMIT= kS
```
Details

By default, rows are not committed to an in-memory table until the DATA step completes. That is, the rows are held in intermediate storage areas in the server and are not included in requests for data or computational results until the DATA step completes. If you specify AUTOCOMMIT=n or AUTOCOMMIT=nR, then the server commits the rows when at least n rows have been received.

If you specify AUTOCOMMIT=kS, the server commits any rows received within k seconds of the start of the append operation or within k seconds of the previous commit.

**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=YES | NO
```

**Example**

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

**NODEFAULTFORMAT= Data Set Option**

specifies whether a default format that is applied to a variable is reported by the server.
**Syntax**

```plaintext
NODEFAULTFORMAT=YES | NO
```

**Details**

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

---

**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 122

**Syntax**

```plaintext
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.

```plaintext
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 122

**Syntax**

```plaintext
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.

SIGNER= Data Set Option
specifies the URI of the SAS LASR Authorization web service.

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

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.
**TEMPNAMES= Data Set Option**

requests that the name space for the table on the SAS session is extended with the specified names.

### Syntax

TEMPNAMES=(variable-name1, variable-name2...)

### Details

The following example shows how to specify two temporary numeric variables named Difference and Ratio:

```sas
proc imstat;
  table lasrlib.prdsale (tempnames=(difference ratio));
run;

fetch actual -- month predict ratio / tempnames=(difference ratio)
  tempexpress="difference = actual - predict; ratio = actual / predict";
run;
```

By default, temporary variables that are specified through the TEMPNAMES= option are of numeric type. To define a temporary character variable, follow the variable name with a dollar sign ($) and an optional length.

The following example shows the variable Cust_Name as a character variable with a length of 20:

```sas
table lasrlib.accounts(tempnames=(total_deposits
  cust_name $ 20
  deposit_count
  branch_count));
```

---

**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=YES | NO
Chapter 11

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
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 12
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=YES | NO>
  <NODEFAULTFORMAT=YES | NO>
  <VERBOSE=YES | NO>;
```

**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= YES | NO
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.

NODEFAULTFORMAT= YES | NO
specifies whether a default format that is applied to a variable is reported by the engine.

If you do not specify a format for a variable when you add a table to HDFS, the engine adds a default format. The server applies BEST. for numeric variables and $. for character variables.

The engine displays this "forced" format in procedures that list variable attributes, such as the CONTENTS and DATASETS procedures. If you specify NODEFAULTFORMAT=YES, then the display of the "forced" format is suppressed.

Note: This setting does not control whether formats are applied to a 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.

VERBOSE= YES | NO
specifies whether the engine accepts and reports extra messages from TKGrid. For more information, see the VERBOSE= option on page 118 for the SAS LASR Analytic Server engine.

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";
option set GRIDINSTALLLOC="/opt/TKGrid";
libname hdfs sashdat;
```
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.

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

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.

```sas
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 13.

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

data hdfs.prdsale(partition=(year month) orderby=(descending prodtype));
set sashelp.prdsale;
```
Example 5: Removing Tables from HDFS
Removing tables from HDFS can be performed with the DATASETS procedure.

```sas
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.

```sas
libname hdfs sashdat host="grid001.example.com" install="/opt/TKGrid" path="/dept";
proc hpds2
   in  = hdfs.allyears(where=(region=212))
   out = hdfs.avgsales(blocksize=32m copies=0);
   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.

```sas
libname csvfiles sashdat host="grid001.example.com" install="/opt/TKGrid" path="/user/sasdemo/csv";
```
proc contents data=csvfiles.rep(filetype=csv getnames=yes);  
run;

**Output 12.1  List the Variables in a CSV File with the CONTENTS Procedure**

<table>
<thead>
<tr>
<th>Data Set Name</th>
<th>/user/sasdemo/csv.csv</th>
<th>Observations</th>
<th>.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Member Type</td>
<td>DATA</td>
<td>Variables</td>
<td>6</td>
</tr>
<tr>
<td>Engine</td>
<td>SASHDAT</td>
<td>Indexes</td>
<td>0</td>
</tr>
<tr>
<td>Created</td>
<td>Tuesday, July 03, 2012 08:53:36 AM</td>
<td>Observation Length</td>
<td>208</td>
</tr>
<tr>
<td>Last Modified</td>
<td>Thursday, June 20, 2012 02:45:41 PM</td>
<td>Deleted Observations</td>
<td>0</td>
</tr>
<tr>
<td>Protection</td>
<td>Compressed</td>
<td></td>
<td>NO</td>
</tr>
<tr>
<td>Data Set Type</td>
<td>Sorted</td>
<td></td>
<td>NO</td>
</tr>
<tr>
<td>Label</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data Representation</td>
<td>SOLARIS_X86_64, LINUX_X86_64, ALPHA_TRU64, LINUX_IA64</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Encoding</td>
<td>Default</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### 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.

```sas
option set=GRIDHOST="grid001.example.com";
option set=GRIDINSTALLLOC="/opt/TKGrid";
libname csvfiles sashdat path="/user/sasdemo/csv";
proc hpds2 in=csvfiles.rep(filetype=csv getnames=yes) out=csvfiles.rephdat(path="/user/sasdemo" copies=0 blocksize=32m);
  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.
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 136

**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
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 13.1 Using the BLOCKSIZE= Data Set Option**

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

### COLUMNS= Data Set Option

This 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 `VARn`.

**Example**

```sas
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 `VARn`, 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

- COLUMN= 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= YES | NO

Details

Specify GETNAMES=YES 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= YES | NO

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 GETOBS=YES, the engine calculates the number of observations in a SASHDAT file.

ods select attributes;

proc datasets library=hdfs;
   contents data=sales2012(getobs=yes);
run;
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=YES | NO

Example

data hdfs.transactions(partition=(cust_id year) hash=yes);
  set somelib.sometable;
run;
**LOGUPDATE= Data Set Option**

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

**Syntax**

```
LOGUPDATE= YES | NO
```

**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=yes);
   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 135

**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.

```sas
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 135

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= YES | NO
About Programming

When programming with SAS LASR Analytic Server, it is important to understand where the computation occurs and memory utilization.

- The IMSTAT procedure always performs the computation in the server, and the analysis is performed against the original in-memory table.
- Other procedures (for example, FREQ, UNIVARIATE, and RANK) transfer the in-memory table to the client machine. After the transfer, the session on the client machine performs the analysis on the copy of the data.
- Most DATA step programs operate by transferring the in-memory table to the client and then performing the computation. However, if a DATA step is written to use in-memory tables for input and output, the DATA step can run in-memory, with restrictions. The next section describes how to use this feature.

DATA Step Programming for Scoring In SAS LASR Analytic Server

Scoring In-Memory Tables Using DATA Step Processing

The DATA step can process in-memory tables for scoring with limitations:

- Only one input file and one output file is allowed.
- Only functions and formats that are supported by the DS2 language compile successfully.
• Some DATA step statements are not allowed, such as those pertaining to input and output.

For more information, see “Requirements for LASR Score Mode DATA Step Processing” on page 151 and “Restrictions in DATA Step Processing” on page 152.

To enable the DATA step to score in-memory SAS tables, set the system option DSACCEL=ANY.

If a SAS program does not meet the requirements for running in the SAS LASR Analytic Server, SAS writes informational or error messages to the log and executes the code in your Base SAS session. In this case, SAS reads and writes large tables over the network.

You can determine whether your code is compliant with the SAS LASR Analytic Server compiler by setting the system option MSGLEVEL= to I. When MSGLEVEL=I, SAS writes log messages that identify the non-compliant code.

Example 1: A DATA Step Program For SAS LASR Analytic Server

This example demonstrates executing a DATA step program in the SAS LASR Analytic Server:

```sas
/* Enable DATA step parallel processing using the system option   */
/* and enable messages to view non-compliant code in the SAS log. */
options dsaccel=any msglevel=i;

/* Create a libref for in-memory tables. */
libname lasr sasiola host="grid001.example.com" port=10010 tag='hps';

/* Create a libref for the input data that is stored on disk. */
libname score '/myScoreData/';

/* Load the input table into memory */
data lasr.intr;
  set score.intrid;
run;

/* Execute the score code using the in-memory tables. */
/* Both tables must use the same libref. */
data lasr.sumnormtable;
  set lasr.intr;
run;

/* Execute the score code. */
if sum > 1000
  then score=1;
run;
```

Example 2: Using User-Defined Formats with In-Memory Tables

You can use user-defined formats in a DATA step by using the CATALOG procedure to copy the format catalog to a library. This example copies the format library to Work:

```sas
/* Enable DATA step parallel processing using the system option   */
/* and enable messages to view non-compliant code in the SAS log. */
```
options dsaccel=any msglevel=i;

/* Create a libref for the in-memory tables. */
libname lasr sasiola host="grid001.example.com" port=10010 tag='hps';

/* Create a libref for the input data and format catalog that is stored on disk. */
libname score '/myScoreData/';

/* Copy the demx format catalog to the Work library */
proc catalog catalog=score.dmex;
  copy out=work.formats;
quit;

/* Enable in-memory processing (dsaccel) and load the input table into memory. */
data lasr.dmex;
  set score.dmex;
run;

/* Enable in-memory processing (dsaccel) and execute the score code using the in-memory tables. */
data lasr.dmexout;
  set lasr.dmex;
  %inc "dmex.sas";
run;

SAS automatically searches the Work and Library libraries for a format catalog. If you copy the format library to a library other than Work or Library, then you must use the FMTSEARCH= system option to let SAS know the location of the format library.

options fmtsearch=(myFmtLib);
You must also specify the FMTSEARCH= system option if the format catalog name is not format:

options fmtsearch=(myFmtLib.myFmts);

Requirements for LASR Score Mode DATA Step Processing

In order to score in-memory tables in SAS LASR Analytic Server, the following is required:

• The DSACCEL=ANY system option is set.
• The code must contain a LIBNAME statement using the SASIOLA engine.
• The input and output tables must use the same libref for the SASIOLA engine.
• The DATA statement must be followed immediately by the SET statement.

This example demonstrates these requirements:

libname lasr sasiola;
data lasr.out;
  set lasr.in;
  /* DATA step code */
run;
Restrictions in DATA Step Processing

Here are the restrictions for using the DATA step in SAS LASR Analytic Server:

- More than one SET statement is not supported. SET statement options are not allowed.
- These statements are not supported:
  - BY (or FIRST. and LAST. variables)
  - CONTINUE
  - DISPLAY
  - FILE
  - Sub-setting IF
  - INFILE
  - INPUT
  - LEAVE
  - MERGE
  - MODIFY
  - OUTPUT
  - PUT
  - REMOVE
  - RENAME
  - REPLACE
  - RETAIN
  - UPDATE
  - WHERE
  - WINDOW
- The ABORT statement has these restrictions:
  - The ABORT statement does not accept arguments.
  - The ABORT statement is not supported within functions. It is valid only in the main program.
- These functions are not supported:
  - DIF
  - LAG
- The INPUT function does not support the question mark (?) and double question mark (??) modifiers.
- Some CALL routines are not supported. Routines are supported if there is an equivalent function.
- You can use only SAS formats and functions that are supported by the DS2 language. These formats and functions are documented in *SAS DS2 Language Reference.*
• Component objects are not supported.
• Scoring input variables cannot be modified.
Appendix 1
Removing a Machine from the Cluster

About Removing a Machine from the Cluster

This information applies to deployments that use a distributed SAS LASR Analytic Server and SAS High-Performance Deployment of Hadoop. This information does not apply to non-distributed deployments.

For deployments that use Teradata EDW, Greenplum DCA, or a commercial distribution of Hadoop, such as Cloudera or Hortonworks, follow the product documentation for the appliance or cluster. If it is possible to run the appliance or cluster in a degraded state, you can follow the steps to remove the host name from the grid.hosts file. This can make it possible to start and run SAS LASR Analytic Server instances.

Which Servers Can I Leave Running When I Remove a Machine?

In most cases, all SAS LASR Analytic Server instances stop automatically if communication with any machine in the cluster fails. However, if any servers are still running, stop them.

Stop SAS High-Performance Deployment of Hadoop to avoid logging messages about the machine being unavailable. For SAS Visual Analytics deployments, stop the SAS LASR Analytic Server Monitor, too.
Remove the Host Name from the grid.hosts File

The host name for the machine to remove must be removed from the /opt/TKGrid/grid.hosts file. (The path might be different for your deployment.)

To remove the host name:

1. Log on to the root node for the deployment as the root user or the user ID that installed the High-Performance Analytics Environment.
2. Edit the /opt/TKGrid/grid.hosts file and delete the line that includes the host name to remove. Save and close the file.
3. Copy the updated grid.hosts file to the same location on all the machines in the cluster. You can use the simcp command to perform this task:

   /opt/TKGrid/bin/simcp /opt/TKGrid/grid.hosts /opt/TKGrid

If your deployment uses the SAS High-Performance Computing Management Console, remove the host name from the /etc/gridhosts file, too.

Remove the Host Name from the Hadoop Slaves File

If the machine is still running, stop any Hadoop processes that are running on it before you continue.

To remove the host name from the Hadoop slaves file:

1. Log on to the root node for the deployment as the root user or the user ID that installed the SAS High-Performance Deployment of Hadoop.
2. Edit the /hadoop/hadoop-0.23.1/etc/hadoop/slaves file and delete the line that includes the host name to remove. Save and close the file.

The path might be different for your deployment.

Restart Servers

You can restart SAS LASR Analytic Server instances at this point. When you load tables from HDFS, data are loaded from redundant blocks on the remaining machines.

When the machine becomes available again (such as replacement hardware or new hardware), follow the steps in “Adding Machines to the Cluster”.
Appendix 2
Adding Machines to the Cluster

About Adding Machines to the Cluster .................................................. 157
Which Servers Can I Leave Running When I Add a Machine? .................. 158
Configure System Settings ................................................................. 158
Add Host Names to Gridhosts ............................................................. 159
Propagate Operating System User IDs .................................................. 160
  Which User IDs Must Be Propagated? .................................................. 160
  About Passwords .................................................................................. 160
  Option 1: User Management Software .................................................. 160
  Option 2: Delete and Re-Add Users ...................................................... 160
  Option 3: Use Operating System Commands to Add Users ................. 160
  Add Groups ......................................................................................... 161
  Add Users ............................................................................................ 161
  Propagating Secure Shell Keys ............................................................ 161
Configure SAS High-Performance Deployment of Hadoop ....................... 161
  Install a Java Runtime Environment .................................................. 161
  Install Hadoop on the Additional Machines ........................................ 162
  Update the Hadoop Configuration Files ............................................. 162
  Start the DataNodes on the Additional Machines .............................. 163
  Copy a File to HDFS ........................................................................ 163
Configure SAS High-Performance Analytics Infrastructure ....................... 164
  Strategies ......................................................................................... 164
  Option 1: Re-Install the Software ....................................................... 164
  Option 2: Copy the Software ............................................................ 164
  Validate the Change ......................................................................... 164
Restart SAS LASR Analytic Server Monitor ........................................... 165
Restart Servers and Redistribute HDFS Blocks ...................................... 165
View Explorations and Reports ............................................................ 165

About Adding Machines to the Cluster

This information applies to deployments that use a distributed SAS LASR Analytic Server and SAS High-Performance Deployment of Hadoop. These steps do not include information about licensing SAS software for additional CPUs. Contact your SAS
representative for information about getting additional licensing and applying the license.

This information does not apply to non-distributed deployments.

For deployments that use Teradata EDW, Greenplum DCA, or a commercial distribution of Hadoop, follow the product documentation for information about expanding the appliance or cluster. After following those procedures, install the SAS High-Performance Analytics environment as described in *SAS High-Performance Analytics Infrastructure: Installation and Configuration Guide*.

---

### Which Servers Can I Leave Running When I Add a Machine?

It is best to stop all SAS LASR Analytic Server instances and SAS High-Performance Deployment of Hadoop. For SAS Visual Analytics deployments, stop the SAS LASR Analytic Server Monitor, too.

If you prefer to deny access to the environment while performing this procedure, you can use the SAS High-Performance Computing Management Console to perform an SSH lockout. Be sure to permit access for root, the SAS installer account, the account that is used to run HDFS, and at least one administrator that can start and stop server instances.

You can ensure that saving files to HDFS is denied by using safe mode. The following command is an example:

```
$HADOOP_HOME/bin/hdfs dfsadmin -safemode enter
```

### Configure System Settings

Each of the additional machines must be configured identically to the existing machines with regard to operating system, drivers, and tuning settings. The high-level tasks are as follows:

- Configure passwordless SSH for the root user ID.
- Disable SELinux if it is disabled on the existing machines.
- Modify `/etc/ssh/sshd_config` with the following setting:
  
  `MaxStartups 1000`

- Modify `/etc/security/limits.conf` with the following settings:
  
  - `soft nproc 65536`
  - `hard nproc 65536`
  - `soft nofile 350000`
  - `hard nofile 350000`

- Modify `/etc/security/limits.d/90-nproc.conf` with the following setting:
  
  `soft nproc 65536`

- Modify `/etc/sysconfig/cpuconfig` with the following setting:
Add Host Names to Gridhosts

If you use SAS High-Performance Computing Management Console to manage your cluster, you must add the host names for the additional machines to the /etc/gridhosts file. You can do this with a text editor, or you can use the Gridhosts File Management feature in the console. If you do not use the console for managing the cluster, then you do not need to perform this task.

If you use the console to add the hosts, make sure that the SSH Connectivity column indicates Yes for all the machines. The following display shows an example.

A No in the SSH Connectivity column indicates that passwordless SSH for the root user ID is not configured properly for the machine identified in the Node Name column.
Propagate Operating System User IDs

Which User IDs Must Be Propagated?

You must propagate the operating system user IDs that are used for Hadoop and for managing SAS LASR Analytic Server instances (starting, stopping, and loading tables).

About Passwords

In most deployments, passwords are not used to log on to the machines in the cluster. Therefore, in most cases, it is not necessary to propagate passwords to the additional machines in the cluster.

However, if you want to preserve passwords and you use the SAS High-Performance Computing Management Console, you can view the value from the /etc/shadow file or within the console. When you add the user with the console, you can paste the value in the Pre-encrypted password field.

Option 1: User Management Software

If your site uses account management software applications such as LDAP, NIS, or Active Directory on Linux for managing user accounts, then use that software to make the user accounts available on the additional machines.

Option 2: Delete and Re-Add Users

If you use SAS High-Performance Computing Management Console to manage the machines in the cluster, then you can delete each user and then add the user back to the system. This is an option when the number of operating system accounts is fairly low. When you add the user, the account is re-created on the original machines, and it is also added to the new machines.

Note: When using this method, be sure to note the UID and primary GID of the user before it is deleted and to reuse the same values when re-creating the user account.

Note: If you choose this option, be aware that using the Generate and Propagate SSH Keys option when you create the user account removes existing SSH keys. If you do not delete the home directory when you delete the user and you do not generate new SSH keys, then the existing keys can be reused.

Option 3: Use Operating System Commands to Add Users

About Using the Simultaneous Shell Command

You can view the existing user IDs and groups with the SAS High-Performance Computing Management Console, or from the /etc/passwd and /etc/group files. You can use operating system commands to add the groups and users to the addition machines.

The following sample commands use the simsh utility that is included with SAS High-Performance Computing Management Console. This utility attempts to configure users
and groups on the machines that already have the user accounts and groups. This results in an error message from those machines. The error message is harmless because the commands do not modify the existing configuration, but you might find them distracting. As an alternative, you can use the console to create a simultaneous utilities machine group that contains the host names for the new machines only. You can then specify the group name with the `simsh` command so that only the new machines are affected.

This document demonstrates using the `simsh` command with a machine group that is named `newnodes` for simplicity.

### Add Groups

Identify the groups to add to the new machine by viewing the console or looking at the `/etc/group` file. Make sure that you identify each group ID.

The following example shows how to add the group that is named `sasdemo` with a group number of 102 to the `newnodes` machine group:

```
/opt/webmin/utilbin/simsh -g newnodes "groupadd -g 102 sasdemo"
```

### Add Users

Identify the user IDs to add to the new machine by viewing the console or looking at the `/etc/passwd` file. Make sure that you identify each user ID.

The following example shows how to add a user:

```
/opt/webmin/utilbin/simsh -g newnodes
   "useradd -u 503 -g 102 -d /home/sasdemo -s /bin/bash sasdemo"
```

*Note:* The command must be entered on a single line.

### Propagating Secure Shell Keys

One way to propagate existing SSH keys to the new machines is to copy them to all the new machines in the cluster. The following example shows one way to perform this operation:

```
simcp /home/user/.ssh /home/user/
simsh chown -R user:group /home/user/.ssh
```

You can use scripting to simplify this task and all the previous operating system commands, too. You also do not need to follow this strategy. Any method that is able to propagate the groups, user IDs, and SSH keys is acceptable.

---

## Configure SAS High-Performance Deployment of Hadoop

### Install a Java Runtime Environment

A Java Runtime Environment or Java Development Kit is necessary to run SAS High-Performance Deployment of Hadoop on the additional machines. Be sure to install it in
the same path as the existing machines. One way to perform this task is to use the
`simcp` command as shown in the following example:

```
./opt/webmin/utilbin/simsh -g newnodes mkdir -p /data/java/
```

```
./opt/webmin/utilbin/simcp -g newnodes /data/java/jdk_1.6.0_29
```

```
./data/java/
```

### Install Hadoop on the Additional Machines

Install Hadoop on the additional machines with the same package,
`sashadoop.tar.gz`, that is installed on the existing machines. Install the software
with the same user account and use the same installation path as the existing machines.

Typically, you create a text file with the host names for all the machines in the cluster
and supply it to the installation program. In this case, create a text file with the host
names for the new machines only. Use a filename such as `~/addhosts.txt`. When
you run the installation program, `hadoopInstall`, supply the fully qualified path to
the `addhosts.txt` file.

The previous tasks result in a new cluster that is independent of the existing cluster.
When the configuration files are overwritten in the next step, the additional machines no
longer belong to their own cluster. They become part of the existing cluster.

When you run the installation program on the new machines, if you are unsure of the
directory paths to specify, you can view the following files on an existing machine:

```
$HADOOP_HOME/etc/hadoop/hdfs-site.xml
```

Look for the values of the `dfs.name.dir` and `dfs.data.dir` properties.

```
$HADOOP_HOME/etc/hadoop/mapred-site.xml
```

Look for the values of the `mapred.system.dir` and `mapred.local.dir`
properties.

### Update the Hadoop Configuration Files

As the user ID that runs HDFS, modify the `$HADOOP_HOME/etc/hadoop/slaves`
file on the existing machine that is used for the NameNode. Add the host names of the
additional machines to the file.

You can use the `simcp` command to copy the file and other configuration files to the
new machines:

```
./opt/webmin/utilbin/simcp -g newnodes $HADOOP_HOME/etc/hadoop/slaves
```

```
$HADOOP_HOME/etc/hadoop/
```

```
./opt/webmin/utilbin/simcp -g newnodes $HADOOP_HOME/etc/hadoop/master
```

```
$HADOOP_HOME/etc/hadoop/
```

```
./opt/webmin/utilbin/simcp -g newnodes $HADOOP_HOME/etc/hadoop/core-site.xml
```

```
$HADOOP_HOME/etc/hadoop/
```

```
./opt/webmin/utilbin/simcp -g newnodes $HADOOP_HOME/etc/hadoop/hdfs-site.xml
```

```
$HADOOP_HOME/etc/hadoop/
```

Start the DataNodes on the Additional Machines

For each of the new machines, run a command that is similar to the following example:

```
ssh hostname /data/hadoop/hadoop-0.23.1/sbin/hadoop-daemon.sh start datanode
```

**Note:** Run the command as the user account that is used for HDFS. Make sure that you specify the actual path to `hadoop-daemon.sh`. Once you have started the DataNode process on each new machine, view the [http://namenode-machine:50070/dfshealth.jsp](http://namenode-machine:50070/dfshealth.jsp) page to view the number of live nodes.

Run `$HADOOP_HOME/bin/hdfs dfsadmin -printTopology` to confirm that the new machines are part of the cluster. The following listing shows a sample of the command output:

```
Rack: /default-rack
  192.168.8.148:50010 (grid103.example.com)
  192.168.8.153:50010 (grid104.example.com)
  192.168.8.217:50010 (grid106.example.com)
  192.168.8.230:50010 (grid105.example.com)
  192.168.9.158:50010 (grid099.example.com)
  192.168.9.159:50010 (grid100.example.com)
  192.168.9.160:50010 (grid101.example.com)
```

Copy a File to HDFS

If you put HDFS in safe mode at the beginning of this procedure, leave that state with a command that is similar to the following:

```
$HADOOP_HOME/bin/hdfs dfsadmin -safemode leave
```

To confirm that the additional machines are used, you can copy a file to HDFS and then list the locations of the blocks. Use a command that is similar to the following:

```
$HADOOP_HOME/bin/hadoop fs -D dfs.blocksize=512 -put /etc/fstab /hps
```

**Note:** The very small block size shown in the example is used to increase the number of blocks written and increase the likelihood that the new machines are used.

You can list the block locations with a command that is similar to the following:

```
$HADOOP_HOME/bin/hdfs fsck /hps/fstab -files -locations -blocks
```

Review the output to check for IP addresses for the new machines.
Configure SAS High-Performance Analytics Infrastructure

Strategies

The SAS High-Performance Analytics Infrastructure software must be installed on the new machines in the cluster. In addition, the existing installations must be updated so that the `grid.hosts` file includes the new host names.

The first option is to re-install the software. This adds the software to the new machines and updates the `grid.hosts` file. This option has the advantage of being very simple.

The second option is to copy the files to the new machines and then copy an updated `grid.hosts` file to all the machines.

Tip: Installing the software to a new directory is not suggested because the path to the software might be specified in numerous server definitions that are registered in SAS metadata.

Option 1: Re-Install the Software

The software is installed by running the `TKGrid_Linux_x86_64.sh` executable. For details about installing the software, see SAS High-Performance Analytics Infrastructure: Installation and Configuration Guide.

If you choose this option, stop all SAS LASR Analytic Server instances. Stopping the servers avoids the possibility of errors related to overwriting executables and libraries.

Option 2: Copy the Software

Use a command that is similar to the following for copying the software to the new machines:

```
/opt/webmin/utilbin/simcp -g newnodes /opt/TKGrid/ /opt/TKGrid/
```

Modify the `/opt/TKGrid/grid.hosts` file and add the host names for the new machines. Then, copy the file to all machines, even the existing machines:

```
/opt/webmin/utilbin/simcp /opt/TKGrid/grid.hosts /opt/TKGrid/
```

Validate the Change

Use the `mpirun` command to confirm that the new machines are accessible.

```
cd /opt/TKGrid/mpich2-install

./bin/mpirun -f ../grid.hosts hostname
```
The **hostname** command is run on each machine and the results are returned. Make sure that the response includes all the host names in the cluster.

```
grid098.example.com
grid103.example.com
grid106.example.com
grid100.example.com
grid105.example.com
grid101.example.com
grid104.example.com
grid099.example.com
```

---

### Restart SAS LASR Analytic Server Monitor

For SAS Visual Analytics deployments, restart the monitor:

```
cd SAS-config-dir/Levn/Applications/SASVisualAnalyticsX.X/
HighPerformanceConfiguration/LASRMonitor.sh restart
```

Restarting the monitor causes an error for any users logged on to SAS Visual Analytics Administrator. Those users need to log off and log on again.

---

### Restart Servers and Redistribute HDFS Blocks

Log on to SAS Visual Analytics Administrator and perform the following steps:

1. Select **LASR ⇒ Monitor Resources** and sure that the additional machines appear in the Real-Time View.
2. Select **LASR ⇒ Manage Servers** and start each of the servers.
3. Select **Tools ⇒ Explore HDFS** and review the block distribution.

Any tables that are loaded from HDFS cannot initially use the additional hardware because the blocks have not been replicated to the additional machines. Within SAS Visual Analytic Administrator, you can view the block distribution from the **HDFS** tab to confirm that blocks are on the original machines only.

Hadoop includes a balancer process that can move blocks to under-utilized machines. The process is intentionally slow so that resource consumption is low and so that it does not interfere with other tasks on the cluster. To begin the balancing process:

```
./hadoop-daemon.sh start balancer
```

As an alternative, if adding the data to HDFS again is possible, then you can delete files with SAS Visual Analytics Administrator and then add them back.

---

### View Explorations and Reports

To confirm that the additional machines are working as intended, view existing explorations and reports.
If you did not delete and then add the data back to HDFS, then make sure that you view explorations and reports that use data that is streamed to the server instance (instead of being loaded from HDFS). Or, make sure that new data sets are added to HDFS and create explorations and reports from the new data.
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.

colocated 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

A
appending data  122

B
BALANCE statement
IMSTAT procedure  48

C
COLUMNINFO statement
IMSTAT procedure  49
COMPUTE statement
IMSTAT procedure  50
connecting to a Hadoop NameNode
LIBNAME statement, SAS Data in HDFS engine  134
connecting to a server
LIBNAME statement, SAS LASR Analytic Server engine  119

D
DELETEROWS statement
IMSTAT procedure  50
DISTRIBUTIONINFO statement
IMSTAT procedure  51
DROPTABLE statement
IMSTAT procedure  52

E
data engine  111, 131

F
FETCH statement
IMSTAT procedure  52
formats
LASR procedure  35
FREE statement

G
gpfdist
distributing data  8
Greenplum
distributing data  6

H
HDFS
accessing with the SAS LASR Analytic Server engine  121
HPDS2 procedure
SAS Data in HDFS engine  136

I
IMPORTCUBE statement
IMSTAT procedure  54
IMSTAT procedure examples
appending tables  84
appending tables to partitioned tables  85
partitioning  75
promote temporary tables  77
rebalancing a table  78
saving tables  81
star schema  82
storing temporary variables  86
IMXFER procedure  89
IMXFER procedure examples
copying a table  93, 94

L
LABEL= option  99
LASR procedure
  accessing tables with the SAS LASR Analytic Server engine 121
  compared with the SAS LASR Analytic Server engine 112
  concepts 23
  FMTLIBXML= option 37
LASR procedure examples
  load a table from Greenplum 35
  load a table from Teradata 34
  Loading a table from HDFS 33
  logging 33
  saving tables 38
  starting a server 32
  stopping a server 36
  unload a table from memory 36
  user-defined formats 37
  working with formats 37
LIBNAME statement, SAS Data in HDFS engine
  syntax 133
LIBNAME statement, SAS LASR Analytic Server engine
  syntax 115
LIFETIME statement
  IMSTAT procedure 55
  logging
  default log file location 14
  insufficient authorization 15
NUMROWS statement
  IMSTAT procedure 56
OLIPHANT procedure
  concepts 95
  syntax 97
OLIPHANT procedure examples
  adding files to HDFS 100
  querying file details from HDFS 101
PARTITION statement
  IMSTAT procedure 56
PARTITIONINFO statement
  IMSTAT procedure 57
PERFORMANCE statement
  LASR procedure 29
PROC IMSTAT statement 47
PROC IMXFER statement 90
PROC LASR statement 24
PROC OLIPHANT statement 97
PROC VASMP statement 104
PROMOTE statement
  IMSTAT procedure 58
PURGETEMPTABLES statement
  VASMP procedure 59
QUIT statement
  VASMP procedure 75, 93, 107
REMOVE statement
  LASR procedure 31
REPLAY statement
  IMSTAT procedure 59
SAS Data in HDFS engine
  BLOCKSIZE= data set option 139
  COPIES= data set option 141, 142
  FILETYPE= data set option 141
  GUESSROWS= data set option 143
  HASH data set option 143
  how it works 131
  LOGUPDATE data set option 144
  ORDERBY= data set option 144
  PARTITION= data set option 145
  PATH= data set option 146
  PERM= data set option 146
  requirements for using 132
  what is supported 132
SAS LASR Analytic Server engine
  accessing tables loaded from HDFS 121
  accessing tables loaded with a DATA step 120
  accessing tables loaded with the LASR procedure 121
  APPEND= data set option 125
  ARRAY= data set option 126
  AUTOCOMMIT= data set option 126
  compared with the LASR procedure 112
  FORMATEXPORT= data set option 127
  HASH= data set option 127
  ORDERBY= data set option 128
  PARTITION= data set option 128
  PERMISSION= data set option 129
  SIGNER= data set option 129
  TAG= data set option 129
  TEMPNAMES= data set option 130
  UCA= data set option 130
understanding server tags 112
SAS LASR Analytic Server engine SAS
  how it works 111
requirements for using 113
what is supported 113
SAS/ACCESS engines 8
SAVE statement
  IMSTAT procedure 60
  LASR procedure 31
SCHEMA statement
  IMSTAT procedure 61
SCORE statement
  IMSTAT procedure 64
server run time 7
SERVER statement 91
server tag 112
  accessing a table loaded with a DATA step 120
  accessing tables loaded from HDFS 121
  accessing tables loaded with the LASR procedure 121
SERVERINFO statement 104
  IMSTAT procedure 67
SERVERPARAM statement 105
  IMSTAT procedure 68
SERVERTERM statement 106
  IMSTAT procedure 69
SERVERWAIT statement
  IMSTAT procedure 69
  VASMP procedure 106
SET statement
  IMSTAT procedure 70
specifying host and port
LIBNAME statement, SAS LASR Analytic Server engine 120
specifying host and software installation location
LIBNAME statement, SAS Data in HDFS engine 135
STORE statement
  IMSTAT procedure 71

T
TABLE statement 91
  IMSTAT procedure 72
TABLEINFO statement 107
  IMSTAT procedure 72
Teradata
  distributing data 9

U
UPDATE statement
  IMSTAT procedure 73

V
VASMP procedure 103
VASMP procedure examples
  more than one Hadoop installation 108