THE POWER TO KNOW.

SAS® 9.3 Interface to Hadoop
Reference

SAS® Documentation

The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2012. *SAS® 9.3 Interface to Hadoop: Reference*. Cary, NC: SAS Institute Inc.

SAS® 9.3 Interface to Hadoop: Reference

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

1st electronic book, March 2012

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

Contents

About This Document

Audience

This document explains and shows how to use the SAS interfaces to Apache Hadoop. These interfaces include SAS/ACCESS to Hadoop, the HADOOP procedure, and the FILENAME statement for the Hadoop access method.

Information in this document is intended for application programmers and end users with these skills.

- They are familiar with Hadoop and the Hadoop Distributed File System (HDFS), and they have knowledge of fundamental Hadoop library and processing using HDFS. They are also familiar with the basics of Hadoop, Hive, and HiveQL. For details, see the Apache Hadoop and Hive Web sites at **`http://hadoop.apache.org`** and **`https://cwiki.apache.org/confluence/display/Hive`**.
- They know how to use their operating environment.
- They can use basic SAS commands and statements.

Database administrators might also want to read this document to understand how to implement and administer the SAS/ACCESS interface.

Recommended Reading

Here is the recommended reading list for this title.

- *Base SAS Procedures Guide*
- *SAS/ACCESS for Relational Databases: Reference*
- SAS® Companion that is specific to your operating environment
- *SAS Data Set Options: Reference*
- *SAS Language Reference: Concepts*
- *SAS Statements: Reference*

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

SAS Publishing Sales
SAS Campus Drive
Cary, NC 27513-2414
Phone: 1-800-727-3228
Fax: 1-919-677-8166
E-mail: sasbook@sas.com
Web address: support.sas.com/bookstore

Chapter 1

SAS/ACCESS Interface to Hadoop

Introduction to SAS/ACCESS Interface to Hadoop

This section describes SAS/ACCESS Interface to Hadoop. For a list of SAS/ACCESS features that are available in this interface, see "SAS/ACCESS Interface to Hadoop: Supported Features" on page 2.

For concepts, usage, and reference information that apply to all SAS/ACCESS interfaces, see *SAS/ACCESS for Relational Databases: Reference*.

SAS/ACCESS Interface to Hadoop: Supported Features

Here are the features that SAS/ACCESS Interface to Hadoop supports.

Table 1.1 *Features by Host Environment for Hadoop*

Platform	SAS/ACCESS LIBNAME Statement	SQL Pass-Through Facility	ACCESS Procedure	DBLOAD Procedure	Bulk-Load Support*
Linux x64	X	X			X
Microsoft Windows x64	X	X			X

* SAS/ACCESS to Hadoop has no differentiation between a bulk load and a standard load process.

For information about these features, see "Methods for Accessing Relational Database Data" in *SAS/ACCESS for Relational Databases: Reference*.

For detailed information about the following topics, see the SAS system requirements and configuration guide documents, which are available at `http://support.sas.com`.

- supported Hadoop versions, character data formats support, and required jar files, environmental variables, and patches—along with other prerequisites, required setup, and considerations for SAS/ACCESS Interface to Hadoop
- how SAS/ACCESS Interface to Hadoop interacts with Hadoop through Hive

LIBNAME Statement Specifics for Hadoop

Overview

This section describes the LIBNAME statement that SAS/ACCESS Interface to Hadoop supports and includes examples. For details about this feature, see "LIBNAME Option for Relational Databases" in *SAS/ACCESS for Relational Databases: Reference*.

Here is the LIBNAME statement syntax for accessing Hadoop.

LIBNAME *libref* **hadoop** <*connection-options*> <*LIBNAME-options*>

Security Limitations

SAS/ACCESS uses the Hadoop Hive Server to read Hadoop data. In creating a JDBC connection to Hive, SAS/ACCESS places the user ID and password that you provided in the JDBC connection string. Through Hive 0.7.1, JDBC ignores these credentials, instead associating Hive permissions with the UNIX user ID that started the Hive Service.

To create new Hive tables, the Hadoop Distributed File System (HDFS) Streaming API requires a valid user ID but ignores any password. When you create or append to a table, the user ID that you provided in the LIBNAME statement is passed to this streaming method. SAS/ACCESS uses the user ID to place your data on the Hadoop cluster. The user ID must be valid on the Hadoop cluster and needs Write access to the Hadoop /tmp and the Hive warehouse directory. Therefore, the Hadoop system administrator should ensure Write access to the HDFS /tmp and Hive warehouse directories for all user IDs that are permitted to create Hive tables with SAS/ACCESS.

Future Hadoop and Hive JDBC editions will address these security limitations. Even though USER= and PASSWORD= are not yet fully enabled, consider using both USER= and PASSWORD= on LIBNAME and CONNECT statements to prepare for more secure future releases.

Arguments

libref
specifies any SAS name that serves as an alias to associate SAS with a database, schema, server, or group of tables and views.

hadoop
specifies the SAS/ACCESS engine name for the Hadoop interface.

connection-options
provide connection information and control how SAS manages the timing and concurrence of the connection to Hadoop. Here is how these options are defined.

USER=<'>*user-name*<'>
lets you connect to a Hadoop database with a user ID that is different from the default ID. USER= is required for creating tables but is ignored for reading them.

PASSWORD=<'>*password*<'>
specifies the Hadoop password that is associated with your user ID. If it contains spaces or nonalphanumeric characters, you must enclose it in quotation marks. If you do not want to enter your Hadoop password in uncoded text on this

statement, see PROC PWENCODE in the *Base SAS Procedures Guide* for a method to encode it.

Alias: PASS=, PWD=, PW=

SERVER=<'>*server-name*<'>
specifies the Hadoop server name that runs the Hive Server. If the server name contains spaces or nonalphanumeric characters, you must enclose it in quotation marks.

Alias: HOST=

PORT=*port*
specifies the port number that is used to connect to the specified Hive Server.

Default: 10000

SCHEMA=*Hive-schema*
specifies the Hive schema.

Default: SCHEMA=**`default`**

Alias: DATABASE=, DB=

LIBNAME-options
define how SAS processes DBMS objects. The following table describes the LIBNAME options for SAS/ACCESS Interface to Hadoop, with the applicable default values. For more detail about these options, see "LIBNAME Options for Relational Databases" in *SAS/ACCESS for Relational Databases: Reference*.

Table 1.2 *SAS/ACCESS LIBNAME Options for Hadoop*

Option	Default Value
ACCESS=	none
AUTHDOMAIN=	none
BL_PORT= Alias: BULKLOAD_PORT=	8020 *Note:* This option is required only if Hadoop HDFS streaming is running on a port other than the default port 8020.
BULKLOAD=	none
DBCREATE_TABLE_OPTS=	none
DBGEN_NAME=	DBMS
DBPROMPT=	NO
DBSASLABEL=	COMPAT
DIRECT_SQL=	YES
SPOOL=	YES
SQL_FUNCTIONS=	none

Hadoop LIBNAME Statement Examples

This example uses the default Hive port and schema.

```
libname hdp hadoop server= hxpduped;
```

This example explicitly specifies the default Hive port and schema.

```
libname hdp hadoop server= hxpduped port=10000 schema=default;
```

Data Set Options for Hadoop

All SAS/ACCESS data set options in this table are supported for Hadoop. Default values are provided where applicable. For details about this feature, see "Data Set Options for Relational Databases" in *SAS/ACCESS for Relational Databases: Reference*.

Table 1.3 *SAS/ACCESS Data Set Options for Hadoop*

Option	Default Value
DBCONDITION=	none
DBCREATE_TABLE_OPTS=	LIBNAME option setting
DBGEN_NAME=	DBMS
DBLABEL=	NO
DBMASTER=	none
DBPROMPT=	LIBNAME option setting
DBSASLABEL=	COMPAT
DBSASTYPE=	see "Data Types for Hadoop" on page 10
DBTYPE=	see "Data Types for Hadoop" on page 10

SQL Pass-Through Facility Specifics for Hadoop

Key Information

For general information about this feature, see "SQL Procedure Interactions with SAS/ACCESS" in *SAS/ACCESS for Relational Databases: Reference*. Hadoop examples on page 6 are available.

Here are the SQL pass-through facility specifics for the Hadoop interface.

- The *dbms-name* is **HADOOP**.
- The CONNECT statement is required.
- PROC SQL supports multiple connections to Hadoop. If you use multiple simultaneous connections, you must use the *alias* argument to identify the different connections. If you do not specify an alias, the default **HADOOP** alias is used.
- The *database-connection-arguments* for the CONNECT statement are identical to its LIBNAME connection options on page 3.

Examples

This example uses the default Hive port and schema.

```
proc sql;
   connect to hadoop (user="myuser" pw="mypwd" server=hxpduped);
```

This example explicitly specifies the default Hive port and schema.

```
proc sql;
   connect to hadoop (user="myuser" pw="mypwd"
      server=hxpduped port=10000 schema=default);
```

Passing SAS Functions to Hadoop

SAS/ACCESS Interface to Hadoop passes the following SAS functions to Hadoop for processing. Where the Hadoop function name differs from the SAS function name, the Hadoop name appears in parentheses. For more information, see "Passing Functions to the DBMS Using PROC SQL" in Chapter 5 of *SAS/ACCESS for Relational Databases: Reference*.

ABS	MAX
ARCOS (ACOS)	MIN
ARSIN (ASIN)	MINUTE
ATAN	MONTH
AVG	SECOND
CEIL	SIN
COALESCE	SQRT
COS	STD (STDDEV_SAMP)
COUNT	STRIP (TRIM)
DAY	SUBSTR
EXP	SUM
FLOOR	TAN
HOUR	TRANSTRN (REGEXP_REPLACE)
INDEX (LOCATE)	UPCASE (UPPER)
LOG (LN)	VAR (VAR_SAMP)
LOG10	YEAR
LOWCASE (LOWER)	

SQL_FUNCTIONS=ALL allows for SAS functions that have slightly different behavior from corresponding Hadoop functions. Due to incompatibility in date and time functions

between Hadoop and SAS, Hadoop might not process them identically. Check your results to determine whether these functions are working as expected.

- COMPRESS (REGEXP_REPLACE)
- DATE (TO_DATE(UNIX_TIMESTAMP))
- DATEPART (TO_DATE(UNIX_TIMESTAMP))
- DATETIME (FROM_UNIXTIME(UNIX_TIMESTAMP))
- LENGTH
- REPEAT
- TODAY (TO_DATE(UNIX_TIMESTAMP)
- TRIMN (RTRIM)

Passing Joins to Hadoop

The SAS/ACCESS engine does not pass LIBNAME-referenced cross-schema joins to Hadoop. To pass a multiple-libref join to Hadoop, the schemas for each LIBNAME statement must be identical. You can use the SQL pass-through facility on page 5 to pass a cross-schema join to Hadoop.

For more information about when and how SAS/ACCESS passes joins to the DBMS, see “Passing Joins to the DBMS” in Chapter 5 of *SAS/ACCESS for Relational Databases: Reference*.

Bulk Loading for Hadoop

Overview

SAS/ACCESS Interface to Hadoop has no differentiation between a bulk load and a standard load process. Although BULKLOAD=YES syntax is supported, it does not change the underlying load process.

Here is how bulk loading works in the Hadoop interface.

1. SAS issues a CREATE TABLE command to the Hive server. The command contains all table metadata (column definitions) and the table properties that are specific to SAS that refine Hive metadata to handle maximum string lengths and date/time formats.
2. SAS initiates an HDFS streaming connection to upload table data to the HDFS /tmp directory. The resulting file is a UTF-8 text file that uses CTRL-A (‘\001’) as a field delimiter and a newline (‘\n’) character as a record separator. These are the defaults that SAS uses for Hive.
3. SAS issues a LOAD DATA command to the Hive server to move the data file from the HDFS /tmp directory to the appropriate Hive warehouse location. The data file is now considered to be an internal table.

Hives considers a table to be a collection of files in a directory that bears the table name. The CREATE TABLE command creates this directory either directly in the Hive

warehouse or in a subdirectory, if a nondefault schema is used. The LOAD DATA command moves the table to the correct location.

When PROC APPEND is used, the CREATE TABLE command is skipped because the base table already exists. The data to be appended is streamed to the HDFS /tmp directory, and the LOAD DATA command moves the file as an additional file into the Hive warehouse directory. After that, multiple files exist for a given table, using unique names. During read, Hive concatenates all files belonging to a table to render the table as a whole.

Note: These functions are not currently supported.

- loading Hbase tables
- loading external HDFS tables
- loading tables that use nontext format and custom serdes
- loading Hive partitioned tables

External Tables

External tables are stored in HDFS but outside of the directories set aside for Hive. Reading an external table that was created manually is transparent to SAS because Hive handles the physical location of the file. However, Hadoop bulk loading always creates internal tables within the Hive warehouse directory structure. Follow these manual steps to create an external table.

1. Issue the appropriate CREATE TABLE command, using either the Hive command interface or by using explicit SQL in SAS.
2. Create the table file in HDFS. One way to do this is by using the Hadoop Access Method on page 31.
3. Issue the LOAD DATA command by using the LOCAL keyword, which indicates that the table file is local to HDFS or external to Hive.

File Formats

SAS can read Hive tables that are formatted in any way that is compatible with Hive. However, Hive tables that SAS creates through bulk loading always use these characteristics:

- They are stored as text files. Numeric data is expressed as strings of formatted ASCII characters.
- The field delimiter is a CTRL-A (\001), which is the Hive default.
- The record delimiter is a newline (\n) character, which also the Hive default.
- Null (empty or missing) values for strings are represented as \N (backslash with a capital N)—not to be confused with \n, which is only a single newline character.
- The Hive default serde (serializer/deserialier) for text files is used (LazySimpleSerDe).

File Locations

Internal Hive tables are stored in the Hive warehouse. By default, this is in the HDFS file system under /user/hive/warehouse. Creating a table creates a directory with the

table name, and all files in that directory are considered to be part of the table. HDFS also splits and replicates the data among the Hadoop member nodes, but that is transparent to operations and commands that are described in this document. To manually check these files, log on to the Hadoop namenode and use the **hadoop fs** command to navigate the HDFS file system. Here are some examples.

Command	Description
hadoop fs -lsr /user/hive/warehouse	Lists all Hive files
hadoop fs -ls /user/hive/warehouse/myschema.db	Lists all Hive table directories in **myschema**schema
hadoop fs -ls /user/hive/warehouse/mytable	Lists all Hive table files that belong to the **mytable** table in the default schema
hadoop fs -cat /user/hive/warehouse/mytable/<filename>	Displays the actual contents of the file
hadoop fs -ls /user/hive/warehouse	Lists all Hive table directories in the default schema

Examples

This example creates and loads the FLIGHTS98 Hadoop table from the SASFLT.FLT98 SAS data set.

```
libname sasflt 'SAS-library';
libname hdp_air hadoop user=louis pwd=louispwd server='hdpcluster' schema=statsdiv;

proc sql;
create table hdp_air.flights98
        as select * from sasflt.flt98;
quit;
```

This example also creates and loads the FLIGHTS98 Hadoop table from the SASFLT.FLT98 SAS data set.

```
data hdp_air.allflights;
set sasflt.allflights;
run;
```

In this example, the SASFLT.FLT98 SAS data set is appended to the existing FLIGHTS98 Hadoop table.

```
proc append base=hdp_air.allflights
data=sasflt.flt98;
run;
```

Naming Conventions for Hive

For general information about this feature, see "SAS Names and Support for DBMS Names" in Chapter 2 of *SAS/ACCESS for Relational Databases: Reference*.

SAS and Hadoop objects include tables, views, table references, columns, and indexes. They follow these naming conventions.

- A SAS name must be from 1 to 32 characters long. When Hive column names and table names are 32 characters or less, SAS handles them seamlessly. When SAS reads Hive column names that are longer than 32 characters, a generated SAS variable name is truncated to 32 characters. Hive table names should be 32 characters or less because SAS cannot truncate a table reference. If you already have a table name that is greater than 32 characters, create a Hive table view or use the explicit SQL feature of PROC SQL to access the table.
- If truncating would result in identical names, SAS generates a unique name.
- Even when it is enclosed in single or double quotation marks, a Hive name does not retain case sensitivity. Hive table and column names can contain uppercase letters A through Z (A-Z), lowercase letters A through Z (a-z), numbers from 0 to 9, and the underscore (_). Hive converts uppercase characters to lowercase. Therefore, such SAS table references as MYTAB and mytab are synonyms—referring to the same table.
- A name can begin with a letter or an underscore but not a number.
- A name cannot be a Hadoop reserved word. If a name generates a Hadoop error, try to append a number or underscore in an appropriate place. For example, if *shipped* results in an error, try *shipped1* or *ship_date*.

Although the PRESERVE_COL_NAMES= and PRESERVE_TAB_NAMES= options are supported for SAS/ACCESS Interface to Hadoop, you should not need to use them.

Data Types for Hadoop

Overview

Hive is a data warehouse that supplies metadata about data that is stored in Hadoop files. Hive includes a data dictionary and an accompanying SQL-like interface called HiveQL or Hive SQL. HiveQL implements data definition language (DDL) and data manipulation language (DML) statements similar to many relational database management systems. Hive tables are defined with a CREATE TABLE statement, so every column in a table has a name and a data type. The data type indicates the format in which the data is stored. Hive exposes data that is stored in HDFS and other file systems through the data types that are described in this section. These are accepted column types in a Hive CREATE TABLE statement. This section includes information about Hive data types and data conversion between Hive and SAS.

For more information about Hive, Hadoop, and data types, see these online documents at **`https://cwiki.apache.org/confluence/display/Hive`**.

- *Hive Getting Started*

- *Hive Language Manual*
- *Hive Language Manual UDF*
- *Hive Tutorial*

Simple Hive Data Types

STRING
specifies variable-length character data. The maximum length is 2 gigabytes.

TINYINT
specifies a signed one-byte integer.

SMALLINT
specifies a signed two-byte integer.

INT
specifies a signed four-byte integer.

BIGINT
specifies a signed eight-byte integer.

FLOAT
specifies a signed four-byte, floating-point number.

DOUBLE
specifies a signed eight-byte, floating-point number.

SAS/ACCESS does not yet support the BOOLEAN and BINARY data types that are available in Hive 8.

Complex Hive Data Types

ARRAY
specifies an array of integers (indexable lists).

MAP
specifies an associated array from strings to string (key-value pairs).

STRUCT
specifies how to access elements within the type—namely, by using the dot (.) notation.

Hive Date, Time, and Timestamp Data

Hive does not define data types for dates, times, or timestamps. By convention, these are typically stored in ANSI format in Hive STRING columns. For example, the last day of this millennium is stored as the string *'2999-12-31'*. SAS/ACCESS assumes ANSI format for dates, times, and timestamps that are stored in Hadoop. Therefore, if you use the DBSASTYPE option to indicate that a Hive STRING column contains a date, SAS/ACCESS expects an ANSI date that it converts to SAS date format. For output, SAS DATE, TIME, and DATETIME formats are converted to ANSI format and are stored in Hive STRING columns.

SAS/ACCESS does not currently support conversion of Hadoop binary UNIX timestamps to SAS DATETIME format.

SAS Data Types

SAS has two fundamental data types, character and numeric. SAS character variables (columns) are of a fixed length with a maximum of 32,767 characters. SAS numeric variables are signed eight-byte, floating-point numbers. When SAS numerics are used in conjunction with SAS formats, they can represent a number of data types, including DATE, TIME, and DATETIME. For more detailed information about SAS data types, see *SAS Language Reference: Concepts*.

Data Conversion from Hive to SAS

This table shows the default SAS formats that are assigned to SAS variables that are created when SAS/ACCESS reads Hive table columns.

Table 1.4 *Hive to SAS: Default SAS Formats for Hive Data Types*

Hive Data Type	SAS Data Type	Default SAS Format
STRING	character	$32767.
TINYINT	numeric	4.
SMALLINT	numeric	6.
INT	numeric	11.
BIGINT	numeric	20.
FLOAT	numeric	none
DOUBLE	numeric	none

Issues When Converting Data from Hive to SAS

Below are some potential conversion issues.

- Hive STRING columns that contain ANSI date, time, or timestamp values do not automatically convert respectively to SAS DATE, TIME, or DATETIME types.
- STRING: Depending on the length of Hadoop STRING data, the SAS character $32767. format might be unnecessarily large for short STRING columns or can truncate Hadoop STRING columns that contain more than 32767 characters.
- BIGINT: Converting Hadoop BIGINT to a SAS numeric can result in loss of precision because the internal SAS eight-byte, floating-point format accurately preserves only 15 digits of precision. A BIGINT preserves up to 19.

Work-arounds are based on how you access data.

- explicitly using pass-through SQL, see “Address Issues When Converting Data from Hive to SAS for Pass-Through SQL” on page 18.
- using the LIBNAME statement, see “Address Issues When Converting Data from Hive to SAS with Table Properties” on page 15.

SAS Table Properties for Hive and Hadoop

While HiveQL supplies critical metadata for Hadoop files, in some cases more metadata is beneficial. Fortunately, HiveQL CREATE TABLE and ALTER TABLE statements provide an extensible feature called table properties. (For more information, see the Hive Language Manual at **https://cwiki.apache.org/confluence/display/Hive**.) SAS/ACCESS uses table properties to describe and interpret the contents of Hive STRING columns. For example, if you create a new Hive table with SAS/ACCESS and a SAS variable (column) has an associated SAS DATETIME format, SAS/ACCESS creates a DATETIME table property for the Hive column. Here is an example.

```
libname hdp hadoop server=dbihadoop user=hadoop_usr pwd= hadoop_pwd;
data hdp.datetime_tableproperty_sample;
format dt_stamp datetime25.6;
dt_stamp=datetime();
run;
```

This code creates a new Hive table, DATETIME_TABLEPROPERTY_SAMPLE, by generating this HiveQL:

```
CREATE TABLE `DATETIME_TABLEPROPERTY_SAMPLE` (`dt_stamp` STRING)
   ROW FORMAT DELIMITED FIELDS TERMINATED BY '\001'
   STORED AS TEXTFILE TBLPROPERTIES ('SASFMT:dt_stamp'='DATETIME(25.6)')
```

SAS stores **dt_stamp** as a Hive ANSI STRING, as in this example:

```
2012-02-23 09:51:37.218
```

Based on the SAS DATETIME25.6 format, SAS/ACCESS also generates the Hive table property that describes STRING column **dt_stamp** as DATETIME(25.6).

When SAS/ACCESS reads this Hive table, the SASFMT table property indicates STRING column **dt_stamp** contains an ANSI timestamp. SAS/ACCESS automatically converts and formats it a SAS DATETIME25.6 variable, as in this example:

```
data;
set hdp.datetime_tableproperty_sample;
put dt_stamp=;
run;
```

```
dt_stamp=23FEB2012:09:51:37.218000
NOTE: There were 1 observations read from the data set
HDP.DATETIME_TABLEPROPERTY_SAMPLE.
```

When SAS/ACCESS creates a new Hive table, it generates table properties for SAS variables with character, date, datetime, and time formats—all of which produce Hive STRING columns. See the generated table properties in "Data Conversion from SAS to Hive" on page 13

Data Conversion from SAS to Hive

This table shows the Hive data types and table properties that are assigned when SAS/ACCESS creates a Hive table.

SAS sets table properties only when creating a new Hive table. It does not create or alter table properties when appending to an existing Hive table.

Table 1.5 Hive Data Types and Table Properties That Are Created for SAS Data Type and Format Combinations

SAS Data Type	SAS Format	Hive Data Type	Hive Table Property
character	$n.	STRING	CHAR(n)
numeric	DATETIMEw.p	STRING	DATETIME(w.p)
numeric	DATEw.	STRING	DATE(w.0)
numeric	TIMEw.p.	STRING	TIME(w.p)
numeric	1. to 4.	SMALLINT	none
numeric	5. to 9.	INT	none
numeric	other numeric formats	DOUBLE	none

Leverage Table Properties for Existing Hive Tables

SAS/ACCESS generates SAS table properties only when creating a new Hive table. Many or perhaps all of your Hive tables are created by other means. For example, your Hadoop administrator might create Hive table definitions by submitting DDL scripts to the Hive CLI. SAS and SAS users can benefit by adding SAS table properties to existing Hive table definitions. Here is an example, where a Hive table has already been defined.

```
CREATE EXTERNAL TABLE weblogs (extract_date STRING,
   extract_type INT, webdata STRING) ROW FORMAT DELIMITED FIELDS
   TERMINATED BY ',' STORED AS TEXTFILE LOCATION '/user/hadoop/web_data'
```

Based on this table definition, here is how SAS interprets the columns.

```
libname hdp sasiohdp server=dbihadoop user=hadoop_usr pwd= hadoop_pwd;
data sheetmetal_sales; set hdp.weblogs(obs=1);
put extract_date= extract_type=;
put webdata=;
run;
```

```
   extract_date=2012-02-21 extract_type=1
   webdata=http://www.sas.com/industry/oilgas
   NOTE: There were 1 observations read from the data set HDP.WEBLOGS.
```

```
proc contents data=hdp.weblogs; run;
```

```
          Alphabetic List of Variables and Attributes
#  Variable       Type   Len     Format    Informat    Label
1  extract_date   Char   32767   $32767.   $32767.     extract_date
2  extract_type   Num        8   11.       11.         extract_type
3  webdata        Char   32767   $32767.   $32767.     webdata
```

Notice that Hive describes the **extract_date** column to SAS as a 32767 length STRING. It also describes the **webdata** column as a 32767 length STRING. So SAS/ACCESS types both of these columns as character data and uses $32767. to format

them. The result is an overly wide SAS data set with an observation (row) width of 64 kilobytes that also does not format **extract_date** to a SAS DATE.

The example below assumes that the length of the **webdata** STRING in Hive never exceeds 1000 characters. A Hadoop user ID with the appropriate authority can issue Hive ALTER TABLE statements to add SAS table properties to the Hive table definition.

```
ALTER TABLE weblogs SET TBLPROPERTIES ('SASFMT:extract_date'='DATE(9.0)')
ALTER TABLE weblogs SET TBLPROPERTIES ('SASFMT:webdata'='CHAR(1000)')
```

SAS/ACCESS honors the added properties, and here is the result.

```
libname hdp sasiohdp server=dbihadoop user=hadoop_usr pwd= hadoop_pwd;
data sheetmetal_sales; set hdp.weblogs(obs=1);
put extract_date= extract_type=;
put webdata=;
run;
```

```
extract_date=21FEB2012 extract_type=1
webdata=http://www.sas.com/industry/oilgas
NOTE: There were 1 observations read from the data set HDP.WEBLOGS.
```

```
proc contents data=hdp.weblogs; run;
```

```
          Alphabetic List of Variables and Attributes
#  Variable       Type    Len     Format    Informat    Label
1  extract_date   Num       8    DATE9.    DATE9.      extract_date
2  extract_type   Num       8    11.       11.         extract_type
3  webdata        Char   1000    $10000    $1000.      webdata
```

The resulting SAS data set that is created from the Hive table has a much smaller observation width, which helps SAS save disk space and reduce CPU consumption. It also automatically converts and formats **extract_date** to SAS standard DATE9. format.

Adding SAS properties to existing Hive tables does not impact table use by software that is not SAS. You can also issue ALTER TABLE and other DDL statements using SAS/ACCESS explicit SQL (see "SQL Pass-Through Facility Specifics for Hadoop" on page 5). Issuing such DDL as an ALTER TABLE statement can be restricted to only the Hadoop administrator.

Address Issues When Converting Data from Hive to SAS with Table Properties

Some issues currently exist when reading Hadoop data into SAS. (See "Issues When Converting Data from Hive to SAS" on page 12.) For example, Hive STRING columns default to the $32767. SAS character format without a defined SASFMT table property or a SAS override option such as DBSASTYPE=.

Here is how you can address specific conversion issues.

STRING issues
: To automatically convert Hive STRING columns that contain ANSI date, timestamp, or time values to suitable SAS formats, you can use the following ALTER TABLE statements. In the statements are these sample Hive columns: **d** contains ANSI date, **ts** contains ANSI timestamp, and **t** contains ANSI time.

```
ALTER TABLE sample_table SET TBLPROPERTIES ('SASFMT:d'='DATE(9.0)')
ALTER TABLE sample_table SET TBLPROPERTIES ('SASFMT:ts'='DATETIME(25.6)')
ALTER TABLE sample_table SET TBLPROPERTIES ('SASFMT:t'='TIME(15.6)')
```

Instead, you could use these statements to create SAS character variables of optimal length that contain the identical ANSI representation as those that are stored in Hive:

```
ALTER TABLE sample_table SET TBLPROPERTIES ('SASFMT:d'='CHAR(9)')
ALTER TABLE sample_table SET TBLPROPERTIES ('SASFMT:ts'='CHAR(25)')
ALTER TABLE sample_table SET TBLPROPERTIES ('SASFMT:t'='CHAR(15)')
```

You can use the following statement for other Hive STRING columns where the maximum length is less than 32767. Here, the **`string_col`** column has a maximum length of 100.

```
ALTER TABLE sample_table SET TBLPROPERTIES ('SASFMT:string_col'='CHAR(100)')
```

However, if you anticipate that the **`string_col`** column in Hive might grow to a maximum length of 200 in the future, you could instead use this statement to set the table property.

```
ALTER TABLE sample_table SET TBLPROPERTIES ('SASFMT:string_col'='CHAR(200)')
```

Hive STRING columns longer than 32767 characters are truncated when they are read into SAS. Here is how the warning for this data loss is flagged in the SAS log:

```
WARNING: Column 'string_col' was truncated 1 times.
Observation (row) number 2 was the first observation truncated.
```

BIGINT issues

Converting a Hadoop BIGINT column to a SAS numeric column can cause a loss of precision. A SAS numeric column can accurately preserve only 15 digits of precision, whereas a BIGINT column can preserve up to 19 significant digits of precision. You can address this issue by applying a CHAR(19) table property format. SAS then automatically reads a Hive BIGINT column into a SAS character string with $19. format, which preserves all BIGINT digits in character format. Here is an example, using the **`bgint`** BIGINT column.

```
ALTER TABLE sample_table SET TBLPROPERTIES ('SASFMT:bgint'='CHAR(19)')
```

Keep this important consideration in mind, however: For Hive tables that SAS/ACCESS creates, you might not need to issue ALTER TABLE statements. See "Data Conversion from SAS to Hive" on page 13 for table properties that SAS/ACCESS automatically generates when it creates a Hive table.

CAUTION:

Do not create multiple table properties for a single Hive column. Unpredictable data conversion can result.

Alternatives to Table Properties for Issues with Data Conversion from Hive to SAS

For various reasons, it might be impractical or undesirable to issue ALTER TABLE statements to create SAS table properties. In such cases, you can instead use these data set options.

DBSASTYPE=

Use DBSASTYPE= in your SAS code to cause data conversion from Hive to SAS that is identical to automatic conversion with table properties. The pairs below are SAS DATA steps with identical behavior. The first of each pair uses a SASFMT table property, the second one uses no table property, and DBSASTYPE= is added to

achieve the same functionality. (For more information about this data set option, see *SAS/ACCESS for Relational Databases: Reference*.) Here is the SAS LIBNAME statement for all of these SAS DATA steps.

```
libname hdp sasiohdp server=dbihadoop user=hadoop_usr pwd= hadoop_pwd;

ALTER TABLE sample_table SET TBLPROPERTIES ('SASFMT:d'='DATE(9.0)')
data work.local_sample; set hdp.sample_table( keep=d ); run;

[---no table property for column 'd'---]
data work.local_sample;
set hdp.sample_table( keep=d dbsastype=(d='DATE(9.0)') ); run;

ALTER TABLE sample_table SET TBLPROPERTIES ('SASFMT:ts'='DATETIME(25.6)')
data work.local_sample; set hdp.sample_table( keep=ts ); run;

[---no table property for column 'ts'---]
data work.local_sample;
set hdp.sample_table( keep=ts dbsastype=(ts='DATETIME(25.6)') ); run;


ALTER TABLE sample_table SET TBLPROPERTIES ('SASFMT:t'='TIME(15.6)')
data work.local_sample; set hdp.sample_table( keep=t ); run;

[---no table property for column 't'---]
data work.local_sample;
set hdp.sample_table( keep=t dbsastype=(t='TIME(15.6)') ); run;


ALTER TABLE sample_table SET TBLPROPERTIES ('SASFMT:string_col'='CHAR(200)')
data work.local_sample; set hdp.sample_table( keep=string_col ); run;

[---no table property for column 'string_col'---]
data work.local_sample;
set hdp.sample_table( keep=string_col dbsastype=(string_col='CHAR(200)') );
run;
```

COMPRESS=YES

If you can use neither table properties nor DBSASTYPE= to reduce the default 32767 SAS character length for Hive STRING columns, consider using this data set option. While SAS character variables still use the $32767. format, using SAS data set compression that you can realize significant savings in terms of disk space and CPU consumption. Here is an example.

```
data work.local_sample( COMPRESS=YES ); set hdp.sample_table; run;
```

Note that you apply COMPRESS=YES to the SAS data set reference, not to the Hadoop table reference.

You might also consider using COMPRESS=YES even if you reduce default SAS character 32767 lengths with table properties or DBSASTYPE= but when many strings in the Hadoop data are much smaller than the reduced SAS character variable length. For example, you could use COMPRESS=YES if you had the table property shown below, but most values of **`address_1`** are less than 100 characters.

```
ALTER TABLE sample_table SET TBLPROPERTIES ('SASFMT:address_1'='CHAR(1000)')
```

For more information about this data set option, see *SAS Data Set Options: Reference*.

Address Issues When Converting Data from Hive to SAS for Pass-Through SQL

Neither table properties nor DBSASTYPE= address Hive-to-SAS data conversion issues if you use pass-through SQL to read Hive data. For pass-through SQL, you might need to explicitly convert and format each Hive column as you want it to be represented in SAS. This is illustrated by using SAS to create a table, with SAS table properties being generated for all but the BIGINT column. Here is the table that SAS creates.

```
libname hdp sasiohdp server=dbihadoop user=hadoop_usr pwd=hadoop_pwd;
data hdp.passthrough_ex( dbtype=(bgint="BIGINT") );
bgint='1234567890123456789';
format ts datetime25.6; ts=datetime();
format d date9.; d=today();
format t time10.; t=time();
format string_col $20.; string_col='hello';
run;
```

SAS issues this HiveQL when creating the table.

```
CREATE TABLE `PASSTHROUGH_EX` (`bgint` BIGINT,`ts` STRING,
   `d` STRING,`t` STRING,`string_col` STRING) ROW FORMAT DELIMITED FIELDS
   TERMINATED BY '\001' STORED AS TEXTFILE TBLPROPERTIES
('SASFMT:ts'='DATETIME(25.6)','SASFMT:d'='DATE(9.0)',
   'SASFMT:t'='TIME(10.0)','SASFMT:string_col'='CHAR(20)')
```

Next, an ALTER TABLE statement is issued to add a table property for **BIGINT column bgint**.

```
ALTER TABLE passthrough_ex SET TBLPROPERTIES ('SASFMT:bgint'='CHAR(19)')
```

A LIBNAME-based table that is read to SAS honors the table properties.

```
data work.local; set hdp.passthrough_ex;
run;
data _null_; set work.local;
put bgint=; put ts=; put d=; put t=; put string_col=;
run;
```

```
   bgint=1234567890123456789
   ts=25FEB2012:02:00:55.141000
   d=25FEB2012
   t=2:00:55
   string_col=hello
```

This pass-through SQL step converts and formats each column identically to the LIBNAME-based step that applied the table properties.

```
proc sql; connect to sasiohdp(server=dbihadoop user=hadoop_usr pwd=hadoop_pwd);
create table work.local as select
bgint length 19 format $19. informat $19.,
input(ts, IS8601DT26.) as ts format datetime25.6 informat datetime25.6,
input(d, yymmdd10.) as d format date9. informat date9.,
input(t, IS8601TM15.) as t format time15.6 informat time15.6,
string_col length  20 format $20. informat $20.
from connection to sasiohdp( select cast(bgint as STRING)
   as bgint,ts,d,t,string_col from passthrough_ex );
quit;
data _null_; set work.local;
```

```
put bgint=; put ts=; put d=; put t=; put string_col=;
run;
```

```
bgint=1234567890123456789
ts=25FEB2012:02:00:55.141000
d=25FEB2012
t=2:00:55.141000
string_col=hello
```

Hadoop Null Values

Hadoop has a special value called NULL. A Hadoop NULL value means an absence of information and is analogous to a SAS missing value. When SAS/ACCESS reads a Hadoop NULL value, it interprets it as a SAS missing value. For more information about how SAS handles NULL values, see "Potential Result Set Differences When Processing Null Data" in *SAS/ACCESS for Relational Databases: Reference*.

Sample Code for Hadoop

Code Snippets

The code snippets in this section resemble those for most other SAS/ACCESS interfaces.

This snippet shows a list of available Hive tables.

```
proc datasets lib=hdp; quit;
```

Here is the metadata for the mytab Hive table.

```
proc contents data=hdp.mytab; quit;
```

This snippet extracts mytab data into SAS.

```
data work.a;
set hdp.mytab;
run;
```

This extracts a subset of the mytab rows and columns into SAS. Subsetting the rows (with a WHERE statement, for example) can help avoid extracting too much data into SAS.

```
data work.a;
set hdp.mytab (keep=col1 col2);
where col2=10;
run;
```

Use DBSASTYPE= to Load Hadoop Data into SAS

This example uses the DBSASTYPE= option (see *SAS/ACCESS for Relational Databases: Reference*) to load Hadoop textual dates, timestamps, and times into the corresponding SAS DATE, DATETIME, and TIME formats. The first step reads in a SAS character string to display the data and make clear what occurs in successive steps.

```
data; set hdp.testHiveDate; put dt; run;
```

```
2011-10-17
2009-07-30 12:58:59
11:30:01
```

```
data; set hdp.testHiveDate(dbsastype=(dt='date')); put dt; run;
```

```
17OCT2011
30JUL2009
.
```

```
data; set hdp.testHiveDate(dbsastype=(dt='datetime')); put dt; run;
```

```
17OCT2011:00:00:00
30JUL2009:12:58:59
.
```

```
data; set hdp.testHiveDate(dbsastype=(dt='time')); put dt; run;
```

```
.
12:58:59
11:30:01
```

This code uses SAS SQL to access a Hadoop table.

```
proc sql;
create table work.a as select * from hdp.newtab;
quit;
```

SAS data is then loaded into Hadoop.

```
data hdp.newtab2;
set work.a;
run;
```

Use implicit pass-through SQL to extract only 10 rows from the newtab table and load the work SAS data set with the results.

```
proc sql;
connect to hadoop (server=hxpduped);
create table work.a as
   select * from connection to hadoop (select * from newtab limit 10);
```

Chapter 2

HADOOP Procedure

Overview: HADOOP Procedure

PROC HADOOP enables SAS to run Apache Hadoop code against Hadoop data. Apache Hadoop is an open-source technology, written in Java, that provides data storage and distributed processing of large amounts of data.

PROC HADOOP interfaces with the Hadoop JobTracker. This is the service within Hadoop that controls tasks to specific nodes in the cluster. PROC HADOOP enables you to submit the following:

- HDFS commands
- MapReduce programs
- Pig language code

Concepts: HADOOP Procedure

Using PROC HADOOP

PROC HADOOP enables you to submit HDFS commands, MapReduce programs, and Pig language code against Hadoop data. To connect to the Hadoop server, a Hadoop configuration file is required that specifies the name and JobTracker addresses for the specific server. **fs.default.name** is the URI (protocol specifier, host name, and port) that describes the NameNode for the cluster. Here is an example of a Hadoop configuration file:

```
<configuration>
   <property>
      <name>fs.default.name</name>
      <value>hdfs://xxx.us.company.com:8020</value>
   </property>
   <property>
      <name>mapred.job.tracker</name>
      <value>xxx.us.company.com:8021</value>
   </property>
</configuration>
```

The PROC HADOOP statement supports several options that control access to the Hadoop server. For example, you can identify the Hadoop configuration file and specify the user ID and password on the Hadoop server. You can specify the server options on all PROC HADOOP statements. For the list of server options, see "Hadoop Server Options" on page 23.

Submitting Hadoop Distributed File System Commands

The Hadoop Distributed File System (HDFS) is a distributed, scalable, and portable file system for the Hadoop framework. HDFS is designed to hold large amounts of data and provide access to the data to many clients that are distributed across a network.

The PROC HADOOP HDFS statement submits HDFS commands to the Hadoop server that are like the Hadoop shell commands that interact with HDFS and manipulate files. For the list of HDFS commands, see "HDFS Statement" on page 24.

Submitting MapReduce Programs

MapReduce is a parallel processing framework that enables developers to write programs to process vast amounts of data. There are two types of key functions in the MapReduce framework: the Map function that separates the data to be processed into independent chunks and the Reduce function that performs analysis on that data.

The PROC HADOOP MAPREDUCE statement submits a MapReduce program into a Hadoop cluster. See "MAPREDUCE Statement" on page 25.

Submitting Pig Language Code

The Apache Pig language is a high-level programming language that creates MapReduce programs that are used with Hadoop.

The PROC HADOOP PIG statement submits Pig language code into a Hadoop cluster. See "PIG Statement" on page 26.

Syntax: HADOOP Procedure

Restriction: PROC HADOOP is supported in the Microsoft Windows and Linux 64-bit operating environments only.

Requirement: JRE 1.6

PROC HADOOP <*hadoop-server-options*>;
HDFS <*hadoop-server-options*> <*hdfs-command-options*>;
MAPREDUCE <*hadoop-server-options*> <*mapreduce-options*>;
PIG <*hadoop-server-options*> <*pig-code-options*>;

Statement	Task	Example
"PROC HADOOP Statement"	Control access to the Hadoop server	Ex. 1, Ex. 2, Ex. 3
"HDFS Statement"	Submit HDFS (Hadoop Distributed File System) commands	Ex. 1
"MAPREDUCE Statement"	Submit MapReduce programs into a Hadoop cluster	Ex. 2
"PIG Statement"	Submit Pig language code into a Hadoop cluster	Ex. 3

PROC HADOOP Statement

Controls access to the Hadoop server.

Syntax

PROC HADOOP <*hadoop-server-options*>;

Hadoop Server Options

These options control access to the Hadoop server and can be specified on all HADOOP procedure statements.

AUTHDOMAIN=*auth-domain*
specifies the name of an authentication domain metadata object in order to connect to the Hadoop server. The authentication domain references credentials (user ID and password) without your having to explicitly specify the credentials. The *auth-domain* name is case sensitive, and it must be enclosed in double quotation marks.

An administrator creates authentication domain definitions while creating a user definition with the User Manager in SAS Management Console. The authentication domain is associated with one or more login metadata objects that provide access to the Hadoop server. The metadata objects are resolved by SAS calling the SAS Metadata Server and returning the authentication credentials.

Requirement: The authentication domain and the associated login definition must be stored in a metadata repository, and the metadata server must be running to resolve the metadata object specification.

Interaction: If you specify AUTHDOMAIN=, do not specify USERNAME= and PASSWORD=.

See: For more information about creating and using authentication domains, see the discussion on credential management in the *SAS Intelligence Platform: Security Administration Guide*.

OPTIONS=*fileref* | '*external-file*'
identifies the Hadoop configuration file to use for the associated PROC statement. The file must be an XML document.

fileref
specifies the SAS fileref that is assigned to the Hadoop configuration file. To assign a fileref, use the FILENAME statement.

'*external-file*'
is the physical location of the XML document. Include the complete pathname and the filename. Enclose the physical name in single or double quotation marks. The maximum length is 200 characters.

PASSWORD=*password*
is the password for the user ID on the Hadoop server. The user ID and password are added to the set of options that are identified by OPTIONS=.

Requirement: To specify PASSWORD=, you must also specify USERNAME=.

USERNAME='*ID*'
is an authorized user ID on the Hadoop server. The user ID and password are added to the set of options that are identified by OPTIONS=.

VERBOSE
enables additional messages that are displayed on the SAS log. VERBOSE is a good error diagnostic tool. If you receive an error message when you invoke SAS, you can use this option to see whether you have an error in your system option specifications.

HDFS Statement

Submits HDFS (Hadoop Distributed File System) commands.

Restriction: The HDFS statement supports only one operation per invocation.

Example: "Example 1: Submitting HDFS Commands" on page 27

Syntax

HDFS <*hadoop-server-options*> <*hdfs-command-options*>;

HDFS Command Options

These options support commands that interact with the HDFS. Include only one operation per HDFS statement.

COPYFROMLOCAL='*local-file*'
copies the specified local file to an HDFS path output location. Specify the complete pathname and filename.

Requirement: Use the OUT= option to specify the HDFS path output location.

COPYTOLOCAL='*HDFS-file*'
copies the specified HDFS file to a local file output location. Specify the complete HDFS directory and filename.

Requirement: Use the OUT= option to specify the local file output location.

DELETE='*HDFS-file*'
deletes the specified HDFS file. Specify the complete HDFS directory and filename.

DELETESOURCE
deletes the input source file after a copy command.

Restriction: Use DELETESOURCE with the COPYFROMLOCAL= or COPYTOLOCAL= options.

MKDIR='*HDFS-path*'
creates the specified HDFS path. Specify the complete HDFS directory.

OUT='*output-location*'
specifies the output location for an HDFS operation. When copying a local file to HDFS, specify the HDFS path. When copying an HDFS file to a local file, specify the external file for your machine. When renaming an HDFS file, specify the new HDFS path and filename.

OVERWRITE
overwrites the output file after a copy command.

Restriction: Use OVERWRITE with the COPYFROMLOCAL= or COPYTOLOCAL= options.

RENAME='*HDFS-file*'
renames the specified HDFS file. Specify the complete HDFS directory and filename.

Requirement: Use the OUT= option to specify the new HDFS path and filename.

MAPREDUCE Statement

Submits MapReduce programs into a Hadoop cluster.

Example: "Example 2: Submitting a MapReduce Program" on page 28

Syntax

MAPREDUCE <*hadoop-server-options*> <*mapreduce-options*>;

MapReduce Options

COMBINE=*class-name*
specifies the name of the combiner class in dot notation.

DELETERESULTS
deletes the MapReduce results.

GROUPCOMPARE=*class-name*
specifies the name of the grouping comparator (GroupComparator) class in dot notation.

INPUT=*HDFS-path*
specifies the HDFS path to the MapReduce input file.

INPUTFORMAT=*class-name*
specifies the name of the input format class in dot notation.

JAR='*external-file(s)*'
specifies the locations of the JAR files that contain the MapReduce program and named classes. Include the complete pathname and the filename. Enclose each location in single or double quotation marks.

MAP=*class-name*
specifies the name of the map class in dot notation. A map class contains elements that are formed by the combination of a key value and a mapped value.

OUTPUT=*HDFS-path*
specifies a new HDFS path for the MapReduce output.

OUTPUTFORMAT=*class-name*
specifies the name of the output format class in dot notation.

OUTPUTKEY=*class-name*
specifies the name of the output key class in dot notation.

OUTPUTVALUE=*class-name*
is the name of the output value class in dot notation.

PARTITIONER=*class-name*
specifies the name of the partitioner class in dot notation. A partitioner class controls the partitioning of the keys of the intermediate map-outputs.

REDUCE=*class-name*
specifies the name of the reducer class in dot notation. The reduce class reduces a set of intermediate values that share a key to a smaller set of values.

REDUCETASKS=*integer*
specifies the number of reduce tasks.

SORTCOMPARE=*class-name*
specifies the name of the sort comparator class in dot notation.

WORKINGDIR=*HDFS-path*
specifies the name of the HDFS working directory path.

PIG Statement

Submits Pig language code into a Hadoop cluster.

Example: “Example 3: Submitting Pig Language Code” on page 29

Syntax

PIG <*hadoop-server-options*> <*pig-code-options*>;

Pig Code Options

CODE=*fileref* | '*external-file*'
specifies the source that contains the Pig language code to execute.

fileref
is a SAS fileref that is assigned to the source file. To assign a fileref, use the FILENAME statement.

'*external-file*'
is the physical location of the source file. Include the complete pathname and the filename. Enclose the physical name in single or double quotation marks.

PARAMETERS=*fileref* | '*external-file*'
specifies the source that contains parameters to be passed as arguments when the Pig code executes.

fileref
is a SAS fileref that is assigned to the source file. To assign a fileref, use the FILENAME statement.

'*external-file*'
is the physical location of the source file. Include the complete pathname and the filename. Enclose the physical name in single or double quotation marks.

REGISTERJAR='*external-file(s)*'
specifies the locations of the JAR files that contain the Pig scripts to execute. Include the complete pathname and the filename. Enclose each location in single or double quotation marks.

Examples: HADOOP Procedure

Example 1: Submitting HDFS Commands

Details

This PROC HADOOP example submits HDFS commands to a Hadoop server. The statements create a directory, delete a directory, and copy a file from HDFS to a local output location.

Program

```
filename cfg "C:\Users\sasabc\hadoop\sample_config.xml";1

proc hadoop options=cfg username="sasabc" password="sasabc" verbose;2
   hdfs mkdir="/user/sasabc/new_directory";3
   hdfs delete="/user/sasabc/temp2_directory";4
   hdfs copytolocal="/user/sasabc/testdata.txt"
```

```
        out="C:\Users\sasabc\Hadoop\testdata.txt" overwrite;5
run;
```

Program Description

1. The FILENAME statement assigns the file reference CFG to the physical location of a Hadoop configuration file that is named sample_config.xml, which is shown in "Using PROC HADOOP" on page 22.
2. The PROC HADOOP statement controls access to the Hadoop server by referencing the Hadoop configuration file with the OPTIONS= option, identifying the user ID and password on the Hadoop server, and specifying the option VERBOSE, which enables additional messages to the SAS log.
3. The first HDFS statement specifies the MKDIR= option to create an HDFS path.
4. The second HDFS statement specifies the DELETE= option to delete an HDFS path.
5. The third HDFS statement specifies the COPYTOLOCAL= option to specify the HDFS file to copy, the OUT= option to specify the output location on the local machine, and the OVERWRITE option to specify that if the output location exists, write over it.

Example 2: Submitting a MapReduce Program

Details

This PROC HADOOP example submits a MapReduce program to a Hadoop server. The example uses the Hadoop MapReduce application WordCount that reads a text input file, breaks each line into words, counts the words, and then writes the word counts to the output text file.

Program

```
filename cfg 'C:\Users\sasabc\Hadoop\sample_config.xml';1

proc hadoop options=cfg username="sasabc" password="sasabc" verbose;2
   mapreduce input='/user/sasabc/architectdoc.txt'
      output='/user/sasabc/outputtest'
      jar='C:\Users\sasabc\Hadoop\jars\WordCount.jar'
      outputkey="org.apache.hadoop.io.Text"
      outputvalue="org.apache.hadoop.io.IntWritable"
      reduce="org.apache.hadoop.examples.WordCount$IntSumReducer"
      combine="org.apache.hadoop.examples.WordCount$IntSumReducer"
      map="org.apache.hadoop.examples.WordCount$TokenizerMapper";3
run;
```

Program Description

1. The FILENAME statement assigns the file reference CFG to the physical location of a Hadoop configuration file that is named **sample_config.xml**, which is shown in "Using PROC HADOOP" on page 22.
2. The PROC HADOOP statement controls access to the Hadoop server by referencing the Hadoop configuration file with the OPTIONS= option, identifying the user ID

and password on the Hadoop server, and specifying the option VERBOSE, which enables additional messages to the SAS log.

3. The MAPREDUCE statement includes the following options:
   - INPUT= to specify the HDFS path of the input Hadoop file named **`architectdoc.txt`**.
   - OUTPUT= to create the HDFS path for the program output location named **`outputtest`**.
   - JAR= to specify the location of the JAR file that contains the MapReduce program named **`WordCount.jar`**.
   - OUTPUTKEY= to specify the name of the output key class. The **`org.apache.hadoop.io.Text`** class stores text using standard UTF8 encoding and provides methods to compare text.
   - OUTPUTVALUE= to specify the name of the output value class **`org.apache.hadoop.io.IntWritable`**.
   - REDUCE= to specify the name of the reducer class **`org.apache.hadoop.examples.WordCount$IntSumReducer`**.
   - COMBINE= to specify the name of the combiner class **`org.apache.hadoop.examples.WordCount$IntSumReducer`**.
   - MAP= to specify the name of the map class **`org.apache.hadoop.examples.WordCount$TokenizerMapper`**.

Example 3: Submitting Pig Language Code

Details

This PROC HADOOP example submits Pig language code into a Hadoop cluster. This is the Pig language code to be executed:

```
A = LOAD '/user/sasabc/testdata.txt' USING PigStorage(',')
    AS (customer_number,account_number,tax_id,date_of_birth,status,
        residence_country_code,marital_status,email_address,phone_number,
        annual_income,net_worth_amount,risk_classification);
B = FILTER A BY marital_status == 'Single';
store B into '/user/sasabc/output_customer' USING PigStorage(',');
```

Program

```
filename cfg "C:\Users\sasabc\hadoop\sample_config.xml";1
filename code "C:\Users\sasabc\hadoop\sample_pig.txt";2

proc hadoop options=cfg username="sasabc" password="sasabc" verbose;3
   pig code=code registerjar="C:\Users\sasabc\Hadoop\jars\myudf.jar";4
run;
```

Program Description

1. The first FILENAME statement assigns the file reference CFG to the physical location of a Hadoop configuration file that is named sample_config.xml, which is shown in "Using PROC HADOOP" on page 22.
2. The second FILENAME statement assigns the file reference CODE to the physical location of the file that contains the Pig language code that is named sample_pig.txt, which is shown above.
3. The PROC HADOOP statement controls access to the Hadoop server by referencing the Hadoop configuration file with the OPTIONS= option, identifying the user ID and password on the Hadoop server with the USERNAME= and PASSWORD= options, and specifying the VERBOSE option, which enables additional messages to the SAS log.
4. The PIG statement includes the following options:
   - CODE= to specify the SAS fileref CODE that is assigned to the physical location of the file that contains the Pig language code
   - REGISTERJAR= to specify the JAR file that contains the Pig scripts to execute.

Chapter 3

FILENAME, Hadoop Access Method

Dictionary

FILENAME Statement, Hadoop Access Method

Enables you to access files on a Hadoop Distributed File System (HDFS) whose location is specified in a configuration file.

Valid in: Anywhere

Category: Data Access

Restriction: Access to Hadoop configurations on systems based on UNIX

Syntax

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

Required Arguments

fileref
is a valid fileref.

Tip: The association between a fileref and an external file lasts only for the duration of the SAS session or until you change it or discontinue it with another FILENAME statement.

HADOOP
specifies the access method that enables you to use Hadoop to read from or write to a file from any host machine that you can connect to on a Hadoop configuration.

'*external-file*'
specifies the physical name of the file that you want to read from or write in an HDFS system. The physical name is the name that is recognized by the operating environment.

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

Tip: Specify *external-file* when you assign a fileref to an external file. You can associate a fileref with a single file or with an aggregate file storage location.

Hadoop Options

hadoop-options can be any of the following values:

BUFFERLEN=*bufferlen*

specifies the maximum buffer length of the data that is passed to Hadoop for its I/O operations.

Default: 503808

Restriction: The maximum buffer length is 1000000.

Tip: Specifying a buffer length that is larger than the default could result in performance improvements.

CFG="*physical-pathname-of-hadoop-configuration-file*" | *fileref-that-references-a-hadoop-configuration-file*

specifies the configuration file that contains the connections settings for a specific Hadoop cluster.

CONCAT

specifies that the HDFS directory name that is specified on the FILENAME HADOOP statement is considered a wildcard specification. The concatenation of all the files in the directory is treated as a single logical file and read as one file.

Restriction: This works for input only.

Tip: For best results, do not concatenate text and binary files.

DIR

enables you to access files in an HDFS directory.

Requirement: You must use valid directory syntax for the specified host.

Interaction: Specify the HDFS directory name in the *external-file* argument.

See: "FILEEXT" on page 32

ENCODING='*encoding-value*'

specifies the encoding to use when SAS is reading from or writing to an external file. The value for ENCODING= indicates that the external file has a different encoding from the current session encoding.

Default: SAS assumes that an external file is in the same encoding as the session encoding.

Note: When you read data from an external file, SAS transcodes the data from the specified encoding to the session encoding. When you write data to an external file, SAS transcodes the data from the session encoding to the specified encoding.

See: "Encoding Values in SAS Language Elements" in the *SAS National Language Support (NLS): Reference Guide*

FILEEXT

specifies that a file extension is automatically appended to the filename when you use the DIR option

Interaction: The autocall macro facility always passes the extension .SAS to the file access method as the extension to use when opening files in the autocall library. The DATA step always passes the extension .DATA. If you define a fileref for an autocall macro library and the files in that library have a file extension of .SAS, use the FILEEXT option. If the files in that library do not have an extension, do not use the FILEEXT option. For example, if you define a fileref for an input file in the DATA step and the file X has an extension of .DATA, you

would use the FILEEXT option to read the file X.DATA. If you use the INFILE or FILE statement, enclose the member name and extension in quotation marks to preserve case.

Tip: The FILEEXT option is ignored if you specify a file extension on the FILE or INFILE statement.

See: "LOWCASE_MEMNAME" on page 33

LOWCASE_MEMNAME
enables autocall macro retrieval of lowercase directory or member names from HDFS systems.

Restriction: SAS autocall macro retrieval always searches for uppercase directory member names. Mixed-case directory or member names are not supported.

See: "FILEEXT" on page 32

LRECL=*logical-record-length*
specifies the logical record length of the data.

Default: 65536

Interaction: Alternatively, you can specify a global logical record length by using the LRECL= system option. For more information, see *SAS System Options: Reference*

MOD
places the file in Update mode and appends updates to the bottom of the file.

PASS='*password*'
specifies the password to use with the user name that is specified in the USER option.

Requirement: The password is case sensitive and it must be enclosed in single or double quotation marks.

Tip: To use an encoded password, use the PWENCODE procedure in order to disguise the text string, and then enter the encoded password for the PASS= option. For more information see the PWENCODE procedure in the *Base SAS Procedures Guide*.

PROMPT
specifies to prompt for the user login, the password, or both, if necessary.

Interaction: The USER= and PASS= options override the PROMPT option if all three options are specified. If you specify the PROMPT option and do not specify the USER= or PASS= option, you are prompted for a user ID and password.

RECFM=*record-format*
where *record-format* is one of three record formats:

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

Tip: The amount of data that is read is controlled by the current LRECL value or the value of the NBYTE= variable in the INFILE statement. The NBYTE= option specifies a variable that is equal to the amount of data to be read. This amount must be less than or equal to LRECL. To avoid problems when you read large binary files like PDF or GIF, set NBYTE=1 to read one byte at a time.

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

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

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

Tip: Any record larger than LRECL is truncated.

Default: V

USER='*username*'
where *username* is used to log on to the Hadoop system.

Requirement: The user name is case sensitive and it must be enclosed in single or double quotation marks.

Details

An HDFS system has defined levels of permissions at both the directory and file level. The Hadoop access method honors those permissions. For example, if a file is available as read-only, you cannot modify it.

Operating Environment Information
Using the FILENAME statement requires information that is specific to your operating environment. The Hadoop access method is fully documented here. For more information about how to specify filenames, see the SAS documentation for your operating environment.

Examples

Example 1: Writing to a New Member of a Directory

This example writes the file **shoes** to the directory **testing**.

```
filename out hadoop '/user/testing/' cfg="/path/cfg.xml"  user='xxxx'
  pass='xxxx' recfm=v  lrecl=32167 dir ;

data _null_;
   file out(shoes) ;
   put 'write data to shoes file';
run;
```

Example 2: Creating and Using a Configuration File

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

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

data _null_;
   file cfg;
   input;
   put _infile_;
   datalines4;
<configuration>
<property>
   <name>fs.default.name</name>
   <value>hdfs://xxx.unx.sas.com:8020</value>
</property>
```

```
</property>
   <name>mapred.job.tracker</name>
   <value>xxx.unx.sas.com:8021</value>
</property>
</configuration>

;;;;

filename foo hadoop '/user/xxxx/acctdata.dat' cfg=cfg user='xxxx'
    pass='xxxx' debug recfm=s lrecl=65536 bufferlen=65536;

data _null_;
   infile foo truncover;
   input a $1024.;
   put a;
run;
```

Example 3: Buffering 1MB of Data during a File Read

This example uses the BUFFERLEN option to buffer 1MB of data at time during the file read. The records of length 1024 are read from this buffer.

```
filename foo hadoop 'file1.dat' cfg='U=/hadoopcfg.xml'
   user='user' pass='apass' recfm=s
   lrecl=1024 bufferlen=1000000;

data _null_;
   infile foo truncover;
input a $1024.;
put a;
run;
```

Example 4: Using the CONCAT Option

This example uses the CONCAT option to read all members of DIRECTORY1 as if they are one file.

```
filename foo hadoop '/directory1/' cfg='U=/hadoopcfg.xml'
   user='user' pass='apass' recfm=s lrecl=1024 concat;

data _null_;
   infile foo truncover;
input a $1024.;
put a;
run;
```

See Also

Statements

- FILENAME Statement in the *SAS Statements: Reference*
- FILENAME Statement, CATALOG Access Method in the *SAS Statements: Reference*
- FILENAME Statement, EMAIL (SMTP) Access Method in the *SAS Statements: Reference*
- FILENAME Statement, FTP Access Method in the *SAS Statements: Reference*

- FILENAME Statement, SOCKET Access Method in the *SAS Statements: Reference*
- FILENAME Statement, URL Access Method in the *SAS Statements: Reference*
- FILENAME Statement, WebDAV Access Method in the *SAS Statements: Reference*

Index