Oracle® Big Data Connectors

User's Guide

Release 1 (1.0)

E27365-06

June 2012

Oracle Big Data Connectors User's Guide, Release 1 (1.0)

E27365-06

Copyright © 2011, 2012, Oracle and/or its affiliates. All rights reserved.

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this is software or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, the following notice is applicable:

U.S. GOVERNMENT END USERS: Oracle programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, delivered to U.S. Government end users are "commercial computer software" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, shall be subject to license terms and license restrictions applicable to the programs. No other rights are granted to the U.S. Government.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Xeon are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Opteron, the AMD logo, and the AMD Opteron logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

This software or hardware and documentation may provide access to or information on content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services.

Cloudera, Cloudera CDH, and Cloudera Manager are registered and unregistered trademarks of Cloudera, Inc.

Contents

3 Oracle Loader for Hadoop

4 Oracle Data Integrator Application Adapter for Hadoop

5 Oracle R Connector for Hadoop

Index

Preface

The *Oracle Big Data Connectors User's Guide* describes how to install and use Oracle Big Data Connectors:

- Oracle Loader for Hadoop
- Oracle R Connector for Hadoop
- Oracle Direct Connector for Hadoop Distributed File System
- Oracle Data Integrator Application Adapter for Hadoop

Audience

This document is intended for users of Oracle Big Data Connectors, including the following:

- Application developers
- Java programmers
- System administrators
- Database administrators

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc`.

Access to Oracle Support

Oracle customers have access to electronic support through My Oracle Support. For information, visit `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info` or visit `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs` if you are hearing impaired.

Related Documents

For more information, see the following documents:

- *Oracle Loader for Hadoop Java API Reference*
- *Oracle Fusion Middleware Application Adapters Guide for Oracle Data Integrator*

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

1

Getting Started with Oracle Big Data Connectors

This chapter introduces you to Oracle Big Data Connectors, provides installation instructions, and identifies the permissions needed for users to access the connectors.

This chapter contains these topics:

About Oracle Big Data Connectors

Oracle Big Data Connectors facilitate data access between data stored in a Hadoop cluster and Oracle Database. They can be licensed for use on either Oracle Big Data Appliance or a Hadoop cluster running on commodity hardware.

These are the connectors:

- **Oracle Direct Connector for Hadoop Distributed File System**: Enables Oracle Database to access data stored in a Hadoop Distributed File System (HDFS). The data can remain in HDFS or it can be loaded into Oracle Database.
- **Oracle Loader for Hadoop**: Provides an efficient and high performance loader for fast movement of data from a Hadoop cluster into a table in an Oracle database. Oracle Loader for Hadoop prepartitions the data if necessary and transforms it into an Oracle-ready format. It optionally sorts records by primary key before loading the data or creating output files. Oracle Loader for Hadoop is a MapReduce application that is invoked as a command line utility. It accepts the generic command-line options that are supported by the Tool interface.
- **Oracle Data Integrator Application Adapter for Hadoop**: Extracts, transforms, and loads data from a Hadoop cluster into tables in Oracle Database, as defined using a graphical user interface.
- **Oracle R Connector for Hadoop**: Provides an interface between a local R environment, Oracle Database, and Hadoop, allowing speed-of-thought, interactive analysis on all three platforms. Oracle R Connector for Hadoop is designed to work independently, but if the enterprise data for your analysis is also

stored in Oracle Database, then the full power of this connector is achieved when it is used with Oracle R Enterprise.

Individual connectors may require that software components are installed in Oracle Database, the Hadoop cluster, and the user's PC. Users may also need additional access privileges in Oracle Database.

> **See Also:** My Oracle Support Master Note 1416116.1 and its related notes

Downloading the Software

You can download Oracle Big Data Connectors from Oracle Technology Network (OTN) or Oracle Delivery Cloud.

To download from OTN:

1. Use any browser to visit this website:

   `http://www.oracle.com/technetwork/bdc/big-data-connectors/downloads/index.html`

2. Click the name of each connector to download a zip file containing the installation files.

To download from Oracle Software Delivery Cloud:

1. You can also download the software from Oracle Software Delivery Cloud at

   `https://edelivery.oracle.com/`

2. Accept the Terms and Restrictions to see the Media Pack Search page.
3. Select the search terms:

   Select a Product Pack: Oracle Database

   Platform: Linux x86-64

4. Click **Go** to display a list of product packs.
5. Select Oracle Big Data Connectors Media Pack for Linux x86-64 (B65965-0*x*), then click **Continue**.
6. Click **Download** for each connector to download a zip file containing the installation files.

Oracle Direct Connector for Hadoop Distributed File System

Oracle Direct Connector for Hadoop Distributed File System (Oracle Direct Connector) is installed and runs on the system where Oracle Database runs. Before installing Oracle Direct Connector, verify that you have the required software.

Required Software

Oracle Direct Connector requires the following software:

- Cloudera's Distribution including Apache Hadoop Version CDH3 or Apache Hadoop 0.20.2.
- Oracle JDK 1.6.0_8 or higher for CDH3. Cloudera recommends version 1.6.0_26.
- Oracle Database Release 11*g* Release 2 (11.2.0.2 or 11.2.0.3) for Linux.

- To support the Data Pump file format, a database one-off patch. To download this patch, go to `http://support.oracle.com` and search for bug 13079417.
- The same version of Hadoop on the database system as your Hadoop cluster, either CDH3 or Apache Hadoop 0.20.2.
- The same version of Oracle JDK on the database system as your Hadoop cluster.

Installing and Configuring Hadoop

Oracle Direct Connector works as an HDFS client. You do not need to configure Hadoop on the database system to run MapReduce jobs for Oracle Direct Connector. However, you must install Hadoop on the database system and minimally configure it for HDFS client use only.

To configure the database system as a Hadoop client:

1. Install CDH3 or Apache Hadoop 0.20.2 on the database system. Follow the installation instructions provided by the distributor (Cloudera or Apache). Do not follow the configuration instructions.
2. Use a text editor to open `conf/hadoop-env.sh` in the Hadoop home directory on the database system, then make these changes:
   a. Uncomment the line that begins `export JAVA_HOME`.
   b. Set `JAVA_HOME` to the directory where JDK1.6 is installed.
3. Edit `conf/core-site.xml` in the same directory to identify the NameNode of your Hadoop cluster as follows:

```
<configuration>
   <property>
      <name>fs.default.name</name>
      <value>hdfs://host:port</value>
    </property>
</configuration>
```

4. Ensure that Oracle Database has access to Hadoop and HDFS:
   a. Log in to the system where Oracle Database is running using the Oracle database account.
   b. Open a bash shell and issue this command:

```
$HADOOP_HOME/bin/hadoop fs -ls /user
```

   In this command, `$HADOOP_HOME` is the absolute path to the Hadoop home directory. You should see a list of files. If not, then first ensure that the Hadoop cluster is up and running. If the problem persists, then you must correct the Hadoop client configuration so that Oracle Database has access to the Hadoop cluster file system.

The database system is now ready for use as a Hadoop client. No other Hadoop configuration steps are needed.

Installing Oracle Direct Connector

To install Oracle Direct Connector:

1. Download the zip file to a directory on the system where Oracle Database runs.
2. Unzip `orahdfs-`*`version`*`.zip` into a directory. The unzipped files have the structure shown in Example 1–1.

3. Open the hdfs_stream bash shell script in a text editor and make these changes:

   - HADOOP_HOME: Set to the absolute path of the Hadoop home directory.
   - DIRECTHDFS_HOME: Set to the absolute path of the Oracle Direct Connector installation directory.

   The hdfs_stream script is the preprocessor script for the HDFS external table. Comments in the script provide complete instructions for making these changes.

4. Run the hdfs_stream script from the Oracle Direct Connector installation directory. You should see this usage information:

   ```
   $ bin/hdfs_stream
   Oracle Direct HDFS Release 1.0.0.0.0 - Production
   Copyright (c) 2011, Oracle and/or its affiliates. All rights reserved.
   Usage: $HADOOP_HOME/bin/hadoop jar orahdfs.jar
   oracle.hadoop.hdfs.exttab.HdfsStream <locationPath>
   ```

   If not, then ensure that the operating system user that Oracle is running under has the following permissions:

   - Read and execute permissions on the hdfs_stream script:

     ```
     $ ls -l DIRECTHDFS_HOME/bin/hdfs_stream
     -rwxr-xr-x 1 oracle oinstall 2273 Apr 27 15:51 hdfs_stream
     ```

     If you do not see these permissions, then issue a chmod command to fix them:

     ```
     $ chmod 755 DIRECTHDFS_HOME/bin/hdfs_stream
     ```

     In these commands, *DIRECTHDFS_HOME* represents the Oracle Direct Connector home directory.

   - Read permission on *DIRECTHDFS_HOME*/jlib/orahdfs.jar.

5. Create a database directory for the orahdfs-*version*/bin directory where hdfs_stream resides. In this example, the Oracle Direct Connector kit is installed in /etc:

   ```
   SQL> CREATE OR REPLACE DIRECTORY hdfs_bin_path AS  '/etc/orahdfs-1.0/bin'
   ```

Example 1–1 Structure of the orahdfs Directory

```
orahdfs-version
   bin/
      hdfs_stream
   jlib/
      orahdfs.jar
   log/
   README.txt
```

Granting User Access to Oracle Direct Connector

Oracle Database users require these privileges to use Oracle Direct Connector:

- CREATE SESSION
- EXECUTE on the UTL_FILE PL/SQL package.
- READ and EXECUTE on the HDFS_BIN_PATH directory created in Step 5. Do not grant write access to anyone. Grant EXECUTE only to those who intend to use Oracle Direct Connector.

Example 1–2 shows the SQL commands granting these privileges to HDFSUSER.

Example 1–2 Granting Users Access to Oracle Direct Connector

```
CONNECT / AS sysdba;
CREATE USER hdfsuser IDENTIFIED BY password;
GRANT CREATE SESSION TO hdfsuser;
GRANT EXECUTE ON SYS.UTL_FILE TO hdfsuser;
GRANT READ, EXECUTE ON DIRECTORY hdfs_bin_path TO hdfsuser;
```

Oracle Loader for Hadoop

Before installing Oracle Loader for Hadoop, verify that you have the required software.

Required Software

Oracle Loader for Hadoop requires the following software:

- A target database system running one of the following:
  - Oracle Database 10*g* Release 2 (10.2.0.5) with required patch
  - Oracle Database 11*g* Release 2 (11.2.0.2) with required patch
  - Oracle Database 11*g* Release 2 (11.2.0.3)

  > **Note:** To use Oracle Loader for Hadoop with Oracle Database 10*g* Release 2 (10.2.0.5) or Oracle Database 11*g* Release 2 (11.2.0.2), you must first apply a one-off patch that addresses bug number 11897896. To access this patch, go to `http://support.oracle.com` and search for the bug number.

- Cloudera's Distribution including Apache Hadoop (CDH3) or Apache Hadoop 0.20.2
- Hive 0.7.0 or 0.7.1, if using the `HiveToAvroInputFormat` class

Installing Oracle Loader for Hadoop

Oracle Loader for Hadoop is packaged with the Oracle Database 11*g* Release 2 client libraries and Oracle Instant Client libraries for connecting to Oracle Database 10.2.0.5, 11.2.0.2, or 11.2.0.3.

To install Oracle Loader for Hadoop:

1. Unpack the content of the `oraloader-`*`version`*`.zip` archive into a directory on your Hadoop cluster.

   A directory named `oraloader-`*`version`* is created with the following subdirectories:

   - `doc`
   - `jlib`
   - `lib`
   - `examples`

   This guide uses the variable `${OLH_HOME}` to refer to this installation directory.

2. Add `${OLH_HOME}/jlib/*` to the `HADOOP_CLASSPATH` variable.

Oracle Data Integrator Application Adapter for Hadoop

Installation requirements for Oracle Data Integrator Application Adapter for Hadoop are provided in these topics:

- System Requirements and Certifications
- Technology Specific Requirements

System Requirements and Certifications

To use the Application Adapter for Hadoop, you must first have Oracle Data Integrator, which is licensed separately from Oracle Big Data Connectors. You can download Oracle Data Integrator from the Oracle website at

`http://www.oracle.com/technetwork/middleware/data-integrator/downloads/index.html`

Oracle Data Integrator Application Adapter for Hadoop Knowledge Modules require a minimum version of Oracle Data Integrator 11.1.1.6.0.

Before performing any installation, read the system requirements and certification documentation to ensure that your environment meets the minimum installation requirements for the products you are installing.

The list of supported platforms and versions is available on Oracle Technical Network:

`http://www.oracle.com/technology/products/oracle-data-integrator/index.html`

Technology Specific Requirements

The list of supported technologies and versions is available on Oracle Technical Network:

`http://www.oracle.com/technology/products/oracle-data-integrator/index.html`

Location of the Oracle Data Integrator Application Adapter for Hadoop

Oracle Data Integrator Application Adapter for Hadoop is available in the `xml-reference` directory of the Oracle Data Integrator Companion CD.

Setting Up the Topology

See Chapter 4, "Oracle Data Integrator Application Adapter for Hadoop."

Oracle R Connector for Hadoop

Oracle R Connector for Hadoop requires the installation of a software environment on the Hadoop side and on a client Linux system.

Installing the Server Software

Oracle Big Data Appliance supports Oracle R Connector for Hadoop without any additional software installation or configuration.

To use Oracle R Connector for Hadoop on any other Hadoop cluster, you must create the necessary environment.

Install these components on third-party servers:

- Java Virtual Machine (JVM), preferably Java HotSpot Virtual Machine 6.
- R distribution 2.13.2 with all base libraries on all nodes in the Hadoop cluster.
- ORHC package installed on each R engine, which must exist on every node of the Hadoop cluster. See the following instructions.

To install ORHC:

1. Set the environment variables for the Hadoop and JVM home directories:

   ```
   $ setenv HADOOP_HOME /usr/lib/hadoop-0.2.0
   $ setenv JAVA_HOME /usr/lib/jdk6
   ```

   In this example, both home directories are in /usr/lib.

2. Unzip the downloaded file:

   ```
   $ unzip orhc.tgz.zip
   Archive:  orhc.tgz.zip
   ```

3. Open R and install the package:

   ```
   > install.packages("/home/tmp/orhc.tgz", repos=NULL)
   Installing package(s) into ...
   .
   .
   .
   Hadoop is up and running.
   ```

4. Alternatively, you can install the package from the Linux command line:

   ```
   $ R CMD INSTALL orhc.tgz
   * installing *source* package 'ORHC' ...
   ** R
   .
   .
   .
   Hadoop is up and running.

   * DONE (ORHC)
   ```

Installing the Client Software

To provide access to a Hadoop cluster to R users, install these components on a Linux server:

- Hadoop Client to allow access to the Hadoop cluster

  For Oracle Big Data Appliance, see the *Oracle Big Data Appliance Software User's Guide* for detailed instructions on setting up remote client access.

- Java Virtual Machine, preferably Java HotSpot Virtual Machine 6
- R distribution 2.13.2
- ORHC R package

  Follow the steps for installing ORHC in "Installing the Server Software" on page 1-6.

- Oracle R Enterprise libraries (optional). They support access to Oracle Database; otherwise, Oracle R Connector for Hadoop operates only with in-memory R objects and local data files without access to the advanced statistical algorithms provided by Oracle R Enterprise. For example:

```
library(DBI)
library(ROracle)
library(OREbase)
library(OREeda)
library(OREgraphics)
library(OREstats)
library(RToXmp)
```

When you are done, ensure that users have the necessary permissions to connect to the Linux server and run R.

2

Oracle Direct Connector for Hadoop Distributed File System

This chapter describes how use Oracle Direct Connector for Hadoop Distributed File System (Oracle Direct Connector) to facilitate data access between a Hadoop Distributed File System (HDFS) and Oracle Database.

This chapter contains the following topics:

About Oracle Direct Connector

Oracle Direct Connector runs on the system where Oracle Database runs. It provides read access to HDFS from Oracle Database by using external tables.

An **external table** is an Oracle Database object that identifies the location of data outside of the database. Oracle Database accesses the data by using the metadata provided when the external table was created. By querying the external tables, users can access data stored in HDFS as if that data were stored in tables in the database. External tables are often used to stage data to be transformed during a database load.

These are a few ways that you can use Oracle Direct Connector:

- Access any data stored in HDFS files
- Access CSV files and Data Pump files generated by Oracle Loader for Hadoop
- Load data extracted and transformed by Oracle Data Integrator

Oracle Direct Connector uses the `ORACLE_LOADER` access driver.

Note: Oracle Direct Connector requires a database patch before it can access Data Pump files produced by Oracle Loader for Hadoop. To download this patch, go to `http://support.oracle.com` and search for bug 13079417.

See Also:

- *Oracle Database Administrator's Guide* for information about external tables
- *Oracle Database Utilities* for more information about Oracle external tables, performance hints, and restrictions when using the ORACLE_LOADER access driver

Creating an External Table for HDFS

You create an external table for HDFS the same as any other external table, except that you must specify this PREPROCESSOR clause in the SQL CREATE TABLE command:

```
PREPROCESSOR HDFS_BIN_PATH:hdfs_stream
```

HDFS_BIN_PATH is the name of the Oracle directory object that points to the bin subdirectory where Oracle Direct Connector is installed. See "Installing Oracle Direct Connector" on page 1-3.

To access Data Pump files, you must also specify this access parameter:

```
EXTERNAL VARIABLE DATA
```

Basic SQL Syntax for the External Table

Following is the basic SQL syntax for creating an external table for HDFS:

```
CREATE TABLE [schema.]table
     (   column datatype, ...
     )
     ORGANIZATION EXTERNAL
     (
         TYPE ORACLE_LOADER
         DEFAULT DIRECTORY directory
         ACCESS PARAMETERS
        (   PREPROCESSOR HDFS_BIN_PATH:hdfs_stream
            access_parameters...
        )
     LOCATION (file1,file2...)
     );
```

schema.table
Name of the external table to be created.

column
Name of the columns in the table.

datatype
Data type of the column, which is limited to the ones supported by ORACLE_LOADER. It performs some data type conversions automatically.

directory
Name of the default directory object used by the external table for all input and output files, such as the location files, log files, and bad record files. Do not use the directory where the data is stored for these files.

access_parameters
Any additional subclauses in the ORACLE_LOADER access_parameters clause, such as record and field formatting.

***file1*, *file2*...**

The names of the location files that identify the paths to the data in HDFS. For CSV content in HDFS, specify two or more location file names, because the degree of parallelism is limited by the number of location files. For Data Pump content in HDFS, specify one location file for each Data Pump file.

Oracle Direct Connector creates these files in the default directory. If the files already exist, they are overwritten.

For other types of external tables, the location files contain the data, but Oracle Direct Connector retains the data in HDFS.

Testing the External Table

After creating the external table, query it to verify that the preprocessor script is configured correctly:

```
SELECT count(*) FROM external_table;
```

If the query returns no rows and no errors, then you can continue.

External Table Example

Example 2–1 creates an external table named `SALES_HDFS_EXT_TAB` in the `SCOTT` schema. The `SALES_HDFS_EXT_TAB` external table is created in a database directory named `SALES_EXT_DIR`. `SCOTT` must have read and write privileges on this directory.

To create the SALES_EXT_DIR database directory:

1. Create the file system directory:

   ```
   $ mkdir /scratch/sales_ext_dir
   $ chmod 664 /scratch/sales_ext_dir
   ```

2. Open a SQL command interface:

   ```
   $ sqlplus / as sysdba
   ```

3. Create the database directory:

   ```
   SQL> CREATE OR REPLACE DIRECTORY sales_ext_dir AS '/scratch/sales_ext_dir'
   ```

4. Grant read and write access to `SCOTT`:

   ```
   SQL> GRANT READ, WRITE ON DIRECTORY sales_ext_dir TO scott;
   ```

Example 2–1 Defining an External Table for HDFS

```
CREATE TABLE "SCOTT"."SALES_HDFS_EXT_TAB"
    (  "PROD_ID"          NUMBER(6),
       "CUST_ID"          NUMBER,
       "TIME_ID"          DATE,
       "CHANNEL_ID"       CHAR(1),
       "PROMO_ID"         NUMBER(6),
       "QUANTITY_SOLD"    NUMBER(3),
       "AMOUNT_SOLD"      NUMBER(10,2)
    )
    ORGANIZATION EXTERNAL
    ( TYPE ORACLE_LOADER
      DEFAULT DIRECTORY  "SALES_EXT_DIR"
      ACCESS PARAMETERS
      (   RECORDS DELIMITED BY NEWLINE
```

```
      FIELDS TERMINATED BY ','
        (
        "PROD_ID" DECIMAL EXTERNAL,
           .
           .
           .
        "TIME_ID" CHAR DATE_FORMAT TIMESTAMP MASK "...",
           .
           .
           .
        )
      PREPROCESSOR HDFS_BIN_PATH:hdfs_stream
  )
  LOCATION ( 'sales1','sale2','sales3')
);
```

Publishing the HDFS Data Paths

The previous procedure for creating an external table only created the metadata in Oracle Database. As mentioned earlier, the location files typically store the data values. In this case, however, the location files are empty. By executing the Oracle Direct Connector `ExternalTable` command-line tool, you populate the location files with the Universal Resource Identifiers (URIs) of the data files in HDFS. When users query the external table, the Oracle Direct Connector preprocessor uses that information to locate the data in HDFS and stream it to the database.

ExternalTable Command

The `ExternalTable` command uses the values of several properties to populate the location files. You can specify these property values in an XML document or individually on the command line.

Altering HADOOP_CLASSPATH

Before issuing the `ExternalTable` command, alter the `HADOOP_CLASSPATH` environment variable to include these JAR files:

- `$ORACLE_HOME/jdbc/lib/ojdbc6.jar`: Required.
- `$ORACLE_HOME/jlib/oraclepki.jar`: Required only if you using Oracle wallet as an external password store.

See "ExternalTable Command Example".

ExternalTable Command Syntax

This is the syntax of the `ExternalTable` command:

```
$HADOOP_HOME/bin/hadoop jar $DIRECTHDFS_HOME/jlib/orahdfs.jar
oracle.hadoop.hdfs.exttab.ExternalTable [-conf config_file | -D property=value]
-publish [-noexecute]
```

-conf *config_file*
Identifies the name of an XML configuration file containing the properties needed to populate the location files. See "Creating a Configuration File" on page 2-5.

-D *property=value*
Assigns a value to a specific property

--noexecute
Generates an execution plan, but does not execute any of the actions.

ExternalTable Command Example

Example 2–2 sets the HADOOP_CLASSPATH variable and publishes HDFS data paths to the external table created in Example 2–1.

Example 2–2 Publishing HDFS Data Paths to an External Table

This example uses the Bash shell.

```
$  export HADOOP_CLASSPATH="$ORACLE_HOME/jdbc/lib/ojdbc6.jar:$ORACLE_
HOME/jlib/oraclepki.jar"

$ $HADOOP_HOME/bin/hadoop jar \
  $DIRECTHDFS_HOME/jlib/orahdfs.jar oracle.hadoop.hdfs.exttab.ExternalTable  \
  -D oracle.hadoop.hdfs.exttab.tableName=SALES_HDFS_EXT_TAB \
  -D oracle.hadoop.hdfs.exttab.datasetPaths=hdfs:/user/scott/data/ \
  -D oracle.hadoop.hdfs.exttab.connection.url=jdbc:oracle:thin@myhost:1521/orcl \
  -D oracle.hadoop.hdfs.exttab.connection.user=scott -publish
```

Where:

- `$HADOOP_HOME` is an environment variable pointing to the Hadoop home directory.
- `$DIRECTHDFS_HOME` is an environment variable pointing to the Oracle Direct Connector installation directory.
- `SALES_HDFS_EXT_TAB` is the external table created in Example 2–1.
- `hdfs:/user/scott/data/` is the location of the HDFS data.
- `@myhost:1521/orcl` is the database connection string.

Creating a Configuration File

A configuration file is an XML document with a very simple structure as follows:

```
<?xml version="1.0"?>
 <configuration>
    <property>
      <name>property</name>
      <value>value</value>
    </property>
        .
        .
        .
</configuration>
```

See "Configuration Properties" on page 2-6 for descriptions of these properties.

Example 2–3 shows a configuration file.

Example 2–3 Configuration File for Oracle Direct Connector

```
<?xml version="1.0"?>
 <configuration>
    <property>
      <name>oracle.hadoop.hdfs.exttab.tableName</name>
      <value>SH.SALES_EXT_DIR</value>
    </property>
   <property>
```

```
        <name>oracle.hadoop.hdfs.exttab.datasetPaths</name>
        <value>/data/s1/*.csv,/data/s2/*.csv</value>
      </property>
      <property>
        <name>oracle.hadoop.hdfs.exttab.datasetCompressionCodec</name>
        <value>org.apache.hadoop.io.compress.DefaultCodec</value>
      </property>
      <property>
        <name>oracle.hadoop.hdfs.exttab.connection.url</name>
        <value>
         jdbc:oracle:thin:@example.com:1521/example.example.com
       </value>
      </property>
      <property>
        <name>oracle.hadoop.hdfs.exttab.connection.user</name>
        <value>SH</value>
      </property>
  </configuration>
```

Configuration Properties

Following are the configuration properties used by the ExternalTable command to create links to the data files in HDFS. These properties are required:

oracle.hadoop.hdfs.exttab.connection.url
oracle.hadoop.hdfs.exttab.datasetPaths
oracle.hadoop.hdfs.exttab.tableName

Property Descriptions

oracle.hadoop.hdfs.exttab.connection.url
The URL of the database connection string. This property overrides all other connection properties. The connecting database user must have the privileges described in "Granting User Access to Oracle Direct Connector" on page 1-4. Required.

Using a Wallet

If you are using an Oracle wallet as an external password store, then the property value must have this form:

```
jdbc:oracle:thin:@db_connect_string
```

The *db_connect_string* must exactly match the credential in the wallet.

This example uses Oracle Net Services syntax:

```
jdbc:oracle:thin:@(DESCRIPTION=(ADDRESS_LIST=
   (ADDRESS=(PROTOCOL=TCP)(HOST=myhost)(PORT=1521)))
      (CONNECT_DATA=(SERVICE_NAME=my_db_service_name)))
```

The next example uses a TNSNAMES entry:

```
jdbc:oracle:thin:@my_tns_entry
```

See also oracle.hadoop.hdfs.exttab.connection.wallet_location.

Not Using a Wallet

If you are not using an Oracle wallet, then use one of the following URL connection styles.

- Thin Connection:

```
jdbc:oracle:thin:@//myhost:1521/my_db_service_name
```

- Oracle Net Services:

```
jdbc:oracle:thin:@(DESCRIPTION=(ADDRESS_LIST=
   (ADDRESS=(PROTOCOL=TCP)(HOST=myhost)(PORT=1521)))
      (CONNECT_DATA=(SERVICE_NAME=my_db_service_name)))
```

- TNS Entry Name:

```
jdbc:oracle:thin:@myTNSEntryName
```

This parameter is required when not using a wallet for connections:

- oracle.hadoop.hdfs.exttab.connection.user

oracle.hadoop.hdfs.exttab.connection.user
An Oracle Database user name.

oracle.hadoop.hdfs.exttab.connection.tns_admin
File path to a directory containing SQL*Net configuration files, such as `sqlnet.ora` and `tnsnames.ora`. The value of the `TNS_ADMIN` environment variable is used for this property by default.

Define this property to use TNS entry names in database connect strings.

oracle.hadoop.hdfs.exttab.connection.tnsEntryName
A TNS entry name defined in the `tnsnames.ora` file. This property is used with oracle.hadoop.hdfs.exttab.connection.tns_admin.

oracle.hadoop.hdfs.exttab.datasetCompressionCodec
The class name of the compression codec that implements the `org.apache.hadoop.io.compress.CompressionCodec` interface. The Decompressor class from the codec is used by the preprocessor script to decompress data for the external table. This codec applies to the entire data set.

Specify this property if the data set contains compressed files.

oracle.hadoop.hdfs.exttab.datasetPathFilter
The class name of a path filter that implements the `org.apache.hadoop.fs.PathFilter` interface. The paths in the data set are selected only if this filter class accepts them.

oracle.hadoop.hdfs.exttab.datasetPaths
A comma-separated list of fully qualified HDFS paths. This parameter enables you to restrict the input by using special pattern-matching characters in the path specification. See Table 2–1. Required.

For example, to select all files in `/data/s2/`, and only the CSV files in `/data/s7/`, `/data/s8/`, and `/data/s9/`, enter this expression:

```
/data/s2/,/data/s[7-9]/*.csv
```

The external table accesses the data contained in all listed files and all files in listed directories. These files compose a single data set.

The data set can contain compressed files or uncompressed files, but not both.

Table 2–1 Pattern Matching Characters

Character	Description
?	Matches any single character
*	Matches zero or more characters
[*abc*]	Matches a single character from the character set {*a, b, c*}.
[*a*-*b*]	Matches a single character from the character range {*a*...*b*}. The character *a* must be less than or equal to *b*.
[^*a*]	Matches a single character that is not from character set or range {*a*}. The carat (^) must immediately follow the left bracket.
\c	Removes any special meaning of character *c*. The backslash is the escape character.
{*ab*\\,*cd*}	Matches a string from the string set {*ab, cd*}. Precede the comma with an escape character (\\) to remove the meaning of the comma as a path separator.
{*ab*\\,*c*{*de*\\,*fh*}}	Matches a string from the string set {*ab, cde, cfh*}. Precede the comma with an escape character (\\) to remove the meaning of the comma as a path separator.

oracle.hadoop.hdfs.exttab.connection.wallet_location
File path to an Oracle wallet where the connection information is stored. When using Oracle wallet as an external password store, set the following additional properties.

For a URL connection:

- oracle.hadoop.hdfs.exttab.connection.wallet_location
- oracle.hadoop.hdfs.exttab.connection.url

For TNS names:

- oracle.hadoop.hdfs.exttab.connection.wallet_location
- oracle.hadoop.hdfs.exttab.connection.tns_admin
- oracle.hadoop.hdfs.exttab.connection.tnsEntryName

oracle.hadoop.hdfs.exttab.tableName
Schema-qualified name of the external table in the format

schemaName.tableName

See "Creating an External Table for HDFS" on page 2-2 about creating an external table for Oracle Direct Connector. Required.

Querying Data in HDFS

Parallel processing is extremely important when working with large volumes of data. When using external tables, always enable parallel query with this SQL command:

```
ALTER SESSION ENABLE PARALLEL QUERY;
```

Before loading data into Oracle Database from the external files created by Oracle Direct Connector, enable parallel DDL:

```
ALTER SESSION ENABLE PARALLEL DDL;
```

Before inserting data into an existing database table, enable parallel DML with this SQL command:

```
ALTER SESSION ENABLE PARALLEL DML;
```

Hints such as APPEND and PQ_DISTRIBUTE also improve performance when inserting data.

3

Oracle Loader for Hadoop

This chapter explains how to use Oracle Loader for Hadoop to copy data from Hadoop files into tables in Oracle Database. It contains these topics:

What is Oracle Loader for Hadoop?

Oracle Loader for Hadoop is an efficient and high performance loader for fast movement of data from a Hadoop cluster into a table in an Oracle database. Oracle Loader for Hadoop prepartitions the data if necessary and transforms it into an Oracle-ready format. It optionally sorts records by primary key before loading the data or creating output files. Oracle Loader for Hadoop is a MapReduce application that is invoked as a command line utility. It accepts the generic command-line options that are supported by the Tool interface.

> **Note:** Partitioning is a database feature for management and efficient querying of very large tables. It provides a way to decompose a large table into smaller and more manageable pieces called partitions, in a manner entirely transparent to applications. For more information on partitioning, see *Oracle Database VLDB and Partitioning Guide*.

After the pre-partitioning and transforming steps, there are two modes for loading the data into an Oracle database from a Hadoop cluster:

- Online database mode: The data is loaded into the database using either a JDBC output format or an OCI Direct Path output format. The OCI Direct Path output format performs a high performance direct path load of the target table. The JDBC output format performs a conventional path load. For more information about these online load methods, including restrictions for the OCI Direct Path output format, see "Output Modes During OraLoader Invocation" on page 3-7.

- Offline database mode: The reducer nodes create binary or text format output files. The Data Pump output format creates binary format files that are ready to be loaded into an Oracle database using an external table and the ORACLE_DATAPUMP access driver. The Delimited Text output format creates text files in delimited record format. (This is usually called comma separated value (CSV) format when the delimiter is a comma.) These text files are ready to be loaded into an Oracle database using an external table and the ORACLE_LOADER access driver. The files can also be loaded using the SQL*Loader utility.

Using Oracle Loader for Hadoop

This section describes the following steps for using Oracle Loader for Hadoop:

1. Implementing InputFormat
2. Creating the loaderMap Document
3. Accessing Table Metadata
4. Invoking OraLoader
5. Loading Files Into an Oracle Database (Offline Loads Only)

See Chapter 1 for installation instructions.

Implementing InputFormat

Oracle Loader for Hadoop is a Map Reduce application that gets its input from an org.apache.hadoop.mapreduce.RecordReader implementation as provided by the org.apache.hadoop.mapreduce.InputFormat class that is specified in the mapreduce.inputformat.class configuration property. Oracle Loader for Hadoop requires that the RecordReader return an Avro IndexedRecord from the getCurrentKey() method. The method signature should be:

```
public org.apache.avro.generic.IndexedRecord getCurrentKey()
throws IOException, InterruptedException;
```

Oracle Loader for Hadoop uses the schema of the IndexedRecord to discover the names of the input fields and map them to the columns of the table to load. This mapping is discussed in more detail in the following sections.

Oracle Loader for Hadoop comes with two built-in input formats; it also provides the source code for two InputFormat examples. The example source code is located in the jsrc/ directory. Table 3–1 lists the class names for all these input formats, along with the types of input they handle and how they generate the Avro schema field names. (Understanding how an InputFormat generates field names is critical to getting the data loaded into the target table.)

The built-in input format classes are described in the next subsections. For the examples, consult the source code and Javadoc for these classes for more information.

Table 3–1 InputFormat Classes, Types, and Field Names

Class	Input Type	Avro Schema Field Names
`oracle.hadoop.loader.lib.input.HiveToAvroInputFormat`	Hive table sources	Hive table's column names - upper cased
`oracle.hadoop.loader.lib.input.DelimitedTextInputFormat`	Delimited text files	Comma-separated list from the property `oracle.hadoop.loader.input.fieldNames` (or F0, F1, … if property not defined)
`oracle.hadoop.loader.examples.CSVInputFormat`	Simple, delimited text files	F0, F1,...
`oracle.hadoop.loader.examples.AvroInputFormat`	Binary format Avro record files	Field names from input files' Avro schemas

HiveToAvroInputFormat

This class presents an input format that reads data from a Hive table. It requires that the hive database and table names be specified using the following configuration properties:

- `oracle.hadoop.loader.input.hive.tableName`
- `oracle.hadoop.loader.input.hive.databaseName`

`HiveToAvroInputFormat` contacts the `HiveMetaStoreClient` to retrieve information about the table's columns, location, InputFormat, serDe, and so on. Depending on the way in which Hive was configured, additional hive-specific properties must be set (such as `hive.metastore.uris` and `hive.metastore.local`).

`HiveToAvroInputFormat` imports the entire table (all the files in the Hive table's directory). All other (file-based) input formats discussed in this document allow "globbing" (that is, appending wildcard patterns to the input directories to restrict the input.)

The Hive table's rows are transformed into Avro records whose field names are the Hive table's column names in upper case. This usually makes the `loaderMap` issues trivial (see "Creating the loaderMap Document" on page 3-4).

DelimitedTextInputFormat

This is an `InputFormat` for delimited text files, such as comma-separated value or tab-separated value files. `DelimitedTextInputFormat` requires that records be separated by newline characters and that fields be delimited using single-character markers.

The `DelimitedTextInputFormat` is meant to emulate the "terminated by **t** [optionally enclosed by **ie** [and **te**]]" behavior of SQL*Loader. The **t** is the field terminator, **ie** is the initial field encloser, and **te** is the trailing field encloser.

`DelimitedTextInputFormat` uses a parser based on the following grammar:

- Line = Token t Line | Token\n
- Token = EnclosedToken | UnenclosedToken
- EnclosedToken = (white-space)* ie [(non-te)* te te]* (non-te)* te (white-space)*
- UnenclosedToken = (white-space)* (non-t)*

- white-space = {c | Character.isWhitespace(c) and c!=t}

Any trailing field encloser character contained inside an enclosed token must be encoded by "doubling it up" (that is, printing it twice).

White space around enclosed tokens is discarded. For unenclosed tokens the leading white space is discarded, but not the trailing white space (if any).

Any empty-string token (either enclosed or unenclosed) is replaced with a null.

This implementation allows custom enclosers and terminator characters (see Table 3–2), but hard codes the record terminator (to newline) and white space (to Java's `Character.isWhitespace()`). The enclosers must be different from the terminator character and white spaces (but can be equal to each other). The terminator can be a white space (but that value is removed from the class of white space characters).

Table 3–2 describes the delimiters available for `DelimitedTextInputFormat`. In the table, HHHH is a big-endian hexadecimal representation of the character in UTF-16.

Table 3–2 Delimiters for DelimitedTextInputFormat

Delimiter Type	Property	Possible Values	Default
Field terminator	oracle.hadoop.loader.input.fieldTerminator	one character \uHHHH	, (comma)
Initial field encloser	oracle.hadoop.loader.input.initialFieldEncloser	one character \uHHHH nothing	No default
Trailing field encloser	oracle.hadoop.loader.input.trailingFieldEncloser	one character \uHHHH nothing	No default

The field enclosers must be either both set or both not set. If they are not set, then the `EnclosedToken` non-terminal is essentially removed from the grammar previously listed. If field enclosers are set, the parser attempts to read each field as an `EnclosedToken` first before reading it as an `UnenclosedToken`. Note that if the initial field encloser is set, the trailing field encloser must also be set, even when the two are the same.

`DelimitedTextInputFormat` reads the field names as a comma-separated list from the configuration property `oracle.hadoop.loader.input.fieldNames`. If parsing a line results in more tokens (fields) than field names, the extra tokens are discarded. If there are fewer tokens than field names, the missing trailing tokens are set to null.

If the `oracle.hadoop.loader.input.fieldNames` property is not set, then the `DelimitedTextInputFormat`'s RecordReader uses F0, F1,... F*n* as field names (where *n* is the largest number of tokens encountered by that RecordReader in any line so far).

Creating the loaderMap Document

Oracle Loader for Hadoop loads data into a single database table. This table is referred to as the target table. You can use the following ways to specify the target table, the columns to load, and how input fields are mapped to database columns:

- To indicate that all columns of the database table will be loaded and that the names of the input fields exactly match the database column names, use the configuration property `oracle.hadoop.loader.targetTable`. It allows you to define a schema-qualified name for the target load table. For each database column, the loader uses the column name to discover an input field with the same name. The value of the field is then loaded into the column.

- If you want to load a subset of the target table columns or if the input field names are not exactly the same as the database column names, then create a `loaderMap` document to specify the target table, columns, and how the input fields should be mapped to the database columns. The location of the `loaderMap` document is specified using the `oracle.hadoop.loader.loaderMapFile` configuration property.

See Also:

- "Target Table Characteristics" on page 3-21
- "Loader Map XML Schema Definition" on page 3-22 for information about the content of the XML schema definition (XSD) document

Example loaderMap Document

The following example `loaderMap` document specifies a list of columns in the `HR.EMPLOYEES` table that should be loaded. It includes a mapping of input data field names to table column names. It also specifies the format of input data that should be used for that column.

```
<?xml version="1.0" encoding="UTF-8"?>
<LOADER_MAP>
<SCHEMA>HR</SCHEMA>
<TABLE>EMPLOYEES</TABLE>
<COLUMN field="empId">EMPLOYEE_ID</COLUMN>
<COLUMN field="lastName">LAST_NAME</COLUMN>
<COLUMN field="email">EMAIL</COLUMN>
<COLUMN field="hireDate" format="MM-dd-yyyy">HIRE_DATE</COLUMN>
<COLUMN field="jobId">JOB_ID</COLUMN>
</LOADER_MAP>
```

Note: If all the columns in the target table are used for loading, and if the input data field names in the `IndexedRecord` input object match the column names exactly, then the `loaderMap` file is not needed unless a table columns is a `DATE`. Input fields mapped to `DATE` columns are parsed using the default Java date format. If the input is in a different format, then you must create a `loaderMap` document and use the format attribute to specify the Java date format string to use when parsing input values.

Accessing Table Metadata

Oracle Loader for Hadoop uses table metadata from Oracle Database to control the execution of a loader job. The loader automatically fetches the metadata whenever a JDBC connection can be established. Sometimes it may be impossible for the loader job to access the database. For example, the Hadoop cluster may be on a different network than the database. In this case, the `OraLoaderMetadata` utility program is used to extract table metadata from the database into an XML document. The metadata document is then transferred to the Hadoop cluster. The configuration property `oracle.hadoop.loader.tableMetadataFile` is used to specify the location of the metadata document. When the loader job runs, it accesses this document to discover all necessary metadata information about the target table.

Running the OraLoaderMetadata Utility

To run the OraLoaderMetadata Java utility, add the following jar files to the CLASSPATH variable:

- ${OLH_HOME}/jlib/oraloader.jar
- ${OLH_HOME}/jlib/ojdbc6.jar
- ${OLH_HOME}/jlib/oraclepki.jar

> **Note:** The oraclepki.jar library is required only if you are connecting to the database using credentials stored in an Oracle Wallet.

Then run the following command:

```
java oracle.hadoop.loader.metadata.OraLoaderMetadata
-user <username> -connection_url <connection URL> [-schema <schemaName>]
-table <tableName> -output <output filename>
```

OraLoaderMetadata Parameters

- -user is the Oracle database user name. The user is prompted for the password.
- -connection_url is the connection URL to connect to the Oracle database.
- -schema is the name of the schema containing the target table. If not specified, then the target table is assumed to be in the user schema specified in the connect URL.
- -table is the name of the target table.
- -output is the output file name to store the metadata document.

Invoking OraLoader

OraLoader is a Hadoop job that you execute using the standard Hadoop tools. OraLoader implements the org.apache.hadoop.util.Tool interface and follows the standard Hadoop methods for building MapReduce applications. OraLoader performs the following actions:

1. Reads and verifies input configuration parameters.
2. Retrieves and verifies table and column metadata information for the target table. Metadata is retrieved from the database whenever a JDBC connection can be made. Otherwise, the loader looks for metadata stored in the location specified by the oracle.hadoop.loader.tableMetadataFile property.
3. Prepares internal configuration information for the MapReduce tasks of OraLoader and stores table metadata information and dependent Java libraries in the distributed cache so that they are available to the Map and Reduce tasks throughout the cluster.
4. Submits the MapReduce job to Hadoop.
5. Consolidates reporting information from individual tasks to create a common log file for the job after the Map and Reduce tasks are complete. The log file is written to the job output directory and is named oraloader-report.txt.

OraLoader is invoked from the command line and accepts any of the generic command line options. The following is an example invocation:

```
HADOOP_CLASSPATH="${HADOOP_CLASSPATH}:$OLH_HOME/jlib/*"
```

```
bin/hadoop ${OLH_HOME}/jlib/oraloader.jar oracle.hadoop.loader.OraLoader
-conf MyConf.xml
```

See Also:

- The Apache Hadoop documentation for information about where to find the Hadoop executable and the setting for the HADOOP_CLASSPATH variable.
- Javadoc for the generic options, which is located at the following Apache site: http://hadoop.apache.org/common/docs/r0.20.2/api/org/apache/hadoop/util/GenericOptionsParser.html

Loading Files Into an Oracle Database (Offline Loads Only)

For offline loads, Oracle Loader for Hadoop produces files that must be copied to the database server and loaded into an Oracle database. The following section describes the available offline load method.

Loading From Delimited Text Files Into an Oracle Database

After you copy the delimited text files to the Oracle database server, use the generated control files to invoke SQL*Loader and load the data from the delimited text files into the database. Alternatively, you can use the generated SQL scripts to perform external table loads into the database. See "Delimited Text Output" on page 3-9.

Output Modes During OraLoader Invocation

This section describes the following output options:

- JDBC Output
- Oracle OCI Direct Path Output
- Delimited Text Output
- Oracle Data Pump Output

JDBC Output

JDBC is an output option in online database mode. The output records of the loader job are loaded directly into the target table by map or reduce tasks as part of the OraLoader process. There is no need to execute additional steps to load the data. A JDBC connection between the Hadoop system and the Oracle database is required for this output option.

The JDBC output option uses standard JDBC batching to increase performance and efficiency. If an error occurs during batch execution, for example a constraint is violated, the JDBC driver stops execution at the first error. Thus, if there are 100 rows in a batch and the tenth row causes an error then nine rows are inserted and 91 rows are not. Moreover, the JDBC driver does not provide information to identify which row caused the error. In this case, Oracle Loader does not know the insert status for any of the rows in the batch. It counts all rows in the batch as "in question" and continues loading the next batch. A load report is produced at the end of the job that details the number of batch errors incurred and the number of rows whose insert status is in question. One way to handle this problem is by using a unique key on the data. After the data is loaded, the key can be enabled and used to discover missing key values. The missing rows must be located in the input data and reloaded after it has been determined why they failed to load.

To select the JDBC Output Format, set the following Hadoop property:

```
<property>
  <name>mapreduce.outputformat.class</name>
  <value>oracle.hadoop.loader.lib.output.JDBCOutputFormat</value>
</property>
```

The relevant property for configuring JDBC Output is:

- `oracle.hadoop.loader.jdbc.defaultExecuteBatch` -- controls the size of the batch

Oracle OCI Direct Path Output

The Oracle OCI Direct Path output format is available in online database mode. This output format uses the OCI Direct Path interface to load rows into the target table. Parallel direct path load is possible because each reducer loads into a distinct database partition.

To select the Oracle OCI Direct Path output format, set the following Hadoop property:

```
<property>
  <name>mapreduce.outputformat.class</name>
  <value>oracle.hadoop.loader.lib.output.OCIOutputFormat</value>
</property>
```

The size of the direct path stream buffer can be controlled using the following property:

```
<property>
  <name>oracle.hadoop.loader.output.dirpathBufsize</name>
  <value>131072</value>
  <description>
   This property is used to set the size, in bytes, of the direct path
   stream buffer for OCIOutputFormat.  If needed, values are rounded
   up to the next nearest multiple of 8k.
  </description>
</property>
```

The Oracle OCI Direct Path output format has the following restrictions:

- It is only available on a Linux x86.64 platform.
- The load target table must be partitioned.
- The number of reducers must be greater than zero.
- OCI Direct Path output cannot load a composite interval partitioned table where the subpartition key contains a `CHAR`, `VARCHAR2`, `NCHAR`, or `NVARCHAR2` column. The loader checks for this condition and stops with an error if the target load table meets this condition. Composite interval partitions where the subpartition key does not contain a character type column are supported.

The Oracle OCI Direct Path output format requires the following configuration steps. These steps enable the loader to locate the C shared libraries that implement the output format. These libraries are automatically distributed to compute nodes using the Hadoop Distributed Cache mechanism.

1. Create the environment variable JAVA_LIBRARY_PATH to point to the directory $OLH_HOME/lib. This environment variable is required only on the node where the job is submitted. The $HADOOP_HOME/bin/hadoop command in CDH3 automatically injects this variable value into the Java system property java.library.path when the job is created. For the Apache Hadoop distribution, you must edit the $HADOOP_HOME/bin/hadoop command so that it concatenates new values to an existing value. The Apache hadoop command begins with an empty JAVA_LIBRARY_PATH value and does not import a value from the environment.
2. Add $OLH_HOME/lib to the LD_LIBRARY_PATH variable on the client where the loader job is submitted.

Delimited Text Output

Delimited text is an output option in offline database mode. Comma separated value (CSV) format files, or other delimited text files, are generated by map or reduce tasks. These files are then loaded into the target table using either SQL*Loader or external tables.

To select the Delimited Text Output Format, set the following Hadoop property:

```
<property>
  <name>mapreduce.outputformat.class</name>
  <value>oracle.hadoop.loader.lib.output.DelimitedTextOutputFormat</value>
</property>
```

Each output task generates a delimited text format file and a SQL*Loader control file or SQL script to load the delimited text file into the target table.

Delimited text files have the following template:

```
oraloader-${taskId}-csv-${partitionId}.dat
```

SQL*Loader control file names have the following template:

```
oraloader-${taskId}-csv-${partitionId}.ctl
```

The SQL scripts for loading with external tables have the following template:

```
oraloader-${taskId}-csv-${partitionId}.sql
```

Definitions of the template parameters are as follows:

${taskId}: mapper (reducer) Id

${partitionId}: Partition identifier

The formatting of records and fields in the delimited text file is controlled by the following properties:

- oracle.hadoop.loader.output.fieldTerminator -- a single character to delimit fields
- oracle.hadoop.loader.output.initialFieldEncloser -- when set, fields are always enclosed between this character and the trailingFieldEncloser
- oracle.hadoop.loader.output.trailingFieldEncloser -- when set, fields are always enclosed between initialFieldEncloser and this character
- oracle.hadoop.loader.output.escapeEnclosers -- use to escape embedded trailing field encloser characters

Example 3–1 Sample SQL*Loader Control File

```
LOAD DATA CHARACTERSET AL32UTF8
INFILE 'oraloader-csv-1-0.dat'
BADFILE 'oraloader-csv-1-0.bad'
DISCARDFILE 'oraloader-csv-1-0.dsc'
INTO TABLE "SCOTT"."CSV_PART" PARTITION(10) APPEND
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
(
"ID"      DECIMAL EXTERNAL,
"NAME"    CHAR,
"DOB"     DATE 'SYYYY-MM-DD HH24:MI:SS'
)
```

Oracle Data Pump Output

The Oracle Data Pump output format is available in offline database mode. The loader produces binary format files that can be loaded into the target table using an external table and the ORACLE_DATAPUMP access driver. The output files must be copied from the HDFS file system to a local file system that is accessible to Oracle Database.

To select the Oracle Data Pump output format, set the following Hadoop property:

```
<property>
  <name>mapreduce.outputformat.class</name>
  <value>oracle.hadoop.loader.lib.output.DataPumpOutputFormat</value>
</property>
```

Oracle Data Pump output file names have the following template:

```
oraloader-${taskId}-dp-${partitionId}.dat
```

Oracle Loader for Hadoop also produces a SQL file that contains commands to perform the following tasks:

1. Create an external table definition using the ORACLE_DATAPUMP access driver. The binary format Data Pump output files are listed in the LOCATION clause of the external table.
2. Create a directory object that is used by the external table. This command must be uncommented before it can be used. To specify the directory name that is produced in the SQL file, set the following property:

   ```
   <property>
     <name>oracle.hadoop.loader.extTabDirectoryName</name>
     <value>OLH_EXTTAB_DIR</value>
     <description>
      The name of the Oracle directory object for the external table's
      LOCATION data files. This property applies only to the CSV and
      DataPump output formats.
     </description>
   </property>
   ```

3. Insert the rows from the external table into the target table. This command must be uncommented before it can be used.

See Also:

- *Oracle Database Administrator's Guide* for more information about creating and managing external tables
- *Oracle Database Utilities* for more information about the `ORACLE_DATAPUMP` access driver

Balancing Loads When Loading Data into Partitioned Tables

To balance loads across reducers when data is loaded into a partitioned database table, use the sampling feature of Oracle Loader for Hadoop.

The execution time of a reducer is usually proportional to the number of records it processes - the more records, the longer the execution time. When the sampling feature is disabled, all records from a given database partition are sent to one reducer. This can result in unbalanced reducer loads because some database partitions may have more records than others. Because the execution time of a Hadoop job is the execution time of its slowest reducer, unbalanced reducer loads can slow down the entire job.

Although hashing records uniformly across reducers can generate balanced reducer loads, it does not necessarily group records by database partition before inserting them into the database.

The sampling feature of Oracle Loader for Hadoop generates an efficient MapReduce partitioning scheme that groups records by database partition while also balancing reducer load.

Using the Sampling Feature

To enable the sampling feature, set the configuration property `oracle.hadoop.loader.sampler.enableSampling` to `true`.

Even if the `enableSampling` property is set to `true`, the loader automatically disables the sampling feature if sampling is not necessary or if the loader determines that a good sample cannot be made. For example, sampling is automatically disabled if the table is not partitioned, or the number of reducer tasks is less than two, or there is too little input data to compute a good load balancing. In those cases, the loader returns an informational message.

> **Note:** The sampler is multi-threaded and each sampler thread instantiates its own copy of the supplied `InputFormat` class. Any new `InputFormat` implementations provided to Oracle Loader for Hadoop should ensure that data structures that are static and mutable are synchronized for multiple thread access.
>
> It is possible for the sampler to return an out-of-memory error on the client node where the loader job is submitted. This can occur when the input splits returned by the `InputFormat` do not fit in memory.
>
> The following are possible solutions to this problem:
>
> - Increase the heap size of the JVM where the job is submitted.
> - Adjust the following properties:
>
>   ```
>   oracle.hadoop.loader.sampler.hintMaxSplitSize
>   oracle.hadoop.loader.sampler.hintNumMapTasks
>   ```
>
>   See "OraLoader for Hadoop Configuration Properties" on page 3-23 for descriptions of these properties.

Tuning Load Balancing and Sampling Behavior

Oracle Loader for Hadoop provides properties that you can use to tune load balancing and sampling behavior. These properties are summarized in Table 3–3 on page 3-13.

Properties to Tune Load Balancing

The goal of load balancing is to generate a MapReduce partitioning scheme that assigns approximately the same amount of work to all reducers. This scheme is used in the partitioning step during Oracle Loader for Hadoop job execution.

Two properties control the quality of load balancing: `maxLoadFactor` and `loadCI`. The sampler uses the expected reducer load factor to evaluate the quality of its partitioning scheme. Load factor is a metric that indicates how much a reducer's load deviates from a perfectly balanced reducer load. A load factor of one indicates a perfectly balanced load (no overload).

Small load factors indicate better load balancing. The property `maxLoadFactor` denotes a load factor of (1+`maxLoadFactor`). The `maxLoadFactor` default of 0.05 indicates that no reducer is ever overloaded by more than 5%. The sampler guarantees this `maxLoadFactor` with a statistical confidence level of `loadCI`. The default value of `loadCI` is 0.95 which means that any reducer's load factor exceeds `maxLoadFactor` in only 5% of the cases.

There is a trade-off between the execution time of the sampler and the quality of load balancing. Lower values of `maxLoadFactor` and higher values of `loadCI` result in more balanced reducer loads at the expense of longer sampling times. The default values of `maxLoadFactor=0.05` and `loadCI=0.95` provide a good trade-off between load balancing quality and execution time.

Properties to Tune Sampling Behavior

By default, the sampler runs until it collects just enough samples to generate a partitioning scheme that satisfies the `maxLoadFactor` and `loadCI` criteria.

However, you can limit the sampler's running time by using the `maxSamplesPct` property which specifies the maximum number of records that the sampler should sample before stopping.

Does Oracle Loader for Hadoop Always Use the Sampler's Partitioning Scheme?

Oracle Loader for Hadoop only uses the generated partitioning scheme if sampling is successful. A sampling is successful if it generates a partitioning scheme with maximum reducer load factor of (1+ maxLoadFactor) guaranteed at a statistical confidence level of loadCI. The default values of maxLoadFactor, loadCI, and maxSamplesPct allow the sampler to successfully generate high-quality partitioning schemes for a variety of different input data distributions. However, in some cases the sampler might be unsuccessful in generating a partitioning scheme that satisfies these constraints (for example, the constraints are too rigid or the number of samples it requires exceed the user-specified maximum of maxSamplesPct). In such cases, Oracle Loader for Hadoop prints a log message saying that there were not enough samples, and defaults to partitioning records by database partition and provides no load balancing guarantees (as described in "Tuning Load Balancing and Sampling Behavior" on page 3-12).

An alternative approach would be to reset the configuration properties to less rigid values. You can do this either by increasing maxSamplesPct, or by decreasing maxLoadFactor or loadCI, or both.

What Happens When a Sampling Feature Property Has an Invalid Value?

If any configuration properties of the sampling feature are set to values outside the accepted range, an exception is **not** returned. Instead, the sampler prints a warning message, resets the property to its default value, and continues executing.

Primary Configuration Properties for the Load Balancing Feature

Table 3–3 describes the primary properties available to tune sampling behavior. See "OraLoader for Hadoop Configuration Properties" on page 3-23 for a complete list of properties.

Table 3–3 Configuration Properties of the Oracle Loader for Hadoop Sampling Feature

Information Type	Values
Name	oracle.hadoop.loader.sampler.maxSamplesPct
Type	Float
Default	0.01
Accepted Range	[0, 1] A value of <=0 disables this property.
Description	The maximum sample size as a percentage of the number of records in the input data. A value of 0.05 indicates that the sampler never samples more than 5% of the total number of records. The sampler may collect fewer samples than this amount.
-	-
Name	oracle.hadoop.loader.sampler.maxLoadFactor
Type	Float
Default	0.05
Accepted Range	>= 0 A value of <=0 resets the property to the default.
Description	Maximum acceptable load factor for reducer work load.

Table 3–3 (Cont.) Configuration Properties of the Oracle Loader for Hadoop Sampling

Information Type	Values
-	-
Name	`oracle.hadoop.loader.sampler.loadCI`
Type	Float
Default	0.95
Accepted Range	>= 0.5 and < 1 Recommended values are >= 0.9 A value of < 0.5 resets the property to the default.
Description	The statistical confidence level for the maximum reducer load factor. Commonly used values are 0.95 and 0.99.

OraLoader Configuration Properties

`OraLoader` uses the standard method in Hadoop for specifying configuration properties. They can be specified in a configuration file or using the `-D property=value option` to `GenericOptionsParser` and `ToolRunner`.

Table 3–4 and Table 3–5 provide brief descriptions of the primary Oracle Loader for Hadoop configuration properties. For a complete list and detailed descriptions of all configuration properties, see the `oraloader-conf.xml` document in "OraLoader for Hadoop Configuration Properties" on page 3-23.

Table 3–4 Primary Job Configuration Properties for Oracle Loader for Hadoop

Information Type	Values
Name	`oracle.hadoop.loader.jobName`
Type	String
Default	OraLoader
Description	A Hadoop job name for this Oracle loader job. Used as input for the `Job.setJobName()` method.
-	-
Name	`oracle.hadoop.loader.targetTable`
Type	String
Default	Not defined
Description	A schema qualified name for the table to be loaded. Use this option to indicate that all columns of the table are to be loaded and that the names of the input fields match the column names. This property takes precedence over the `oracle.hadoop.loader.loaderMapFile` property.
-	-
Name	`oracle.hadoop.loader.loaderMapFile`
Type	String
Default	Not defined
Description	Path to the loader map file.
-	-
Name	`oracle.hadoop.loader.tableMetadataFile`

Table 3–4 (Cont.) Primary Job Configuration Properties for Oracle Loader for Hadoop

Information Type	Values
Type	String
Default	Not defined
Description	Path to the target table metadata file. Use this option when running in disconnected mode. The table metadata file is created by running the `OraLoaderMetadata` utility.
-	-
Name	`oracle.hadoop.loader.olhcachePath`
Type	String
Default	${mapred.output.dir}/.../olhcache
Description	Path to a directory where Oracle Loader for Hadoop can create files that are loaded into the `DistributedCache`. In distributed mode, the value must be an HDFS path.
-	-
Name	`oracle.hadoop.loader.extTabDirectoryName`
Type	String
Default	`OLH_EXTTAB_DIR`
Description	The name of the Oracle directory object for the external table's `LOCATION` data files. This property applies only to the Delimited Text and Data Pump output formats.
-	-
Name	`oracle.hadoop.loader.sampler.enableSampling`
Type	Boolean
Default	true
Description	Indicates whether the sampling feature is enabled.
-	-
Name	`oracle.hadoop.loader.enableSorting`
Type	Boolean
Default	true
Description	Indicates whether output records within each reducer group should be sorted by the primary key for the table.
-	-
Name	`oracle.hadoop.loader.connection.url`
Type	String
Default	Not defined
Description	Specifies the URL of the database connection string. This property takes precedence over all other connection properties. If Oracle wallet is configured as an external password store, then the property value must start with the following driver prefix: `jdbc:oracle:thin:@` and the `db_connect_string` must exactly match the credential defined in the wallet.
-	-
Name	`oracle.hadoop.loader.connection.user`
Type	String

Table 3–4 (Cont.) Primary Job Configuration Properties for Oracle Loader for Hadoop

Information Type	Values
Default	Not defined
Description	Name for the database login.
-	-
Name	`oracle.hadoop.loader.connection.password`
Type	String
Default	Not defined
Description	Password for the connecting user.
-	-
Name	`oracle.hadoop.loader.connection.wallet_location`
Type	String
Default	Not defined
Description	File path to an Oracle wallet where the connection information is stored. This property is used only for JDBC connections. For JDBC output format, when using Oracle Wallet as an external password store, set the following two properties: ■ `oracle.hadoop.loader.connection.wallet_location` ■ `oracle.hadoop.loader.connection.url` Or, set the following three properties: ■ `oracle.hadoop.loader.connection.wallet_location` ■ `oracle.hadoop.loader.connection.tnsEntryName` ■ `oracle.hadoop.loader.connection.tns_admin` For the OCI output format, set the `oracle.hadoop.loader.connection.tns_admin` property to indicate wallet location. Note that JDBC connections are always made for online loads, even when the OCI Direct Path output format is specified. The same wallet can be used for both connection types.
-	-
Name	`oracle.hadoop.loader.connection.tnsEntryName`
Type	String
Default	Not defined
Description	Specifies a TNS entry name defined in the `tnsnames.ora` file. This property is used together with the `oracle.hadoop.loader.connection.tns_admin` property.
-	-
Name	`oracle.hadoop.loader.connection.tns_admin`
Type	String
Default	Not defined

Table 3–4 (Cont.) Primary Job Configuration Properties for Oracle Loader for Hadoop

Information Type	Values
Description	File path to a directory containing SQL*Net configuration files such as `sqlnet.ora` and `tnsnames.ora`. If this value is not set, then the value (if any) of the environment variable `TNS_ADMIN`, is used. Define this property in order to use TNS entry names in database connect strings. This property must be defined when using an Oracle Wallet with OCI connections.
-	-
Name	`oracle.hadoop.loader.connection.defaultExecuteBatch`
Type	Integer
Default	100
Description	Applicable only for the JDBC and OCI Direct Path output formats. The default value for the number of records to be inserted in a batch for each trip to the database. Specify a value greater than 1 to override the default value. If the specified value is less than 1, then this property assumes the default value. Although the maximum value is unlimited, using very large batch sizes is not recommended because it results in a large memory footprint without much increase in performance.
-	-
Name	`oracle.hadoop.loader.connection.sessionTimeZone`
Type	String
Default	LOCAL
Description	This property is used to alter the session time zone for database connections. Valid values are as follows: [+|-] hh:mm - hours and minutes before or after UTC LOCAL - the default time zone of the JVM time_zone_region - a valid time zone region This property also determines the default time zone used when parsing input data that is loaded into the following database column types: `TIMESTAMP`, `TIMESTAMP WITH TIME ZONE` and `TIMESTAMP WITH LOCAL TIME ZONE`.
-	-
Name	`oracle.hadoop.loader.output.dirpathBufsize`
Type	Integer
Default	131072
Description	This property is used to set the size, in bytes, of the direct path stream buffer for `OCIOutputFormat`.If needed, values are rounded up to the next nearest multiple of 8 KB.
-	-
Name	`oracle.hadoop.loader.output.fieldTerminator`
Type	String
Default	, (comma)
Description	A single character to delimit fields for `DelimitedTextOutputFormat`. Alternate representation: \uHHHH (where HHHH is the character's UTF-16 encoding).

Table 3–4 (Cont.) Primary Job Configuration Properties for Oracle Loader for Hadoop

Information Type	Values
-	-
Name	`oracle.hadoop.loader.output.initialFieldEncloser`
Type	String
Default	None
Description	When this value is set, fields are always enclosed between the specified character and `${oracle.hadoop.loader.output.trailingFieldEncloser}`. If this value is set, it must be either a single character, or \uHHHH (where HHHH is the character's UTF-16 encoding). `${oracle.hadoop.loader.output.initialFieldEncloser}` and `${oracle.hadoop.loader.output.trailingFieldEncloser}` must be either both not set, or both set. A zero length value means no enclosers (default value). Use this when some field may contain the `fieldTerminator`. If some field may also contain the `trailingFieldEncloser`, then the `escapeEnclosers` property should be set to `true`.
-	-
Name	`oracle.hadoop.loader.output.trailingFieldEncloser`
Type	String
Default	None
Description	When this value is set, fields are always enclosed between `${oracle.hadoop.loader.output.initialFieldEncloser}` and the specified character for this property. If this value is set, it must be either a single character, or \uHHHH (where HHHH is the character's UTF-16 encoding). `${oracle.hadoop.loader.output.initialFieldEncloser}` and `${oracle.hadoop.loader.output.trailingFieldEncloser}` must be either both not set, or both set. A zero length value means no enclosers (default value). Use this when some field may contain the `fieldTerminator`. If some field may also contain the `trailingFieldEncloser`, then the `escapeEnclosers` property should be set to `true`.
-	-
Name	`oracle.hadoop.loader.output.escapeEnclosers`
Type	Boolean
Default	false
Description	When this is set to `true` and both initial and trailing field enclosers are set, fields are scanned and embedded trailing encloser characters are escaped. Use this option when some of the field values may contain the trailing encloser character.
-	-
Name	`oracle.hadoop.loader.input.fieldTerminator`
Type	String
Default	, (comma)

Table 3–4 (Cont.) Primary Job Configuration Properties for Oracle Loader for Hadoop

Information Type	Values
Description	A single character to delimit fields for `DelimitedTextInputFormat`. Alternate representation: \uHHHH (where HHHH is the character's UTF-16 encoding).
-	-
Name	`oracle.hadoop.loader.input.initialFieldEncloser`
Type	String
Default	None
Description	When this value is set, fields are allowed to be enclosed between the specified character and `${oracle.hadoop.loader.input.trailingFieldEncloser}`. If this value is set, it must be either a single character, or \uHHHH (where HHHH is the character's UTF-16 encoding). `${oracle.hadoop.loader.input.initialFieldEncloser}` and `${oracle.hadoop.loader.input.trailingFieldEncloser}` must be either both not set, or both set. A zero length value means no enclosers (default value).
-	-
Name	`oracle.hadoop.loader.input.trailingFieldEncloser`
Type	String
Default	None
Description	When this value is set, fields are allowed to be enclosed between `${oracle.hadoop.loader.input.initialFieldEncloser}` and the specified character. f this value is set, it must be either a single character, or \uHHHH (where HHHH is the character's UTF-16 encoding). `${oracle.hadoop.loader.input.initialFieldEncloser}` and `${oracle.hadoop.loader.input.trailingFieldEncloser}` must be either both not set, or both set. A zero length value means no enclosers (default value).
-	-
Name	`oracle.hadoop.loader.input.fieldNames`
Type	Comma-separated list of strings
Default	F0,F1,F2,...
Description	Names to assign to input fields. The names are used to create the Avro schema for the record. The strings must be valid JSON name strings.

Table 3–5 General Properties for Oracle Loader for Hadoop

Property Name	Description
`mapreduce.inputformat.class`	Name of the class implementing `InputFormat`.

Table 3–5 (Cont.) General Properties for Oracle Loader for Hadoop

Property Name	Description
mapreduce.outputformat.class	Output options supported by Oracle Loader for Hadoop. The values can be: ■ oracle.hadoop.loader.lib.output.DelimitedTextOutputFormat Writes data records to delimited text format files such as comma separated value (CSV) format files. ■ oracle.hadoop.loader.lib.output.JDBCOutputFormat Inserts data records into the target table using JDBC ■ oracle.hadoop.loader.lib.output.OCIOutputFormat Inserts rows into the target table using the Oracle OCI Direct Path interface. ■ oracle.hadoop.loader.lib.output.DataPumpOutputFormat Writes rows into binary format files that can be loaded into the target table using an external table.

Example of Using Oracle Loader for Hadoop

The example shown in this section uses Oracle Loader for Hadoop in the online database mode using JDBC. It involves the following steps:

1. Create a table in the database. This example uses the HR.EMPLOYEES table available as part of the HR sample schema in Oracle Database.
2. Implement an InputFormat class similar to the examples in the oracle.hadoop.loader.examples package.
3. Set the configuration properties. The MyLoaderMap.xml document contains the mapping of input data fields to columns in the HR.EMPLOYEES table, as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<LOADER_MAP>
<SCHEMA>HR</SCHEMA>
<TABLE>EMPLOYEES</TABLE>
<COLUMN field="empId">EMPLOYEE_ID</COLUMN>
<COLUMN field="lastName">LAST_NAME</COLUMN>
<COLUMN field="email">EMAIL</COLUMN>
<COLUMN field="hireDate" format="MM-dd-yyyy">HIRE_DATE</COLUMN>
<COLUMN field="jobId">JOB_ID</COLUMN>
</LOADER_MAP>
```

The configuration properties in MyConf.xml are as follows:

```
<configuration>
  <property>
    <name>mapreduce.inputformat.class</name>
    <value><full_class_name>.MyInputFormat</value>
    <description> Name of the class implementing InputFormat </description>
  </property>

  <property>
```

```
    <name>mapreduce.outputformat.class</name>
    <value>oracle.hadoop.loader.lib.output.JDBCOutputFormat</value>
    <description> Output mode after the loader job executes on Hadoop
</description>
  </property>

  <property>
    <name>oracle.hadoop.loader.loaderMapFile</name>
    <value>MyLoaderMap.xml</value>
    <description> The loaderMap file specifying the mapping of input data
     fields to the table columns </description>
  </property>

 <property>
   <name>oracle.hadoop.loader.connection.user</name>
   <value>HR</value>
   <description> Name of the user connecting to the database</description>
 </property>

<property>
  <name>oracle.hadoop.loader.connection.password</name>
  <value>[HR password]</value>
  <description>Password of the user connecting to the database</description>
</property>

 <property>
   <name>oracle.hadoop.loader.connection.url</name>
   <value>jdbc:oracle:thin:@//example.com:1521/serviceName</value>
   <description> Database connection string </description>
 </property>
</configuration>
```

4. Invoke `OraLoader`.

```
bin/hadoop jar oraloader.jar oracle.hadoop.loader.OraLoader -libjars
avro-1.4.1.jar, MyInputFormat.jar -conf MyConf.xml
-fs [<local|namenode:port>]
-jt [<local|jobtracker:port>]
```

Target Table Characteristics

Oracle Loader for Hadoop supports loads into a single table, which is referred to as the target table. The target table must exist in the Oracle database. It can contain data or it can be empty.

Supported Data Types

Oracle Loader for Hadoop supports the following Oracle built-in data types:

- VARCHAR2
- CHAR
- NVARCHAR2
- NCHAR
- NUMBER
- FLOAT
- RAW

- `BINARY_FLOAT`
- `BINARY_DOUBLE`
- `DATE`
- `TIMESTAMP`
- `TIMESTAMP WITH TIMEZONE`
- `TIMESTAMP WITH LOCAL TIME ZONE`
- `INTERVAL YEAR TO MONTH`
- `INTERVAL DAY TO SECOND`

The target table can contain columns with unsupported data types, but these columns must be nullable, or otherwise set to a value.

Supported Partitioning Strategies

Oracle Loader for Hadoop supports the following single and composite level partitioning strategies:

- Range
- List
- Hash
- Interval
- Range-Range
- Range-Hash
- Range-List
- List-Range
- List-Hash
- List-List
- Hash-Range
- Hash-Hash
- Hash-List
- Interval-Range
- Interval-Hash
- Interval-List

Oracle Loader for Hadoop does not support tables with reference or virtual column based partitioning.

Oracle Loader for Hadoop supports NOT NULL constraints during loads. No other constraints are enforced.

Loader Map XML Schema Definition

This is the XML schema definition (XSD) for the loader map that specifies the columns to be loaded into the target table:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema elementFormDefault="qualified" attributeFormDefault="unqualified"
```

```
         xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:attributeGroup name="columnAttrs">
    <xs:annotation>
      <xs:documentation>Column attributes define how to map input fields to the
                        database column. field - is the name of the field in the
                        IndexedRecord input object. The field name need not be
                        unique. This means that the same input field can map to
                        different columns in the database table. format - is a
                        format string for interpreting the input. For example,
                        if the field is a date then the format is a date format
                        string suitable for interpreting dates</xs:documentation>
    </xs:annotation>
    <xs:attribute name="field" type="xs:token" use="optional"/>
    <xs:attribute name="format" type="xs:token" use="optional"/>
  </xs:attributeGroup>
  <xs:simpleType name="TOKEN_T">
    <xs:restriction base="xs:token">
      <xs:minLength value="1"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:element name="LOADER_MAP">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="SCHEMA" type="TOKEN_T" minOccurs="0"/>
        <xs:element name="TABLE" type="TOKEN_T" nillable="false"/>
        <xs:element name="COLUMN" maxOccurs="unbounded" minOccurs="0">
          <xs:annotation>
            <xs:documentation>specifies the database column name that will be
                              loaded. Each column name must be unique.
            </xs:documentation>
          </xs:annotation>
          <xs:complexType>
            <xs:simpleContent>
              <xs:extension base="TOKEN_T">
                <xs:attributeGroup ref="columnAttrs"/>
              </xs:extension>
            </xs:simpleContent>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
 </xs:schema>
```

OraLoader for Hadoop Configuration Properties

This is the `oraloader-conf.xml` document, which describes the configuration properties for Oracle Loader for Hadoop:

```
<?xml version="1.0"?>
<!--
 Copyright (c) 2011, Oracle and/or its affiliates. All rights reserved.

   NAME
     oraloader-conf.xml

   DESCRIPTION
     Config properties for OLH.

     This file is loaded as the very first conf resource.
     Properties without default values are commented out.
-->
<configuration>
  <property>
    <name>oracle.hadoop.loader.libjars</name>
    <value>${oracle.hadoop.loader.olh_home}/jlib/ojdbc6.jar,
${oracle.hadoop.loader.olh_home}/jlib/orai18n.jar,
${oracle.hadoop.loader.olh_home}/jlib/orai18n-utility.jar,
${oracle.hadoop.loader.olh_home}/jlib/orai18n-mapping.jar,
${oracle.hadoop.loader.olh_home}/jlib/orai18n-collation.jar,
${oracle.hadoop.loader.olh_home}/jlib/oraclepki.jar,
${oracle.hadoop.loader.olh_home}/jlib/osdt_cert.jar,
${oracle.hadoop.loader.olh_home}/jlib/osdt_core.jar,
${oracle.hadoop.loader.olh_home}/jlib/commons-math-2.2.jar,
${oracle.hadoop.loader.olh_home}/jlib/jackson-core-asl-1.5.2.jar,
${oracle.hadoop.loader.olh_home}/jlib/jackson-mapper-asl-1.5.2.jar,
${oracle.hadoop.loader.olh_home}/jlib/avro-1.5.4.jar,
${oracle.hadoop.loader.olh_home}/jlib/avro-mapred-1.5.4.jar</value>
    <description>Comma separated list of library jar files. These jars get
                 injected into the command-line arguments under the
                 GenericOptionsParser's "-libjars" option. When a "-libjars"
                 option is used as a command-line argument, then this list of
                 jars is prepended to the list following "-libjars". Users can
                 distribute their application jars using this property in place
                 of, or in combination with, the "-libjars" option.</description>
  </property>

  <property>
    <name>oracle.hadoop.loader.sharedLibs</name>
    <value>${oracle.hadoop.loader.olh_home}/lib/libolh11.so,
${oracle.hadoop.loader.olh_home}/lib/libclntsh.so.11.1,
${oracle.hadoop.loader.olh_home}/lib/libnnz11.so,
${oracle.hadoop.loader.olh_home}/lib/libociei.so</value>
  </property>

  <property>
    <name>oracle.hadoop.loader.olh_home</name>
    <value/>
    <description>
      A path to the OLH_HOME on the node where the OraLoader job
      is initiated. OraLoader uses this path to locate required libraries.
      If this property is not set, OraLoader will use the value in the environment
      variable OLH_HOME.
    </description>
  </property>

  <property>
    <name>oracle.hadoop.loader.jobName</name>
    <value>OraLoader</value>
    <description>
      Hadoop job name for this Oracle loader job. Used as input for
      the Job.setJobName() method.
    </description>
  </property>

  <property>
    <name>oracle.hadoop.loader.targetTable</name>
    <value/>
    <description>
      A schema qualified name for the table to be loaded. Use this
      property to indicate that all columns of the table will be
      loaded and that the names of the input fields match the
      column names. This property takes precedence over the
      oracle.hadoop.loader.loaderMapFile property. The default
      value is null.
    </description>
  </property>

  <property>
    <name>oracle.hadoop.loader.loaderMapFile</name>
    <value/>
    <description>
      Path to the loader map file. Use a file:// schema to indicate a local file.
    </description>
  </property>

  <property>
    <name>oracle.hadoop.loader.tableMetadataFile</name>
    <value/>
    <description>
      Path to the target table metadata file. Use this property when
      running in disconnected mode. The table metadata file is
      created by running the OraLoaderMetadata utility.
      Use a file:// schema to indicate a local file.
    </description>
  </property>

  <property>
    <name>oracle.hadoop.loader.olhcachePath</name>
    <value>${mapred.output.dir}/../olhcache</value>
    <description>
      Path to a directory where Oracle Loader for Hadoop can create
      files that will be loaded into the DistributedCache.
      Unique file names are generated every time; one may want to empty it,
      or it will grow bigger and bigger if jobs are run
      using the same olhcache directory.

      The default value is a directory called 'olhcache' in the parent directory
      of the job's output directory (i.e. ${mapred.output.dir}).

      In distributed mode, the value must be a hdfs path
      (see javaDoc for org.apache.hadoop.filecache.DistributedCache).
    </description>
  </property>

  <property>
    <name>oracle.hadoop.loader.loadByPartition</name>
    <value>true</value>
    <description>
      Instructs the output format to perform a partition-aware load.
      For DelimitedText output format, this option controls whether the
      keyword "PARTITION" appears in the generated .ctl file(s).
    </description>
  </property>

  <property>
    <name>oracle.hadoop.loader.extTabDirectoryName</name>
    <value>OLH_EXTTAB_DIR</value>
    <description>
      The name of the Oracle directory object for the external table's
      LOCATION data files. This property applies only to the DelimitedText
      and DataPump output formats.
    </description>
  </property>

  <property>
    <name>oracle.hadoop.loader.sampler.enableSampling</name>
    <value>true</value>
    <description>
      Indicates whether the sampling feature is enabled.
      Set the value to false to disable this feature.
    </description>
  </property>

  <property>
    <name>oracle.hadoop.loader.enableSorting</name>
    <value>true</value>
    <description>
      Indicates whether output records within each reducer group
      should be sorted by the primary key for the table.
    </description>
  </property>

  <property>
     <name>oracle.hadoop.loader.configuredCounters</name>
     <value>MAPPER,OUTPUT</value>
     <description>
       Turns ON Oracle Loader for Hadoop counters by category. The value is a
       comma separated list of zero or more of the following keywords:
       MAPPER, REDUCER, OUTPUT, and SAMPLER.

       Note that the input error counters (displayed in the
       "map phase counters" section of the final report) are always on,
       regardless of the presence of the keyword MAPPER in this list.

       Newer release of Hadoop (0.20.203, cdh3u3) impose a hard limit on the
       total number of counters a job can use (see property
       mapreduce.job.counters.limit in mapred-site.xml). Note that this
       limit cannot be changed on a per-job basis, and the cluster needs
       to be restarted after the property has beed updated on all nodes.

       In order to turn off all the Oracle Loader for Hadoop specific counters,
       set this property's value to an empty list using either:

       -D oracle.hadoop.loader.configuredCounters=

       or

       <property>
         <name>oracle.hadoop.loader.configuredCounters</name>
         <value>,</value>
       </property>

     </description>
  </property>

  <!-- CONNECTION properties -->

  <property>
    <name>oracle.hadoop.loader.connection.url</name>
    <value/>
    <description>
      Specifies the URL of the database connection string. This property
      takes precedence and overrides all other connection properties.

      If Oracle Wallet is configured as an external password store,
      the property value must start with the driver prefix: jdbc:oracle:thin:@
      and the db_connect_string must exactly match the credential defined in the
      wallet.

        Example 1: ( using oracle net syntax)
        jdbc:oracle:thin:@(DESCRIPTION=(ADDRESS_LIST=
            (ADDRESS=(PROTOCOL=TCP)(HOST=myhost)(PORT=1521)))
                     (CONNECT_DATA=(SERVICE_NAME=my_db_service_name)))

        Example 2: ( using TNS entry)
          jdbc:oracle:thin:@myTNS

        - Also see documentation for
          oracle.hadoop.loader.connection.wallet_location

      If Oracle Wallet is NOT used, then set the following conf properties:
      oracle.hadoop.loader.connection.url

        Examples of connection URL styles:
          thin-style:
            jdbc:oracle:thin:@//myhost:1521/my_db_service_name
            jdbc:oracle:thin:user/password@//myhost:1521/my_db_service_name

          Oracle Net:
            jdbc:oracle:thin:@(DESCRIPTION=(ADDRESS_LIST=
                (ADDRESS=(PROTOCOL=TCP)(HOST=myhost)(PORT=1521)))
                         (CONNECT_DATA=(SERVICE_NAME=my_db_service_name)))

          TNSEntry Name:
            jdbc:oracle:thin:@myTNSEntryName

     AND
     oracle.hadoop.loader.connection.user
     oracle.hadoop.loader.connection.password

     If OCIOutputFormat is configured,and Oracle Wallet is not used,then
     username and password must be specified in these separate properties.

    </description>
  </property>

   <property>
     <name>oracle.hadoop.loader.connection.user</name>
     <value/>
     <description>Name for the database login.</description>
   </property>

  <property>
    <name>oracle.hadoop.loader.connection.password</name>
    <value/>
    <description>Password for the connecting user.</description>
  </property>

  <property>
```

```
  <name>oracle.hadoop.loader.connection.wallet_location</name>
  <value/>
  <description>File path to an Oracle wallet where the connection information
   is stored. This property is used only for JDBC connections. For JDBC output
   format, when using Oracle Wallet as an external password store, set the
   following two properties:

   - oracle.hadoop.loader.connection.wallet_location
   - oracle.hadoop.loader.connection.url

   Or, set the following three properties:

   - oracle.hadoop.loader.connection.wallet_location
   - oracle.hadoop.loader.connection.tnsEntryName
   - oracle.hadoop.loader.connection.tns_admin

   For the OCI output format, set the
   oracle.hadoop.loader.connection.tns_admin property to indicate
   wallet location.

   Note that JDBC connections are always made for online loads, even when
   the OCI Direct Path output format is specified. The same wallet can be
   used for both connection types.
  </description>
</property>

<property>
  <name>oracle.hadoop.loader.connection.tnsEntryName</name>
  <value/>
  <description>Specifies a TNS entry name defined in the tnsnames.ora file.
  This property is used together with the
  oracle.hadoop.loader.connection.tns_admin property.
  </description>
</property>

<property>
  <name>oracle.hadoop.loader.connection.tns_admin</name>
  <value/>
  <description>File path to a directory containing
    SQL*Net configuration files like sqlnet.ora and tnsnames.ora.
    If this property is not set, the value of the environment
    variable TNS_ADMIN will be used. Define this property in order
    to use TNS entry names in database connect strings.
    This property must be defined when using an Oracle Wallet with OCI
    connections.
  </description>
</property>

<property>
  <name>oracle.hadoop.loader.connection.defaultExecuteBatch</name>
  <value>100</value>
  <description>
     Applicable only for JDBC and OCI output formats. The default
     value for the number of records to be inserted in a batch for
     each trip to the database. Specify a value >= 1 to
     override the default value. If the specified value is less than 1,
     this property assumes the default value. Though the maximum
     value is unlimited, using very large batch sizes is not
     recommended, as it results in a large memory footprint without
     much increase in performance.
   </description>
</property>

<property>
  <name>oracle.hadoop.loader.connection.sessionTimezone</name>
  <value>LOCAL</value>
  <description>
    This property is used to alter the session time zone for
    database connections.  Valid values are:

      [+|-] hh:mm      - hours and minutes before or after UTC
      LOCAL            - the default timezone of the JVM
      time_zone_region - a valid time zone region

    This property also determines the default timezone when parsing
    input data that will be loaded to database column types:
    TIMESTAMP, TIMESTAMP WITH TIME ZONE and TIMESTAMP WITH LOCAL TIME ZONE
  </description>
</property>

<!-- properties for OCIOutputFormat -->
<property>
  <name>oracle.hadoop.loader.output.dirpathBufsize</name>
  <value>131072</value>
  <description>
    This property is used to set the size, in bytes, of the direct path stream
    buffer for OCIOutputFormat.  If needed, values are rounded up to the next
    nearest multiple of 8k.
  </description>
</property>

<property>
  <name>oracle.hadoop.loader.compressionFactors</name>
  <value>BASIC=5.0,OLTP=5.0,QUERY_LOW=10.0,QUERY_HIGH=10.0,
    ARCHIVE_ LOW=10.0,ARCHIVE_HIGH=10.0</value>
  <description>
    This property is used to define the compression factor for different types
    of compression. The format is a comma separated list of name=value pairs
    where name is one of BASIC, OLTP, QUERY_LOW, QUERY_HIGH, ARCHIVE_LOW, or
    ARCHIVE_HIGH.  Value is a decimal number.
  </description>
</property>

<!-- properties for DelimitedTextOutputFormat -->
  <property>
    <name>oracle.hadoop.loader.output.fieldTerminator</name>
    <value>,</value>
    <description>
      A single character to delimit fields for DelimitedTextOutputFormat.
      Alternate representation: \uHHHH (where HHHH is the character's UTF-16
      encoding).
    </description>
  </property>

  <property>
    <name>oracle.hadoop.loader.output.initialFieldEncloser</name>
    <value></value>
    <description>
      When this value is set, fields are always enclosed between the
      specified character and
      ${oracle.hadoop.loader.output.trailingFieldEncloser}.

      If this value is set, it must be either a single character, or \uHHHH
      (where HHHH is the character's UTF-16 encoding).

      ${oracle.hadoop.loader.output.initialFieldEncloser} and
      ${oracle.hadoop.loader.output.trailingFieldEncloser} must be either
      both not set, or both set.
      A zero length value means no enclosers (default value).

      Use this when some field may contain the fieldTerminator.
      If some field may also contain the trailingFieldEncloser, then
      the escapeEnclosers property should be set to true.
    </description>
  </property>

  <property>
    <name>oracle.hadoop.loader.output.trailingFieldEncloser</name>
    <value></value>
    <description>
      When this value is set, fields are always enclosed between
      ${oracle.hadoop.loader.output.initialFieldEncloser} and the
      specified character for this property.

      If this value is set, it must be either a single character, or \uHHHH
      (where HHHH is the character's UTF-16 encoding).

      ${oracle.hadoop.loader.output.initialFieldEncloser} and
      ${oracle.hadoop.loader.output.trailingFieldEncloser} must be either
      both not set, or both set.
      A zero length value means no enclosers (default value).

      Use this when some field may contain the fieldTerminator.
      If some field may also contain the trailingFieldEncloser, then
      the escapeEnclosers property should be set to true.
    </description>
  </property>

  <property>
    <name>oracle.hadoop.loader.output.escapeEnclosers</name>
    <value>false</value>
    <description>
      When this is set to true and both initial and trailing field enclosers
      are set, fields will be scanned, and embedded trailing encloser
      characters will be escaped. Use this option when some of the field
      values may contain the trailing encloser character.
    </description>
  </property>

<!-- properties for DelimitedTextInputFormat -->
  <property>
    <name>oracle.hadoop.loader.input.fieldTerminator</name>
    <value>,</value>
    <description>
      A single character to delimit fields for DelimitedTextInputFormat.
      Alternate representation: \uHHHH (where HHHH is the character's UTF-16
      encoding).
    </description>
  </property>
```

```
 <property>
  <name>oracle.hadoop.loader.input.initialFieldEncloser</name>
  <value></value>
  <description>
   When this value is set, fields are allowed to be enclosed
   between the specified character and
   ${oracle.hadoop.loader.input.trailingFieldEncloser}.

   If this value is set, it must be either a single character, or \uHHHH
   (where HHHH is the character's UTF-16 encoding).

   ${oracle.hadoop.loader.input.initialFieldEncloser} and
   ${oracle.hadoop.loader.input.trailingFieldEncloser} must be either
   both not set, or both set.
   A zero length value means no enclosers (default value).
  </description>
 </property>

 <property>
  <name>oracle.hadoop.loader.input.trailingFieldEncloser</name>
  <value></value>
  <description>
   When this value is set, fields are allowed to be enclosed
   between ${oracle.hadoop.loader.input.initialFieldEncloser}
   and the specified character.

   If this value is set, it must be either a single character, or \uHHHH
   (where HHHH is the character's UTF-16 encoding).

   ${oracle.hadoop.loader.input.initialFieldEncloser} and
   ${oracle.hadoop.loader.input.trailingFieldEncloser} must be either
   both not set, or both set.
   A zero length value means no enclosers (default value).
  </description>
 </property>

 <!--Properties for tuning the sampler-->
 <!-- set numThreads > 1 for large datasets -->
 <property>
  <name>oracle.hadoop.loader.sampler.numThreads</name>
  <value>5</value>
  <description>Number of sampler threads.
  This value should be set based on the processor and memory resources
  available to the job tracker node. A higher number of sampler threads
  implies higher concurrency in sampling.
  The default value is 5 threads.
  </description>
 </property>

<property>
  <name>oracle.hadoop.loader.sampler.maxLoadFactor</name>
  <value>0.05</value>
  <description>
   The maximum acceptable reducer load factor.
   In a perfectly load balanced job, every reducer is assigned
   an equal amount of work (or load).
   Load factor is the percent overload per reducer
   i.e. (assigned load - ideal load)%
   For example: a value of 0.05, indicates that it is acceptable for
   reducers to be assigned up to 5% more data than their ideal load.
```

```
     If load balancing is successful, it guarantees this
     maximum load factor at the specified confidence.
     (see oracle.hadoop.loader.sampler.loadCI)
     Default = 0.05, another common value is 0.1.
   </description>
</property>

<property>
  <name>oracle.hadoop.loader.sampler.loadCI</name>
  <value>0.95</value>
  <description>
    The confidence level for the specified
    maximum reducer load factor.
    (See oracle.hadoop.loader.sampler.maxLoadFactor)
    Default = 0.95, other common values = 0.90, 0.99
  </description>
</property>

<property>
  <name>oracle.hadoop.loader.sampler.minSplits</name>
  <value>5</value>
  <description>
    The minimum number of splits that will be
    read by the sampler. If the total number of splits
    is lesser than this value, then the sampler will read
    all splits. Splits may be read partially.
    A non-positive value is equivalent to minSplits=1.
    The default value is 5.
  </description>
</property>

<property>
  <name>oracle.hadoop.loader.sampler.hintMaxSplitSize</name>
  <value>1048576</value>
  <description>
    The sampler sets Hadoop configuration parameter
    mapred.max.split.size to this value before it calls the InputFormat's
    getSplits() method.
    The value of mapred.max.split.size is only set to this value for the
    duration of sampling, it is not changed in the actual job
    configuration. Some InputFormats (e.g. FileInputFormat) use the
    maximum split size as a hint to determine the number of splits
    returned by getSplits(). Smaller split sizes imply that more
    chunks of data will be sampled at random (good). While large splits are
    better for IO performance, they are not necessarily better for sampling.
    Set this value to be small enough for good sampling performance,
    but not any smaller: extremely small values can cause inefficient IO
    performance and cause getSplits() to run out of memory by returning too
    many splits.
    The recommended minimum value for this property is 1048576 bytes (1 MB).
    This value can be increased for larger datasets (e.g. tens of terabytes)
    or if the InputFormat's getSplits() method throws an OutOfMemoryError.
    If the specified value is less than 1, this property is ignored.
    The default value is 1048576 bytes (1 MB).
  </description>
</property>

<property>
  <name>oracle.hadoop.loader.sampler.hintNumMapTasks</name>
  <value>100</value>
```

```
    <description>
      The sampler sets Hadoop configuration parameter
      mapred.map.tasks to this value for the duration of sampling.
      The value of mapred.map.tasks is not changed in the actual job
      configuration. Some InputFormats (e.g. DBInputFormat) use the
      number of map tasks parameter as a hint to determine the number of
      splits returned by getSplits(). Higher values imply that more chunks
      of data will be sampled at random (good). The default value is 100.
      This value should typically be increased for large datasets (e.g. more
      than a million rows), while keeping in mind that extremely large values
      can cause the InputFormat's getSplits() method to run out of memory by
      returning too many splits.
      If the specified value is less than 1, this property is ignored.
      The default value is 100.
    </description>
  </property>

  <property>
    <name>oracle.hadoop.loader.sampler.maxSamplesPct</name>
    <value>0.01</value>
    <description>
      This property specifies the maximum data to sample, as a
      percentage of the total amount of data. In general, the
      sampler will stop sampling if any one of the following is true:
      (1) it has collected the minimum number of samples
          required for optimal load-balancing, or
      (2) the percent of data sampled exceeds
          oracle.hadoop.loader.sampler.maxSamplesPct, or
      (3) the number of bytes sampled exceeds
          oracle.hadoop.loader.sampler.maxHeapBytes.
      If this parameter is set to a negative value,
      condition (2) is not imposed.
      The default value is 0.01 (1%).
    </description>
  </property>

  <property>
    <name>oracle.hadoop.loader.sampler.maxHeapBytes</name>
    <value>-1</value>
    <description>
      This value specifies the maximum memory available to
      the sampler in bytes. In general, the sampler will
      stop sampling when any one of these conditions is true:
      (1) it has collected the minimum number of samples
          required for optimal load-balancing, or
      (2) the percent of data sampled exceeds
          oracle.hadoop.loader.sampler.maxSamplesPct, or
      (3) the number of bytes sampled exceeds
          oracle.hadoop.loader.sampler.maxHeapBytes.
      If this parameter is set to a negative value,
      condition (3) is not imposed.
      Default = -1 (no memory restrictions on the sampler).
    </description>
  </property>

</configuration>
```

Third-Party Licenses for Bundled Software

Oracle Loader for Hadoop installs the following third-party products:

- Apache Avro
- Apache Commons Mathematics Library
- Jackson JSON Processor

Oracle Loader for Hadoop includes Oracle 11*g* Release 2 (11.2) client libraries. For information about third party product included with Oracle Database 11*g* Release 2 (11.2), refer to*Oracle Database Licensing Information*.

Unless otherwise specifically noted, or as required under the terms of the third party license (e.g., LGPL), the licenses and statements herein, including all statements regarding Apache-licensed code, are intended as notices only.

Apache Licensed Code

The following is included as a notice in compliance with the terms of the Apache 2.0 License, and applies to all programs licensed under the Apache 2.0 license:

You may not use the identified files except in compliance with the Apache License, Version 2.0 (the "License.")

You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

A copy of the license is also reproduced below.

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

See the License for the specific language governing permissions and limitations under the License.

Apache License

Version 2.0, January 2004
http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. **Definitions**

   "License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

   "Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

   "Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

   "You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

   "Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. **Grant of Copyright License**. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. **Grant of Patent License**. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. **Redistribution**. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

   a. You must give any other recipients of the Work or Derivative Works a copy of this License; and

b. You must cause any modified files to carry prominent notices stating that You changed the files; and

c. You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

d. If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. **Submission of Contributions**. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. **Trademarks**. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. **Disclaimer of Warranty**. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. **Limitation of Liability**. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. **Accepting Warranty or Additional Liability**. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work

To apply the Apache License to your work, attach the following boilerplate notice, with the fields enclosed by brackets "[]" replaced with your own identifying information. (Do not include the brackets!) The text should be enclosed in the appropriate comment syntax for the file format. We also recommend that a file or class name and description of purpose be included on the same "printed page" as the copyright notice for easier identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

`http://www.apache.org/licenses/LICENSE-2.0`

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

This product includes software developed by The Apache Software Foundation (`http://www.apache.org/`) (listed below):

Apache Avro avro-1.5.4.jar

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

`http://www.apache.org/licenses/LICENSE-2.0`

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

Apache Commons Mathematics Library 2.2

Copyright 2001-2011 The Apache Software Foundation

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

`http://www.apache.org/licenses/LICENSE-2.0`

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

Jackon JSON Library 1.5.2

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

4

Oracle Data Integrator Application Adapter for Hadoop

This chapter describes Oracle Data Integrator. It contains these sections:

See Also: *Oracle Fusion Middleware Application Adapters Guide for Oracle Data Integrator*

This chapter describes how to work with Hadoop suite of Knowledge Modules in the Oracle Data Integrator.

Introduction

Apache Hadoop is designed to handle and process data from data sources that are typically non-RDBMS and data volumes that are typically beyond what is handled by relational databases.

The Oracle Data Integrator Application Adapter for Hadoop enables data integration developers to integrate and transform data easily within Hadoop using Oracle Data Integrator. Employing familiar and easy-to-use tools and preconfigured knowledge modules, the adapter provides the following capabilities:

- Loading data into Hadoop from the local file system and HDFS.
- Performing validation and transformation of data within Hadoop.
- Loading processed data from Hadoop to Oracle Database for further processing and generating reports.

Concepts

Typical processing in Hadoop includes data validation and transformations that are programmed as MapReduce jobs. Designing and implementing a MapReduce job requires expert programming knowledge. However, using Oracle Data Integrator and the Oracle Data Integrator Application Adapter for Hadoop, you do not need to write MapReduce jobs. Oracle Data Integrator uses Hive and the Hive Query Language (HiveQL), a SQL-like language for implementing MapReduce jobs. The Oracle Data

Integrator graphical user interface enhancing the developer's experience and productivity while enabling them to create Hadoop integrations.

When implementing a big data processing scenario, the first step is to load the data into Hadoop. The data source is typically in the local file system, HDFS, Hive tables, or external Hive tables.

After the data is loaded, you can validate and transform the data using HiveQL like you do in SQL. You can perform data validation such as checking for NULLS and primary keys, and transformations such as filtering, aggregations, set operations, and derived tables. You can also include customized procedural snippets (scripts) for processing the data.

When the data has been aggregated, condensed, or crunched down, you can load it into Oracle Database for further processing and analysis. Oracle Loader for Hadoop is recommended for optimal loading into Oracle Database.

Knowledge Modules

Oracle Data Integrator provides the knowledge modules described in Table 4–1 for use with Hadoop.

Table 4–1 Oracle Data Integrator Application Adapter for Hadoop Knowledge Modules

KM Name	Description	Source	Target
IKM File To Hive (Load Data)	Loads data from local and HDFS files into Hive tables. It provides options for better performance through Hive partitioning and fewer data movements. This KM supports wild cards (*,?).	File System	Hive
IKM Hive Control Append	Integrates data into a Hive target table in truncate/ insert (append) mode. Data can be controlled (validated). Invalid data is isolated in an error table and can be recycled.	Hive	Hive
IKM Hive Transform	Integrates data into a Hive target table after the data has been transformed by a customized script such as Perl or Python.	Hive	Hive
IKM File-Hive to Oracle (OLH)	Integrates data from an HDFS file or Hive source into an Oracle Database target using Oracle Loader for Hadoop.	File System or Hive	Oracle Database
CKM Hive	Validates data against constraints.	NA	Hive
RKM Hive	Reverse engineers Hive tables.	Hive Metadata	NA

Security

For security information for Oracle Data Integrator, see the *Oracle Fusion Middleware Developer's Guide for Oracle Data Integrator*.

Setting up the Topology

This step declares, in Oracle Data Integrator, the data server and the physical and logical schemas that are used to store the file system and Hive information.

This section contains the following topics:

- Setting up the File Data Source
- Setting Up the Hive Data Source
- Setting Up the Oracle Data Integrator Agent to Execute Hadoop Jobs
- Configuring Oracle Data Integrator Studio for Executing Hadoop Jobs on the Local Agent

Setting up the File Data Source

In the Hadoop context there is a distinction between files in the Hadoop Distributed File System (HDFS) and local files (files outside of HDFS).

To define a data source:

1. Create a DataServer object under File technology.
2. Create a Physical Schema object for every directory to be accessed.
3. Create a Logical Schema object for every directory to be accessed.
4. Create a Model for every LogicalSchema.
5. Create one or more data stores for each different type of file and wildcard name pattern.
6. For HDFS files, create a DataServer object under File technology by entering the HDFS name node in the field JDBC URL. For example:

   ```
   hdfs://bda1node01.example.com:9000
   ```

   Note: There is no dedicated technology defined for HDFS files.

Setting Up the Hive Data Source

The following steps in Oracle Data Integrator are required for connecting to a Hive system. Oracle Data Integrator connects to Hive using JDBC.

Prerequisites

The Hive technology must be included in the standard Oracle Data Integrator technologies. If it is not, then import the technology in `INSERT_UPDATE` mode from the xml-reference folder.

All Hive-specific FlexFields must be added. For pre-11.1.1.6.0 repositories, the FlexFields are added as part of the repository upgrade process.

To set up a Hive data source:

1. Place all required Hive JDBC jars into the Oracle Data Integrator user lib folder:

   ```
   $HIVE_HOME/lib/*.jar
   $HADOOP_HOME/hadoop-*-core*.jar,
   $HADOOP_HOME/Hadoop-*-tools*.jar
   ```

2. Create a DataServer object under Hive technology.
3. Set the following locations under JDBC:

   JDBC Driver: `org.apache.hadoop.hive.jdbc.HiveDriver`

   JDBC URL: for example, `jdbc:hive://BDA:10000/default`

4. Set the following under Flexfields:

Hive Metastore URIs: for example, thrift://BDA:10000

5. Create a Physical Default Schema.

   As of Hive 0.7.0, no schemas or databases are supported. Only Default is supported. Enter default in both schema fields of the physical schema definition.

6. Ensure that the Hive server is up and running.
7. Test the connection to the DataServer.
8. Create a Logical Schema object.
9. Create at least one Model for the LogicalSchema.
10. Import RKM Hive as a global KM or into a project.
11. Create a new model for Hive Technology pointing to the logical schema.
12. Perform a custom reverse using RKM Hive.

At the end of this process, the Hive DataModel contains all Hive tables with their columns, partitioning, and clustering details stored as FlexField values.

Setting Up the Oracle Data Integrator Agent to Execute Hadoop Jobs

After setting up an Oracle Data Integrator agent, configure it to work with the Oracle Data Integrator Application Adapter for Hadoop.

To configure the Oracle Data Integrator agent:

1. Install Hadoop on your Oracle Data Integrator Agent computer. Ensure that the HADOOP_HOME environment variable is set.

   For Oracle Big Data Appliance, see the *Oracle Big Data Appliance Software User's Guide* for instructions for setting up a remote Hadoop client.

2. Install Hive on your Oracle Data Integrator Agent computer. Ensure that the HIVE_HOME environment variable is set.

3. Copy these jar files to the Oracle Data Integrator agent drivers directory.

   ```
   $HIVE_HOME/lib/*.jar,
   $HADOOP_HOME/hadoop-*-core*.jar
   $HADOOP_HOME/hadoop-*-tools*.jar
   ```

   See the *Oracle Fusion Middleware Installation Guide for Oracle Data Integrator* for information about adding drivers to the agent. This step enables the Oracle Data Integrator agent to load the Hive JDBC driver.

4. Set environment variable ODI_HIVE_SESSION_JARS to include Hive RegEx SerDe:

   ```
   ODI_HIVE_SESSION_JARS=$HIVE_HOME/lib/hive-contrib-0.7.1-cdh3u3.jar
   ```

   Include other jars as required, such as custom SerDes jars. These jars are added to every Hive JDBC session and thus are added to every Hive MapReduce job.

5. Set environment variable HADOOP_CLASSPATH:

   ```
   HADOOP_CLASSPATH=$HIVE_HOME/lib/hive-metastore-0.7.1-cdh3u3.jar:$HIVE_
   HOME/lib/libthrift.jar:$HIVE_HOME/lib/libfb303.jar:$HIVE_
   HOME/lib/hive-common-0.7.1-cdh3u3.jar:$HIVE_
   HOME/lib/hive-exec-0.7.1-cdh3u3.jar.
   ```

   This setting enables the Hadoop script to start Hive MapReduce jobs.

To use Oracle Loader for Hadoop:

1. Install Oracle Loader for Hadoop on your Oracle Data Integrator Agent system.
2. Install Oracle client on your Oracle Data Integrator Agent system. See the Oracle Loader for Hadoop requirements for the Oracle client version.
3. Set environment variable `OLH_HOME`.
4. Set environment variable `ODI_OLH_JARS`.

   You must list all jar files required for Oracle Loader for Hadoop. See the `oracle.hadoop.loader.libjars` property in "OraLoader for Hadoop Configuration Properties" on page 3-23.

   This is a comma-separated list of jar files:

   ```
   ODI_OLH_JARS=OLH_HOME/jlib/ojdbc6.jar,$OLH_HOME/jlib/orai18n.jar,$OLH_
   HOME/jlib/orai18n-utility.jar,$OLH_HOME/jlib/orai18n-mapping.jar,$OLH_
   HOME/jlib/orai18n-collation.jar,$OLH_HOME/jlib/oraclepki.jar,$OLH_
   HOME/jlib/osdt_cert.jar,$OLH_HOME/jlib/osdt_core.jar,$OLH_
   HOME/jlib/commons-math-2.2.jar,$OLH_HOME/jlib/jackson-core-asl-1.5.2.jar,$OLH_
   HOME/jlib/jackson-mapper-asl-1.5.2.jar,$OLH_HOME/jlib/avro-1.5.4.jar,$OLH_
   HOME/jlib/avro-mapred-1.5.4.jar,$OLH_HOME/jlib/oraloader.jar,$HIVE_
   HOME/lib/hive-metastore-0.7.1-cdh3u3.jar,$HIVE_HOME/lib/libthrift.jar,$HIVE_
   HOME/lib/libfb303.jar,$HIVE_HOME/lib/hive-common-0.7.1-cdh3u3.jar,$HIVE_
   HOME/lib/hive-exec-0.7.1-cdh3u3.jar
   ```

5. Add paths to `HADOOP_CLASSPATH`:

   ```
   $HADOOP_CLASSPATH= $OLH_HOME/jlib:$HADOOP_CLASSPATH
   ```

6. Verify that `ODI_OLH_SHAREDLIBS` lists all native libraries required for Oracle Loader for Hadoop. See the `oracle.hadoop.loader.sharedLibs` property in the "OraLoader for Hadoop Configuration Properties" on page 3-23.

   This is a comma separated list of shared libraries files:

   ```
   ODI_OLH_SHAREDLIBS= $OLH_HOME/lib/libolh11.so,$OLH_
   HOME/lib/libclntsh.so.11.1,$OLH_HOME/lib/libnnz11.so,$OLH_HOME/lib/libociei.so
   ```

7. If Oracle Loader for Hadoop is used in OCI mode, then check these variables:
   - `JAVA_LIBRARY_PATH` must include `$OLH_HOME/lib`. It locates the Oracle Loader for Hadoop native library `libolh11.so`.
   - `LD_LIBRARY_PATH` must include `$ORACLE_HOME/lib`.

Configuring Oracle Data Integrator Studio for Executing Hadoop Jobs on the Local Agent

For executing Hadoop jobs on the local agent of an Oracle Data Integrator Studio installation, follow the steps in the previous section with the following change: Copy jar files into the Oracle Data Integrator `userlib` directory instead of the driver directory.

Setting up an Integration Project

Setting up a project follows the standard procedures. See *Oracle Fusion Middleware Developer's Guide for Oracle Data Integrator*.

Import the following KMs into Oracle Data Integrator project:

- IKM File To Hive (Load Data)

- IKM Hive Control Append
- IKM Hive Transform
- IKM File-Hive to Oracle (OLH)
- CKM Hive
- RKM Hive

Creating Oracle Data Integrator Model from a Reverse-Engineering Hive Model

This section contains the following topics:

- Creating a Model
- Reverse-Engineering Hive Tables

Creating a Model

Create a model based on the technology hosting Hive and on the logical schema created when configuring the Hive connection using the standard procedure, as described in the *Oracle Fusion Middleware Developer's Guide for Oracle Data Integrator*.

Reverse-Engineering Hive Tables

The Hive RKM is used to reverse-engineer Hive tables and views. To perform a Customized Reverse-Engineering of Hive tables with the Hive RKM, follow the usual procedures, as described in the *Oracle Fusion Middleware Developer's Guide for Oracle Data Integrator*. This topic details information specific to Hive tables.

The reverse-engineering process creates the data stores for the corresponding Hive table or views. You can use the data stores as either a source or a target in an integration interface.

The RKM reverses these metadata elements:

- Hive tables and views as Oracle Data Integrator data stores.

  Specify the reverse mask in the Mask field, then select the tables and views to reverse. The Mask field in the Reverse tab filters reverse-engineered objects based on their names. The Mask field cannot be empty and must contain at least the percentage symbol (%).
- Hive columns as Oracle Data Integrator columns with their data types.
- Information about buckets, partitioning, cluster, and sort columns are set in the respective FlexFields in the Oracle Data Integrator data store or column metadata.

  Table 4–2 describes the created FlexFields.

Table 4–2 FlexFields for Reverse-Engineered Hive Tables and Views

Object	FlexField Name	FlexField Code	FlexField Type	Description
DataStore	Hive Buckets	`HIVE_BUCKETS`	String	Number of Buckets to be used for clustering
Column	Hive Partition Column	`HIVE_PARTITION_COLUMN`	Numeric	All partitioning columns are marked as "1". Partition information can come from the following: ■ Mapped Source Column ■ Constant value specified in the target column ■ File name fragment
Column	Hive Cluster Column	`HIVE_CLUSTER_COLUMN`	Numeric	All cluster columns are marked as "1".
Column	Hive Sort Column	`HIVE_SORT_COLUMN`	Numeric	All sort columns are marked as "1".

Table 4–3 describes the options for Hive RKM.

Table 4–3 Hive RKM Options

Option	Description
`USE_LOG`	Log intermediate results?
`LOG_FILE_NAME`	Path and file name of log file. Default path is the user home and default file name is reverse.log.

Designing the Interface

After reverse engineering Hive tables and configuring them, you can choose from these interface configurations:

- Loading Data from Files into Hive
- Validating and Transforming Data Within Hive
- Loading into Oracle from Hive and HDFS

Loading Data from Files into Hive

To load data from the local or the HDFS file system into Hive tables:

1. Create the data stores for local files and HDFS files.

   Refer to the *Oracle Fusion Middleware Connectivity and Knowledge Modules Guide for Oracle Data Integrator* for information on reverse engineering and configuring local file data sources.

2. Create an interface using the file data store as the source and the corresponding Hive table as the target. Use the IKM File To Hive (Load Data) knowledge module specified in the flow tab of the interface. This IKM loads data from flat files into Hive, replacing or appending to any existing data.

IKM File to Hive

IKM File to Hive (Load Data) supports:

- One or more input files. To load multiple source files, enter an asterisk or a question mark as a wildcard character in the resource name of the file DataStore, for example, `webshop_*.log`.
- File formats:
  - Fixed length
  - Delimited
  - Customized format
- Loading options:
  - Immediate or deferred loading
  - Overwrite or append
  - Hive external tables

Table 4–4 describes the options for IKM File To Hive (Load Data). See the KM for additional details.

Table 4–4 IKM File To Hive Options

Option	Description
CREATE_TARG_TABLE	Create target table.
TRUNCATE	Truncate data in target table.
FILE_IS_LOCAL	Is the file in the local file system or in HDFS?
EXTERNAL_TABLE	Use an externally managed Hive table
USE_STAGING_TABLE	Use a Hive staging table. Select this option if the source and target do not match or if the partition column value is part of the data file. If the partitioning value is provided by a file name fragment or a constant in target mapping, then set this value to `false`.
DELETE_TEMPORARY_OBJECTS	Remove temporary objects after the interface execution.
DEFER_TARGET_LOAD	Load data into the final target now or defer?
OVERRIDE_ROW_FORMAT	Provide a parsing expression for handling a custom file format to perform the mapping from source to target.
STOP_ON_FILE_NOT_FOUND	Stop if no source file is found?

Validating and Transforming Data Within Hive

After loading data into Hive, you can validate and transform the data using the following KMs.

IKM Hive Control Append

This KM validates and controls the data, and integrates it into a Hive target table in truncate/insert (append) mode. Invalid data is isolated in an error table and can be recycled. This KM supports inline view interfaces that use either IKM Hive Control Append or IKM Hive Transform.

Table 4–5 lists the options. See the KM for additional details.

Table 4–5 IKM Hive Control Append Options

Option	Description
FLOW_CONTROL	Validate incoming data?
RECYCLE_ERRORS	Reintegrate data from error table?
STATIC_CONTROL	Validate data after load?
CREATE_TARG_TABLE	Create target table?
DELETE_TEMPORARY_OBJECTS	Remove temporary objects after execution?
TRUNCATE	Truncate data in target table?

CKM Hive

This KM checks data integrity for Hive tables. It verifies the validity of the constraints of a Hive data store and rejects the invalid records into an error table. You can use CKM Hive for static control and flow control. You must also define these constraints on the stored data.

Table 4–6 lists the options for this CKM. See the KM for additional details.

Table 4–6 CKM Hive Options

Option	Description
DROP_ERROR_TABLE	Drop error table before execution?

IKM Hive Transform

This KM performs transformations. It uses a shell script to transform the data, then integrates it into a Hive target table using replace mode. The KM supports inline view interfaces and can be used as an inline-view for IKM Hive Control Append.

The transformation script must expect the input columns in the order defined by the source data store. Only mapped source columns are streamed into the transformations. The transformation script must provide the output columns in the order defined by the target data store.

Table 4–7 lists the options for this IKM. See the KM for additional details.

Table 4–7 IKM Hive Transform Options

Option	Description
CREATE_TARG_TABLE	Create target table?
DELETE_TEMPORARY_OBJECTS	Remove temporary objects after execution?
TRANSFORM_SCRIPT_NAME	Script file name
TRANSFORM_SCRIPT	Script content
PRE_TRANSFORM_DISTRIBUTE	Provides an optional, comma-separated list of source column names, which enables the KM to distribute the data before the transformation script is applied.
PRE_TRANSFORM_SORT	Provide an optional, comma-separated list of source column names, which enables the KM to sort the data before the transformation script is applied
POST_TRANSFORM_DISTRIBUTE	Provides an optional, comma-separated list of target column names, which enables the KM to distribute the data after the transformation script is applied.

Table 4–7 (Cont.) IKM Hive Transform Options

Option	Description
POST_TRANSFORM_SORT	Provides an optional, comma-separated list of target column names, which enables the KM to sort the data after the transformation script is applied.

Loading into Oracle from Hive and HDFS

IKM File-Hive to Oracle (OLH) integrates data from an HDFS file or Hive source into an Oracle Database target using Oracle Loader for Hadoop. Using the interface configuration and the selected options, the KM generates an appropriate Oracle target instance. Hive and Hadoop versions must follow the Oracle Loader for Hadoop requirements.

See Also:

- "Oracle Loader for Hadoop" on page 1-5 for required versions of Hadoop and Hive
- "Setting Up the Oracle Data Integrator Agent to Execute Hadoop Jobs" on page 4-4 for required environment variable settings

Table 4–8 lists the options for this IKM. See the KM for additional details.

Table 4–8 IKM File - Hive to Oracle (OLH)

Option	Description
OLH_OUTPUT_MODE	Specify either JDBC, OCI, or Data Pump for data transfer.
CREATE_TARG_TABLE	Create target table?
USE_HIVE_STAGING_TABLE	Materialize Hive source data before extract?
USE_ORACLE_STAGING_TABLE	Use an Oracle database staging table?
EXT_TAB_DIR_LOCATION	Shared file path used for Oracle Data Pump transfer.
TEMP_DIR	Local path for temporary files.
MAPRED_OUTPUT_BASE_DIR	HDFS directory for Oracle Loader for Hadoop output files.
FLOW_TABLE_OPTIONS	Options for flow (stage) table creation when an Oracle database staging table is used.
DELETE_TEMPORARY_OBJECTS	Remove temporary objects after execution?
OVERRIDE_INPUTFORMAT	Set to handle custom file formats.
EXTRA_OLH_CONF_PROPERTIES	Optional Oracle Loader for Hadoop configuration file properties
TRUNCATE	Truncate data in target table?
DELETE_ALL	Delete all data in target table?

5

Oracle R Connector for Hadoop

This chapter describes R support for big data. It contains these topics:

About Oracle R Connector for Hadoop

Oracle R Connector for Hadoop is an R package that provides an interface between the local R environment and Hadoop. You install and load this package the same as you would for any other R package. Using simple R functions, you can copy data between R memory, the local file system, and HDFS. You can schedule R programs to execute as Hadoop MapReduce jobs and return the results to any of those locations.

Oracle R Connector for Hadoop APIs

Oracle R Connector for Hadoop provides API access from a local R client to Hadoop, using these APIs:

- `hadoop`: Provides an interface to Hadoop MapReduce.
- `hdfs`: Provides an interface to HDFS.
- `orhc`: Provides an interface between the local R instance and Oracle Database.

All of these functions are included in the ORHC library. The functions are listed in this chapter in alphabetical order.

Access to Oracle Database

A separate package R package, Oracle R Enterprise, provides access to Oracle Database. Access to the data stored in Oracle Database is always restricted to the access rights granted by your Oracle DBA.

Oracle R Enterprise provides direct access to Oracle Database objects and enables you to perform statistical analysis on database tables, views, and other data objects. Users can develop R scripts for deployment while retaining the results in the secure environment of Oracle Database.

Oracle R Enterprise is included in the Oracle Database Advanced Analytics option; it is not included in Oracle Big Data Connectors.

See Also: *Oracle R Enterprise User's Guide*

Scenarios for Using Oracle R Packages

The following scenario may help you identify opportunities for using Oracle R Connector for Hadoop with Oracle R Enterprise.

Using the Oracle R Connector for Hadoop, you might look for files that you have access to on HDFS and schedule R calculations to execute on data in one such file. Furthermore, you can upload data stored in text files on your local file system into HDFS for calculations, schedule an R script for execution on the Hadoop cluster, and download the results into a local file.

Using the Oracle Database Advanced Analytics option, you can open the R interface and connect to Oracle Database to work on the tables and views that are visible based on your database privileges. You can filter out rows, add derived columns, project new columns, and perform visual and statistical analysis using Oracle R Enterprise.

Again using the Oracle R Connector for Hadoop, you might deploy a MapReduce job on Hadoop for CPU-intensive calculations written in R. The calculation can use data stored in HDFS or with the Oracle Database Advanced Analytics option, in Oracle Database You can return the output of the calculation to Oracle Database and the R console for visualization or additional processing.

Security Notes for Oracle R Connector for Hadoop

Oracle R Connector for Hadoop invokes the Sqoop utility to connect to Oracle Database either to extract data or to store results. Sqoop is a command-line utility for Hadoop that imports and exports data between HDFS or Hive and structured databases, such as Oracle Database. The name Sqoop comes from "SQL to Hadoop."

The following explains how Oracle R Connector for Hadoop stores a database user password and sends it to Sqoop.

Oracle R Connector for Hadoop stores a user password only when the user establishes the database connection in a mode that does not require reentering the password each time. The password is stored encrypted in memory. See orhc.connect on page 5-29.

Oracle R Connector for Hadoop generates a configuration file for Sqoop and uses it to invoke Sqoop locally. The file contains the user's database password obtained by either prompting the user or from the encrypted in-memory representation. The file has local user access permissions only. The file is created, the permissions are set explicitly, then the file is open for writing and filled with data.

Sqoop uses the configuration file to generate custom JAR files dynamically for the specific database job and passes the JAR files to the Hadoop client software. The password is stored inside the compiled JAR file; it is not stored in plain text.

The JAR file is transferred to the Hadoop cluster over a network connection. The network connection and the transfer protocol is specific to Hadoop, such as port 5900.

The configuration file is deleted after Sqoop finishes compiling its JAR files and starts its own Hadoop jobs.

Functions in Alphabetical Order

`hadoop.exec`
`hadoop.run`
`hdfs.attach`

hdfs.cd
hdfs.download
hdfs.exists
hdfs.get
hdfs.ls
hdfs.mkdir
hdfs.parts
hdfs.pull
hdfs.push
hdfs.put
hdfs.pwd
hdfs.rm
hdfs.rmdir
hdfs.sample
hdfs.size
hdfs.upload
orhc.connect
orhc.disconnect
orhc.reconnect
orhc.which

Functions By Category

The OHRC functions are grouped into these categories:

- Making Connections
- Copying Data
- Exploring Files
- Executing Scripts

Making Connections

orhc.connect
orhc.disconnect
orhc.reconnect
orhc.which

Copying Data

hdfs.upload
hdfs.download
hdfs.get
hdfs.push
hdfs.put
hdfs.pull

Exploring Files

hdfs.attach
hdfs.cd
hdfs.exists
hdfs.ls
hdfs.mkdir
hdfs.parts
hdfs.pwd
hdfs.rm
hdfs.rmdir

hdfs.sample
hdfs.size

Executing Scripts

hadoop.exec
hadoop.run

hadoop.exec

Starts the Hadoop engine and sends the mapper, reducer and combiner R functions for execution. You must load the data into HDFS first.

Usage

```
hadoop.exec(
        dfs.id,
        mapper,
        reducer,
        combiner,
        export)
```

Arguments

dfs.id
Object identifier in HDFS.

mapper
Name of a mapper function written in the R language.

reducer
Name of a reducer function written in the R language (optional).

combiner
Name of a combiner function written in the R language (optional).

export
Names of exported R objects from your current R environment that are referenced by any of your mapper, reducer, or combiner functions (optional).

Usage Notes

This function provides more control of the data flow than hadoop.run. You must use `hadoop.exec` when chaining several mappers and reducers in a pipeline, because the data does not leave HDFS. The results are stored in HDFS.

Return Value

Data object identifier in HDFS.

Example

This sample script uses `hdfs.attach` to obtain the object identifier of a small, sample data file in HDFS named ontime_R.

```
dfs <- hdfs.attach('ontime_R')
res <- NULL
res <- hadoop.exec(
    dfs,
    mapper = function(key, ontime) {
        if (key == 'SFO') {
            keyval(key, ontime)
        }
    },
    reducer = function(key, vals) {
        sumAD <- 0
        count <- 0
        for (x in vals) {
           if (!is.na(x$ARRDELAY)) {sumAD <- sumAD + x$ARRDELAY; count <- count +
1}
        }
        res <- sumAD / count
        keyval(key, res)
    }
)
```

After the script runs, the location of the results is identified by the `res` variable, in an HDFS file named /user/oracle/xq/orhc3d0b8218:

```
R> res
[1] "/user/oracle/xq/orhc3d0b8218"
attr(,"dfs.id")
[1] TRUE
R> print(hdfs.get(res))
  key     val1
1 SFO 27.05804
```

hadoop.run

Starts the Hadoop engine and sends the mapper, reducer and combiner R functions for execution. If the data is not already stored in HDFS, then `hadoop.run` first copies the data there.

Usage

```
hadoop.run(
        data,
        mapper,
        reducer,
        combiner,
        export)
```

Arguments

data
Data frame, Oracle R Enterprise frame (`ore.frame`), or an HDFS file descriptor.

mapper
Name of a mapper function written in the R language.

reducer
Name of a reducer function written in the R language (optional).

combiner
Name of a combiner function written in the R language (optional).

export
Names of exported R objects.

Usage Notes

The `hadoop.run` function returns the results from HDFS to the source of the input data. For example, the results for HDFS input data are kept in HDFS, and the results for `ore.frame` input data are pulled into Oracle Database.

Return Value

An object in the same format as the input data.

Example

This sample script uses `hdfs.attach` to obtain the object identifier of a small, sample data file in HDFS named ontime_R.

```
dfs <- hdfs.attach('ontime_R')
res <- NULL
res <- hadoop.run(
    dfs,
    mapper = function(key, ontime) {
        if (key == 'SFO') {
            keyval(key, ontime)
        }
    },
    reducer = function(key, vals) {
        sumAD <- 0
        count <- 0
        for (x in vals) {
           if (!is.na(x$ARRDELAY)) {sumAD <- sumAD + x$ARRDELAY; count <- count +
1}
        }
        res <- sumAD / count
        keyval(key, res)
    }
)
```

After the script runs, the location of the results is identified by the `res` variable, in an HDFS file named /user/oracle/xq/orhc3d0b8218:

```
R> res
[1] "/user/oracle/xq/orhc3d0b8218"
attr(,"dfs.id")
[1] TRUE
R> print(hdfs.get(res))
  key     val1
1 SFO 27.05804
```

hdfs.attach

Pulls data from an unstructured data file in HDFS into the Oracle R Connector for Hadoop framework. By default, data files in HDFS are not visible to the R Connector. However, if you know the name of the data file, you can use this function to attach it to the R Connector name space.

If the data does not have metadata identifying the names and data types of the columns, then the function samples the data to deduce the data type (number or string). It then re-creates the file with the appropriate metadata.

Usage

```
hdfs.attach(dfs.name)
```

Arguments

dfs.name
The name of a file in HDFS.

Usage Notes

Use this function to attach an HDFS file to your R environment, the same as you might attach a data frame.

Return Value

The object ID of the file in HDFS, or `NULL` if the operation failed.

Example

This example stores the object ID of `ontime_R` in a variable named `dfs`, then displays its value.

```
R> dfs <- hdfs.attach('ontime_R')
R> dfs
[1] "/user/oracle/xq/ontime_R"
attr(,"dfs.id")
[1] TRUE
```

hdfs.cd

Sets the default HDFS path.

Usage

```
hdfs.cd(dfs.path)
```

Arguments

dfs.path
A path that is either absolute or relative to the current path.

Return Value

TRUE if the path is changed successfully, or FALSE if the operation failed.

Example

This example changes the current directory from /user/oracle to /user/oracle/sample:

```
R> hdfs.cd("sample")
[1] "/user/oracle/sample"
```

hdfs.download

Copies a file from HDFS to the local file system.

Usage

```
hdfs.download(
        dfs.id,
        filename,
        overwrite)
```

Arguments

dfs.id
The object ID of the file in HDFS.

filename
The name of a file in the local file system where the data is copied.

overwrite
Controls whether the operation can overwrite an existing local file. Set to `TRUE` to overwrite *`filename`*, or `FALSE` to signal an error (default).

Usage Notes

This function provides the fastest and easiest way to copy a file from HDFS. No data transformations occur except merging multiple parts into a single file. The local file has the exact same data as the HDFS file.

Return Value

Local file name, or `NULL` if the copy failed.

Example

This example displays a list of files in the current HDFS directory and copies ontime2000.DB to the local file system as /home/oracle/ontime2000.dat.

```
R> hdfs.ls()
[1] "ontime2000_DB" "ontime_DB"     "ontime_File"   "ontime_R"      "testdata.dat"
R> tmpfile <- hdfs.download("ontime2000_DB", "/home/oracle/ontime2000.dat",
overwrite=F)
R> tmpfile
[1] "/home/oracle/ontime2000.dat"
```

hdfs.exists

Verifies that an object exists in HDFS.

Usage

```
hdfs.exists(
        dfs.id)
```

Arguments

dfs.id
An object ID or file name in HDFS.

Usage Notes

If this function returns TRUE, then you can attach the data and use it in a hadoop.run function. You can also use this function to validate an HDFS identifier and ensure that the data exists.

Return Value

TRUE if the identifier is valid and the data exists, or FALSE if the object is not found.

Example

This example shows that the ontime_R file exists.

```
R> hdfs.exists("ontime_R")
[1] TRUE
```

hdfs.get

Copies data from HDFS into a data frame in the local R environment. All metadata is extracted and all attributes, such as column names and data types, are restored if the data originated in an R environment. Otherwise, generic attributes like val1 and val2 are assigned.

Usage

```
hdfs.get(
        dfs.id,
        sep)
```

Arguments

dfs.id
The object ID of the file in HDFS.

sep
The symbol used to separate fields in the file. A comma (,) is the default separator.

Usage Notes

If the HDFS file is small enough to fit into an in-memory R data frame, then you can copy the file using this function instead of `hdfs.pull`. The `hdfs.get` function can be faster, because it does not use Sqoop and thus does not have the overhead incurred by `hdfs.pull`.

Return Value

A `data.frame` object in memory in the local R environment pointing to the exported data set, or `NULL` if the operation failed.

Example

This example returns the contents of a data frame named `res`.

```
R> print(hdfs.get(res))
   key      val1
1   AA 1361.4643
2   AS  515.8000
3   CO 2507.2857
4   DL 1601.6154
5   HP  549.4286
6   NW 2009.7273
7   TW 1906.0000
8   UA 1134.0821
9   US 2387.5000
10  WN  541.1538
```

hdfs.ls

Lists the names of all HDFS directories containing data in the specified path.

Usage

```
hdfs.ls(dfs.path)
```

Arguments

dfs.path
A path relative to the current default path. The default path is the current working directory.

Usage Notes

Use hdfs.cd on page 5-10 to set the default path.

Return Value

A list of data object names in HDFS, or NULL if the specified path is invalid.

Example

This example lists the subdirectories in the current directory:

```
R> hdfs.ls()
[1] "ontime_DB"   "ontime_FILE"   "ontime_R"
```

The next example lists directories in the parent directory:

```
R> hdfs.ls("..")
[1] "demo"   "input"   "olhcache"   "output"   "sample"   "xq"
```

This example returns NULL because the specified path is not in HDFS.

```
R> hdfs.ls("/bin")
NULL
```

hdfs.mkdir

Creates a subdirectory in HDFS relative to the current working directory.

Usage

```
hdfs.mkdir(
        dfs.name,
        cd)
```

Arguments

dfs.name
Name of the new directory.

cd
`TRUE` to change the current working directory to the new subdirectory, or `FALSE` to keep the current working directory (default).

Usage Notes

Text.

Return Value

Full path of the new directory as a String, or `NULL` if the directory was not created.

Example

This example creates the `/user/oracle/sample` directory.

```
R> hdfs.mkdir('sample', cd=T)
[1] "/user/oracle/sample"
attr(,"dfs.path")
[1] TRUE
```

hdfs.parts

Returns the number of parts composing an object in HDFS.

Usage

```
hdfs.parts(
        dfs.id)
```

Arguments

dfs.id
Object identifier in HDFS.

Usage Notes

HDFS splits large files into parts, which provide a basis for the parallelization of MapReduce jobs. The more parts an HDFS file has, the more mappers can run in parallel.

Return Value

Number of parts composing the object, or 0 if the object does not exist in HDFS

Example

This example shows that the `ontime_R` file has one part:

```
R> hdfs.parts("ontime_R")
[1] 1
```

hdfs.pull

Copies data from HDFS into Oracle Database.

This operation requires authentication by Oracle Database. See orhc.connect on page 5-29.

Usage

```
hdfs.pull(
        dfs.id,
        sep,
        db.name,
        overwrite,
        driver)
```

Arguments

dfs.id
The file name in HDFS.

sep
The symbol used to separate fields in the file. A comma (,) is the default separator.

db.name
The name of a table in Oracle Database. (Optional)

overwrite
Controls whether db.name can overwrite a table with the same name. Set to TRUE to overwrite the table, or FALSE to signal an error (default).

driver
Identifies the driver used to copy the data. The default driver is sqoop.

Usage Notes

Because this operation is synchronous, copying a large data set may appear to hang the R environment. You regain use of R when copying is complete.

To copy large volumes of data into Oracle Database, consider using Oracle Loader for Hadoop. With the Oracle Database Advanced Analytics option, you can use Oracle R Enterprise to analyze the data in an Oracle database.

Return Value

An ore.frame object that points to the database table with data loaded from HDFS, or NULL if the operation failed

See Also

Oracle R Enterprise User's Guide for a description of ore.frame objects.

hdfs.push

Copies data from Oracle Database to HDFS.

This operation requires authentication by Oracle Database. See `orhc.connect` on page 5-29.

> **Note:** The Oracle R Enterprise (ORE) library must be attached to use this function.

Usage

```
hdfs.push(
        x,
        key,
        dfs.name,
        overwrite,
        driver,
        split.by)
```

Arguments

x
An `ore.frame` object with the data in Oracle Database to be pushed.

key
The index or name of the key column.

dfs.name
Unique name for the object in HDFS.

overwrite
`TRUE` to allow `dfs.name` to overwrite an object with the same name, or `FALSE` to signal an error (default).

driver
Driver for copying the data (optional). The default driver is `sqoop`.

split.by
The column to use for data partitioning. (Optional)

Usage Notes

Because this operation is synchronous, copying a large data set may appear to hang the R environment. You regain use of R when copying is complete.

An `ore.frame` object is an Oracle R Enterprise metadata object that points to a database table. It corresponds to an R `data.frame` object.

Return Value

HDFS object ID pointing to the exported data set, or `NULL` if the operation failed

See Also

Oracle R Enterprise User's Guide

Example

This examples creates an `ore.frame` object named `ontime_s2000` that contains the rows from the `ONTIME_S` table in Oracle Database where the year equals 2000. Then `hdfs.push` uses `ontime_s2000` to create /user/oracle/xq/ontime2000_DB in HDFS.

```
R> ontime_s2000 <- ONTIME_S[ONTIME_S$YEAR == 2000,]
R> class(ontime_s2000)
[1] "ore.frame"
attr(,"package")
[1] "OREbase"
R> ontime2000.dfs <- hdfs.push(ontime_s2000, key='DEST', dfs.name='ontime2000_DB')
R> ontime2000.dfs
[1] "/user/oracle/xq/ontime2000_DB"
attr(,"dfs.id")
[1] TRUE
```

hdfs.put

Copies data from an ORE data frame to HDFS. Column names, data types, and other attributes are stored as metadata in HDFS.

> **Note:** The Oracle R Enterprise (ORE) library must be attached to use this function.

Usage

```
hdfs.put(
        data,
        key,
        dfs.name,
        overwrite)
```

Arguments

data
An `ore.frame` object in the local R environment to be copied to HDFS.

key
The index or name of the key column.

dfs.name
A unique name for the new file.

overwrite
Controls whether `dfs.name` can overwrite a file with the same name. Set to `TRUE` to overwrite the file, or `FALSE` to signal an error.

Usage Notes

You can use this function to transfer control parameters or to look up data relevant to a Hadoop R calculation from the R environment into an HDFS file.

You can also use `hdfs.put` instead of `hdfs.push` to copy data from `ore.frame` objects, such as database tables, to HDFS. The table must be small enough to fit in R memory, otherwise the function fails. The `hdfs.put` function first reads all table data into R memory and then transfers it to HDFS. For a small table, this function can be faster because it does not use Sqoop and thus does not have the overhead incurred by `hdfs.push`.

Return Value

The object ID of the new file, or `NULL` if the operation failed.

Example

This example creates a file named /user/oracle/xq/testdata.dat with the contents of the `dat` data frame.

```
R> myfile <- hdfs.put(dat, key='DEST', dfs.name='testdata.dat')
R> print(myfile)
[1] "/user/oracle/xq/testdata.dat"
attr(,"dfs.id")
```

```
[1] TRUE
```

hdfs.pwd

Identifies the current working directory in HDFS.

Usage

```
hdfs.pwd()
```

Return Value

The current working directory, or `NULL` if you are not connected to HDFS.

Example

This example shows that /user/oracle is the current working directory.

```
R> hdfs.pwd()
[1] "/user/oracle/"
```

hdfs.rm

Removes a file or directory from HDFS.

Usage

```
hdfs.rm(dfs.id)
```

Arguments

dfs.id
The object ID of a file in HDFS to be removed.

Usage Notes

All object identifiers in Hadoop pointing to this data are invalid after this operation.

Return Value

`TRUE` if the data is deleted, or `FALSE` if the operation failed.

Example

```
R> hdfs.rm("data1.log")
[1] TRUE
```

hdfs.rmdir

Deletes a subdirectory in HDFS relative to the current working directory.

Usage

```
hdfs.rmdir(
        dfs.name)
```

Arguments

dfs.name
Name of the directory in HDFS to delete.

Usage Notes

This function deletes all data objects stored in the directory, which invalidates all associated object identifiers in HDFS.

Return Value

`TRUE` if the directory is deleted successfully, or `FALSE` if the operation fails.

Example

```
R> hdfs.rmdir("mydata")
[1] TRUE
```

hdfs.sample

Copies a random sample of data from a Hadoop file into an R in-memory object. Use this function to copy a small sample of the original HDFS data for developing the R calculation that you ultimately want to execute on the entire HDFS data set on the Hadoop cluster.

Usage

```
hdfs.sample(
        dfs.id,
        lines,
        sep)
```

Arguments

dfs.id
HDFS object ID where the data is located.

lines
Number of lines to return as a sample. The default value is 1000 lines.

sep
The symbol used to separate fields in the Hadoop file. A comma (,) is the default separator.

Usage Notes

If the data originated in an R environment, then all metadata is extracted and all attributes are restored, including column names and data types. Otherwise, generic attribute names, like `val1` and `val2`, are assigned.

Return Value

A `data.frame` object with the sample data set, or `NULL` if the operation failed.

Example

This example displays the first three lines of the ontime_R file.

```
R> hdfs.sample("ontime_R", lines=3)
  YEAR MONTH MONTH2 DAYOFMONTH DAYOFMONTH2 DAYOFWEEK DEPTIME...
1 2000    12     NA         31          NA         7     1730...
2 2000    12     NA         31          NA         7     1752...
3 2000    12     NA         31          NA         7     1803...
```

hdfs.size

Returns the size in bytes of an object in HDFS.

Usage

```
hdfs.size(
        dfs.id)
```

Arguments

dfs.id
Object identifier in HDFS.

Usage Notes

Use this interface to determine, for instance, whether you can pull the contents of the entire HDFS file into local R memory or a local file, or if you can only sample the data while creating a prototype of your R calculation.

Return Value

Size in bytes of the object, or 0 if the object does not exist in HDFS

Example

This example returns a file size for ontime_R of 999,839 bytes.

```
R> hdfs.size("ontime_R")
[1] 999839
```

hdfs.upload

Copies a file from the local file system into HDFS.

Usage

```
hdfs.upload(
        filename,
        dfs.name,
        overwrite,
        split.size,
        header)
```

Arguments

filename
Name of a file in the local file system.

dfs.name
Name of the new directory in HDFS.

overwrite
Controls whether `db.name` can overwrite a directory with the same name. Set to `TRUE` to overwrite the directory, or `FALSE` to signal an error (default).

split.size
Maximum number of bytes in each part of the Hadoop file. (Optional)

header
Indicates whether the first line of the local file is a header containing column names. Set to `TRUE` if it has a header, or `FALSE` if it does not (default).

A header enables you to exact the column names and reference the data fields by name instead of by index in your MapReduce R scripts.

Usage Notes

This function provides the fastest and easiest way to copy a file into HDFS. If the file is larger than *`split.size`*, then Hadoop splits it into two or more parts. The new Hadoop file gets a unique object ID, and each part is named `part-0000`*`x`*. Hadoop automatically creates metadata for the file.

Return Value

HDFS object ID for the loaded data, or `NULL` if the copy failed.

See Also

`hdfs.download` on page 5-11
`hdfs.get` on page 5-13
`hdfs.put` on page 5-20

Example

This example uploads a file named ontime_s2000.dat into HDFS and shows the location of the file, which is stored in a variable named `ontime.dfs_File`.

```
R> ontime.dfs_File <- hdfs.upload('ontime_s2000.dat', dfs.name='ontime_File')
```

```
R> print(ontime.dfs_File)
[1] "/user/oracle/xq/ontime_File"
```

orhc.connect

Establishes a connection to Oracle Database.

Usage

```
orhc.connect(
        host,
        user,
        sid,
        passwd,
        port,
        secure,
        driver,
        silent)
```

Arguments

host
Host name or IP address of the server where Oracle Database is running.

user
Database user name.

passwd
Password for the database user.

sid
System ID (SID) for the Oracle Database instance.

port
Port number for the Oracle Database listener. The default value is 1521.

secure
Authentication setting for Oracle Database:

- `TRUE`: You must enter a database password each time you attempt to connect. (Default)
- `FALSE`: You enter a database password once. It is encrypted in memory and used every time a database connection is required.

driver
Driver used to connect to Oracle Database (optional). Sqoop is the default driver.

silent
`TRUE` to suppress the prompts for missing host, user, password, port, and SID values, or `FALSE` to see them (default).

Usage Notes

Use this function when your analysis requires access to data stored in an Oracle database or to return the results to the database.

With an Oracle Database Advanced Analytics license for Oracle R Enterprise and a connection to Oracle Database, you can work directly with the data stored in database tables and pass processed data frames to R calculations on Hadoop.

Return Value

TRUE for a successful and validated connection, or FALSE for a failed connection attempt

See Also

orhc.disconnect on page 5-31

Example

This example installs the OHRC library and connects to the local Oracle database:

```
R> library(ORHC)
Oracle R Connector for Hadoop
Hadoop is up and running.
R> orhc.connect("localhost", 'RQUSER", "orcl")
Connecting ORCH to RDBMS via [sqoop]
    Host: localhost
    Port: 1521
    SID: orcl
    User: RQUSER
Enter password for [RQUSER]: password
Connected.
[1] TRUE
```

orhc.disconnect

Disconnects the local R session from Oracle Database.

Usage

```
orhc.disconnect()
```

Usage Notes

No `orhc` functions work without a connection to Oracle Database.

You can use the return value of this function to reestablish a connection using `orhc.reconnect`.

Return Value

An Oracle Database connection object, or `NULL` if Oracle Database refuses to disconnect

See Also

`orhc.connect` on page 5-29
`orhc.reconnect` on page 5-32

Example

```
R> orhc.disconnect()
Disconnected.
```

orhc.reconnect

Reconnects to Oracle Database with the credentials previously returned by `orhc.disconnect`.

Usage

```
orhc.reconnect(dbcon)
```

Arguments

dbcon
Credentials previously returned by orhc.disconnect.

Usage Notes

Oracle R Connector for Hadoop preserves all user credentials and connection attributes, enabling you to reconnect to a previously disconnected session. Depending on the `orhc.connect` secure setting for the original connection, you may be prompted for a password. After reconnecting, you can continue data transfer operations between Oracle Database and HDFS.

Reconnecting to a session is faster than opening a new one, because reconnecting does not require extensive connectivity checks.

Return Value

`TRUE` for a successfully reestablished and validated connection, or `FALSE` for a failed attempt

See Also

`orhc.connect` on page 5-29
`orhc.disconnect` on page 5-31

orhc.which

Displays information about the current connection to Oracle Database, excluding the authentication credentials.

Usage

```
orhc.which()
```

Return Value

None

Usage Notes

This function is useful when connecting to multiple Oracle databases during your analysis task.

Example

This example describes a connection by RQUSER to the local Oracle database:

```
R> orhc.which()
Connected to RDBMS via [sqoop]
    Host: localhost
    Port: 1521
    SID: orcl
    User: RQUSER
```

Index