

Oracle® Big Data Connectors

User's Guide

Release 2 (2.3)

E49333-02

November 2013

Describes installation and use of Oracle Big Data Connectors: Oracle SQL Connector for Hadoop Distributed File System, Oracle Loader for Hadoop, Oracle Data Integrator Application Adapter for Hadoop, Oracle XQuery for Hadoop, and Oracle R Advanced Analytics for Hadoop.

Oracle Big Data Connectors User's Guide, Release 2 (2.3)

E49333-02

Copyright © 2011, 2013, 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

Preface	xi
Audience	xi
Documentation Accessibility	xi
Related Documents	xi
Text Conventions	xii
Syntax Conventions	xii
Changes in This Release for Oracle Big Data Connectors User's Guide	xiii
Changes in Oracle Big Data Connectors Release 2 (2.3)	xiii
Changes in Oracle Big Data Connectors Release 2 (2.2)	xiv
Changes in Oracle Big Data Connectors Release 2 (2.0)	xiv
1 Getting Started with Oracle Big Data Connectors	
About Oracle Big Data Connectors	1-1
Big Data Concepts and Technologies	1-2
What is MapReduce?	1-2
What is Apache Hadoop?	1-3
Downloading the Oracle Big Data Connectors Software	1-3
Oracle SQL Connector for Hadoop Distributed File System Setup	1-4
Software Requirements	1-4
Installing and Configuring a Hadoop Client on the Oracle Database System	1-5
Installing Oracle SQL Connector for HDFS	1-6
Providing Support for Hive Tables	1-8
Granting User Privileges in Oracle Database	1-10
Setting Up User Accounts on the Oracle Database System	1-11
Using Oracle SQL Connector for HDFS on a Secure Hadoop Cluster	1-11
Oracle Loader for Hadoop Setup	1-12
Software Requirements	1-12
Installing Oracle Loader for Hadoop	1-12
Providing Support for Offline Database Mode	1-13
Using Oracle Loader for Hadoop on a Secure Hadoop Cluster	1-14
Oracle Data Integrator Application Adapter for Hadoop Setup	1-14
System Requirements and Certifications	1-14
Technology-Specific Requirements	1-15
Location of Oracle Data Integrator Application Adapter for Hadoop	1-15

Setting Up the Topology	1-15
Oracle XQuery for Hadoop Setup	1-15
Software Requirements	1-15
Installing Oracle XQuery for Hadoop.....	1-15
Troubleshooting the File Paths.....	1-16
Oracle R Advanced Analytics for Hadoop Setup.....	1-17
Installing the Software on Hadoop.....	1-17
Installing Additional R Packages.....	1-20
Providing Remote Client Access to R Users.....	1-22

2 Oracle SQL Connector for Hadoop Distributed File System

About Oracle SQL Connector for HDFS.....	2-1
Getting Started With Oracle SQL Connector for HDFS	2-2
Configuring Your System for Oracle SQL Connector for HDFS.....	2-5
Using the ExternalTable Command-Line Tool.....	2-6
About ExternalTable	2-6
ExternalTable Command-Line Tool Syntax	2-6
Creating External Tables	2-8
Creating External Tables with the ExternalTable Tool.....	2-8
Creating External Tables from Data Pump Format Files	2-8
Creating External Tables from Hive Tables	2-10
Creating External Tables from Delimited Text Files.....	2-13
Creating External Tables in SQL.....	2-16
Publishing the HDFS Data Paths	2-16
Listing Location File Metadata and Contents	2-17
Describing External Tables	2-18
More About External Tables Generated by the ExternalTable Tool.....	2-18
About Configurable Column Mappings.....	2-18
What Are Location Files?	2-20
Enabling Parallel Processing	2-20
Location File Management	2-20
Location File Names	2-21
Configuring Oracle SQL Connector for HDFS	2-21
Creating a Configuration File.....	2-21
Oracle SQL Connector for HDFS Configuration Property Reference	2-22
Performance Tips for Querying Data in HDFS.....	2-31

3 Oracle Loader for Hadoop

What Is Oracle Loader for Hadoop?	3-1
About the Modes of Operation.....	3-2
Online Database Mode	3-2
Offline Database Mode.....	3-3
Getting Started With Oracle Loader for Hadoop	3-3
Creating the Target Table	3-5
Supported Data Types for Target Tables.....	3-5
Supported Partitioning Strategies for Target Tables.....	3-5
Creating a Job Configuration File	3-6

About the Target Table Metadata	3-8
Providing the Connection Details for Online Database Mode	3-8
Generating the Target Table Metadata for Offline Database Mode	3-8
About Input Formats	3-10
Delimited Text Input Format.....	3-11
Complex Text Input Formats.....	3-12
Hive Table Input Format.....	3-12
Avro Input Format.....	3-13
Oracle NoSQL Database Input Format	3-13
Custom Input Formats	3-14
Mapping Input Fields to Target Table Columns	3-15
Automatic Mapping.....	3-15
Manual Mapping.....	3-15
Converting a Loader Map File	3-15
About Output Formats	3-17
JDBC Output Format	3-17
Oracle OCI Direct Path Output Format	3-18
Delimited Text Output Format	3-19
Oracle Data Pump Output Format	3-20
Running a Loader Job	3-21
Specifying Hive Input Format JAR Files	3-22
Specifying Oracle NoSQL Database Input Format JAR Files	3-22
Job Reporting	3-22
Handling Rejected Records	3-22
Logging Rejected Records in Bad Files	3-23
Setting a Job Reject Limit	3-23
Balancing Loads When Loading Data into Partitioned Tables	3-23
Using the Sampling Feature	3-23
Tuning Load Balancing	3-23
Tuning Sampling Behavior.....	3-24
When Does Oracle Loader for Hadoop Use the Sampler's Partitioning Scheme?	3-24
Resolving Memory Issues	3-24
What Happens When a Sampling Feature Property Has an Invalid Value?	3-25
Optimizing Communications Between Oracle Engineered Systems	3-25
Oracle Loader for Hadoop Configuration Property Reference	3-25
Third-Party Licenses for Bundled Software	3-40
Apache Licensed Code	3-40
Apache Avro 1.7.3	3-43
Apache Commons Mathematics Library 2.2.....	3-44
Jackson JSON 1.8.8	3-44

4 Oracle Data Integrator Application Adapter for Hadoop

Introduction	4-1
Concepts	4-1
Knowledge Modules.....	4-2
Security	4-2
Setting Up the Topology	4-3

Setting Up File Data Sources	4-3
Setting Up Hive Data Sources	4-3
Setting Up the Oracle Data Integrator Agent to Execute Hadoop Jobs	4-5
Configuring Oracle Data Integrator Studio for Executing Hadoop Jobs on the Local Agent	4-6
Setting Up an Integration Project	4-6
Creating an Oracle Data Integrator Model from a Reverse-Engineered Hive Model	4-7
Creating a Model.....	4-7
Reverse Engineering Hive Tables.....	4-7
Designing the Interface	4-8
Loading Data from Files into Hive	4-8
Validating and Transforming Data Within Hive	4-9
Loading Data into an Oracle Database from Hive and HDFS.....	4-10

5 Using Oracle XQuery for Hadoop

What Is Oracle XQuery for Hadoop?	5-1
Getting Started With Oracle XQuery for Hadoop	5-2
Basic Steps	5-2
Example: Hello World!.....	5-3
About the Adapters	5-3
About the Oracle XQuery for Hadoop Functions	5-4
About the Avro File Adapter.....	5-4
About the Oracle Database Adapter	5-4
About the Oracle NoSQL Database Adapter	5-4
About the Sequence File Adapter	5-5
About the Text File Adapter	5-5
About the XML File Adapter.....	5-5
About Other Modules for Use With Oracle XQuery for Hadoop.....	5-5
Creating an XQuery Transformation	5-6
XQuery Transformation Requirements	5-6
About XQuery Language Support.....	5-7
Accessing Data in the Hadoop Distributed Cache.....	5-7
Calling Custom Java Functions from XQuery	5-7
Accessing User-Defined XQuery Library Modules and XML Schemas	5-8
XQuery Transformation Examples.....	5-8
Running a Query	5-13
Oracle XQuery for Hadoop Options	5-13
Generic Options.....	5-13
About Running Queries Locally	5-14
Oracle XQuery for Hadoop Configuration Properties	5-14
Third-Party Licenses for Bundled Software	5-16
Apache Licensed Code	5-16
ANTLR 3.2.....	5-16
Apache Ant 1.7.1	5-17
Apache Avro 1.7.3, 1.7.4.....	5-18
Apache Xerces.....	5-21
Apache XMLBeans 2.5.....	5-21
Jackson 1.8.8	5-22

6 Oracle XQuery for Hadoop Reference

Avro File Adapter	6-2
Built-in Functions for Reading Avro Files.....	6-3
Custom Functions for Reading Avro Container Files.....	6-5
Custom Functions for Writing Avro Files.....	6-7
About Converting Values Between Avro and XML.....	6-10
Oracle Database Adapter	6-19
Custom Functions for Writing to Oracle Database.....	6-20
%oracle-property Annotations and Corresponding Oracle Loader for Hadoop Configuration Properties	6-26
Oracle NoSQL Database Adapter	6-29
Prerequisites for Using the Oracle NoSQL Database Adapter.....	6-30
Built-in Functions for Reading from and Writing to Oracle NoSQL Database.....	6-31
Custom Functions for Reading Values from Oracle NoSQL Database.....	6-38
Custom Functions for Retrieving Single Values from Oracle NoSQL Database.....	6-41
Custom Functions for Writing to Oracle NoSQL Database.....	6-43
Oracle NoSQL Database Adapter Configuration Properties.....	6-44
Sequence File Adapter	6-47
Built-in Functions for Reading and Writing Sequence Files.....	6-48
Custom Functions for Reading Sequence Files.....	6-54
Custom Functions for Writing Sequence Files.....	6-56
Text File Adapter	6-58
Built-in Functions for Reading and Writing Text Files.....	6-59
Custom Functions for Reading Text Files.....	6-63
Custom Functions for Writing Text Files.....	6-65
XML File Adapter	6-68
Built-in Functions for Reading XML Files.....	6-69
Custom Functions for Reading XML Files.....	6-72
JSON Module	6-75
Built-in Functions for Reading JSON.....	6-76
Utility Module	6-79
Duration, Date, and Time Functions.....	6-80
String Functions.....	6-84
Hadoop Module	6-86
Serialization Annotations	6-87

7 Oracle XML Extensions for Hive

What are the XML Extensions for Apache Hive?	7-1
Using the Hive Extensions	7-2
Creating XML Tables	7-2
Hive CREATE TABLE Syntax for XML Tables.....	7-3
CREATE TABLE Examples.....	7-5
XML Function Library for Apache Hive	7-10
Online Documentation of Functions.....	7-11

About Hive Access to External Files	7-12
About Data Type Conversions.....	7-13
xml_query	7-14
xml_query_as_primitive	7-16
xml_exists	7-19
xml_table	7-21

8 Using Oracle R Advanced Analytics for Hadoop

About Oracle R Advanced Analytics for Hadoop	8-1
Access to HDFS Files	8-2
Access to Apache Hive	8-3
ORE Functions for Hive	8-3
Generic R Functions Supported in Hive	8-4
Support for Hive Data Types	8-5
Usage Notes for Hive Access.....	8-7
Example: Loading Hive Tables into Oracle R Advanced Analytics for Hadoop	8-7
Access to Oracle Database	8-8
Usage Notes for Oracle Database Access	8-8
Scenario for Using Oracle R Advanced Analytics for Hadoop with Oracle R Enterprise	8-8
Analytic Functions in Oracle R Advanced Analytics for Hadoop	8-9
ORCH mapred.config Class	8-10
Examples and Demos of Oracle R Advanced Analytics for Hadoop	8-11
Using the Demos	8-11
Using the Examples	8-19
Security Notes for Oracle R Advanced Analytics for Hadoop	8-30

9 ORCH Library Reference

Functions in Alphabetical Order	9-1
Functions by Category	9-2
Making Connections.....	9-3
Copying Data.....	9-3
Exploring Files.....	9-3
Writing MapReduce Functions	9-3
Debugging Scripts.....	9-3
Using Hive Data	9-4
Writing Analytical Functions	9-4
hadoop.exec	9-5
hadoop.run	9-8
hdfs.attach	9-11
hdfs.cd	9-13
hdfs.cp	9-14
hdfs.describe	9-15
hdfs.download	9-16
hdfs.exists	9-17
hdfs.get	9-18
hdfs.head	9-19
hdfs.id	9-20

hdfs.ls.....	9-21
hdfs.mkdir	9-22
hdfs.mv	9-23
hdfs.parts.....	9-24
hdfs.pull	9-25
hdfs.push.....	9-27
hdfs.put	9-29
hdfs.pwd	9-30
hdfs.rm.....	9-31
hdfs.rmdir	9-32
hdfs.root	9-33
hdfs.sample.....	9-34
hdfs.setroot	9-35
hdfs.size.....	9-36
hdfs.tail.....	9-37
hdfs.upload.....	9-38
is.hdfs.id.....	9-40
orch.connect.....	9-41
orch.connected	9-44
orch.dbcon.....	9-45
orch.dbg.lasterr	9-47
orch.dbg.off	9-48
orch.dbg.on	9-49
orch.dbg.output.....	9-50
orch.dbinfo	9-51
orch.disconnect	9-52
orch.dryrun	9-54
orch.export	9-55
orch.keyval.....	9-56
orch.keyvals.....	9-57
orch.pack	9-59
orch.reconnect	9-60
orch.temp.path	9-61
orch.unpack	9-62
orch.version	9-63

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 SQL Connector for Hadoop Distributed File System
- Oracle XQuery for Hadoop
- Oracle Data Integrator Application Adapter for Hadoop
- Oracle R Advanced Analytics for Hadoop

Audience

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

- Application developers
- Java programmers
- XQuery 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*

- *Oracle Big Data Appliance Software User's Guide.*

Text 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.
<i>italic</i>	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.

Syntax Conventions

The syntax in [Chapter 2](#) is presented in a simple variation of Backus-Naur Form (BNF) that uses the following symbols and conventions:

Symbol or Convention	Description
[]	Brackets enclose optional items.
{ }	Braces enclose a choice of items, only one of which is required.
	A vertical bar separates alternatives within brackets or braces.
...	Ellipses indicate that the preceding syntactic element can be repeated.
delimiters	Delimiters other than brackets, braces, and vertical bars must be entered as shown.

Changes in This Release for Oracle Big Data Connectors User's Guide

This preface lists changes in the *Oracle Big Data Connectors User's Guide*.

Changes in Oracle Big Data Connectors Release 2 (2.3)

The following are changes in *Oracle Big Data Connectors User's Guide* for Oracle Big Data Connectors Release 2 (2.3).

The name Oracle R Connector for Hadoop changed to Oracle R Advanced Analytics for Hadoop.

New Features

The following features are new in this release.

- **Oracle XQuery for Hadoop**

Oracle XQuery for Hadoop is a transformation engine for semi-structured data stored in Apache Hadoop. It runs transformations expressed in the XQuery language by translating them into a series of MapReduce jobs, which are executed in parallel on the Hadoop cluster.

See [Part III, "Oracle XQuery for Hadoop"](#)

- **Oracle SQL Connector for HDFS**

- You can now override the default data type mappings by setting various configuration properties.

See ["About Configurable Column Mappings"](#) on page 2-18.

- Oracle SQL Connector for HDFS supports extended data types (VARCHAR2, NVARCHAR, and CLOB for Apache Hive and text data sources.

See ["oracle.hadoop.exctab.colMap.columnLength"](#) on page 2-22. and ["oracle.hadoop.exctab.colMap.columnType"](#) on page 2-23.

- The `--output` option to the `hadoop` command can write the generated DDL to a file.

See ["Using the ExternalTable Command-Line Tool"](#) on page 2-6.

- **Oracle Loader for Hadoop**

Configuration parameters replace loader maps. A utility is provided to convert a loader map file into a configuration file.

See ["Converting a Loader Map File"](#) on page 3-15.

Other Changes

The following are additional changes in the release:

- **Oracle R Advanced Analytics for Hadoop**
Two additional, dependent ORE packages must be installed on client nodes.
See "[Oracle R Advanced Analytics for Hadoop Setup](#)" on page 1-17.

Changes in Oracle Big Data Connectors Release 2 (2.2)

The following are changes in *Oracle Big Data Connectors User's Guide* for Oracle Big Data Connectors Release 2 (2.2).

New Features

The following features are new in this release.

- **Oracle SQL Connector for Hadoop Distributed File System**
Supports the Apache Hive decimal data type.
- **Oracle Loader for Hadoop**
Supports Hive 0.10.0.

Deprecated Features

The following features are deprecated in this release, and may be desupported in a future release

- **Oracle Loader for Hadoop**
 - `oracle.hadoop.loader.libjars`
 - `oracle.hadoop.loader.sharedlibs`

Other Changes

The following are additional changes in the release:

- **Oracle Loader for Hadoop**
The file names of the two kits in the installation zip archive have changed to the following format:
 - `oraloader-version-h1.x86_64.zip` for CDH4
 - `oraloader-version-h2.x86_64.zip` for Apache Hadoop 0.20.2 and CDH3

Changes in Oracle Big Data Connectors Release 2 (2.0)

The following are changes in *Oracle Big Data Connectors User's Guide* for Oracle Big Data Connectors Release 2 (2.0).

New Features

Oracle Big Data Connectors support Cloudera's Distribution including Apache Hadoop version 4 (CDH4). For other supported platforms, see the individual connectors in [Chapter 1](#).

The name of Oracle Direct Connector for Hadoop Distributed File System changed to Oracle SQL Connector for Hadoop Distributed File System.

- **Oracle SQL Connector for Hadoop Distributed File System**

- Automatic creation of Oracle Database external tables from Hive tables, Data Pump files, or delimited text files.
- Management of location files.

See [Chapter 2](#).

- **Oracle Loader for Hadoop**

- Support for Sockets Direct Protocol (SDP) for direct path loads
- Support for secondary sort on user-specified columns
- New input formats for regular expressions and Oracle NoSQL Database. The Avro record InputFormat is supported code instead of sample code.
- Simplified date format specification
- New reject limit threshold
- Improved job reporting and diagnostics

See [Chapter 3](#).

- **Oracle R Advanced Analytics for Hadoop**

Several analytic algorithms are now available: linear regression, neural networks for prediction, matrix completion using low rank matrix factorization, clustering, and nonnegative matrix factorization.

Oracle R Advanced Analytics for Hadoop supports Hive data sources in addition to HDFS files.

Oracle R Advanced Analytics for Hadoop can move data between HDFS and Oracle Database. Oracle R Enterprise is not required for this basic transfer of data.

The following functions are new in this release:

```
as.ore.*
hadoop.jobs
hdfs.head
hdfs.tail
is.ore.*
orch.connected
orch.dbg.lasterr
orch.evaluate
orch.export.fit
orch.lm
orch.lmf
orch.neural
orch.nmf
orch.nmf.NMFalgo
orch.temp.path
ore.*
predict.orch.lm
print.summary.orch.lm
summary.orch.lm
```

See [Chapter 8](#).

Deprecated Features

The following features are deprecated in this release, and may be desupported in a future release:

- **Oracle SQL Connector for Hadoop Distributed File System**

- **Location file format (version 1):** Existing external tables with content published using Oracle Direct Connector for HDFS version 1 must be republished using Oracle SQL Connector for HDFS version 2, because of incompatible changes to the location file format.

When Oracle SQL Connector for HDFS creates new location files, it does not delete the old location files.

See [Chapter 2](#).

- **oracle.hadoop.hdfs.exttab namespace (version 1):** Oracle SQL Connector for HDFS uses the following new namespaces for all configuration properties:

- * `oracle.hadoop.connection`: Oracle Database connection and wallet properties

- * `oracle.hadoop.exttab`: All other properties

See [Chapter 2](#).

- **HDFS_BIN_PATH directory:** The preprocessor directory name is now OSCH_BIN_PATH.

See "[Oracle SQL Connector for Hadoop Distributed File System Setup](#)" on page 1-4.

- **Oracle R Advanced Analytics for Hadoop**

- `keyval`: Use [orch.keyval](#) to generate key-value pairs.

- `orch.reconnect`: Use [orch.connect](#) to reconnect using a connection object returned by [orch.dbcon](#).

Desupported Features

The following features are no longer supported by Oracle.

- **Oracle Loader for Hadoop**

- `oracle.hadoop.loader.configuredCounters`

See [Chapter 3](#).

Other Changes

The following are additional changes in the release:

- **Oracle Loader for Hadoop**

The installation zip archive now contains two kits:

- `oraloader-2.0.0-1.x86_64.zip` for Apache Hadoop 0.20.2 and CDH3
- `oraloader-2.0.0-2.x86_64.zip` for CDH4

See "[Oracle Loader for Hadoop Setup](#)" on page 1-12.

Part I

Setup

Part I contains the following chapter:

- [Chapter 1, "Getting Started with Oracle Big Data Connectors"](#)

Getting Started with Oracle Big Data Connectors

This chapter describes the Oracle Big Data Connectors and provides installation instructions.

This chapter contains the following sections:

- [About Oracle Big Data Connectors](#)
- [Big Data Concepts and Technologies](#)
- [Downloading the Oracle Big Data Connectors Software](#)
- [Oracle SQL Connector for Hadoop Distributed File System Setup](#)
- [Oracle Loader for Hadoop Setup](#)
- [Oracle Data Integrator Application Adapter for Hadoop Setup](#)
- [Oracle XQuery for Hadoop Setup](#)
- [Oracle R Advanced Analytics for Hadoop Setup](#)

About Oracle Big Data Connectors

Oracle Big Data Connectors facilitate data access to data stored in an Apache Hadoop cluster. It can be licensed for use on either Oracle Big Data Appliance or a Hadoop cluster running on commodity hardware.

These are the connectors:

- **Oracle SQL Connector for Hadoop Distributed File System (previously Oracle Direct Connector for HDFS):** Enables an Oracle external table to access data stored in Hadoop Distributed File System (HDFS) files or a table in Apache Hive. The data can remain in HDFS or the Hive table, or it can be loaded into an Oracle database. Oracle SQL Connector for HDFS is a command-line utility that accepts generic command line arguments supported by the `org.apache.hadoop.util.Tool` interface. It also provides a preprocessor for Oracle external tables.
- **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 repartitions the data if necessary and transforms it into a database-ready format. It optionally sorts records by primary key or user-defined columns 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 `org.apache.hadoop.util.Tool` interface.

- **Oracle Data Integrator Application Adapter for Hadoop:** Extracts, transforms, and loads data from a Hadoop cluster into tables in an Oracle database, as defined using a graphical user interface.
- **Oracle XQuery for Hadoop:** Runs transformations expressed in the XQuery language by translating them into a series of MapReduce jobs, which are executed in parallel on the Hadoop cluster. The input data can be located in a file system accessible through the Hadoop File System API, such as the Hadoop Distributed File System (HDFS), or stored in Oracle NoSQL Database. Oracle XQuery for Hadoop can write the transformation results to HDFS, Oracle NoSQL Database, or Oracle Database.
- **Oracle R Advanced Analytics 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 Advanced Analytics 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 be installed in Oracle Database and either the Hadoop cluster or an external system set up as a Hadoop client for the cluster. Users may also need additional access privileges in Oracle Database.

See Also: My Oracle Support Information Center: Big Data Connectors (ID 1487399.2) and its related information centers.

Big Data Concepts and Technologies

Enterprises are seeing large amounts of data coming from multiple sources. Click-stream data in web logs, GPS tracking information, data from retail operations, sensor data, and multimedia streams are just a few examples of vast amounts of data that can be of tremendous value to an enterprise if analyzed. The unstructured and semi-structured information provided by raw data feeds is of little value in and of itself. The data must be processed to extract information of real value, which can then be stored and managed in the database. Analytics of this data along with the structured data in the database can provide new insights into the data and lead to substantial business benefits.

What is MapReduce?

MapReduce is a parallel programming model for processing data on a distributed system. It can process vast amounts of data in a timely manner and can scale linearly. It is particularly effective as a mechanism for batch processing of unstructured and semi-structured data. MapReduce abstracts lower level operations into computations over a set of keys and values.

A simplified definition of a MapReduce job is the successive alternation of two phases, the map phase and the reduce phase. Each map phase applies a transform function over each record in the input data to produce a set of records expressed as key-value pairs. The output from the map phase is input to the reduce phase. In the reduce phase, the map output records are sorted into key-value sets so that all records in a set have the same key value. A reducer function is applied to all the records in a set and a set of output records are produced as key-value pairs. The map phase is logically run in parallel over each record while the reduce phase is run in parallel over all key values.

Note: Oracle Big Data Connectors do not support the Yet Another Resource Negotiator (YARN) implementation of MapReduce.

What is Apache Hadoop?

Apache Hadoop is the software framework for the development and deployment of data processing jobs based on the MapReduce programming model. At the core, Hadoop provides a reliable shared storage and analysis system¹. Analysis is provided by MapReduce. Storage is provided by the Hadoop Distributed File System (HDFS), a shared storage system designed for MapReduce jobs.

The Hadoop ecosystem includes several other projects including Apache Avro, a data serialization system that is used by Oracle Loader for Hadoop.

Cloudera's Distribution including Apache Hadoop (CDH) is installed on Oracle Big Data Appliance. You can use Oracle Big Data Connectors on a Hadoop cluster running CDH or the equivalent Apache Hadoop components, as described in the setup instructions in this chapter.

See Also:

- For conceptual information about the Hadoop technologies, the following third-party publication:
Hadoop: The Definitive Guide, Third Edition by Tom White (O'Reilly Media Inc., 2012, ISBN: 978-1449311520).
- For information about Cloudera's Distribution including Apache Hadoop (CDH4), the Oracle Cloudera website at <http://oracle.cloudera.com/>
- For information about Apache Hadoop, the website at <http://hadoop.apache.org/>

Downloading the Oracle Big Data Connectors Software

You can download Oracle Big Data Connectors from Oracle Technology Network or Oracle Software Delivery Cloud.

To download from Oracle Technology Network:

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. Use any browser to visit this website:
<https://edelivery.oracle.com/>
2. Accept the Terms and Restrictions to see the Media Pack Search page.
3. Select the search terms:

¹ *Hadoop: The Definitive Guide, Third Edition* by Tom White (O'Reilly Media Inc., 2012, 978-1449311520).

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 (B65965-0x), and then click **Continue**.
6. Click **Download** for each connector to download a zip file containing the installation files.

Oracle SQL Connector for Hadoop Distributed File System Setup

You install and configure Oracle SQL Connector for Hadoop Distributed File System (HDFS) on the system where Oracle Database runs. If Hive tables are used as the data source, then you must also install and run Oracle SQL Connector for HDFS on a Hadoop client where users access Hive.

Oracle SQL Connector for HDFS is installed already on Oracle Big Data Appliance if it was configured for Oracle Big Data Connectors. This installation supports users who connect directly to Oracle Big Data Appliance to run their jobs.

This section contains the following topics:

- [Software Requirements](#)
- [Installing and Configuring a Hadoop Client on the Oracle Database System](#)
- [Installing Oracle SQL Connector for HDFS](#)
- [Providing Support for Hive Tables](#)
- [Granting User Privileges in Oracle Database](#)
- [Setting Up User Accounts on the Oracle Database System](#)
- [Using Oracle SQL Connector for HDFS on a Secure Hadoop Cluster](#)

Software Requirements

Oracle SQL Connector for HDFS requires the following software:

On the Hadoop cluster:

- Cloudera's Distribution including Apache Hadoop version 3 (CDH3) or version 4 (CDH4), Apache Hadoop 1.0 (formerly 0.20.2), or Apache Hadoop 1.1.
- Java Development Kit (JDK) 1.6_08 or later. Consult the distributor of your Hadoop software (Cloudera or Apache) for the recommended version.
- Hive 0.7.0, 0.7.1, 0.8.1, 0.9.0, or 0.10 (required for Hive table access, otherwise optional)

This software is already installed on Oracle Big Data Appliance.

On the Oracle Database system and Hadoop client systems:

- Oracle Database 12c, Oracle Database 11g release 2 (11.2.0.2 or 11.2.0.3), or Oracle Database 10g release 2 (10.2.0.5) for Linux.
- To support the Oracle Data Pump file format in Oracle Database release 11.2.0.2, an Oracle Database one-off patch. To download this patch, go to <http://support.oracle.com> and search for bug 14557588.
Release 11.2.0.3 and later releases do not require this patch.

- The same version of Hadoop as your Hadoop cluster: CDH3, CDH4, Apache Hadoop 1.0, or Apache Hadoop 1.1.

If you have a secure Hadoop cluster configured with Kerberos, then the Hadoop client on the database system must be set up to access a secure cluster. See "[Using Oracle SQL Connector for HDFS on a Secure Hadoop Cluster](#)" on page 1-11.

- The same version of JDK as your Hadoop cluster.

Installing and Configuring a Hadoop Client on the Oracle Database System

Oracle SQL Connector for HDFS works as a Hadoop client. You must install Hadoop on the Oracle Database system and minimally configure it for Hadoop client use only. You do not need to perform a full configuration of Hadoop on the Oracle Database system to run MapReduce jobs for Oracle SQL Connector for HDFS.

You can optionally set up additional Hadoop client systems by following these instructions.

To configure the Oracle Database system as a Hadoop client:

1. Install and configure the same version of CDH or Apache Hadoop on the Oracle Database system as on the Hadoop cluster. If you are using Oracle Big Data Appliance, then complete the procedures for providing remote client access in the *Oracle Big Data Appliance Software User's Guide*. Otherwise, follow the installation instructions provided by the distributor (Cloudera or Apache).

Note: Do not start Hadoop on the Oracle Database system. If it is running, then Oracle SQL Connector for HDFS attempts to use it instead of the Hadoop cluster. Oracle SQL Connector for HDFS just uses the Hadoop JAR files and the configuration files from the Hadoop cluster on the Oracle Database system.

2. If your cluster is secured with Kerberos, then obtain a Kerberos ticket:

```
> kinit
> password
```

3. Ensure that Oracle Database has access to HDFS:

- a. Log in to the system where Oracle Database is running by using the Oracle Database account.

- b. Open a Bash shell and enter this command:

```
hadoop fs -ls /user
```

You might need to add the directory containing the Hadoop executable file to the `PATH` environment variable. The default path for CDH is `/usr/bin`.

You should see the same list of directories that you see when you run the `hadoop fs` command directly on the Hadoop cluster. 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 Oracle Database system is now ready for use as a Hadoop client. No other Hadoop configuration steps are needed.

Installing Oracle SQL Connector for HDFS

Follow this procedure to install Oracle SQL Connector for HDFS.

To install Oracle SQL Connector for HDFS on the Oracle Database system:

1. Download the zip file to a directory on the system where Oracle Database runs.
2. Unpack the content of `oraosch-version.zip`.

```
$ unzip oraosch-2.3.0.zip
Archive:  oraosch-2.3.0.zip
  extracting:  orahdfs-2.3.0.zip
    inflating:  README.txt
```

3. Unpack `orahdfs-version.zip` into a permanent directory:

```
$ unzip orahdfs-2.3.0.zip
Archive:  orahdfs-2.3.0.zip
  creating:  orahdfs-2.3.0/
  creating:  orahdfs-2.3.0/bin/
  inflating:  orahdfs-2.3.0/bin/hdfs_stream
  .
  .
  .
```

The unzipped files have the structure shown in [Example 1-1](#).

4. Open the `orahdfs-2.3.0/bin/hdfs_stream` Bash shell script in a text editor, and make the changes indicated by the comments in the script, if necessary

The `hdfs_stream` script does not inherit any environment variable settings, and so they are set in the script if Oracle SQL Connector for HDFS needs them:

- **PATH:** If the `hadoop` script is not in `/usr/bin:bin` (the path initially set in `hdfs_stream`), then add the Hadoop bin directory, such as `/usr/lib/hadoop/bin`.
- **JAVA_HOME:** If Hadoop does not detect Java, then set this variable to the Java installation directory. For example, `/usr/bin/java`.
- **OSCH_HOME:** If you moved the script from the `orahdfs-version/bin` subdirectory, then set this variable to the full path of the `orahdfs-2.3.0` directory, which was created in Step 3. Otherwise, Oracle SQL Connector for HDFS detects its location automatically.

See the comments in the script for more information about these environment variables.

The `hdfs_stream` script is the preprocessor for the Oracle Database external table created by Oracle SQL Connector for HDFS.

5. If your cluster is secured with Kerberos, then obtain a Kerberos ticket:

```
> kinit
> password
```

6. Run `hdfs_stream` from the Oracle SQL Connector for HDFS `/bin` directory. You should see this usage information:

```
$ ./hdfs_stream
Usage: hdfs_stream locationFile
```

If you do not see the usage statement, then ensure that the operating system user that Oracle Database is running under (such as `oracle`) has the following permissions:

- Read and execute permissions on the `hdfs_stream` script:

```
$ ls -l OSCH_HOME/bin/hdfs_stream
-rwxr-xr-x 1 oracle oinstall Nov 27 15:51 hdfs_stream
```

- Read permission on `orahdfs.jar`.

```
$ ls -l OSCH_HOME/jlib/orahdfs.jar
-rwxr-xr-x 1 oracle oinstall Nov 27 15:51 orahdfs.jar
```

If you do not see these permissions, then enter a `chmod` command to fix them, for example:

```
$ chmod 755 OSCH_HOME/bin/hdfs_stream
```

In the previous commands, `OSCH_HOME` represents the Oracle SQL Connector for HDFS home directory.

7. Log in to Oracle Database and create a database directory for the `orahdfs-version/bin` directory where `hdfs_stream` resides. In this example, Oracle SQL Connector for HDFS is installed in `/etc`:

```
SQL> CREATE OR REPLACE DIRECTORY osch_bin_path AS '/etc/orahdfs-2.3.0/bin';
```

8. If you plan to access only data stored in HDFS and Data Pump format files, then you are done. If you also plan to access Hive tables, then you must also install Hive software on the Hadoop client, as described in "[Providing Support for Hive Tables](#)" on page 1-8.

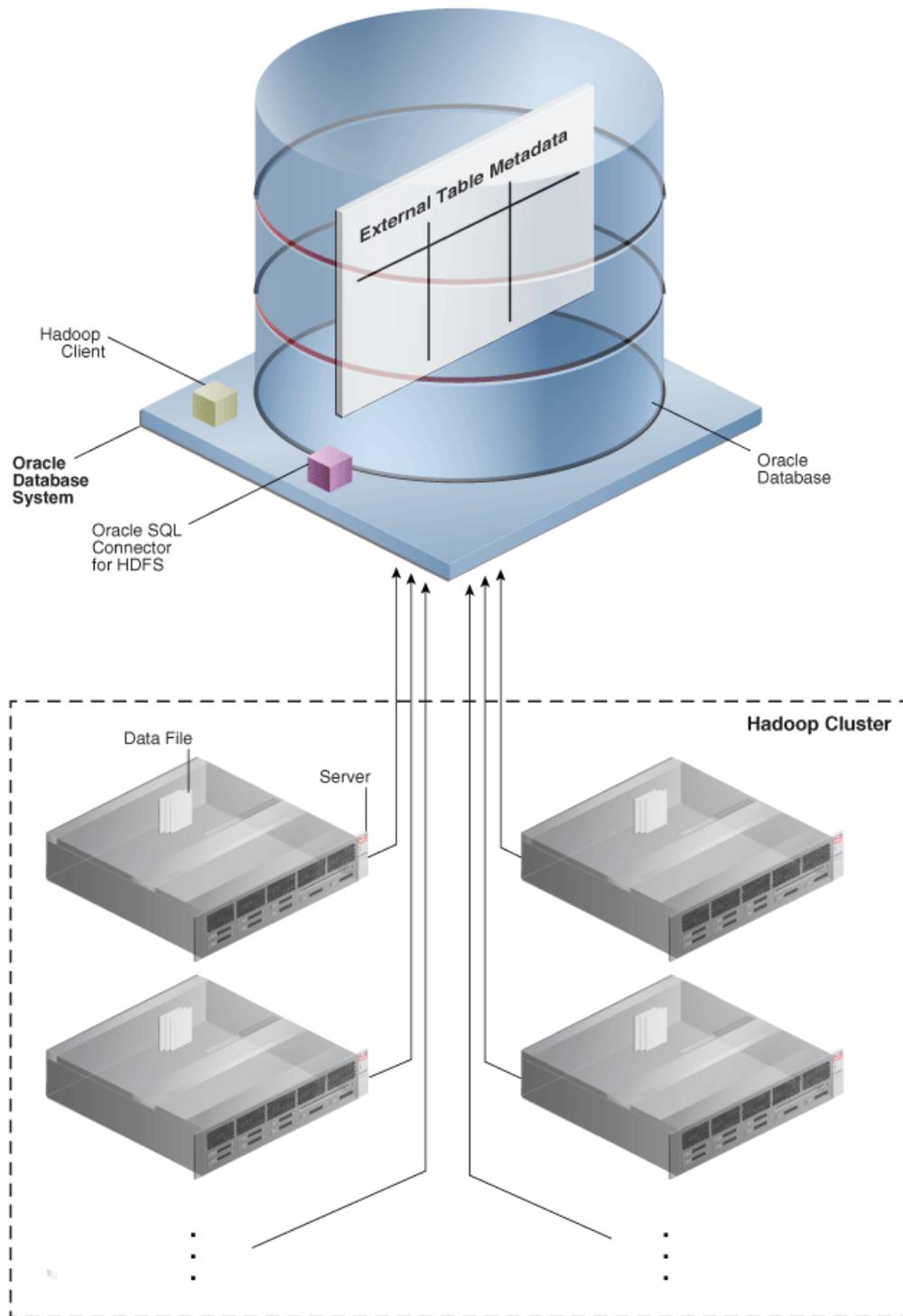
The unzipped files have the structure shown in [Example 1–1](#).

Example 1–1 Structure of the `orahdfs` Directory

```
orahdfs-version
bin/
  hdfs_stream
doc/
  README.txt
jlib/
  ojdbc6.jar
  ora-hadoop-common.jar
  oraclepki.jar
  orahdfs.jar
  osdt_cert.jar
  osdt_core.jar
log/
```

[Figure 1–1](#) illustrates shows the flow of data and the components locations.

Figure 1-1 Oracle SQL Connector for HDFS Installation for HDFS and Data Pump Files



Providing Support for Hive Tables

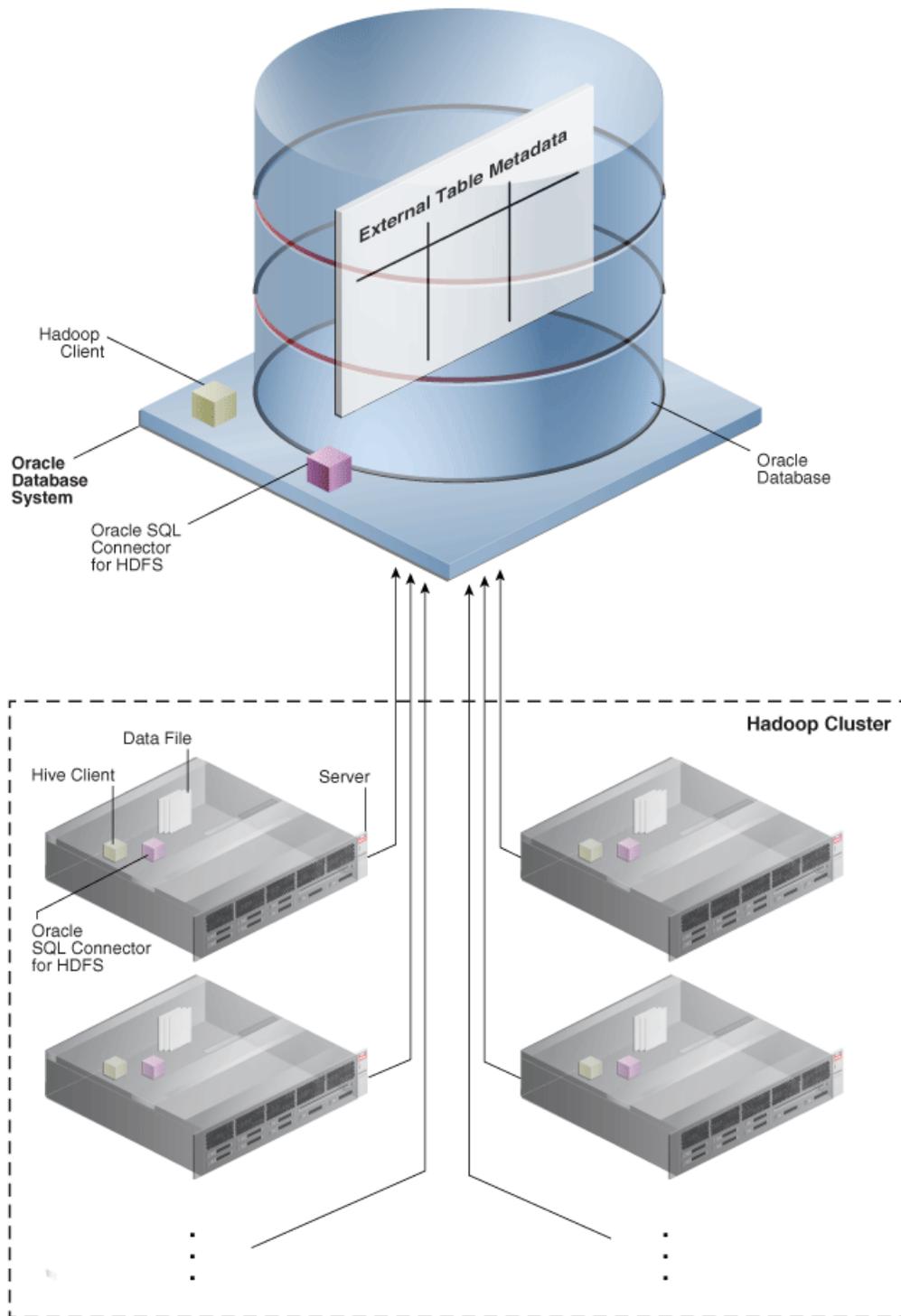
To support access to Hive tables, complete the following procedure on a system set up as a Hive client for the cluster, where users access HDFS and Hive. If you only plan to access HDFS and Data Pump format files, then you can omit this procedure.

To support Hive tables from a Hive client system:

1. Download or copy the `orahdfs-version.zip` file to the system where Hive is installed.
2. Unzip `orahdfs-version.zip` into a directory.
3. Add the Hive JAR files and the Hive conf directory to the `HADOOP_CLASSPATH` environment variable. To avoid JAR conflicts among the various Hadoop products, Oracle recommends that you set `HADOOP_CLASSPATH` in the local shell initialization script of Oracle SQL Connector for HDFS users instead of making a global change to `HADOOP_CLASSPATH`.

[Figure 1-2](#) illustrates the flow of data and the component locations.

Figure 1–2 Oracle SQL Connector for HDFS Installation for Hive Tables



Granting User Privileges in Oracle Database

Oracle Database users require these privileges when using Oracle SQL Connector for HDFS to create external tables:

- CREATE SESSION

- CREATE TABLE
- EXECUTE on the UTL_FILE PL/SQL package
- READ and EXECUTE on the OSCH_BIN_PATH directory created during the installation of Oracle SQL Connector for HDFS. Do not grant write access to anyone. Grant EXECUTE only to those who intend to use Oracle SQL Connector for HDFS.
- READ and WRITE on a database directory for storing external tables, or the CREATE ANY DIRECTORY system privilege.
- A tablespace and quota for copying data into the Oracle database. Optional.

[Example 1–2](#) shows the SQL commands granting these privileges to HDFSUSER.

Example 1–2 Granting Users Access to Oracle SQL Connector for HDFS

```
CONNECT / AS sysdba;
CREATE USER hdfsuser IDENTIFIED BY password
  DEFAULT TABLESPACE hdfsdata
  QUOTA UNLIMITED ON hdfsdata;
GRANT CREATE SESSION, CREATE TABLE TO hdfsuser;
GRANT EXECUTE ON sys.utl_file TO hdfsuser;
GRANT READ, EXECUTE ON DIRECTORY osch_bin_path TO hdfsuser;
GRANT READ, WRITE ON DIRECTORY external_table_dir TO hdfsuser;
```

Note: To query an external table that uses Oracle SQL Connector for HDFS, users only need the SELECT privilege on the table.

Setting Up User Accounts on the Oracle Database System

To create external tables for HDFS and Data Pump format files, users can log in to either the Oracle Database system or another system set up as a Hadoop client.

You can set up an account on these systems the same as you would for any other operating system user. HADOOP_CLASSPATH must include *path/orahdfs-2.3.0/jlib/**. You can add this setting to the shell profile as part of this installation procedure, or users can set it themselves. The following example alters HADOOP_CLASSPATH in the Bash shell where Oracle SQL Connector for HDFS is installed in /usr/bin:

```
export HADOOP_CLASSPATH="$HADOOP_CLASSPATH:/usr/bin/orahdfs-2.3.0/jlib/*
```

Using Oracle SQL Connector for HDFS on a Secure Hadoop Cluster

When users access an external table that was created using Oracle SQL Connector for HDFS, the external table acts like a Hadoop client running on the system where the Oracle database is running. It uses the identity of the operating system user where Oracle is installed.

A secure Hadoop cluster has Kerberos installed and configured to authenticate client activity. You must configure Oracle SQL Connector for HDFS for use with a Hadoop cluster secured by Kerberos.

For a user to authenticate using kinit:

- A Hadoop administrator must register the operating system user (such as oracle) and password in the Key Distribution Center (KDC) for the cluster.
- A system administrator for the Oracle Database system must configure /etc/krb5.conf and add a domain definition that refers to the KDC managed by the secure cluster.

These steps enable the operating system user to authenticate with the `kinit` utility before submitting Oracle SQL Connector for HDFS jobs. The `kinit` utility typically uses a Kerberos keytab file for authentication without an interactive prompt for a password. The system should run `kinit` on a regular basis, before letting the Kerberos ticket expire, to enable Oracle SQL Connector for HDFS to authenticate transparently.

Use `cron` or a similar utility to run `kinit`. Do not call `kinit` within the Oracle SQL Connector for HDFS preprocessor script (`hdfs_stream`), because it could trigger a high volume of concurrent calls to `kinit` and create internal Kerberos caching errors.

Note: Oracle Big Data Appliance configures Kerberos security automatically as a configuration option. For details about setting up client systems for a secure Oracle Big Data Appliance cluster, see *Oracle Big Data Appliance Software User's Guide*.

Oracle Loader for Hadoop Setup

Follow the instructions in these sections for setting up Oracle Loader for Hadoop:

- [Software Requirements](#)
- [Installing Oracle Loader for Hadoop](#)
- [Providing Support for Offline Database Mode](#)
- [Using Oracle Loader for Hadoop on a Secure Hadoop Cluster](#)

Software Requirements

Oracle Loader for Hadoop requires the following software:

- A target database system running one of the following:
 - Oracle Database 12c
 - Oracle Database 11g release 2 (11.2.0.3)
 - Oracle Database 11g release 2 (11.2.0.2) with required patch
 - Oracle Database 10g release 2 (10.2.0.5)

Note: To use Oracle Loader for Hadoop with Oracle Database 11g 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 version 3 (CDH3) or version 4 (CDH4), or Apache Hadoop 1.0 (formerly 0.20.2).
- Apache Hive 0.7.0, 0.8.1, 0.9.0, or 0.10.0 if you are loading data from Hive tables.

Installing Oracle Loader for Hadoop

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

To install Oracle Loader for Hadoop:

1. Unpack the content of `oraloader-version.x86_64.zip` into a directory on your Hadoop cluster or on a system configured as a Hadoop client. This archive contains two archives:
 - `oraloader-version-h1.x86_64.zip`: Use with CDH3 and Apache Hadoop 1.0
 - `oraloader-version-h2.x86_64.zip`: Use with CDH4
2. Unzip the appropriate archive into a directory on your Hadoop cluster. A directory named `oraloader-version-hn` is created with the following subdirectories:


```
doc
jlib
lib
examples
```
3. Create a variable named `OLH_HOME` and set it to the installation directory.
4. Add the following paths to the `HADOOP_CLASSPATH` variable:
 - For all installations:


```
$OLH_HOME/jlib/*
```
 - To support data loads from Hive tables:


```
path/orahdfs-2.3.0/jlib/*
/usr/lib/hive/lib/*
/etc/hive/conf
```

See "[Specifying Hive Input Format JAR Files](#)" on page 3-22.
 - To read data from Oracle NoSQL Database Release 2:


```
$KVHOME/lib/kvstore.jar
```

Providing Support for Offline Database Mode

In a typical installation, Oracle Loader for Hadoop can connect to the Oracle Database system from the Hadoop cluster or a Hadoop client. If this connection is impossible—for example, the systems are located on distinct networks—then you can use Oracle Loader for Hadoop in offline database mode. See "[About the Modes of Operation](#)" on page 3-2.

To support offline database mode, you must install Oracle Loader for Hadoop on two systems:

- The Hadoop cluster or a system set up as a Hadoop client, as described in "[Installing Oracle Loader for Hadoop](#)" on page 1-12.
- The Oracle Database system or a system with network access to Oracle Database, as described in the following procedure.

To support Oracle Loader for Hadoop in offline database mode:

1. Unpack the content of `oraloader-version.zip` into a directory on the Oracle Database system or a system with network access to Oracle Database.
2. Unzip the same version of the software as you installed on the Hadoop cluster, either for CDH3 or CDH4.
3. Create a variable named `OLH_HOME` and set it to the installation directory. This example uses the Bash shell syntax:

```
$ export OLH_HOME="/usr/bin/oraloader-2.3.0-h2/"
```

4. Add the Oracle Loader for Hadoop JAR files to the CLASSPATH environment variable. This example uses the Bash shell syntax:

```
$ export CLASSPATH=$CLASSPATH:$OLH_HOME/jlib/*
```

Using Oracle Loader for Hadoop on a Secure Hadoop Cluster

A secure Hadoop cluster has Kerberos installed and configured to authenticate client activity. An operating system user must be authenticated before initiating an Oracle Loader for Hadoop job to run on a secure Hadoop cluster. For authentication, the user must log in to the operating system where the job will be submitted and use the standard Kerberos `kinit` utility.

For a user to authenticate using `kinit`:

- A Hadoop administrator must register the operating system user and password in the Key Distribution Center (KDC) for the cluster.
- A system administrator for the client system, where the operating system user will initiate an Oracle Loader for Hadoop job, must configure `/etc/krb5.conf` and add a domain definition that refers to the KDC managed by the secure cluster.

Typically, the `kinit` utility obtains an authentication ticket that lasts several days. Subsequent Oracle Loader for Hadoop jobs authenticate transparently using the unexpired ticket.

Note: Oracle Big Data Appliance configures Kerberos security automatically as a configuration option. For details about setting up client systems for a secure Oracle Big Data Appliance cluster, see *Oracle Big Data Appliance Software User's Guide*.

Oracle Data Integrator Application Adapter for Hadoop Setup

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

- [System Requirements and Certifications](#)
- [Technology-Specific Requirements](#)
- [Location of Oracle Data Integrator Application Adapter for Hadoop](#)
- [Setting Up the Topology](#)

System Requirements and Certifications

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

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

Oracle Data Integrator Application Adapter for Hadoop requires 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 that you are installing.

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

<http://www.oracle.com/technetwork/middleware/data-integrator/overview/index.html>

Technology-Specific Requirements

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

<http://www.oracle.com/technetwork/middleware/data-integrator/overview/index.html>

Location of 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

To set up the topology, see [Chapter 4, "Oracle Data Integrator Application Adapter for Hadoop."](#)

Oracle XQuery for Hadoop Setup

You install and configure Oracle XQuery for Hadoop on the Hadoop cluster. If you are using Oracle Big Data Appliance, then the software is already installed.

The following topics describe the software installation:

- [Software Requirements](#)
- [Installing Oracle XQuery for Hadoop](#)
- [Troubleshooting the File Paths](#)

Software Requirements

Oracle Big Data Appliance 2.3 meets the following software requirements. However, if you are installing Oracle XQuery for Hadoop on a third-party cluster, then you must ensure that these components are installed.

- Java 6.x or 7.x
- Cloudera's Distribution including Apache Hadoop Version 3 (CDH 3.3 or above) or Version 4 (CDH 4.1.2 or above)
- Oracle NoSQL Database 2.x to support reading and writing to Oracle NoSQL Database
- Oracle Loader for Hadoop 2.3 to support writing tables in Oracle databases

Installing Oracle XQuery for Hadoop

Take the following steps to install Oracle XQuery for Hadoop.

To install Oracle XQuery for Hadoop:

1. Unpack the contents of `oxh-version.zip` into the installation directory:

```
$ unzip oxh-2.3.0-cdh-4.4.0.zip
Archive:  oxh-2.3.0-cdh-4.4.0.zip
  creating: oxh-2.3.0-cdh-4.4.0/
  creating: oxh-2.3.0-cdh-4.4.0/lib/
  inflating: oxh-2.3.0-cdh-4.4.0/lib/ant-launcher.jar
  inflating: oxh-2.3.0-cdh-4.4.0/lib/ant.jar
  inflating: oxh-2.3.0-cdh-4.4.0/lib/apache-xmlbeans.jar
  inflating: oxh-2.3.0-cdh-4.4.0/lib/avro-mapred-1.7.4-hadoop2.jar
  .
  .
  .
```

You can now run Oracle XQuery for Hadoop.

2. To support data loads into Oracle Database, install Oracle Loader for Hadoop as follows:
 - a. Unpack the content of `oraloader-version.x86_64.zip` into a directory on your Hadoop cluster or on a system configured as a Hadoop client. This archive contains two archives:
 - `oraloader-version-h1.x86_64.zip`: Use with CDH3 and Apache Hadoop 1.0
 - `oraloader-version-h2.x86_64.zip`: Use with CDH4
 - b. Unzip the appropriate archive into a directory on your Hadoop cluster.

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

```
doc
jlib
lib
examples
```
 - c. Create an environment variable named `OLH_HOME` and set it to the installation directory. Do not set `HADOOP_CLASSPATH`.
3. To support data loads into Oracle NoSQL Database, install it, and then set an environment variable named `KVHOME` to the Oracle NoSQL Database installation directory.

Troubleshooting the File Paths

If Oracle XQuery for Hadoop fails to find its own or third-party libraries when running queries, then first ensure that the environment variables are set, as described in ["Installing Oracle XQuery for Hadoop"](#) on page 1-15.

If they are set correctly, then you may need to edit `lib/oxh-lib.xml`. This file identifies the location of Oracle XQuery for Hadoop system JAR files and other libraries, such as Avro, Oracle Loader for Hadoop, and Oracle NoSQL Database.

If necessary, you can reference environment variables in this file as `${env.variable}`, such as `${env.OLH_HOME}`. You can also reference Hadoop properties as `${property}`, such as `${mapred.output.dir}`.

Oracle R Advanced Analytics for Hadoop Setup

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

Installing the Software on Hadoop

Oracle Big Data Appliance supports Oracle R Advanced Analytics for Hadoop without any additional software installation or configuration. However, you do need to verify whether certain R packages are installed. See "[Installing Additional R Packages](#)" on page 1-20.

However, to use Oracle R Advanced Analytics for Hadoop on any other Hadoop cluster, you must create the necessary environment.

Software Requirements for a Third-Party Hadoop Cluster

You must install several software components on a third-party Hadoop cluster to support Oracle R Advanced Analytics for Hadoop.

Install these components on third-party servers:

- Cloudera's Distribution including Apache Hadoop version 4 (CDH4) or Apache Hadoop 2.0.0, using MapReduce 1.
Complete the instructions provided by the distributor.
- Apache Hive 0.7.1 or 0.9.0
See "[Installing Hive on a Hadoop Cluster](#)" on page 1-18.
- Sqoop for the execution of functions that connect to Oracle Database. Oracle R Advanced Analytics for Hadoop does not require Sqoop to install or load.
See "[Installing Sqoop on a Hadoop Cluster](#)" on page 1-18.
- Mahout for the execution of (orch_lmf_mahout_als.R).
- Java Virtual Machine (JVM), preferably Java HotSpot Virtual Machine 6.
Complete the instructions provided at the download site at <http://www.oracle.com/technetwork/java/javase/downloads/index.html>
- Oracle R Distribution 2.15.1 with all base libraries on all nodes in the Hadoop cluster.
See "[Installing R on a Hadoop Cluster](#)" on page 1-19.
- The ORCH package on each R engine, which must exist on every node of the Hadoop cluster.
See "[Installing the ORCH Package on a Hadoop Cluster](#)" on page 1-19.
- Oracle Loader for Hadoop to support the OLH driver (optional). See "[Oracle Loader for Hadoop Setup](#)" on page 1-12.

Note: Do not set `HADOOP_HOME` on the Hadoop cluster. CDH4 does not need it, and it interferes with Oracle R Advanced Analytics for Hadoop when it checks the status of the JobTracker. This results in the error "Something is terribly wrong with Hadoop MapReduce."

If you must set `HADOOP_HOME` for another application, then also set `HADOOP_LIBEXEC_DIR` in the `/etc/bashrc` file. For example:

```
export HADOOP_LIBEXEC_DIR=/usr/lib/hadoop/libexec
```

Installing Sqoop on a Hadoop Cluster

Sqoop provides a SQL-like interface to Hadoop, which is a Java-based environment. Oracle R Advanced Analytics for Hadoop uses Sqoop for access to Oracle Database.

Note: Sqoop is required even when using Oracle Loader for Hadoop as a driver for loading data into Oracle Database. Sqoop performs additional functions, such as copying data from a database to HDFS and sending free-form queries to a database. The driver also uses Sqoop to perform operations that Oracle Loader for Hadoop does not support.

To install and configure Sqoop for use with Oracle Database:

1. Install Sqoop if it is not already installed on the server.

For Cloudera's Distribution including Apache Hadoop, see the Sqoop installation instructions in the *CDH Installation Guide* at

<http://oracle.cloudera.com/>

2. Download the appropriate Java Database Connectivity (JDBC) driver for Oracle Database from Oracle Technology Network at

<http://www.oracle.com/technetwork/database/features/jdbc/index-091264.html>

3. Copy the driver JAR file to `$SQOOP_HOME/lib`, which is a directory such as `/usr/lib/sqoop/lib`.
4. Provide Sqoop with the connection string to Oracle Database.

```
$ sqoop import --connect jdbc_connection_string
```

For example, `sqoop import --connect jdbc:oracle:thin@myhost:1521/orcl`.

Installing Hive on a Hadoop Cluster

Hive provides an alternative storage and retrieval mechanism to HDFS files through a querying language called HiveQL. Oracle R Advanced Analytics for Hadoop uses the data preparation and analysis features of HiveQL, while enabling you to use R language constructs.

To install Hive:

1. Follow the instructions provided by the distributor (Cloudera or Apache) for installing Hive.
2. Verify that the installation is working correctly:

```
$ hive -H
```

```

usage: hive
-d,--define <key=value>          Variable substitution to apply to hive
                                  commands. e.g. -d A=B or --define A=B
--database <databasename>        Specify the database to use
-e <quoted-query-string>         SQL from command line
-f <filename>                     SQL from files
-H,--help                         Print help information
-h <hostname>                     connecting to Hive Server on remote host
--hiveconf <property=value>     Use value for given property
--hivevar <key=value>           Variable substitution to apply to hive
                                  commands. e.g. --hivevar A=B
-i <filename>                     Initialization SQL file
-p <port>                          connecting to Hive Server on port number
-S,--silent                       Silent mode in interactive shell
-v,--verbose                       Verbose mode (echo executed SQL to the
                                  console)

```

3. If the command fails or you see warnings in the output, then fix the Hive installation.

Installing R on a Hadoop Cluster

You can download Oracle R Distribution 2.15.1 and get the installation instructions from the website at

<http://www.oracle.com/technetwork/indexes/downloads/r-distribution-1532464.html>

Alternatively, you can download R from a Comprehensive R Archive Network (CRAN) website at

<http://www.r-project.org>

Installing the ORCH Package on a Hadoop Cluster

ORCH is the name of the Oracle R Advanced Analytics for Hadoop package.

To install the ORCH package:

1. Set the environment variables for the supporting software:

```

$ export JAVA_HOME="/usr/lib/jdk6"
$ export R_HOME="/usr/lib64/R"
$ export SQOOP_HOME "/usr/lib/sqoop"

```

2. Unzip the downloaded file:

```

$ unzip orch-version.zip
Archive:  orch-linux-x86_64-2.3.0.zip
  extracting: ORCH2.3.0/ORCH_2.3.0_R_x86_64-unknown-linux-gnu.tar.gz
  inflating: ORCH2.3.0/ORCHcore_2.3.0_R_x86_64-unknown-linux-gnu.tar.gz
  inflating: ORCH2.3.0/OREbase_1.3.1_R_x86_64-unknown-linux-gnu.tar.gz
  inflating: ORCH2.3.0/OREstats_1.3.1_R_x86_64-unknown-linux-gnu.tar.gz
.
.
.

```

3. Change to the new directory:

```

$ cd ORCH2.3.0

```

4. Install the packages in the exact order shown here:

```
R --vanilla CMD INSTALL OREbase_1.4_R_x86_64-unknown-linux-gnu.tar.gz
R --vanilla CMD INSTALL OREstats_1.4_R_x86_64-unknown-linux-gnu.tar.gz
R --vanilla CMD INSTALL OREmodels_1.4_R_x86_64-unknown-linux-gnu.tar.gz
R --vanilla CMD INSTALL OREserver_1.4_R_x86_64-unknown-linux-gnu.tar.gz
R --vanilla CMD INSTALL ORCHcore_2.3.0_R_x86_64-unknown-linux-gnu.tar.gz
R --vanilla CMD INSTALL ORCHstats_2.3.0_R_x86_64-unknown-linux-gnu.tar.gz
R --vanilla CMD INSTALL ORCH_2.3.0_R_x86_64-unknown-linux-gnu.tar.gz
```

5. You must also install these packages on all other nodes of the cluster:

- OREbase
- OREmodels
- OREserver
- OREstats

The following examples use the `dcli` utility, which is available on Oracle Big Data Appliance but not on third-party clusters, to copy and install the OREserver package:

```
$ dcli -C -f OREserver_1.4_R_x86_64-unknown-linux-gnu.tar.gz -d /tmp/
OREserver_1.4_R_x86_64-unknown-linux-gnu.tar.gz

$ dcli -C " R --vanilla CMD INSTALL /tmp/OREserver_1.4_R_x86_
64-unknown-linux-gnu.tar.gz"
```

Installing Additional R Packages

Your Hadoop cluster must have `libpng-devel` installed on every node. If you are using a cluster running on commodity hardware, then you can follow the same basic procedures. However, you cannot use the `dcli` utility to replicate the commands across all nodes. See the *Oracle Big Data Appliance Owner's Guide* for the syntax of the `dcli` utility.

To install `libpng-devel`:

1. Log in as root to any node in your Hadoop cluster.
2. Check whether `libpng-devel` is already installed:

```
# dcli rpm -qi libpng-devel
bda1node01: package libpng-devel is not installed
bda1node02: package libpng-devel is not installed
.
.
.
```

If the package is already installed on all servers, then you can skip this procedure.

3. If you need a proxy server to go outside a firewall, then set the `HTTP_PROXY` environment variable:

```
# dcli export HTTP_PROXY="http://proxy.example.com"
```

4. Change to the yum directory:

```
# cd /etc/yum.repos.d
```

5. Download and configure the appropriate configuration file for your version of Linux:

For Enterprise Linux 5 (EL5):

- a. Download the yum configuration file:

```
# wget http://public-yum.oracle.com/public-yum-el5.repo
```

- b. Open public-yum-el5.repo in a text editor and make these changes:

Under el5_latest, set enabled=1

Under el5_addons, set enabled=1

- c. Save your changes and exit.

- d. Copy the file to the other Oracle Big Data Appliance servers:

```
# dcli -d /etc/yum.repos.d -f public-yum-el5.repo
```

For Oracle Linux 6 (OL6):

- a. Download the yum configuration file:

```
# wget http://public-yum.oracle.com/public-yum-ol6.repo
```

- b. Open public-yum-ol6.repo in a text editor and make these changes:

Under ol6_latest, set enabled=1

Under ol6_addons, set enabled=1

- c. Save your changes and exit.

- d. Copy the file to the other Oracle Big Data Appliance servers:

```
# dcli -d /etc/yum.repos.d -f public-yum-ol6.repo
```

6. Install the package on all servers:

```
# dcli yum -y install libpng-devel
```

```
bda1node01: Loaded plugins: rhnplugin, security
```

```
bda1node01: Repository 'bda' is missing name in configuration, using id
```

```
bda1node01: This system is not registered with ULN.
```

```
bda1node01: ULN support will be disabled.
```

```
bda1node01: http://bda1node01-master.us.oracle.com/bda/repodata/repomd.xml:
```

```
bda1node01: [Errno 14] HTTP Error 502: notresolvable
```

```
bda1node01: Trying other mirror.
```

```
.
```

```
.
```

```
.
```

```
bda1node01: Running Transaction
```

```
bda1node01: Installing      : libpng-devel                                1/2
```

```
bda1node01: Installing      : libpng-devel                                2/2
```

```
bda1node01: Installed:
```

```
bda1node01: libpng-devel.i386 2:1.2.10-17.el5_8  libpng-devel.x86_64
```

```
2:1.2.10-17.el5_8
```

```
bda1node01: Complete!
```

```
bda1node02: Loaded plugins: rhnplugin, security
```

```
.
```

```
.
```

```
.
```

7. Verify that the installation was successful on all servers:

```
# dcli rpm -qi libpng-devel
```

```
bda1node01: Name          : libpng-devel
```

```
Relocations: (not
```

```
relocatable)
```

```

bdalnode01: Version      : 1.2.10                      Vendor: Oracle
America
bdalnode01: Release      : 17.el5_8                      Build Date: Wed 25
Apr 2012 06:51:15 AM PDT
bdalnode01: Install Date: Tue 05 Feb 2013 11:41:14 AM PST  Build Host:
ca-build56.us.oracle.com
bdalnode01: Group        : Development/Libraries          Source RPM:
libpng-1.2.10-17.el5_8.src.rpm
bdalnode01: Size         : 482483                        License: zlib
bdalnode01: Signature    : DSA/SHA1, Wed 25 Apr 2012 06:51:41 AM PDT, Key ID
66ced3de1e5e0159
bdalnode01: URL          : http://www.libpng.org/pub/png/
bdalnode01: Summary      : Development tools for programs to manipulate PNG
image format files.
bdalnode01: Description :
bdalnode01: The libpng-devel package contains the header files and static
bdalnode01: libraries necessary for developing programs using the PNG (Portable
bdalnode01: Network Graphics) library.
.
.
.

```

Providing Remote Client Access to R Users

Whereas R users will run their programs as MapReduce jobs on the Hadoop cluster, they do not typically have individual accounts on that platform. Instead, an external Linux server provides remote access.

Software Requirements for Remote Client Access

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

- The same version of Hadoop as your Hadoop cluster; otherwise, unexpected issues and failures can occur
- The same version of Sqoop as your Hadoop cluster; required only to support copying data in and out of Oracle databases
- Mahout; required only for the `orch.ls` function with the Mahout ALS-WS algorithm
- The same version of the Java Development Kit (JDK) as your Hadoop cluster
- R distribution 2.15.1 with all base libraries
- ORCH R package

To provide access to database objects, you must have the Oracle Advanced Analytics option to Oracle Database. Then you can install this additional component on the Hadoop client:

- Oracle R Enterprise Client Packages

Configuring the Server as a Hadoop Client

You must install Hadoop on the client and minimally configure it for HDFS client use.

To install and configure Hadoop on the client system:

1. Install and configure CDH3 or Apache Hadoop 0.20.2 on the client system. This system can be the host for Oracle Database. If you are using Oracle Big Data Appliance, then complete the procedures for providing remote client access in the

Oracle Big Data Appliance Software User's Guide. Otherwise, follow the installation instructions provided by the distributor (Cloudera or Apache).

2. Log in to the client system as an R user.
3. Open a Bash shell and enter this Hadoop file system command:

```
$HADOOP_HOME/bin/hadoop fs -ls /user
```
4. If you see a list of files, then you are done. If not, then ensure that the Hadoop cluster is up and running. If that does not fix the problem, then you must debug your client Hadoop installation.

Installing Sqoop on a Hadoop Client

Complete the same procedures on the client system for installing and configuring Sqoop as those provided in "[Installing Sqoop on a Hadoop Cluster](#)" on page 1-18.

Installing R on a Hadoop Client

You can download R 2.13.2 and get the installation instructions from the Oracle R Distribution website at

<http://oss.oracle.com/ORD/>

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

You may also want to install RStudio Server to facilitate access by R users. See the RStudio website at

<http://rstudio.org/>

Installing the ORCH Package on a Hadoop Client

Complete the procedures on the client system for installing ORCH as described in "[Installing the Software on Hadoop](#)" on page 1-17.

Installing the Oracle R Enterprise Client Packages (Optional)

To support full access to Oracle Database using R, install the Oracle R Enterprise Release 1.3.1 or later client packages. Without them, Oracle R Advanced Analytics for Hadoop does not have access to the advanced statistical algorithms provided by Oracle R Enterprise.

See Also: *Oracle R Enterprise User's Guide* for information about installing R and Oracle R Enterprise

Part II

Oracle Database Connectors

This part contains the following chapters:

- [Chapter 2, "Oracle SQL Connector for Hadoop Distributed File System"](#)
- [Chapter 3, "Oracle Loader for Hadoop"](#)
- [Chapter 4, "Oracle Data Integrator Application Adapter for Hadoop"](#)

Oracle SQL Connector for Hadoop Distributed File System

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

This chapter contains the following sections:

- [About Oracle SQL Connector for HDFS](#)
- [Getting Started With Oracle SQL Connector for HDFS](#)
- [Configuring Your System for Oracle SQL Connector for HDFS](#)
- [Using the ExternalTable Command-Line Tool](#)
- [Creating External Tables](#)
- [Publishing the HDFS Data Paths](#)
- [Listing Location File Metadata and Contents](#)
- [Describing External Tables](#)
- [More About External Tables Generated by the ExternalTable Tool](#)
- [Configuring Oracle SQL Connector for HDFS](#)
- [Performance Tips for Querying Data in HDFS](#)

About Oracle SQL Connector for HDFS

Using Oracle SQL Connector for HDFS, you can use Oracle Database to access and analyze data residing in Apache Hadoop in these formats:

- Data Pump files in HDFS
- Delimited text files in HDFS
- Delimited text files in Apache Hive tables

For other file formats, such as JSON files, you can stage the input in Hive tables before using Oracle SQL Connector for HDFS. Partitioned Hive tables are not supported.

Oracle SQL Connector for HDFS uses external tables to provide Oracle Database with read access to Hive tables, and to delimited text files and Data Pump files in HDFS. An **external table** is an Oracle Database object that identifies the location of data outside of a database. Oracle Database accesses the data by using the metadata provided when the external table was created. By querying the external tables, you can access data stored in HDFS and Hive tables as if that data were stored in tables in an Oracle database.

To create an external table for this purpose, you use the `ExternalTable` command-line tool provided with Oracle SQL Connector for HDFS. You provide `ExternalTable` with information about the data source in Hadoop and about your schema in an Oracle Database. You provide this information either as options to the `ExternalTable` command or in an XML file.

When the external table is ready, you can query the data the same as any other database table. You can query and join data in HDFS or a Hive table with other database-resident data.

You can also perform bulk loads of data into Oracle database tables using SQL. You may prefer that the data resides in an Oracle database—all of it or just a selection—if it is queried routinely.

Getting Started With Oracle SQL Connector for HDFS

The following list identifies the basic steps that you take when using Oracle SQL Connector for HDFS.

1. The first time you use Oracle SQL Connector for HDFS, ensure that the software is installed and configured.
See "[Configuring Your System for Oracle SQL Connector for HDFS](#)" on page 2-5.
2. Log in to the appropriate system, either the Oracle Database system or a node in the Hadoop cluster. If you are connecting to a secure cluster, then you must run `kinit` to authenticate yourself.
See "[Configuring Your System for Oracle SQL Connector for HDFS](#)" on page 2-5.
3. Create an XML document describing the connections and the data source, unless you are providing these properties in the `ExternalTable` command.
See "[Describing External Tables](#)" on page 2-18.
4. Create a shell script containing an `ExternalTable` command.
See "[Using the ExternalTable Command-Line Tool](#)" on page 2-6.
5. Run the shell script.
6. If the job fails, then use the diagnostic messages in the output to identify and correct the error. Depending on how far the job progressed before failing, you may need to delete the table definition from the Oracle database before rerunning the script.
7. After the job succeeds, connect to Oracle Database as the owner of the external table. Query the table to ensure that the data is accessible.
8. If the data will be queried frequently, then you may want to load it into a database table. External tables do not have indexes or partitions.

[Example 2-1](#) illustrates these steps.

Example 2-1 Accessing HDFS Data Files from Oracle Database

```
$ cat moviefact_hdfs.sh
# Add environment variables
export OSCH_HOME="/u01/connectors/orahdfs-2.3.0"

hadoop jar $OSCH_HOME/jlib/orahdfs.jar \
  oracle.hadoop.exttab.ExternalTable \
  -conf /home/oracle/movies/moviefact_hdfs.xml \
```

```

        -createTable

$ cat moviefact_hdfs.xml
<?xml version="1.0"?>
<configuration>
  <property>
    <name>oracle.hadoop.exctab.tableName</name>
    <value>MOVIE_FACTS_EXT</value>
  </property>
  <property>
    <name>oracle.hadoop.exctab.locationFileCount</name>
    <value>4</value>
  </property>
  <property>
    <name>oracle.hadoop.exctab.dataPaths</name>
    <value>/user/oracle/moviework/data/part*</value>
  </property>
  <property>
    <name>oracle.hadoop.exctab.fieldTerminator</name>
    <value>\u0009</value>
  </property>
  <property>
    <name>oracle.hadoop.exctab.defaultDirectory</name>
    <value>MOVIEDEMO_DIR</value>
  </property>
  <property>
    <name>oracle.hadoop.exctab.columnNames</name>
    <value>CUST_ID,MOVIE_ID,GENRE_ID,TIME_ID,RECOMMENDED,ACTIVITY_
ID,RATING,SALES</value>
  </property>
  <property>
    <name>oracle.hadoop.exctab.colMap.TIME_ID.columnType</name>
    <value>TIMESTAMP</value>
  </property>
  <property>
    <name>oracle.hadoop.exctab.colMap.RECOMMENDED.columnType</name>
    <value>NUMBER</value>
  </property>
  <property>
    <name>oracle.hadoop.exctab.colMap.ACTIVITY_ID.columnType</name>
    <value>NUMBER</value>
  </property>
  <property>
    <name>oracle.hadoop.exctab.colMap.RATING.columnType</name>
    <value>NUMBER</value>
  </property>
  <property>
    <name>oracle.hadoop.exctab.colMap.SALES.columnType</name>
    <value>NUMBER</value>
  </property>
  <property>
    <name>oracle.hadoop.exctab.sourceType</name>
    <value>text</value>
  </property>
  <property>
    <name>oracle.hadoop.connection.url</name>
    <value>jdbc:oracle:thin:@localhost:1521:orcl</value>
  </property>
  <property>
    <name>oracle.hadoop.connection.user</name>

```

```

    <value>MOVIEDEMO</value>
  </property>
</configuration>

```

```
$ sh moviefact_hdfs.sh
```

```
Oracle SQL Connector for HDFS Release 2.3.0 - Production
```

```
Copyright (c) 2011, 2013, Oracle and/or its affiliates. All rights reserved.
```

```
[Enter Database Password: password]
```

```
The create table command succeeded.
```

```

CREATE TABLE "MOVIEDEMO"."MOVIE_FACTS_EXT"
(
  "CUST_ID"                VARCHAR2(4000),
  "MOVIE_ID"               VARCHAR2(4000),
  "GENRE_ID"               VARCHAR2(4000),
  "TIME_ID"                TIMESTAMP(9),
  "RECOMMENDED"           NUMBER,
  "ACTIVITY_ID"           NUMBER,
  "RATING"                 NUMBER,
  "SALES"                  NUMBER
)
ORGANIZATION EXTERNAL
(
  TYPE ORACLE_LOADER
  DEFAULT DIRECTORY "MOVIEDEMO_DIR"
  ACCESS PARAMETERS
  (
    RECORDS DELIMITED BY 0X'0A'
    CHARACTERSET AL32UTF8
    PREPROCESSOR "OSCH_BIN_PATH":'hdfs_stream'
    FIELDS TERMINATED BY 0X'09'
    MISSING FIELD VALUES ARE NULL
  (
    "CUST_ID" CHAR(4000),
    "MOVIE_ID" CHAR(4000),
    "GENRE_ID" CHAR(4000),
    "TIME_ID" CHAR,
    "RECOMMENDED" CHAR,
    "ACTIVITY_ID" CHAR,
    "RATING" CHAR,
    "SALES" CHAR
  )
  )
  )
  LOCATION
  (
    'osch-20131114064206-5250-1',
    'osch-20131114064206-5250-2',
    'osch-20131114064206-5250-3',
    'osch-20131114064206-5250-4'
  )
  )
) PARALLEL REJECT LIMIT UNLIMITED;

```

```
The following location files were created.
```

```
osch-20131114064206-5250-1 contains 1 URI, 12754882 bytes
```

```
12754882
```

```
hdfs://localhost.localdomain:8020/user/oracle/moviework/data/part-00001
```

```

osch-20131114064206-5250-2 contains 1 URI, 438 bytes

      438
hdfs://localhost.localdomain:8020/user/oracle/moviework/data/part-00002

osch-20131114064206-5250-3 contains 1 URI, 432 bytes

      432
hdfs://localhost.localdomain:8020/user/oracle/moviework/data/part-00003

osch-20131114064206-5250-4 contains 1 URI, 202 bytes

      202
hdfs://localhost.localdomain:8020/user/oracle/moviework/data/part-00004

$ sqlplus moviedemo

SQL*Plus: Release 12.1.0.1.0 Production on Fri Nov 15 09:24:18 2013

Copyright (c) 1982, 2013, Oracle. All rights reserved.

Enter password: password
Last Successful login time: Thu Nov 14 2013 18:42:01 -05:00

Connected to:
Oracle Database 12c Enterprise Edition Release 12.1.0.1.0 - 64bit Production
With the Partitioning, OLAP, Advanced Analytics and Real Application Testing
options

SQL> DESCRIBE movie_facts_ext;

```

Name	Null?	Type
CUST_ID		VARCHAR2(4000)
MOVIE_ID		VARCHAR2(4000)
GENRE_ID		VARCHAR2(4000)
TIME_ID		TIMESTAMP(9)
RECOMMENDED		NUMBER
ACTIVITY_ID		NUMBER
RATING		NUMBER
SALES		NUMBER

```

SQL> CREATE TABLE movie_facts AS SELECT * FROM movie_facts_ext;

Table created.

```

Configuring Your System for Oracle SQL Connector for HDFS

You can run Oracle SQL Connector for HDFS on either the Oracle Database system or the Hadoop cluster:

- For Hive sources, you must log in to either a node in the Hadoop cluster or a system set up as a Hadoop client for the cluster.
- For text and Data Pump format files, you can log in to either the Oracle Database system or a node in the Hadoop cluster.

Oracle SQL Connector for HDFS requires additions to the `HADOOP_CLASSPATH` environment variable on the system where you log in. Your system administrator may have set them up for you when creating your account, or may have left that task for you. See ["Setting Up User Accounts on the Oracle Database System"](#) on page 1-11.

Setting up the environment variables:

- Verify that `HADOOP_CLASSPATH` includes the path to the JAR files for Oracle SQL Connector for HDFS:

```
path/orahdfs-2.3.0/jlib/*
```

- If you are logged in to a Hadoop cluster with Hive data sources, then verify that `HADOOP_CLASSPATH` also includes the Hive JAR files and conf directory. For example:

```
/usr/lib/hive/lib/*  
/etc/hive/conf
```

- For your convenience, you can create an `OSCH_HOME` environment variable. The following is the Bash command for setting it on Oracle Big Data Appliance:

```
$ export OSCH_HOME="/opt/oracle/orahdfs-2.3.0"
```

See Also: ["Oracle SQL Connector for Hadoop Distributed File System Setup"](#) on page 1-4 for instructions for installing the software and setting up user accounts on both systems.

`OSCH_HOME/doc/README.txt` for information about known problems with Oracle SQL Connector for HDFS.

Using the ExternalTable Command-Line Tool

Oracle SQL Connector for HDFS provides a command-line tool named `ExternalTable`. This section describes the basic use of this tool. See ["Creating External Tables"](#) on page 2-8 for the command syntax that is specific to your data source format.

About ExternalTable

The `ExternalTable` tool uses the values of several properties to do the following tasks:

- Create an external table
- Populate the location files
- Publish location files to an existing external table
- List the location files
- Describe an external table

You can specify these property values in an XML document or individually on the command line. See ["Configuring Oracle SQL Connector for HDFS"](#) on page 2-21..

ExternalTable Command-Line Tool Syntax

This is the full syntax of the `ExternalTable` command-line tool:

```
hadoop jar OSCH_HOME/jlib/orahdfs.jar \  
oracle.hadoop.exttab.ExternalTable \  
[-conf config_file]... \  
[-D property=value]... \  
-createTable [--noexecute [--output filename.sql]]  
| -publish [--noexecute]  
| -listlocations [--details]  
| -getDDL
```

You can either create the `OSCH_HOME` environment variable or replace `OSCH_HOME` in the command syntax with the full path to the installation directory for Oracle SQL Connector for HDFS. On Oracle Big Data Appliance, this directory is:

```
/opt/oracle/orahdfs-version
```

For example, you might run the ExternalTable command-line tool with a command like this:

```
hadoop jar /opt/oracle/orahdfs-2.3.0/jlib/orahdfs.jar \
oracle.hadoop.exttab.ExternalTable \
.
.
.
```

Options

-conf *config_file*

Identifies the name of an XML configuration file containing properties needed by the command being executed. See ["Configuring Oracle SQL Connector for HDFS"](#) on page 2-21.

-D *property=value*

Assigns a value to a specific property.

-createTable [--noexecute [--output *filename*]]

Creates an external table definition and publishes the data URIs to the location files of the external table. The output report shows the DDL used to create the external table and lists the contents of the location files.

Oracle SQL Connector for HDFS also checks the database to ensure that the required database directories exist and that you have the necessary permissions.

Use the `--noexecute` option to see the execution plan of the command. The operation is not executed, but the report includes the details of the execution plan and any errors. The `--output` option writes the table DDL from the `-createTable` command to a file.

Oracle recommends that you first execute a `-createTable` command with `--noexecute`.

-publish [--noexecute]

Publishes the data URIs to the location files of an existing external table. Use this option after adding new data files, so that the existing external table can access them.

Use the `--noexecute` option to see the execution plan of the command. The operation is not executed, but the report shows the planned SQL `ALTER TABLE` command and location files. The report also shows any errors.

Oracle recommends that you first execute a `-publish` command with `--noexecute`.

See ["Publishing the HDFS Data Paths"](#) on page 2-16.

-listLocations [--details]

Shows the location file content as text. With the `--details` option, this command provides a detailed listing. See ["What Are Location Files?"](#) on page 2-20.

-getDDL

Prints the table definition of an existing external table. See ["Describing External Tables"](#) on page 2-18.

See Also: ["Syntax Conventions"](#) on page xii

Creating External Tables

You can create external tables automatically using the `ExternalTable` tool provided in Oracle SQL Connector for HDFS.

Creating External Tables with the ExternalTable Tool

To create an external table using the `ExternalTable` tool, follow the instructions for your data source:

- [Creating External Tables from Data Pump Format Files](#)
- [Creating External Tables from Hive Tables](#)
- [Creating External Tables from Delimited Text Files](#)

When the `ExternalTable -createTable` command finishes executing, the external table is ready for use. `ExternalTable` also manages the location files for the external table. See ["Location File Management"](#) on page 2-20.

To create external tables manually, follow the instructions in ["Creating External Tables in SQL"](#) on page 2-16.

ExternalTable Syntax for -createTable

Use the following syntax to create an external table and populate its location files:

```
hadoop jar OSCH_HOME/jlib/orahdfs.jar oracle.hadoop.exttab.ExternalTable \  
[-conf config_file]... \  
[-D property=value]... \  
-createTable [--noexecute]
```

See Also: ["ExternalTable Command-Line Tool Syntax"](#) on page 2-6

Creating External Tables from Data Pump Format Files

Oracle SQL Connector for HDFS supports only Data Pump files produced by Oracle Loader for Hadoop, and does not support generic Data Pump files produced by Oracle Utilities.

Oracle SQL Connector for HDFS creates the external table definition for Data Pump files by using the metadata from the Data Pump file header. It uses the `ORACLE_LOADER` access driver with the `preprocessor` access parameter. It also uses a special access parameter named `EXTERNAL_VARIABLE_DATA`, which enables `ORACLE_LOADER` to read the Data Pump format files generated by Oracle Loader for Hadoop.

Note: Oracle SQL Connector for HDFS requires a patch to Oracle Database 11.2.0.2 before the connector 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 14557588.

Release 11.2.0.3 and later releases do not require this patch.

Required Properties

These properties are required:

- `oracle.hadoop.exctab.tableName`
- `oracle.hadoop.exctab.defaultDirectory`
- `oracle.hadoop.exctab.dataPaths`
- `oracle.hadoop.exctab.sourceType=datapump`
- `oracle.hadoop.connection.url`
- `oracle.hadoop.connection.user`

See "Configuring Oracle SQL Connector for HDFS" on page 2-21 for descriptions of the properties used for this data source.

Optional Properties

This property is optional:

- `oracle.hadoop.exctab.logDirectory`

Defining Properties in XML Files for Data Pump Format Files

[Example 2-2](#) is an XML template containing the properties that describe a Data Pump file. To use the template, cut and paste it into a text file, enter the appropriate values to describe your Data Pump file, and delete any optional properties that you do not need. For more information about using XML templates, see "Creating a Configuration File" on page 2-21.

Example 2-2 XML Template with Properties for a Data Pump Format File

```
<?xml version="1.0"?>

<!-- Required Properties -->

<configuration>
  <property>
    <name>oracle.hadoop.exctab.tableName</name>
    <value>value</value>
  </property>
  <property>
    <name>oracle.hadoop.exctab.defaultDirectory</name>
    <value>value</value>
  </property>
  <property>
    <name>oracle.hadoop.exctab.dataPaths</name>
    <value>value</value>
  </property>
  <property>
    <name>oracle.hadoop.exctab.sourceType</name>
    <value>datapump</value>
  </property>
  <property>
    <name>oracle.hadoop.connection.url</name>
    <value>value</value>
  </property>
  <property>
    <name>oracle.hadoop.connection.user</name>
    <value>value</value>
  </property>

<!-- Optional Properties -->
```

```

<property>
  <name>oracle.hadoop.exctab.logDirectory</name>
  <value>value</value>
</property>
</configuration>

```

Example

[Example 2–3](#) creates an external table named SALES_DP_XTAB to read Data Pump files.

Example 2–3 Defining an External Table for Data Pump Format Files

Log in as the operating system user that Oracle Database runs under (typically the oracle user), and create a file-system directory. For Oracle RAC, you must create a cluster-wide directory on a distributed file system.

```
$ mkdir /data/sales_dp_dir
```

Create a database directory and grant read and write access to it:

```

$ sqlplus / as sysdba
SQL> CREATE OR REPLACE DIRECTORY sales_dp_dir AS '/data/sales_dp_dir'
SQL> GRANT READ, WRITE ON DIRECTORY sales_dp_dir TO scott;

```

Create the external table:

```

hadoop jar OSCH_HOME/jlib/orahdfs.jar \
oracle.hadoop.exctab.ExternalTable \
-D oracle.hadoop.exctab.tableName=SALES_DP_XTAB \
-D oracle.hadoop.exctab.sourceType=datapump \
-D oracle.hadoop.exctab.dataPaths=hdfs:///user/scott/olh_sales_dpoutput/ \
-D oracle.hadoop.exctab.defaultDirectory=SALES_DP_DIR \
-D oracle.hadoop.connection.url=jdbc:oracle:thin:@//myhost:1521/myservicename \
-D oracle.hadoop.connection.user=SCOTT \
-createTable

```

Creating External Tables from Hive Tables

Oracle SQL Connector for HDFS creates the external table definition from a Hive table by contacting the Hive metastore client to retrieve information about the table columns and the location of the table data. In addition, the Hive table data paths are published to the location files of the Oracle external table.

To read Hive table metadata, Oracle SQL Connector for HDFS requires that the Hive JAR files are included in the HADOOP_CLASSPATH variable. Oracle SQL Connector for HDFS must be installed and running on a computer with a working Hive client.

Ensure that you add the Hive configuration directory to the HADOOP_CLASSPATH environment variable. You must have a correctly functioning Hive client.

For Hive managed tables, the data paths come from the warehouse directory.

For Hive external tables, the data paths from an external location in HDFS are published to the location files of the Oracle external table. Hive external tables can have no data, because Hive does not check if the external location is defined when the table is created. If the Hive table is empty, then one location file is published with just a header and no data URIs.

The Oracle external table is not a "live" Hive table. When changes are made to a Hive table, you must use the ExternalTable tool to either republish the data or create a new external table.

Hive Table Requirements

Oracle SQL Connector for HDFS supports non-partitioned Hive tables that are defined using `ROW FORMAT DELIMITED` and `FILE FORMAT TEXTFILE` clauses. Both Hive-managed tables and Hive external tables are supported.

Hive tables can be either bucketed or not bucketed. All primitive types from Hive 0.10.0 are supported.

Data Type Mappings

Table 2–1 shows the default data-type mappings between Hive and Oracle. To change the data type of the target columns created in the Oracle external table, set the `oracle.hadoop.exttab.hive.columnType.*` properties listed under "Optional Properties" on page 2-11..

Table 2–1 *Hive Data Type Mappings*

Data Type of Source Hive Column	Default Data Type of Target Oracle Column
INT, BIGINT, SMALLINT, TINYINT	INTEGER
DECIMAL, DOUBLE, FLOAT	NUMBER
BOOLEAN	VARCHAR2 (5)
STRING	VARCHAR2 (4000)
TIMESTAMP	TIMESTAMP with format mask YYYY-MM-DD HH24:MI:SS.FF

Required Properties

These properties are required for Hive table sources:

- `oracle.hadoop.exttab.tableName`
- `oracle.hadoop.exttab.defaultDirectory`
- `oracle.hadoop.exttab.sourceType=hive`
- `oracle.hadoop.exttab.hive.tableName`
- `oracle.hadoop.exttab.hive.databaseName`
- `oracle.hadoop.connection.url`
- `oracle.hadoop.connection.user`

See "Configuring Oracle SQL Connector for HDFS" on page 2-21 for descriptions of the properties used for this data source.

Optional Properties

These properties are optional for Hive table sources:

- `oracle.hadoop.exttab.hive.columnType.*`
- `oracle.hadoop.exttab.locationFileCount`
- `oracle.hadoop.exttab.colMap.columnLength`
- `oracle.hadoop.exttab.colMap.column_name.columnLength`
- `oracle.hadoop.exttab.colMap.columnType`
- `oracle.hadoop.exttab.colMap.column_name.columnType`
- `oracle.hadoop.exttab.colMap.dateMask`

- `oracle.hadoop.exctab.colMap.column_name.dateMask`
- `oracle.hadoop.exctab.colMap.fieldLength`
- `oracle.hadoop.exctab.colMap.column_name.fieldLength`
- `oracle.hadoop.exctab.colMap.timestampMask`
- `oracle.hadoop.exctab.colMap.column_name.timestampMask`
- `oracle.hadoop.exctab.colMap.timestampTZMask`
- `oracle.hadoop.exctab.colMap.column_name.timestampTZMask`

Defining Properties in XML Files for Hive Tables

[Example 2-4](#) is an XML template containing the properties that describe a Hive table. To use the template, cut and paste it into a text file, enter the appropriate values to describe your Hive table, and delete any optional properties that you do not need. For more information about using XML templates, see "[Creating a Configuration File](#)" on page 2-21.

Example 2-4 XML Template with Properties for a Hive Table

```
<?xml version="1.0"?>

<!-- Required Properties -->

<configuration>
  <property>
    <name>oracle.hadoop.exctab.tableName</name>
    <value>value</value>
  </property>
  <property>
    <name>oracle.hadoop.exctab.defaultDirectory</name>
    <value>value</value>
  </property>
  <property>
    <name>oracle.hadoop.exctab.sourceType</name>
    <value>hive</value>
  </property>
  <property>
    <name>oracle.hadoop.exctab.hive.tableName</name>
    <value>value</value>
  </property>
  <property>
    <name>oracle.hadoop.exctab.hive.databaseName</name>
    <value>value</value>
  </property>
  <property>
    <name>oracle.hadoop.connection.url</name>
    <value>value</value>
  </property>
  <property>
    <name>oracle.hadoop.connection.user</name>
    <value>value</value>
  </property>

<!-- Optional Properties -->

  <property>
    <name>oracle.hadoop.exctab.locationFileCount</name>
```

```

    <value>value</value>
  </property>
</property>
  <name>oracle.hadoop.exctab.hive.columnType.TYPE</name>
  <value>value</value>
</property>
</configuration>

```

Example

[Example 2-5](#) creates an external table named SALES_HIVE_XTAB to read data from a Hive table. The example defines all the properties on the command line instead of in an XML file.

Example 2-5 Defining an External Table for a Hive Table

Log in as the operating system user that Oracle Database runs under (typically the oracle user), and create a file-system directory:

```
$ mkdir /data/sales_hive_dir
```

Create a database directory and grant read and write access to it:

```
$ sqlplus / as sysdba
SQL> CREATE OR REPLACE DIRECTORY sales_hive_dir AS '/data/sales_hive_dir'
SQL> GRANT READ, WRITE ON DIRECTORY sales_hive_dir TO scott;
```

Create the external table:

```
hadoop jar $OSCH_HOME/jlib/orahdfs.jar \
oracle.hadoop.exctab.ExternalTable \
-D oracle.hadoop.exctab.tableName=SALES_HIVE_XTAB \
-D oracle.hadoop.exctab.sourceType=hive \
-D oracle.hadoop.exctab.locationFileCount=2 \
-D oracle.hadoop.exctab.hive.tableName=sales_country_us \
-D oracle.hadoop.exctab.hive.databaseName=salesdb \
-D oracle.hadoop.exctab.defaultDirectory=SALES_HIVE_DIR \
-D oracle.hadoop.connection.url=jdbc:oracle:thin:@//myhost:1521/myservername \
-D oracle.hadoop.connection.user=SCOTT \
-createTable
```

Creating External Tables from Delimited Text Files

Oracle SQL Connector for HDFS creates the external table definition for delimited text files using configuration properties that specify the number of columns, the text delimiter, and optionally, the external table column names. By default, all text columns in the external table are VARCHAR2. If column names are not provided, they default to C1 to Cn, where n is the number of columns specified by the [oracle.hadoop.exctab.columnCount](#) property.

Data Type Mappings

All text data sources are automatically mapped to VARCHAR2(4000). To change the data type of the target columns created in the Oracle external table, set the `oracle.hadoop.exctab.colMap.*` properties listed under "[Optional Properties](#)" on page 2-14.

Required Properties

These properties are required for delimited text sources:

- `oracle.hadoop.exctab.tableName`
- `oracle.hadoop.exctab.defaultDirectory`
- `oracle.hadoop.exctab.dataPaths`
- `oracle.hadoop.exctab.columnCount` or `oracle.hadoop.exctab.columnNames`
- `oracle.hadoop.connection.url`
- `oracle.hadoop.connection.user`

See "Configuring Oracle SQL Connector for HDFS" on page 2-21 for descriptions of the properties used for this data source.

Optional Properties

These properties are optional for delimited text sources:

- `oracle.hadoop.exctab.recordDelimiter`
- `oracle.hadoop.exctab.fieldTerminator`
- `oracle.hadoop.exctab.initialFieldEncloser`
- `oracle.hadoop.exctab.trailingFieldEncloser`
- `oracle.hadoop.exctab.locationFileCount`
- `oracle.hadoop.exctab.colMap.columnLength`
- `oracle.hadoop.exctab.colMap.column_name.columnLength`
- `oracle.hadoop.exctab.colMap.columnType`
- `oracle.hadoop.exctab.colMap.column_name.columnType`
- `oracle.hadoop.exctab.colMap.dateMask`
- `oracle.hadoop.exctab.colMap.column_name.dateMask`
- `oracle.hadoop.exctab.colMap.fieldLength`
- `oracle.hadoop.exctab.colMap.column_name.fieldLength`
- `oracle.hadoop.exctab.colMap.timestampMask`
- `oracle.hadoop.exctab.colMap.column_name.timestampMask`
- `oracle.hadoop.exctab.colMap.timestampTZMask`
- `oracle.hadoop.exctab.colMap.column_name.timestampTZMask`

Defining Properties in XML Files for Delimited Text Files

[Example 2-6](#) is an XML template containing all the properties that describe a delimited text file. To use the template, cut and paste it into a text file, enter the appropriate values to describe your data files, and delete any optional properties that you do not need. For more information about using XML templates, see "[Creating a Configuration File](#)" on page 2-21.

Example 2-6 XML Template with Properties for a Delimited Text File

```
<?xml version="1.0"?>

<!-- Required Properties -->

<configuration>
  <property>
```

```

        <name>oracle.hadoop.exctab.tableName</name>
        <value>value</value>
    </property>
<property>
    <name>oracle.hadoop.exctab.defaultDirectory</name>
    <value>value</value>
</property>
<property>
    <name>oracle.hadoop.exctab.dataPaths</name>
    <value>value</value>
</property>

<!-- Use either columnCount or columnNames -->

<property>
    <name>oracle.hadoop.exctab.columnCount</name>
    <value>value</value>
</property>
<property>
    <name>oracle.hadoop.exctab.columnNames</name>
    <value>value</value>
</property>

<property>
    <name>oracle.hadoop.connection.url</name>
    <value>value</value>
</property>
<property>
    <name>oracle.hadoop.connection.user</name>
    <value>value</value>
</property>

<!-- Optional Properties -->

<property>
    <name>oracle.hadoop.exctab.colMap.TYPE</name>
    <value>value</value>
</property>
<property>
    <name>oracle.hadoop.exctab.recordDelimiter</name>
    <value>value</value>
</property>
<property>
    <name>oracle.hadoop.exctab.fieldTerminator</name>
    <value>value</value>
</property>
<property>
    <name>oracle.hadoop.exctab.initialFieldEncloser</name>
    <value>value</value>
</property>
<property>
    <name>oracle.hadoop.exctab.trailingFieldEncloser</name>
    <value>value</value>
</property>
<property>
    <name>oracle.hadoop.exctab.locationFileCount</name>
    <value>value</value>
</property>
</configuration>

```

Example

[Example 2-7](#) creates an external table named SALES_DT_XTAB from delimited text files.

Example 2-7 Defining an External Table for Delimited Text Files

Log in as the operating system user that Oracle Database runs under (typically the oracle user), and create a file-system directory:

```
$ mkdir /data/sales_dt_dir
```

Create a database directory and grant read and write access to it:

```
$ sqlplus / as sysdba
SQL> CREATE OR REPLACE DIRECTORY sales_dt_dir AS '/data/sales_dt_dir'
SQL> GRANT READ, WRITE ON DIRECTORY sales_dt_dir TO scott;
```

Create the external table:

```
hadoop jar $OSCH_HOME/jlib/orahdfs.jar \
oracle.hadoop.exttab.ExternalTable \
-D oracle.hadoop.exttab.tableName=SALES_DT_XTAB \
-D oracle.hadoop.exttab.locationFileCount=2 \
-D oracle.hadoop.exttab.dataPaths="hdfs:///user/scott/olh_sales/*.dat" \
-D oracle.hadoop.exttab.columnCount=10 \
-D oracle.hadoop.exttab.defaultDirectory=SALES_DT_DIR \
-D oracle.hadoop.connection.url=jdbc:oracle:thin:@//myhost:1521/myservername \
-D oracle.hadoop.connection.user=SCOTT \
-createTable
```

Creating External Tables in SQL

You can create an external table manually for Oracle SQL Connector for HDFS. For example, the following procedure enables you to use external table syntax that is not exposed by the ExternalTable -createTable command.

Additional syntax might not be supported for Data Pump format files.

To create an external table manually:

1. Use the -createTable --noexecute command to generate the external table DDL.
2. Make whatever changes are needed to the DDL.
3. Run the DDL from Step 2 to create the table definition in the Oracle database.
4. Use the ExternalTable -publish command to publish the data URIs to the location files of the external table.

Publishing the HDFS Data Paths

The -createTable command creates the metadata in Oracle Database and populates the location files with the Universal Resource Identifiers (URIs) of the data files in HDFS. However, you might publish the URIs as a separate step from creating the external table in cases like these:

- You want to publish new data into an already existing external table.
- You created the external table manually instead of using the ExternalTable tool.

In both cases, you can use ExternalTable with the -publish command to populate the external table location files with the URIs of the data files in HDFS. See "[Location File Management](#)" on page 2-20.

ExternalTable Syntax for Publish

```
hadoop jar OSCH_HOME/jlib/orahdfs.jar \
oracle.hadoop.exttab.ExternalTable \
[-conf config_file]... \
[-D property=value]... \
-publish [--noexecute]
```

See Also: ["ExternalTable Command-Line Tool Syntax"](#) on page 2-6

ExternalTable Command-Line Tool Example

[Example 2-8](#) sets HADOOP_CLASSPATH and publishes the HDFS data paths to the external table created in [Example 2-3](#). See ["Configuring Your System for Oracle SQL Connector for HDFS"](#) on page 2-5 for more information about setting this environment variable.

Example 2-8 Publishing HDFS Data Paths to an External Table for Data Pump Format Files

This example uses the Bash shell.

```
$ export HADOOP_CLASSPATH="OSCH_HOME/jlib/*"
$ hadoop jar OSCH_HOME/jlib/orahdfs.jar oracle.hadoop.exttab.ExternalTable \
-D oracle.hadoop.exttab.tableName=SALES_DP_XTAB \
-D oracle.hadoop.exttab.sourceType=datapump \
-D oracle.hadoop.exttab.dataPaths=hdfs:/user/scott/data/ \
-D oracle.hadoop.connection.url=jdbc:oracle:thin:@//myhost:1521/myservername \
-D oracle.hadoop.exttab.connection.user=scott -publish
```

In this example:

- OSCH_HOME is the full path to the Oracle SQL Connector for HDFS installation directory.
- SALES_DP_XTAB is the external table created in [Example 2-3](#).
- hdfs:/user/scott/data/ is the location of the HDFS data.
- @myhost:1521 is the database connection string.

Listing Location File Metadata and Contents

The `-listLocations` command is a debugging and diagnostic utility that enables you to see the location file metadata and contents. You can use this command to verify the integrity of the location files of an Oracle external table.

These properties are required to use this command:

- `oracle.hadoop.exttab.tableName`
- The JDBC connection properties; see ["Connection Properties"](#) on page 2-30.

ExternalTable Syntax for -listLocations

```
hadoop jar OSCH_HOME/jlib/orahdfs.jar \
oracle.hadoop.exttab.ExternalTable \
[-conf config_file]... \
[-D property=value]... \
-listLocations [--details]
```

See Also: ["ExternalTable Command-Line Tool Syntax"](#) on page 2-6

Describing External Tables

The `-getDDL` command is a debugging and diagnostic utility that prints the definition of an existing external table. This command follows the security model of the PL/SQL `DBMS_METADATA` package, which enables non-privileged users to see the metadata for their own objects.

These properties are required to use this command:

- `oracle.hadoop.exctab.tableName`
- The JDBC connection properties; see ["Connection Properties"](#) on page 2-30.

ExternalTable Syntax for -getDDL

```
hadoop jar OSCH_HOME/jlib/orahdfs.jar \
oracle.hadoop.exctab.ExternalTable \
[-conf config_file]... \
[-D property=value]... \
-getDDL
```

See Also: ["ExternalTable Command-Line Tool Syntax"](#) on page 2-6

More About External Tables Generated by the ExternalTable Tool

Because external tables are used to access data, all of the features and limitations of external tables apply. Queries are executed in parallel with automatic load balancing. However, update, insert, and delete operations are not allowed and indexes cannot be created on external tables. When an external table is accessed, a full table scan is always performed.

Oracle SQL Connector for HDFS uses the `ORACLE_LOADER` access driver. The `hdfs_stream` preprocessor script (provided with Oracle SQL Connector for HDFS) modifies the input data to a format that `ORACLE_LOADER` can process.

See Also:

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

About Configurable Column Mappings

Oracle SQL Connector for HDFS uses default data type mappings to create columns in an Oracle external table with the appropriate data types for the Hive and text sources. You can override these defaults by setting various configuration properties, for either all columns or a specific column.

For example, a field in a text file might contain a timestamp. By default, the field is mapped to a `VARCHAR2` column. However, you can specify a `TIMESTAMP` column and provide a datetime mask to cast the values correctly into the `TIMESTAMP` data type. The `TIMESTAMP` data type supports time-based queries and analysis that are unavailable when the data is presented as text.

Default Column Mappings

Text sources are mapped to VARCHAR2 columns, and Hive columns are mapped to columns with the closest equivalent Oracle data type. Table 2–1 shows the default mappings.

All Column Overrides

The following properties apply to all columns in the external table. For Hive sources, these property settings override the `oracle.hadoop.exttab.hive.*` property settings.

- `oracle.hadoop.exttab.colMap.columnLength`
- `oracle.hadoop.exttab.colMap.columnType`
- `oracle.hadoop.exttab.colMap.dateMask`
- `oracle.hadoop.exttab.colMap.fieldLength`
- `oracle.hadoop.exttab.colMap.timestampMask`
- `oracle.hadoop.exttab.colMap.timestampTZMask`

One Column Overrides

The following properties apply to only one column, whose name is the `column_name` part of the property name. These property settings override all other settings.

- `oracle.hadoop.exttab.colMap.column_name.columnLength`
- `oracle.hadoop.exttab.colMap.column_name.columnType`
- `oracle.hadoop.exttab.colMap.column_name.dateMask`
- `oracle.hadoop.exttab.colMap.column_name.fieldLength`
- `oracle.hadoop.exttab.colMap.column_name.timestampMask`
- `oracle.hadoop.exttab.colMap.column_name.timestampTZMask`

Mapping Override Examples

The following properties create an external table in which all columns are the default VARCHAR2 data type:

```
oracle.hadoop.exttab.tableName=MOVIE_FACT_EXT_TAB_TXT
oracle.hadoop.exttab.columnNames=CUST_ID,MOVIE_ID,GENRE_ID,TIME_ID,RECOMMENDED,ACTIVITY_ID,RATING,SALES
```

In this example, the following properties are set to override the data type of several columns:

```
oracle.hadoop.exttab.colMap.TIME_ID.columnType=TIMESTAMP
oracle.hadoop.exttab.colMap.RECOMMENDED.columnType=NUMBER
oracle.hadoop.exttab.colMap.ACTIVITY_ID.columnType=NUMBER
oracle.hadoop.exttab.colMap.RATING.columnType=NUMBER
oracle.hadoop.exttab.colMap.SALES.columnType=NUMBER
```

Oracle SQL Connector for HDFS creates an external table with the specified data types:

```
SQL> DESCRIBE movie_facts_ext
Name                               Null?      Type
-----
CUST_ID                             VARCHA2 (4000)
MOVIE_ID                             VARCHA2 (4000)
GENRE_ID                             VARCHA2 (4000)
```

```

TIME_ID                TIMESTAMP (9)
RECOMMENDED           NUMBER
ACTIVITY_ID           NUMBER
RATINGS               NUMBER
SALES                 NUMBER
    
```

The next example adds the following property settings to change the length of the VARCHAR2 columns:

```

oracle.hadoop.exctab.colMap.CUST_ID.columnLength=12
oracle.hadoop.exctab.colMap.MOVIE_ID.columnLength=12
oracle.hadoop.exctab.colMap.GENRE_ID.columnLength=12
    
```

All columns now have custom data types:

```

SQL> DESCRIBE movie_facts_ext
Name                               Null?    Type
-----
CUST_ID                            VARCHA2 (12)
MOVIE_ID                            VARCHA2 (12)
GENRE_ID                            VARCHA2 (12)
TIME_ID                             TIMESTAM (9)
RECOMMENDED                         NUMBER
ACTIVITY_ID                         NUMBER
RATINGS                             NUMBER
SALES                               NUMBER
    
```

What Are Location Files?

A **location file** is a file specified in the location clause of the external table. Oracle SQL Connector for HDFS creates location files that contain only the Universal Resource Identifiers (URIs) of the data files. A **data file** contains the data stored in HDFS.

Enabling Parallel Processing

To enable parallel processing with external tables, you must specify multiple files in the location clause of the external table. The number of files, also known as the **degree of parallelism**, determines the number of child processes started by the external table during a table read. Ideally, the degree of parallelism is no larger than the number of data files, to avoid idle child processes.

Location File Management

The Oracle SQL Connector for HDFS command-line tool, `ExternalTable`, creates an external table and publishes the HDFS URI information to location files. The external table location files are stored in the directory specified by the `oracle.hadoop.exctab.defaultDirectory` property. For an Oracle RAC database, this directory must reside on a distributed file system that is accessible to each database server.

`ExternalTable` manages the location files of the external table, which involves the following operations:

- Generating new location files in the database directory after checking for name conflicts
- Deleting existing location files in the database directory as necessary
- Publishing data URIs to new location files
- Altering the `LOCATION` clause of the external table to match the new location files

Location file management for the supported data sources is described in the following topics.

Data Pump File Format

The `ORACLE_LOADER` access driver is required to access Data Pump files. The driver requires that each location file corresponds to a single Data Pump file in HDFS. Empty location files are not allowed, and so the number of location files in the external table must exactly match the number of data files in HDFS.

Oracle SQL Connector for HDFS automatically takes over location file management and ensures that the number of location files in the external table equals the number of Data Pump files in HDFS.

Delimited Files in HDFS and Hive Tables

The `ORACLE_LOADER` access driver has no limitation on the number of location files. Each location file can correspond to one or more data files in HDFS. The number of location files for the external table is suggested by the `oracle.hadoop.exttab.locationFileCount` configuration property.

See "[Connection Properties](#)" on page 2-30.

Location File Names

This is the format of a location file name:

`osch-timestamp-number-n`

In this syntax:

- `timestamp` has the format `yyyyMMddhhmmss`, for example, 20121017103941 for October 17, 2012, at 10:39:41.
- `number` is a random number used to prevent location file name conflicts among different tables.
- `n` is an index used to prevent name conflicts between location files for the same table.

For example, `osch-20121017103941-6807-1`.

Configuring Oracle SQL Connector for HDFS

You can pass configuration properties to the `ExternalTable` tool on the command line with the `-D` option, or you can create a configuration file and pass it on the command line with the `-conf` option. These options must precede the command to be executed (`-createTable`, `-publish`, `-listLocations`, or `-getDDL`).

For example, this command uses a configuration file named `example.xml`:

```
hadoop jar $OSCH_HOME/jlib/orahdfs.jar \
  oracle.hadoop.exttab.ExternalTable \
  -conf /home/oracle/example.xml \
  -createTable
```

See "[ExternalTable Command-Line Tool Syntax](#)" on page 2-6.

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>

```

Example 2–9 shows a configuration file. See ["Oracle SQL Connector for HDFS Configuration Property Reference"](#) on page 2-22 for descriptions of these properties.

Example 2–9 Configuration File for Oracle SQL Connector for HDFS

```

<?xml version="1.0"?>
<configuration>
  <property>
    <name>oracle.hadoop.exctab.tableName</name>
    <value>SH.SALES_EXT_DIR</value>
  </property>
  <property>
    <name>oracle.hadoop.exctab.dataPaths</name>
    <value>/data/s1/*.csv,/data/s2/*.csv</value>
  </property>
  <property>
    <name>oracle.hadoop.exctab.dataCompressionCodec</name>
    <value>org.apache.hadoop.io.compress.DefaultCodec</value>
  </property>
  <property>
    <name>oracle.hadoop.connection.url</name>
    <value>jdbc:oracle:thin:@//myhost:1521/myservername</value>
  </property>
  <property>
    <name>oracle.hadoop.connection.user</name>
    <value>SH</value>
  </property>
</configuration>

```

Oracle SQL Connector for HDFS Configuration Property Reference

The following is a complete list of the configuration properties used by the ExternalTable command-line tool. The properties are organized into these categories:

- [General Properties](#)
- [Connection Properties](#)

General Properties

oracle.hadoop.exctab.colMap.columnLength

Specifies the length of all external table columns of type CHAR, VARCHAR2, NCHAR, NVARCHAR2, and RAW. Optional.

Default Value: The maximum length allowed by the column type

For Oracle Database 12c, Oracle SQL Connector for HDFS sets the length of VARCHAR2, NVARCHAR2, and RAW columns depending on whether the database MAX_STRING_SIZE option is set to STANDARD or EXTENDED.

Valid values: Integer

oracle.hadoop.exctab.colMap.columnType

Specifies the data type mapping of all columns for Hive and text sources. Optional.

You can override this setting for specific columns by setting `oracle.hadoop.exctab.colMap.column_name.columnType`.

Default value: VARCHAR2 for text; see [Table 2–1](#) for Hive

Valid values: The following Oracle data types are supported:

VARCHAR2
 NVARCHAR2
 CHAR
 NCHAR
 CLOB
 NCLOB
 NUMBER
 INTEGER
 FLOAT
 BINARY_DOUBLE
 BINARY_FLOAT
 RAW*
 DATE
 TIMESTAMP
 TIMESTAMP WITH TIME ZONE
 TIMESTAMP WITH LOCAL TIME ZONE
 INTERVAL DAY TO SECOND
 INTERVAL YEAR TO MONTH

* RAW binary data in delimited text files must be encoded in hexadecimal.

oracle.hadoop.exctab.colMap.dateMask

Specifies the format mask used in the *date_format_spec* clause of the external table for all DATE columns. This clause indicates that a character data field contains a date in the specified format.

Default value: The default globalization format mask, which is set by the NLS_DATE_FORMAT database parameter

Valid values: A datetime format model as described in the *Oracle Database SQL Language Reference*. However, it cannot contain quotation marks.

oracle.hadoop.exctab.colMap.fieldLength

Sets the character buffer length used by the ORACLE_LOADER access driver for all CLOB columns. The value is used in the *field_list* clause of the external table definition, which identifies the fields in the data file and their data types.

Default value: 4000 bytes

Valid values: Integer

oracle.hadoop.exctab.colMap.timestampMask

Specifies the format mask used in the *date_format_spec* clause of the external table for all TIMESTAMP and TIMESTAMP WITH LOCAL TIME ZONE columns. This clause indicates that a character data field contains a timestamp in the specified format.

Default value: The default globalization format mask, which is set by the NLS_TIMESTAMP_FORMAT database parameter

Valid values: A datetime format model as described in the *Oracle Database SQL Language Reference*. However, it cannot contain quotation marks.

oracle.hadoop.exttab.colMap.timestampTZMask

Specifies the format mask used in the *date_format_spec* clause of the external table for all `TIMESTAMP WITH TIME ZONE` columns. This clause indicates that a character data field contains a timestamp in the specified format.

Default value: The default globalization format mask, which is set by the `NLS_TIMESTAMP_TZ_FORMAT` database parameter

Valid values: A datetime format model as described in the *Oracle Database SQL Language Reference*. However, it cannot contain quotation marks.

oracle.hadoop.exttab.colMap.column_name.columnLength

Specifies the length of all external table columns of type `CHAR`, `VARCHAR2`, `NCHAR`, `NVARCHAR2`, and `RAW`. Optional.

Default Value: The value of `oracle.hadoop.exttab.colMap.columnLength`; if that property is not set, then the maximum length allowed by the data type

Valid values: Integer

oracle.hadoop.exttab.colMap.column_name.columnType

Overrides the data type mapping for *column_name*. Optional.

The *column_name* is case-sensitive. It must exactly match the name of a column in a Hive table or a column listed in `oracle.hadoop.exttab.columnNames`.

Default value: The value of `oracle.hadoop.exttab.colMap.columnType`; if that property is not set, then the default data type identified in [Table 2-1](#)

Valid values: See `oracle.hadoop.exttab.colMap.columnType`

oracle.hadoop.exttab.colMap.column_name.dateMask

Overrides the format mask for *column_name*. Optional.

The *column_name* is case-sensitive. It must exactly match the name of a column in a Hive table or a column listed in `oracle.hadoop.exttab.columnNames`.

Default value: The value of `oracle.hadoop.exttab.colMap.dateMask`.

Valid values: A datetime format model as described in the *Oracle Database SQL Language Reference*. However, it cannot contain quotation marks.

oracle.hadoop.exttab.colMap.column_name.fieldLength

Overrides the character buffer length used by the `ORACLE_LOADER` access driver for *column_name*. This property is especially useful for `CLOB` and extended data type columns. Optional.

The *column_name* is case-sensitive. It must exactly match the name of a column in a Hive table or a column listed in `oracle.hadoop.exttab.columnNames`.

Default value: Oracle SQL Connector for HDFS sets the default field lengths as shown in [Table 2-2](#).

Table 2-2 Field Length Calculations

Data Type of Target Column	Field Length
<code>VARCHAR2</code> , <code>NVARCHAR2</code> , <code>CHAR</code> , <code>NCHAR</code>	Value of <code>oracle.hadoop.exttab.colMap.column_name.columnLength</code>
<code>RAW</code>	2 * <code>columnLength</code> property
<code>CLOB</code> , <code>NCLOB</code>	Value of <code>oracle.hadoop.exttab.colMap.fieldLength</code>
All other types	255 (default size for external tables)

Valid values: Integer

oracle.hadoop.exctab.colMap.column_name.timestampMask

Overrides the format mask for *column_name*. Optional.

The *column_name* is case-sensitive. It must exactly match the name of a column in a Hive table or a column listed in [oracle.hadoop.exctab.columnNames](#).

Default value: The value of [oracle.hadoop.exctab.colMap.timestampMask](#).

Valid values: A datetime format model as described in the *Oracle Database SQL Language Reference*. However, it cannot contain quotation marks.

oracle.hadoop.exctab.colMap.column_name.timestampTZMask

Overrides the format mask for *column_name*. Optional.

The *column_name* is case-sensitive. It must exactly match the name of a column in a Hive table or a column listed in [oracle.hadoop.exctab.columnNames](#).

Default value: The value of [oracle.hadoop.exctab.colMap.timestampTZMask](#).

Valid values: A datetime format model as described in the *Oracle Database SQL Language Reference*. However, it cannot contain quotation marks.

oracle.hadoop.exctab.columnCount

The number of columns for the external table created from delimited text files. The column names are set to C1, C2,... Cn, where *n* is value of this property.

This property is ignored if [oracle.hadoop.exctab.columnNames](#) is set.

The `-createTable` command uses this property when [oracle.hadoop.exctab.sourceType=text](#).

You must set either this property or [oracle.hadoop.exctab.columnNames](#) when creating an external table from delimited text files.

oracle.hadoop.exctab.columnNames

A comma-separated list of column names for an external table created from delimited text files. If this property is not set, then the column names are set to C1, C2,... Cn, where *n* is the value of the [oracle.hadoop.exctab.columnCount](#) property.

The column names are read as SQL identifiers: unquoted values are capitalized, and double-quoted values stay exactly as entered.

The `-createTable` command uses this property when [oracle.hadoop.exctab.sourceType=text](#).

You must set either this property or [oracle.hadoop.exctab.columnCount](#) when creating an external table from delimited text files.

oracle.hadoop.exctab.dataCompressionCodec

The name of the compression codec class used to decompress the data files. Specify this property when the data files are compressed. Optional.

This property specifies the class name of the compression codec that implements the `org.apache.hadoop.io.compress.CompressionCodec` interface. This codec applies to all data files.

Several standard codecs are available in Hadoop, including the following:

- **bzip2:** `org.apache.hadoop.io.compress.BZip2Codec`
- **gzip:** `org.apache.hadoop.io.compress.GzipCodec`

Default value: None

oracle.hadoop.exctab.dataPaths

A comma-separated list of fully qualified HDFS paths. This property enables you to restrict the input by using special pattern-matching characters in the path specification. See Table 2-3. This property is required for the `-createTable` and `-publish` commands using Data Pump or delimited text files. The property is ignored for Hive data sources.

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 2-3 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.exctab.dataPathFilter

The path filter class. This property is ignored for Hive data sources.

Oracle SQL Connector for HDFS uses a default filter to exclude hidden files, which begin with a dot or an underscore. If you specify another path filter class using the this property, then your filter acts in addition to the default filter. Thus, only visible files accepted by your filter are considered.

oracle.hadoop.exctab.defaultDirectory

Specifies the default directory for the Oracle external table. This directory is used for all input and output files that do not explicitly name a directory object.

Valid value: The name of an existing database directory

Unquoted names are changed to upper case. Double-quoted names are not changed; use them when case-sensitivity is desired. Single-quoted names are not allowed for default directory names.

The `-createTable` command requires this property.

oracle.hadoop.exctab.fieldTerminator

Specifies the field terminator for an external table when `oracle.hadoop.exctab.sourceType=text`. Optional.

Default value: , (comma)

Valid values: A string in one of the following formats:

- One or more regular printable characters; it cannot start with `\u`. For example, `\t` represents a tab.
- One or more encoded characters in the format `\uHHHH`, where `HHHH` is a big-endian hexadecimal representation of the character in UTF-16. For example, `\u0009` represents a tab. The hexadecimal digits are case insensitive.

Do not mix the two formats.

oracle.hadoop.exctab.hive.columnType.*

Maps a Hive data type to an Oracle data type. The property name identifies the Hive data type, and its value is an Oracle data type. The target columns in the external table are created with the Oracle data type indicated by this property.

When Hive `TIMESTAMP` column is mapped to an Oracle `TIMESTAMP` column, then the format mask is `YYYY-MM-DD H24:MI:SS.FF`. When a Hive `STRING` column is mapped to an Oracle `TIMESTAMP` column, then the NLS parameter settings for the database are used by default. You can override these defaults by using either the `oracle.hadoop.exctab.colMap.timestampMask` or `oracle.hadoop.exctab.colMap.timestampTZMask` properties.

Default values: Table 2–4 lists the Hive column type properties and their default values.

Valid values: See the valid values for `oracle.hadoop.exctab.colMap.columnType`.

Table 2–4 Hive Column Type Mapping Properties

Property	Default Value
<code>oracle.hadoop.exctab.hive.columnType.BIGINT</code>	INTEGER
<code>oracle.hadoop.exctab.hive.columnType.BOOLEAN</code>	VARCHAR2
<code>oracle.hadoop.exctab.hive.columnType.DECIMAL</code>	NUMBER
<code>oracle.hadoop.exctab.hive.columnType.DOUBLE</code>	NUMBER
<code>oracle.hadoop.exctab.hive.columnType.FLOAT</code>	NUMBER
<code>oracle.hadoop.exctab.hive.columnType.INT</code>	INTEGER
<code>oracle.hadoop.exctab.hive.columnType.SMALLINT</code>	INTEGER
<code>oracle.hadoop.exctab.hive.columnType.STRING</code>	VARCHAR2
<code>oracle.hadoop.exctab.hive.columnType.TIMESTAMP</code>	TIMESTAMP
<code>oracle.hadoop.exctab.hive.columnType.TINYINT</code>	INTEGER

oracle.hadoop.exctab.hive.databaseName

The name of a Hive database that contains the input data table.

The `-createTable` command requires this property when `oracle.hadoop.exctab.sourceType=hive`.

oracle.hadoop.exctab.hive.tableName

The name of an existing Hive table.

The `-createTable` command requires this property when `oracle.hadoop.exctab.sourceType=hive`.

oracle.hadoop.exctab.initialFieldEncloser

Specifies the initial field encloser for an external table created from delimited text files. Optional.

Default value: null; no enclosers are specified for the external table definition.

The `-createTable` command uses this property when `oracle.hadoop.exctab.sourceType=text`.

Valid values: A string in one of the following formats:

- One or more regular printable characters; it cannot start with `\u`.
- One or more encoded characters in the format `\uHHHH`, where `HHHH` is a big-endian hexadecimal representation of the character in UTF-16. The hexadecimal digits are case insensitive.

Do not mix the two formats.

oracle.hadoop.exctab.locationFileCount

Specifies the desired number of location files for the external table. Applicable only to non-Data-Pump files.

Default value: 4

This property is ignored if the data files are in Data Pump format. Otherwise, the number of location files is the lesser of:

- The number of data files
- The value of this property

At least one location file is created.

See "[Enabling Parallel Processing](#)" on page 2-20 for more information about the number of location files.

oracle.hadoop.exctab.logDirectory

Specifies a database directory where log files, bad files, and discard files are stored. The file names are the default values used by external tables. For example, the name of a log file is the table name followed by `_%p.log`.

This is an optional property for the `-createTable` command.

These are the default file name extensions:

- Log files: log
- Bad files: bad
- Discard files: dsc

Valid values: An existing Oracle directory object name.

Unquoted names are changed to uppercase. Quoted names are not changed. [Table 2-5](#) provides examples of how values are transformed.

Table 2-5 Examples of Quoted and Unquoted Values

Specified Value	Interpreted Value
<code>my_log_dir:'sales_tab_%p.log '</code>	<code>MY_LOG_DIR/sales_tab_%p.log</code>
<code>'my_log_dir': 'sales_tab_%p.log'</code>	<code>my_log_dir/sales_tab_%p.log</code>

Table 2–5 (Cont.) Examples of Quoted and Unquoted Values

Specified Value	Interpreted Value
"my_log_dir": "sales_tab_%p.log"	my_log_dir/sales_tab_%p.log

oracle.hadoop.exctab.preprocessorDirectory

Specifies the database directory for the preprocessor. The file-system directory must contain the `hdfs_stream` script.

Default value: `OSCH_BIN_PATH`

The preprocessor directory is used in the `PREPROCESSOR` clause of the external table.

oracle.hadoop.exctab.recordDelimiter

Specifies the record delimiter for an external table created from delimited text files. Optional.

Default value: `\n`

The `-createTable` command uses this property when `oracle.hadoop.exctab.sourceType=text`.

Valid values: A string in one of the following formats:

- One or more regular printable characters; it cannot start with `\u`.
- One or more encoded characters in the format `\uHHHH`, where `HHHH` is a big-endian hexadecimal representation of the character in UTF-16. The hexadecimal digits are case insensitive.

Do not mix the two formats.

oracle.hadoop.exctab.sourceType

Specifies the type of source files. The `-createTable` and `-publish` operations require the value of this property.

Default value: `text`

Valid values: `datapump`, `hive`, or `text`

oracle.hadoop.exctab.stringSizes

Indicates whether the lengths specified for character strings are bytes or characters. This value is used in the `STRING SIZES ARE IN` clause of the external table. Use characters when loading multibyte character sets. See *Oracle Database Utilities*.

Default value: `BYTES`

Valid values: `BYTES` or `CHARACTERS`

oracle.hadoop.exctab.tableName

Schema-qualified name of the external table in this format:

schemaName.tableName

If you omit *schemaName*, then the schema name defaults to the connection user name.

Default value: none

Required property for all operations.

oracle.hadoop.exctab.trailingFieldEncloser

Specifies the trailing field encloser for an external table created from delimited text files. Optional.

Default value: null; defaults to the value of `oracle.hadoop.exttab.initialFieldEncloser`

The `-createTable` command uses this property when `oracle.hadoop.exttab.sourceType=text`.

Valid values: A string in one of the following formats:

- One or more regular printable characters; it cannot start with `\u`.
- One or more encoded characters in the format `\uHHHH`, where `HHHH` is a big-endian hexadecimal representation of the character in UTF-16. The hexadecimal digits are case insensitive.

Do not mix the two formats.

Connection Properties

oracle.hadoop.connection.url

Specifies the database connection string in the thin-style service name format:

```
jdbc:oracle:thin:@//host_name:port/service_name
```

If you are unsure of the service name, then enter this SQL command as a privileged user:

```
SQL> show parameter service
```

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

This property takes precedence over all other connection properties.

Default value: Not defined

Valid values: A string

oracle.hadoop.connection.user

An Oracle database log-in name. The `externalTable` tool prompts for a password. This property is required unless you are using Oracle wallet as an external password store.

Default value: Not defined

Valid values: A string

oracle.hadoop.connection.tnsEntryName

Specifies a TNS entry name defined in the `tnsnames.ora` file.

This property is used with the `oracle.hadoop.connection.tns_admin` property.

Default value: Not defined

Valid values: A string

oracle.hadoop.connection.tns_admin

Specifies the directory that contains the `tnsnames.ora` file. Define this property to use transparent network substrate (TNS) entry names in database connection strings. When using TNSNames with the JDBC thin driver, you must set either this property or the Java `oracle.net.tns_admin` property. When both are set, this property takes precedence over `oracle.net.tns_admin`.

This property must be set when using Oracle Wallet as an external password store. See [oracle.hadoop.connection.wallet_location](#).

Default value: The value of the Java `oracle.net.tns_admin` system property

Valid values: A string

oracle.hadoop.connection.wallet_location

A file path to an Oracle wallet directory where the connection credential is stored.

Default value: Not defined

Valid values: A string

When using Oracle Wallet as an external password store, set these properties:

- `oracle.hadoop.connection.wallet_location`
- `oracle.hadoop.connection.url` *or* `oracle.hadoop.connection.tnsEntryName`
- `oracle.hadoop.connection.tns_admin`

Performance Tips for Querying Data in HDFS

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

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

Before loading the data into an Oracle database from the external files created by Oracle SQL Connector for HDFS, 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 you are inserting data.

Oracle Loader for Hadoop

This chapter explains how to use Oracle Loader for Hadoop to copy data from Apache Hadoop into tables in an Oracle database. It contains the following sections:

- [What Is Oracle Loader for Hadoop?](#)
- [About the Modes of Operation](#)
- [Getting Started With Oracle Loader for Hadoop](#)
- [Creating the Target Table](#)
- [Creating a Job Configuration File](#)
- [About the Target Table Metadata](#)
- [About Input Formats](#)
- [Mapping Input Fields to Target Table Columns](#)
- [About Output Formats](#)
- [Running a Loader Job](#)
- [Handling Rejected Records](#)
- [Balancing Loads When Loading Data into Partitioned Tables](#)
- [Optimizing Communications Between Oracle Engineered Systems](#)
- [Oracle Loader for Hadoop Configuration Property Reference](#)
- [Third-Party Licenses for Bundled Software](#)

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. It prepartitions the data if necessary and transforms it into a database-ready format. It can also sort records by primary key or user-specified columns before loading the data or creating output files. Oracle Loader for Hadoop uses the parallel processing framework of Hadoop to perform these preprocessing operations, which other loaders typically perform on the database server as part of the load process. Offloading these operations to Hadoop reduces the CPU requirements on the database server, thereby lessening the performance impact on other database tasks.

Oracle Loader for Hadoop is a Java MapReduce application that balances the data across reducers to help maximize performance. It works with a range of input data formats that present the data as records with fields. It can read from sources that have

the data already in a record format (such as Avro files or Apache Hive tables), or it can split the lines of a text file into fields.

You run Oracle Loader for Hadoop using the `hadoop` command-line utility. In the command line, you provide configuration settings with the details of the job. You typically provide these settings in a job configuration file.

If you have Java programming skills, you can extend the types of data that the loader can handle by defining custom input formats. Then Oracle Loader for Hadoop uses your code to extract the fields and records.

About the Modes of Operation

Oracle Loader for Hadoop operates in two modes:

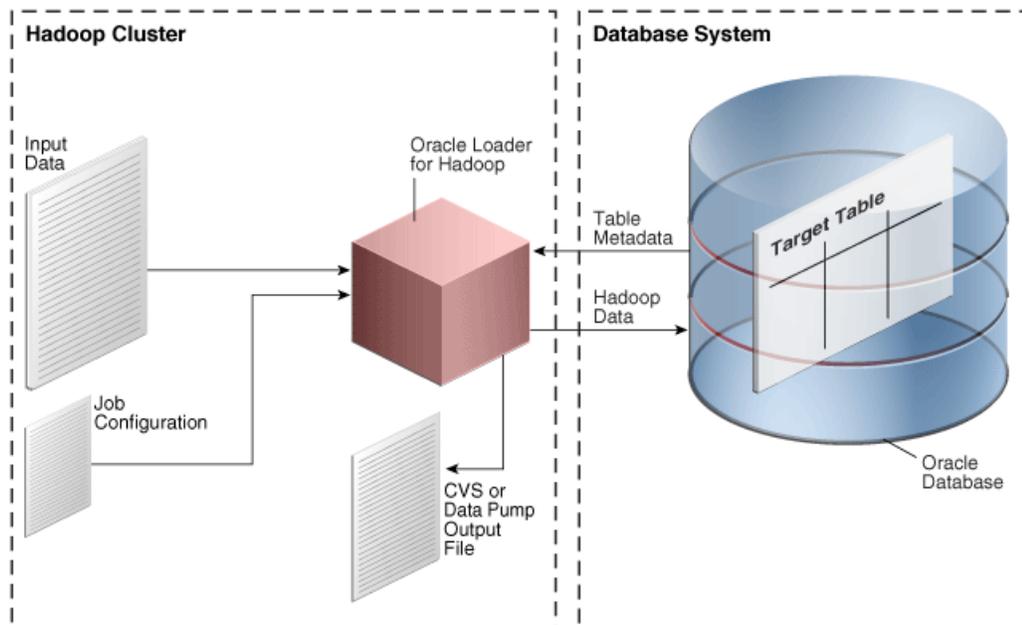
- [Online Database Mode](#)
- [Offline Database Mode](#)

Online Database Mode

In online database mode, Oracle Loader for Hadoop can connect to the target database using the credentials provided in the job configuration file or in an Oracle wallet. The loader obtains the table metadata from the database. It can insert new records directly into the target table or write them to a file in the Hadoop cluster. You can load records from an output file when the data is needed in the database, or when the database system is less busy.

Figure 3–1 shows the relationships among elements in online database mode.

Figure 3–1 Online Database Mode

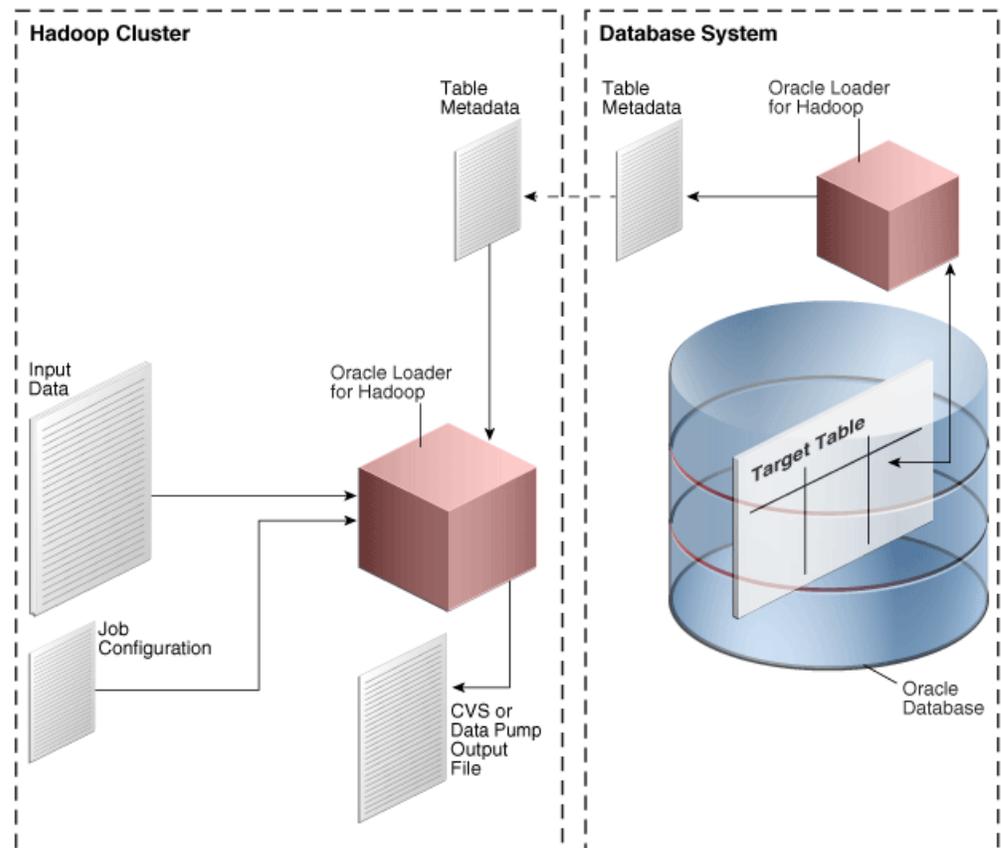


Offline Database Mode

Offline database mode enables you to use Oracle Loader for Hadoop when the Oracle Database system is on a separate network from the Hadoop cluster, or is otherwise inaccessible. In this mode, Oracle Loader for Hadoop uses the information supplied in a table metadata file, which you generate using a separate utility. The loader job stores the output data in binary or text format output files on the Hadoop cluster. Loading the data into Oracle Database is a separate procedure using another utility, such as Oracle SQL Connector for Hadoop Distributed File System (HDFS) or SQL*Loader.

Figure 3–2 shows the relationships among elements in offline database mode. The figure does not show the separate procedure of loading the data into the target table.

Figure 3–2 Offline Database Mode



Getting Started With Oracle Loader for Hadoop

You take the following basic steps when using Oracle Loader for Hadoop:

1. The first time you use Oracle Loader for Hadoop, ensure that the software is installed and configured.
See ["Oracle Loader for Hadoop Setup"](#) on page 1-12.
2. Connect to Oracle Database and create the target table.
See ["Creating the Target Table"](#) on page 3-5.

3. If you are using offline database mode, then generate the table metadata.
See ["Generating the Target Table Metadata for Offline Database Mode"](#) on page 3-8.
4. Log in to either a node in the Hadoop cluster or a system set up as a Hadoop client for the cluster.
5. If you are using offline database mode, then copy the table metadata to the Hadoop system where you are logged in.
6. Create a configuration file. This file is an XML document that describes configuration information, such as access to the target table metadata, the input format of the data, and the output format.
See ["Creating a Job Configuration File"](#) on page 3-6.
7. Create an XML document that maps input fields to columns in the Oracle database table. Optional.
See ["Mapping Input Fields to Target Table Columns"](#) on page 3-15.
8. Create a shell script to run the Oracle Loader for Hadoop job.
See ["Running a Loader Job"](#) on page 3-21.
9. If you are connecting to a secure cluster, then you run `kinit` to authenticate yourself.
10. Run the shell script.
11. If the job fails, then use the diagnostic messages in the output to identify and correct the error.
See ["Job Reporting"](#) on page 3-22.
12. After the job succeeds, check the command output for the number of rejected records. If too many records were rejected, then you may need to modify the input format properties.
13. If you generated text files or Data Pump-format files, then load the data into Oracle Database using one of these methods:
 - Create an external table using Oracle SQL Connector for HDFS (online database mode only).
See [Chapter 2](#).
 - Copy the files to the Oracle Database system and use `SQL*Loader` or external tables to load the data into the target database table. Oracle Loader for Hadoop generates scripts that you can use for these methods.
See ["About DelimitedTextOutputFormat"](#) on page 3-19 or ["About DataPumpOutputFormat"](#) on page 3-20.
14. Connect to Oracle Database as the owner of the target table. Query the table to ensure that the data loaded correctly. If it did not, then modify the input or output format properties as needed to correct the problem.
15. Before running the OraLoader job in a production environment, employ these optimizations:
 - [Balancing Loads When Loading Data into Partitioned Tables](#)
 - [Optimizing Communications Between Oracle Engineered Systems](#)

Creating the Target Table

Oracle Loader for Hadoop loads data into one **target table**, which must exist in the Oracle database. The table can be empty or contain data already. Oracle Loader for Hadoop does not overwrite existing data.

Create the table the same way that you would create one for any other purpose. It must comply with the following restrictions:

- [Supported Data Types for Target Tables](#)
- [Supported Partitioning Strategies for Target Tables](#)

Supported Data Types for Target Tables

You can define the target table using any of these data types:

- BINARY_DOUBLE
- BINARY_FLOAT
- CHAR
- DATE
- FLOAT
- INTERVAL DAY TO SECOND
- INTERVAL YEAR TO MONTH
- NCHAR
- NUMBER
- NVARCHAR2
- RAW
- TIMESTAMP
- TIMESTAMP WITH LOCAL TIME ZONE
- TIMESTAMP WITH TIME ZONE
- VARCHAR2

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 for Target Tables

Partitioning is a database feature for managing and efficiently querying 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.

You can define the target table using any of the following single-level and composite-level partitioning strategies.

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

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

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

See Also: *Oracle Database VLDB and Partitioning Guide*

Creating a Job Configuration File

A configuration file is an XML document that provides Hadoop with all the information it needs to run a MapReduce job. This file can also provide Oracle Loader for Hadoop with all the information it needs. See "[Oracle Loader for Hadoop Configuration Property Reference](#)" on page 3-25.

Configuration properties provide the following information, which is required for all Oracle Loader for Hadoop jobs:

- How to obtain the target table metadata.
See "[About the Target Table Metadata](#)" on page 3-8.
- The format of the input data.
See "[About Input Formats](#)" on page 3-10.
- The format of the output data.
See "[About Output Formats](#)" on page 3-17.

OraLoader implements the `org.apache.hadoop.util.Tool` interface and follows the standard Hadoop methods for building MapReduce applications. Thus, you can supply the configuration properties in a file (as shown here) or on the `hadoop` command line. See "[Running a Loader Job](#)" on page 3-21.

You can use any text or XML editor to create the file. [Example 3-1](#) provides an example of a job configuration file.

Example 3-1 Job Configuration File

```
<?xml version="1.0" encoding="UTF-8" ?>
<configuration>

<!--                               Input settings                               -->

  <property>
    <name>mapreduce.inputformat.class</name>
```

```

    <value>oracle.hadoop.loader.lib.input.DelimitedTextInputFormat</value>
</property>

<property>
  <name>mapred.input.dir</name>
  <value>/user/oracle/moviedemo/session/*0000</value>
</property>

<property>
  <name>oracle.hadoop.loader.input.fieldTerminator</name>
  <value>\u0009</value>
</property>

<!--          Output settings          -->
<property>
  <name>mapreduce.outputformat.class</name>
  <value>oracle.hadoop.loader.lib.output.OCIOutputFormat</value>
</property>

<property>
  <name>mapred.output.dir</name>
  <value>temp_out_session</value>
</property>

<!--          Table information          -->

<property>
  <name>oracle.hadoop.loader.loaderMapFile</name>
  <value>file:///home/oracle/movie/moviedemo/olh/loaderMap_
moviesession.xml</value>
</property>

<!--          Connection information          -->

<property>
  <name>oracle.hadoop.loader.connection.url</name>
  <value>jdbc:oracle:thin:@${HOST}:${TCPPOORT}/${SERVICE_NAME}</value>
</property>

<property>
  <name>TCPPOORT</name>
  <value>1521</value>
</property>

<property>
  <name>HOST</name>
  <value>myoraclehost.example.com</value>
</property>

<property>
  <name>SERVICE_NAME</name>
  <value>orcl</value>
</property>

<property>
  <name>oracle.hadoop.loader.connection.user</name>
  <value>MOVIEDEMO</value>
</property>

```

```
<property>
  <name>oracle.hadoop.loader.connection.password</name>
  <value>oracle</value>
  <description> A password in clear text is NOT RECOMMENDED. Use an Oracle wallet
  instead.</description>
</property>

</configuration>
```

About the Target Table Metadata

You must provide Oracle Loader for Hadoop with information about the target table. The way that you provide this information depends on whether you run Oracle Loader for Hadoop in online or offline database mode. See "[About the Modes of Operation](#)" on page 3-2.

Providing the Connection Details for Online Database Mode

Oracle Loader for Hadoop uses table metadata from the Oracle database to identify the column names, data types, partitions, and so forth. The loader automatically fetches the metadata whenever a JDBC connection can be established.

Oracle recommends that you use a wallet to provide your credentials. To use an Oracle wallet, enter the following properties in the job configuration file:

- `oracle.hadoop.loader.connection.wallet_location`
- `oracle.hadoop.loader.connection.tns_admin`
- `oracle.hadoop.loader.connection.url` *or*
`oracle.hadoop.loader.connection.tnsEntryName`

Oracle recommends that you do not store passwords in clear text; use an Oracle wallet instead to safeguard your credentials. However, if you are not using an Oracle wallet, then enter these properties:

- `oracle.hadoop.loader.connection.url`
- `oracle.hadoop.loader.connection.user`
- `oracle.hadoop.loader.connection.password`

Generating the Target Table Metadata for Offline Database Mode

Under some circumstances, the loader job cannot access the database, such as when the Hadoop cluster is on a different network than Oracle Database. In such cases, you can use the OraLoaderMetadata utility to extract and store the target table metadata in a file.

To provide target table metadata in offline database mode:

1. Log in to the Oracle Database system.
2. The first time you use offline database mode, ensure that the software is installed and configured on the database system.
See "[Providing Support for Offline Database Mode](#)" on page 1-13.
3. Export the table metadata by running the OraLoaderMetadata utility program. See "[OraLoaderMetadata Utility](#)" on page 3-9.
4. Copy the generated XML file containing the table metadata to the Hadoop cluster.

- Use the `oracle.hadoop.loader.tableMetadataFile` property in the job configuration file to specify the location of the XML metadata file on the Hadoop cluster.

When the loader job runs, it accesses this XML document to discover the target table metadata.

OraLoaderMetadata Utility

Use the following syntax to run the OraLoaderMetadata utility on the Oracle Database system. You must enter the `java` command on a single line, although it is shown here on multiple lines for clarity:

```
java oracle.hadoop.loader.metadata.OraLoaderMetadata
  -user userName
  -connection_url connection
  [-schema schemaName]
  -table tableName
  -output fileName.xml
```

To see the OraLoaderMetadata Help file, use the command with no options.

Options

-user *userName*

The Oracle Database user who owns the target table. The utility prompts you for the password.

-connection_url *connection*

The database connection string in the thin-style service name format:

```
jdbc:oracle:thin:@//hostName:port/serviceName
```

If you are unsure of the service name, then enter this SQL command as a privileged user:

```
SQL> show parameter service
```

NAME	TYPE	VALUE
service_names	string	orcl

-schema *schemaName*

The name of the schema containing the target table. Unquoted values are capitalized, and unquoted values are used exactly as entered. If you omit this option, then the utility looks for the target table in the schema specified in the `-user` option.

-table *tableName*

The name of the target table. Unquoted values are capitalized, and unquoted values are used exactly as entered.

-output *fileName.xml*

The output file name used to store the metadata document.

[Example 3-2](#) shows how to store the target table metadata in an XML file.

Example 3-2 Generating Table Metadata

Run the OraLoaderMetadata utility:

```
$ java -cp '/tmp/oraloader-2.3.0-h1/jlib/*'
```

```
oracle.hadoop.loader.metadata.OraLoaderMetadata -user HR -connection_url
jdbc:oracle:thin://@localhost:1521/orcl.example.com -table EMPLOYEES -output
employee_metadata.xml
```

The OraLoaderMetadata utility prompts for the database password.

```
Oracle Loader for Hadoop Release 2.3.0 - Production
```

```
Copyright (c) 2011, 2013, Oracle and/or its affiliates. All rights reserved.
```

```
[Enter Database Password:] password
```

OraLoaderMetadata creates the XML file in the same directory as the script.

```
$ more employee_metadata.xml
<?xml version="1.0" encoding="UTF-8"?>
<!--
Oracle Loader for Hadoop Release 2.3.0 - Production

Copyright (c) 2011, 2013, Oracle and/or its affiliates. All rights reserved.

-->
<DATABASE>
<ROWSET><ROW>
<TABLE_T>
  <VERS_MAJOR>2</VERS_MAJOR>
  <VERS_MINOR>5 </VERS_MINOR>
  <OBJ_NUM>78610</OBJ_NUM>
  <SCHEMA_OBJ>
    <OBJ_NUM>78610</OBJ_NUM>
    <DATAOBJ_NUM>78610</DATAOBJ_NUM>
    <OWNER_NUM>87</OWNER_NUM>
    <OWNER_NAME>HR</OWNER_NAME>
    <NAME>EMPLOYEES</NAME>
  .
  .
  .
```

About Input Formats

An input format reads a specific type of data stored in Hadoop. Several input formats are available, which can read the data formats most commonly found in Hadoop:

- [Delimited Text Input Format](#)
- [Complex Text Input Formats](#)
- [Hive Table Input Format](#)
- [Avro Input Format](#)
- [Oracle NoSQL Database Input Format](#)

You can also use your own custom input formats. The descriptions of the built-in formats provide information that may help you develop custom Java InputFormat classes. See "[Custom Input Formats](#)" on page 3-14.

You specify a particular input format for the data that you want to load into a database table, by using the `mapreduce.inputformat.class` configuration property in the job configuration file.

Note: The built-in text formats do not handle header rows or newline characters (`\n`) embedded in quoted values.

Delimited Text Input Format

To load data from a delimited text file, set `mapreduce.inputformat.class` to `oracle.hadoop.loader.lib.input.DelimitedTextInputFormat`

About DelimitedTextInputFormat

The input file must comply with these requirements:

- Records must be separated by newline characters.
- Fields must be delimited using single-character markers, such as commas or tabs.

Any empty-string token, whether enclosed or unenclosed, is replaced by a null.

`DelimitedTextInputFormat` emulates the tokenization method of `SQL*Loader`: Terminated by `t`, and optionally enclosed by `ie`, or by `ie` and `te`.

`DelimitedTextInputFormat` uses the following syntax rules, where `t` is the field terminator, `ie` is the initial field encloser, `te` is the trailing field encloser, and `c` is one character.

- `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}`

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

This implementation enables you to define custom enclosers and terminator characters, but it hard codes the record terminator as a newline, and white space as `Java Character.isWhitespace`. A white space can be defined as the field terminator, but then that character is removed from the class of white space characters to prevent ambiguity.

Required Configuration Properties

None. The default format separates fields with commas and has no field enclosures.

Optional Configuration Properties

Use one or more of the following properties to define the field delimiters for `DelimitedTextInputFormat`:

- `oracle.hadoop.loader.input.fieldTerminator`
- `oracle.hadoop.loader.input.initialFieldEncloser`
- `oracle.hadoop.loader.input.trailingFieldEncloser`

Use the following property to provide names for the input fields:

- `oracle.hadoop.loader.input.fieldNames`

Complex Text Input Formats

To load data from text files that are more complex than `DelimitedTextInputFormat` can handle, set `mapreduce.inputformat.class` to

```
oracle.hadoop.loader.lib.input.RegexInputFormat
```

For example, a web log might delimit one field with quotes and another field with square brackets.

About `RegexInputFormat`

`RegexInputFormat` requires that records be separated by newline characters. It identifies fields in each text line by matching a regular expression:

- The regular expression must match the entire text line.
- The fields are identified using the capturing groups in the regular expression.

`RegexInputFormat` uses the `java.util.regex` regular expression-based pattern matching engine.

See Also: The *Java Platform Standard Edition 6 Java Reference* for more information about `java.util.regex`:

<http://docs.oracle.com/javase/6/docs/api/java/util/regex/package-summary.html>

Required Configuration Properties

Use the following property to describe the data input file:

- `oracle.hadoop.loader.input.regexPattern`

Optional Configuration Properties

Use the following property to identify the names of all input fields:

- `oracle.hadoop.loader.input.fieldNames`

Use this property to enable case-insensitive matches:

- `oracle.hadoop.loader.input.regexCaseInsensitive`

Hive Table Input Format

To load data from a Hive table, set `mapreduce.inputformat.class` to

```
oracle.hadoop.loader.lib.input.HiveToAvroInputFormat
```

About `HiveToAvroInputFormat`

`HiveToAvroInputFormat` imports the entire table, which includes all the files in the Hive table directory or, for partition tables, all files in each partition directory.

Oracle Loader for Hadoop rejects all rows with complex (non-primitive) column values. UNIONTYPE fields that resolve to primitive values are supported. See "[Handling Rejected Records](#)" on page 3-22.

`HiveToAvroInputFormat` transforms rows in the Hive table into Avro records, and capitalizes the Hive table column names to form the field names. This automatic capitalization improves the likelihood that the field names match the target table column names. See "[Mapping Input Fields to Target Table Columns](#)" on page 3-15.

Required Configuration Properties

You must specify the Hive database and table names using the following configuration properties:

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

Avro Input Format

To load data from binary Avro data files containing standard Avro-format records, set `mapreduce.inputformat.class` to

```
oracle.hadoop.loader.lib.input.AvroInputFormat
```

To process only files with the `.avro` extension, append `*.avro` to directories listed in the `mapred.input.dir` configuration property.

Configuration Properties

None

Oracle NoSQL Database Input Format

To load data from Oracle NoSQL Database, set `mapreduce.inputformat.class` to

```
oracle.kv.hadoop.KVAvroInputFormat
```

This input format is defined in Oracle NoSQL Database 11g, Release 2 and later releases.

About KVAvroInputFormat

Oracle Loader for Hadoop uses `KVAvroInputFormat` to read data directly from Oracle NoSQL Database.

`KVAvroInputFormat` passes the value but not the key from the key-value pairs in Oracle NoSQL Database. If you must access the Oracle NoSQL Database keys as Avro data values, such as storing them in the target table, then you must create a Java `InputFormat` class that implements `oracle.kv.hadoop.AvroFormatter`. Then you can specify the `oracle.kv.formatterClass` property in the Oracle Loader for Hadoop configuration file.

The `KVAvroInputFormat` class is a subclass of `org.apache.hadoop.mapreduce.InputFormat<oracle.kv.Key, org.apache.avro.generic.IndexedRecord>`

See Also: Javadoc for the `KVInputFormatBase` class at

<http://docs.oracle.com/cd/NOSQL/html/index.html>

Required Configuration Properties

You must specify the name and location of the key-value store using the following configuration properties:

- `oracle.kv.hosts`
- `oracle.kv.kvstore`

See "Oracle NoSQL Database Configuration Properties" on page 3-38.

Custom Input Formats

If the built-in input formats do not meet your needs, then you can write a Java class for a custom input format. The following information describes the framework in which an input format works in Oracle Loader for Hadoop.

About Implementing a Custom Input Format

Oracle Loader for Hadoop gets its input from a class extending `org.apache.hadoop.mapreduce.InputFormat`. You must specify the name of that class in the `mapreduce.inputformat.class` configuration property.

The input format must create `RecordReader` instances that return an Avro `IndexedRecord` input object from the `getCurrentValue` method. Use this method signature:

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

Oracle Loader for Hadoop uses the schema of the `IndexedRecord` input object to discover the names of the input fields and map them to the columns of the target table.

About Error Handling

If processing an `IndexedRecord` value results in an error, Oracle Loader for Hadoop uses the object returned by the `getCurrentKey` method of the `RecordReader` to provide feedback. It calls the `toString` method of the key and formats the result in an error message. `InputFormat` developers can assist users in identifying the rejected records by returning one of the following:

- Data file URI
- `InputSplit` information
- Data file name and the record offset in that file

Oracle recommends that you do not return the record in clear text, because it might contain sensitive information; the returned values can appear in Hadoop logs throughout the cluster. See ["Logging Rejected Records in Bad Files"](#) on page 3-23.

If a record fails and the key is null, then the loader generates no identifying information.

Supporting Data Sampling

Oracle Loader for Hadoop uses a sampler to improve performance of its MapReduce job. The sampler is multithreaded, and each sampler thread instantiates its own copy of the supplied `InputFormat` class. When implementing a new `InputFormat`, ensure that it is thread-safe. See ["Balancing Loads When Loading Data into Partitioned Tables"](#) on page 3-23.

InputFormat Source Code Example

Oracle Loader for Hadoop provides the source code for an `InputFormat` example, which is located in the `examples/jsrc/` directory.

The sample format loads data from a simple, comma-separated value (CSV) file. To use this input format, specify `oracle.hadoop.loader.examples.CSVInputFormat` as the value of `mapreduce.inputformat.class` in the job configuration file.

This input format automatically assigns field names of F0, F1, F2, and so forth. It does not have configuration properties.

Mapping Input Fields to Target Table Columns

Mapping identifies which input fields are loaded into which columns of the target table. You may be able to use the automatic mapping facilities, or you can always manually map the input fields to the target columns.

Automatic Mapping

Oracle Loader for Hadoop can automatically map the fields to the appropriate columns when the input data complies with these requirements:

- All columns of the target table are loaded.
- The input data field names in the `IndexedRecord` input object exactly match the column names.
- All input fields that are mapped to `DATE` columns can be parsed using the same Java date format.

Use these configuration properties for automatic mappings:

- `oracle.hadoop.loader.loaderMap.targetTable`: Identifies the target table.
- `oracle.hadoop.loader.defaultDateFormat`: Specifies a default date format that applies to all `DATE` fields.

Manual Mapping

For loads that do not comply with the requirements for automatic mapping, you must define additional properties. These properties enable you to:

- Load data into a subset of the target table columns.
- Create explicit mappings when the input field names are not exactly the same as the database column names.
- Specify different date formats for different input fields.

Use these properties for manual mappings:

- `oracle.hadoop.loader.loaderMap.targetTable` configuration property to identify the target table. Required.
- `oracle.hadoop.loader.loaderMap.columnNames`: Lists the columns to be loaded.
- `oracle.hadoop.loader.defaultDateFormat`: Specifies a default date format that applies to all `DATE` fields.
- `oracle.hadoop.loader.loaderMap.column_name.format`: Specifies the data format for a particular column.
- `oracle.hadoop.loader.loaderMap.column_name.field`: Identifies the name of an Avro record field mapped to a particular column.

Converting a Loader Map File

The following utility converts a loader map file from earlier releases to a configuration file:

```
hadoop oracle.hadoop.loader.metadata.LoaderMap -convert map_file conf_file
```

Options

map_file

The name of the input loader map file on the local file system (not HDFS).

conf_file

The name of the output configuration file on the local file system (not HDFS).

[Example 3–3](#) shows a sample conversion.

Example 3–3 Converting a Loader File to Configuration Properties

```
$ HADOOP_CLASSPATH="$HADOOP_CLASSPATH:$OLH_HOME/jlib/*"
$ hadoop oracle.hadoop.loader.metadata.LoaderMap -convert loadermap.xml conf.xml
Oracle Loader for Hadoop Release 2.3.0 - Production
```

Copyright (c) 2011, 2013, Oracle and/or its affiliates. All rights reserved.

Input Loader Map File loadermap.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<LOADER_MAP>
  <SCHEMA>HR</SCHEMA>
  <TABLE>EMPLOYEES</TABLE>
  <COLUMN field="F0">EMPLOYEE_ID</COLUMN>
  <COLUMN field="F1">LAST_NAME</COLUMN>
  <COLUMN field="F2">EMAIL</COLUMN>
  <COLUMN field="F3" format="MM-dd-yyyy">HIRE_DATE</COLUMN>
  <COLUMN field="F4">JOB_ID</COLUMN>
</LOADER_MAP>
```

Output Configuration File conf.xml

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<configuration>
  <property>
    <name>oracle.hadoop.loader.loaderMap.targetTable</name>
    <value>HR.EMPLOYEES</value>
  </property>
  <property>
    <name>oracle.hadoop.loader.loaderMap.columnNames</name>
    <value>EMPLOYEE_ID, LAST_NAME, EMAIL, HIRE_DATE, JOB_ID</value>
  </property>
  <property>
    <name>oracle.hadoop.loader.loaderMap.EMPLOYEE_ID.field</name>
    <value>F0</value>
  </property>
  <property>
    <name>oracle.hadoop.loader.loaderMap.EMPLOYEE_ID.format</name>
    <value></value>
  </property>
  <property>
    <name>oracle.hadoop.loader.loaderMap.LAST_NAME.field</name>
    <value>F1</value>
  </property>
  <property>
    <name>oracle.hadoop.loader.loaderMap.LAST_NAME.format</name>
    <value></value>
  </property>
</configuration>
```

```

        <name>oracle.hadoop.loader.loaderMap.EMAIL.field</name>
        <value>F2</value>
    </property>
    <property>
        <name>oracle.hadoop.loader.loaderMap.EMAIL.format</name>
        <value></value>
    </property>
    <property>
        <name>oracle.hadoop.loader.loaderMap.HIRE_DATE.field</name>
        <value>F3</value>
    </property>
    <property>
        <name>oracle.hadoop.loader.loaderMap.HIRE_DATE.format</name>
        <value>MM-dd-yyyy</value>
    </property>
    <property>
        <name>oracle.hadoop.loader.loaderMap.JOB_ID.field</name>
        <value>F4</value>
    </property>
    <property>
        <name>oracle.hadoop.loader.loaderMap.JOB_ID.format</name>
        <value></value>
    </property>
</configuration>

```

About Output Formats

In online database mode, you can choose between loading the data directly into an Oracle database table or storing it in a file. In offline database mode, you are restricted to storing the output data in a file, which you can load into the target table as a separate procedure. You specify the output format in the job configuration file using the `mapreduce.outputformat.class` property.

Choose from these output formats:

- **JDBC Output Format:** Loads the data directly into the target table.
- **Oracle OCI Direct Path Output Format:** Loads the data directly into the target table.
- **Delimited Text Output Format:** Stores the data in a local file.
- **Oracle Data Pump Output Format:** Stores the data in a local file.

JDBC Output Format

You can use a JDBC connection between the Hadoop system and Oracle Database to load the data. 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, in online database mode. No additional steps are required to load the data.

A JDBC connection must be open between the Hadoop cluster and the Oracle Database system for the duration of the job.

To use this output format, set `mapreduce.outputformat.class` to

```
oracle.hadoop.loader.lib.output.JDBCOutputFormat
```

About JDBCOutputFormat

JDBCOutputFormat uses standard JDBC batching to optimize performance and efficiency. If an error occurs during batch execution, such as a constraint violation, the JDBC driver stops execution immediately. 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.

The JDBC driver does not identify the row that caused the error, and so Oracle Loader for Hadoop does not know the insert status of any of the rows in the batch. It counts all rows in a batch with errors as "in question," that is, the rows may or may not be inserted in the target table. The loader then continues loading the next batch. It generates a load report at the end of the job that details the number of batch errors and the number of rows in question.

One way that you can handle this problem is by defining a unique key in the target table. For example, the HR.EMPLOYEES table has a primary key named EMPLOYEE_ID. After loading the data into HR.EMPLOYEES, you can query it by EMPLOYEE_ID to discover the missing employee IDs. Then you can locate the missing employee IDs in the input data, determine why they failed to load, and try again to load them.

Configuration Properties

To control the batch size, set this property:

```
oracle.hadoop.loader.connection.defaultExecuteBatch
```

Oracle OCI Direct Path Output Format

You can use the direct path interface of Oracle Call Interface (OCI) to load data into the target table. Each reducer loads into a distinct database partition in online database mode, enabling the performance gains of a parallel load. No additional steps are required to load the data.

The OCI connection must be open between the Hadoop cluster and the Oracle Database system for the duration of the job.

To use this output format, set `mapreduce.outputformat.class` to

```
oracle.hadoop.loader.lib.output.OCIOutputFormat
```

About OCIOutputFormat

OCIOutputFormat has the following restrictions:

- It is available only on the Linux x86.64 platform.
- The MapReduce job must create one or more reducers.
- The target table must be partitioned.
- For Oracle Database 11g (11.2.0.3), apply the patch for bug 13498646 if the target table is a composite interval partitioned table in which the subpartition key contains a CHAR, VARCHAR2, NCHAR, or NVARCHAR2 column. Later versions of Oracle Database do not require this patch.

Configuration Properties

To control the size of the direct path stream buffer, set this property:

```
oracle.hadoop.loader.output.dirpathBufsize
```

Delimited Text Output Format

You can create delimited text output files on the Hadoop cluster. The map or reduce tasks generate delimited text files, using the field delimiters and enclosers that you specify in the job configuration properties. Afterward, you can load the data into an Oracle database as a separate procedure. See "[About DelimitedTextOutputFormat](#)" on page 3-19.

This output format can use either an open connection to the Oracle Database system to retrieve the table metadata in online database mode, or a table metadata file generated by the `OraloaderMetadata` utility in offline database mode.

To use this output format, set `mapreduce.outputformat.class` to `oracle.hadoop.loader.lib.output.DelimitedTextOutputFormat`

About DelimitedTextOutputFormat

Output tasks generate delimited text format files, and one or more corresponding SQL*Loader control files, and SQL scripts for loading with external tables.

If the target table is not partitioned or if `oracle.hadoop.loader.loadByPartition` is `false`, then `DelimitedTextOutputFormat` generates the following files:

- A data file named `oraloader-taskId-csv-0.dat`.
- A SQL*Loader control file named `oraloader-csv.ctl` for the entire job.
- A SQL script named `oraloader-csv.sql` to load the delimited text file into the target table.

For partitioned tables, multiple output files are created with the following names:

- Data files: `oraloader-taskId-csv-partitionId.dat`
- SQL*Loader control files: `oraloader-taskId-csv-partitionId.ctl`
- SQL script: `oraloader-csv.sql`

In the generated file names, `taskId` is the mapper or reducer identifier, and `partitionId` is the partition identifier.

If the Hadoop cluster is connected to the Oracle Database system, then you can use Oracle SQL Connector for HDFS to load the delimited text data into an Oracle database. See [Chapter 2](#).

Alternatively, you can copy the delimited text files to the database system and load the data into the target table in one of the following ways:

- Use the generated control files to run SQL*Loader and load the data from the delimited text files.
- Use the generated SQL scripts to perform external table loads.

The files are located in the `${map.output.dir}/_olh` directory.

Configuration Properties

The following properties control the formatting of records and fields in the output files:

- `oracle.hadoop.loader.output.escapeEnclosers`
- `oracle.hadoop.loader.output.fieldTerminator`
- `oracle.hadoop.loader.output.initialFieldEncloser`

- `oracle.hadoop.loader.output.trailingFieldEncloser`

[Example 3-4](#) shows a sample SQL*Loader control file that might be generated by an output task.

Example 3-4 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 Format

You can create Data Pump format files on the Hadoop cluster. The map or reduce tasks generate Data Pump files. Afterward, you can load the data into an Oracle database as a separate procedure. See ["About DataPumpOutputFormat"](#) on page 3-20.

This output format can use either an open connection to the Oracle Database system in online database mode, or a table metadata file generated by the `OraloaderMetadata` utility in offline database mode.

To use this output format, set `mapreduce.outputformat.class` to

```
oracle.hadoop.loader.lib.output.DataPumpOutputFormat
```

About DataPumpOutputFormat

`DataPumpOutputFormat` generates data files with names in this format:

```
oraloader-taskId-dp-partitionId.dat
```

In the generated file names, *taskId* is the mapper or reducer identifier, and *partitionId* is the partition identifier.

If the Hadoop cluster is connected to the Oracle Database system, then you can use Oracle SQL Connector for HDFS to load the Data Pump files into an Oracle database. See [Chapter 2](#).

Alternatively, you can copy the Data Pump files to the database system and load them using a SQL script generated by Oracle Loader for Hadoop. The script performs the following tasks:

1. Creates an external table definition using the `ORACLE_DATAPUMP` access driver. The binary format Oracle Data Pump output files are listed in the `LOCATION` clause of the external table.
2. Creates a directory object that is used by the external table. You must uncomment this command before running the script. To specify the directory name used in the script, set the `oracle.hadoop.loader.extTabDirectoryName` property in the job configuration file.
3. Insert the rows from the external table into the target table. You must uncomment this command before running the script.

The SQL script is located in the `${map.output.dir}/_olh` directory.

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

Running a Loader Job

To run a job using Oracle Loader for Hadoop, you use the `OraLoader` utility in a `hadoop` command.

The following is the basic syntax:

```
hadoop jar $OLH_HOME/jlib/oraloader.jar oracle.hadoop.loader.OraLoader \
-conf job_config.xml \
-libjars input_file_format1.jar[,input_file_format2.jar...]
```

You can include any generic `hadoop` command-line option. `OraLoader` implements the `org.apache.hadoop.util.Tool` interface and follows the standard Hadoop methods for building MapReduce applications.

Basic Options

-conf job_config.xml

Identifies the job configuration file. See "[Creating a Job Configuration File](#)" on page 3-6.

-libjars

Identifies the JAR files for the input format.

- When using the example input format, specify `$OLH_HOME/jlib/oraloader-examples.jar`.
- When using the Hive or Oracle NoSQL Database input formats, you must specify additional JAR files, as described later in this section.
- When using a custom input format, specify its JAR file. (Also remember to add it to `HADOOP_CLASSPATH`.)

Separate multiple file names with commas, and list each one explicitly. Wildcard characters and spaces are not allowed.

Oracle Loader for Hadoop prepares internal configuration information for the MapReduce tasks. It stores table metadata information and the dependent Java libraries in the distributed cache, so that they are available to the MapReduce tasks throughout the cluster.

Example of Running OraLoader

The following example uses a built-in input format and a job configuration file named `MyConf.xml`:

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

```
hadoop jar $OLH_HOME/jlib/oraloader.jar oracle.hadoop.loader.OraLoader \
-conf MyConf.xml -libjars $OLH_HOME/jlib/oraloader-examples.jar
```

See Also:

- For the full hadoop command syntax and generic options:

<http://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-common/CommandsManual.html>

Specifying Hive Input Format JAR Files

When using `HiveToAvroInputFormat`, you must add the Hive configuration directory to the `HADOOP_CLASSPATH` environment variable:

```
HADOOP_CLASSPATH="$HADOOP_CLASSPATH:$OLH_HOME/jlib/*:hive_home/lib/*:hive_conf_dir"
```

You must also add the following Hive JAR files, in a comma-separated list, to the `-libjars` option of the hadoop command. Replace the stars (*) with the complete file names on your system:

- `hive-exec-*.jar`
- `hive-metastore-*.jar`
- `libfb303*.jar`

This example shows the full file names in Cloudera's Distribution including Apache Hadoop (CDH) 4.4:

```
# hadoop jar $OLH_HOME/jlib/oraloader.jar oracle.hadoop.loader.OraLoader \  
-conf MyConf.xml \  
-libjars \  
hive-exec-0.10.0-cdh4.4.0.jar,hive-metastore-0.10.0-cdh4.4.0.jar,libfb303-0.9.0.jar
```

Specifying Oracle NoSQL Database Input Format JAR Files

When using `KVAvroInputFormat` from Oracle NoSQL Database 11g, Release 2, you must include `$KVHOME/lib/kvstore.jar` in your `HADOOP_CLASSPATH` and you must include the `-libjars` option in the hadoop command:

```
hadoop jar $OLH_HOME/jlib/oraloader.jar oracle.hadoop.loader.OraLoader \  
-conf MyConf.xml \  
-libjars $KVHOME/lib/kvstore.jar
```

Job Reporting

Oracle Loader for Hadoop consolidates reporting information from individual tasks into a file named `${map.output.dir}/_olh/oraloader-report.txt`. Among other statistics, the report shows the number of errors, broken out by type and task, for each mapper and reducer.

Handling Rejected Records

Oracle Loader for Hadoop may reject input records for a variety of reasons, such as:

- Errors in the mapping properties
- Missing fields in the input data
- Records mapped to invalid table partitions

- Badly formed records, such as dates that do not match the date format or records that do not match regular expression patterns

Logging Rejected Records in Bad Files

By default, Oracle Loader for Hadoop does not log the rejected records into Hadoop logs; it only logs information on how to identify the rejected records. This practice prevents user-sensitive information from being stored in Hadoop logs across the cluster.

You can direct Oracle Loader for Hadoop to log rejected records by setting the `oracle.hadoop.loader.logBadRecords` configuration property to `true`. Then Oracle Loader for Hadoop logs bad records into one or more "bad" files in the `_olh/` directory under the job output directory.

Setting a Job Reject Limit

Some problems can cause Oracle Loader for Hadoop to reject every record in the input. To mitigate the loss of time and resources, Oracle Loader for Hadoop aborts the job after rejecting 1000 records.

You can change the maximum number of rejected records allowed by setting the `oracle.hadoop.loader.rejectLimit` configuration property. A negative value turns off the reject limit and allows the job to run to completion regardless of the number of rejected records.

Balancing Loads When Loading Data into Partitioned Tables

The goal of load balancing is to generate a MapReduce partitioning scheme that assigns approximately the same amount of work to all reducers.

The sampling feature of Oracle Loader for Hadoop balances loads across reducers when data is loaded into a partitioned database table. It generates an efficient MapReduce partitioning scheme that assigns database partitions to the reducers.

The execution time of a reducer is usually proportional to the number of records that it processes—the more records, the longer the execution time. When sampling 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 usually dominated by the execution time of its slowest reducer, unbalanced reducer loads slow down the entire job.

Using the Sampling Feature

You can turn the sampling feature on or off by setting the `oracle.hadoop.loader.sampler.enableSampling` configuration property. Sampling is turned on by default.

Tuning Load Balancing

These job configuration properties control the quality of load balancing:

- `oracle.hadoop.loader.sampler.maxLoadFactor`
- `oracle.hadoop.loader.sampler.loadCI`

The sampler uses the expected reducer load factor to evaluate the quality of its partitioning scheme. The **load factor** is the relative overload for each reducer,

calculated as $(assigned_load - ideal_load) / ideal_load$. This metric indicates how much a reducer's load deviates from a perfectly balanced reducer load. A load factor of 1.0 indicates a perfectly balanced load (no overload).

Small load factors indicate better load balancing. The `maxLoadFactor` default of 0.05 means that no reducer is ever overloaded by more than 5%. The sampler guarantees this `maxLoadFactor` with a statistical confidence level determined by the value 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` achieve more balanced reducer loads at the expense of longer sampling times. The default values of `maxLoadFactor=0.05` and `loadCI=0.95` are a good trade-off between load balancing quality and execution time.

Tuning 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 running time by setting the `oracle.hadoop.loader.sampler.maxSamplesPct` property, which specifies the maximum number of sampled records.

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

Oracle Loader for Hadoop uses the generated partitioning scheme only if sampling is successful. A sampling is successful if it generates a partitioning scheme with a 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, the sampler might be unsuccessful in generating a partitioning scheme using custom property values, such as when the constraints are too rigid or the number of required samples exceeds the user-specified maximum of `maxSamplesPct`. In these cases, Oracle Loader for Hadoop generates a log message identifying the problem, partitions the records using the database partitioning scheme, and does not guarantee load balancing.

Alternatively, you can reset the configuration properties to less rigid values. Either increase `maxSamplesPct`, or decrease `maxLoadFactor` or `loadCI`, or both.

Resolving Memory Issues

A custom input format may return input splits that do not fit in memory. If this happens, the sampler returns an out-of-memory error on the client node where the loader job is submitted.

To resolve 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`

If you are developing a custom input format, then see ["Custom Input Formats"](#) on page 3-14.

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.

Optimizing Communications Between Oracle Engineered Systems

If you are using Oracle Loader for Hadoop to load data from Oracle Big Data Appliance to Oracle Exadata Database Machine, then you can increase throughput by configuring the systems to use Sockets Direct Protocol (SDP) over the InfiniBand private network. This setup provides an additional connection attribute whose sole purpose is serving connections to Oracle Database to load data.

To specify SDP protocol:

1. Add JVM options to the `HADOOP_OPTS` environment variable to enable JDBC SDP export:

```
HADOOP_OPTS="-Doracle.net.SDP=true -Djava.net.preferIPv4Stack=true"
```

2. Configure the Oracle listener on Exadata with a specific port address for SDP (such as 1522). In the job configuration file, set `oracle.hadoop.loader.connection.url` to SDP using this syntax:

```
jdbc:oracle:thin:@(DESCRIPTION=(ADDRESS=(PROTOCOL=SDP)
(HOST=hostName) (PORT=portNumber))
(CONNECT_DATA=(SERVICE_NAME=serviceName)))
```

Replace *hostName*, *portNumber*, and *serviceName* with the appropriate values to identify Oracle Database on your Oracle Exadata Database Machine.

See Also: *Oracle Big Data Appliance Software User's Guide* for more information about configuring communications over InfiniBand

Oracle Loader for Hadoop Configuration Property Reference

OraLoader uses the standard methods of specifying configuration properties in the `hadoop` command. You can use the `-conf` option to identify configuration files, and the `-D` option to specify individual properties. See ["Running a Loader Job"](#) on page 3-21.

This section describes the OraLoader configuration properties, the Oracle NoSQL Database configuration properties, and a few generic Hadoop MapReduce properties that you typically must set for an OraLoader job:

- [MapReduce Configuration Properties](#)
- [OraLoader Configuration Properties](#)
- [Oracle NoSQL Database Configuration Properties](#)

A configuration file showing all OraLoader properties is in `$OLH_HOME/doc/oraloader-conf.xml`.

See Also: Hadoop documentation for job configuration files at <http://wiki.apache.org/hadoop/JobConfFile>

MapReduce Configuration Properties

mapred.job.name

Type: String

Default Value: OraLoader

Description: The Hadoop job name. A unique name can help you monitor the job using tools such as the Hadoop JobTracker web interface and Cloudera Manager.

mapred.input.dir

Type: String

Default Value: Not defined

Description: A comma-separated list of input directories.

mapreduce.inputformat.class

Type: String

Default Value: Not defined

Description: Identifies the format of the input data. You can enter one of the following built-in input formats, or the name of a custom `InputFormat` class:

- `oracle.hadoop.loader.lib.input.AvroInputFormat`
- `oracle.hadoop.loader.lib.input.DelimitedTextInputFormat`
- `oracle.hadoop.loader.lib.input.HiveToAvroInputFormat`
- `oracle.hadoop.loader.lib.input.RegexInputFormat`
- `oracle.kv.hadoop.KVAvroInputFormat`

See "[About Input Formats](#)" on page 3-10 for descriptions of the built-in input formats.

mapred.output.dir

Type: String

Default Value: Not defined

Description: A comma-separated list of output directories, which cannot exist before the job runs. Required.

mapreduce.outputformat.class

Type: String

Default Value: Not defined

Description: Identifies the output type. The values can be:

- `oracle.hadoop.loader.lib.output.DataPumpOutputFormat`
Writes data records into binary format files that can be loaded into the target table using an external table.
- `oracle.hadoop.loader.lib.output.DelimitedTextOutputFormat`
Writes data records to delimited text format files such as comma-separated values (CSV) format files.
- `oracle.hadoop.loader.lib.output.JDBCOutputFormat`

Inserts rows into the target table using a JDBC connection.

- `oracle.hadoop.loader.lib.output.OCIOutputFormat`

Inserts rows into the target table using the Oracle OCI Direct Path interface.

See "[About Output Formats](#)" on page 3-17.

OraLoader Configuration Properties

oracle.hadoop.loader.badRecordFlushInterval

Type: Integer

Default Value: 500

Description: Sets the maximum number of records that a task attempt can log before flushing the log file. This setting limits the number of records that can be lost when the record reject limit (`oracle.hadoop.loader.rejectLimit`) is reached and the job stops running.

The `oracle.hadoop.loader.logBadRecords` property must be set to `true` for a flush interval to take effect.

oracle.hadoop.loader.compressionFactors

Type: Decimal

Default Value: BASIC=5.0, OLTP=5.0, QUERY_LOW=10.0, QUERY_HIGH=10.0, ARCHIVE_LOW=10.0, ARCHIVE_HIGH=10.0

Description: Defines the compression factor for different types of compression. The value is a comma-delimited list of *name=value* pairs. The name can be one of the following keywords:

```
ARCHIVE_HIGH
ARCHIVE_LOW
BASIC
OLTP
QUERY_HIGH
QUERY_LOW
```

oracle.hadoop.loader.connection.defaultExecuteBatch

Type: Integer

Default Value: 100

Description: The number of records inserted in one trip to the database. It applies only to `JDBCOutputFormat` and `OCIOutputFormat`.

Specify a value greater than or equal to 1. Although the maximum value is unlimited, very large batch sizes are not recommended because they result in a large memory footprint without much increase in performance.

A value less than 1 sets the property to the default value.

oracle.hadoop.loader.connection.oci_url

Type: String

Default Value: Value of `oracle.hadoop.loader.connection.url`

Description: The database connection string used by `OCIOutputFormat`. This property enables the OCI client to connect to the database using different connection parameters than the JDBC connection URL.

The following example specifies Socket Direct Protocol (SDP) for OCI connections.

```
(DESCRIPTION=(ADDRESS_LIST=(ADDRESS=(PROTOCOL=SDP)(HOST=myhost)(PORT=1521)))(CONNECT_DATA=(SERVICE_NAME=my_db_service_name)))
```

This connection string does not require a "jdbc:oracle:thin:@" prefix. All characters up to and including the first at-sign (@) are removed.

oracle.hadoop.loader.connection.password

Type: String

Default Value: Not defined

Description: Password for the connecting user. Oracle recommends that you do not store your password in clear text. Use an Oracle wallet instead.

oracle.hadoop.loader.connection.sessionTimeZone

Type: String

Default Value: LOCAL

Description: Alters the session time zone for database connections. Valid values are:

- [+|-]hh:mm: Hours and minutes before or after Coordinated Universal Time (UTC), such as -5:00 for Eastern Standard Time
- LOCAL: The default time zone of the JVM
- *time_zone_region*: A valid JVM time zone region, such as EST (for Eastern Standard Time) or America/New_York

This property also determines the default time zone for input data that is loaded into `TIMESTAMP WITH TIME ZONE` and `TIMESTAMP WITH LOCAL TIME ZONE` database column types.

oracle.hadoop.loader.connection.tns_admin

Type: String

Default Value: Not defined

Description: File path to a directory on each node of the Hadoop cluster, which contains SQL*Net configuration files such as `sqlnet.ora` and `tnsnames.ora`. Set this property so that you can use TNS entry names in database connection strings.

You must set this property when using an Oracle wallet as an external password store. See [oracle.hadoop.loader.connection.wallet_location](#).

oracle.hadoop.loader.connection.tnsEntryName

Type: String

Default Value: Not defined

Description: A TNS entry name defined in the `tnsnames.ora` file. Use this property with [oracle.hadoop.loader.connection.tns_admin](#).

oracle.hadoop.loader.connection.url

Type: String

Default Value: Not defined

Description: The URL of the database connection. This property overrides all other connection properties.

If an Oracle wallet is configured as an external password store, then the property value must start with the `jdbc:oracle:thin:@` driver prefix, and the database connection

string must exactly match the credential in the wallet. See [oracle.hadoop.loader.connection.wallet_location](#).

The following examples show valid values of connection URLs:

- Oracle Net Format:

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

- Oracle Net Format for InfiniBand:

```
jdbc:oracle:thin:@(DESCRIPTION=(ADDRESS=(PROTOCOL=SDP)
  (HOST=myhost) (PORT=1522))
(CONNECT_DATA=(SERVICE_NAME=example_service_name)))
```

- TNS Entry Format:

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

- Thin Style:

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

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

oracle.hadoop.loader.connection.user

Type: String

Default Value: Not defined

Description: A database user name. When using online database mode, you must set either this property or [oracle.hadoop.loader.connection.wallet_location](#).

oracle.hadoop.loader.connection.wallet_location

Type: String

Default Value: Not defined

Description: File path to an Oracle wallet directory on each node of the Hadoop cluster, where the connection credentials are stored.

When using an Oracle wallet, you must also set the following properties:

- [oracle.hadoop.loader.connection.tns_admin](#)
- [oracle.hadoop.loader.connection.url](#) *or* [oracle.hadoop.loader.connection.tnsEntryName](#)

oracle.hadoop.loader.defaultDateFormat

Type: String

Default Value: yyyy-MM-dd HH:mm:ss

Description: Parses an input field into a DATE column using a `java.text.SimpleDateFormat` pattern and the default locale. If the input file requires different patterns for different fields, then use the manual mapping properties. See "Manual Mapping" on page 3-15.

oracle.hadoop.loader.enableSorting

Type: Boolean

Default Value: true

Description: Controls whether output records within each reducer group are sorted. Use the `oracle.hadoop.loader.sortKey` property to identify the columns of the target table to sort by. Otherwise, Oracle Loader for Hadoop sorts the records by the primary key.

oracle.hadoop.loader.extTabDirectoryName

Type: String

Default Value: OLH_EXTTAB_DIR

Description: The name of the database directory object for the external table `LOCATION` data files. Oracle Loader for Hadoop does not copy data files into this directory; the file output formats generate a SQL file containing external table DDL, where the directory name appears.

This property applies only to `DelimitedTextOutputFormat` and `DataPumpOutputFormat`.

oracle.hadoop.loader.input.fieldNames

Type: String

Default Value: F0, F1, F2, . . .

Description: A comma-delimited list of names for the input fields.

For the built-in input formats, specify names for all fields in the data, not just the fields of interest. If an input line has more fields than this property has field names, then the extra fields are discarded. If a line has fewer fields than this property has field names, then the extra fields are set to null. See "[Mapping Input Fields to Target Table Columns](#)" on page 3-15 for loading only selected fields.

The names are used to create the Avro schema for the record, so they must be valid JSON name strings.

oracle.hadoop.loader.input.fieldTerminator

Type: String

Default Value: , (comma)

Description: A character that indicates the end of an input field for `DelimitedTextInputFormat`. The value can be either a single character or `\uHHHH`, where `HHHH` is the character's UTF-16 encoding.

oracle.hadoop.loader.input.hive.databaseName

Type: String

Default Value: Not defined

Description: The name of the Hive database where the input table is stored.

oracle.hadoop.loader.input.hive.tableName

Type: String

Default Value: Not defined

Description: The name of the Hive table where the input data is stored.

oracle.hadoop.loader.input.initialFieldEncloser

Type: String

Default Value: Not defined

Description: A character that indicates the beginning of a field. The value can be either a single character or `\uHHHH`, where `HHHH` is the character's UTF-16 encoding. To

restore the default setting (no enclosure), enter a zero-length value. A field enclosure cannot equal the terminator or white-space character defined for the input format.

When this property is set, the parser attempts to read each field as an enclosed token (value) before reading it as an unenclosed token. If the field enclosures are not set, then the parser reads each field as an unenclosed token.

If you set this property but not `oracle.hadoop.loader.input.trailingFieldEncloser`, then the same value is used for both properties.

oracle.hadoop.loader.input.regexCaseInsensitive

Type: Boolean

Default Value: false

Description: Controls whether pattern matching is case-sensitive. Set to true to ignore case, so that "string" matches "String", "STRING", "string", "StRiNg", and so forth. By default, "string" matches only "string".

This property is the same as the `input.regex.case.insensitive` property of `org.apache.hadoop.hive.contrib.serde2.RegexSerDe`.

oracle.hadoop.loader.input.regexPattern

Type: Text

Default Value: Not defined

Description: The pattern string for a regular expression.

The regular expression must match each text line in its entirety. For example, a correct regex pattern for input line "a,b,c," is "`([^\,]*) , ([^\,]*) , ([^\,]*) , "`. However, "`([^\,]*) , "` is invalid, because the expression is *not* applied repeatedly to a line of input text.

`RegexInputFormat` uses the capturing groups of regular expression matching as fields. The special group zero is ignored because it stands for the entire input line.

This property is the same as the `input.regex` property of `org.apache.hadoop.hive.contrib.serde2.RegexSerDe`.

See Also: For descriptions of regular expressions and capturing groups, the entry for `java.util.regex` in the *Java Platform Standard Edition 6 API Specification* at

<http://docs.oracle.com/javase/6/docs/api/java/util/regex/Pattern.html>

oracle.hadoop.loader.input.trailingFieldEncloser

Type: String

Default Value: The value of `oracle.hadoop.loader.input.initialFieldEncloser`

Description: Identifies a character that marks the end of a field. The value can be either a single character or `\uHHHH`, where `HHHH` is the character's UTF-16 encoding. For no trailing encloser, enter a zero-length value.

A field enclosure cannot be the terminator or a white-space character defined for the input format.

If the trailing field enclosure character is embedded in an input field, then the character must be doubled up to be parsed as literal text. For example, an input field must have `' '` (two single quotes) to load `'` (one single quote).

If you set this property, then you must also set [oracle.hadoop.loader.input.initialFieldEncloser](#).

oracle.hadoop.loader.loadByPartition

Type: Boolean

Default Value: true

Description: Specifies a partition-aware load. Oracle Loader for Hadoop organizes the output by partition for all output formats on the Hadoop cluster; this task does not impact the resources of the database system.

`DelimitedTextOutputFormat` and `DataPumpOutputFormat` generate multiple files, and each file contains the records from one partition. For `DelimitedTextOutputFormat`, this property also controls whether the `PARTITION` keyword appears in the generated control files for `SQL*Loader`.

`OCIOutputFormat` requires partitioned tables. If you set this property to `false`, then `OCIOutputFormat` turns it back on. For the other output formats, you can set `loadByPartition` to `false`, so that Oracle Loader for Hadoop handles a partitioned table as if it were unpartitioned.

oracle.hadoop.loader.loaderMap.columnNames

Type: String

Default Value: Not specified? Or all columns?

Description: A comma-separated list of column names in the target table, in any order. The names can be quoted or unquoted. Quoted names begin and end with double quotes (") and are used exactly as entered. Unquoted names are converted to upper case.

You must set [oracle.hadoop.loader.loaderMap.targetTable](#), or this property is ignored. You can optionally set [oracle.hadoop.loader.loaderMap.column_name.field](#) and [oracle.hadoop.loader.loaderMap.column_name.format](#).

oracle.hadoop.loader.loaderMap.column_name.field

Type: String

Default Value: Normalized column name

Description: The name of a field that contains Avro records, which is mapped to the column identified in the property name. The column name can be quoted or unquoted. A quoted name begins and ends with double quotes (") and is used exactly as entered. An unquoted name is converted to upper case. Optional.

You must set [oracle.hadoop.loader.loaderMap.columnNames](#), or this property is ignored.

oracle.hadoop.loader.loaderMap.column_name.format

Type: String

Default Value: Not defined

Description: Specifies the data format of the data being loaded into the column identified in the property name. Use a `java.text.SimpleDateFormat` pattern for a date format or regular expression patterns for text. Optional.

You must set [oracle.hadoop.loader.loaderMap.columnNames](#), or this property is ignored.

oracle.hadoop.loader.loaderMap.targetTable

Type: String

Default Value: Not defined

Description: A schema-qualified name for the table to be loaded. This property takes precedence over `oracle.hadoop.loader.loaderMapFile`.

To load a subset of columns, set the `oracle.hadoop.loader.loaderMap.columnNames` property. With `columnNames`, you can optionally set `oracle.hadoop.loader.loaderMap.column_name.field` to specify the names of the fields that are mapped to the columns, and `oracle.hadoop.loader.loaderMap.column_name.format` to specify the format of the data in those fields. If all the columns of a table will be loaded, and the input field names match the database column names, then you do not need to set `columnNames`.

oracle.hadoop.loader.loaderMapFile

Loader maps are deprecated in Release 2.3. The `oracle.hadoop.loader.loaderMap.*` configuration properties replace loader map files. See "Manual Mapping" on page 3-15.

oracle.hadoop.loader.logBadRecords

Type: Boolean

Default Value: false

Description: Controls whether Oracle Loader for Hadoop logs bad records to a file.

This property applies only to records rejected by input formats and mappers. It does not apply to errors encountered by the output formats or by the sampling feature.

oracle.hadoop.loader.log4j.propertyPrefix

Type: String

Default Value: `log4j.logger.oracle.hadoop.loader`

Description: Identifies the prefix used in Apache log4j properties loaded from its configuration file.

Oracle Loader for Hadoop enables you to specify log4j properties in the `hadoop` command using the `-conf` and `-D` options. For example:

```
-D log4j.logger.oracle.hadoop.loader.OraLoader=DEBUG
-D log4j.logger.oracle.hadoop.loader.metadata=INFO
```

All configuration properties starting with this prefix are loaded into log4j. They override the settings for the same properties that log4j loaded from `/${log4j.configuration}`. The overrides apply to the Oracle Loader for Hadoop job driver, and its map and reduce tasks.

The configuration properties are copied to log4j with RAW values; any variable expansion is done in the context of log4j. Any configuration variables to be used in the expansion must also start with this prefix.

oracle.hadoop.loader.olh_home

Type: String

Default Value: Value of the `OLH_HOME` environment variable

Description: The path of the Oracle Loader for Hadoop home directory on the node where you start the OraLoader job. This path identifies the location of the required libraries.

oracle.hadoop.loader.olhcachePath

Type: String

Default Value: `${mapred.output.dir}/../olhcache`

Description: Identifies the full path to an HDFS directory where Oracle Loader for Hadoop can create files that are loaded into the MapReduce distributed cache.

The distributed cache is a facility for caching large, application-specific files and distributing them efficiently across the nodes in a cluster.

See Also: The description of `org.apache.hadoop.filecache.DistributedCache` in the Java documentation at

<http://hadoop.apache.org/>

oracle.hadoop.loader.output.dirpathBufsize

Type: Integer

Default Value: 131072 (128 KB)

Description: Sets the size in bytes of the direct path stream buffer for `OCIOutputFormat`. Values are rounded up to the next multiple of 8 KB.

oracle.hadoop.loader.output.escapeEnclosers

Type: Boolean

Default Value: false

Description: Controls whether the embedded trailing enclosure character is handled as literal text (that is, escaped). Set this property to `true` when a field may contain the trailing enclosure character as part of the data value. See [oracle.hadoop.loader.output.trailingFieldEncloser](#).

oracle.hadoop.loader.output.fieldTerminator

Type: String

Default Value: , (comma)

Description: A character that indicates the end of an output field for `DelimitedTextInputFormat`. The value can be either a single character or `\uHHHH`, where `HHHH` is the character's UTF-16 encoding.

oracle.hadoop.loader.output.granuleSize

Type: Integer

Default Value: 10240000

Description: The granule size in bytes for generated Data Pump files.

A granule determines the work load for a parallel process (PQ slave) when loading a file through the `ORACLE_DATAPUMP` access driver.

See Also: *Oracle Database Utilities* for more information about the `ORACLE_DATAPUMP` access driver.

oracle.hadoop.loader.output.initialFieldEncloser

Type: String

Default Value: Not defined

Description: A character generated in the output to identify the beginning of a field. The value must be either a single character or `\uHHHH`, where `HHHH` is the character's UTF-16 encoding. A zero-length value means that no enclosers are generated in the output (default value).

Use this property when a field may contain the value of `oracle.hadoop.loader.output.fieldTerminator`. If a field may also contain the value of `oracle.hadoop.loader.output.trailingFieldEncloser`, then set `oracle.hadoop.loader.output.escapeEnclosers` to true.

If you set this property, then you must also set `oracle.hadoop.loader.output.trailingFieldEncloser`.

oracle.hadoop.loader.output.trailingFieldEncloser

Type: String

Default Value: Value of `oracle.hadoop.loader.output.initialFieldEncloser`

Description: A character generated in the output to identify the end of a field. The value must be either a single character or `\uHHHH`, where `HHHH` is the character's UTF-16 encoding. A zero-length value means that there are no enclosers (default value).

Use this property when a field may contain the value of `oracle.hadoop.loader.output.fieldTerminator`. If a field may also contain the value of `oracle.hadoop.loader.output.trailingFieldEncloser`, then set `oracle.hadoop.loader.output.escapeEnclosers` to true.

If you set this property, then you must also set `oracle.hadoop.loader.output.initialFieldEncloser`.

oracle.hadoop.loader.rejectLimit

Type: Integer

Default Value: 1000

Description: The maximum number of rejected or skipped records allowed before the job stops running. A negative value turns off the reject limit and allows the job to run to completion.

If `mapred.map.tasks.speculative.execution` is true (the default), then the number of rejected records may be inflated temporarily, causing the job to stop prematurely.

Input format errors do not count toward the reject limit because they are irrecoverable and cause the map task to stop. Errors encountered by the sampling feature or the online output formats do not count toward the reject limit either.

oracle.hadoop.loader.sampler.enableSampling

Type: Boolean

Default Value: true

Description: Controls whether the sampling feature is enabled. Set this property to false to disable sampling.

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

oracle.hadoop.loader.sampler.hintMaxSplitSize

Type: Integer

Default Value: 1048576 (1 MB)

Description: Sets the Hadoop `mapred.max.split.size` property for the sampling process; the value of `mapred.max.split.size` does not change for the job configuration. A value less than 1 is ignored.

Some input formats (such as `FileInputFormat`) use this property as a hint to determine the number of splits returned by `getSplits`. Smaller values imply that more chunks of data are sampled at random, which results in a better sample.

Increase this value for data sets with tens of terabytes of data, or if the input format `getSplits` method throws an out-of-memory error.

Although large splits are better for I/O performance, they are not necessarily better for sampling. Set this value small enough for good sampling performance, but no smaller. Extremely small values can cause inefficient I/O performance, and can cause `getSplits` to run out of memory by returning too many splits.

The `org.apache.hadoop.mapreduce.lib.input.FileInputFormat` method always returns splits at least as large as the minimum split size setting, regardless of the value of this property.

oracle.hadoop.loader.sampler.hintNumMapTasks

Type: Integer

Default Value: 100

Description: Sets the value of the Hadoop `mapred.map.tasks` configuration property for the sampling process; the value of `mapred.map.tasks` does not change for the job configuration. A value less than 1 is ignored.

Some input formats (such as `DBInputFormat`) use this property as a hint to determine the number of splits returned by the `getSplits` method. Higher values imply that more chunks of data are sampled at random, which results in a better sample.

Increase this value for data sets with more than a million rows, but remember that extremely large values can cause `getSplits` to run out of memory by returning too many splits.

oracle.hadoop.loader.sampler.loadCl

Type: Decimal

Default Value: 0.95

Description: The statistical confidence indicator for the maximum reducer load factor.

This property accepts values greater than or equal to 0.5 and less than 1 ($0.5 \leq \text{value} < 1$). A value less than 0.5 resets the property to the default value. Typical values are 0.90, 0.95, and 0.99.

See [oracle.hadoop.loader.sampler.maxLoadFactor](#).

oracle.hadoop.loader.sampler.maxHeapBytes

Type: Integer

Default Value: -1

Description: Specifies in bytes the maximum amount of memory available to the sampler.

Sampling stops when one of these conditions is true:

- The sampler has collected the minimum number of samples required for load balancing.

- The percent of sampled data exceeds [oracle.hadoop.loader.sampler.maxSamplesPct](#).
- The number of sampled bytes exceeds [oracle.hadoop.loader.sampler.maxHeapBytes](#). This condition is not imposed when the property is set to a negative value.

oracle.hadoop.loader.sampler.maxLoadFactor

Type: Float

Default Value: 0.05 (5%)

Description: The maximum acceptable load factor for a reducer. A value of 0.05 indicates that reducers can be assigned up to 5% more data than their ideal load.

This property accepts values greater than 0. A value less than or equal to 0 resets the property to the default value. Typical values are 0.05 and 0.1.

In a perfectly balanced load, every reducer is assigned an equal amount of work (or load). The load factor is the relative overload for each reducer, calculated as $(assigned_load - ideal_load) / ideal_load$. If load balancing is successful, the job runs within the maximum load factor at the specified confidence.

See [oracle.hadoop.loader.sampler.loadCI](#).

oracle.hadoop.loader.sampler.maxSamplesPct

Type: Float

Default Value: 0.01 (1%)

Description: Sets the maximum sample size as a fraction 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.

This property accepts a range of 0 to 1 (0% to 100%). A negative value disables it.

Sampling stops when one of these conditions is true:

- The sampler has collected the minimum number of samples required for load balancing, which can be fewer than the number set by this property.
- The percent of sampled data exceeds [oracle.hadoop.loader.sampler.maxSamplesPct](#).
- The number of sampled bytes exceeds [oracle.hadoop.loader.sampler.maxHeapBytes](#). This condition is not imposed when the property is set to a negative value.

oracle.hadoop.loader.sampler.minSplits

Type: Integer

Default Value: 5

Description: The minimum number of input splits that the sampler reads from before it makes any evaluation of the stopping condition. If the total number of input splits is less than `minSplits`, then the sampler reads from all the input splits.

A number less than or equal to 0 is the same as a value of 1.

oracle.hadoop.loader.sampler.numThreads

Type: Integer

Default Value: 5

Description: The number of sampler threads. A higher number of threads allows higher concurrency in sampling. A value of 1 disables multithreading for the sampler.

Set the value based on the processor and memory resources available on the node where you start the Oracle Loader for Hadoop job.

oracle.hadoop.loader.sortKey

Type: String

Default Value: Not defined

Description: A comma-delimited list of column names that forms a key for sorting output records within a reducer group.

The column names can be quoted or unquoted identifiers:

- A quoted identifier begins and ends with double quotation marks (").
- An unquoted identifier is converted to uppercase before use.

oracle.hadoop.loader.tableMetadataFile

Type: String

Default Value: Not defined

Description: Path to the target table metadata file. Set this property when running in offline database mode.

Use the `file://` syntax to specify a local file, for example:

```
file:///home/jdoe/metadata.xml
```

To create the table metadata file, run the `OraLoaderMetadata` utility. See "[OraLoaderMetadata Utility](#)" on page 3-9.

oracle.hadoop.loader.targetTable

Deprecated. Use `oracle.hadoop.loader.loaderMap.targetTable`.

Oracle NoSQL Database Configuration Properties

oracle.kv.kvstore

Type: String

Default Value: Not defined

Description: The name of the KV store with the source data.

oracle.kv.hosts

Type: String

Default Value: Not defined

Description: An array of one or more `hostname:port` pairs that identify the hosts in the KV store with the source data. Separate multiple pairs with commas.

oracle.kv.batchSize

Type: Key

Default Value: Not defined

Description: The desired number of keys for `KVAvroInputFormat` to fetch during each network round trip. A value of zero (0) sets the property to a default value.

oracle.kv.parentKey**Type:** String**Default Value:** Not defined

Description: Restricts the returned values to only the child key-value pairs of the specified key. A major key path must be a partial path, and a minor key path must be empty. A null value (the default) does not restrict the output, and so `KVAvroInputFormat` returns all keys in the store.

oracle.kv.subRange**Type:** KeyRange**Default Value:** Not defined

Description: Further restricts the returned values to a particular child under the parent key specified by `oracle.kv.parentKey`.

oracle.kv.depth**Type:** Depth**Default Value:** PARENT_AND_DESCENDANTS

Description: Restricts the returned values to a particular hierarchical depth under the value of `oracle.kv.parentKey`. The following keywords are valid values:

- CHILDREN_ONLY: Returns the children, but not the specified parent.
- DESCENDANTS_ONLY: Returns all descendants, but not the specified parent.
- PARENT_AND_CHILDREN: Returns the children and the parent.
- PARENT_AND_DESCENDANTS: Returns all descendants and the parent.

oracle.kv.consistency**Type:** Consistency**Default Value:** NONE_REQUIRED

Description: The consistency guarantee for reading child key-value pairs. The following keywords are valid values:

- ABSOLUTE: Requires the master to service the transaction so that consistency is absolute.
- NONE_REQUIRED: Allows replicas to service the transaction, regardless of the state of the replicas relative to the master.

oracle.kv.timeout**Type:** Long**Default Value:**

Description: Sets a maximum time interval in milliseconds for retrieving a selection of key-value pairs. A value of zero (0) sets the property to its default value.

oracle.kv.formatterClass**Type:** String**Default Value:** Not defined

Description: Specifies the name of a class that implements the `AvroFormatter` interface to format `KeyValueVersion` instances into Avro `IndexedRecord` strings.

Because the Avro records from Oracle NoSQL Database pass directly to Oracle Loader for Hadoop, the NoSQL keys are not available for mapping into the target Oracle

Database table. However, the formatter class receives both the NoSQL key and value, enabling the class to create and return a new Avro record that contains both the value and key, which can be passed to Oracle Loader for Hadoop.

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 11g Release 2 (11.2) client libraries. For information about third party products included with Oracle Database 11g 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 1.7.3

Licensed under the Apache License, Version 2.0 (the "License"); you may not use Apache Avro 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 the Apache Commons Mathematics library 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.

Jackson JSON 1.8.8

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this library 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.

Oracle Data Integrator Application Adapter for Hadoop

This chapter describes how to use the knowledge modules in Oracle Data Integrator (ODI) Application Adapter for Hadoop. It contains the following sections:

- [Introduction](#)
- [Setting Up the Topology](#)
- [Setting Up an Integration Project](#)
- [Creating an Oracle Data Integrator Model from a Reverse-Engineered Hive Model](#)
- [Designing the Interface](#)

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

Introduction

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

Oracle Data Integrator (ODI) 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 (KMs), the application 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 an Oracle database for further processing and generating reports

Knowledge modules (KMs) contain the information needed by Oracle Data Integrator to perform a specific set of tasks against a specific technology. An application adapter is a group of knowledge modules. Thus, Oracle Data Integrator Application Adapter for Hadoop is a group of knowledge modules for accessing data stored in Hadoop.

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, when you use Oracle Data

Integrator and Oracle Data Integrator Application Adapter for Hadoop, you do not need to write MapReduce jobs. Oracle Data Integrator uses Apache Hive and the Hive Query Language (HiveQL), a SQL-like language for implementing MapReduce jobs.

When you implement 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 it by using HiveQL like you use 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 processed into a smaller data set, you can load it into an Oracle database for further processing and analysis. Oracle Loader for Hadoop is recommended for optimal loading into an Oracle database.

Knowledge Modules

Oracle Data Integrator provides the knowledge modules (KMs) 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 knowledge module supports wildcards (*,?).	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, Oracle SQL Connector for HDFS, or both.	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

To set up the topology in Oracle Data Integrator, you identify 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 File Data Sources](#)
- [Setting Up Hive Data Sources](#)
- [Setting Up the Oracle Data Integrator Agent to Execute Hadoop Jobs](#)
- [Configuring Oracle Data Integrator Studio for Executing Hadoop Jobs on the Local Agent](#)

Note: Many of the environment variables described in the following sections are already configured for Oracle Big Data Appliance. See the configuration script at `/opt/oracle/odiagent-version/agent_standalone/oracledi/agent/bin/HadoopEnvSetup.sh`

Setting Up File Data Sources

In the Hadoop context, there is a distinction between files in 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 Logical Schema.
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://bdalnode01.example.com:8020
```

Note: No dedicated technology is defined for HDFS files.

Setting Up Hive Data Sources

The following steps in Oracle Data Integrator are required for connecting to a Hive system. Oracle Data Integrator connects to Hive by 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` directory.

You must add all Hive-specific flex fields. For pre-11.1.1.6.0 repositories, the flex fields are added during the repository upgrade process.

To set up a Hive data source:

1. Ensure that the following environment variables are set, and note their values. The following list shows typical values, although your installation may be different:
 - `$HIVE_HOME`: `/usr/lib/hive`
 - `$HADOOP_HOME`: `/usr/lib/hadoop` (contains configuration files such as `core-site.xml`)
 - `$OSCH_HOME`: `/opt/oracle/orahdfs-version`
2. Open `~/.odi/oracledi/userlib/additional_path.txt` in a text editor and add the paths listed in [Table 4–2](#). Enter the full path obtained in Step 1 instead of the variable name.

This step enables ODI Studio to access the JAR files.

Table 4–2 JAR File Paths

Description	CDH4 Path	CDH3 Path
Hive JAR Files	<code>\$HIVE_HOME/lib/*.jar</code>	<code>\$HIVE_HOME/*.jar</code>
Hadoop Client JAR Files	<code>\$HADOOP_HOME/client/*.jar</code>	<code>\$HADOOP_HOME/hadoop-*-core*.jar¹</code> <code>\$HADOOP_HOME/hadoop-*-tools*.jar¹</code>
Hadoop Configuration Directory	<code>\$HADOOP_HOME</code>	<code>\$HADOOP_HOME</code>
Oracle SQL Connector for HDFS JAR Files (optional)	<code>\$OSCH_HOME/jlib/*.jar</code>	<code>\$OSCH_HOME/jlib/*.jar</code>

¹ Replace the stars (*) with the full file name.

3. Ensure that the Hadoop configuration directory is in the ODI class path.
The Hadoop configuration directory contains files such as `core-default.xml`, `core-site.xml`, and `hdfs-site.xml`.
4. Create a DataServer object under Hive technology.
5. Set the following locations under JDBC:
 - JDBC Driver: `org.apache.hadoop.hive.jdbc.HiveDriver`
 - JDBC URL: for example, `jdbc:hive://BDA:10000/default`
6. Set the following under Flexfields:
 - Hive Metastore URIs: for example, `thrift://BDA:10000`
7. 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.
8. Ensure that the Hive server is up and running.
9. Test the connection to the DataServer.
10. Create a Logical Schema object.
11. Create at least one Model for the LogicalSchema.
12. Import RKM Hive as a global knowledge module or into a project.

13. Create a new model for Hive Technology pointing to the logical schema.
14. Perform a custom reverse-engineering operation 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 flex field 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 Oracle Data Integrator Application Adapter for Hadoop.

Note: Many file names contain the version number. When you see a star (*) in a file name, check your installation and enter the full file name.

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. Ensure that the Hadoop configuration directory is in the ODI class path.

The Hadoop configuration directory contains files such as `core-default.xml`, `core-site.xml`, and `hdfs-site.xml`.

4. Add paths to `ODI_ADDITIONAL_CLASSPATH`, so that the ODI agent can access the JAR files. If you are not using Oracle SQL Connector for HDFS, then omit `$OSCH_HOME` from the setting.

Note: In these commands, `$HADOOP_CONF` points to the directory containing the Hadoop configuration files. This directory is often the same as `$HADOOP_HOME`.

- For CDH4, use a command like the following:

```
ODI_ADDITIONAL_CLASSPATH=$HIVE_HOME/lib/'*':$HADOOP_HOME/client/'*':$OSCH_HOME/jlib/'*':$HADOOP_CONF
```

- For CDH3, use a command like the following, replacing `hadoop-*-core*.jar` and `hadoop-*-tools*.jar` with the full path names:

```
ODI_ADDITIONAL_CLASSPATH=$HIVE_HOME/lib/'*':$HADOOP_HOME/hadoop-*-core*.jar:$HADOOP_HOME/hadoop-*-tools*.jar:$OSCH_HOME/jlib/'*':$HADOOP_CONF
```

5. Set environment variable `ODI_HIVE_SESSION_JARS` to include Hive Regex SerDe:

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

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

6. Set environment variable HADOOP_CLASSPATH:

```
HADOOP_CLASSPATH=$HIVE_HOME/lib/hive-metastore-*.jar:$HIVE_
HOME/lib/libthrift.jar:$HIVE_HOME/lib/libfb*.jar:$HIVE_
HOME/lib/hive-common-*.jar:$HIVE_HOME/lib/hive-exec-*.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. See ["Installing Oracle Loader for Hadoop"](#) on page 1-12.
2. Set environment variable OLH_HOME.
3. Optionally, set environment variable ODI_OLH_JARS. You must list any JAR files required for custom input formats, Hive, Hive SerDes, and so forth, used by Oracle Loader for Hadoop. Do not include the Oracle Loader for Hadoop JAR files.

For example, for extracting data from Hive, you need the extra jars listed in ["Specifying Hive Input Format JAR Files"](#) on page 3-22. Enter valid file names for your installation.

```
$HIVE_HOME/lib/hive-metastore-*.jar,
$HIVE_HOME/lib/libthrift.jar,
$HIVE_HOME/lib/libfb*.jar
```

4. Add paths to HADOOP_CLASSPATH:

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

5. Set environment variable ODI_HIVE_SESSION_JARS to include Hive Regex SerDe:

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

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

6. To use Oracle SQL Connector for HDFS (OLH_OUTPUT_MODE=DP_OSCH or OSCH), you must first install it. See ["Oracle SQL Connector for Hadoop Distributed File System Setup"](#) on page 1-4.

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 configuration steps in the previous section with the following change: Copy JAR files into the Oracle Data Integrator userlib directory instead of the drivers directory.

Setting Up an Integration Project

Setting up a project follows the standard procedures. See the *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 an Oracle Data Integrator Model from a Reverse-Engineered Hive Model

This section contains the following topics:

- [Creating a Model](#)
- [Reverse Engineering Hive Tables](#)

Creating a Model

To create a model that is based on the technology hosting Hive and on the logical schema created when you configured the Hive connection, follow the standard procedure described in the *Oracle Fusion Middleware Developer's Guide for Oracle Data Integrator*.

Reverse Engineering Hive Tables

RKM Hive is used to reverse engineer Hive tables and views. To perform a customized reverse-engineering of Hive tables with RKM Hive, 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.

RKM Hive

RKM Hive reverses these metadata elements:

- Hive tables and views as Oracle Data Integrator data stores.
Specify the reverse mask in the Mask field, and 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 percent sign (%).
- Hive columns as Oracle Data Integrator columns with their data types.
- Information about buckets, partitioning, clusters, and sort columns are set in the respective flex fields in the Oracle Data Integrator data store or column metadata.

[Table 4–3](#) describes the options for RKM Hive.

Table 4–3 RKM Hive 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 the default file name is reverse.log.

Table 4–4 describes the created flex fields.

Table 4–4 Flex Fields for Reverse-Engineered Hive Tables and Views

Object	Flex Field Name	Flex Field Code	Flex Field 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: <ul style="list-style-type: none"> ■ 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".

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 Data into an Oracle Database from Hive and HDFS](#)

Loading Data from Files into Hive

To load data from the local file system 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 about 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 integration knowledge module loads data from flat files into Hive, replacing or appending 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 4–5](#) describes the options for IKM File to Hive (Load Data). See the knowledge module for additional details.

Table 4–5 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 <i>false</i> .
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 knowledge modules.

IKM Hive Control Append

This knowledge module 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. IKM Hive Control Append supports inline view interfaces that use either this knowledge module or IKM Hive Transform.

[Table 4–6](#) lists the options. See the knowledge module for additional details.

Table 4–6 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 knowledge module checks data integrity for Hive tables. It verifies the validity of the constraints of a Hive data store and diverts the invalid records to 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 4–7](#) lists the options for this check knowledge module. See the knowledge module for additional details.

Table 4–7 *CKM Hive Options*

Option	Description
DROP_ERROR_TABLE	Drop error table before execution?

IKM Hive Transform

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

The transformation script must read 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 4–8](#) lists the options for this integration knowledge module. See the knowledge module for additional details.

Table 4–8 *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 knowledge module 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 knowledge module 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 knowledge module to distribute the data after the transformation script is applied
POST_TRANSFORM_SORT	Provides an optional, comma-separated list of target column names, which enables the knowledge module to sort the data after the transformation script is applied

Loading Data into an Oracle Database 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 knowledge module generates an appropriate Oracle Database target instance. Hive and Hadoop versions must follow the Oracle Loader for Hadoop requirements.

See Also:

- ["Oracle Loader for Hadoop Setup"](#) on page 1-12 for required versions of Hadoop and Hive
- ["Setting Up the Oracle Data Integrator Agent to Execute Hadoop Jobs"](#) on page 4-5 for required environment variable settings

Table 4–9 lists the options for this integration knowledge module. See the knowledge module for additional details.

Table 4–9 IKM File - Hive to Oracle (OLH)

Option	Description
OLH_OUTPUT_MODE	Specify JDBC, OCI, or Data Pump for data transfer.
CREATE_TARG_TABLE	Create target table?
REJECT_LIMIT	Maximum number of errors for Oracle Loader for Hadoop and EXTTab.
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 you are using an Oracle database staging table.
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?

Part III

Oracle XQuery for Hadoop

This part contains the following chapters:

- [Chapter 5, "Using Oracle XQuery for Hadoop"](#)
- [Chapter 6, "Oracle XQuery for Hadoop Reference"](#)
- [Chapter 7, "Oracle XML Extensions for Hive"](#)

Using Oracle XQuery for Hadoop

This chapter explains how to use Oracle XQuery for Hadoop to extract and transform large volumes of semistructured data. It contains the following sections:

- [What Is Oracle XQuery for Hadoop?](#)
- [Getting Started With Oracle XQuery for Hadoop](#)
- [About the Adapters](#)
- [Creating an XQuery Transformation](#)
- [Running a Query](#)
- [Oracle XQuery for Hadoop Configuration Properties](#)
- [Third-Party Licenses for Bundled Software](#)

What Is Oracle XQuery for Hadoop?

Oracle XQuery for Hadoop is a transformation engine for semi-structured big data. Oracle XQuery for Hadoop runs transformations expressed in the XQuery language by translating them into a series of MapReduce jobs, which are executed in parallel on the Apache Hadoop cluster. You can focus on data movement and transformation logic, instead of the complexities of Java and MapReduce, without sacrificing scalability or performance.

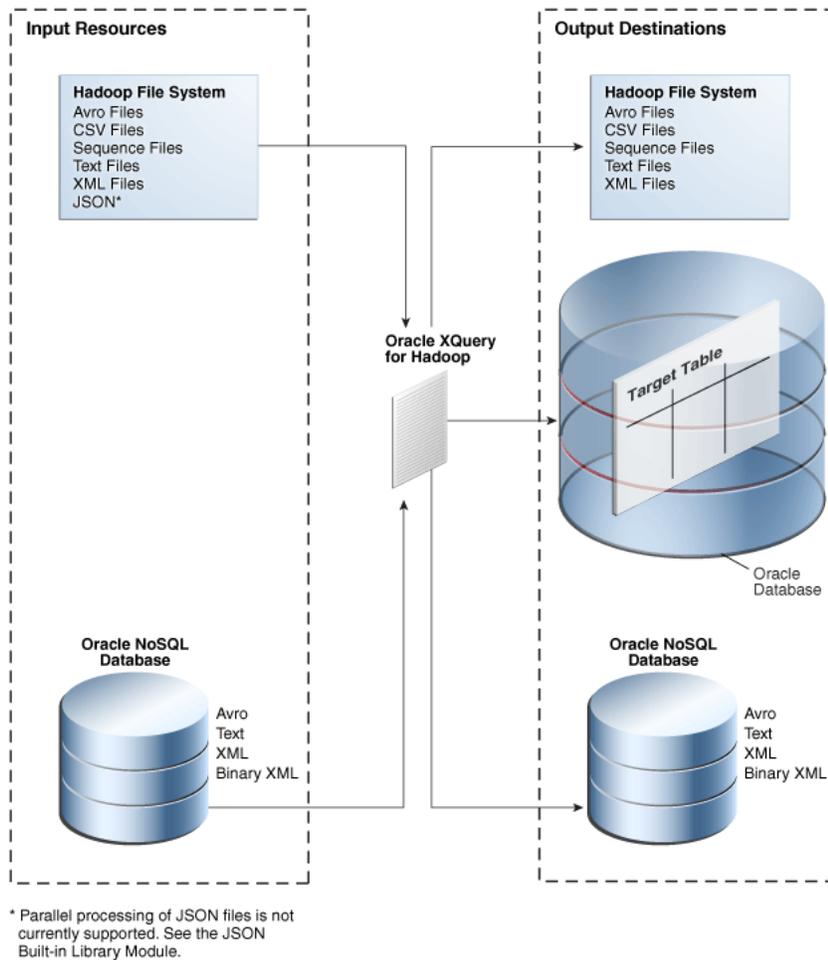
The input data can be located in a file system accessible through the Hadoop File System API, including the Hadoop Distributed File System (HDFS), or stored in Oracle NoSQL Database. Oracle XQuery for Hadoop can write the transformation results to Hadoop files, Oracle NoSQL Database, or Oracle Database.

Oracle XQuery for Hadoop also provides extensions to Apache Hive to support massive XML files. These extensions are available only on Oracle Big Data Appliance.

Oracle XQuery for Hadoop is based on mature industry standards including XPath, XQuery, and XQuery Update Facility. It is fully integrated with other Oracle products, and so it:

- Loads data efficiently into Oracle Database using Oracle Loader for Hadoop.
- Provides read and write support to Oracle NoSQL Database.

[Figure 5–1](#) provides an overview of the data flow using Oracle XQuery for Hadoop.

Figure 5–1 Oracle XQuery for Hadoop Data Flow

Getting Started With Oracle XQuery for Hadoop

Oracle XQuery for Hadoop is designed for use by XQuery developers. If you are already familiar with XQuery, then you are ready to begin. However, if you are new to XQuery, then you must first acquire the basics of the language. This guide does not attempt to cover this information.

See Also:

- "XQuery Tutorial" by W3Schools at <http://www.w3schools.com/xquery/>
- "XQuery 3.0: An XML Query Language" at <http://www.w3.org/TR/xquery-30>

Basic Steps

Take the following basic steps when using Oracle XQuery for Hadoop:

1. The first time you use Oracle XQuery for Hadoop, ensure that the software is installed and configured.

See "Oracle XQuery for Hadoop Setup" on page 1-15.

2. Log in to either a node in the Hadoop cluster or a system set up as a Hadoop client for the cluster.
3. Create an XQuery transformation that uses the Oracle XQuery for Hadoop functions. It can use various adapters for input and output.
See ["About the Adapters"](#) on page 5-3 and ["Creating an XQuery Transformation"](#) on page 5-6.
4. Execute the XQuery transformation.
See ["Running a Query"](#) on page 5-13.

Example: Hello World!

Follow these steps to create and run a simple query using Oracle XQuery for Hadoop:

1. Create a text file named `hello.txt` in the current directory that contains the line `Hello`.

```
$ echo "Hello" > hello.txt
```

2. Copy the file to HDFS:

```
$ hdfs dfs -copyFromLocal hello.txt
```

3. Create a query file named `hello.xq` in the current directory with the following content:

```
import module "oxh:text";  
for $line in text:collection("hello.txt")  
return text:put($line || " World!")
```

4. Run the query:

```
$ hadoop jar $OXH_HOME/lib/oxh.jar hello.xq -output ./myout -print
```

5. Check the output file:

```
$ hdfs dfs -cat ./myout/part-m-00000  
Hello World!
```

About the Adapters

This section contains the following topics:

- [About the Oracle XQuery for Hadoop Functions](#)
- [About the Avro File Adapter](#)
- [About the Oracle Database Adapter](#)
- [About the Oracle NoSQL Database Adapter](#)
- [About the Sequence File Adapter](#)
- [About the Text File Adapter](#)
- [About the XML File Adapter](#)
- [About Other Modules for Use With Oracle XQuery for Hadoop](#)

About the Oracle XQuery for Hadoop Functions

Oracle XQuery for Hadoop reads from and writes to big data sets using collection and put functions:

- A **collection function** reads data from Hadoop files or Oracle NoSQL Database as a collection of items. A Hadoop file is one that is accessible through the Hadoop File System API. On Oracle Big Data Appliance and most Hadoop clusters, this file system is Hadoop Distributed File System (HDFS).
- A **put function** adds a single item to a data set stored in Oracle Database, Oracle NoSQL Database, or a Hadoop file.

The following is a simple example of an Oracle XQuery for Hadoop query that reads items from one source and writes to another:

```
for $x in collection(...)
return put($x)
```

Oracle XQuery for Hadoop comes with a set of adapters that you can use to define put and collection functions for specific formats and sources. Each adapter has two components:

- A set of built-in put and collection functions that are predefined for your convenience.
- A set of XQuery function annotations that you can use to define custom put and collection functions.

About the Avro File Adapter

The Avro file adapter provides access to Avro container files stored in HDFS. It includes collection and put functions for reading from and writing to Avro container files.

See Also: ["Avro File Adapter"](#) on page 6-2

About the Oracle Database Adapter

The Oracle Database adapter loads data into Oracle Database. This adapter supports a custom put function for direct output to a table in an Oracle database using JDBC or OCI. If a live connection to the database is not available, the adapter also supports output to Data Pump or delimited text files in HDFS; the files can be loaded into the Oracle database with a different utility, such as SQL*Loader, or using external tables. This adapter does not move data out of the database, and therefore does not have collection or get functions.

See Also:

- ["Software Requirements"](#) on page 1-12 for the supported versions of Oracle Database
- ["Oracle Database Adapter"](#) on page 6-19

About the Oracle NoSQL Database Adapter

The Oracle NoSQL Database adapter provides access to data stored in Oracle NoSQL Database. The data can be read from or written as Avro, XML, binary XML, or text. This adapter includes collection, get, and put functions.

See Also: ["Oracle NoSQL Database Adapter"](#) on page 6-29

About the Sequence File Adapter

The sequence file adapter provides access to Hadoop sequence files. A sequence file is a Hadoop format composed of key-value pairs.

This adapter includes collection and put functions for reading from and writing to HDFS sequence files that contain text, XML, or binary XML.

See Also: ["Sequence File Adapter"](#) on page 6-47

About the Text File Adapter

The text file adapter provides access to text files, such as CSV files. It contains collection and put functions for reading from and writing to text files.

The JSON library module extends the support for JSON objects stored in text files.

See Also:

- ["Text File Adapter"](#) on page 6-58
- ["JSON Module"](#) on page 6-75

About the XML File Adapter

The XML file adapter provides access to XML files stored in HDFS. It contains collection functions for reading large XML files. You must use another adapter to write the output.

See Also: ["XML File Adapter"](#) on page 6-68

About Other Modules for Use With Oracle XQuery for Hadoop

You can use functions from these additional modules in your queries:

Standard XQuery Functions

The standard XQuery math functions are available.

See "Functions and Operators on Numerics" in *W3C XPath and XQuery Functions and Operators 3.0* at

<http://www.w3.org/TR/xpath-functions-30/#numeric-functions>

Hadoop Functions

The Hadoop module is a collection of functions that are specific to Hadoop.

See ["Hadoop Module"](#) on page 6-86.

JSON Functions

The JSON module is a collection of helper functions for parsing JSON data.

See ["JSON Module"](#) on page 6-75.

Duration, Date, and Time Functions

These functions parse duration, date, and time values.

See ["Duration, Date, and Time Functions"](#) on page 6-80.

String-Processing Functions

These functions add and remove white space that surrounds data values.

See ["String Functions"](#) on page 6-84.

Creating an XQuery Transformation

This chapter describes how to create XQuery transformations using Oracle XQuery for Hadoop. It contains the following topics:

- [XQuery Transformation Requirements](#)
- [About XQuery Language Support](#)
- [Accessing Data in the Hadoop Distributed Cache](#)
- [Calling Custom Java Functions from XQuery](#)
- [Accessing User-Defined XQuery Library Modules and XML Schemas](#)
- [XQuery Transformation Examples](#)

XQuery Transformation Requirements

You create a transformation for Oracle XQuery for Hadoop the same way as any other XQuery transformation, except that you must comply with these additional requirements:

- The main XQuery expression (the query body) must be in one of the following forms:

FLWOR₁

or

(FLWOR₁, FLWOR₂, . . . , FLWOR_N)

In this syntax FLWOR is a top-level "For, Let, Where, Order by, Return" expression.

See Also: "FLWOR Expressions" in *W3C XQuery 3.0: An XML Query Language* at

<http://www.w3.org/TR/xquery-30/#id-flwor-expressions>

- Each top-level FLWOR expression must have a `for` clause that iterates over an Oracle XQuery for Hadoop collection function. This `for` clause cannot have a positional variable.
See [Chapter 6](#) for the collection functions.
- Each top-level FLWOR expression must return one or more results from calling the Oracle XQuery for Hadoop `put` function. See [Chapter 6](#) for the `put` functions.

The query body must be an updating expression. Because all `put` functions are classified as updating functions, all Oracle XQuery for Hadoop queries are updating queries.

In Oracle XQuery for Hadoop, a `%*:put` annotation indicates that the function is updating. The `%updating` annotation or `updating` keyword is not required with it.

See Also: For a description of updating expressions, "Extensions to XQuery 1.0" in *W3C XQuery Update Facility 1.0* at

<http://www.w3.org/TR/xquery-update-10/#dt-updating-expression>

- Each top-level FLOWR expression can have optional `let`, `where`, and `group by` clauses. Other types of clauses are invalid, such as `order by`, `count`, and `window` clauses.

About XQuery Language Support

Oracle XQuery for Hadoop supports the XQuery 1.0 specification:

- For the language, see <http://www.w3.org/TR/xquery/>
- For the functions, see <http://www.w3.org/TR/xpath-functions/>

In addition, Oracle XQuery for Hadoop supports the following XQuery 3.0 features. The links are to the relevant sections of W3C *XQuery 3.0: An XML Query Language*.

- `group by` clause
See <http://www.w3.org/TR/xquery-30/#id-group-by>.
- `for` clause with the `allowing` empty modifier
See <http://www.w3.org/TR/xquery-30/#id-xquery-for-clause>.
- Annotations
See <http://www.w3.org/TR/xquery-30/#id-annotations>.
- String concatenation expressions
See <http://www.w3.org/TR/xquery-30/#id-string-concat-expr>.
- Standard functions:
 - `fn:analyze-string`
 - `fn:unparsed-text`
 - `fn:unparsed-text-lines`
 - `fn:unparsed-text-available`
 - `fn:serialize`
 - `fn:parse-xml`
 See <http://www.w3.org/TR/xpath-functions-30/>.
- Trigonometric and exponential functions
See <http://www.w3.org/TR/xpath-functions-30/#trigonometry>.

Accessing Data in the Hadoop Distributed Cache

You can use the Hadoop distributed cache facility to access auxiliary job data. This mechanism can be useful in a join query when one side is a relatively small file.

The query might execute faster if the smaller file is accessed from the distributed cache.

To place a file into the distributed cache, use the `-files` Hadoop command line option when calling Oracle XQuery for Hadoop.

For a query to read a file from the distributed cache, it must call the `fn:doc` function for XML, and either `fn:unparsed-text` or `fn:unparsed-text-lines` for text files.

See [Example 7](#).

Calling Custom Java Functions from XQuery

Oracle XQuery for Hadoop is extensible with custom external functions implemented in the Java language.

The Java implementation must be a static method with the parameter and return types as defined by the *XQuery API for Java (XQJ)* specification.

The custom Java function binding is defined in Oracle XQuery for Hadoop by annotating an external function definition with the `%ora-java:binding` annotation. This annotation has the following syntax:

```
%ora-java:binding("java.class.name[#method]")
```

java.class.name

The fully qualified name of a Java class that contains the implementation method.

method

A Java method name. It defaults to the XQuery function name. Optional.

See [Example 8](#) for an example of `%ora-java:binding`.

All JAR files that contain custom Java functions must be listed in the `-libjars` command line option. For example:

```
hadoop jar $OXH_HOME/lib/oxh.jar -libjars myfunctions.jar query.xq
```

See Also: "XQuery API for Java (XQJ)" at

<http://www.jcp.org/en/jsr/detail?id=225>

Accessing User-Defined XQuery Library Modules and XML Schemas

Oracle XQuery for Hadoop supports user-defined XQuery library modules and XML schemas when you comply with these criteria:

- Locate the library module or XML schema file in the same directory where the main query resides on the client calling Oracle XQuery for Hadoop.
- Import the library module or XML schema from the main query using the location URI parameter of the import module/schema statement.
- Specify the library module or XML schema file in the `-files` command line option when calling Oracle XQuery for Hadoop.

See [Example 9](#).

See Also: "Location URIs" in the W3C *XQuery 3.0: An XML Query Language* at

<http://www.w3.org/TR/xquery-30/#id-module-handling-location-uris>

XQuery Transformation Examples

For these examples, the following text files are in HDFS. The files contain a log of visits to different web pages. Each line represents a visit to a web page and contains the time, user name, page visited, and the status code.

```
mydata/visits1.log
```

```
2013-10-28T06:00:00, john, index.html, 200
2013-10-28T08:30:02, kelly, index.html, 200
2013-10-28T08:32:50, kelly, about.html, 200
2013-10-30T10:00:10, mike, index.html, 401
```

```
mydata/visits2.log
```

```
2013-10-30T10:00:01, john, index.html, 200
2013-10-30T10:05:20, john, about.html, 200
```

```
2013-11-01T08:00:08, laura, index.html, 200
2013-11-04T06:12:51, kelly, index.html, 200
2013-11-04T06:12:40, kelly, contact.html, 200
```

Example 1 Basic Filtering

This query filters out pages visited by user kelly and writes those files into a text file

```
import module "oxh:text";

for $line in text:collection("mydata/visits*.log")
let $split := fn:tokenize($line, "\s*\s*")
where $split[2] eq "kelly"
return text:put($line)
```

The query creates text files in the output directory that contain the following lines:

```
2013-11-04T06:12:51, kelly, index.html, 200
2013-11-04T06:12:40, kelly, contact.html, 200
2013-10-28T08:30:02, kelly, index.html, 200
2013-10-28T08:32:50, kelly, about.html, 200
```

Example 2 Group By and Aggregation

The next query computes the number of page visits per day:

```
import module "oxh:text";

for $line in text:collection("mydata/visits*.log")
let $split := fn:tokenize($line, "\s*\s*")
let $time := xs:dateTime($split[1])
let $day := xs:date($time)
group by $day
return text:put($day || " => " || fn:count($line))
```

The query creates text files that contain the following lines:

```
2013-10-28 => 3
2013-10-30 => 3
2013-11-01 => 1
2013-11-04 => 2
```

Example 3 Inner Joins

This example queries the following text file in HDFS, in addition to the other files. The file contains user profile information such as user ID, full name, and age, separated by a colon (:).

```
mydata/users.txt

john:John Doe:45
kelly:Kelly Johnson:32
laura:Laura Smith:
phil:Phil Johnson:27
```

The following query performs a join between users.txt and the log files. It computes how many times users older than 30 visited each page.

```
import module "oxh:text";

for $userLine in text:collection("mydata/users.txt")
let $userSplit := fn:tokenize($userLine, "\s*:\s*")
let $userId := $userSplit[1]
let $userAge := xs:integer($userSplit[3][. castable as xs:integer])
```

```

for $visitLine in text:collection("mydata/visits*.log")
let $visitSplit := fn:tokenize($visitLine, "\s*\s*")
let $visitUserId := $visitSplit[2]
where $userId eq $visitUserId and $userAge gt 30
group by $page := $visitSplit[3]
return text:put($page || " " || fn:count($userLine))

```

The query creates text files that contain the following lines:

```

about.html 2
contact.html 1
index.html 4

```

The next query computes the number of visits for each user who visited any page; it omits users who never visited any page.

```

import module "oxh:text";

for $userLine in text:collection("mydata/users.txt")
let $userSplit := fn:tokenize($userLine, "\s*:\s*")
let $userId := $userSplit[1]

for $visitLine in text:collection("mydata/visits*.log")
  [$userId eq fn:tokenize(., "\s*\s*")[2]]

group by $userId
return text:put($userId || " " || fn:count($visitLine))

```

The query creates text files that contain the following lines:

```

john 3
kelly 4
laura 1

```

Note: When the results of two collection functions are joined, only equijoins are supported. If one or both sources are not from a collection function, then any join condition is allowed.

Example 4 Left Outer Joins

This example is similar to the second query in [Example 3](#), but also counts users who did not visit any page.

```

import module "oxh:text";

for $userLine in text:collection("mydata/users.txt")
let $userSplit := fn:tokenize($userLine, "\s*:\s*")
let $userId := $userSplit[1]

for $visitLine allowing empty in text:collection("mydata/visits*.log")
  [$userId eq fn:tokenize(., "\s*\s*")[2]]

group by $userId
return text:put($userId || " " || fn:count($visitLine))

```

The query creates text files that contain the following lines:

```

john 3
kelly 4
laura 1

```

phil 0

Example 5 Semijoins

The next query finds users who have ever visited a page.

```
import module "oxh:text";

for $userLine in text:collection("mydata/users.txt")
let $userId := fn:tokenize($userLine, "\s*:\s*")[1]

where some $visitLine in text:collection("mydata/visits*.log")
satisfies $userId eq fn:tokenize($visitLine, "\s*,\s*")[2]

return text:put($userId)
```

The query creates text files that contain the following lines:

```
john
kelly
laura
```

Example 6 Multiple Outputs

The next query finds web page visits with a 401 code and writes them to trace* files using the XQuery text:trace() function. It writes the remaining visit records into the default output files.

```
import module "oxh:text";

for $visitLine in text:collection("mydata/visits*.log")
let $visitCode := xs:integer(fn:tokenize($visitLine, "\s*,\s*")[4])
return if ($visitCode eq 401) then text:trace($visitLine) else
text:put($visitLine)
```

The query generates a trace* text file that contains the following line:

```
2013-10-30T10:00:10, mike, index.html, 401
```

The query also generates default output files that contain the following lines:

```
2013-10-30T10:00:01, john, index.html, 200
2013-10-30T10:05:20, john, about.html, 200
2013-11-01T08:00:08, laura, index.html, 200
2013-11-04T06:12:51, kelly, index.html, 200
2013-11-04T06:12:40, kelly, contact.html, 200
2013-10-28T06:00:00, john, index.html, 200
2013-10-28T08:30:02, kelly, index.html, 200
2013-10-28T08:32:50, kelly, about.html, 200
```

Example 7 Accessing Auxiliary Input Data

The next query is an alternative version of the second query in [Example 3](#), but it uses the fn:unparsed-text-lines function to access a file in the Hadoop distributed cache.

```
import module "oxh:text";

for $visitLine in text:collection("mydata/visits*.log")
let $visitUserId := fn:tokenize($visitLine, "\s*,\s*")[2]

for $userLine in fn:unparsed-text-lines("users.txt")
let $userSplit := fn:tokenize($userLine, "\s*:\s*")
let $userId := $userSplit[1]
```

```

where $userId eq $visitUserId

group by $userId
return text:put($userId || " " || fn:count($visitLine))

```

The hadoop command to run the query must use the Hadoop `-files` option. See ["Accessing Data in the Hadoop Distributed Cache"](#) on page 5-7.

```
hadoop jar $OXH_HOME/lib/oxh.jar -files users.txt query.xq
```

The query creates text files that contain the following lines:

```

john 3
kelly 4
laura 1

```

Example 8 Calling a Custom Java Function from XQuery

The next query formats input data using the `java.lang.String#format` method.

```

import module "oxh:text";

declare %ora-java:binding("java.lang.String#format")
  function local:string-format($pattern as xs:string, $data as xs:anyAtomicType*)
  as xs:string external;

for $line in text:collection("mydata/users*.txt")
let $split := fn:tokenize($line, "\s*\s*")
return text:put(local:string-format("%s,%s,%s", $split))

```

The query creates text files that contain the following lines:

```

john,John Doe,45
kelly,Kelly Johnson,32
laura,Laura Smith,
phil,Phil Johnson,27

```

See Also: *Java Platform Standard Edition 7 API Specification for Class String* at

[http://docs.oracle.com/javase/7/docs/api/java/lang/String.html#format\(java.lang.String, java.lang.Object...\)](http://docs.oracle.com/javase/7/docs/api/java/lang/String.html#format(java.lang.String,java.lang.Object...))

Example 9 Using User-defined XQuery Library Modules and XML Schemas

This example uses a library module named `mytools.xq`.

```

module namespace mytools = "urn:mytools";

declare %ora-java:binding("java.lang.String#format")
  function mytools:string-format($pattern as xs:string, $data as
xs:anyAtomicType*) as xs:string external;

```

The next query is equivalent to the previous one, but it calls a `string-format` function from the `mytools.xq` library module:

```

import module namespace mytools = "urn:mytools" at "mytools.xq";
import module "oxh:text";

for $line in text:collection("mydata/users*.txt")
let $split := fn:tokenize($line, "\s*\s*")
return text:put(mytools:string-format("%s,%s,%s", $split))

```

The query creates text files that contain the following lines:

```
john,John Doe,45
kelly,Kelly Johnson,32
laura,Laura Smith,
phil,Phil Johnson,27
```

Running a Query

To run a query, call the OXH utility using the `hadoop jar` command.

The following is the basic syntax:

```
hadoop jar $OXH_HOME/lib/oxh.jar [generic options] query.xq -output directory
[-clean] [-ls] [-print] [-skiperrors] [-version]
```

Oracle XQuery for Hadoop Options

query.xq

Identifies the XQuery file. See ["Creating an XQuery Transformation"](#) on page 5-6.

-clean

Deletes all files from the output directory before running the query.

-ls

Lists the contents of the output directory after the query executes.

-output *directory*

Specifies the output directory of the query. The put functions of the file adapters create files in this directory. Written values are spread across one or more files. The number of files created depends on how the query is distributed among tasks. By default, each output file has a name that starts with `part`, such as `part-m-00000`.

See ["About the Oracle XQuery for Hadoop Functions"](#) on page 5-4 for a description of put functions.

-print

Prints the contents of all files in the output directory to the standard output (your screen). When printing Avro files, each record prints as JSON text.

-skiperrors

Turns on error recovery, so that an error does not halt processing.

All errors that occur during query processing are counted, and the total is logged at the end of the query. The error messages of the first 20 errors per task are also logged.

See [oracle.hadoop.xquery.skiperrors.counters](#), [oracle.hadoop.xquery.skiperrors.max](#), and [oracle.hadoop.xquery.skiperrors.log.max](#).

-version

Displays the Oracle XQuery for Hadoop version and exits without running a query.

Generic Options

You can include any generic `hadoop` command-line option. OXH implements the `org.apache.hadoop.util.Tool` interface and follows the standard Hadoop methods for building MapReduce applications.

The following generic options are commonly used with Oracle XQuery for Hadoop:

-conf *job_config.xml*

Identifies the job configuration file. See "Oracle XQuery for Hadoop Configuration Properties" on page 5-14.

When you are working with the Oracle Database or Oracle NoSQL Database adapters, you can set various job properties in this file. See "%oracle-property Annotations and Corresponding Oracle Loader for Hadoop Configuration Properties" on page 6-26 and "Oracle NoSQL Database Adapter Configuration Properties" on page 6-44.

-D *property=value*

Identifies a configuration property. See "Oracle XQuery for Hadoop Configuration Properties" on page 5-14.

-files

Specifies a comma-delimited list of files that are added to the distributed cache. See "Accessing Data in the Hadoop Distributed Cache" on page 5-7.

See Also: For full descriptions of the generic options:

http://hadoop.apache.org/docs/r1.1.2/commands_manual.html#Generic+Options

About Running Queries Locally

When developing queries, you can run them locally before submitting them to the cluster. A local run enables you to see how the query behaves on small data sets and diagnose potential problems quickly.

In local mode, relative URIs resolve against the local file system instead of HDFS, and the query runs in a single process.

To run a query in local mode:

1. Set the Hadoop `-jt` and `-fs` generic arguments to `local`. This example runs the query described in "Example: Hello World!" on page 5-3 in local mode:

```
$ hadoop jar $OXH_HOME/lib/oxh.jar -jt local -fs local ./hello.xq -output ./myoutput -print
```

2. Check the result file in the local output directory of the query, as shown in this example:

```
$ cat ./myoutput/part-m-00000
Hello World!
```

Oracle XQuery for Hadoop Configuration Properties

Oracle XQuery for Hadoop uses the generic methods of specifying configuration properties in the `hadoop` command. You can use the `-conf` option to identify configuration files, and the `-D` option to specify individual properties. See "Running a Query" on page 5-13.

See Also: Hadoop documentation for job configuration files at

<http://wiki.apache.org/hadoop/JobConfFile>

oracle.hadoop.xquery.output

Type: String

Default Value: Not defined

Description: Sets the output directory for the query. This property is equivalent to the `-output` command line option.

oracle.hadoop.xquery.scratch

Type: String

Default Value: `/tmp/user_name/oxh`. The `user_name` is the name of the user running Oracle XQuery for Hadoop.

Description: Sets the HDFS temp directory for Oracle XQuery for Hadoop to store temporary files.

oracle.hadoop.xquery.timezone

Type: String

Default Value: Client system time zone

Description: The XQuery implicit time zone, which is used in a comparison or arithmetic operation when a date, time or date`Time` value does not have a time zone. The value must be in the format described by the Java `TimeZone` class. See the Java 7 API Specification at

<http://docs.oracle.com/javase/7/docs/api/java/util/TimeZone.html>

oracle.hadoop.xquery.skiperrors

Type: Boolean

Default Value: `false`

Description: Set to `true` to turn on error recovery, or set to `false` to stop processing when an error occurs. This property is equivalent to the `-skiperrors` command line option.

oracle.hadoop.xquery.skiperrors.counters

Type: Boolean

Default Value: `true`

Description: Set to `true` to group errors by error code, or set to `false` to report all errors in a single counter.

oracle.hadoop.xquery.skiperrors.max

Type: Integer

Default Value: Unlimited

Description: Sets the maximum number of errors that a single MapReduce task can recover from (skip).

oracle.hadoop.xquery.skiperrors.log.max

Type: Integer

Default Value: 20

Description: Sets the maximum number of errors that a single MapReduce task logs.

log4j.logger.oracle.hadoop.xquery

Type: String

Default Value: Not defined

Description: Configures the `log4j` logger for each task with the specified threshold level. Set the property to one of these values: `OFF`, `FATAL`, `ERROR`, `WARN`, `INFO`, `DEBUG`, `ALL`. If this property is not set, then Oracle XQuery for Hadoop does not configure `log4j`.

Third-Party Licenses for Bundled Software

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

- [ANTLR 3.2](#)
- [Apache Ant 1.7.1](#)
- [Apache Avro 1.7.3, 1.7.4](#)
- [Apache Xerces](#)
- [Apache XMLBeans 2.5](#)
- [Jackson 1.8.8](#)
- [Woodstox XML Parser 4.2](#)

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 in "[Apache Licensed Code](#)" on page 3-40.

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.

ANTLR 3.2

[The BSD License]

Copyright © 2010 Terence Parr

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of the author nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Apache Ant 1.7.1

Copyright 1999-2008 The Apache Software Foundation

This product includes software developed by The Apache Software Foundation (<http://www.apache.org>).

This product includes also software developed by:

- the W3C consortium (<http://www.w3c.org>)
- the SAX project (<http://www.saxproject.org>)

The <sync> task is based on code Copyright (c) 2002, Landmark Graphics Corp that has been kindly donated to the Apache Software Foundation.

Portions of this software were originally based on the following:

- software copyright (c) 1999, IBM Corporation, <http://www.ibm.com>.
- software copyright (c) 1999, Sun Microsystems, <http://www.sun.com>.
- voluntary contributions made by Paul Eng on behalf of the Apache Software Foundation that were originally developed at iClick, Inc., software copyright (c) 1999

W3C® SOFTWARE NOTICE AND LICENSE

<http://www.w3.org/Consortium/Legal/2002/copyright-software-20021231>

This work (and included software, documentation such as READMEs, or other related items) is being provided by the copyright holders under the following license. By obtaining, using and/or copying this work, you (the licensee) agree that you have read, understood, and will comply with the following terms and conditions.

Permission to copy, modify, and distribute this software and its documentation, with or without modification, for any purpose and without fee or royalty is hereby granted, provided that you include the following on ALL copies of the software and documentation or portions thereof, including modifications:

1. The full text of this NOTICE in a location viewable to users of the redistributed or derivative work.
2. Any pre-existing intellectual property disclaimers, notices, or terms and conditions. If none exist, the W3C Software Short Notice should be included (hypertext is preferred, text is permitted) within the body of any redistributed or derivative code.

3. Notice of any changes or modifications to the files, including the date changes were made. (We recommend you provide URIs to the location from which the code is derived.)

THIS SOFTWARE AND DOCUMENTATION IS PROVIDED "AS IS," AND COPYRIGHT HOLDERS MAKE NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF THE SOFTWARE OR DOCUMENTATION WILL NOT INFRINGE ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADEMARKS OR OTHER RIGHTS.

COPYRIGHT HOLDERS WILL NOT BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF ANY USE OF THE SOFTWARE OR DOCUMENTATION.

The name and trademarks of copyright holders may NOT be used in advertising or publicity pertaining to the software without specific, written prior permission. Title to copyright in this software and any associated documentation will at all times remain with copyright holders.

This formulation of W3C's notice and license became active on December 31 2002. This version removes the copyright ownership notice such that this license can be used with materials other than those owned by the W3C, reflects that ERCIM is now a host of the W3C, includes references to this specific dated version of the license, and removes the ambiguous grant of "use". Otherwise, this version is the same as the previous version and is written so as to preserve the Free Software Foundation's assessment of GPL compatibility and OSI's certification under the Open Source Definition. Please see our Copyright FAQ for common questions about using materials from our site, including specific terms and conditions for packages like libwww, Amaya, and Jigsaw. Other questions about this notice can be directed to site-policy@w3.org.

Joseph Reagle <site-policy@w3.org>

This license came from: <http://www.megginson.com/SAX/copying.html>

However please note future versions of SAX may be covered under <http://saxproject.org/?selected=pd>

SAX2 is Free!

I hereby abandon any property rights to SAX 2.0 (the Simple API for XML), and release all of the SAX 2.0 source code, compiled code, and documentation contained in this distribution into the Public Domain. SAX comes with NO WARRANTY or guarantee of fitness for any purpose.

David Megginson, david@megginson.com

2000-05-05

Apache Avro 1.7.3, 1.7.4

Copyright 2010 The Apache Software Foundation

This product includes software developed at The Apache Software Foundation (<http://www.apache.org/>).

C JSON parsing provided by Jansson and written by Petri Lehtinen. The original software is available from <http://www.digip.org/jansson/>.

Licensed under the Apache License, Version 2.0 (the "License"); you may not use Apache Avro 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.

License for the Jansson C JSON parser used in the C implementation:

Copyright (c) 2009 Petri Lehtinen <petri@digip.org>

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

License for the Json.NET used in the C# implementation:

Copyright (c) 2007 James Newton-King

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

License for msinttypes used in the C implementation:

Source from:

<http://code.google.com/p/msinttypes/downloads/detail?name=msinttypes-r26.zip>

Copyright (c) 2006-2008 Alexander Chemeris

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. The name of the author may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE AUTHOR "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

License for Diredt API for Microsoft Visual Studio used in the C implementation:

Source from:

<http://www.softagalleria.net/download/diredt/diredt-1.11.zip>

Copyright (C) 2006 Toni Ronkko

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL TONI RONKKO BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Apache Xerces

Xerces Copyright © 1999-2002 The Apache Software Foundation. All rights reserved. Licensed under the Apache 1.1 License Agreement.

The names "Xerces" and "Apache Software Foundation" must not be used to endorse or promote products derived from this software or be used in a product name without prior written permission. For written permission, please contact apache@apache.org.

This software consists of voluntary contributions made by many individuals on behalf of the Apache Software Foundation. For more information on the Apache Software Foundation, please see <http://www.apache.org>.

The Apache Software License, Version 1.1

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. The end-user documentation included with the redistribution, if any, must include the acknowledgements set forth above in connection with the software ("This product includes software developed by the) Alternately, this acknowledgement may appear in the software itself, if and wherever such third-party acknowledgements normally appear.
4. The names identified above with the specific software must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact apache@apache.org.
5. Products derived from this software may not be called "Apache" nor may "Apache" appear in their names without prior written permission of the Apache Group.

THIS SOFTWARE IS PROVIDED "AS IS" AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE APACHE SOFTWARE FOUNDATION OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Apache XMLBeans 2.5

This product includes software developed by The Apache Software Foundation (<http://www.apache.org/>).

Portions of this software were originally based on the following:

- software copyright (c) 2000-2003, BEA Systems, <<http://www.bea.com/>>.

Aside from contributions to the Apache XMLBeans project, this software also includes:

- one or more source files from the Apache Xerces-J and Apache Axis products, Copyright (c) 1999-2003 Apache Software Foundation
- W3C XML Schema documents Copyright 2001-2003 (c) World Wide Web Consortium (Massachusetts Institute of Technology, European Research Consortium for Informatics and Mathematics, Keio University)
- resolver.jar from Apache Xml Commons project, Copyright (c) 2001-2003 Apache Software Foundation
- Piccolo XML Parser for Java from <http://piccolo.sourceforge.net/>, Copyright 2002 Yuval Oren under the terms of the Apache Software License 2.0
- JSR-173 Streaming API for XML from <http://sourceforge.net/projects/xmlpullparser/>, Copyright 2005 BEA under the terms of the Apache Software License 2.0

Jackson 1.8.8

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.

Woodstox XML Parser 4.2

This copy of Woodstox XML processor is licensed under the Apache (Software) License, version 2.0 ("the License"). See the License for details about distribution rights, and the specific rights regarding derivate works.

You may obtain a copy of the License at:

<http://www.apache.org/licenses/>

A copy is also included with both the downloadable source code package and jar that contains class bytecodes, as file "ASL 2.0". In both cases, that file should be located next to this file: in source distribution the location should be "release-notes/asl"; and in jar "META-INF/"

This product currently only contains code developed by authors of specific components, as identified by the source code files.

Since product implements StAX API, it has dependencies to StAX API classes.

For additional credits (generally to people who reported problems) see CREDITS file.

Oracle XQuery for Hadoop Reference

This chapter describes the adapters available in Oracle XQuery for Hadoop:

- [Avro File Adapter](#)
- [Oracle Database Adapter](#)
- [Oracle NoSQL Database Adapter](#)
- [Sequence File Adapter](#)
- [Text File Adapter](#)
- [XML File Adapter](#)
- [Serialization Annotations](#)

This chapter also describes several other library modules:

- [JSON Module](#)
- [Hadoop Module](#)
- [Utility Module](#)

Avro File Adapter

The Avro file adapter provides functions to read and write Avro container files in HDFS. It is described in the following topics:

- [Built-in Functions for Reading Avro Files](#)
- [Custom Functions for Reading Avro Container Files](#)
- [Custom Functions for Writing Avro Files](#)
- [About Converting Values Between Avro and XML](#)

Built-in Functions for Reading Avro Files

To use the built-in functions in your query, you must import the Avro file module as follows:

```
import module "oxh:avro";
```

The Avro file module contains the following functions:

- [avro:collection-avroxml](#)
- [avro:get](#)

There are no built-in functions for writing Avro container files. To write Avro files, you must use a custom function that specifies the Avro writer schema.

avro:collection-avroxml

Accesses a collection of Avro files in HDFS. The files may be split up and processed in parallel by multiple tasks. The function returns an XML element for each object. See ["About Converting Values Between Avro and XML"](#) on page 6-10.

Signature

```
declare %avro:collection("avroxml") function
  avro:collection-avroxml($uris as xs:string*) as element()* external;
```

Parameters

\$uris

The Avro file URIs

Returns

One XML element for each Avro object.

avro:get

Retrieves an entry from an Avro map modeled as XML

If `$map` not present, then the behavior is identical to calling the two-argument function and using the context item for `$map`.

Signature

```
avro:get($key as xs:string?, $map as node()?) as element(oxh:entry)?
```

```
avro:get($key as xs:string?) as element(oxh:entry)?
```

Returns

The value of this XPath expression:

```
$map/oxh:entry[@key eq $key]
```

Example

These function calls are equivalent:

```
$var/avro:get("key")
```

```
avro:get("key", $var)  
$var/oxh:entry[@key eq "key"]
```

In this example, \$var is an Avro map modeled as XML. See ["Reading Maps"](#) on page 6-11.

Custom Functions for Reading Avro Container Files

You can use the following annotations to define functions that read collections of Avro container files in HDFS. These annotations provide additional functionality that is not available using the built-in functions.

Signature

Custom functions for reading Avro files must have the following signature:

```
declare %avro:collection("avroxml") [additional annotations]
    function local:myFunctionName($uris as xs:string*) as element()* external;
```

Annotations

%avro:collection("avroxml")

Declares the avroxml collection function. Required.

A collection function accesses Avro files in HDFS. The files might be split up and processed in parallel by multiple tasks. The function returns an XML element for each object. See ["About Converting Values Between Avro and XML"](#) on page 6-10.

%avro:schema("avro-schema")

Provides the Avro reader schema as the value of the annotation. Optional.

The objects in the file are mapped to the reader schema when specified. For example:

```
%avro:schema('
  {
    "type": "record",
    "name": "Person",
    "fields" : [
      {"name": "full_name", "type": "string"},
      {"name": "age", "type": ["int", "null"]}
    ]
  }
')
```

You cannot combine this annotation with %avro:schema-file or %avro:schema-kv.

See Also: "Schema Resolution" in the Apache Avro Specification at <http://avro.apache.org/docs/current/spec.html#Schema+Resolution>

%avro:schema-file("avro-schema-uri")

Like %avro:schema, but the annotation value is a file URI that contains the Avro reader schema. Relative URIs are resolved against the current working directory of the client's local file system. Optional.

For example, %avro:schema-file("schemas/person.avsc").

You cannot combine this annotation with %avro:schema or %avro:schema-kv.

%avro:schema-kv("schema-name")

Like %avro:schema, but the annotation value is a fully qualified record name. The record schema is retrieved from the Oracle NoSQL Database catalog. Optional.

For example, %avro:schema-kv("org.example.PersonRecord").

You must specify the connection parameters to Oracle NoSQL Database when you use this annotation. See ["Oracle NoSQL Database Adapter Configuration Properties"](#) on page 6-44.

You cannot combine this annotation with `%avro:schema` or `%avro:schema-file`.

`%avro:split-max("split-size")`

Specifies the maximum split size as either an integer or a string value. The split size controls how the input file is divided into tasks. Hadoop calculates the split size as `max($split-min, min($split-max, $block-size))`. Optional.

In a string value, you can append `K`, `k`, `M`, `m`, `G`, or `g` to the value to indicate kilobytes, megabytes, or gigabytes instead of bytes (the default unit). These qualifiers are not case sensitive. The following examples are equivalent:

```
%avro:split-max(1024)
%avro:split-max("1024")
%avro:split-max("1K")
```

`%avro:split-min("split-size")`

Specifies the minimum split size as either an integer or a string value. The split size controls how the input file is divided into tasks. Hadoop calculates the split size as `max($split-min, min($split-max, $block-size))`. Optional.

In a string value, you can append `K`, `k`, `M`, `m`, `G`, or `g` to the value to indicate kilobytes, megabytes, or gigabytes instead of bytes (the default unit). These qualifiers are not case sensitive. The following examples are equivalent:

```
%avro:split-min(1024)
%avro:split-min("1024")
%avro:split-min("1K")
```

Custom Functions for Writing Avro Files

You can use the following annotations to define functions that write Avro files.

Signature

Custom functions for writing Avro files must have the following signature:

```
declare %avro:put("avroxml") [additional annotations]
    local:myFunctionName($value as item()) external;
```

Annotations

%avro:put("avroxml")

Declares the avroxml put function. Required.

An Avro schema must be specified using one of the following annotations:

- %avro:schema
- %avro:schema-file
- %avro:schema-kv

The input XML value is converted to an instance of the schema. See ["Writing XML as Avro"](#) on page 6-14.

%avro:schema("avro-schema")

Specifies the schema of the files. For example:

```
%avro:schema('
  {
    "type": "record",
    "name": "Person",
    "fields" : [
      {"name": "full_name", "type": "string"},
      {"name": "age", "type": ["int", "null"]}
    ]
  }
')
```

You cannot combine this annotation with %avro:schema-file or %avro:schema-kv.

%avro:schema-file("avro-schema-uri")

Like %avro:schema, but the annotation value is a file URI that contains the Avro reader schema. Relative URIs are resolved against the current working directory of the client's local file system.

For example, %avro:schema-file("schemas/person.avsc").

You cannot combine this annotation with %avro:schema or %avro:schema-kv.

%avro:schema-kv("schema-name")

Like %avro:schema, but the annotation value is a fully qualified record name. The record schema is retrieved from the Oracle NoSQL Database catalog.

For example, %avro:schema-kv("org.example.PersonRecord").

You must specify the connection parameters to Oracle NoSQL Database when you use this annotation. See ["Oracle NoSQL Database Adapter Configuration Properties"](#) on page 6-44.

You cannot combine this annotation with `%avro:schema` or `%avro:schema-file`.

%avro:compress("method", [level]?)

Specifies the compression format used on the output.

The *codec* is one of the following string literal values:

- **deflate**: The *level* controls the trade-off between speed and compression. Valid values are 1 to 9, where 1 is the fastest and 9 is the most compressed.
- **snappy**: This algorithm is designed for high speed and moderate compression.

The default is no compression.

The *level* is an integer value. It is optional and only supported when *codec* is deflate.

For example:

```
%avro:compress("snappy")
%avro:compress("deflate")
%avro:compress("deflate", 3)
```

%avro:file("name")

Specifies the output file name prefix. The default prefix is `part`.

Examples of Avro File Adapter Functions

For this example, the following text file is in HDFS:

```
mydata/ages.txt
```

```
john,45
kelly,36
laura,
mike,27
```

Example 1 Converting a Text File to Avro

The following query converts the file into compressed Avro container files:

```
import module "oxh:text";

declare
  %avro:put("avroxml")
  %avro:compress("snappy")
  %avro:schema('
    {
      "type": "record",
      "name": "AgeRec",
      "fields" : [
        {"name": "user", "type": "string"},
        {"name": "age", "type": ["int", "null"]}
      ]
    }
  ')
function local:put($arg as item()) external;

for $line in text:collection("mydata/ages.txt")
let $split := fn:tokenize($line, ",")
return
  local:put(
    <rec>
      <user>{$split[1]}</user>
```

```

        {
          if ($split[2] castable as xs:int) then
            <age>{$split[2]}</age>
          else
            ()
        }
      </rec>
    )

```

The query generates an Avro file with the following records, represented here as JSON:

```

{"user": "john", "age": {"int": 45}}
{"user": "kelly", "age": {"int": 36}}
{"user": "laura", "age": null}
{"user": "mike", "age": {"int": 27}}

```

Example 2 Querying Records in Avro Container Files

The next query selects records in which the age is either null or greater than 30, from the myoutput directory. The query in [Example 1](#) generated the records.

```

import module "oxh:text";
import module "oxh:avro";

for $rec in avro:collection-avroxml("myoutput/part*.avro")
where $rec/age/nilled() or $rec/age gt 30
return
  text:put($rec/user)

```

This query creates files that contain the following lines:

```

john
kelly
laura

```

About Converting Values Between Avro and XML

This section describes how Oracle XQuery for Hadoop converts data between Avro and XML:

- [Reading Avro as XML](#)
- [Writing XML as Avro](#)

Reading Avro as XML

Both the Avro file adapter and the Oracle NoSQL Database adapter have an `avroxml` method, which you can use with the collection functions to read Avro records as XML. After the Avro is converted to XML, you can query and transform the data using XQuery.

The following topics describe how Oracle XQuery for Hadoop reads Avro:

- [Reading Records](#)
- [Reading Maps](#)
- [Reading Arrays](#)
- [Reading Unions](#)
- [Reading Primitives](#)

Reading Records

An Avro record is converted to an `<oxh:item>` element with one child element for each field in the record.

For example, consider the following Avro schema:

```
{
  "type": "record",
  "name": "Person",
  "fields" : [
    { "name": "full_name", "type": "string" },
    { "name": "age", "type": ["int", "null"] }
  ]
}
```

This is an instance of the record modeled as XML:

```
<oxh:item>
  <full_name>John Doe</full_name>
  <age>46</age>
</oxh:item>
```

Converting Avro records to XML allows them to be queried using XQuery. The next example queries an Avro container file named `person.avro`, which contains Person records. The query converts the records to a CSV text file in which each line contains the `full_name` and `age` values:

```
import module "oxh:avro";
import module "oxh:text";

for $x in avro:collection-avroxml("person.avro")
return
  text:put($x/full_name || ", " || $x/age)
```

Null values are converted to nilled elements. A **nilled** element has an `xsi:nil` attribute set to `true`; it is always empty. You can use the XQuery `fn:nilled` function to test if a record field is null. For example, the following query writes the name of Person records that have a null value for age:

```
import module "oxh:avro";
import module "oxh:text";

for $x in avro:collection-avroxml("person.avro")
where $x/age/nilled()
return
  text:put($x/full_name)
```

For nested records, the fields of the inner schema become child elements of the element that corresponds to the field in the outer schema. For example, this schema has a nested record:

```
{
  "type": "record",
  "name": "PersonAddress",
  "fields" : [
    { "name": "full_name", "type": "string" },
    { "name": "address", "type":
      { "type" : "record",
        "name" : "Address",
        "fields" : [
          { "name" : "street", "type" : "string" },
          { "name" : "city", "type" : "string" }
        ]
      }
    }
  ]
}
```

This is an instance of the record as XML:

```
<oxh:item>
  <full_name>John Doe</full_name>
  <address>
    <street>123 First St.</street>
    <city>New York</city>
  </address>
</oxh:item>
```

The following example queries an Avro container file named `people-address.avro` that contains `PersonAddress` records, and writes the names of the people that live in New York to a text file:

```
import module "oxh:avro";
import module "oxh:text";

for $person in avro:collection-avroxml("examples/person-address.avro")
where $person/address/city eq "New York"
return
  text:put($person/full_name)
```

Reading Maps

Avro map values are converted to an element that contains one child `<oxh:entry>` element for each entry in the map. For example, consider the following schema:

```
{
```

```

    "type": "record",
    "name": "PersonProperties",
    "fields" : [
      { "name": "full_name", "type": "string"},
      { "name": "properties", "type":
        { "type": "map", "values": "string"}
      }
    ]
  }
}

```

This is an instance of the schema as XML:

```

<oxh:item>
  <full_name>John Doe</full_name>
  <properties>
    <oxh:entry key="employer">Oracle</oxh:entry>
    <oxh:entry key="hair color">brown</oxh:entry>
    <oxh:entry key="favorite author">George RR Martin</oxh:entry>
  </properties>
</oxh:item>

```

The following example queries a file named `person-properties.avro` that contains `PersonAddress` records, and writes the names of the people that are employed by Oracle. The query shows how regular XPath expressions can retrieve map entries. Moreover, you can use the `avro:get` function as a shortcut to retrieve map entries.

```

import module "oxh:avro";
import module "oxh:text";

for $person in avro:collection-avroxml("person-properties.avro")
where $person/properties/oxh:entry[@key eq "employer"] eq "Oracle"
return
  text:put($person/full_name)

```

The following query uses the `avro:get` function to retrieve the employer entry. It is equivalent to the previous query.

```

import module "oxh:avro";
import module "oxh:text";

for $person in avro:collection-avroxml("person-properties.avro")
where $person/properties/avro:get("employer") eq "Oracle"
return
  text:put($person/full_name)

```

You can use XQuery `fn:nilled` function to test for null values. This example returns true if the map entry is null:

```

$var/avro:get("key")/nilled()

```

Reading Arrays

Oracle XQuery for Hadoop converts Avro array values to an element that contains a child `<oxh:item>` element for each item in the array. For example, consider the following schema:

```

{
  "type": "record",
  "name": "PersonScores",
  "fields" : [
    { "name": "full_name", "type": "string"},
    { "name": "scores", "type":

```

```

        {"type": "array", "items": "int"}
      }
    ]
  }
}

```

This is an instance of the schema as XML:

```

<oxh:item>
  <full_name>John Doe</full_name>
  <scores>
    <oxh:item>128</oxh:item>
    <oxh:item>151</oxh:item>
    <oxh:item>110</oxh:item>
  </scores>
</oxh:item>

```

The following example queries a file named `person-scores.avro` that contains `PersonScores` records, and writes the sum and count of scores for each person:

```

import module "oxh:avro";
import module "oxh:text";

for $person in avro:collection-avroxml("person-scores.avro")
let $scores := $person/scores/*
return
  text:put($person/full_name || ", " || sum($scores) || ", " || count($scores))

```

You can access a specific element of an array by using a numeric XPath predicate. For example, this path expression selects the second score. XPath indexing starts at 1 (not 0).

```
$person/scores/oxh:item[2]
```

Reading Unions

Oracle XQuery for Hadoop converts an instance of an Avro union type based on the actual member type of the value. The name of the member type is added as an XML `avro:type` attribute to the enclosing element, which ensures that queries can distinguish between instances of different member types. However, the attribute is not added for trivial unions where there are only two member types and one of them is null.

For example, consider the following union of two records:

```

[
  {
    "type": "record",
    "name": "Person1",
    "fields" : [
      {"name": "full_name", "type": "string"}
    ]
  },
  {
    "type": "record",
    "name": "Person2",
    "fields" : [
      {"name": "fname", "type": "string"}
    ]
  }
]

```

This is an instance of the schema as XML:

```
<oxh:item avro:type="Person2">
  <fname>John Doe</fname>
</oxh:item>
```

The following example queries a file named `person-union.avro` that contains instances of the previous union schema, and writes the names of the people from both record types to a text file:

```
import module "oxh:avro";
import module "oxh:text";

for $person in avro:collection-avroxml("examples/person-union.avro")
return
  if ($person/@avro:type eq "Person1") then
    text:put($person/full_name)
  else if ($person/@avro:type eq "Person2") then
    text:put($person/fname)
  else
    error(xs:QName("UNEXPECTED"), "Unexpected record type:" ||
$person/@avro:type)
```

Reading Primitives

[Table 6–1](#) shows how Oracle XQuery for Hadoop maps Avro primitive types to XQuery atomic types.

Table 6–1 Mapping Avro Primitive Types to XQuery Atomic Types

Avro	XQuery
boolean	xs:boolean
int	xs:int
long	xs:long
float	xs:float
double	xs:double
bytes	xs:hexBinary
string	xs:string

Avro null values are mapped to empty nilled elements. To distinguish between a null string value and an empty string value, use the XQuery `nilled` function. This path expression only returns true if the field value is null:

```
$record/field/nilled()
```

Avro fixed values are mapped to `xs:hexBinary`, and enums are mapped to `xs:string`.

Writing XML as Avro

Both the Avro file adapter and the Oracle NoSQL Database adapter have an `avroxml` method, which you can use with the `put` functions to write XML as Avro. The following topics describe how the XML is converted to an Avro instance:

- [Writing Records](#)
- [Writing Maps](#)
- [Writing Arrays](#)

- [Writing Unions](#)
- [Writing Primitives](#)

Writing Records

Oracle XQuery for Hadoop maps the XML to an Avro record schema by matching the child element names to the field names of the record. For example, consider the following Avro schema:

```
{
  "type": "record",
  "name": "Person",
  "fields" : [
    {"name": "full_name", "type": "string"},
    {"name": "age", "type": ["int", "null"]}
  ]
}
```

You can use the following XML element to write an instance of this record in which the `full_name` field is John Doe and the `age` field is 46. The name of the root element (`Person`) is inconsequential. Only the names of the child elements are used to map to the Avro record fields (`full_name` and `age`).

```
<person>
  <full_name>John Doe</full_name>
  <age>46</age>
</person>
```

The next example uses the following CSV file named `people.csv`:

```
John Doe,46
Jane Doe,37
.
.
.
```

This query converts values from the CSV file to Avro `Person` records:

```
import module "oxh:avro";
import module "oxh:text";

declare
  %avro:put("avroxml")
  %avro:schema('
    {
      "type": "record",
      "name": "Person",
      "fields" : [
        {"name": "full_name", "type": "string"},
        {"name": "age", "type": ["int", "null"]}
      ]
    }
  ')

function local:put-person($person as element()) external;

for $line in text:collection("people.csv")
let $split := tokenize($line, ",")
return
  local:put-person(
    <person>
      <full_name>{$split[1]}</full_name>
```

```

        <age>{$split[2]}</age>
    </person>
)

```

For null values, you can omit the element or set the `xsi:nil="true"` attribute. For example, this modified query sets age to null when the value is not numeric:

```

.
.
.
for $line in text:collection("people.csv")
let $split := tokenize($line, ",")
return
    local:put-person(
        <person>
            <full_name>{$split[1]}</full_name>
            {
                if ($split[2] castable as xs:int) then
                    <age>{$split[2]}</age>
                else
                    ()
            }
        </person>
    )

```

In the case of nested records, the values are obtained from nested elements. The next example uses the following schema:

```

{
  "type": "record",
  "name": "PersonAddress",
  "fields" : [
    { "name": "full_name", "type": "string" },
    { "name": "address", "type":
      { "type" : "record",
        "name" : "Address",
        "fields" : [
          { "name" : "street", "type" : "string" },
          { "name" : "city", "type" : "string" }
        ]
      }
    }
  ]
}

```

You can use following XML to write an instance of this record:

```

<person>
  <full_name>John Doe</full_name>
  <address>
    <street>123 First St.</street>
    <city>New York</city>
  </address>
</person>

```

Writing Maps

Oracle XQuery for Hadoop converts XML to an Avro map with one map entry for each `<oxh:entry>` child element. For example, consider the following schema:

```

{

```

```

"type": "record",
"name": "PersonProperties",
"fields" : [
  {"name": "full_name", "type": "string"},
  {"name": "properties", "type":
    {"type": "map", "values": "string"}
  }
]
}

```

You can use the following XML element to write an instance of this schema in which the `full_name` field is John Doe, and the `properties` field is set to a map with three entries:

```

<person>
  <full_name>John Doe</full_name>
  <properties>
    <oxh:entry key="hair color">brown</oxh:entry>
    <oxh:entry key="favorite author">George RR Martin</oxh:entry>
    <oxh:entry key="employer">Oracle</oxh:entry>
  </properties>
</person>

```

Writing Arrays

Oracle XQuery for Hadoop converts XML to an Avro array with one item for each `<oxh:item>` child element. For example, consider the following schema:

```

{
  "type": "record",
  "name": "PersonScores",
  "fields" : [
    {"name": "full_name", "type": "string"},
    {"name": "scores", "type":
      {"type": "array", "items": "int"}
    }
  ]
}

```

You can use the following XML element to write an instance of this schema in which the `full_name` field is John Doe and the `scores` field is set to [128, 151, 110]:

```

<person>
  <full_name>John Doe</full_name>
  <scores>
    <oxh:item>128</oxh:item>
    <oxh:item>151</oxh:item>
    <oxh:item>110</oxh:item>
  </scores>
</person>

```

Writing Unions

When writing an Avro union type, Oracle XQuery for Hadoop bases the selection of a member type on the value of the `avro:type` attribute.

This example uses the following schema:

```

[
  {
    "type": "record",
    "name": "Person1",
    "fields" : [

```

```

        {"name": "full_name", "type": "string"}
      ]
    }
  ,
  {
    "type": "record",
    "name": "Person2",
    "fields" : [
      {"name": "fname", "type": "string"}
    ]
  }
]

```

The following XML is mapped to an instance of the `Person1` record:

```

<person avro:type="Person1">
  <full_name>John Doe</full_name>
</person>

```

This XML is mapped to an instance of the `Person2` record:

```

<person avro:type="Person2">
  <fname>John Doe</fname>
</person>

```

The `avro:type` attribute selects the member type of the union. For trivial unions that contain a null and one other type, the `avro:type` attribute is unnecessary. If the member type cannot be determined, then an error is raised.

Writing Primitives

To map primitive values, Oracle XQuery for Hadoop uses the equivalent data types shown in [Table 6-1](#) to cast an XML value to the corresponding Avro type. If the value cannot be converted to the Avro type, an error is raised.

This example uses the following schema:

```

{
  "type": "record",
  "name": "Person",
  "fields" : [
    {"name": "full_name", "type": "string"},
    {"name": "age", "type": ["int", "null"]}
  ]
}

```

Attempting to map the following XML to an instance to this schema raises an error, because the string value `apple` cannot be converted to an int:

```

<person>
  <full_name>John Doe</full_name>
  <age>apple</age>
</person>

```

Oracle Database Adapter

The Oracle Database adapter provides custom functions for loading data into tables in Oracle Database.

A custom `put` function supported by this adapter automatically calls Oracle Loader for Hadoop at run time, either to load the data immediately or to output it to HDFS. You can declare and use multiple custom Oracle Database adapter `put` functions within a single query. For example, you might load data into different tables or into different Oracle databases with a single query.

Ensure that Oracle Loader for Hadoop is installed on your system, and that the `OLH_HOME` environment variable is set to the installation directory. See Steps 1 to 3 "[Installing Oracle Loader for Hadoop](#)" on page 1-12. Although not required, you might find it helpful to familiarize yourself with Oracle Loader for Hadoop before using this adapter.

The Oracle Database adapter is described in the following topics:

- [Custom Functions for Writing to Oracle Database](#)
- [%oracle-property Annotations and Corresponding Oracle Loader for Hadoop Configuration Properties](#)

See Also:

- ["Software Requirements"](#) on page 1-12 for the supported versions of Oracle Database
- [Chapter 3, "Oracle Loader for Hadoop"](#)

Custom Functions for Writing to Oracle Database

You can use the following annotations to define functions that write to tables in an Oracle database either directly or by generating binary or text files for subsequent loading with another utility, such as SQL*Loader.

Signature

Custom functions for writing to Oracle database tables must have the following signature:

```
declare %oracle:put(["jdbc" | "oci" | "text" | "datapump"])
        [%oracle:columns(col1 [, col2...])] [%oracle-property annotations]
        function local:myPut($column1 [as xs:allowed_type_name[?]], [$column2 [as
xs:allowed_type_name[?]], ...]) external;
```

Annotations

%oracle:put("output_mode"?)

Declares the put function and the output mode. Required.

The optional *output_mode* parameter can be one of the following string literal values:

- `jdbc`: Writes to an Oracle database table using a JDBC connection. Default.
See "[JDBC Output Format](#)" on page 3-17.
- `oci`: Writes to an Oracle database table using an Oracle Call Interface (OCI) connection.
See "[Oracle OCI Direct Path Output Format](#)" on page 3-18.
- `datapump`: Creates Data Pump files and associated scripts in HDFS for subsequent loading with another utility.
See "[Oracle Data Pump Output Format](#)" on page 3-20.
- `text`: Creates delimited text files and associated scripts in HDFS
See "[Delimited Text Output Format](#)" on page 3-19.

To write directly to an Oracle database table using either JDBC or OCI, all systems involved in processing the query must be able to connect to the Oracle Database system. See "[About the Modes of Operation](#)" on page 3-2.

%oracle:columns(col1 [, col2...])

Identifies a selection of one or more column names in the target table. The order of column names corresponds to the order of the function parameters. See "[Parameters](#)" on page 6-21. Optional.

This annotation supports loading a subset of the table columns. If omitted, the put function attempts to load all columns of the target table.

%oracle-property:property_name (value)

Controls various aspects of connecting to the database and writing data. See "[%oracle-property Annotations and Corresponding Oracle Loader for Hadoop Configuration Properties](#)" on page 6-26 for descriptions of the supported properties. You can specify multiple `%oracle-property` annotations.

These annotations correspond to the Oracle Loader for Hadoop configuration properties. Every `%oracle-property` annotation has an equivalent Oracle Loader for

Hadoop configuration property, which you can specify in a configuration file passed to Hadoop with the generic `-conf` option. See ["Running a Query"](#) on page 5-13.

The `%oracle-property` annotations are optional. However, the various loading scenarios require you to specify some of them or their equivalent configuration properties. For example, to load data into an Oracle database using JDBC or OCI, you must specify the target table and the connection information.

The following example specifies a target table named `VISITS`, a user name of `db`, a password of `password`, and the URL connection string:

```
%oracle-property:targetTable('visits')
%oracle-property:connection.user('db')
%oracle-property:connection.password('password')
%oracle-property:connection.url('jdbc:oracle:thin:@//localhost:1521/orcl.example.com')
```

Parameters

\$column1 [as xs:allowed_type_name[?]], [\$column2 [as xs:allowed_type_name[?]],...]

Enter a parameter for each column in the same order as the Oracle table columns to load all columns, or use the `%oracle:columns` annotation to load selected columns.

Because the correlation between parameters and database columns is positional, the name of the parameter (*column1* in the parameter syntax) is not required to match the name of the database column.

You can omit the explicit `as xs:allowed_type_name` type declaration for any parameter. For example, you can declare the parameter corresponding to a `NUMBER` column simply as `$column1`. In this case, the parameter is automatically assigned an XQuery type of `item()*`. At run time, the input value is cast to the allowed XQuery type for the corresponding table column type, as described in [Table 6-2](#). For example, data values that are mapped to a column with a `NUMBER` data type are automatically cast as `xs:decimal`. An error is raised if the cast fails.

Alternatively, you can specify the type or its subtype for any parameter. In this case, compile-time type checking is performed. For example, you can declare a parameter corresponding to a `NUMBER` column as `$column as xs:decimal`. You can also declare it as any subtype of `xs:decimal`, such as `xs:integer`.

You can include the `?` optional occurrence indicator for each specified parameter type. This indicator allows the empty sequence to be passed as a parameter value at run time, so that a null is inserted into the database table. Any occurrence indicator other than `?` raises a compile-time error.

[Table 6-2](#) describes the appropriate mappings of XQuery data types with the supported Oracle Database data types. In addition to the listed XQuery data types, you can also use the subtypes, such as `xs:integer` instead of `xs:decimal`. Oracle data types are more restrictive than XQuery data types, and these restrictions are identified in the table.

Table 6-2 Data Type Mappings Between Oracle Database and XQuery

Database Type	XQuery Type
VARCHAR2	xs:string Limited by the VARCHAR2 maximum size of 4000 bytes

Table 6–2 (Cont.) Data Type Mappings Between Oracle Database and XQuery

Database Type	XQuery Type
CHAR	xs:string Limited by the CHAR maximum size of 2000 bytes
NVARCHAR2	xs:string Limited by the NVARCHAR2 maximum size of 4000 bytes
NCHAR	xs:string Limited by the NCHAR maximum size of 2000 bytes
DATE	xs:dateTime Limited to the range of January 1, 4712 BC to December 31, 9999 CE. If a time zone is specified in the xs:dateTime value, then the time zone information is dropped. Fractional seconds are also dropped. A time value of 24:00:00 is not valid.
TIMESTAMP	xs:dateTime Limited to the range of January 1, 4712 BC to December 31, 9999 CE. If a time zone is specified in the xs:dateTime value, then the time zone information is dropped. Fractional seconds are limited to a precision of 0 to 9 digits. A time value of 24:00:00 is not valid.
TIMESTAMP W LOCAL TIME ZONE	xs:dateTime Limited to the range of January 1, 4712 BC to December 31, 9999 CE. In the offset from UTC, the time-zone hour field is limited to -12:00 to 14:00. Fractional seconds are limited to a precision of 0 to 9 digits. See "About Session Time Zones" on page 6-23.
TIMESTAMP W TIME ZONE	xs:dateTime Limited to the range of January 1, 4712 BC to December 31, 9999 CE. In the offset from UTC, the time-zone hour field is limited to -12:00 to 14:00. Fractional seconds are limited to a precision of 0 to 9 digits. See "About Session Time Zones" on page 6-23.
INTERVAL DAY TO SECOND	xs:dateTimeDuration The day and fractional seconds are limited by a precision of 0 to 9 digits each. The hour is limited to a range of 0 to 23, and minutes and seconds are limited to a range of 0 to 59.
INTERVAL YEAR TO MONTH	xs:yearMonthDuration The year is limited by a precision of 0 to 9 digits, and the month is limited to a range of 0 to 11.
BINARY_FLOAT	xs:float
BINARY_DOUBLE	xs:double
NUMBER	xs:decimal Limited by the NUMBER precision of 1 to 38 decimal digits and scale of -84 to 127 decimal digits.
FLOAT	xs:decimal Limited by the FLOAT precision of 1 to 126 binary digits
RAW	xs:hexBinary Limit by the RAW maximum size of 2000 bytes

About Session Time Zones

If an `xs:dateTime` value with no time zone is loaded into `TIMESTAMP W TIME ZONE` or `TIMESTAMP W LOCAL TIME ZONE`, then the time zone is set to the value of the `sessionTimeZone` parameter, which defaults to the JVM time zone. Using Oracle XQuery for Hadoop, you can set the `sessionTimeZone` property, as described in "[%oracle-property Annotations and Corresponding Oracle Loader for Hadoop Configuration Properties](#)" on page 6-26.

Notes

With JDBC or OCI output modes, the Oracle Database Adapter loads data directly into the database table. It also creates a directory with the same name as the custom `put` function name, under the query output directory. For example, if your query output directory is `myoutput`, and your custom function is `myPut`, then the `myoutput/myPut` directory is created.

For every custom Oracle Database Adapter `put` function, a separate directory is created. This directory contains output produced by the Oracle Loader for Hadoop job. When you use `datapump` or `text` output modes, the data files are written to this directory. The control and SQL scripts for loading the files are written to the `_olh` subdirectory, such as `myoutput/myPut/_olh`.

For descriptions of the generated files, see "[Delimited Text Output Format](#)" on page 3-19 and "[Oracle Data Pump Output Format](#)" on page 3-20.

Examples of Oracle Database Adapter Functions

These examples use following text files in HDFS. The files contain a log of visits to different web pages. Each line represents a visit to a web page and contains the time, user name, and page visited.

`mydata/visits1.log`

```
2013-10-28T06:00:00, john, index.html, 200
2013-10-28T08:30:02, kelly, index.html, 200
2013-10-28T08:32:50, kelly, about.html, 200
2013-10-30T10:00:10, mike, index.html, 401
```

`mydata/visits2.log`

```
2013-10-30T10:00:01, john, index.html, 200
2013-10-30T10:05:20, john, about.html, 200
2013-11-01T08:00:08, laura, index.html, 200
2013-11-04T06:12:51, kelly, index.html, 200
2013-11-04T06:12:40, kelly, contact.html, 200
```

The examples also uses the following file in HDFS, which contains anonymous page visits.

`mydata/anonvisits.log`

```
2011-10-30T10:01:01, index.html, 401
2011-11-04T06:15:40, contact.html, 401
```

The `VISITS` table in the Oracle database is created with this SQL command:

```
CREATE TABLE visits (time TIMESTAMP, name VARCHAR2(15), page VARCHAR2(15), code
NUMBER)
```

Example 1 Loading All Columns

The first query loads all information related to the page visit (time of visit, user name, page visited, and status code) to the VISITS table. In case of anonymous access, the user name is missing, so the query specifies () to insert a null into the table. The target table name, user name, password, and connection URL are specified with %oracle-property annotations.

While the example uses a clear-text user name and password, Oracle recommends that you use a wallet instead for security. You can configure an Oracle wallet using either Oracle Loader for Hadoop properties or their equivalent %oracle-property annotations. The specific properties that you must set are described in ["Providing the Connection Details for Online Database Mode"](#) on page 3-8.

```
import module "oxh:text";

declare
  %oracle:put
  %oracle-property:targetTable('visits')
  %oracle-property:connection.user('db')
  %oracle-property:connection.password('password')
  %oracle-property:connection.url('jdbc:oracle:thin:@//localhost:1521/orcl.example.com')
function local:myPut($c1, $c2, $c3, $c4) external;

for $line in text:collection("*visits*.log")
let $split := fn:tokenize($line, "\s*\s*")
return
  if (count($split) > 3) then
    local:myPut(xs:dateTime($split[1]), $split[2], $split[3], $split[4])
  else
    local:myPut(xs:dateTime($split[1]), (), $split[2], $split[3])
```

The VISITS table contains the following data after the query runs:

TIME	NAME	PAGE	CODE
30-OCT-13 10.00.01.000000 AM	john	index.html	200
30-OCT-13 10.05.20.000000 AM	john	about.html	200
01-NOV-13 08.00.08.000000 AM	laura	index.html	200
04-NOV-13 06.12.51.000000 AM	kelly	index.html	200
04-NOV-13 06.12.40.000000 AM	kelly	contact.html	200
28-OCT-13 06.00.00.000000 AM	john	index.html	200
28-OCT-13 08.30.02.000000 AM	kelly	index.html	200
28-OCT-13 08.32.50.000000 AM	kelly	about.html	200
30-OCT-13 10.00.10.000000 AM	mike	index.html	401
30-OCT-11 10.01.01.000000 AM		index.html	401
04-NOV-11 06.15.40.000000 AM		contact.html	401

Example 2 Loading Selected Columns

This example uses the %oracle:columns annotation to load only the time and name columns of the table. It also loads only visits by john.

The column names specified in %oracle:columns are positionally correlated to the put function parameters. Data values provided for the \$c1 parameter are loaded into the TIME column, and data values provided for the \$c2 parameter are loaded into the NAME column.

```
import module "oxh:text";

declare
```

```

%oracle:put
%oracle:columns('time', 'name')
%oracle-property:targetTable('visits')
%oracle-property:connection.user('db')
%oracle-property:connection.password('password')
%oracle-property:connection.url('jdbc:oracle:thin:@//localhost:1521/orcl.example.com')
function local:myPut($c1, $c2) external;

for $line in text:collection("*visits*.log")
let $split := fn:tokenize($line, "\s*\s*")
where $split[2] eq 'john'
return
    local:myPut(xs:dateTime($split[1]), $split[2])

```

If the VISITS table is empty before the query runs, then it contains the following data afterward:

TIME	NAME	PAGE	CODE
30-OCT-13 10.00.01.000000 AM	john		
30-OCT-13 10.05.20.000000 AM	john		
28-OCT-13 06.00.00.000000 AM	john		

%oracle-property Annotations and Corresponding Oracle Loader for Hadoop Configuration Properties

When you use the Oracle Database adapter of Oracle XQuery for Hadoop, you indirectly use Oracle Loader for Hadoop. Oracle Loader for Hadoop defines configuration properties that control various aspects of connecting to Oracle Database and writing data. Oracle XQuery for Hadoop supports many of these properties. All supported properties are listed in [Table 6-3](#).

In Oracle XQuery for Hadoop, you can specify these properties with the generic `-conf` and `-D Hadoop` command-line options, just like you specify them when you use Oracle Loader for Hadoop directly. Alternatively, you can specify these properties as custom put function annotations with the `%oracle-property` prefix.

When you specify a property with the Hadoop options, it applies to all custom Oracle Database Adapter put functions in your query. When you specify it as an `%oracle-property` annotation, it applies only to the particular custom put function that has this annotation.

For example, this annotation sets the target table to `VISITS` only for the particular put function that has the annotation in the declaration.

```
%oracle-property:connection.url('visits')
```

Alternatively, you can specify the table with `-D` by using the corresponding property name. This property sets the target table to `VISITS` for all Oracle Database adapter put functions in your query,

```
-D oracle.hadoop.loader.targetTable=visits
```

This flexibility is provided for convenience. For example, if a query has multiple put functions, each writing to a different table in the same database, you might specify the properties like this:

- The database connection URL in the `oracle.hadoop.loader.connection.url` property in the configuration file. Then identify the configuration file using the `-conf` option, thereby applying it to all put functions in the query.
- The different table names using the `%oracle-property:targetTable` annotation in each put function.

If you already use Oracle Loader for Hadoop and have a configuration file, you can simply copy-and-paste these properties into the Oracle XQuery for Hadoop configuration file and identify it to Hadoop with the `-conf` option.

[Table 6-3](#) identifies the annotations and their equivalent Oracle Loader for Hadoop properties. Oracle XQuery for Hadoop only supports the Oracle Loader for Hadoop properties listed in this table.

Table 6-3 %oracle-property Annotations and Corresponding Oracle Loader for Hadoop Configuration Properties

Annotation	Property
<code>%oracle-property:targetTable</code>	<code>oracle.hadoop.loader.targetTable</code>
<code>%oracle-property:connection.user</code>	<code>oracle.hadoop.loader.connection.user</code>
<code>%oracle-property:connection.password</code>	<code>oracle.hadoop.loader.connection.password</code>
<code>%oracle-property:connection.url</code>	<code>oracle.hadoop.loader.connection.url</code>

Table 6–3 (Cont.) %oracle-property Annotations and Corresponding Oracle Loader for Hadoop Configuration Properties

Annotation	Property
%oracle-property:connection.wallet_location	oracle.hadoop.loader.connection.wallet_location
%oracle-property:connection.tns_admin	oracle.hadoop.loader.connection.tns_admin
%oracle-property:connection.tnsEntryName	oracle.hadoop.loader.connection.tnsEntryName
%oracle-property:connection.defaultExecuteBatch	oracle.hadoop.loader.connection.defaultExecuteBatch
%oracle-property:connection.oci_url	oracle.hadoop.loader.connection.oci_url
%oracle-property:connection.sessionTimeZone	oracle.hadoop.loader.connection.sessionTimeZone
%oracle-property:extTabDirectoryName	oracle.hadoop.loader.extTabDirectoryName
%oracle-property:input.initialFieldEncloser	oracle.hadoop.loader.input.initialFieldEncloser
%oracle-property:output.escapeEnclosers	oracle.hadoop.loader.output.escapeEnclosers
%oracle-property:output.fieldTerminator	oracle.hadoop.loader.output.fieldTerminator
%oracle-property:output.granuleSize	oracle.hadoop.loader.output.granuleSize
%oracle-property:output.initialFieldEncloser	oracle.hadoop.loader.output.initialFieldEncloser
%oracle-property:output.trailingFieldEncloser	oracle.hadoop.loader.output.trailingFieldEncloser
%oracle-property:sampler.enableSampling	oracle.hadoop.loader.sampler.enableSampling
%oracle-property:sampler.hintMaxSplitSize	oracle.hadoop.loader.sampler.hintMaxSplitSize
%oracle-property:sampler.hintNumMapTasks	oracle.hadoop.loader.sampler.hintNumMapTasks
%oracle-property:sampler.loadCI	oracle.hadoop.loader.sampler.loadCI
%oracle-property:sampler.maxHeapBytes	oracle.hadoop.loader.sampler.maxHeapBytes
%oracle-property:sampler.maxLoadFactor	oracle.hadoop.loader.sampler.maxLoadFactor
%oracle-property:sampler.maxSamplesPct	oracle.hadoop.loader.sampler.maxSamplesPct
%oracle-property:sampler.minSplits	oracle.hadoop.loader.sampler.minSplits
%oracle-property:sampler.numThreads	oracle.hadoop.loader.sampler.numThreads
%oracle-property:tableMetadataFile	oracle.hadoop.loader.tableMetadataFile
%oracle-property:loadByPartition	oracle.hadoop.loader.loadByPartition
%oracle-property:enableSorting	oracle.hadoop.loader.enableSorting
%oracle-property:sortKey	oracle.hadoop.loader.sortKey
%oracle-property:rejectLimit	oracle.hadoop.loader.rejectLimit

Table 6–3 (Cont.) %oracle-property Annotations and Corresponding Oracle Loader for Hadoop Configuration Properties

Annotation	Property
%oracle-property:badRecordFlushInterval	<code>oracle.hadoop.loader.badRecordFlushInterval</code>
%oracle-property:compressionFactors	<code>oracle.hadoop.loader.compressionFactors</code>
%oracle-property:logBadRecords	<code>oracle.hadoop.loader.logBadRecords</code>
%oracle-property:dirpathBufsize	<code>oracle.hadoop.loader.output.dirpathBufsize</code>

Oracle NoSQL Database Adapter

This adapter provides functions to read and write values stored in Oracle NoSQL Database.

This adapter is described in the following topics:

- [Prerequisites for Using the Oracle NoSQL Database Adapter](#)
- [Built-in Functions for Reading from and Writing to Oracle NoSQL Database](#)
- [Custom Functions for Reading Values from Oracle NoSQL Database](#)
- [Custom Functions for Retrieving Single Values from Oracle NoSQL Database](#)
- [Custom Functions for Writing to Oracle NoSQL Database](#)
- [Oracle NoSQL Database Adapter Configuration Properties](#)

Prerequisites for Using the Oracle NoSQL Database Adapter

Before you write queries that use the Oracle NoSQL Database adapter, you must configure Oracle XQuery for Hadoop to use your Oracle NoSQL Database server.

You must set the following:

- The `KVHOME` environment variable to the local directory containing the Oracle NoSQL database lib directory.
- The `oracle.kv.hosts` and `oracle.kv.kvstore` configuration properties.

You can set the configuration properties using either the `-D` or `-conf` arguments in the `hadoop` command when you run the query. See ["Running a Query"](#) on page 5-13.

This example sets `KVHOME` and uses the Hadoop `-D` argument in a query to set `oracle.kv.kvstore`:

```
$ export KVHOME=/local/path/to/kvstore/  
$ hadoop jar $OXH_HOME/lib/oxh.jar -D oracle.kv.hosts=example.com:5000 -D  
oracle.kv.kvstore=kvstore ./myquery.xq -output ./myoutput
```

See ["Oracle NoSQL Database Adapter Configuration Properties"](#) on page 6-44.

Built-in Functions for Reading from and Writing to Oracle NoSQL Database

To use the built-in functions in your query, you must import the Oracle NoSQL Database module as follows

```
import module "oxh:kv";
```

The Oracle NoSQL Database module contains the following functions:

- [kv:collection-text](#)
- [kv:collection-avroxml](#)
- [kv:collection-xml](#)
- [kv:collection-binxml](#)
- [kv:put-text](#)
- [kv:put-xml](#)
- [kv:put-binxml](#)
- [kv:get-text](#)
- [kv:get-avroxml](#)
- [kv:get-xml](#)
- [kv:get-binxml](#)
- [kv:key-range](#)

kv:collection-text

Accesses a collection of values in the database. Each value is decoded as UTF-8 and returned as a string.

Signature

```
declare %kv:collection("text") function
  kv:collection-text($parent-key as xs:string?, $depth as xs:int?, $subrange as
  xs:string?) as xs:string* external;
```

Parameters

See "[Parameters](#)" on page 6-39.

Returns

One string for each value

kv:collection-text

This function is equivalent to `kv:collection-text($parent-key, $depth, ())`.

Signature

```
declare %kv:collection("text") function
  kv:collection-text($parent-key as xs:string?, $depth as xs:int?) as xs:string*
  external;
```

kv:collection-text

This function is equivalent to `kv:collection-text($parent-key, ())`.

Signature

```
declare %kv:collection("text") function
    kv:collection-text($parent-key as xs:string?) as xs:string* external;
```

kv:collection-avroxml

Accesses a collection of values in the database. Each value is read as an Avro record and returned as an XML element. The records are converted to XML as described in ["Reading Records"](#) on page 6-10..

Signature

```
declare %kv:collection("avroxml") function
    kv:collection-avroxml($parent-key as xs:string?, $depth as xs:int?, $subrange
as xs:string?) as element()* external;
```

Parameters

See ["Parameters"](#) on page 6-39.

Returns

One XML element for each Avro record

kv:collection-avroxml

This function is equivalent to `kv:collection-avroxml($parent-key, $depth, ())`.

Signature

```
declare %kv:collection("avroxml") function
    kv:collection-avroxml($parent-key as xs:string?, $depth as xs:int?) as
element()* external;
```

kv:collection-avroxml

This function is equivalent to `kv:collection-avroxml($parent-key, ())`.

Signature

```
declare %kv:collection("avroxml") function
    kv:collection-avroxml($parent-key as xs:string?) as element()* external;
```

kv:collection-xml

Accesses a collection of values in the database. Each value is read as a sequence of bytes and parsed as XML.

Signature

```
declare %kv:collection("xml") function
    kv:collection-xml($parent-key as xs:string?, $depth as xs:int?, $subrange as
xs:string?) as document-node()* external;
```

Parameters

See ["Parameters"](#) on page 6-39.

Returns

One XML document for each value.

kv:collection-xml

This function is equivalent to `kv:collection-xml($parent-key, $depth, ())`.

Signature

```
declare %kv:collection("xml") function
    kv:collection-xml($parent-key as xs:string?, $depth as xs:int?) as
    document-node()* external;
```

kv:collection-xml

This function is equivalent to `kv:collection-xml($parent-key, ())`.

Signature

```
declare %kv:collection("xml") function
    kv:collection-xml($parent-key as xs:string?) as document-node()* external;
```

kv:collection-binxml

Accesses a collection of values in the database. Each value is read as XDK binary XML and returned as an XML document.

Signature

```
declare %kv:collection("binxml") function
    kv:collection-binxml($parent-key as xs:string?, $depth as xs:int?, $subrange as
    xs:string?) as document-node()* external;
```

Parameters

See ["Parameters"](#) on page 6-39.

Returns

One XML document for each value.

See Also

Oracle XML Developer's Kit Programmer's Guide

kv:collection-binxml

This function is equivalent to `kv:collection-binxml($parent-key, $depth, ())`.

Signature

```
declare %kv:collection("binxml") function
    kv:collection-binxml($parent-key as xs:string?, $depth as xs:int?) as
    document-node()* external;
```

kv:collection-binxml

This function is equivalent to `kv:collection-binxml($parent-key, ())`.

Signature

```
declare %kv:collection("binxml") function
    kv:collection-binxml($parent-key as xs:string?) as document-node()* external;
```

kv:collection-binxml

This function is equivalent to `kv:collection-binxml($parent-key, ())`.

Signature

```
declare %kv:collection("binxml") function
    kv:collection-binxml($parent-key as xs:string?) as document-node()* external;
```

kv:put-text

Writes a key-value pair. The `$value` is encoded as UTF-8.

Signature

```
declare %kv:put("text") function
    kv:put-text($key as xs:string, $value as xs:string) external;
```

kv:put-xml

Writes a key/value pair. The `$xml` is serialized and encoded as UTF-8.

Signature

```
declare %kv:put("xml") function
    kv:put-xml($key as xs:string, $xml as node()) external;
```

kv:put-binxml

Puts a key/value pair. The `$xml` is encoded as XDK binary XML. See *Oracle XML Developer's Kit Programmer's Guide*.

Signature

```
declare %kv:putkv:put-binxml("binxml") function
    ($key as xs:string, $xml as node()) external;
```

kv:get-text

Obtains the value associated with the key. The value is decoded as UTF-8 and returned as a string.

Signature

```
declare %kv:get("text") function
    kv:get-text($key as xs:string) as xs:string? external;
```

kv:get-avroxml

Obtains the value associated with the key. The value is read as an Avro record and returned as an XML element. The records are converted to XML as described in ["Reading Records"](#) on page 6-10..

Signature

```
declare %kv:get("avroxml") function
  kv:get-avroxml($key as xs:string) as element()? external;
```

kv:get-xml

Obtains the value associated with the key. The value is read as a sequence of bytes and parsed as XML.

```
declare %kv:get("xml") function
  kv:get-xml($key as xs:string) as document-node()? external;
```

kv:get-binxml

Obtains the value associated with the key. The value is read as XDK binary XML and returned as an XML document.

Signature

```
declare %kv:get("binxml") function
  kv:get-binxml($key as xs:string) as document-node()? external;
```

See Also

Oracle XML Developer's Kit Programmer's Guide

kv:key-range

Defines a prefix range. The prefix defines both the lower and upper inclusive boundaries.

Use this function as the *subrange* argument of a `kv:collection` function.

Signature

```
kv:key-range($prefix as xs:string) as xs:string;
```

kv:key-range

Specifies a key range.

Use this function as the *subrange* argument of a `kv:collection` function.

Signature

```
kv:key-range($start as xs:string, $start-inclusive as xs:boolean, $end as
xs:string, $end-inclusive as xs:boolean) as xs:string;
```

Parameters

start

Defines the lower boundary of the key range.

start-inclusive

A value of `true` includes `$start` in the range, or `false` omits it.

end

Defines the upper boundary of the key range. It must be greater than `$start`.

end-inclusive

A value of `true` includes `$end` in the range, or `false` omits it.

Oracle NoSQL Database Adapter Examples**Example 1 Writing and Reading Text in Oracle NoSQL Database**

This example uses the following text file in HDFS. The file contains user profile information such as user ID, full name, and age, separated by a colon (:).

```
mydata/users.txt
```

```
john:John Doe:45
kelly:Kelly Johnson:32
laura:Laura Smith:
phil:Phil Johnson:27
```

The first query stores the lines of this text file in Oracle NoSQL Database as text values.

```
import module "oxh:text";
import module "oxh:kv";

for $line in text:collection("mydata/users.txt")
let $split := fn:tokenize($line, ":")
let $key := "/users/text/" || $split[1]
return
  kv:put-text($key, $line)
```

The next query reads the values from the database:

```
import module "oxh:text";
import module "oxh:kv";

for $value in kv:collection-text("/users/text")
let $split := fn:tokenize($value, ":")
where $split[2] eq "Phil Johnson"
return
  text:put($value)
```

The query creates a text file that contains the following line:

```
phil:Phil Johnson:27
```

Example 2 Writing and Reading Avro in Oracle NoSQL Database

In this example, the following Avro schema is registered with Oracle NoSQL Database:

```
{
  "type": "record",
  "name": "User",
  "namespace": "com.example",
  "fields" : [
    {"name": "id", "type": "string"},
    {"name": "full_name", "type": "string"},
    {"name": "age", "type": ["int", "null"]}
  ]
}
```

The next query writes the user names to the database as Avro records.

```
import module "oxh:text";

declare %kv:put("avroxml") %avro:schema-kv("com.example.User")
```

```

function local:put-user($key as xs:string, $value as node()) external;

for $line in text:collection("mydata/users.txt")
let $split := fn:tokenize($line, ":")
let $id := $split[1]
let $key := "/users/avro/" || $id
return
  local:put-user(
    $key,
    <user>
      <id>{$id}</id>
      <full_name>{$split[2]}</full_name>
      {
        if ($split[3] castable as xs:int) then
          <age>{$split[3]}</age>
        else
          ()
      }
    </user>
  )

```

This query reads the values from the database:

```

import module "oxh:text";
import module "oxh:kv";

for $user in kv:collection-avroxml("/users/avro")
where $user/age gt 30
return
  text:put($user/full_name)

```

The query creates a text files with the following lines:

```

John Doe
Kelly Johnson

```

Custom Functions for Reading Values from Oracle NoSQL Database

You can use the following functions to read values from Oracle NoSQL Database. These annotations provide additional functionality that is not available using the built-in functions.

Signature

Custom functions for reading collections of NoSQL values must have one of the following signatures:

```
declare %kv:collection("text") [additional annotations]
    function local:myFunctionName($parent-key as xs:string?, $depth as xs:int?,
    $subrange as xs:string?) as xs:string* external;
```

```
declare %kv:collection(["xml"|"binxml"]) [additional annotations]
    function local:myFunctionName($parent-key as xs:string?, $depth as xs:int?,
    $subrange as xs:string?) as document-node()* external;
```

```
declare %kv:collection("avroxml") [additional annotations]
    function local:myFunctionName($parent-key as xs:string?, $depth as xs:int?,
    $subrange as xs:string?) as element()* external;
```

Annotations

%kv:collection("method")

Declares the NoSQL Database collection function. Required.

The *method* parameter is one of the following values:

- `text`: Each value is decoded using the character set specified by the `%output:encoding` annotation.
- `avroxml`: Each value is read as an Avro record and returned as an XML element. The records are converted to XML as described in ["Reading Records"](#) on page 6-10.
- `binxml`: Each value is read as XDK binary XML and returned as an XML document.
- `xml`: Each value is parsed as XML, and returned as an XML document.

%kv:key("true" | "false")

Controls whether the key of a key-value pair is set as the document-uri of the returned value. Specify `true` to return the key.

The default setting is `true` when *method* is `xml`, `avroxml`, or `binxml`, and `false` when it is `text`. Text functions with this annotation set to `true` must be declared to return `text()?` instead of `xs:string?`. Atomic `xs:string` values are not associated with a document node, but text nodes are. For example:

```
declare %kv:collection("text") %kv:key("true")
    function local:col($parent-key as xs:string?) as text()* external;
```

When the key is returned, you can obtain its string representation by using the `kv:key()` function. For example:

```
for $value in local:col(...)
let $key := $value/kv:key()
return ...
```

%avro:schema-kv("schema-name")

Specifies the Avro reader schema. This annotation is valid only when *method* is `avroxml`. Optional.

The *schema-name* is a fully qualified record name. The record schema is retrieved from the Oracle NoSQL Database catalog. The record value is mapped to the reader schema. For example, `%avro:schema-kv("org.example.PersonRecord")`.

See Also: For information about Avro schemas, the *Oracle NoSQL Database Getting Started Guide* at

<http://docs.oracle.com/cd/NOSQL/html/GettingStartedGuide/schmaevolution.html>

%output:encoding

Specifies the character encoding of text values. UTF-8 is assumed when this annotation is not used. The valid encodings are those supported by the JDK.

This annotation currently only applies to the text method. For XML files, the document's encoding declaration is used if it is available.

See Also: "Supported Encodings" in the Oracle Java SE documentation at

<http://docs.oracle.com/javase/7/docs/technotes/guides/intl/encoding.doc.html>

Parameters

Parameter 1: \$parent-key as xs:string?

Specifies the parent key whose child KV pairs are returned by the function. The major key path must be a partial path and the minor key path must be empty. An empty sequence results in fetching all keys in the store.

See Also: For the format of the key, the *Oracle NoSQL Database Java Reference* at

<http://docs.oracle.com/cd/NOSQL/html/javadoc/oracle/kv/Key.html#toString>

Parameter 2: \$depth as xs:int?

Specifies whether parents, children, descendants, or a combination are returned. The following values are valid:

- `kv:depth-parent-and-descendants()`: Selects the parents and all descendants.
- `kv:depth-children-only()`: Selects only the immediately children, but not the parent.
- `kv:depth-descendants-only()`: Selects all descendants, but not the parent.
- `kv:depth-parent-and-children()`: Selects the parent and the immediate children.

An empty sequence implies `kv:depth-parent-and-descendants()`.

This example selects all the descendants, but not the parent:

```
kv:collection-text("/parent/key", kv:depth-descendants-only(), ...)
```

Parameter 3: \$subRange as xs:string?

Specifies a subrange to further restrict the range under parentKey to the major path components. The format of the string is:

<startType>/<start>/<end>/<endType>

The startType and endType are either I for inclusive or E for exclusive.

The start and end are the starting and ending key strings.

If the range does not have a lower boundary, then omit the leading startType/start specification from the string representation. Similarly, if the range does not have an upper boundary, then omit the trailing end/endType specification. A KeyRange requires at least one boundary, thus at least one specification must be present in the string representation.

The kv:key-range function provides a convenient way to create a range string.

The value can also be the empty sequence.

The following examples are valid subrange specifications:

Example	Description
I/alpha/beta/E	From alpha inclusive to beta exclusive
E//0123/I	From "" exclusive to 0123 inclusive
I/chi/	From chi inclusive to infinity
E//	From "" exclusive to infinity
/chi/E	From negative infinity to chi exclusive
//I	From negative infinity to "" inclusive

Custom Functions for Retrieving Single Values from Oracle NoSQL Database

The Oracle NoSQL Database adapter has `get` functions, which enable you to retrieve a single value from the database. Unlike `collection` functions, calls to `get` functions are not distributed across the cluster. When a `get` function is called, the value is retrieved by a single task.

Signature

Custom `get` functions must have one of the following signatures:

```
declare %kv:get("text") [additional annotations]
    function local:myFunctionName($key as xs:string) as xs:string? external;

declare %kv:get("avroxml") [additional annotations]
    function local:myFunctionName($key as xs:string) as element()? external;

declare %kv:get(["xml"|"binxml"]) [additional annotations]
    function local:myFunctionName($key as xs:string) as document-node()?
```

Annotations

%kv:get("method")

Declares the NoSQL Database `get` function. Required.

The *method* parameter is one of the following values:

- `text`: The value is decoded using the character set specified by the `%output:encoding` annotation.
- `avroxml`: The value is read as an Avro record and returned as an XML element. The records are converted to XML as described in ["Reading Records"](#) on page 6-10.
- `binxml`: The value is read as XDK binary XML and returned as an XML document.
- `xml`: The value is parsed as XML, and returned as an XML document.

%kv:key("true" | "false")

Controls whether the key of a key-value pair is set as the `document-uri` of the returned value. Specify `true` to return the key.

The default setting is `true` when *method* is `xml`, `avroxml`, or `binxml`, and `false` when it is `text`. Text functions with this annotation set to `true` must be declared to return `text()?` instead of `xs:string?`. Atomic `xs:string` values are not associated with a document node, but text nodes are.

When the key is returned, you can obtain its string representation by using the `kv:key()` function.

%avro:schema-kv("schema-name")

Specifies the Avro reader schema. This annotation is valid only when *method* is `avroxml`. Optional.

The *schema-name* is a fully qualified record name. The record schema is retrieved from the Oracle NoSQL Database catalog. The record value is mapped to the reader schema. For example, `%avro:schema-kv("org.example.PersonRecord")`.

See Also: For information about Avro schemas, the *Oracle NoSQL Database Getting Started Guide* at

<http://docs.oracle.com/cd/NOSQL/html/GettingStartedGuide/schmaevolution.html>

%output:encoding

Specifies the character encoding of text values. UTF-8 is assumed when this annotation is not used. The valid encodings are those supported by the JDK.

This annotation currently only applies to the text method. For XML files, the document encoding declaration is used, if it is available.

See Also: "Supported Encodings" in the Oracle Java SE documentation at

<http://docs.oracle.com/javase/7/docs/technotes/guides/intl/encoding.doc.html>

Custom Functions for Writing to Oracle NoSQL Database

You can use the following annotations to define functions that write to Oracle NoSQL Database.

Signature

Custom functions for writing to Oracle NoSQL Database must have one of the following signatures:

```
declare %kv:put("text") function
    local:myFunctionName($key as xs:string, $value as xs:string) external;
```

```
declare %kv:put(["xml"|"binxml"|"avroxml"]) function
    local:myFunctionName($key as xs:string, $xml as node()) external;
```

Annotations

%kv:put("method")

Declares the NoSQL Database module `put` function. Required.

The *method* determines how the value is stored. It must be one of the following values:

- `text`: `$value` is serialized and encoded using the character set specified by the `%output:encoding` annotation.
- `avroxml`: `$xml` is mapped to an instance of the Avro record specified by the `%avro:schema-kv` annotation. See "Writing XML as Avro" on page 6-14.
- `binxml`: `$xml` is encoded as XDK binary XML
- `xml`: `$xml` is serialized and encoded using the character set specified by the `%output:encoding` annotation. You can specify other XML serialization parameters using `%output:*`.

%avro:schema-kv("schema-name")

Specifies the record schema of the values to be written. The annotation value is a fully qualified record name. The record schema is retrieved from the Oracle NoSQL Database catalog.

For example: `%avro:schema-kv("org.example.PersonRecord")`

%output:*

A standard XQuery serialization parameter for the output method (text or XML) specified in `%kv:put`. See "Serialization Annotations" on page 6-87.

See Also: "The Influence of Serialization Parameters" sections for XML and text output methods in *XSLT and XQuery Serialization 3.0* at

<http://www.w3.org/TR/xslt-xquery-serialization-30/>

Oracle NoSQL Database Adapter Configuration Properties

Oracle XQuery for Hadoop uses the generic options for specifying configuration properties in the Hadoop command. You can use the `-conf` option to identify configuration files, and the `-D` option to specify individual properties. See "Running a Query" on page 5-13.

You can set various configuration properties for the Oracle NoSQL Database adapter that control the durability characteristics and timeout periods. You must set `oracle.kv.hosts` and `oracle.kv.kvstore`.

The following properties configure the Oracle NoSQL Database adapter.

oracle.hadoop.xquery.kv.config.durability

Type: String

Default Value: NO_SYNC, NO_SYNC, SIMPLE_MAJORITY

Description: Defines the durability characteristics associated with `%kv:put` operations. The value consists of three parts, which you specify in order and separate with commas:

MasterPolicy, ReplicaPolicy, ReplicaAck

- *MasterPolicy:* The synchronization policy used when committing a transaction to the master database. Set this part to one of the following constants:
 - NO_SYNC: Do not write or synchronously flush the log on a transaction commit.
 - SYNC: Write and synchronously flush the log on a transaction commit.
 - WRITE_NO_SYNC: Write but do not synchronously flush the log on a transaction commit.
- *ReplicaPolicy:* The synchronization policy used when committing a transaction to the replica databases. Set this part to NO_SYNC, SYNC, or WRITE_NO_SYNC, as described under *MasterPolicy*.
- *ReplicaAck:* The acknowledgment policy used to obtain transaction acknowledgments from the replica databases. Set this part to one of the following constants:
 - ALL: All replicas must acknowledge that they have committed the transaction.
 - NONE: No transaction commit acknowledgments are required, and the master does not wait for them.
 - SIMPLE_MAJORITY: A simple majority of replicas (such as 3 of 5) must acknowledge that they have committed the transaction.

See Also: "Durability Guarantees" in *Getting Started with Oracle NoSQL Database* at

<http://docs.oracle.com/cd/NOSQL/html/GettingStartedGuide/durability.html>

oracle.hadoop.xquery.kv.config.requestLimit

Type: Comma-separated list of integer

Default Value: 100, 90, 80

Description: Limits the number of simultaneous requests to prevent nodes with long service times from consuming all threads in the KV store client. The value consists of three integers, which you specify in order and separate with commas:

maxActiveRequests, requestThresholdPercent, nodeLimitPercent

- *maxActiveRequests:* The maximum number of active requests permitted by the KV client. This number is typically derived from the maximum number of threads that the client has set aside for processing requests.
- *requestThresholdPercent:* The percentage of *maxActiveRequests* at which requests are limited.
- *nodeLimitPercent:* The maximum number of active requests that can be associated with a node when the number of active requests exceeds the threshold specified by *requestThresholdPercent*.

oracle.hadoop.xquery.kv.config.requestTimeout

Type: Long

Default Value: 5000 ms

Description: Configures the request timeout period in milliseconds. The value must be greater than zero (0).

oracle.hadoop.xquery.kv.config.socketOpenTimeout

Type: Long

Default Value: 5000 ms

Description: Configures the open timeout used when establishing sockets for client requests, in milliseconds. Shorter timeouts result in more rapid failure detection and recovery. The default open timeout is adequate for most applications. The value must be greater than zero (0).

oracle.hadoop.xquery.kv.config.socketReadTimeout

Type: Long

Default Value: 30000 ms

Description: Configures the read timeout period associated with the sockets that make client requests, in milliseconds. Shorter timeouts result in more rapid failure detection and recovery. Nonetheless, the timeout period should be sufficient to allow the longest timeout associated with a request.

oracle.kv.kvstore

Type: String

Default Value: Not defined

Description: The name of the KV store with the source data.

oracle.kv.hosts

Type: String

Default Value: Not defined

Description: An array of one or more *hostname:port* pairs that identify the hosts in the KV store with the source data. Separate multiple pairs with commas.

oracle.kv.batchSize

Type: Key

Default Value: Not defined

Description: The desired number of keys for the InputFormat to fetch during each network round trip. A value of zero (0) sets the property to a default value.

oracle.kv.consistency

Type: Consistency

Default Value: NONE_REQUIRED

Description: The consistency guarantee for reading child key-value pairs. The following keywords are valid values:

- **ABSOLUTE:** Requires the master to service the transaction so that consistency is absolute.
- **NONE_REQUIRED:** Allows replicas to service the transaction, regardless of the state of the replicas relative to the master.

oracle.kv.timeout

Type: Long

Default Value: Not defined

Description: Sets a maximum time interval in milliseconds for retrieving a selection of key-value pairs. A value of zero (0) sets the property to its default value.

See Also: *Oracle NoSQL Database Java API Reference* at

<http://docs.oracle.com/cd/NOSQL/html/javadoc/oracle/kv/hadoop/KVInputFormatBase.html>

Sequence File Adapter

The sequence file adapter provides functions to read and write Hadoop sequence files. A sequence file is a Hadoop-specific file format composed of key-value pairs.

The functions are described in the following topics:

- [Built-in Functions for Reading and Writing Sequence Files](#)
- [Custom Functions for Reading Sequence Files](#)
- [Custom Functions for Writing Sequence Files](#)

See Also: The Hadoop wiki for a description of Hadoop sequence files at

<http://wiki.apache.org/hadoop/SequenceFile>

Built-in Functions for Reading and Writing Sequence Files

To use the built-in functions in your query, you must import the sequence file module as follows:

```
import module "oxh:seq";
```

The sequence file module contains the following functions:

- [seq:collection](#)
- [seq:collection-xml](#)
- [seq:collection-binxml](#)
- [seq:put](#)
- [seq:put-xml](#)
- [seq:put-binxml](#)

For examples, see ["Examples of Sequence File Adapter Functions"](#) on page 6-52.

seq:collection

Accesses a collection of sequence files in HDFS and returns the values as strings. The files may be split up and processed in parallel by multiple tasks.

Signature

```
declare %seq:collection("text") function  
    seq:collection($uris as xs:string*) as xs:string* external;
```

Parameters

\$uris

The sequence file URIs. The values in the sequence files must be either `org.apache.hadoop.io.Text` or `org.apache.hadoop.io.BytesWritable`. For `BytesWritable` values, the bytes are converted to a string using a UTF-8 decoder.

Return Value

One string for each value in each file

seq:collection-xml

Accesses a collection of sequence files in HDFS, parses each value as XML, and returns it. Each file may be split up and processed in parallel by multiple tasks.

Signature

```
declare %seq:collection("xml") function  
    seq:collection-xml($uris as xs:string*) as document-node()* external;
```

Parameters

\$uris

The sequence file URIs. The values in the sequence files must be either `org.apache.hadoop.io.Text` or `org.apache.hadoop.io.BytesWritable`. For

BytesWritable values, the XML document encoding declaration is used, if it is available.

Returns

One XML document for each value in each file.

seq:collection-binxml

Accesses a collection of sequence files in the HDFS, reads each value as binary XML, and returns it. Each file may be split up and processed in parallel by multiple tasks.

Signature

```
declare %seq:collection("binxml") function
  seq:collection-binxml($uris as xs:string*) as document-node()* external;
```

Parameters

\$uris

The sequence file URIs. The values in the sequence files must be `org.apache.hadoop.io.BytesWritable`. The bytes are decoded as binary XML.

Returns

One XML document for each value in each file

Notes

You can use this function to read files that were created by `seq:put-binxml` in a previous query. See "[seq:put-binxml](#)" on page 6-51.

See Also

Oracle XML Developer's Kit Programmer's Guide

seq:put

Writes the string value of a key-value pair to a sequence file in the output directory of the query. The values are spread across one or more sequence files.

This function writes the values as `org.apache.hadoop.io.Text`, and sets the key class to `org.apache.hadoop.io.NullWritable` because there are no key values.

Signature

```
declare %seq:put("text") function
  seq:put($value as xs:string) external;
```

Parameters

\$value

The string to write

Returns

`empty-sequence()`

Notes

The number of files created depends on how the query is distributed among tasks. Each file has a name that starts with "part," such as part-m-00000. You specify the output directory when the query executes. See ["Running a Query"](#) on page 5-13.

seq:put

Writes a key and string value to a sequence file in the output directory of the query. The values are spread across one or more sequence files.

This function writes the keys and values as `org.apache.hadoop.io.Text`.

Signature

```
declare %seq:put("text") function
    seq:put($key as xs:string, $value as xs:string) external;
```

Parameters

\$key

The key of a key-value pair

\$value

The value of the key-value pair

Returns

`empty-sequence()`

seq:put-xml

Writes an XML value to a sequence file in the output directory of the query. The values are spread across one or more sequence files.

This function writes the values as `org.apache.hadoop.io.Text`, and sets the key class to `org.apache.hadoop.io.NullWritable` because there are no key values.

Signature

```
declare %seq:put("xml") function
    seq:put-xml($xml as node()) external;
```

Parameters

\$value

The XML to write

Returns

`empty-sequence()`

Notes

The number of files created depends on how the query is distributed among tasks. Each file has a name that starts with "part," such as part-m-00000. You specify the output directory when the query executes. See ["Running a Query"](#) on page 5-13.

seq:put-xml

Writes a key and XML value to a sequence file in the output directory of the query. The values are spread across one or more sequence files.

This function writes the keys and values as `org.apache.hadoop.io.Text`.

Signature

```
declare %seq:put("xml") function
  seq:put-xml($key as xs:string, $xml as node()) external;
```

Parameters**\$key**

The key of a key-value pair

\$value

The value of a key-value pair

Returns

`empty-sequence()`

Notes

The number of files created depends on how the query is distributed among tasks. Each file has a name that starts with "part," such as part-m-00000. You specify the output directory when the query executes. See ["Running a Query"](#) on page 5-13.

seq:put-binxml

Encodes an XML value as binary XML and writes the resulting bytes to a sequence file in the output directory of the query. The values are spread across one or more sequence files.

This function writes the values as `org.apache.hadoop.io.BytesWritable`, and sets the key class to `org.apache.hadoop.io.NullWritable` because there are no key values.

Signature

```
declare %seq:put("binxml") function
  seq:put-binxml($xml as node()) external;
```

Parameters**\$value**

The XML to write

Return Value

`empty-sequence()`

Notes

The number of files created depends on how the query is distributed among tasks. Each file has a name that starts with "part," such as part-m-00000. You specify the output directory when the query executes. See ["Running a Query"](#) on page 5-13.

You can use the `seq:collection-binxml` function to read the files created by this function. See "[seq:collection-binxml](#)" on page 6-49.

See Also

Oracle XML Developer's Kit Programmer's Guide

seq:put-binxml

Encodes an XML value as binary XML and writes the resulting bytes to a sequence file in the output directory of the query. The values are spread across one or more sequence files.

This function writes the keys as `org.apache.hadoop.io.Text` and the values as `org.apache.hadoop.io.BytesWritable`.

Signature

```
declare %seq:put("binxml") function
  seq:put-binxml($key as xs:string, $xml as node()) external;
```

Parameters

\$key

The key of a key-value pair

\$value

The value to write

Returns

`empty-sequence()`

Notes

The number of files created depends on how the query is distributed among tasks. Each file has a name that starts with "part," such as `part-m-00000`. You specify the output directory when the query executes. See "[Running a Query](#)" on page 5-13.

You can use the `seq:collection-binxml` function to read the files created by this function. See "[seq:collection-binxml](#)" on page 6-49.

Examples of Sequence File Adapter Functions

This example queries three XML files in HDFS with the following contents. Each XML file contains comments made by users on a specific day. Each comment can have zero or more "likes" from other users.

`mydata/comments1.xml`

```
<comments date="2013-12-30">
  <comment id="12345" user="john" text="It is raining :( "/>
  <comment id="56789" user="kelly" text="I won the lottery!">
    <like user="john"/>
    <like user="mike"/>
  </comment>
</comments>
```

`mydata/comments2.xml`

```
<comments date="2013-12-31">
  <comment id="54321" user="mike" text="Happy New Year!">
    <like user="laura"/>
  </comment>
</comments>
```

mydata/comments3.xml

```
<comments date="2014-01-01">
  <comment id="87654" user="mike" text="I don't feel so good."/>
  <comment id="23456" user="john" text="What a beautiful day!">
    <like user="kelly"/>
    <like user="phil"/>
  </comment>
</comments>
```

Example 1

The following query stores the comment elements in sequence files.

```
import module "oxh:seq";
import module "oxh:xmlf";

for $comment in xmlf:collection("mydata/comments*.xml", "comment")
return
  seq:put-xml($comment)
```

Example 2

The next query reads the sequence files generated by the previous query, which are stored in an output directory named myoutput. The query then writes the names of users who made multiple comments to a text file.

```
import module "oxh:seq";
import module "oxh:text";

for $comment in seq:collection-xml("myoutput/part*")/comment
let $user := $comment/@user
group by $user
let $count := count($comment)
where $count gt 1
return
  text:put($user || " " || $count)
```

The text file created by the previous query contain the following lines:

```
john 2
mike 2
```

See ["XML File Adapter"](#) on page 6-68.

Custom Functions for Reading Sequence Files

You can use the following annotations to define functions that read collections of sequence files. These annotations provide additional functionality that is not available using the built-in functions.

Signature

Custom functions for reading sequence files must have one of the following signatures:

```
declare %seq:collection("text") [additional annotations]
    function local:myFunctionName($uris as xs:string*) as xs:string* external;

declare %seq:collection(["xml"|"binxml"]) [additional annotations]
    function local:myFunctionName($uris as xs:string*) as document-node()*
external;
```

Annotations

%seq:collection(["method"])

Declares the sequence file collection function, which reads sequence files. Required.

The optional *method* parameter can be one of the following values:

- **text**: The values in the sequence files must be either `org.apache.hadoop.io.Text` or `org.apache.hadoop.io.BytesWritable`¹. They are returned as `xs:string`. Default.
- **xml**: The values in the sequence files must be either `org.apache.hadoop.io.Text` or `org.apache.hadoop.io.BytesWritable`. The values are parsed as XML and returned by the function.
- **binxml**: The values in the sequence files must be `org.apache.hadoop.io.BytesWritable`. The values are read as XDK binary XML and returned by the function. See the *Oracle XML Developer's Kit Programmer's Guide*.

%output:encoding("charset")

Specifies the character encoding of the input files. The valid encodings are those supported by the JDK. UTF-8 is the default encoding.

See Also: "Supported Encodings" in the Oracle Java SE documentation at

<http://docs.oracle.com/javase/7/docs/technotes/guides/intl/encoding.doc.html>

%seq:key("true" | "false")

Controls whether the key of a key-value pair is set as the document-uri of the returned value. Specify `true` to return the keys. The default setting is `true` when *method* is `binxml` or `xml`, and `false` when it is `text`.

Text functions with this annotation set to `true` must return `text()*` instead of `xs:string*` because atomic `xs:string` is not associated with a document.

¹ Bytes are decoded using the character set specified by the `%output:encoding` annotation.

When the keys are returned, you can obtain their string representations by using `seq:key` function.

This example returns text instead of string values because `%seq:key` is set to `true`.

```
declare %seq:collection("text") %seq:key("true")
function local:col($uris as xs:string*) as text()* external;
```

The next example uses the `seq:key` function to obtain the string representations of the keys:

```
for $value in local:col(...)
let $key := $value/seq:key()
return
.
.
.
```

%seq:split-max("split-size")

Specifies the maximum split size as either an integer or a string value. The split size controls how the input file is divided into tasks. Hadoop calculates the split size as `max($split-min, min($split-max, $block-size))`. Optional.

In a string value, you can append `K`, `k`, `M`, `m`, `G`, or `g` to the value to indicate kilobytes, megabytes, or gigabytes instead of bytes (the default unit). These qualifiers are not case sensitive. The following examples are equivalent:

```
%seq:split-max(1024)
%seq:split-max("1024")
%seq:split-max("1K")
```

%seq:split-min("split-size")

Specifies the minimum split size as either an integer or a string value. The split size controls how the input file is divided into tasks. Hadoop calculates the split size as `max($split-min, min($split-max, $block-size))`. Optional.

In a string value, you can append `K`, `k`, `M`, `m`, `G`, or `g` to the value to indicate kilobytes, megabytes, or gigabytes instead of bytes (the default unit). These qualifiers are not case sensitive. The following examples are equivalent:

```
%seq:split-min(1024)
%seq:split-min("1024")
%seq:split-min("1K")
```

Custom Functions for Writing Sequence Files

You can use the following annotations to define functions that write collections of sequence files in HDFS.

Signature

Custom functions for writing sequence files must have one of the following signatures. You can omit the `$key` argument when you are not writing a key value.

```
declare %seq:put("text") [additional annotations]
    function local:myFunctionName($key as xs:string, $value as xs:string) external;
```

```
declare %seq:put(["xml"|"binxml"]) [additional annotations]
    function local:myFunctionName($key as xs:string, $xml as node()) external;
```

Annotations

%seq:put("method")

Declares the sequence file put function, which writes key-value pairs to a sequence file. Required.

If you use the `$key` argument in the signature, then the key is written as `org.apache.hadoop.io.Text`. If you omit the `$key` argument, then the key class is set to `org.apache.hadoop.io.NullWritable`.

Set the `method` parameter to `text`, `xml`, or `binxml`. The `method` determines the type used to write the value:

- `text`: String written as `org.apache.hadoop.io.Text`
- `xml`: XML written as `org.apache.hadoop.io.Text`
- `binxml`: XML encoded as XDK binary XML and written as `org.apache.hadoop.io.BytesWritable`

%seq:compress("codec", "compressionType")

Specifies the compression format used on the output. The default is no compression. Optional.

The `codec` parameter identifies a compression codec. The first registered compression codec that matches the value is used. The value matches a codec if it equals one of the following:

1. The fully qualified class name of the codec
2. The unqualified class name of the codec
3. The prefix of the unqualified class name before `Codec` (case insensitive)

Set the `compressionType` parameter to one of these values:

- `block`: Keys and values are collected in groups and compressed together. Block compression is generally more compact, because the compression algorithm can take advantage of similarities among different values.
- `record`: Only the values in the sequence file are compressed.

All of these examples use the default codec and block compression:

```
%seq:compress("org.apache.hadoop.io.compress.DefaultCodec", "block")
%seq:compress("DefaultCodec", "block")
%seq:compress("default", "block")
```

%seq:file("name")

Specifies the output file name prefix. The default prefix is part.

%output:parameter

A standard XQuery serialization parameter for the output method (text or XML) specified in %seq:put. See "[Serialization Annotations](#)" on page 6-87.

See Also:

The Hadoop Wiki SequenceFile topic at

<http://wiki.apache.org/hadoop/SequenceFile>

"The Influence of Serialization Parameters" sections for XML and text output methods in *XSLT and XQuery Serialization 3.0* at

<http://www.w3.org/TR/xslt-xquery-serialization-30/>

Examples of Sequence File Adapter Functions

The following query extracts comment elements from XML files and stores them in compressed sequence files. Before storing each comment, it deletes the id attribute and uses the value as the key in the sequence files.

```
import module "oxh:xmlf";

declare
  %seq:put("xml")
  %seq:compress("default", "block")
  %seq:file("comments")
function local:myPut($key as xs:string, $value as node()) external;

for $comment in xmlf:collection("mydata/comments*.xml", "comment")
let $id := $comment/@id
let $newComment :=
  copy $c := $comment
  modify delete node $c/@id
  return $c
return
  local:myPut($id, $newComment)
```

The next query reads the sequence files that the previous query created in an output directory named myoutput. The query automatically decompresses the sequence files.

```
import module "oxh:text";
import module "oxh:seq";

for $comment in seq:collection-xml("myoutput/comments*")/comment
let $id := $comment/seq:key()
where $id eq "12345"
return
  text:put-xml($comment)
```

The previous query creates a text file that contains the following line:

```
<comment id="12345" user="john" text="It is raining :( "/>
```

Text File Adapter

The text file adapter provides functions to read and write text files stored in HDFS. It is described in the following topics:

- [Built-in Functions for Reading and Writing Text Files](#)
- [Custom Functions for Reading Text Files](#)
- [Custom Functions for Writing Text Files](#)
- [Examples of Text File Functions](#)

Built-in Functions for Reading and Writing Text Files

To use the built-in functions in your query, you must import the text file module as follows:

```
import module "oxh:text";
```

The text file module contains the following functions:

- [text:collection](#)
- [text:collection-xml](#)
- [text:put](#)
- [text:put-xml](#)
- [text:trace](#)

For examples, see "[Examples of Text File Adapter Functions](#)" on page 6-61.

text:collection

Accesses a collection of text files in HDFS. The files might be split up and processed in parallel by multiple tasks.

Signature

```
declare %text:collection("text") function
  text:collection($uris as xs:string*) as xs:string* external;
```

Parameters

\$uris

The text file URIs

Returns

One string value for each line in each file

text:collection-xml

Accesses a collection of text files in HDFS. The files might be split up and processed in parallel by multiple tasks. Each line in each file is parsed as an XML document and returned by the function. Therefore, each line must fully contain a single XML document, and any new lines in the XML must be escaped with XML character references.

Signature

```
declare %text:collection("xml") function
  text:collection-xml($uris as xs:string*) as document-node()* external;
```

Parameters

\$uris

The text file URIs

Returns

One XML document for each line in each file

text:put

Writes a line to a text file in the output directory of the query. The lines are spread across one or more files.

Signature

```
declare %text:put("text") function
    text:put($value as xs:string) external;
```

Parameters

\$value

The text to write

Returns

`empty-sequence()`

Notes

The number of files created depends on how the query is distributed among tasks. Each file has a name that starts with "part" (such as part-m-00000). You specify the output directory when the query executes. See ["Running a Query"](#) on page 5-13.

text:put-xml

Writes XML to a line in a text file. The lines are spread across one or more files in the output directory of the query.

Newline characters in the serialized XML are replaced with character references to ensure the XML does not span multiple lines. For example, `
` replaces the linefeed character (`\n`).

Signature

```
declare %text:put("xml") function
    text:put-xml($value as node()) external;
```

Parameters

\$value

The XML to write

Returns

`empty-sequence()`

Notes

The number of files created depends on how the query is distributed among tasks. Each file has a name that starts with "part" (such as part-m-00000). You specify the output directory when the query executes. See ["Running a Query"](#) on page 5-13.

text:trace

Writes a line to a text file named trace-* in the output directory of the query. The lines are spread across one or more files.

This function provides you with a quick way to write to an alternate output. For example, you might create a trace file to identify invalid rows within a query, while loading the data into an Oracle database table.

Signature

```
declare %text:put("text") %text:file("trace") function
    text:trace($value as xs:string) external;
```

Parameters**\$value**

The text to write

Returns

empty-sequence()

Examples of Text File Adapter Functions

This example uses following text files in HDFS. The files contain a log of visits to different web pages. Each line represents a visit to a web page and contains the time, user name, and page visited.

mydata/visits1.log

```
2013-10-28T06:00:00, john, index.html, 200
2013-10-28T08:30:02, kelly, index.html, 200
2013-10-28T08:32:50, kelly, about.html, 200
2013-10-30T10:00:10, mike, index.html, 401
```

mydata/visits2.log

```
2013-10-30T10:00:01, john, index.html, 200
2013-10-30T10:05:20, john, about.html, 200
2013-11-01T08:00:08, laura, index.html, 200
2013-11-04T06:12:51, kelly, index.html, 200
2013-11-04T06:12:40, kelly, contact.html, 200
```

The following query filters out the pages visited by john and writes only the date and page visited to a new text file:

```
import module "oxh:text";

for $line in text:collection("mydata/visits*.log")
let $split := fn:tokenize($line, "\s*\s*")
where $split[2] eq "john"
return
    text:put($split[1] || " " || $split[3])
```

The query creates a text file that contains the following lines:

```
2013-10-28T06:00:00 index.html
2013-10-30T10:00:01 index.html
2013-10-30T10:05:20 about.html
```

The next query computes the number of page visits per day:

```
import module "oxh:text";

for $line in text:collection("mydata/visits*.log")
let $split := fn:tokenize($line, "\s*\s*")
let $time := xs:dateTime($split[1])
let $day := xs:date($time)
group by $day
return
  text:put($day || " => " || count($line))
```

The query creates text files that contain the following lines:

```
2013-10-28 => 3
2013-10-30 => 3
2013-11-01 => 1
2013-11-04 => 2
```

Custom Functions for Reading Text Files

You can use the following annotations to define functions that read collections of text files in HDFS. These annotations provide additional functionality that is not available using the built-in functions.

Signature

Custom functions for reading text files must have one of the following signatures:

```
declare %text:collection("text") [additional annotations]
    function local:myFunctionName($uris as xs:string*) as xs:string* external;

declare %text:collection("xml") [additional annotations]
    function local:myFunctionName($uris as xs:string*) as document-node()*
external;
```

Annotations

%text:collection(["*method*"])

Declares the `text` collection function. Required.

The optional *method* parameter can be one of the following values:

- `text`: Each line in the text file is returned as `xs:string`. Default.
- `xml`: Each line in the text file is parsed as XML and returned as `document-node`. Each XML document must be fully contained on a single line. Newline characters inside the document must be represented by a numeric character reference.

%xmlf:split-max(" *split-size* ")

Specifies the maximum split size as either an integer or a string value. The split size controls how the input file is divided into tasks. Hadoop calculates the split size as `max($split-min, min($split-max, $block-size))`. Optional.

In a string value, you can append `K`, `k`, `M`, `m`, `G`, or `g` to the value to indicate kilobytes, megabytes, or gigabytes instead of bytes (the default unit). These qualifiers are not case sensitive. The following examples are equivalent:

```
%xmlf:split-max(1024)
%xmlf:split-max("1024")
%xmlf:split-max("1K")
```

%xmlf:split-min(" *split-size* ")

Specifies the minimum split size as either an integer or a string value. The split size controls how the input file is divided into tasks. Hadoop calculates the split size as `max($split-min, min($split-max, $block-size))`. Optional.

In a string value, you can append `K`, `k`, `M`, `m`, `G`, or `g` to the value to indicate kilobytes, megabytes, or gigabytes instead of bytes (the default unit). These qualifiers are not case sensitive. The following examples are equivalent:

```
%xmlf:split-min(1024)
%seq:split-min("1024")
%xmlf:split-min("1K")
```

Parameters

\$uris as xs:string*

Lists the HDFS file URIs. Required.

Return Type

`xs:string*` for the `text` method if `%seq:key=false` or is not set

`text()*` if `%seq:key=true` or is set

Custom Functions for Writing Text Files

You can use the following annotations to define functions that write text files in HDFS.

Signature

Custom functions for writing text files must have one of the following signatures:

```
declare %text:put("text") [additional annotations] function
  text:myFunctionName($value as xs:string) external;
```

```
declare %text:put("xml") [additional annotations] function
  text:myFunctionName($value as node()) external;
```

Annotations

%text:put(["*method*"])

Declares the `text` put function. Required.

The optional *method* parameter can be one of the following values:

- `text`: Writes data to a text file. Default.
- `xml`: Writes data to an XML file. The XML is serialized and newline characters are replaced with character references. This assures that the resulting XML document is one text line with no line breaks.

%text:compress("codecs")

Specifies the compression format used on the output. The default is no compression. Optional.

The *codecs* parameter identifies a compression codec. The first registered compression codec that matches the value is used. The value matches a codec if it equals one of the following:

1. The fully qualified class name of the codec
2. The unqualified class name of the codec
3. The prefix of the unqualified class name before "Codec" (case insensitive)

All of these examples use the default codec and block compression:

```
%text:compress("org.apache.hadoop.io.compress.DefaultCodec", "block")
%text:compress("DefaultCodec", "block")
%text:compress("default", "block")
```

%text:file("name")

Specifies the output file name prefix. The default prefix is `part`.

%output:parameter

A standard XQuery serialization parameter for the output method (text or XML) specified in `%text:put`. See ["Serialization Annotations"](#) on page 6-87.

UTF-8 is currently the only supported character encoding.

Examples of Text File Functions

Example 1 Querying Simple Delimited Formats

This example uses the `fn:tokenize` function to parse the lines of a text file. This technique works well for simple delimited formats.

The following query declares custom `put` and `collection` functions. It computes the number of hits and the number of unique users for each page in the logs.

```
import module "oxh:text";

declare
  %text:collection("text")
  %text:split-max("32m")
function local:col($uris as xs:string*) as xs:string* external;

declare
  %text:put("xml")
  %text:compress("gzip")
  %text:file("pages")
function local:out($arg as node()) external;

for $line in local:col("mydata/visits*.log")
let $split := fn:tokenize($line, "\s*,\s*")
let $user := $split[2]
let $page := $split[3]
group by $page
return
  local:out(
    <page>
      <name>{$page}</name>
      <hits>{count($line)}</hits>
      <users>{fn:count(fn:distinct-values($user))}</users>
    </page>
  )
```

The output directory of the previous query is named `myoutput`. The following lines are written to `myoutput/pages-r-*.gz`.

```
<page><name>about.html</name><hits>2</hits><users>2</users></page>
<page><name>contact.html</name><hits>1</hits><users>1</users></page>
<page><name>index.html</name><hits>6</hits><users>4</users></page>
```

The files are compressed with the `gzip` codec. The following query reads the output files, and writes the page name and total hits as plain text. The collection function automatically decodes the compressed files.

```
import module "oxh:text";

for $page in text:collection-xml("myoutput/page*.gz")/page
return
  text:put($page/name || ", " || $page/hits)
```

This query creates text files that contains the following lines:

```
about.html,2
contact.html,1
index.html,6
```

Example 2 Querying Complex Text Formats

The `fn:tokenize` function might not be adequate for complex formats that contain variety of data types and delimiters. This example uses the `fn:analyze-string` function to process a log file in the Apache Common Log format.

A text file named `mydata/access.log` in HDFS contains the following lines:

```
192.0.2.0 - - [30/Sep/2013:16:39:38 +0000] "GET /inddex.html HTTP/1.1" 404 284
192.0.2.0 - - [30/Sep/2013:16:40:54 +0000] "GET /index.html HTTP/1.1" 200 12390
192.0.2.4 - - [01/Oct/2013:12:10:54 +0000] "GET /index.html HTTP/1.1" 200 12390
192.0.2.4 - - [01/Oct/2013:12:12:12 +0000] "GET /about.html HTTP/1.1" 200 4567
192.0.2.1 - - [02/Oct/2013:08:39:38 +0000] "GET /indexx.html HTTP/1.1" 404 284
192.0.2.1 - - [02/Oct/2013:08:40:54 +0000] "GET /index.html HTTP/1.1" 200 12390
192.0.2.1 - - [02/Oct/2013:08:42:38 +0000] "GET /aobut.html HTTP/1.1" 404 283
```

The following query computes the requests made after September 2013 when the server returned a status code 404 (Not Found) error. It uses a regular expression and `fn:analyze-string` to match the components of the log entries. The time format cannot be cast directly to `xs:dateTime`, as shown in [Example 1](#). Instead, the `ora-fn:dateTime-from-string-with-format` function converts the string to an instance of `xs:dateTime`.

```
import module "oxh:text";

declare variable $REGEX :=
  '(\S+) (\S+) (\S+) \[[^\]]+\] \"([^\"]+)\" (\S+) (\S+)';

for $line in text:collection("mydata/access.log")
let $match := fn:analyze-string($line, $REGEX)/fn:match
let $time :=
  ora-fn:dateTime-from-string-with-format(
    "dd/MMM/yyyy:HH:mm:ss Z",
    $match/fn:group[4]
  )
let $status := $match/fn:group[6]
where
  $status eq "404" and
  $time ge xs:dateTime("2013-10-01T00:00:00")
let $host := $match/fn:group[1]
let $request := $match/fn:group[5]
return
  text:put($host || ", " || $request)
```

The query creates text files that contain the following lines:

```
192.0.2.1,GET /indexx.html HTTP/1.1
192.0.2.1,GET /aobut.html HTTP/1.1
```

See Also:

- The W3C *XPath and XQuery Functions and Operators 3.0* specification for information about the `fn:tokenize` and `fn:analyze-string` functions:

<http://www.w3.org/TR/xpath-functions-30/#func-tokenize>

<http://www.w3.org/TR/xpath-functions-30/#func-analyze-string>

- For information about the Apache Common log format:

<http://httpd.apache.org/docs/current/logs.html>

XML File Adapter

The XML file adapter provides access to XML files stored in HDFS. The adapter optionally splits individual XML files so that a single file can be processed in parallel by multiple tasks.

This adapter is described in the following topics:

- [Built-in Functions for Reading XML Files](#)
- [Custom Functions for Reading XML Files](#)

Built-in Functions for Reading XML Files

To use the built-in functions in your query, you must import the XML file module as follows:

```
import module "oxh:xmlf";
```

The XML file module contains the following functions:

- [xmlf:collection](#)

See "[Examples of XML File Adapter Functions](#)" on page 6-70.

xmlf:collection

Accesses a collection of XML documents in HDFS. Multiple files can be processed concurrently, but each individual file is parsed by a single task.

Note: HDFS does not perform well when data is stored in many small files. For large data sets with many small XML documents, use Hadoop sequence files and the [Sequence File Adapter](#).

Signature

```
declare %xmlf:collection function
  xmlf:collection($uris as xs:string*) as document-node()* external;
```

Parameters

\$uris
The XML file URIs

Returns

One XML document for each file

xmlf:collection

Accesses a collection of XML documents in HDFS. The files might be split and processed by multiple tasks simultaneously. The function returns only elements that match a specified name. This enables very large XML files to be processed efficiently.

This function only supports XML files that meet certain requirements. See "[Restrictions on Splitting XML Files](#)" on page 6-73.

Signature

```
declare %xmlf:collection function
  xmlf:collection($uris as xs:string*, $names as xs:anyAtomicType+) as element()*
  external;
```

Parameters

\$uris
The XML file URIs

\$names

The names of the elements to be returned by the function. The names can be either strings or QNames. For QNames, the XML parser uses the namespace binding implied by the QName prefix and namespace.

Returns

Each element that matches one of the names specified by the \$names argument

Examples of XML File Adapter Functions

This example queries three XML files in HDFS with the following contents. Each XML file contains comments made by users on a specific day. Each comment can have zero or more "likes" from other users.

```
mydata/comments1.xml
```

```
<comments date="2013-12-30">
  <comment id="12345" user="john" text="It is raining :( "/>
  <comment id="56789" user="kelly" text="I won the lottery!">
    <like user="john"/>
    <like user="mike"/>
  </comment>
</comments>
```

```
mydata/comments2.xml
```

```
<comments date="2013-12-31">
  <comment id="54321" user="mike" text="Happy New Year!">
    <like user="laura"/>
  </comment>
</comments>
```

```
mydata/comments3.xml
```

```
<comments date="2014-01-01">
  <comment id="87654" user="mike" text="I don't feel so good.">
  <comment id="23456" user="john" text="What a beautiful day!">
    <like user="kelly"/>
    <like user="phil"/>
  </comment>
</comments>
```

This query writes the number of comments made each year to a text file. No element names are passed to `xmlf:collection`, and so it returns three documents, one for each file. Each file is processed serially by a single task.

```
import module "oxh:xmlf";
import module "oxh:text";

for $comments in xmlf:collection("mydata/comments*.xml")/comments
let $date := xs:date($comments/@date)
group by $year := fn:year-from-date($date)
return
  text:put($year || ", " || fn:count($comments/comment))
```

The query creates text files that contain the following lines:

```
2013, 3
2014, 2
```

The next example writes the number of comments and the average number of likes for each user. Each input file is split, so that it can be processed in parallel by multiple tasks. The `xmlf:collection` function returns five elements, one for each comment.

```
import module "oxh:xmlf";
import module "oxh:text";

for $comment in xmlf:collection("mydata/comments*.xml", "comment")
let $likeCt := fn:count($comment/like)
group by $user := $comment/@user
return
  text:put($user || ", " || fn:count($comment) || ", " || fn:avg($likeCt))
```

This query creates text files that contain the following lines:

```
john, 2, 1
kelly, 1, 2
mike, 2, 0.5
```

Custom Functions for Reading XML Files

You can use the following annotations to define functions that read collections of XML files in HDFS. These annotations provide additional functionality that is not available using the built-in functions.

Signature

Custom functions for reading XML files must have one of the following signatures:

```
declare %xmlf:collection [additional annotations]
    function local:myFunctionName($uris as xs:string*) as node()* external;

declare %xmlf:collection [additional annotations]
    function local:myFunctionName($uris as xs:string*, $names as xs:anyAtomicType+
as element()* external;
```

Annotations

%xmlf:collection

Declares the collection function. This annotation does not accept parameters. Required.

%xmlf:split("element-name1" [... "element-nameN"])

Specifies the element names used for parallel XML parsing. This annotation can be used instead of the \$names argument.

When this annotation is specified, only the single argument version of the function is allowed. This enables the element names to be specified statically, so they do not need to be specified when the function is called.

%output:encoding("charset")

Identifies the text encoding of the input documents.

When this encoding is used with the %xmlf:split annotation or the \$names argument, only ISO-8859-1, US-ASCII, and UTF-8 are valid encodings. Otherwise, the valid encodings are those supported by the JDK. UTF-8 is assumed when this annotation is omitted.

See Also: "Supported Encodings" in the Oracle Java SE documentation at

<http://docs.oracle.com/javase/7/docs/technotes/guides/intl/encoding.doc.html>

%xmlf:split-namespace("prefix", "namespace")

This annotation provides extra namespace declarations to the parser. You can specify it multiple times to declare one or more namespaces.

Use this annotation to declare the namespaces of ancestor elements. When XML is processed in parallel, only elements that match the specified names are processed by an XML parser. If a matching element depends on the namespace declaration of one of its ancestor elements, then the declaration is not visible to the parser and an error may occur.

These namespace declarations can also be used in element names when specifying the split names. For example:

```
declare
    %xmlf:collection
```

```
%xmlf:split("eg:foo")
%xmlf:split-namespace("eg", "http://example.org")
function local:myFunction($uris as xs:string*) as document-node() external;
```

%xmlf:split-entity("entity-name", "entity-value")

Provides entity definitions to the XML parser. When XML is processed in parallel, only elements that match the specified split names are processed by an XML parser. The DTD of an input document that is split and processed in parallel is not processed.

In this example, the XML parser expands `&foo;` entity references as "Hello World":

```
%xmlf:split-entity("foo", "Hello World")
```

%xmlf:split-max("split-size")

Specifies the maximum split size as either an integer or a string value. The split size controls how the input file is divided into tasks. Hadoop calculates the split size as `max($split-min, min($split-max, $block-size))`. Optional.

In a string value, you can append `K`, `k`, `M`, `m`, `G`, or `g` to the value to indicate kilobytes, megabytes, or gigabytes instead of bytes (the default unit). These qualifiers are not case sensitive. The following examples are equivalent:

```
%xmlf:split-max(1024)
%xmlf:split-max("1024")
%xmlf:split-max("1K")
```

%seq:split-min("split-size")

Specifies the minimum split size as either an integer or a string value. The split size controls how the input file is divided into tasks. Hadoop calculates the split size as `max($split-min, min($split-max, $block-size))`. Optional.

In a string value, you can append `K`, `k`, `M`, `m`, `G`, or `g` to the value to indicate kilobytes, megabytes, or gigabytes instead of bytes (the default unit). These qualifiers are not case sensitive. The following examples are equivalent:

```
%xmlf:split-min(1024)
%xmlf:split-min("1024")
%xmlf:split-min("1K")
```

Notes

Restrictions on Splitting XML Files

Individual XML documents can be processed in parallel when the element names are specified using either the `$names` argument or the `%xmlf:split` annotation.

The input documents must meet the following constraints in order to be processed in parallel:

- XML cannot contain a comment, CDATA section, or processing instruction that contains text that matches one of the specified element names (that is, a `<` character followed by a name that expands to a QName). Otherwise, such content might be parsed incorrectly as an element.
- An element in the file that matches a specified element name cannot contain a descendant element that also matches a specified name. Otherwise, multiple processors might pick up the matching descendant and cause the function to produce incorrect results.
- An element that matches one of the specified element names (and all of its descendants) must not depend on the namespace declarations of any of its

ancestors. Because the ancestors of a matching element are not parsed, the namespace declarations in these elements are not processed.

You can work around this limitation by manually specifying the namespace declarations with the `%xmlf:split-namespace` annotation.

Oracle recommends that the specified element names do not match elements in the file that are bigger than the split size. If they do, then the adapter functions correctly but not efficiently.

Processing XML in parallel is difficult, because parsing cannot begin in the middle of an XML file. XML constructs like CDATA sections, comments, and namespace declarations impose this limitation. A parser starting in the middle of an XML document cannot assume that, for example, the string `<foo>` is a begin element tag, without searching backwards to the beginning of the document to ensure that it is not in a CDATA section or a comment. However, large XML documents typically contain sequences of similarly structured elements and thus are amenable to parallel processing. If you specify the element names, then each task works by scanning a portion of the document for elements that match one of the specified names. Only elements that match a specified name are given to a true XML parser. Thus, the parallel processor does not perform a true parse of the entire document.

Example

Example 1 Querying XML Files

The following example declares a custom function to access XML files:

```
import module "oxh:text";

declare
  %xmlf:collection
  %xmlf:split("comment")
  %xmlf:split-max("32M")
function local:comments($uris as xs:string*) as element()* external;

for $c in local:comments("mydata/comment*.xml")
where $c/@user eq "mike"
return text:put($c/@id)
```

The query creates a text file that contains the following lines:

```
54321
87654
```

JSON Module

This module contains functions for working with JSON data. You can use it to process JSON that is embedded in other file formats. For example, you can query JSON that is stored as lines in a large text file by using `json:parse-as-xml` with the `text:collection` function.

Processing large JSON files in parallel is not currently supported.

Built-in Functions for Reading JSON

To use the built-in functions in your query, you must import the JSON module as follows:

```
import module "oxh:json";
```

The JSON module contains the following functions:

- [json:parse-as-xml](#)
- [json:get](#)

json:parse-as-xml

Parses a JSON value as XML.

Signature

```
json:parse-as-xml($arg as xs:string?) as element(*)?
```

Parameters

\$arg

Can be the empty sequence.

Returns

An XML element that models the JSON value. An empty sequence if \$arg is an empty sequence.

Notes

About Converting JSON Objects to XML

JSON objects are similar to Avro maps and are converted to the same XML structure. See ["Reading Maps"](#) on page 6-11.

For example, the following JSON object is converted to an XML element:

```
{
  "user" : "john",
  "full_name" : "John Doe",
  "age" : 45
}
```

The object is modeled as the following element:

```
<oxh:item>
  <oxh:entry key="user">john</oxh:entry>
  <oxh:entry key="full_name">John Doe</oxh:entry>
  <oxh:entry key="age">45</oxh:entry>
</oxh:item>
```

About Converting JSON Arrays to XML

JSON arrays are similar to Avro arrays and are converted to the same XML structure. See ["Reading Arrays"](#) on page 6-12.

For example, the following JSON array is converted to an XML element:

```
[ "red", "blue", "green" ]
```

The array is modeled as the following element:

```
<oxh:item>
  <oxh:item>red</oxh:item>
  <oxh:item>blue</oxh:item>
  <oxh:item>green</oxh:item>
</oxh:item>
```

About Converting Other JSON Types

The other JSON values are mapped as follows:

Table 6–4 JSON Type Conversions

JSON	XML
null	An empty (nilled) element
true/false	xs:boolean
number	xs:decimal
string	xs:string

json:get

Retrieves an entry from a JSON object modeled as XML.

Signature

```
json:get($key as xs:string?, $obj as node()?) as element(oxh:entry)?
```

Or

```
json:get($key as xs:string?) as element(oxh:entry)?
```

Parameters

\$key

The JSON data key.

\$obj

The JSON object value.

Returns

The value of the following XPath expression:

```
$obj/oxh:entry[@key eq $key]
```

If \$input not present, the behavior is identical to calling the two-argument function using the context item for \$obj. See the Notes.

Notes

The following are equivalent:

```
$var/json:get("key")
```

```
json:get("key", $var)
```

```
$var/oxh:entry[@key eq "key"]
```

\$var is a JSON object modeled as XML. See ["Reading Maps"](#) on page 6-11.

Examples of JSON Functions

These examples query the following text file in HDFS.

```
mydata/users-json.txt
```

```
{ "user" : "john", "full name" : "John Doe", "age" : 45 }
{ "user" : "kelly", "full name" : "Kelly Johnson", "age" : 32 }
{ "user" : "laura", "full name" : "Laura Smith", "age" : null }
{ "user" : "phil", "full name" : "Phil Johnson", "age" : 27 }
```

Example 1

The following query selects the names of users that are older than 30:

```
import module "oxh:text";
import module "oxh:json";

for $line in text:collection("mydata/users-json.txt")
let $user := json:parse-as-xml($line)
where $user/json:get("age") gt 30
return
  text:put-text($user/json:get("full name"))
```

This query generates text files that contain the following lines:

```
John Doe
Kelly Johnson
```

Example 2

The next query selects the names of employees that have a null age value:

```
import module "oxh:text";
import module "oxh:json";

for $line in text:collection("mydata/users-json.txt")
let $user := json:parse-as-xml($line)
where $user/json:get("age")/nilled()
return
  text:put-text($user/json:get("full name"))
```

This query generates a text file that contains the following line:

```
Laura Smith
```

Utility Module

The utility module contains `ora-fn` functions for handling strings and dates. These functions are defined in XDK XQuery, whereas the `oxh` functions are specific to Oracle XQuery for Hadoop.

The utility functions are described in the following topics:

- [Duration, Date, and Time Functions](#)
- [String Functions](#)

Duration, Date, and Time Functions

These functions are in the `http://xmlns.oracle.com/xdk/xquery/function` namespace. The `ora-fn` prefix is predeclared and the module is automatically imported.

ora-fn:date-from-string-with-format(\$format as xs:string?, \$dateString as xs:string?) as xs:date?

Returns a new date value from a string according to the specified pattern.

`$format` is the pattern. See ["Format Argument"](#) on page 6-82.

`$dateString` is an input string that represents a date.

This example returns the specified date in the current time zone:

```
ora-fn:date-from-string-with-format("yyyy-MM-dd G", "2013-06-22 AD")
```

ora-fn:date-from-string-with-format(\$format as xs:string?, \$dateString as xs:string?, \$locale as xs:string*) as xs:date?

Returns a new date value from a string according to the specified pattern.

`$format` is the pattern. See ["Format Argument"](#) on page 6-82.

`$dateString` is an input string that represents a date.

`$locale` is a one- to three-field value that represents the locale. See ["Locale Argument"](#) on page 6-82.

ora-fn:date-to-string-with-format(\$format as xs:string?, \$date as xs:date?) as xs:string?

Returns a date string with the specified pattern.

`$format` is the pattern. See ["Format Argument"](#) on page 6-82.

`$date` is the date.

This example returns the string "2013-07-15":

```
ora-fn:date-to-string-with-format("yyyy-mm-dd", xs:date("2013-07-15"))
```

ora-fn:date-to-string-with-format(\$format as xs:string?, \$date as xs:date?, *\$locale as xs:string?) as xs:string?

Returns a date string with the specified pattern.

`$format` is the pattern. See ["Format Argument"](#) on page 6-82.

`$date` is the date.

`$locale` is a one- to three-field value that represents the locale. See ["Locale Argument"](#) on page 6-82.

ora-fn:dateTime-from-string-with-format(\$format as xs:string?, \$dateTimeString as xs:string?) as xs:dateTime?

Returns a new dateTime value from an input string according to the specified pattern.

`$format` is the pattern. See ["Format Argument"](#) on page 6-82.

`$dateTimeString` is the date and time.

This example returns the specified date and 11:04:00AM in the current time zone:

```
ora-fn:dateTime-from-string-with-format("yyyy-MM-dd 'at' hh:mm", "2013-06-22 at 11:04")
```

The next example returns the specified date and 12:00:00AM in the current time zone:

```
ora-fn:dateTime-from-string-with-format("yyyy-MM-dd G", "2013-06-22 AD")
```

ora-fn:dateTime-from-string-with-format(\$format as xs:string?, \$dateTimeString as xs:string?, \$locale as xs:string?) as xs:dateTime?

Returns a new dateTime value from an input string according to the specified pattern.

\$format is the pattern. See ["Format Argument"](#) on page 6-82.

\$dateTimeString is the date and time.

\$locale is a one- to three-field value that represents the locale. See ["Locale Argument"](#) on page 6-82.

ora-fn:dateTime-to-string-with-format(\$format as xs:string?, \$dateTime as xs:dateTime?) as xs:string?

Returns a date and time string with the specified pattern.

\$format is the pattern. See ["Format Argument"](#) on page 6-82.

\$dateTime is the date and time.

This example returns the string "07 JAN 2013 10:09 PM AD":

```
ora-fn:dateTime-to-string-with-format("dd MMM yyyy hh:mm a G",
xs:dateTime("2013-01-07T22:09:44"))
```

The next example returns the string "01-07-2013":

```
ora-fn:dateTime-to-string-with-format("MM-dd-yyyy",
xs:dateTime("2013-01-07T22:09:44"))
```

ora-fn:dateTime-to-string-with-format(\$format as xs:string?, \$dateTime as xs:dateTime?, \$locale as xs:string?) as xs:string?

Returns a date and time string with the specified pattern.

\$format is the pattern. See ["Format Argument"](#) on page 6-82.

\$dateTime is the date and time.

\$locale is a one- to three-field value that represents the locale. See ["Locale Argument"](#) on page 6-82.

ora-fn:time-from-string-with-format(\$format as xs:string?, \$timeString as xs:string?) as xs:time?

Returns a new time value from an input string according to the specified pattern.

\$format is the pattern. See ["Format Argument"](#) on page 6-82.

\$timeString is the time.

This example returns 9:45:22PM in the current time zone:

```
ora-fn:time-from-string-with-format("HH.mm.ss", "21.45.22")
```

The next example returns 8:07:22PM in the current time zone:

```
fn-bea:time-from-string-with-format("hh:mm:ss a", "8:07:22 PM")
```

ora-fn:time-from-string-with-format(\$format as xs:string?, \$timeString as xs:string?, \$locale as xs:string?) as xs:time?

Returns a new time value from an input string according to the specified pattern.

\$format is the pattern. See ["Format Argument"](#) on page 6-82.

`$timeString` is the time.

`$locale` is a one- to three-field value that represents the locale. See "[Locale Argument](#)" on page 6-82.

ora-fn:time-to-string-with-format(\$format as xs:string?, \$time as xs:time?) as xs:string?

Returns a time string with the specified pattern.

`$format` is the pattern. See "[Format Argument](#)" on page 6-82.

`$time` is the time.

This example returns the string "10:09 PM":

```
ora-fn:time-to-string-with-format("hh:mm a", xs:time("22:09:44"))
```

The next example returns the string "22:09 PM":

```
ora-fn:time-to-string-with-format("HH:mm a", xs:time("22:09:44"))
```

ora-fn:time-to-string-with-format(\$format as xs:string?, \$time as xs:time?, \$locale as xs:string?) as xs:string?

Returns a time string with the specified pattern.

`$format` is the pattern. See "[Format Argument](#)" on page 6-82.

`$time` is the time.

`$locale` is a one- to three-field value that represents the locale. See "[Locale Argument](#)" on page 6-82.

Format Argument

The `$format` argument identifies the various fields that compose a date or time value.

See Also: The `SimpleDateFormat` class in the *Java Standard Edition 7 Reference* at

<http://docs.oracle.com/javase/7/docs/api/java/text/SimpleDateFormat.html>

Locale Argument

The `$locale` represents a specific geographic, political, or cultural region defined by up to three fields:

1. **Language code:** The ISO 639 alpha-2 or alpha-3 language code, or the registered language subtags of up to eight letters. For example, `en` for English and `ja` for Japanese.
2. **Country code:** The ISO 3166 alpha-2 country code or the UN M.49 numeric-3 area code. For example, `US` for the United States and `029` for the Caribbean.
3. **Variant:** Indicates a variation of the locale, such as a particular dialect. Order multiple values in order of importance and separate them with an underscore (`_`). These values are case sensitive.

See Also:

- The `Locale` class in the *Java Standard Edition 7 Reference* at <http://docs.oracle.com/javase/7/docs/api/java/util/Locale.html>
- All language, country, and variant codes in the Internet Assigned Numbers Authority (IANA) Language Subtag Registry at <http://www.iana.org/assignments/language-subtag-registry/language-subtag-registry>

String Functions

These functions are in the `http://xmlns.oracle.com/xdk/xquery/function` namespace. The `ora-fn` prefix is predeclared and the module is automatically imported.

ora-fn:pad-left(\$str as xs:string?, \$size as xs:integer?) as xs:string?

Adds spaces (ASCII 32) to the left of a string to create a fixed-length string. If the input string exceeds the specified size, then it is truncated to return a substring of the specified length.

`$str` is the input string.

`$size` is the desired fixed length, which is obtained by adding padding characters to `$str`.

If either argument is an empty sequence, then the function returns an empty sequence.

This example prefixes spaces to the string up to the specified maximum of six characters. The returned string has two spaces: " abcd":

```
ora-fn:pad-left("abcd", 6)
```

The next example returns only "ab" because the input string exceeds the specified fixed length:

```
ora-fn:pad-left("abcd", 2)
```

ora-fn:pad-left(\$str as xs:string?, \$size as xs:integer?, \$pad as xs:string?) as xs:string?

Adds padding characters to the left of a string to create a fixed-length string. If the input string exceeds the specified size, then it is truncated to return a substring of the specified length.

`$str` is the input string.

`$size` is the desired fixed length, which is obtained by adding padding characters to `$str`.

`$pad` is the padding character

This example prefixes "01" to the input string up to the maximum of six characters. The returned string is "010abc". The function returns one complete and one partial pad character.

```
ora-fn:pad-left("abc", 6, "01")
```

The next example returns only "ab" because the input string exceeds the specified fixed length:

```
ora-fn:pad-left("abcd", 2, "01")
```

ora-fn:pad-right(\$str as xs:string?, \$size as xs:integer?) as xs:string?

Adds spaces (ASCII 32) to the right of a string to create a fixed-length string. If the input string exceeds the specified size, then it is truncated to return a substring of the specified length.

`$str` is the input string.

`$size` is the desired fixed length, which is obtained by adding padding characters to `$str`.

If either argument is an empty sequence, then the function returns an empty sequence.

This example appends spaces to the string up to the specified maximum of six characters. The returned string has two spaces: "abcd ":

```
ora-fn:pad-right("abcd", 6)
```

The next example returns only "ab" because the input string exceeds the specified fixed length:

```
ora-fn:pad-right("abcd", 2)
```

ora-fn:pad-right(\$str as xs:string?, \$size as xs:integer?, \$pad as xs:string?) as xs:string?

Adds padding characters to the right of a string to create a fixed-length string. If the input string exceeds the specified size, then it is truncated to return a substring of the specified length.

\$str is the input string.

\$size is the desired fixed length, which is obtained by adding padding characters to \$str.

\$pad is the padding character.

This example appends "01" to the input string up to the maximum of six characters. The returned string is "abc010". The function returns one complete and one partial pad character.

```
ora-fn:pad-right("abc", 6, "01")
```

The next example returns only "ab" because the input string exceeds the specified fixed length:

```
ora-fn:pad-right("abcd", 2, "01")
```

ora-fn:trim(\$input as xs:string?) as xs:string?

Removes any leading or trailing white space from a string.

\$input is the string to trim. If \$input is an empty sequence, then the function returns an empty sequence. Other data types trigger an error.

This example returns the string "abc":

```
ora-fn:trim(" abc ")
```

ora-fn:trim-left(\$input as xs:string?) as xs:string?

Removes any leading white space.

\$input is the string to trim. If \$input is an empty sequence, then the function returns an empty sequence. Other data types trigger an error.

This example removes the leading spaces and returns the string "abc ":

```
ora-fn:trim-left("  abc  ")
```

ora-fn:trim-right(\$input as xs:string?) as xs:string?

Removes any trailing white space.

\$input is the string to trim. If \$input is an empty sequence, then the function returns an empty sequence. Other data types trigger an error.

This example removes the trailing spaces and returns the string " abc":

```
ora-fn:trim-right(" abc ")
```

Hadoop Module

These functions are in the `http://xmlns.oracle.com/hadoop/xquery` namespace. The `oxh` prefix is predeclared and the module is automatically imported.

oxh:find(\$pattern as xs:string?) as xs:string*

Returns a sequence of file paths that match a pattern.

See Also: For the file pattern, the `globStatus` method in the Apache Hadoop Main 2.2.0 API at

<http://hadoop.apache.org/docs/current/api/org/apache/hadoop/fs/FileSystem.html#globStatus>

oxh:increment-counter(\$groupName as xs:string, \$counterName as xs:string

Increments a user-defined MapReduce job counter by one.

oxh:increment-counter(\$groupName as xs:string, \$counterName as xs:string, \$value as xs:integer

Increments the MapReduce job counter by the given value.

oxh:property(\$name as xs:string?) as xs:string?

Returns the value of a Hadoop configuration property.

\$name is the configuration property.

Serialization Annotations

Several adapters have serialization annotations (%output: *). The following lists identify the serialization parameters that Oracle XQuery for Hadoop supports.

Serialization parameters supported for the `text` output method:

- `encoding`
- `normalization-form`

Serialization parameters supported for the `xml` output method:

- `cdata-section-elements`
- `doctype-public`
- `doctype-system`
- `encoding`
- `indent`
- `normalization-form`
- `omit-xml-declaration`
- `standalone`

See Also: "The Influence of Serialization Parameters" sections for XML and text output methods in *XSLT and XQuery Serialization 3.0* at

<http://www.w3.org/TR/xslt-xquery-serialization-30/>

Oracle XML Extensions for Hive

This chapter explains how to use the XML extensions for Apache Hive provided on Oracle Big Data Appliance with Oracle XQuery for Hadoop. The chapter contains the following sections:

- [What are the XML Extensions for Apache Hive?](#)
- [Using the Hive Extensions](#)
- [Creating XML Tables](#)
- [XML Function Library for Apache Hive](#)

Note: The features described in this chapter are available only on Oracle Big Data Appliance. You cannot use them on other Hadoop clusters.

What are the XML Extensions for Apache Hive?

Oracle XQuery for Hadoop provides XML processing support that enables you to do the following:

- Query large XML files in HDFS as Hive tables
- Query XML strings in Hive tables
- Query XML file resources in the Hadoop distributed cache
- Efficiently extract atomic values from XML without using expensive DOM parsing
- Retrieve, generate, and transform complex XML elements
- Generate multiple table rows from a single XML value
- Manage missing and dirty data in XML

The XML extensions also support these W3C modern standards:

- XQuery 1.0
- XQuery Update Facility 1.0 (transform expressions)
- XPath 2.0
- XML Schema 1.0
- XML Namespaces

The XML extensions have two components:

- XML InputFormat and SerDe for creating XML tables

See ["Creating XML Tables"](#) on page 7-2.

- XML function library

See ["XML Function Library for Apache Hive"](#) on page 7-10.

Using the Hive Extensions

To enable the Oracle XQuery for Hadoop extensions, use the `--auxpath` and `-i` arguments when starting Hive:

```
$ hive --auxpath $OXH_HOME/hive/lib -i $OXH_HOME/hive/init.sql
```

The first time you use the extensions, you might want to verify that they are accessible. The following procedure creates a table named `SRC`, loads one row into it, and then calls the `xml_query` function.

The `SRC` table is needed only to fulfill a `SELECT` syntax requirement. It is like the `DUAL` table in Oracle Database, which is referenced in `SELECT` statements to test SQL functions.

To verify that the extensions are accessible:

1. Log in to an Oracle Big Data Appliance server where you plan to work.
2. Create a text file named `src.txt` that contains one line:

```
$ echo "XXX" > src.txt
```

3. Start the Hive command-line interface (CLI):

```
$ hive --auxpath $OXH_HOME/hive/lib -i $OXH_HOME/hive/init.sql
```

The `init.sql` file contains the `CREATE TEMPORARY FUNCTION` statements that declare the XML functions.

4. Create a simple table and load data from `src.txt`:

```
hive> CREATE TABLE src(dummy STRING);  
hive> LOAD DATA LOCAL INPATH 'src.txt' OVERWRITE INTO TABLE src;
```

5. Call an Oracle XQuery for Hadoop function for Hive:

```
hive> SELECT xml_query("x/y", "<x><y>123</y><z>456</z></x>") FROM src;
```

If the extensions are accessible, then the query returns `["123"]`.

Creating XML Tables

This section describes how you can use the Hive `CREATE TABLE` statement to create tables over large XML documents.

Hive queries over XML tables scale well, because Oracle XQuery for Hadoop splits up the XML so that the MapReduce framework can process it in parallel.

To support scalable processing and operate in the MapReduce framework, the table adapter scans for elements to use to create table rows. It parses only the elements that it identifies as being part of the table; the rest of the XML is ignored. Thus, the XML table adapter does not perform a true parse of the entire XML document, which imposes limitations on the input XML. Because of these limitations, you can create tables only over XML documents that meet the constraints listed in ["XQuery](#)

[Transformation Requirements](#)" on page 5-6. Otherwise, you may get errors or incorrect results.

Hive CREATE TABLE Syntax for XML Tables

The following is the basic syntax to use when creating a Hive table over XML files:

```
CREATE TABLE table_name (columns)
ROW FORMAT
  SERDE 'oracle.hadoop.xquery.hive.OXMLSerDe'
STORED AS
  INPUTFORMAT 'oracle.hadoop.xquery.hive.OXMLInputFormat'
  OUTPUTFORMAT 'oracle.hadoop.xquery.hive.OXMLOutputFormat'
TBLPROPERTIES (configuration)
```

columns: All column types in an XML table must be one of the Hive primitive types given in ["About Data Type Conversions"](#) on page 7-13.

configuration: Any of the properties described in ["CREATE TABLE Configuration Properties"](#) on page 7-3.

Note: Inserting into XML tables is not supported.

CREATE TABLE Configuration Properties

oxh-elements

Identifies the names of elements in the XML that map to rows in the table, in a comma-delimited list. This property must be specified one time. Required.

This example maps each element named `foo` in the XML to a single row in the Hive table:

```
"oxh-elements" = "foo"
```

The next example maps each element named either `foo` or `bar` in the XML to a row in the Hive table:

```
"oxh-elements" = "foo, bar"
```

oxh-column.name

Specifies how an element selected by the `oxh-elements` property is mapped to columns in a row. In this property name, replace *name* with the name of a column in the table. The value can be any XQuery expression. The initial context item of the expression (the `"."` variable) is bound to the selected element.

Be sure to check log files even when a query executes successfully. If a column expression returns no value or raises a dynamic error, the column value is `NULL`. The first time an error occurs, it is logged and query processing continues. Subsequent errors raised by the same column expression are not logged.

In the following example, the `oxh-elements` property specifies that each element named `foo` in the XML is mapped to a single row in the table. The `oxh-column` properties specify that the Hive table column named `BAR` gets the value of the child element named `bar` converted to `STRING`, and the column named `ZIP` gets the value of the child element named `zip` converted to `INT`.

```
CREATE TABLE example (bar STRING, zip INT)
ROW FORMAT
  SERDE 'oracle.hadoop.xquery.hive.OXMLSerDe'
```

```

STORED AS
  INPUTFORMAT 'oracle.hadoop.xquery.hive.OXMLInputFormat'
  OUTPUTFORMAT 'oracle.hadoop.xquery.hive.OXMLOutputFormat'
TBLPROPERTIES (
  "oxh-elements" = "foo",
  "oxh-column.bar" = "./bat",
  "oxh-column.zip" = "./zip"
)

```

In this modified definition of the zip column, the column receives a value of -1 if the foo element does not have a child zip element, or the zip element contains a nonnumeric value.

```

"oxh-column.zip" = "
  if (./zip castable as xs:int) then
    xs:int(./zip)
  else
    -1
"

```

Any column of the table that does not have a corresponding oxh-column property behaves as if the following property is specified:

```
"oxh-column.name" = "(./name | ./@name)[1]"
```

Thus, the default behavior is to select the first child element or attribute that matches the table column name.

The following two table definitions are equivalent:

Table Definition 1:

```

CREATE TABLE example (bar STRING, zip INT)
ROW FORMAT
  SERDE 'oracle.hadoop.xquery.hive.OXMLSerDe'
STORED AS
  INPUTFORMAT 'oracle.hadoop.xquery.hive.OXMLInputFormat'
  OUTPUTFORMAT 'oracle.hadoop.xquery.hive.OXMLOutputFormat'
TBLPROPERTIES (
  "oxh-elements" = "foo",
  "oxh-column.bar" = "(./bar | ./@bar)[1]",
  "oxh-column.zip" = "(./zip | ./@zip)[1]"
)

```

Table Definition 2:

```

CREATE TABLE example (bar STRING, zip INT)
ROW FORMAT
  SERDE 'oracle.hadoop.xquery.hive.OXMLSerDe'
STORED AS
  INPUTFORMAT 'oracle.hadoop.xquery.hive.OXMLInputFormat'
  OUTPUTFORMAT 'oracle.hadoop.xquery.hive.OXMLOutputFormat'
TBLPROPERTIES (
  "oxh-elements" = "foo"
)

```

oxh-namespace.prefix

Defines a namespace binding.

This example binds the prefix myns to the namespace `http://example.org`:

```
"oxh-namespace.myns" = "http://example.org"
```

You can use this property multiple times to define additional namespaces. The namespace definitions are used when parsing the XML. The `oxh-element` and `oxh-column` property values can also reference them.

In the following examples, only `foo` elements in the `http://example.org` namespace are mapped to table rows:

```
"oxh-namespace.myns" = "http://example.org",
"oxh-elements" = "myns:foo",
"oxh-column.bar" = "./myns:bar"
```

oxh-entity.name

Defines a set of entity reference definitions.

In the following example, entity references in the XML are expanded from `&foo`; to "foo value" and from `&bar`; to "bar value".

```
"oxh-entity.foo" = "foo value"
"oxh-entity.bar" = "bar value"
```

oxh-charset

Specifies the character encoding of the XML files. The supported encodings are UTF-8 (default), ISO-8859-1, and US-ASCII.

This example sets the character set:

```
"oxh-charset" = "ISO-8859-1"
```

CREATE TABLE Examples

This section includes some examples using a simple data set, and other examples using a very large and complex real-world data set:

- [Simple Examples](#)
- [Detailed Examples](#)

Simple Examples

These examples show how to create Hive tables over a small XML document that contains comments posted by users of a fictitious website. Each `comment` element in the document has one or more `like` elements that indicate that the user liked the comment.

```
<comments>
  <comment id="12345" user="john" text="It is raining :( "/>
  <comment id="56789" user="kelly" text="I won the lottery!">
    <like user="john"/>
    <like user="mike"/>
  </comment>
  <comment id="54321" user="mike" text="Happy New Year!">
    <like user="laura"/>
  </comment>
</comments>
```

In the `CREATE TABLE` examples, the `comments.xml` input file is in the current working directory of the local file system.

Example 1 Creating a Table

This example creates a table named `COMMENTS` with a row for each comment, with the user, number of likes, and text.

```
hive>
CREATE TABLE comments (usr STRING, content STRING, likeCt INT)
ROW FORMAT
SERDE 'oracle.hadoop.xquery.hive.OXMLSerDe'
STORED AS
INPUTFORMAT 'oracle.hadoop.xquery.hive.OXMLInputFormat'
OUTPUTFORMAT 'oracle.hadoop.xquery.hive.OXMLOutputFormat'
TBLPROPERTIES (
  "oxh-elements" = "comment",
  "oxh-column.usr" = "./@user",
  "oxh-column.content" = "./@text",
  "oxh-column.likeCt" = "fn:count(./like)"
);
```

The Hive `LOAD DATA` command loads `comments.xml` into the `COMMENTS` table. See ["Simple Examples"](#) on page 7-5 for the contents of the file.

```
hive> LOAD DATA LOCAL INPATH 'comments.xml' OVERWRITE INTO TABLE comments;
```

The following query shows the content of the `COMMENTS` table.

```
hive> SELECT usr, content, likeCt FROM comments;

john  It is raining :(      0
kelly  I won the lottery!    2
mike   Happy New Year!      1
```

Example 2 Querying an XML Column

This example is like [Example 1](#), except that the like elements are produced as XML in a `STRING` column.

```
hive>
CREATE TABLE comments2 (usr STRING, content STRING, likes STRING)
ROW FORMAT
SERDE 'oracle.hadoop.xquery.hive.OXMLSerDe'
STORED AS
INPUTFORMAT 'oracle.hadoop.xquery.hive.OXMLInputFormat'
OUTPUTFORMAT 'oracle.hadoop.xquery.hive.OXMLOutputFormat'
TBLPROPERTIES (
  "oxh-elements" = "comment",
  "oxh-column.usr" = "./@user",
  "oxh-column.content" = "./@text",
  "oxh-column.likes" = "fn:serialize(<likes>{./like}</likes>)"
);
```

The Hive `LOAD DATA` command loads `comments.xml` into the table. See ["Simple Examples"](#) on page 7-5 for the contents of the file.

```
hive> LOAD DATA LOCAL INPATH 'comments.xml' OVERWRITE INTO TABLE comments2;
```

The following query shows the content of the `COMMENTS2` table.

```
hive> SELECT usr, content, likes FROM comments2;

john  It is raining :(      <likes/>
kelly  I won the lottery!    <likes><like user="john"/><like user="mike"/></likes>
mike   Happy New Year!      <likes><like user="laura"/></likes>
```

The next query extracts the user names from the like elements:

```
hive>
SELECT usr, t.user
```

```

FROM comments2
LATERAL VIEW
  xml_table("likes/like", comments2.likes, struct("./@user")) t AS user;

kelly john
kelly mike
mike laura

```

Example 3 Generating XML in a Single String Column

This example creates a table named `COMMENTS3` with a row for each comment, and produces the XML in a single `STRING` column.

```

hive>
CREATE TABLE comments3 (xml STRING)
ROW FORMAT
  SERDE 'oracle.hadoop.xquery.hive.OXMLSerDe'
STORED AS
  INPUTFORMAT 'oracle.hadoop.xquery.hive.OXMLInputFormat'
  OUTPUTFORMAT 'oracle.hadoop.xquery.hive.OXMLOutputFormat'
TBLPROPERTIES(
  "oxh-elements" = "comment",
  "oxh-column.xml" = "fn:serialize(.)"
);

```

The Hive `LOAD DATA` command loads `comments.xml` into the table. See ["Simple Examples"](#) on page 7-5 for the contents of the file.

```

hive> LOAD DATA LOCAL INPATH 'comments.xml' OVERWRITE INTO TABLE comments3;

```

The following query shows the contents of the XML column:

```

hive> SELECT xml FROM comments3;

<comment id="12345" user="john" text="It is raining :( "/>
<comment id="56789" user="kelly" text="I won the lottery!">
  <like user="john"/>
  <like user="mike"/>
</comment>
<comment id="54321" user="mike" text="Happy New Year!">
  <like user="laura"/>
</comment>

```

The next query extracts the IDs and converts them to integers:

```

hive> SELECT xml_query_as_int("comment/@id", xml) FROM comments3;

12345
56789
54321

```

Detailed Examples

These examples use data from OpenStreetMap, which provides free map data for the entire world. You can export the data as XML for specific geographic regions or the entire planet. An OpenStreetMap XML document mainly contains a sequence of `node`, `way`, and `relation` elements.

In these examples, the OpenStreetMap XML files are stored in the `/user/name/osm` HDFS directory.

See Also:

- To download OpenStreetMap data:
<http://www.openstreetmap.org/export>
- For information about the OpenStreetMap XML format:
http://wiki.openstreetmap.org/wiki/OSM_XML

Example 1 Creating a Table Over OpenStreetMap XML

This example creates a table over OpenStreetMap XML with one row for each node element as follows:

- The `id`, `lat`, `lon`, and `user` attributes of the `node` element are mapped to table columns.
- The year is extracted from the `timestamp` attribute and mapped to the `YEAR` column. If a node does not have a `timestamp` attribute, then `-1` is used for the year.
- If the node element has any child `tag` elements, then they are stored as an XML string in the `TAGS` column. If node has no child `tag` elements, then column value is `NULL`.

```
hive>
CREATE EXTERNAL TABLE nodes (
  id BIGINT,
  latitude DOUBLE,
  longitude DOUBLE,
  year SMALLINT,
  tags STRING
)
ROW FORMAT
SERDE 'oracle.hadoop.xquery.hive.OXMLSerDe'
STORED AS
INPUTFORMAT 'oracle.hadoop.xquery.hive.OXMLInputFormat'
OUTPUTFORMAT 'oracle.hadoop.xquery.hive.OXMLOutputFormat'
LOCATION '/user/name/osm'
TBLPROPERTIES (
  "oxh-elements" = "node",
  "oxh-column.id" = "./@id",
  "oxh-column.latitude" = "./@lat",
  "oxh-column.longitude" = "./@lon",
  "oxh-column.year" = "
    if (fn:exists(./@timestamp)) then
      fn:year-from-dateTime(xs:dateTime(./@timestamp))
    else
      -1
  ",
  "oxh-column.tags" = "
    if (fn:exists(./tag)) then
      fn:serialize(<tags>{./tag}</tags>)
    else
      ()
  "
);
```

The following query returns the number of nodes per year:

```
hive> SELECT year, count(*) FROM nodes GROUP BY year;
```

This query returns the total number of tags across nodes:

```
hive>
  SELECT sum(xml_query_as_int("count(tags/tag)", tags))
  FROM nodes;
```

Example 2

In OpenStreetMap XML, the node, way, and relation elements share a set of common attributes such as the user who contributed the data. The next table produces one row for each node, way, and relation element.

See Also: For a description of the OpenStreetMap elements and attributes:

<http://wiki.openstreetmap.org/wiki/Elements>

```
hive>
  CREATE EXTERNAL TABLE osm (
    id BIGINT,
    uid BIGINT,
    type STRING
  )
  ROW FORMAT
  SERDE 'oracle.hadoop.xquery.hive.OXMLSerDe'
  STORED AS
  INPUTFORMAT 'oracle.hadoop.xquery.hive.OXMLInputFormat'
  OUTPUTFORMAT 'oracle.hadoop.xquery.hive.OXMLOutputFormat'
  LOCATION '/user/name/osm'
  TBLPROPERTIES (
    "oxh-elements" = "node, way, relation",
    "oxh-column.id" = "./@id",
    "oxh-column.uid" = "./@uid",
    "oxh-column.type" = "./name()"
  );
```

The following query returns the number of node, way, and relation elements. The TYPE column is set to the name of the selected element, which is either node, way, or relation.

```
hive>
  SELECT type, count(*)
  FROM osm
  GROUP BY type;
```

This query returns the number of distinct user IDs:

```
hive>
  SELECT count(*)
  FROM (
    SELECT uid
    FROM osm
    GROUP BY uid
  ) t;
```

XML Function Library for Apache Hive

This section describes the functions provided with the XML Extensions for Hive. It contains the following topics:

- [Online Documentation of Functions](#)
- [About Hive Access to External Files](#)
- [About Data Type Conversions](#)

The XML Extensions for Hive contain the following functions:

- [xml_query](#)
- [xml_query_as_primitive](#)
- [xml_exists](#)
- [xml_table](#)

Online Documentation of Functions

You can get online Help for the Hive extension functions by using this command:

```
DESCRIBE FUNCTION [EXTENDED] function_name;
```

This example provides a brief description of the `xml_query` function:

```
hive> describe function xml_query;  
OK  
xml_query(query, bindings) - Returns the result of the query as a STRING array
```

The `EXTENDED` option provides a detailed description and examples:

```
hive> describe function extended xml_query;  
OK  
xml_query(query, bindings) - Returns the result of the query as a STRING array  
Evaluates an XQuery expression with the specified bindings. The query argument  
must be a STRING and the bindings argument must be a STRING or a STRUCT. If the  
bindings argument is a STRING, it is parsed as XML and bound to the initial  
context item of the query. For example:  
  
  > SELECT xml_query("x/y", "<x><y>hello</y><z/><y>world</y></x>") FROM src LIMIT  
1;  
  ["hello", "world"]  
  .  
  .  
  .
```

About Hive Access to External Files

The Hive functions have access to the following external file resources:

- XML schemas
See <http://www.w3.org/TR/xquery/#id-schema-import>.
- XML documents
See <http://www.w3.org/TR/xpath-functions/#func-doc>.
- XQuery library modules
See <http://www.w3.org/TR/xquery/#id-module-import>.

You can address these files by their URI from either HTTP (by using the `http://...` syntax) or the local file system (by using the `file://...` syntax). In this example, relative file locations are resolved against the local working directory of the task, so that URIs such as `bar.xsd` can be used to access files that were added to the distributed cache:

```
xml_query("
  import schema namespace tns='http://example.org' at 'bar.xsd';
  validate { ... }
  ",
  .
  .
  .
```

To access a local file, first add it to the Hadoop distributed cache using the Hive `ADD FILE` command. For example:

```
ADD FILE /local/mydir/thisfile.xsd;
```

Otherwise, you must ensure that the file is available on all nodes of the cluster, such as by mounting the same network drive or simply copying the file to every node. The default base URI is set to the local working directory.

See Also: For information about the default base URI, see *XQuery 1.0: An XML Query Language* at

<http://www.w3.org/TR/xquery/#dt-base-uri>

About Data Type Conversions

Table 7-1 shows the conversions that occur automatically between Hive primitives and XML schema types.

Table 7-1 *Data Type Equivalents*

Hive	XML schema
TINYINT	xs:byte
SMALLINT	xs:short
INT	xs:int
BIGINT	xs:long
BOOLEAN	xs:boolean
FLOAT	xs:float
DOUBLE	xs:double
STRING	xs:string

xml_query

Returns the result of a query as an array of `STRING` values.

Signature

```
xml_query(
  STRING query,
  { STRING | STRUCT } bindings
) as ARRAY<STRING>
```

Description

query

An XQuery or XPath expression. It must be a constant value, because it is only read the first time the function is evaluated. The initial query string is compiled and reused in all subsequent calls.

You can access files that are stored in the Hadoop distributed cache and HTTP resources (`http://...`). Use the XQuery `fn:doc` function for XML documents, and the `fn:unparsed-text` and `fn:parsed-text-lines` functions to access plain text files. See [Example 4](#).

If an error occurs while compiling the query, the function raises an error. If an error occurs while evaluating the query, the error is logged (not raised) and an empty array is returned.

bindings

The input that the query processes. The value can be an XML `STRING` or a `STRUCT` of variable values:

- `STRING`: The string is bound to the initial context item of the query as XML. See [Example 1](#).
- `STRUCT`: A `STRUCT` with an even number of fields. Each pair of fields defines a variable binding (name, value) for the query. The name fields must be type `STRING`, and the value fields can be any supported primitive. See "[About Data Type Conversions](#)" on page 7-13 and [Example 2](#).

Return Value

A Hive array of `STRING` values, which are the result of the query converted to a sequence of atomic values. If the result of the query is empty, then the return value is an empty array.

Examples

Example 1 Using a `STRING` Binding

This example parses and binds the input XML string to the initial context item of the query "x/y":

```
> SELECT xml_query("x/y", "<x><y>hello</y><z/><y>world</y></x>") FROM src LIMIT 1;
["hello", "world"]
```

Example 2 Using a STRUCT Binding

In this example, the second argument is a `STRUCT` that defines two query variables, `$data` and `$value`. The values of the variables in the `STRUCT` are converted to XML schema types as described in ["About Data Type Conversions"](#) on page 7-13.

```
> SELECT xml_query(
  "fn:parse-xml($data)/x/y[@id = $value]",
  struct(
    "data", "<x><y id='1'>hello</y><z/><y id='2'>world</y></x>",
    "value", 1
  )
) FROM src LIMIT 1;
["hello"]
```

Example 3 Obtaining Serialized XML

This example uses the `fn:serialize` function to return serialized XML:

```
> SELECT xml_query(
  "for $y in x/y
  return fn:serialize($y)
",
  "<x><y>hello</y><z/><y>world</y></x>"
) FROM src LIMIT 1;
["<y>hello</y>", "<y>world</y>"]
```

Example 4 Accessing the Hadoop Distributed Cache

This example adds a file named `test.xml` to the distributed cache, and then queries it using the `fn:doc` function. The file contains the value `<x><y>hello</y><z/><y>world</y></x>`.

```
> ADD FILE test.xml;
> SELECT xml_query("fn:doc('test.xml')/x/y", NULL) FROM src LIMIT 1;
["hello", "world"]
```

Example 5 Results of a Failed Query

The next example returns an empty array because the input XML is invalid. The XML parsing error will be written to the log:

```
> SELECT xml_query("x/y", "<x><y>hello</y></invalid") FROM src LIMIT 1;
[]
```

xml_query_as_primitive

Maps the result of an XQuery expression to a Hive primitive value. Each Hive primitive data type has a separate function named for it:

- `xml_query_as_bigint`
- `xml_query_as_boolean`
- `xml_query_as_double`
- `xml_query_as_float`
- `xml_query_as_int`
- `xml_query_as_smallint`
- `xml_query_as_string`
- `xml_query_as_tinyint`

Signature

```
xml_query_as_primitive (  
    STRING query,  
    {STRUCT | STRING} bindings,  
) as primitive
```

Description

query

An XQuery or XPath expression.

bindings

The input that the query processes. The value can be an XML `STRING` or a `STRUCT` of variable values:

- `STRING`: The string is bound to the initial context item of the query as XML. See [Example 1](#).
- `STRUCT`: A `STRUCT` with an even number of fields. Each pair of fields defines a variable binding (name, value) for the query. The name fields must be type `STRING`, and the value fields can be any supported primitive. See "[About Data Type Conversions](#)" on page 7-13 and [Example 2](#).

The first item in the result of the query is cast to the XML schema type that maps to the primitive type of the function. If the query returns multiple items, then all but the first are ignored.

Return Value

The result of the query as a Hive primitive value

NULL if the query returns an empty result

Examples

Example 1 Using a STRING Binding

This example parses and binds the input XML string to the initial context item of the query "x/y":

```
> SELECT xml_query_as_string("x/y", "<x><y>hello</y></x>") FROM src LIMIT 1;
"hello"
```

The following are string binding examples that use other primitive functions:

```
> SELECT xml_query_as_int("x/y", "<x><y>123</y></x>") FROM src LIMIT 1;
123
```

```
> SELECT xml_query_as_double("x/y", "<x><y>12.3</y></x>") FROM src LIMIT 1;
12.3
```

```
> SELECT xml_query_as_boolean("x/y", "<x><y>>true</y></x>") FROM src LIMIT 1;
true
```

Example 2 Using a STRUCT Binding

In this example, the second argument is a STRUCT that defines two query variables, \$data and \$value. The values of the variables in the STRUCT are converted to XML schema types as described in ["About Data Type Conversions"](#) on page 7-13.

```
> SELECT xml_query_as_string(
    "fn:parse-xml($data)/x/y[@id = $value]",
    struct(
        "data", "<x><y id='1'>hello</y><z/><y id='2'>world</y></x>",
        "value", 2
    )
) FROM src LIMIT 1;
"world"
```

Example 3 Returning Multiple Query Results

This example returns only the first item (hello) from the query. The second item (world) is discarded.

```
> SELECT xml_query_as_string("x/y", "<x><y>hello</y><z/><y>world</y></x>") FROM
src LIMIT 1;
"hello"
```

Example 4 Returning Empty Query Results

This example returns NULL because the result of the query is empty:

```
> SELECT xml_query_as_string("x/foo", "<x><y>hello</y><z/><y>world</y></x>") FROM
src LIMIT 1;
NULL
```

Example 5 Obtaining Serialized XML

These examples use the fn:serialize function to return complex XML elements as a STRING value:

```
> SELECT xml_query_as_string("fn:serialize(x/y[1])",
"<x><y>hello</y><z/><y>world</y></x>") FROM src LIMIT 1;
"<y>hello</y>"
```

```
> SELECT xml_query_as_string(
    "fn:serialize(<html><head><title>{$desc}</title></head><body>Name:"
```

```
{ $name } </body></html> ",
  struct(
    "desc", "Employee Details",
    "name", "John Doe"
  )
) FROM src LIMIT 1;
"<html><head><title>Employee Details</title></head><body>Name: John
Doe</body></html>"
```

Example 6 Accessing the Hadoop Distributed Cache

This example adds a file named test.xml to the distributed cache, and then queries it using the fn:doc function. The file contains the value

```
<x><y>hello</y><z/><y>world</y></x>.
```

```
> ADD FILE test.xml;
> SELECT xml_query_as_string("fn:doc('test.xml')/x/y[1]", NULL) FROM src LIMIT 1;
"hello"
```

Example 7 Results of a Failed Query

This example returns NULL because </invalid is missing an angle bracket. An XML parsing error is written to the log:

```
> SELECT xml_query_as_string("x/y", "<x><y>hello</invalid") FROM src LIMIT 1;
NULL
```

This example returns NULL because foo cannot be cast as xs:float. A cast error is written to the log:

```
> SELECT xml_query_as_float("x/y", "<x><y>foo</y></x>") FROM src LIMIT 1;
NULL
```

xml_exists

Tests if the result of a query is empty.

Signature

```
xml_exists(
  STRING query,
  { STRING | STRUCT } bindings
) as BOOLEAN
```

Description

query

An XQuery or XPath expression.

bindings

The input that the query processes. The value can be an XML STRING or a STRUCT of variable values:

- STRING: The string is bound to the initial context item of the query as XML.
- STRUCT: A STRUCT with an even number of fields. Each pair of fields defines a variable binding (name, value) for the query. The name fields must be type STRING, and the value fields can be any supported primitive. See ["About Data Type Conversions"](#) on page 7-13.

Return Value

true if the result of the query is not empty, and false otherwise.

false if the query raises a dynamic error.

Notes

The first dynamic error raised by a query is logged, but subsequent errors are suppressed.

Examples

Example 1 STRING Binding

This example parses and binds the input XML string to the initial context item of the query "x/y":

```
> SELECT xml_exists("x/y", "<x><y>123</y></x>") FROM src LIMIT 1;
true
```

Example 2 STRUCT Binding

This example defines two query variables, \$data and \$value.

```
> SELECT xml_exists(
  "parse-xml($data)/x/y[@id = $value]",
  struct(
    "data", "<x><y id='1' /><y id='2' /></x>",
    "value", 2
  )
)
```

```
    ) FROM src LIMIT 1;  
true
```

Example 3 Error Logging

In this example, an error is written to the log, because the input XML is invalid.

```
> SELECT xml_exists("x/y", "<x><y>123</invalid></x>") FROM src LIMIT 1;  
false
```

xml_table

A table-generating function (UDTF) that maps an XML value to zero or more table rows. This function enables nested repeating elements in XML to be mapped to Hive table rows.

Signature

```
xml_table(
  STRUCT? namespaces,
  STRING query,
  {STRUCT | STRING} bindings,
  STRUCT? columns
)
```

Description

namespaces

Identifies the namespaces that the query and column expressions can use. Optional.

The value is a `STRUCT` with an even number of `STRING` fields. Each pair of fields defines a namespace binding (prefix, URI) that can be used by the query or the column expressions. See [Example 3](#).

query

An XQuery or XPath expression that generates a table row for each returned value. It must be a constant value, because it is only read the first time the function is evaluated. The initial query string is compiled and reused in all subsequent calls.

If a dynamic error occurs during query processing, then the function does not raise an error, but logs it the first time. Subsequent dynamic errors are not logged.

bindings

The input that the query processes. The value can be an XML `STRING` or a `STRUCT` of variable values:

- `STRING`: The string is bound to the initial context item of the query as XML. See [Example 1](#).
- `STRUCT`: A `STRUCT` with an even number of fields. Each pair of fields defines a variable binding (name, value) for the query. The name fields must be type `STRING`, and the value fields can be any supported primitive. See "[About Data Type Conversions](#)" on page 7-13.

columns

The XQuery or XPath expressions that define the columns of the generated rows. Optional.

The value is a `STRUCT` that contains additional XQuery expressions, which define the column values in the row. The XQuery expressions must be constant `STRING` values, because they are only read the first time the function is evaluated. For each column expression in the `STRUCT`, there is one column in the table.

For each item returned by the query, the column expressions are evaluated with the current item as the initial context item of the expression. The results of the column expressions are converted to `STRING` values and become the values of the row.

If the result of a column expression is empty or if a dynamic error occurs while evaluating the column expression, then the corresponding column value is NULL. If a column expression returns more than one item, then all but the first are ignored.

Omitting the *columns* argument is the same as specifying 'struct(". ")'. See [Example 2](#).

Return Value

One table row for each item returned by the *query* argument.

Notes

The XML table adapter enables Hive tables to be created over large XML files in HDFS. See "[Creating XML Tables](#)" on page 7-2.

Examples

Example 1 Using a STRING Binding

The query "x/y" returns two <y> elements, and so two table rows are generated. Because there are two column expressions (". /z", ". /w"), each row has two columns.

```
> SELECT xml_table(
  "x/y",
  "<x>
    <y>
      <z>a</z>
      <w>b</w>
    </y>
    <y>
      <z>c</z>
    </y>
  </x>
",
  struct(". /z", ". /w")
) AS (z, w)
FROM src;
```

```
a  b
c  NULL
```

Example 2 Using the *Columns* Argument

The following two queries are equivalent. The first query explicitly specifies the value of the *columns* argument:

```
> SELECT xml_table(
  "x/y",
  "<x><y>hello</y><y>world</y></x>",
  struct(".")
) AS (y)
FROM src;
```

```
hello
world
```

The second query omits the *columns* argument, which defaults to struct(". "):

```
> SELECT xml_table(
  "x/y",
```

```

    "<x><y>hello</y><y>world</y></x>"
  ) AS (y)
FROM src;

```

```

hello
world

```

Example 3 Using the *Namespaces* Argument

This example specifies the optional namespaces argument, which has an ns prefix and a URI of `http://example.org`.

```

> SELECT xml_table(
    struct("ns", "http://example.org"),
    "ns:x/ns:y",
    "<x xmlns='http://example.org'><y><z/></y><y><z/><z/></y></x>",
    struct("count(./ns:z)")
  ) AS (y)
FROM src;

```

```

1
2

```

Example 4 Querying a Hive Table of XML Documents

This example queries a table named `COMMENTS`, which has a single column named `XML_STR` of type `STRING`. It contains these three rows:

```

> SELECT xml_str FROM comments;

<comment id="12345" user="john" text="It is raining:"/>
<comment id="56789" user="kelly" text="I won the lottery!"><like
user="john"/><like user="mike"/></comment>
<comment id="54321" user="mike" text="Happy New Year!"><like
user="laura"/></comment>

```

The following query shows how to extract the user, text, and number of likes from the `COMMENTS` table.

```

> SELECT t.id, t.usr, t.likes
    FROM comments LATERAL VIEW xml_table(
        "comment",
        comments.xml_str,
        struct("./@id", "./@user", "fn:count(./like)")
    ) t AS id, usr, likes;

```

```

12345  john    0
56789  kelly   2
54321  mike    1

```

Note: You could use the `xml_query_as_string` function to achieve the same result in this example. However, `xml_table` is more efficient, because a single function call sets all three column values and parses the input XML only once for each row. The `xml_query_as_string` function requires a separate function call for each of the three columns and reparses the same input XML value each time.

Example 5 Mapping Nested XML Elements to Table Rows

This example shows how to use `xml_table` to flatten nested, repeating XML elements into table rows. See [Example 4](#) for the `COMMENTS` table.

```
> SELECT t.i, t.u, t.l
       FROM comments LATERAL VIEW xml_table (
         "let $comment := ./comment
         for $like in $comment/like
         return
           <r>
             <id>{$comment/@id/data()}</id>
             <user>{$comment/@user/data()}</user>
             <like>{$like/@user/data()}</like>
           </r>
         ",
         comments.xml_str,
         struct("./id", "./user", "./like")
       ) t AS i, u, l;

56789  kelly  john
56789  kelly  mike
54321  mike   laura
```

Example 6 Mapping Optional Nested XML Elements to Table Rows

This example is a slight modification of [Example 5](#) that still produces a row when a comment has no likes. See [Example 4](#) for the `COMMENTS` table.

```
> SELECT t.i, t.u, t.l
       FROM comments LATERAL VIEW xml_table (
         "let $comment := ./comment
         for $like allowing empty in $comment/like
         return
           <r>
             <id>{$comment/@id/data()}</id>
             <user>{$comment/@user/data()}</user>
             <like>{$like/@user/data()}</like>
           </r>
         ",
         comments.xml_str,
         struct("./id", "./user", "./like")
       ) t AS i, u, l;

12345  john
56789  kelly  john
56789  kelly  mike
54321  mike   laura
```

Example 7 Creating a New View

You can create views and new tables using `xml_table`, the same as any table-generating function. This example creates a new view named `COMMENTS_LIKES` from the `COMMENTS` table:

```
> CREATE VIEW comments_likes AS
  SELECT xml_table(
    "comment",
    comments.xml_str,
    struct("./@id", "count(./like)")
  ) AS (id, likeCt)
  FROM comments;
```

This example queries the new view:

```
> SELECT * FROM comments_likes
   WHERE CAST(likeCt AS INT) != 0;
```

```
56789  2
54321  1
```

Example 8 Accessing the Hadoop Distributed Cache

You can access XML documents and text files added to the distributed cache by using the `fn:doc` and `fn:unparsed-text` functions.

This example queries a file named `test.xml` that contains the string

```
"<x><y>hello</y><z/><y>world</y></x>"
```

```
> ADD FILE test.xml;
> SELECT xml_table("fn:doc('test.xml')/x/y", NULL) AS y FROM src;
```

```
hello
world
```


Part IV

Oracle R Advanced Analytics for Hadoop

This part contains the following chapters:

- [Chapter 8, "Using Oracle R Advanced Analytics for Hadoop"](#)
- [Chapter 9, "ORCH Library Reference"](#)

Using Oracle R Advanced Analytics for Hadoop

This chapter describes R support for big data. It contains the following sections:

- [About Oracle R Advanced Analytics for Hadoop](#)
- [Access to HDFS Files](#)
- [Access to Apache Hive](#)
- [Access to Oracle Database](#)
- [Analytic Functions in Oracle R Advanced Analytics for Hadoop](#)
- [ORCH mapred.config Class](#)
- [Examples and Demos of Oracle R Advanced Analytics for Hadoop](#)
- [Security Notes for Oracle R Advanced Analytics for Hadoop](#)

About Oracle R Advanced Analytics for Hadoop

Oracle R Advanced Analytics for Hadoop is a collection of R packages that provide:

- Interfaces to work with Apache Hive tables, the Apache Hadoop compute infrastructure, the local R environment, and Oracle database tables
- Predictive analytic techniques, written in R or Java as Hadoop MapReduce jobs, that can be applied to data in HDFS files

You install and load this package as you would any other R package. Using simple R functions, you can perform tasks like these:

- Access and transform HDFS data using a Hive-enabled transparency layer
- Use the R language for writing mappers and reducers
- Copy data between R memory, the local file system, HDFS, Hive, and Oracle databases
- Schedule R programs to execute as Hadoop MapReduce jobs and return the results to any of those locations

Several analytic algorithms are available in Oracle R Advanced Analytics for Hadoop: linear regression, neural networks for prediction, matrix completion using low rank matrix factorization, clustering, and nonnegative matrix factorization. They are written in either Java or R.

To use Oracle R Advanced Analytics for Hadoop, you should be familiar with MapReduce programming, R programming, and statistical methods.

Oracle R Advanced Analytics for Hadoop APIs

Oracle R Advanced Analytics for Hadoop provides access from a local R client to Apache Hadoop using functions with these prefixes:

- `hadoop`: Identifies functions that provide an interface to Hadoop MapReduce
- `hdfs`: Identifies functions that provide an interface to HDFS
- `orch`: Identifies a variety of functions; `orch` is a general prefix for ORCH functions
- `ore`: Identifies functions that provide an interface to a Hive data store

Oracle R Advanced Analytics for Hadoop uses data frames as the primary object type, but it can also operate on vectors and matrices to exchange data with HDFS. The APIs support the numeric, integer, and character data types in R.

All of the APIs are included in the `ORCH` library. The functions are listed in [Chapter 9](#) in alphabetical order.

See Also: The R Project website at <http://www.r-project.org/>

Access to HDFS Files

For Oracle R Advanced Analytics for Hadoop to access the data stored in HDFS, the input files must comply with the following requirements:

- All input files for a MapReduce job must be stored in one directory as the parts of one logical file. Any valid HDFS directory name and file name extensions are acceptable.
- Any file in that directory with a name beginning with an underscore (`_`) is ignored.
- The input files must be in comma-separated value (CSV) format as follows:
 - `key := string | ""`
 - `value := string[,value]`
 - `line := [key\t]value`
 - `string := char[string]`
 - `char := regexp([^\n])`

Input File Format Examples

The following are examples of acceptable CSV input files:

- CSV files with a key:


```
"1\tHello,world"
"2\tHi,there"
```
- CSV files with no key:


```
"Hello,world"
"Hi,there"
```
- CSV files with a NULL key:


```
"\tHello,world"
"\tHi,there"
```

Access to Apache Hive

Apache Hive provides an alternative storage and retrieval mechanism to HDFS files through a querying language called HiveQL, which closely resembles SQL. Hive uses MapReduce for distributed processing. However, the data is structured and has additional metadata to support data discovery. Oracle R Advanced Analytics for Hadoop uses the data preparation and analysis features of HiveQL, while enabling you to use R language constructs.

See Also: The Apache Hive website at <http://hive.apache.org>

ORE Functions for Hive

You can connect to Hive and manage objects using R functions that have an `ore` prefix, such as `ore.connect`. If you are also using Oracle R Enterprise, then you recognize these functions. The `ore` functions in Oracle R Enterprise create and manage objects in an Oracle database, and the `ore` functions in Oracle R Advanced Analytics for Hadoop create and manage objects in a Hive database. You can connect to one database at a time, either Hive or Oracle Database, but not both simultaneously.

The following ORE functions are supported in Oracle R Advanced Analytics for Hadoop:

```
as.ore
as.ore.character
as.ore.frame
as.ore.integer
as.ore.logical
as.ore.numeric
as.ore.vector
is.ore
is.ore.character
is.ore.frame
is.ore.integer
is.ore.logical
is.ore.numeric
is.ore.vector
ore.create
ore.drop
ore.get
ore.pull
ore.push
ore.recode
```

This release does not support `ore.factor`, `ore.list`, or `ore.object`.

These aggregate functions from OREStats are also supported:

```
aggregate
fivenum
IQR
median
quantile
sd
var*
```

*For vectors only

Generic R Functions Supported in Hive

Oracle R Advanced Analytics for Hadoop also overloads the following standard generic R functions with methods to work with Hive objects.

Character methods

casefold, chartr, gsub, nchar, substr, substring, tolower, toupper

This release does not support grepl or sub.

Frame methods

- attach, show
- [, \$, \$<-, [[, [[<-
- Subset functions: head, tail
- Metadata functions: dim, length, NROW, nrow, NCOL, ncol, names, names<-, colnames, colnames<-
- Conversion functions: as.data.frame, as.env, as.list
- Arithmetic operators: +, -, *, ^, %%, %/%, /
- Compare, Logic, xor, !
- Test functions: is.finite, is.infinite, is.na, is.nan
- Mathematical transformations: abs, acos, asin, atan, ceiling, cos, exp, expm1, floor, log, log10, log1p, log2, logb, round, sign, sin, sqrt, tan, trunc
- Basic statistics: colMeans, colSums, rowMeans, rowSums, Summary, summary, unique
- by, merge
- unlist, rbind, cbind, data.frame, eval

This release does not support dimnames, interaction, max.col, row.names, row.names<-, scale, split, subset, transform, with, or within.

Logical methods

ifelse, Logic, xor, !

Matrix methods

Not supported

Numeric methods

- Arithmetic operators: +, -, *, ^, %%, %/%, /
- Test functions: is.finite, is.infinite, is.nan
- abs, acos, asin, atan, ceiling, cos, exp, expm1, floor, log, log1p, log2, log10, logb, mean, round, sign, sin, sqrt, Summary, summary, tan, trunc, zapsmall

This release does not support atan2, besselI, besselK, besselJ, bessely, diff, factorial, lfactorial, pmax, pmin, or tabulate.

Vector methods

- show, length, c
- Test functions: is.vector, is.na
- Conversion functions: as.vector, as.character, as.numeric, as.integer, as.logical
- [, [<-, |

- by, Compare, head, %in%, paste, sort, table, tail, tapply, unique

This release does not support interaction, lengthb, rank, or split.

[Example 8-1](#) shows simple data preparation and processing. For additional details, see ["Support for Hive Data Types"](#) on page 8-5.

Example 8-1 Using R to Process Data in Hive Tables

```
# Connect to Hive
ore.connect(type="HIVE")

# Attach the current envt. into search path of R
ore.attach()

# create a Hive table by pushing the numeric columns of the iris data set
IRIS_TABLE <- ore.push(iris[1:4])

# Create bins based on Petal Length
IRIS_TABLE$PetalBins = ifelse(IRIS_TABLE$Petal.Length < 2.0, "SMALL PETALS",
+                             ifelse(IRIS_TABLE$Petal.Length < 4.0, "MEDIUM PETALS",
+                             ifelse(IRIS_TABLE$Petal.Length < 6.0,
+                             "MEDIUM LARGE PETALS", "LARGE PETALS"))

#PetalBins is now a derived column of the HIVE object
> names(IRIS_TABLE)
[1] "Sepal.Length" "Sepal.Width" "Petal.Length" "Petal.Width" "PetalBins"

# Based on the bins, generate summary statistics for each group
aggregate(IRIS_TABLE$Petal.Length, by = list(PetalBins = IRIS_TABLE$PetalBins),
+         FUN = summary)
1      LARGE PETALS      6 6.025000 6.200000 6.354545 6.612500 6.9 0
2 MEDIUM LARGE PETALS  4 4.418750 4.820000 4.888462 5.275000 5.9 0
3      MEDIUM PETALS   3 3.262500 3.550000 3.581818 3.808333 3.9 0
4      SMALL PETALS    1 1.311538 1.407692 1.462000 1.507143 1.9 0
Warning message:
ORE object has no unique key - using random order
```

Support for Hive Data Types

Oracle R Advanced Analytics for Hadoop can access any Hive table containing columns with string and numeric data types such as tinyint, smallint, bigint, int, float, and double. There is no support for these complex data types:

```
array
binary
map
struct
timestamp
union
```

If you attempt to access a Hive table containing an unsupported data type, then you get an error message. To access the table, you must convert the column to a supported data type.

To convert a column to a supported data type:

1. Open the Hive command interface:

```
$ hive
hive>
```

- Identify the column with an unsupported data type:

```
hive> describe table_name;
```

- View the data in the column:

```
hive> select column_name from table_name;
```

- Create a table for the converted data, using only supported data types.

- Copy the data into the new table, using an appropriate conversion tool.

[Example 8-2](#) shows the conversion of an array. [Example 8-3](#) and [Example 8-4](#) show the conversion of timestamp data.

Example 8-2 Converting an Array to String Columns

```
R> ore.sync(table="t1")
Warning message:
table t1 contains unsupported data types
.
.
.
hive> describe t1;
OK
      col1  int
      col2  array<string>

hive> select * from t1;
OK
 1      ["a","b","c"]
 2      ["d","e","f"]
 3      ["g","h","i"]

hive> create table t2 (c1 string, c2 string, c2 string);
hive> insert into table t2 select col2[0], col2[1], col2[2] from t1;
.
.
.
R> ore.sync(table="t2")
R> ore.ls()
[1] "t2"
R> t2$c1
[1] "a" "d" "g"
```

[Example 8-3](#) uses automatic conversion of the timestamp data type into string. The data is stored in a table named t5 with a column named tstamp.

Example 8-3 Converting a Timestamp Column

```
hive> select * from t5;

hive> create table t6 (tstamp string);
hive> insert into table t6 SELECT tstamp from t5;
```

[Example 8-4](#) uses the Hive `get_json_object` function to extract the two columns of interest from the JSON table into a separate table for use by Oracle R Advanced Analytics for Hadoop.

Example 8–4 Converting a Timestamp Column in a JSON File

```
hive> select * from t3;
OK

{"custId":1305981,"movieId":null,"genreId":null,"time":"2010-12-30:23:59:32","recommended":null,"activity":9}

hive> create table t4 (custid int, time string);

hive> insert into table t4 SELECT cast(get_json_object(c1, '$.custId') as int),
cast(get_json_object(c1, '$.time') as string) from t3;
```

Usage Notes for Hive Access

The Hive command language interface (CLI) is used for executing queries and provides support for Linux clients. There is no JDBC or ODBC support.

The `ore.create` function creates Hive tables only as text files. However, Oracle R Advanced Analytics for Hadoop can access Hive tables stored as either text files or sequence files.

You can use the `ore.exec` function to execute Hive commands from the R console. For a demo, run the `hive_sequencefile` demo.

Oracle R Advanced Analytics for Hadoop can access tables and views in the default Hive database only. To allow read access to objects in other databases, you must expose them in the default database. For example, you can create views.

Oracle R Advanced Analytics for Hadoop does not have a concept of ordering in Hive. An R frame persisted in Hive might not have the same ordering after it is pulled out of Hive and into memory. Oracle R Advanced Analytics for Hadoop is designed primarily to support data cleanup and filtering of huge HDFS data sets, where ordering is not critical. You might see warning messages when working with unordered Hive frames:

```
Warning messages:
1: ORE object has no unique key - using random order
2: ORE object has no unique key - using random order
```

To suppress these warnings, set the `ore.warn.order` option in your R session:

```
R> options(ore.warn.order = FALSE)
```

Example: Loading Hive Tables into Oracle R Advanced Analytics for Hadoop

[Table 9–1](#) provides an example of loading a Hive table into an R data frame for analysis. It uses these Oracle R Advanced Analytics for Hadoop functions:

```
hdfs.attach
ore.attach
ore.connect
ore.create
ore.hiveOptions
ore.sync
```

Example 8–5 Loading a Hive Table

```
# Connect to HIVE metastore and sync the HIVE input table into the R session.
ore.connect(type="HIVE")
ore.sync(table="datatab")
ore.attach()
```

```
# The "datatab" object is a Hive table with columns named custid, movieid,
activity, and rating.
# Perform filtering to remove missing (NA) values from custid and movieid columns
# Project out three columns: custid, movieid and rating
t1 <- datatab[!is.na(datatab$custid) &
  !is.na(datatab$movieid) &
  datatab$activity==1, c("custid", "movieid", "rating")]

# Set HIVE field delimiters to ','. By default, it is Ctrl+a for text files but
# ORCH 2.0 supports only ',' as a file separator.
ore.hiveOptions(delim=',')

# Create another Hive table called "datatab1" after the transformations above.
ore.create (t1, table="datatab1")

# Use the HDFS directory, where the table data for datatab1 is stored, to attach
# it to ORCH framework. By default, this location is "/user/hive/warehouse"
dfs.id <- hdfs.attach("/user/hive/warehouse/datatab1")

# dfs.id can now be used with all hdfs.*, orch.* and hadoop.* APIs of ORCH for
further processing and analytics.
```

Access to Oracle Database

Oracle R Advanced Analytics for Hadoop provides a basic level of database access. You can move the contents of a database table to HDFS, and move the results of HDFS analytics back to the database.

You can then perform additional analysis on this smaller set of data using a separate product named Oracle R Enterprise. It enables you to perform statistical analysis on database tables, views, and other data objects using the R language. You have transparent access to database objects, including support for Business Intelligence and in-database analytics.

Access to the data stored in an Oracle database is always restricted to the access rights granted by your DBA.

Oracle R Enterprise is included in the Oracle Advanced Analytics option to Oracle Database Enterprise Edition. It is not one of the Oracle Big Data Connectors.

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

Usage Notes for Oracle Database Access

Oracle R Advanced Analytics for Hadoop uses Sqoop to move data between HDFS and Oracle Database. Sqoop imposes several limitations on Oracle R Advanced Analytics for Hadoop:

- You cannot import Oracle tables with `BINARY_FLOAT` or `BINARY_DOUBLE` columns. As a work-around, you can create a view that casts these columns to `NUMBER`.
- All column names must be in upper case.

Scenario for Using Oracle R Advanced Analytics for Hadoop with Oracle R Enterprise

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

Using Oracle R Advanced Analytics for Hadoop, you can look for files that you have access to on HDFS and execute R calculations on data in one such file. You can also 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 using `DBMS_SCHEDULER`, and download the results into a local file.

Using Oracle R Enterprise, 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.

Again using Oracle R Advanced Analytics 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 Oracle R Enterprise, in an Oracle database. You can return the output of the calculation to an Oracle database and to the R console for visualization or additional processing.

Analytic Functions in Oracle R Advanced Analytics for Hadoop

Table 8–1 describes the analytic functions. For more information, use R Help.

Table 8–1 Descriptions of the Analytic Functions

Function	Description
<code>orch.evaluate</code>	Evaluates a fit generated by <code>orch.lmf</code> . This information can be helpful when you are tuning the model parameters.
<code>orch.export.fit</code>	Exports a model (W and H factor matrices) to the specified destination for <code>orch.lmf</code> , <code>jellyfish</code> or <code>orch.nmf</code> . It is not used for <code>orch.mahout.lmf.als</code> .
<code>orch.lm</code>	Fits a linear model using tall-and-skinny QR (TSQR) factorization and parallel distribution. The function computes the same statistical parameters as the Oracle R Enterprise <code>ore.lm</code> function.
<code>orch.lmf</code>	Fits a low rank matrix factorization model using either the jellyfish algorithm or the Mahout alternating least squares with weighted regularization (ALS-WR) algorithm.
<code>orch.neural</code>	Provides a neural network to model complex, nonlinear relationships between inputs and outputs, or to find patterns in the data.
<code>orch.nmf</code>	Provides the main entry point to create a nonnegative matrix factorization model using the jellyfish algorithm. This function can work on much larger data sets than the R NMF package, because the input does not need to fit into memory.
<code>orch.nmf.NMFalgo</code>	Plugs in to the R NMF package framework as a custom algorithm. This function is used for benchmark testing.
<code>orch.recommend</code>	Computes the top <i>n</i> items to be recommended for each user that has predicted ratings based on the input <code>orch.mahout.lmf.als</code> model.
<code>predict.orch.lm</code>	Predicts future results based on the fit calculated by <code>orch.lm</code> .
<code>print.orch.lm</code>	Prints a model returned by the <code>orch.lmf</code> function.
<code>print.summary.orch.lm</code>	Prints a summary of the fit calculated by <code>orch.lm</code> .
<code>summary.orch.lm</code>	Prepares a summary of the fit calculated by <code>orch.lm</code> .

ORCH mapred.config Class

The `hadoop.exec` and `hadoop.run` functions have an optional argument, `config`, for configuring the resultant MapReduce job. This argument is an instance of the `mapred.config` class.

The `mapred.config` class has these slots:

hdfs.access

Set to `TRUE` to allow access to the `HDFS.*` functions in the mappers, reducers, and combiners, or set to `FALSE` to restrict access (default).

job.name

A descriptive name for the job so that you can monitor its progress more easily.

map.input

A mapper input data-type keyword: `data.frame`, `list`, or `vector` (default).

map.output

A sample R data frame object that defines the output structure of data from the mappers. It is required only if the mappers change the input data format. Then the reducers require a sample data frame to parse the input stream of data generated by the mappers correctly.

If the mappers output exactly the same records as they receive, then you can omit this option.

map.split

The number of rows that your mapper function receives from a mapper.

- `0` sends all rows given by Hadoop to a specific mapper to your mapper function. Use this setting only if you are sure that the chunk of data for each mapper can fit into R memory. If it does not, then the R process will fail with a memory allocation error.
- `1` sends one row only to the mapper at a time (Default). You can improve the performance of a mapper by increasing this value, which decreases the number of invocations of your mapper function. This is particularly important for small functions.
- `n` sends a minimum of `n` rows to the mapper at a time. In this syntax, `n` is an integer greater than 1. Some algorithms require a minimum number of rows to function.

map.tasks

The number of mappers to run. Specify `1` to run the mappers sequentially; specify a larger integer to run the mappers in parallel. If you do not specify a value, then the number of mappers is determined by the capacity of the cluster, the workload, and the Hadoop configuration.

map.valkey

Set to `TRUE` to duplicate the keys as data values for the mapper, or `FALSE` to use the keys only as keys (default).

min.split.size

Controls the lower boundary for splitting HDFS files before sending them to the mappers. This option reflects the value of the Hadoop `mapred.min.split.size` option and is set in bytes.

reduce.input

A reducer input data type keyword: `data.frame` or `list` (default).

reduce.output

A sample R data frame object that defines the output structure of data from the reducers. This is optional parameter.

The sample data frame is used to generate metadata for the output HDFS objects. It reduces the job execution time by eliminating the sampling and parsing step. It also results in a better output data format, because you can specify column names and other metadata. If you do not specify a sample data frame, then the output HDFS files are sampled and the metadata is generated automatically.

reduce.split

The number of data values given simultaneously to the reducer. See the values for [map.split](#). The reducer expects to receive all values simultaneously, so you must handle partial data sets in the reducer if you set `reduce.split` to a value other than 0 (zero).

reduce.tasks

The number of reducers to run. Specify 1 to run the reducers sequentially; specify a larger integer to run the reducers in parallel. If you do not specify a value, then the number of reducers is determined by the capacity of the cluster, the workload, and the Hadoop configuration.

reduce.valkey

Set to `TRUE` to duplicate the keys as data values for the reducer, or `FALSE` to use the keys only as keys (default).

verbose

Set to `TRUE` to generate diagnostic information, or `FALSE` otherwise.

Examples and Demos of Oracle R Advanced Analytics for Hadoop

The ORCH package includes sample code to help you learn to adapt your R programs to run on a Hadoop cluster using Oracle R Advanced Analytics for Hadoop. This topic describes these examples and demonstrations.

- [Using the Demos](#)
- [Using the Examples](#)

Using the Demos

Oracle R Advanced Analytics for Hadoop provides an extensive set of demos. Instructions for running them are included in the following descriptions.

If an error occurs, then exit from R without saving the workspace image and start a new session. You should also delete the temporary files created in both the local file system and the HDFS file system:

```
# rm -r /tmp/orch*
# hadoop fs -rm -r /tmp/orch*
```

Demo R Programs

hdfs_cpmv.R

Demonstrates copying, moving, and deleting HDFS files and directories.

This program uses the following ORCH functions:

```
hdfs.cd
hdfs.cp
hdfs.exists
hdfs.ls
hdfs.mkdir
hdfs.mv
hdfs.put
hdfs.pwd
hdfs.rmdir
hdfs.root
hdfs.setroot
```

To run this demo, use the following command:

```
R> demo("hdfs_cpmv", package="ORCH")
```

hdfs_datatrans.R

Demonstrates data transfers between HDFS and the local file system, and between HDFS and an Oracle database.

Note: This demo requires that Oracle R Enterprise client is installed on Oracle Big Data Appliance, and Oracle R Enterprise server is installed on the Oracle Database host.

You must connect to Oracle Database using both Oracle R Advanced Analytics for Hadoop and Oracle R Enterprise to run this demo to completion. See [orch.connect](#) on page 9-41 and [ore.connect](#) help.

This program uses the following ORCH functions:

```
hdfs.cd
hdfs.describe
hdfs.download
hdfs.get
hdfs.pull
hdfs.pwd
hdfs.rm
hdfs.root
hdfs.setroot
hdfs.size
hdfs.upload
orch.connected
ore.drop1
ore.is.connected1
```

To run this demo, use the following commands, replacing the parameters shown here for `ore.connect` and `orch.connect` with those appropriate for your Oracle database:

```
R> ore.connect("RUSER", "orcl", "localhost", "welcome1")
Loading required package: ROracle
Loading required package: DBI
R> orch.connect("localhost", "RUSER", "orcl", "welcome1", secure=F)
Connecting ORCH to RDBMS via [sqoop]
  Host: localhost
  Port: 1521
```

¹ Use the R help function for usage information

```

      SID: orcl
      User: RQUSER
Connected.
[1] TRUE
R> demo("hdfs_datatrans", package="ORCH")

```

hdfs_dir.R

Demonstrates the use of functions related to HDFS directories.

This program uses the following ORCH functions:

```

hdfs.cd
hdfs.ls
hdfs.mkdir
hdfs.pwd
hdfs.rmdir
hdfs.root
hdfs.setroot

```

To run this demo, use the following command:

```
R> demo("hdfs_dir", package="ORCH")
```

hdfs_putget.R

Demonstrates how to transfer data between an R data frame and HDFS.

This program uses the following ORCH functions:

```

hdfs.attach
hdfs.describe
hdfs.exists
hdfs.get
hdfs.put
hdfs.pwd
hdfs.rm
hdfs.root
hdfs.sample
hdfs.setroot

```

To run this demo, use the following command:

```
R> demo("hdfs_putget", package="ORCH")
```

hive_aggregate.R

Moves a selection of the Iris data set into Hive and performs these aggregations for each species: summary, mean, minimum, maximum, standard deviation, median, and interquartile range (IQR).

This program uses the following ORCH functions to set up a Hive table. These functions are used in all of the Hive demo programs and are documented in R Help:

```

ore.attach
ore.connect
ore.push

```

To run this demo, use the following command:

```
R> demo("hive_aggregate", package="ORCH")
```

hive_analysis.R

Demonstrates basic analysis and data processing operations.

To run this demo, use the following command:

```
R> demo("hive_analysis", package="ORCH")
```

hive_basic.R

Using basic R functions, this demo obtains information about a Hive table, including the number of rows (`nrow`), the number of columns (`length`), the first five rows (`head`), the class and data type of a column (`class` and `is.*`), and the number of characters in each column value (`nchar`).

To run this program, use the following command:

```
R> demo("hive_basic", package="ORCH")
```

hive_binning.R

Creates bins in Hive based on petal size in the Iris data set.

To run this program, use the following command:

```
R> demo("hive_binning", package="ORCH")
```

hive_columnfns.R

Uses the Iris data set to show the use of several column functions including `min`, `max`, `sd`, `mean`, `fivenum`, `var`, `IQR`, `quantile`, `log`, `log2`, `log10`, `abs`, and `sqrt`.

To run this program, use the following command:

```
R> demo("hive_columnfns", package="ORCH")
```

hive_nulls.R

Demonstrates the differences between handling nulls in R and Hive, using the `AIRQUALITY` data set.

This program uses the following ORCH functions, which are documented in R Help:

```
ore.attach  
ore.connect  
ore.na.extract  
ore.push
```

To run this program, use the following command:

```
R> demo("hive_nulls", package="ORCH")
```

hive_pushpull.R

Shows processing of the Iris data set split between Hive and the client R desktop.

This program uses the following ORCH functions, which are documented in R Help:

```
ore.attach  
ore.connect  
ore.pull  
ore.push
```

To run this program, use the following command:

```
R> demo("hive_pushpull", package="ORCH")
```

hive_sequencefile.R

Shows how to create and use Hive tables stored as sequence files, using the `cars` data set.

This program uses the following ORCH functions, which are documented in R Help:

```
ore.attach  
ore.connect  
ore.create  
ore.exec  
ore.sync
```

```
ore.drop
```

To run this program, use the following command:

```
R> demo("hive_sequencefile", package="ORCH")
```

mapred_basic.R

Provides a simple example of mappers and reducers written in R. It uses the cars data set.

This program uses the following ORCH functions:

```
hadoop.run  
hdfs.get  
hdfs.put  
hdfs.rm  
hdfs.setroot  
orch.keyval
```

To run this program, use the following command:

```
R> demo("mapred_basic", package="ORCH")
```

mapred_modelbuild.R

Runs a mapper and a reducer in a parallel model build, saves plots in HDFS in parallel, and displays the graphs. It uses the iris data set.

This program uses the following ORCH functions:

```
hadoop.run  
hdfs.download  
hdfs.exists  
hdfs.get  
hdfs.id  
hdfs.put  
hdfs.rm  
hdfs.rmdir  
hdfs.root  
hdfs.setroot  
hdfs.upload  
orch.export  
orch.pack  
orch.unpack
```

To run this program, use the following command:

```
R> demo("mapred_modelbuild", package="ORCH")
```

orch_lm.R

Fits a linear model using the `orch.lm` algorithm and compares it with the R `lm` algorithm. It uses the iris data set.

This program uses the following ORCH functions:

```
hdfs.cd  
hdfs.get  
hdfs.put  
hdfs.pwd  
hdfs.rm  
hdfs.root  
hdfs.setroot  
orch.lm1  
predict.orch.lm1
```

To run this program, use the following command:

```
R> demo("orch_lm", package="ORCH")
```

orch_lm_jellyfish.R

Fits a low rank matrix factorization model using the jellyfish algorithm.

This program uses the following ORCH functions:

```
hdfs.cd  
hdfs.get  
hdfs.mkdir  
hdfs.pwd  
hdfs.rmdir  
hdfs.root  
hdfs.setroot  
orch.evaluate1  
orch.export.fit1  
orch.lmf1
```

To run this program, use the following command:

```
R> demo("orch_lm_jellyfish", package="ORCH")
```

orch_lm_mahout_als.R

Fits a low rank matrix factorization model using the Mahout ALS-WR algorithm.

This program uses the following ORCH functions:

```
hdfs.cd  
hdfs.mkdir  
hdfs.pwd  
hdfs.rmdir  
hdfs.root  
orch.evaluate1  
orch.lmf1  
orch.recommend1
```

To run this program, use the following command:

```
R> demo("orch_lm_mahout_als", package="ORCH")
```

orch_neural.R

Builds a model using the Oracle R Advanced Analytics for Hadoop neural network algorithm. It uses the `iris` data set.

This program uses the following ORCH functions:

```
hdfs.cd  
hdfs.put  
hdfs.pwd  
hdfs.rm  
hdfs.root  
hdfs.sample  
hdfs.setroot  
orch.neural1
```

To run this program, use the following command:

```
R> demo("orch_neural", package="ORCH")
```

orch_nmf.R

Demonstrates the integration of the `orch.nmf` function with the framework of the R NMF package to create a nonnegative matrix factorization model.

This demo uses the Golub data set provided with the NMF package.

This program uses the following ORCH functions:

```
hdfs.cd
hdfs.mkdir
hdfs.pwd
hdfs.rmdir
hdfs.root
hdfs.setroot
orch.nmf.NMFalgo1
```

This program requires several R packages that you may not have already installed: NMF, Biobase, BiocGenerics, and RColorBrewer. Instructions for installing them are provided here.

Caution: When asked whether you want to update the dependent packages, always respond **Update/None**. The installed R packages are the supported versions. If you update them, then the Oracle Big Data Appliance software checks fail, which causes problems during routine maintenance. Always run the `bdchecksw` utility after installing new packages. See the `bdchecksw` utility in the *Oracle Big Data Appliance Owner's Guide*.

If you are using a Hadoop client, then install the packages on the client instead of Oracle Big Data Appliance.

To install NMF:

1. Download NMF from the CRAN archives at
http://cran.r-project.org/src/contrib/Archive/NMF/NMF_0.5.06.tar.gz
2. Install NMF using a standard installation method:

```
R> install.packages("/full_path/NMF_0.5.06.tar.gz", REPOS=null)
```

To install Biobase and BiocGenerics from the BioConductor project:

1. Source in the `biocLite` package:

```
R> source("http://bioconductor.org/biocLite.R")
```
2. Install the Biobase package:

```
R> biocLite("Biobase")
```
3. Install the BiocGenerics package:

```
R> biocLite("BiocGenerics")
```

To install RColorBrewer:

- Use a standard method of installing a package from CRAN:

```
R> install.packages("RColorBrewer")
```

To run the `orch_nmf` demo:

1. Load the required packages, if they are not already loaded in this session:

```
R> library("NMF")
R> library("Biobase")
R> library("BiocGenerics")
R> library("RColorBrewer")
```

2. Run the demo:

```
R> demo("orch_nmf", package="ORCH")
```

demo-bagged.clust.R

Provides an example of bagged clustering using randomly generated values. The mappers perform k-means clustering analysis on a subset of the data and generate centroids for the reducers. The reducers combine the centroids into a hierarchical cluster and store the `hclust` object in HDFS.

This program uses the following ORCH functions:

```
hadoop.exec
hdfs.put
is.hdfs.id
orch.export
orch.keyval
orch.pack
orch.unpack
```

To run this program, ensure that `root` is set to `/tmp`, and then source the file:

```
R> hdfs.setroot("/tmp")
[1] "/tmp"
R> source("/usr/lib64/R/library/ORCHcore/demos/demo-bagged.clust.R")
```

```
Call:
c("hclust", "d", "single")
```

```
Cluster method : single
Distance       : euclidean
Number of objects: 6
```

demo-kmeans.R

Performs k-means clustering.

This program uses the following ORCH functions:

```
hadoop.exec
hdfs.get
orch.export
orch.keyval
orch.pack
orch.unpack
```

To run this program, ensure that `root` is set to `/tmp`, and then source the file:

```
R> hdfs.setroot("/tmp")
[1] "/tmp"
R> source("/usr/lib64/R/library/ORCHcore/demos/demo-kmeans.R")
```

```
centers
      x      y
1 0.3094056 1.32792762
2 1.7374920 -0.06730981
3 -0.6271576 -1.20558920
4 0.2668979 -0.98279426
```

```

5 0.4961453 1.11632868
6 -1.7328809 -1.26335598
removing NaNs if any
final centroids
      x      y
1 0.666782828 2.36620810
2 1.658441962 0.33922811
3 -0.627157587 -1.20558920
4 -0.007995672 -0.03166983
5 1.334578589 1.41213532
6 -1.732880933 -1.26335598

```

Using the Examples

The examples show how you use the Oracle R Advanced Analytics for Hadoop API. You can view them in a text editor after extracting them from the installation archive files. They are located in the ORCH2.1.0/ORCHcore/examples directory. You can run the examples as shown in the following descriptions.

Example R Programs

example-filter1.R

Shows how to use key-value pairs. The mapper function selects cars with a distance value greater than 30 from the `cars` data set, and the reducer function calculates the mean distance for each speed.

This program uses the following ORCH functions:

```

hadoop.run
hdfs.get
hdfs.put
orch.keyval

```

To run this program, ensure that `root` is set to `/tmp`, and then source the file:

```

R> hdfs.setroot("/tmp")
[1] "/tmp"
R> source("/usr/lib64/R/library/ORCHcore/examples/example-filter1.R")
Running cars filtering and mean example #1:
  val1  val2
1    10 34.00000
2    13 38.00000
3    14 58.66667
4    15 54.00000
5    16 36.00000
6    17 40.66667
7    18 64.50000
8    19 50.00000
9    20 50.40000
10   22 66.00000
11   23 54.00000
12   24 93.75000
13   25 85.00000

```

example-filter2.R

Shows how to use values only. The mapper function selects cars with a distance greater than 30 and a speed greater than 14 from the `cars` data set, and the reducer function calculates the mean speed and distance as one value pair.

This program uses the following ORCH functions:

```
hadoop.run  
hdfs.get  
hdfs.put  
orch.keyval
```

To run this program, ensure that `root` is set to `/tmp`, and then source the file:

```
R> hdfs.setroot("/tmp")  
[1] "/tmp"  
R> source("/usr/lib64/R/library/ORCHcore/examples/example-filter2.R")  
Running cars filtering and mean example #2:  
  val1 val2  
1 59.52 19.72
```

example-filter3.R

Shows how to load a local file into HDFS. The mapper and reducer functions are the same as [example-filter2.R](#).

This program uses the following ORCH functions:

```
hadoop.run  
hdfs.download  
hdfs.upload  
orch.keyval
```

To run this program, ensure that `root` is set to `/tmp`, and then source the file:

```
R> hdfs.setroot("/tmp")  
[1] "/tmp"  
R> source("/usr/lib64/R/library/ORCHcore/examples/example-filter3.R")  
Running cars filtering and mean example #3:  
[1] "\t59.52,19.72"
```

example-group.apply.R

Shows how to build a parallel model and generate a graph. The mapper partitions the data based on the petal lengths in the `iris` data set, and the reducer uses basic R statistics and graphics functions to fit the data into a linear model and plot a graph.

This program uses the following ORCH functions:

```
hadoop.run  
hdfs.download  
hdfs.exists  
hdfs.get  
hdfs.id  
hdfs.mkdir  
hdfs.put  
hdfs.rmdir  
hdfs.upload  
orch.pack  
orch.export  
orch.unpack
```

To run this program, ensure that `root` is set to `/tmp`, and then source the file:

```
R> hdfs.setroot("/tmp")  
[1] "/tmp"  
R> source("/usr/lib64/R/library/ORCHcore/examples/example-group.apply.R")  
Running groupapply example.  
[[1]]  
[[1]]$predict  
  .  
  .
```

```

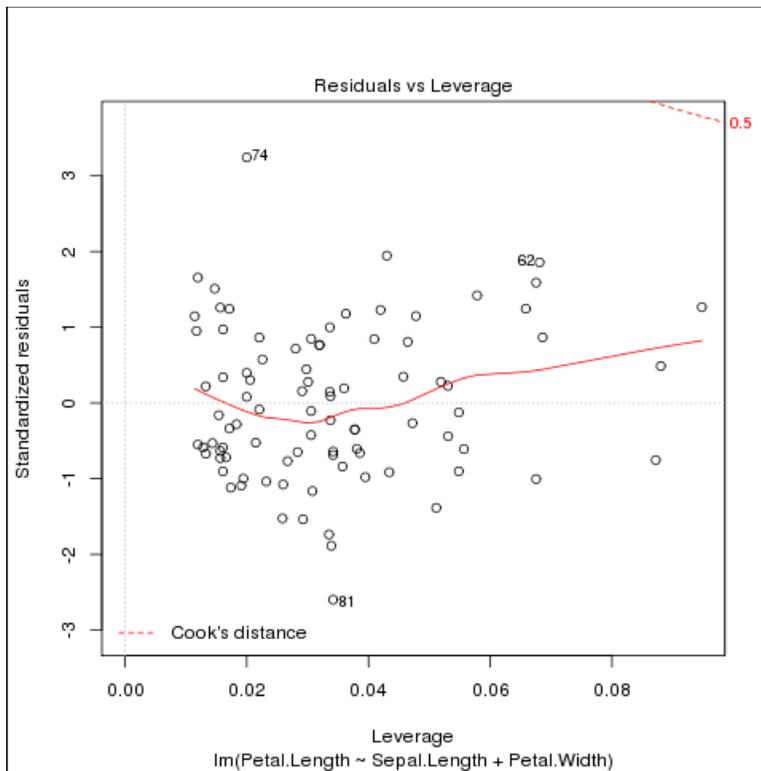
[3]
[[3]]$predict
      2      210      3      4      5      6      7      8
4.883662 5.130816 4.854156 5.163097 4.109921 4.915944 4.233498 4.886437
      9      10      11      12      13      14      15      16
4.789593 4.545214 4.136653 4.574721 4.945451 4.359849 4.013077 4.730579
      17      18      19      20      21      22      23      24
4.827423 4.480651 4.792367 4.386581 4.666016 5.007239 5.227660 4.607002
      25      26      27      28      29      30      31      32
4.701072 4.236272 4.701072 5.039521 4.604228 4.171709 4.109921 4.015851
      33      34      35      36      37      38      39      40
4.574721 4.201216 4.171709 4.139428 4.233498 4.542439 4.233498 5.733065
      41      42      43      44      45      46      47      48
4.859705 5.851093 5.074577 5.574432 6.160034 4.115470 5.692460 5.321730
      49      50      51      52      53      54      55      56
6.289160 5.386293 5.230435 5.665728 4.891986 5.330054 5.606714 5.198153
      57      58      59      60      61      62      63      64
6.315892 6.409962 4.607002 5.915656 4.830198 6.127753 5.074577 5.603939
      65      66      67      68      69      70      71      72
5.630671 5.012788 4.951000 5.418574 5.442532 5.848318 6.251329 5.512644
      73      74      75      76      77      78      79      80
4.792367 4.574721 6.409962 5.638995 5.136365 4.889212 5.727516 5.886149
      81      82      83      84      85      86      87      88
5.915656 4.859705 5.853867 5.980218 5.792079 5.168646 5.386293 5.483137
      89
4.827423

[[3]]$pngfile
[1] "/user/oracle/pngfiles/3"

[1] "/tmp/orch6e295a5a5da9"

```

This program generates three graphics files in the /tmp directory. [Figure 8–1](#) shows the last one.

Figure 8–1 Example example-group.apply.R Output in fit-3.png**example-kmeans.R**

Defines a k-means clustering function and generates random points for a clustering test. The results are printed or graphed.

This program uses the following ORCH functions:

```
hadoop.exec
hdfs.get
hdfs.put
orch.export
```

To run this program, ensure that root is set to /tmp, and then source the file:

```
R> hdfs.setroot("/tmp")
[1] "/tmp"
R> source("/usr/lib64/R/library/ORCHcore/examples/example-kmeans.R")
Running k-means example.
      val1      val2
1 1.005255389 1.9247858
2 0.008390976 2.5178661
3 1.999845464 0.4918541
4 0.480725254 0.4872837
5 1.677254045 2.6600670
```

example-lm.R

Shows how to define multiple mappers and one reducer that merges all results. The program calculates a linear regression using the iris data set.

This program uses the following ORCH functions:

```
hadoop.run
hdfs.get
hdfs.put
```

```
orch.export
orch.pack
orch.unpack
```

To run this program, ensure that root is set to /tmp, and then source the file:

```
R> hdfs.setroot("/tmp")
[1] "/tmp"
R> source("/usr/lib64/R/library/ORCHcore/examples/example-lm.R")
Running linear model example.
Model rank 3, yy 3.0233000000E+02, nRows 150
Model coefficients
-0.2456051 0.2040508 0.5355216
```

example-logreg.R

Performs a one-dimensional, logistic regression on the cars data set.

This program uses the following ORCH functions:

```
hadoop.run
hdfs.put
orch.export
```

To run this program, ensure that root is set to /tmp, and then source the file:

```
R> hdfs.setroot("/tmp")
[1] "/tmp"
R> source("/usr/lib64/R/library/ORCHcore/examples/example-logreg.R")
Running logistic regression.
[1] 1924.1
```

example-map.df.R

Shows how to run the mapper with an unlimited number of records input simultaneously as a data frame. The mapper selects cars with a distance greater than 30 from the cars data set and calculates the mean distance. The reducer merges the results.

This program uses the following ORCH functions:

```
hadoop.run
hdfs.get
hdfs.put
```

To run this program, ensure that root is set to /tmp, and then source the file:

```
R> hdfs.setroot("/tmp")
[1] "/tmp"
R> source("/usr/lib64/R/library/ORCHcore/examples/example-map.df.R")
Running example of data.frame mapper input:
      val1    val2
1 17.66667 50.16667
2 13.25000 47.25000
```

example-map.list.R

Shows how to run the mapper with an unlimited number of records input simultaneously as a list. The mapper selects cars with a distance greater than 30 from the cars data set and calculates the mean distance. The reducer merges the results.

This program uses the following ORCH functions:

```
hadoop.run
hdfs.get
hdfs.put
```

To run this program, ensure that root is set to /tmp, and then source the file:

```
R> hdfs.setroot("/tmp")
[1] "/tmp"
R> source("/usr/lib64/R/library/ORCHcore/examples/example-map.list.R")
Running example of list mapper input:
  val1 val2
1 15.9  49
```

example-model.plot.R

Shows how to create models and graphs using HDFS. The mapper provides key-value pairs from the *iris* data set to the reducer. The reducer creates a linear model from data extracted from the data set, plots the results, and saves them in three HDFS files.

This program uses the following ORCH functions:

```
hadoop.run
hdfs.download
hdfs.exists
hdfs.get
hdfs.id
hdfs.mkdir
hdfs.put
hdfs.rmdir
hdfs.upload
orch.export
orch.pack
```

To run this program, ensure that root is set to /tmp, and then source the file:

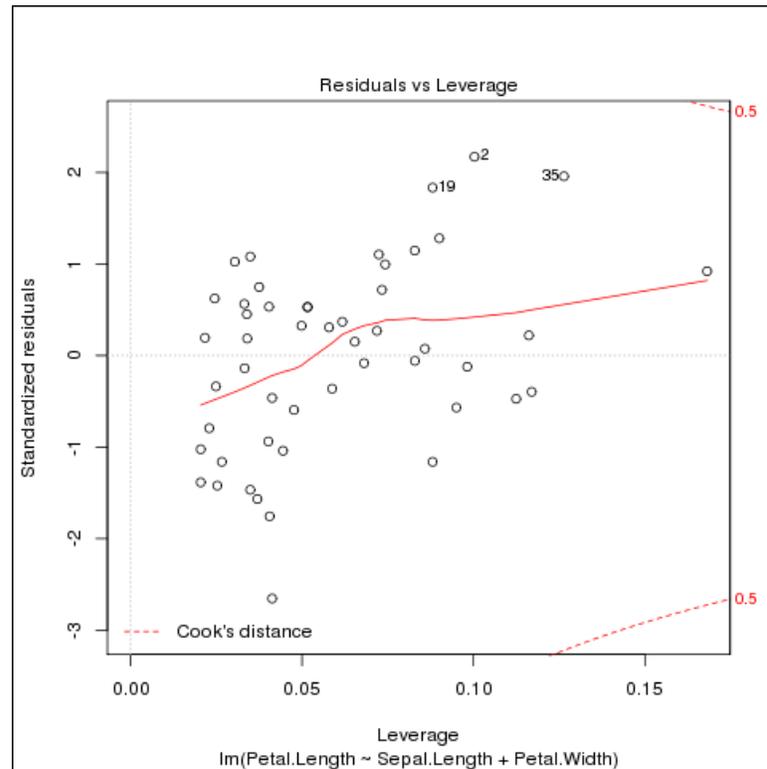
```
R> hdfs.setroot("/tmp")
[1] "/tmp"
R> source("/usr/lib64/R/library/ORCHcore/examples/example-model.plot.R")
Running model building and graph example.
[[1]]
[[1]]$predict
  .
  .
  .
[[3]]$predict
   2    210    3    4    5    6    7    8
5.423893 4.955698 5.938086 5.302979 5.517894 6.302640 4.264954 6.032087
   9   10   11   12   13   14   15   16
5.594622 6.080090 5.483347 5.393163 5.719353 4.900061 5.042065 5.462257
  17   18   19   20   21   22   23   24
5.448801 6.392824 6.410097 5.032427 5.826811 4.827150 6.358277 5.302979
  25   26   27   28   29   30   31   32
5.646442 5.959176 5.230068 5.157157 5.427710 5.924630 6.122271 6.504099
  33   34   35   36   37   38   39   40
5.444983 5.251159 5.088064 6.410097 5.406619 5.375890 5.084247 5.792264
  41   42   43   44   45   46   47   48
5.698262 5.826811 4.955698 5.753900 5.715536 5.680989 5.320252 5.483347
  49   50
5.316435 5.011336

[[3]]$pngfile
[1] "/user/oracle/pngfiles/virginica"

[1] "/tmp/orch6e29190de160"
```

The program generates three graphics files. [Figure 8–2](#) shows the last one.

Figure 8–2 Example `example-model.plot.R` Output in `fit-virginica.png`



`example-model.prep.R`

Shows how to distribute data across several map tasks. The mapper generates a data frame from a slice of input data from the `iris` data set. The reducer merges the data frames into one output data set.

This program uses the following ORCH functions:

```
hadoop.exec
hdfs.get
hdfs.put
orch.export
orch.keyvals
```

To run this program, ensure that `root` is set to `/tmp`, and then source the file:

```
R> hdfs.setroot("/tmp")
[1] "/tmp"
R> source("/usr/lib64/R/library/ORCHcore/examples/example-model.prep.R")
  v1    v2      v3      v4
1  5.1  4.90  4.055200 0.33647224
2  4.9  4.20  4.055200 0.33647224
3  4.7  4.16  3.669297 0.26236426
.
.
.
299 6.2 18.36 221.406416 1.68639895
300 5.9 15.30 164.021907 1.62924054
```

example-rlm.R

Shows how to convert a simple R program into one that can run as a MapReduce job on a Hadoop cluster. In this example, the program calculates and graphs a linear model on the cars data set using basic R functions.

This program uses the following ORCH functions:

```
hadoop.run
orch.keyvals
orch.unpack
```

To run this program, ensure that root is set to /tmp, and then source the file:

```
R> hdfs.setroot("/tmp")
[1] "/tmp"
R> source("/usr/lib64/R/library/ORCHcore/examples/example-rlm.R")
[1] "--- Client lm:"
```

Call:

```
lm(formula = speed ~ dist, data = cars)
```

Residuals:

```
      Min       1Q   Median       3Q      Max
-7.5293 -2.1550  0.3615  2.4377  6.4179
```

Coefficients:

```
              Estimate Std. Error t value Pr(>|t|)
(Intercept)  8.28391    0.87438   9.474 1.44e-12 ***
dist         0.16557    0.01749   9.464 1.49e-12 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

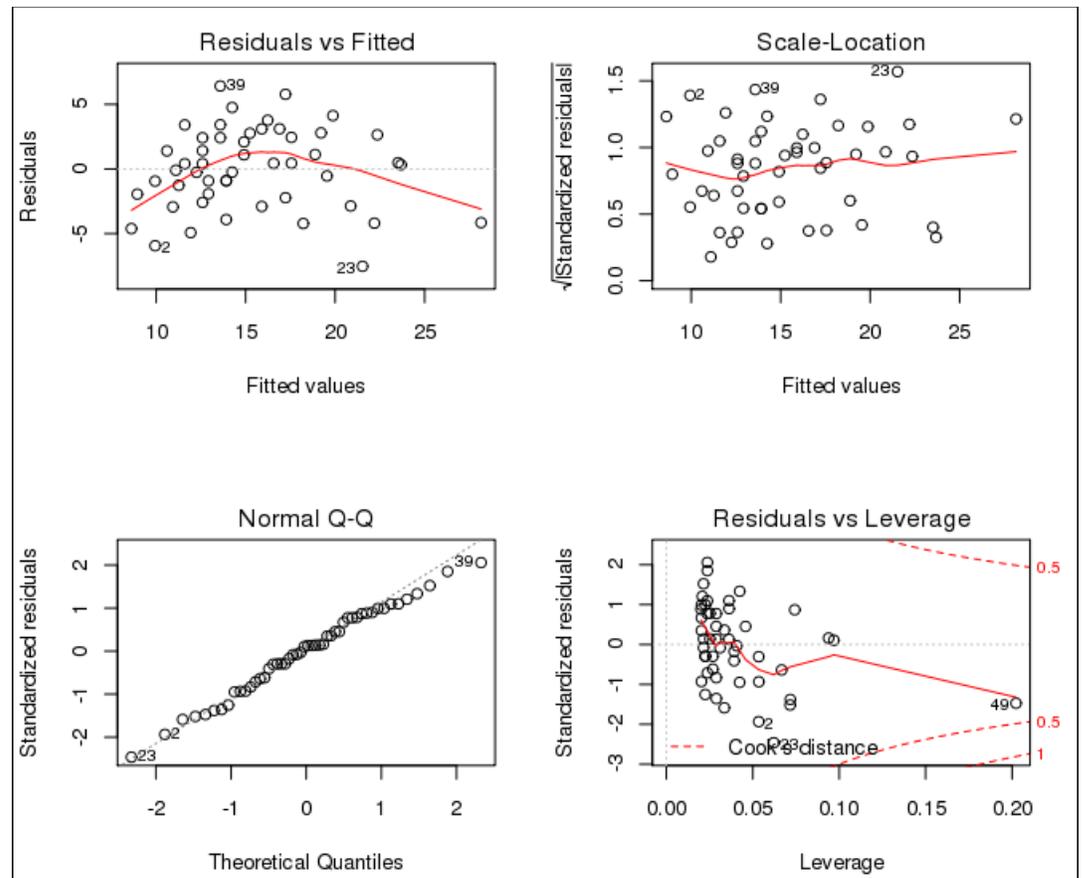
```
Residual standard error: 3.156 on 48 degrees of freedom
Multiple R-squared:  0.6511,    Adjusted R-squared:  0.6438
F-statistic: 89.57 on 1 and 48 DF,  p-value: 1.49e-12
```

Enter Ctrl+C to get the R prompt, and then close the graphics window:

```
R> dev.off()
```

Figure 8–3 shows the four graphs generated by the program.

Figure 8-3 Example-rlm.R Output

**example-split.map.R**

Shows how to split the data in the mapper. The first job runs the mapper in list mode and splits the list in the mapper. The second job splits a data frame in the mapper. Both jobs use the cars data set.

This program uses the following ORCH functions:

```
hadoop.run
hdfs.get
hdfs.put
orch.keyval
```

To run this program, ensure that root is set to /tmp, and then source the file:

```
R> hdfs.setroot("/tmp")
[1] "/tmp"
R> source("/usr/lib64/R/library/ORCHcore/examples/example-split.map.R")
Running example of list splitting in mapper
  key count splits
1 NA    50     6
Running example of data.frame splitting in mapper
  key count splits
1 NA    50     8
```

example-split.reduce.R

Shows how to split the data from the cars data set in the reducer.

This program uses the following ORCH functions:

```
hadoop.run
hdfs.get
hdfs.put
orch.keyval
```

To run this program, ensure that root is set to /tmp, and then source the file:

```
R> hdfs.setroot("/tmp")
[1] "/tmp"
R> source("/usr/lib64/R/library/ORCHcore/examples/example-split.reduce.R")
Running example of data.frame reducer input
^C  key count
1   1     9
2   1     9
3   1     9
.
.
.
20  1     9
21  1     5
```

example-sum.R

Shows how to perform a sum operation in a MapReduce job. The first job sums a vector of numeric values, and the second job sums all columns of a data frame.

This program uses the following ORCH functions:

```
hadoop.run
orch.keyval
```

To run this program, ensure that root is set to /tmp, and then source the file:

```
R> hdfs.setroot("/tmp")
[1] "/tmp"
R> source("/usr/lib64/R/library/ORCHcore/examples/example-sum.R")
val2
1   6
val2 val3
1  770 2149
```

example-teragen.matrix.R

Shows how to generate large data sets in a matrix for testing programs in Hadoop. The mappers generate samples of random data, and the reducers merge them.

This program uses the following ORCH functions:

```
hadoop.run
hdfs.put
orch.export
orch.keyvals
```

To run this program, ensure that root is set to /tmp, and then source the file. The program runs without printing any output.

```
R> hdfs.setroot("/tmp")
[1] "/tmp"
R> source("/usr/lib64/R/library/ORCHcore/examples/example-teragen.matrix.R")
Running TeraGen-PCA example:
R>
```

example-teragen.xy.R

Shows how to generate large data sets in a data frame for testing programs in Hadoop. The mappers generate samples of random data, and the reducers merge them.

This program uses the following ORCH functions:

```
hadoop.run
hdfs.put
orch.export
orch.keyvals
```

To run this program, ensure that `root` is set to `/tmp`, and then source the file. The program runs without printing any output.

```
R> hdfs.setroot("/tmp")
[1] "/tmp"
R> source("/usr/lib64/R/library/ORCHcore/examples/example-teragen2.xy.R")
Running TeraGen2 example.R>
```

example-teragen2.xy.R

Shows how to generate large data sets in a data frame for testing programs in Hadoop. One mapper generates small samples of random data, and the reducers merge them.

This program uses the following ORCH functions:

```
hadoop.run
hdfs.put
orch.export
orch.keyvals
```

To run this program, ensure that `root` is set to `/tmp`, and then source the file. The program runs without printing any output.

```
R> hdfs.setroot("/tmp")
[1] "/tmp"
R> source("/usr/lib64/R/library/ORCHcore/examples/example-teragen2.xy.R")
Running TeraGen2 example.
R>
```

example-terasort.R

Provides an example of a TeraSort job on a set of randomly generated values.

This program uses the following ORCH function:

```
hadoop.run
```

To run this program, ensure that `root` is set to `/tmp`, and then source the file:

```
R> hdfs.setroot("/tmp")
[1] "/tmp"
R> source("/usr/lib64/R/library/ORCHcore/examples/example-terasort.R")
Running TeraSort example:
      vall
1 -0.001467344
2 -0.004471376
3 -0.005928546
4 -0.007001193
5 -0.010587280
6 -0.011636190
```

Security Notes for Oracle R Advanced Analytics for Hadoop

Oracle R Advanced Analytics for Hadoop can invoke 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. The name Sqoop comes from "SQL to Hadoop." The following explains how Oracle R Advanced Analytics for Hadoop stores a database user password and sends it to Sqoop.

Oracle R Advanced Analytics 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 [orch.connect](#) on page 9-41.

Oracle R Advanced Analytics 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, and 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 are 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.

ORCH Library Reference

This chapter describes the function in the ORCH library. It contains the following sections:

- [Functions in Alphabetical Order](#)
- [Functions by Category](#)

Functions in Alphabetical Order

Use R help for the `ore` functions for Apache Hive and the analytic functions:

```
R> help("function_name")
```

There are no Help (Rd) files for orch functions. They are described in this chapter.

```
as.ore
as.ore.character
as.ore.frame
as.ore.integer
as.ore.logical
as.ore.numeric
as.ore.vector
hadoop.exec
hadoop.run
hdfs.attach
hdfs.cd
hdfs.cp
hdfs.describe
hdfs.download
hdfs.exists
hdfs.get
hdfs.head
hdfs.id
hdfs.ls
hdfs.mkdir
hdfs.mv
hdfs.parts
hdfs.pull
hdfs.push
hdfs.put
hdfs.pwd
hdfs.rm
hdfs.rmdir
hdfs.root
hdfs.sample
hdfs.setroot
```

[hdfs.size](#)
[hdfs.tail](#)
[hdfs.upload](#)
[is.hdfs.id](#)
[is.ore](#)
[is.ore.character](#)
[is.ore.frame](#)
[is.ore.integer](#)
[is.ore.logical](#)
[is.ore.numeric](#)
[is.ore.vector](#)
[ore.create](#)
[ore.drop](#)
[ore.get](#)
[ore.pull](#)
[ore.push](#)
[ore.recode](#)
[orch.connect](#)
[orch.connected](#)
[orch.dbcon](#)
[orch.dbg.lasterr](#)
[orch.dbg.off](#)
[orch.dbg.on](#)
[orch.dbg.output](#)
[orch.dbinfo](#)
[orch.disconnect](#)
[orch.dryrun](#)
[orch.evaluate](#)
[orch.export](#)
[orch.export.fit](#)
[orch.keyval](#)
[orch.keyvals](#)
[orch.lm](#)
[orch.lmf](#)
[orch.neural](#)
[orch.nmf](#)
[orch.nmf.NMFalgo](#)
[orch.pack](#)
[orch.reconnect](#)
[orch.temp.path](#)
[orch.unpack](#)
[orch.version](#)
[predict.orch.lm](#)
[print.orch.lm](#)
[print.summary.orch.lm](#)
[summary.orch.lm](#)

Functions by Category

The functions are grouped into these categories:

- [Making Connections](#)
- [Copying Data](#)
- [Exploring Files](#)
- [Writing MapReduce Functions](#)
- [Debugging Scripts](#)
- [Using Hive Data](#)

- Writing Analytical Functions

Making Connections

```
orch.connect  
orch.connected  
orch.dbcon  
orch.dbinfo  
orch.disconnect  
orch.reconnect
```

Copying Data

```
hdfs.attach  
hdfs.cp  
hdfs.download  
hdfs.get  
hdfs.mv  
hdfs.pull  
hdfs.push  
hdfs.put  
hdfs.upload  
orch.export  
orch.pack  
orch.unpack
```

Exploring Files

```
hdfs.cd  
hdfs.describe  
hdfs.exists  
hdfs.head  
hdfs.id  
hdfs.ls  
hdfs.mkdir  
hdfs.parts  
hdfs.pwd  
hdfs.rm  
hdfs.rmdir  
hdfs.root  
hdfs.sample  
hdfs.setroot  
hdfs.size  
hdfs.tail  
is.hdfs.id  
orch.temp.path
```

Writing MapReduce Functions

```
hadoop.exec  
hadoop.run  
orch.dryrun  
orch.keyval  
orch.keyvals
```

Debugging Scripts

```
orch.dbg.lasterr  
orch.dbg.off
```

`orch.dbg.on`
`orch.dbg.output`
`orch.version`

Using Hive Data

See "[ORE Functions for Hive](#)" on page 8-3

Writing Analytical Functions

See "[Analytic Functions in Oracle R Advanced Analytics for Hadoop](#)" on page 8-9.

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,  
  init,  
  final,  
  job.name,  
  config)
```

Arguments

dfs.id

The name of a file in HDFS containing data to be processed. The file name can include a path that is either absolute or relative to the current path.

mapper

Name of a mapper function written in the R language.

reducer

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

combiner

Not supported in this release.

export

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

init

A function that is executed once before the mapper function begins (optional).

final

A function that is executed once after the reducer function completes (optional).

job.name

A descriptive name that you can use to track the progress of the MapReduce job instead of the automatically generated job name (optional).

config

Sets the configuration parameters for the MapReduce job (optional).

This argument is an instance of the `mapred.config` class, and thus it has this format:

```
config = new("mapred.config", param1, param2,...)
```

See "[ORCH mapred.config Class](#)" on page 8-10 for a description of this class.

Usage Notes

Oracle R Advanced Analytics for Hadoop does not support mapper-only jobs. Use [orch.keyvals](#) as a reducer body. See the example in [orch.keyvals](#).

This function provides more control of the data flow than the [hadoop.run](#) function. 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 files.

Return Value

Data object identifier in HDFS

See Also

[hadoop.run](#) on page 9-8, [orch.dryrun](#) on page 9-54, [orch.keyvals](#) on page 9-57

Example

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

The MapReduce function counts the number of on-time flights arriving in the San Francisco International Airport (SFO).

```
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 `res` variable identifies the location of the results in an HDFS file named `/user/oracle/xq/orch3d0b8218`:

```
R> res
[1] "/user/oracle/xq/orch3d0b8218"
attr(,"dfs.id")
[1] TRUE
R> print(hdfs.get(res))
  val1    val2
1 SFO 27.05804
```

This code fragment is extracted from [example-kmeans.R](#). The `export` option identifies the location of the `ncenters` generated data set, which is exported as an HDFS file. The `config` options provide a MapReduce job name of `k-means.1`, and the mapper output format of a data frame.

```
mapf <- data.frame(key=0, val1=0, val2=0)
dfs.points <- hdfs.put(points)
dfs.centers <- hadoop.exec(
  dfs.id = dfs.points,
  mapper = function(k,v) {
    keyval(sample(1:ncenters,1), v)
  },
  reducer = function(k,vv) {
    vv <- sapply(vv, unlist)
    keyval(NULL, c(mean(vv[1,]), mean(vv[2,])))
  },
  export = orch.export(ncenters),
  config = new("mapred.config",
    job.name = "k-means.1",
    map.output = mapf)
```

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,  
  init,  
  final,  
  job.name,  
  config)
```

Arguments

data

A data frame, Oracle R Enterprise frame (`ore.frame`), or an HDFS file name.

mapper

The name of a mapper function written in the R language.

reducer

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

combiner

Not supported in this release.

export

The names of exported R objects.

init

A function that is executed once before the mapper function begins (optional).

final

A function that is executed once after the reducer function completes (optional).

job.name

A descriptive name that you can use to track the progress of the job instead of the automatically generated job name (optional).

config

Sets the configuration parameters for the MapReduce job (optional).

This argument is an instance of the `mapred.config` class, and so it has this format:

```
config = new("mapred.config", param1, param2, ...)
```

See "[ORCH mapred.config Class](#)" on page 8-10 for a description of this class.

Usage Notes

Oracle R Advanced Analytics for Hadoop does not support mapper-only jobs. Use [orch.keyvals](#) as a reducer body. See the example in [orch.keyvals](#).

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 copied into an Oracle database.

Return Value

An object in the same format as the input data

See Also

[hadoop.exec](#) on page 9-5, [orch.dryrun](#) on page 9-54, [orch.keyvals](#) on page 9-57

Example

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

The MapReduce function counts the number of on-time flights arriving in the San Francisco International Airport (SFO).

```
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/orch3d0b8218`:

```
R> res
[1] "/user/oracle/xq/orch3d0b8218"
attr(,"dfs.id")
[1] TRUE
R> print(hdfs.get(res))
  val1    val2
1 SFO 27.05804
```

The next example shows o

```
hadoop.run(x,
  mapper = function(k,v) {
    orch.keyval(k, v+1) # increment all values
  },
```

```
reducer = function(k, vv) {  
    orch.keyvals(k, vv) # pass-through  
}
```

hdfs.attach

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

Usage

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

Arguments

dfs.name

The name of a file in HDFS.

force

Controls whether the function attempts to discover the structure of the file and the data type of each column.

FALSE for comma-separated value (CSV) files (default). If a file does not have metadata identifying the names and data types of the columns, then the function samples the data to deduce the data type as number or string. It then re-creates the file with the appropriate metadata.

TRUE for non-CVS files, including binary files. This setting prevents the function from trying to discover the metadata; instead, it simply attaches the file.

Usage Notes

Use this function to attach a CSV file to your R environment, just as you might attach a data frame.

Oracle R Advanced Analytics for Hadoop does not support the processing of attached non-CVS files. Nonetheless, you can attach a non-CVS file, download it to your local computer, and use it as desired. Alternatively, you can attach the file for use as input to a Hadoop application.

This function can become slow when processing large input HDFS files, as the result of inherited limitations in the Hadoop command-line interface.

Return Value

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

See Also

[hdfs.download](#) on page 9-16

Example

This example stores the object ID of `ontime_R` in a variable named `dfs`, and 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.cp

Copies an HDFS file from one location to another.

Usage

```
hdfs.cp(  
    dfs.src,  
    dfs.dst,  
    force)
```

Arguments

dfs.src

The name of the source file to be copied. The file name can include a path that is either absolute or relative to the current path.

dfs.dst

The name of the copied file. The file name can include a path that is either absolute or relative to the current path.

force

Set to `TRUE` to overwrite an existing file, or set to `FALSE` to display an error message (default).

Return Value

`NULL` for a successful copy, or `FALSE` for a failed attempt

Example

This example copies a file named `weblog` in the parent directory and overwrites the existing `weblog` file:

```
R> hdfs.cp("weblog", "..", force=T)
```

hdfs.describe

Returns the metadata associated with a file in HDFS.

Usage

```
hdfs.describe(
  dfs.id)
```

Arguments

dfs.id

The name of a file in HDFS. The file name can include a path that is either absolute or relative to the current path.

Return Value

A data frame containing the metadata, or `NULL` if no metadata was available in HDFS

Example

This example provides information about an HDFS file named `ontime_DB`:

```
R> hdfs.describe('ontime_DB')
      name
1      path
2      origin
3      class
4      types
5      dim
6      names
7      has.key
8      key.column
9      null.key
10     has.rownames
11     size
12     parts

      values
2      ontime_DB
3      unknown
.
.
.
```

hdfs.download

Copies a file from HDFS to the local file system.

Usage

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

Arguments

dfs.id

The name of a file in HDFS. The file name can include a path that is either absolute or relative to the current path.

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 a file exists in HDFS.

Usage

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

Arguments

dfs.id

The name of a file in HDFS. The file name can include a path that is either absolute or relative to the current path.

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

See Also

[is.hdfs.id](#) on page 9-40

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 name of a file in HDFS. The file name can include a path that is either absolute or relative to the current path.

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 the `hdfs.pull` function. 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))  
  val1      val2  
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.head

Copies a specified number of lines from the beginning of a file in HDFS.

Usage

```
hdfs.head(
  dfs.id,
  n)
```

Arguments

dfs.id

The name of a file in HDFS. The file name can include a path that is either absolute or relative to the current path.

n

The number of lines to retrieve from the file. The default is 10.

Return Value

The first *n* lines of the file.

See Also

[hdfs.tail](#) on page 9-37

Example

This example returns the first two lines of `ontime_R`:

```
R> hdfs.head('ontime_R', 2)
[1]
"\", \"YEAR\", \"MONTH\", \"MONTH2\", \"DAYOFMONTH\", \"DAYOFMONTH2\", \"DAYOFWEEK\", \"
DEPTIME\", \"CRSDEPTIME\", \"ARRTIME\", \"CRSARRTIME\", \"UNIQUECARRIER\", \"FLIGHTNUM
\", \"TAILNUM\", \"ACTUALELAPSEDTIME\", \"CRSELAPSEDTIME\", \"AIRTIME\", \"ARRDELAY\", \"
DEPDELAY\", \"ORIGIN\", \"DEST\", \"DISTANCE\", \"TAXIIN\", \"TAXIOUT\", \"CANCELLED\",
\"CANCELLATIONCODE\", \"DIVERTED\"
[2]
\"1\", 1994, 8, \"M08\", 2, \"D02\", 2, 840, 840, 1126, 1130, \"NW\", 878, NA, 106, 110, NA, -4, 0,
\"MEM\", \"TPA\", 656, NA, NA, 0, NA, \"0\"
```

hdfs.id

Converts an HDFS path name to an R `dfs.id` object.

Usage

```
hdfs.id(  
  dfs.x,  
  force)
```

Arguments

dfs.x

A string or text expression that resolves to an HDFS file name.

force

Set to `TRUE` if the file need not exist, or set to `FALSE` to ensure that the file does exist.

Return Value

`TRUE` if the string matches an HDFS file name, or `NULL` if a file by that name is not found

Example

This example creates a `dfs.id` object for `/user/oracle/demo`:

```
R> hdfs.id('/user/oracle/demo')  
[1] "user/oracle/demo"  
attr(,"dfs.id")  
[1] TRUE
```

The next example creates a `dfs.id` object named `id` for a nonexistent directory named `/user/oracle/newdemo`, after first failing:

```
R> id<-hdfs.id('/user/oracle/newdemo')  
DBG: 16:11:38 [ER] "/user/oracle/newdemo" is not found  
R> id<-hdfs.id('/user/oracle/newdemo', force=T)  
R> id  
[1] "user/oracle/newdemo"  
attr(,"dfs.id")  
[1] TRUE
```

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.

Return Value

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

See Also

[hdfs.cd](#) on page 9-13

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"  "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).

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.mv

Moves an HDFS file from one location to another.

Usage

```
hdfs.mv(  
    dfs.src,  
    dfs.dst,  
    force)
```

Arguments

dfs.src

The name of the source file to be moved. The file name can include a path that is either absolute or relative to the current path.

dfs.dst

The name of the moved file. The file name can include a path that is either absolute or relative to the current path.

force

Set to `TRUE` to overwrite an existing destination file, or `FALSE` to cancel the operation and display an error message (default).

Return Value

`NULL` for a successful copy, or `FALSE` for a failed attempt

Example

This example moves a file named `weblog` to the `demo` subdirectory and overwrites the existing `weblog` file:

```
R> hdfs.mv("weblog", "./demo", force=T)
```

hdfs.parts

Returns the number of parts composing a file in HDFS.

Usage

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

Arguments

dfs.id

The name of a file in HDFS. The file name can include a path that is either absolute or relative to the current path.

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

The 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 in HDFS has one part:

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

hdfs.pull

Copies data from HDFS into an Oracle database.

This operation requires authentication by Oracle Database. See [orch.connect](#) on page 9-41.

Usage

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

Arguments

dfs.id

The name of a file in HDFS. The file name can include a path that is either absolute or relative to the current path.

sep

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

db.name

The name of a table in an Oracle database.

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: `Sqoop` (default) or `olh` to use Oracle Loader for Hadoop. You must set up Oracle Loader for Hadoop before using it as a driver. See the Usage Notes and "[Oracle Loader for Hadoop Setup](#)" on page 1-12..

Usage Notes

With the Oracle Advanced Analytics option, you can use Oracle R Enterprise to analyze the data after loading it into an Oracle database.

Choosing a Driver

Sqoop is synchronous, and copying a large data set may take a while. The prompt reappears and you regain use of R when copying is complete.

Oracle Loader for Hadoop is much faster than Sqoop, and so you should use it as the driver if possible.

Correcting Problems With the OLH Driver

If Oracle Loader for Hadoop is available, then you see this message when the ORCH library is loading:

```
OLH 2.0.0 is up
```

If you do not see this message, then Oracle Loader for Hadoop is not installed properly. Check that these environment variables are set correctly:

- `OLH_HOME`: Set to the installation directory
- `HADOOP_CLASSPATH`: Includes `$OLH_HOME/jlib/*`
- `CLASSPATH`: Includes `$OLH_HOME/jlib/*`

If `hdfs.pull` fails and `HADOOP_CLASSPATH` is set correctly, then the version of Oracle Loader for Hadoop may be incorrect for the version of CDH. Check the Oracle Loader for Hadoop log file.

See Also: ["Oracle Loader for Hadoop Setup"](#) on page 1-12 for installation instructions

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 an Oracle database to HDFS.

This operation requires authentication by Oracle Database. See [orch.connect](#) on page 9-41.

Usage

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

Arguments

x

An `ore.frame` object with the data in an 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

Identifies the driver used to copy the data. This argument is currently ignored because Sqoop is the only supported driver.

split.by

The column to use for data partitioning (required).

Usage Notes

Because this operation is synchronous, copying a large data set may take a while. The prompt reappears and 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.

If you omit the `split.by` argument, then `hdfs.push` might import only a portion of the data into HDFS.

Return Value

The full path to the file that contains the data set, or `NULL` if the operation failed

See Also

Oracle R Enterprise User's Guide

Example

This example creates an `ore.frame` object named `ontime_s2000` that contains the rows from the `ONTIME_S` database table in 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',
'split.by='YEAR')
R> ontime2000.dfs
[1] "/user/oracle/xq/ontime2000_DB"
attr(,"dfs.id")
[1] TRUE
```

hdfs.put

Copies data from an R in-memory object (`data.frame`) to HDFS. All data attributes, like column names and data types, are stored as metadata with the data.

Usage

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

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.

rownames

Set to `TRUE` to add a sequential number to the beginning of each line of the file, or `FALSE` otherwise.

Usage Notes

You can 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 local R memory and then transfers it to HDFS. For a small table, this function can be faster than `hdfs.push` 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 your R environment is 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,  
  force)
```

Arguments

dfs.id

The name of a file or directory in HDFS. The name can include a path that is either absolute or relative to the current path.

force

Controls whether a directory that contains files is deleted. Set to `TRUE` to delete the directory and all its files, or `FALSE` to cancel the operation (default).

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

This example removes the file named `data1.log` in the current working HDFS directory:

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

hdfs.rmdir

Deletes a directory in HDFS.

Usage

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

Arguments

dfs.name

Name of the directory in HDFS to delete. The directory can be an absolute path or relative to the current working directory.

force

Controls whether a directory that contains files is deleted. Set to `TRUE` to delete the directory and all its files, or `FALSE` to cancel the operation (default).

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.root

Returns the HDFS root directory.

Usage

```
hdfs.root()
```

Return Value

A data frame with the full path of the HDFS root directory

Usage Notes

The default HDFS root is set to /. Users do not have write privileges to this directory. To change the HDFS root to a path where you have write access, use the [hdfs.setroot](#) function.

See Also

[hdfs.setroot](#) on page 9-35

Example

This example identifies /user/oracle as the root directory of HDFS.

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

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

The name of a file in HDFS. The file name can include a path that is either absolute or relative to the current path.

lines

The 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 `va11` and `va12`, are assigned.

This function can become slow when processing large input HDFS files, as the result of inherited limitations in the Hadoop command-line interface.

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.setroot

Sets the HDFS root directory.

Usage

```
hdfs.setroot(dfs.root)
```

Arguments

dfs.root

The full path in HDFS to be set as the current root directory. If this argument is omitted, then your HDFS home directory is used as the root.

Usage Notes

Use [hdfs.root](#) on page 9-33 to see the current root directory.

All HDFS paths and operations using Oracle R Advanced Analytics for Hadoop are relative to the current HDFS root. You cannot change the current working directory above the root directory. Users do not have write access to the HDFS / directory, which is the default HDFS root directory.

Return Value

The full path to the newly set HDFS root directory, or NULL if an error prevented a new root directory from being set

Example

This example changes the HDFS root directory from /user/oracle to /user/oracle/demo.

```
R> hdfs.root()
[1] "/user/oracle"
R> hdfs.setroot("/user/oracle/demo")
R> hdfs.root()
[1] "/user/oracle/demo"
```

hdfs.size

Returns the size of a file in HDFS.

Usage

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

Arguments

dfs.id

The name of a file in HDFS. The file name can include a path that is either absolute or relative to the current path.

units

Specifies a unit of measurement for the return value:

- KB (kilobytes)
- MB (megabytes)
- GB (gigabytes)
- TB (terabytes)
- PB (petabytes)

The unit defaults to bytes if you omit the argument or enter an unknown value.

Usage Notes

Use this interface to determine, for instance, whether you can copy the contents of an 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 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.tail

Copies a specified number of lines from the end of a file in HDFS.

Usage

```
hdfs.tail(  
  dfs.id,  
  n)
```

Arguments

dfs.id

The name of a file in HDFS. The file name can include a path that is either absolute or relative to the current path.

n

The number of lines to retrieve from the file.

Return Value

The last *n* lines of the file.

See Also

[hdfs.head](#) on page 9-19

Example

This example returns the last three lines of `ontime.R`:

```
R> hdfs.tail('ontime_R', 3)  
[1]  
"\"219930\",1994,8,\"M08\",1,\"D01\",1,1947,1945,2038,2048,\"US\",1073,NA,51,63,NA  
, -10,2,\"PIT\", \"LAN\", 275,NA,NA,0,NA, \"0\""  
[2]  
"\"219931\",1994,8,\"M08\",2,\"D02\",2,802,805,948,950,\"AA\",1726,NA,106,105,NA,-  
2,-3,\"DFW\", \"BNA\", 631,NA,NA,0,NA, \"0\""  
[3]  
"\"219932\",1994,8,\"M08\",2,\"D02\",2,829,832,1015,1011,\"US\",1191,NA,166,159,NA  
,4,-3,\"BWI\", \"MSY\", 998,NA,NA,0,NA, \"0\""
```

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 `dfs.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-0000x`. 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 9-16, [hdfs.get](#) on page 9-18, [hdfs.put](#) on page 9-29

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"
```

is.hdfs.id

Indicates whether an R object contains a valid HDFS file identifier.

Usage

```
is.hdfs.id(x)
```

Arguments

x
The name of an R object.

Return Value

TRUE if x is a valid HDFS identifier, or FALSE if it is not

See Also

[hdfs.attach](#) on page 9-11, [hdfs.id](#) on page 9-20

Example

This example shows that `dfs` contains a valid HDFS identifier, which was returned by `hdfs.attach`:

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

The next example shows that a valid file name passed as a string is not recognized as a valid file identifier:

```
R> is.hdfs.id(' /user/oracle/xq/ontime_R')
[1] FALSE
```

orch.connect

Establishes a connection to Oracle Database.

Usage

```
orch.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.

sid

System ID (SID) for the Oracle Database instance.

passwd

Password for the database user. If you omit the password, you are prompted for it.

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 must enter a database password only once during a session. The encrypted password is kept in memory and used for subsequent connection attempts.

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 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.

You can reconnect to Oracle Database using the connection object returned by the `orch.dbcon` function.

Return Value

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

See Also

[orch.dbcon](#) on page 9-45, [orch.disconnect](#) on page 9-52

Example

This example installs the ORCH library and connects to Oracle Database on the local system:

```
R> library(ORCH)
Loading required package: OREbase

Attaching package: OREbase

The following object(s) are masked from package:base:

  cbind, data.frame, eval, interaction, order, paste, pmax, pmin,
  rbind, table

Loading required package: OREstats
Loading required package: MASS
Loading required package: ORCHcore
Oracle R Connector for Hadoop 2.0
Hadoop 2.0.0-cdh4.1.2 is up
Sqoop 1.4.1-cdh4.1.2 is up
OLH 2.0.0 is up

R> orch.connect("localhost", "RUSER", "orcl")
Connecting ORCH to RDBMS via [sqoop]
  Host: localhost
  Port: 1521
  SID: orcl
  User: RUSER
Enter password for [RUSER]: password
Connected.
[1] TRUE
```

Note: If you see the message "p with Hadoop MapReduce" when you load ORCH, then clear the `HADOOP_HOME` environment variable. See ["Installing the Software on Hadoop"](#) on page 1-17.

The next example uses a connection object to reconnect to Oracle Database:

```
R> conn<-orch.dbcon()
R> orch.disconnect()
Disconnected from a database.
```

```
R> orch.connect(conn)
Connecting ORCH to RDBMS via [sqoop]
  Host: localhost
  Port: 1521
  SID: orcl
  User: RQUSER
Enter password for [RQUSER]: password
Connected
[1] TRUE
```

orch.connected

Checks whether Oracle R Advanced Analytics for Hadoop is connected to Oracle Database.

Usage

```
orch.connected()
```

Return Value

TRUE if Oracle R Advanced Analytics for Hadoop has a connection to Oracle Database, or FALSE if it does not.

Example

This example shows that Oracle R Advanced Analytics for Hadoop does not have a connection to Oracle Database:

```
R> orch.connected()  
[1] TRUE
```

orch.dbcon

Returns a connection object for the current connection to Oracle Database, excluding the authentication credentials.

Usage

```
orch.dbcon()
```

Return Value

A data frame with the connection settings for Oracle Database

Usage Notes

Use the connection object returned by `orch.dbcon` to reconnect to Oracle Database using `orch.connect`.

See Also

[orch.connect](#) on page 9-41

Example

This example shows how you can reconnect to Oracle Database using the connection object returned by `orch.dbcon`:

```
R> orch.connect('localhost', 'RUSER', 'orcl')
Connecting ORCH to RDBMS via [sqoop]
  Host: localhost
  Port: 1521
  SID: orcl
  User: RUSER
Enter password for [RUSER]: password
Connected
[1] TRUE
```

```
R> conn<-orch.dbcon()
R> orch.disconnect()
Disconnected from a database.
```

```
R> orch.connect(conn)
Connecting ORCH to RDBMS via [sqoop]
  Host: localhost
  Port: 1521
  SID: orcl
  User: RUSER
Enter password for [RUSER]: password
Connected
[1] TRUE
```

The following shows the connection object returned by `orch.dbcon` in the previous example:

```
R> conn
Object of class "orch.dbcon"
data frame with 0 columns and 0 rows
Slot "ok":
```

```
[1] TRUE

Slot "host":
[1] "localhost"

Slot "port":
[1] 1521

Slot "sid"
[1] "orcl"

Slot "user":
[1] "RQUSER"

Slot "passwd":
[1] ""

Slot "secure":
[1] TRUE

Slot "drv":
[1] "sqoop"
```

orch.dbg.lasterr

Returns the last error message.

Usage

```
orch.dbg.lasterr()
```

Return Value

The last error message.

Example

This example returns the last error message, which indicates a problem with the Hadoop installation.

```
R> orch.dbg.lasterr()  
[1] "split column must be specified"
```

orch.dbg.off

Turns off debugging mode.

Usage

```
orch.dbg.off()
```

Return Value

FALSE

See Also

[orch.dbg.on](#) on page 9-49

Example

This example turns off debugging:

```
R> orch.dbg.off()
```

orch.dbg.on

Turns on debugging mode.

Usage

```
orch.dbg.on(severity)
```

Arguments

severity

Identifies the type of error messages that are displayed. You can identify the severity by the number or by the case-insensitive keyword shown in [Table 9-1](#).

Table 9-1 *Debugging Severity Levels*

Keyword	Number	Description
all	11	Return all messages.
critical	1	Return only critical errors.
error	2	Return all errors.
warning	3	Return all warnings.

Return Value

The severity level

See Also

[orch.dbg.output](#) on page 9-50, [orch.dbg.off](#) on page 9-48

Example

This example turns on debugging for all errors:

```
R> severe<-orch.dbg.on(severity<-2)
R> severe
[1] "ERROR" "2"
```

orch.dbg.output

Directs the output from the debugger.

Usage

```
orch.dbg.output (con)
```

Arguments

con

Identifies the stream where the debugging information is sent: `stderr()`, `stdout()`, or a file name.

Usage Notes

You must first turn on debugging mode before redirecting the output.

Return Value

The current stream

See Also

[orch.dbg.on](#) on page 9-49

Example

This example turns on debugging mode and sends the debugging information to `stderr`. The `orch.dbg.output` function returns a description of `stderr`.

```
R> orch.dbg.on('all')
R> err<-orch.dbg.output(stderr())
17:32:11 [SY] debug output set to "stderr"
R> print(err)
description      class      mode      text      opened      can read      can write
  "stderr"        "terminal"  "w"      "text"    "opened"    "no"          "yes"
```

The next example redirects the output to a file named `debug.log`:

```
R> err<-orch.dbg.output('debug.log')
17:37:45 [SY] debug output set to "debug.log"
R> print(err)
[1] "debug.log"
```

orch.dbinfo

Displays information about the current connection.

Usage

```
orch.dbinfo (dbcon)
```

Arguments

dbcon

An Oracle Database connection object.

Return Value

NULL

See Also

[orch.dbcon](#) on page 9-45

Example

This example displays the information stored in a connection object named `conn`.

```
R> orch.dbinfo(conn)  
Connected to a database via [sqoop]  
Host: localhost  
Port: 1521  
SID: orcl  
User: RQUSER
```

orch.disconnect

Disconnects the local R session from Oracle Database.

Usage

```
orch.disconnect(  
  silent,  
  dbcon)
```

Arguments

silent

Set to `TRUE` to suppress all messages, or `FALSE` to see them (default).

dbcon

Set to `TRUE` to display the connection details, or `FALSE` to suppress this information (default).

Usage Notes

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

Return Value

An Oracle Database connection object when `dbcon` is `TRUE`, otherwise `NULL`

See Also

[orch.connect](#) on page 9-41

Example

This example disconnects the local R session from Oracle Database:

```
R> orch.disconnect()  
Disconnected from a database.
```

The next example disconnects the local R session from Oracle Database and displays the returned connection object:

```
R> oid<-orch.disconnect(silent=TRUE, dbcon=TRUE)  
R> oid  
Object of class "orch.dbcon"  
data frame with 0 columns and 0 rows  
slow "ok":  
[1] TRUE  
  
Slot "host":  
[1] "localhost"  
  
Slot "port":  
[1] 1521  
  
Slot "sid":  
[1] orcl  
  
Slot "user":
```

```
[1] RUSER
```

```
Slot "passwd":
```

```
[1] ""
```

```
Slot "secure":
```

```
[1] TRUE
```

```
Slot "drv":
```

```
[1] "sqoop"
```

orch.dryrun

Switches the execution platform between the local host and the Hadoop cluster. No changes in the R code are required for a dry run.

Usage

```
orch.dryrun (onoff)
```

Arguments

onoff

Set to `TRUE` to run a MapReduce program locally, or `FALSE` to run the program on the Hadoop cluster.

Usage Notes

The `orch.dryrun` function enables you to run a MapReduce program locally on a laptop using a small data set before running it on a Hadoop cluster using a very large data set. The mappers and reducers are run sequentially on row streams from HDFS. The Hadoop cluster is not required for a dry run.

Return Value

The current setting of `orch.dryrun`

See Also

[hadoop.exec](#) on page 9-5, [hadoop.run](#) on page 9-8

Example

This example changes the value of `orch.dryrun` from `FALSE` to `TRUE`.

```
R> orch.dryrun()
[1] FALSE
R> orch.dryrun(onoff<-T)
R> orch.dryrun()
[1] TRUE
```

orch.export

Makes R objects from a user's local R session available in the Hadoop execution environment, so that they can be referenced in MapReduce jobs.

Usage

```
orch.export(...)
```

Arguments

...

One or more variables, data frames, or other in-memory objects, by name or as an explicit definition, in a comma-separated list.

Usage Notes

You can use this function to prepare local variables for use in [hadoop.exec](#) and [hadoop.run](#) functions. The `mapper`, `reducer`, `combiner`, `init`, and `final` arguments can reference the exported variables.

Return Value

A list object

See Also

[hadoop.exec](#) on page 9-5, [hadoop.run](#) on page 9-8

Example

This code fragment shows `orch.export` used in the `export` argument of the `hadoop.run` function:

```
hadoop.run(x,  
  export = orch.export(a=1, b=2),  
  mapper = function(k,v) {  
    x <- a + b  
    orch.keyval(key=NULL, val=x)  
  }  
)
```

The following is a similar code fragment, except that the variables are defined outside the `hadoop.run` function:

```
a=1  
b=2  
hadoop.run(x,  
  export = orch.export(a, b),  
  .  
  .  
  .  
)
```

orch.keyval

Outputs key-value pairs in a MapReduce job.

Usage

```
orch.keyval(  
    key,  
    value)
```

Arguments

key

A scalar value.

value

A data structure such as a scalar, list, data frame, or vector.

Usage Notes

This function can only be used in the mapper, reducer, or combiner arguments of [hadoop.exec](#) and [hadoop.run](#). Because the `orch.keyval` function is not exposed in the ORCH client API, you cannot call it anywhere else.

Return Value

(key, value) structures

See Also

[orch.pack](#) on page 9-59

Example

This code fragment creates a mapper function using `orch.keyval`:

```
hadoop.run(data,  
    mapper = function(k,v) {  
        orch.keyval(k,v)  
    })
```

orch.keyvals

Outputs a set of key-value pairs in a MapReduce job.

Usage

```
orch.keyvals(
  key,
  value)
```

Arguments

key

A scalar value.

value

A data structure such as a scalar, list, data frame, or vector.

Usage Notes

This function can only be used in the mapper, reducer, or combiner arguments of [hadoop.exec](#) and [hadoop.run](#). Because the `orch.keyvals` function is not exposed in the ORCH client API, you cannot call it anywhere else.

See Also

[orch.keyval](#) on page 9-56, [orch.pack](#) on page 9-59

Return Value

(key, value) structures

Example

This code fragment creates a mapper function using `orch.keyval` and a reducer function using `orch.keyvals`:

```
hadoop.run(data,
  mapper(k,v) {
    if (v$value > 10) {
      orch.keyval(k, v)
    }
    else {
      NULL
    }
  },
  reducer(k,vals) {
    orch.keyvals(k,vals)
  }
)
```

The following code fragment shows `orch.keyval` in a for loop to perform the same reduce operation as `orch.keyvals` in the previous example:

```
reducer(k,vals) {
  out <- list()
  for (v in vals) {
```

```
        out <- list(out, orch.keyval(k,vals))
    }
    out
}
```

orch.pack

Compresses one or more in-memory R objects that the mappers or reducers must write as the values in key-value pairs.

Usage

```
orch.pack(...)
```

Arguments

...

One or more variables, data frames, or other in-memory objects in a comma-separated list.

Usage Notes

You should use this function when passing nonscalar or complex R objects, such as data frames and R classes, between the mapper and reducer functions. You do not need to use it on scalar or other simple objects. You can use `orch.pack` to vary the data formats, data sets, and variable names for each output value.

You should also use `orch.pack` when storing the resultant data set in HDFS. The compressed data set is not corrupted by being stored in an HDFS file.

The `orch.pack` function must always be followed by the `orch.unpack` function to restore the data to a usable format.

Return Value

Compressed character-type data as a long string with no special characters

See Also

[hadoop.exec](#) on page 9-5, [hadoop.run](#) on page 9-8, [orch.keyval](#) on page 9-56, [orch.unpack](#) on page 9-62

Example

This code fragment compresses the content of several R objects into a serialized stream using `orch.pack`, and then creates key-value pairs using `orch.keyval`:

```
orch.keyval(NULL, orch.pack(  
  r = r,  
  qY = qY,  
  YY = YY,  
  nRows = nRows))
```

orch.reconnect

Reconnects to Oracle Database with the credentials previously returned by [orch.disconnect](#).

Note: The `orch.reconnect` function is deprecated in release 1.1 and will be desupported in release 2.0. Use [orch.connect](#) to reconnect using a connection object returned by [orch.dbcon](#).

Usage

```
orch.reconnect (dbcon)
```

Arguments

dbcon

Credentials previously returned by [orch.disconnect](#).

Usage Notes

Oracle R Advanced Analytics for Hadoop preserves all user credentials and connection attributes, enabling you to reconnect to a previously disconnected session. Depending on the `orch.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

[orch.connect](#) on page 9-41

Example

```
R> orch.reconnect (oid)
Connecting ORCH to RDBMS via [sqoop]
  Host: localhost
  Port: 1521
  SID:  orcl
  User: RQUSER
Enter password for [RQUSER]: password
Connected
[1] TRUE
```

orch.temp.path

Sets the path where temporary data is stored.

Usage

```
orch.temp.path(path)
```

Arguments

path

The full path to an existing HDFS directory.

Return Value

The current temporary path

Example

This example returns /tmp as the current temporary directory.

```
R> orch.temp.path()  
[1] "/tmp"
```

orch.unpack

Restores the R objects that were compressed with a previous call to `orch.pack`.

Usage

```
orch.unpack(...)
```

Arguments

...

The name of a compressed object.

Usage Notes

This function is typically used at the beginning of a mapper or reducer function to obtain and prepare the input. However, it can also be used externally, such as at the R console, to unpack the results of a MapReduce job.

Consider this data flow:

ORCH client to mapper to combiner to reducer to ORCH client.

If the data is packed at one stage, then it must be unpacked at the next stage.

Return Value

Uncompressed list-type data, which can contain any number and any type of variables

See Also

[orch.pack](#) on page 9-59

Example

This code fragment restores the data that was compressed in the [orch.pack](#) example:

```
reducer = function(key, vals) {  
  x <-orch.unpack(vals[[1]])  
  r <-x$qy  
  yy <- x$yy  
  nRow <- x$nRows  
  .  
  .  
  .  
}
```

orch.version

Identifies the version of the ORCH package.

Usage

```
orch.version()
```

Return Value

ORCH package version number

Example

This example shows that the ORCH version number is 2.0.

```
R> orch.version()  
[1] "2.0"
```


A

- access privileges, Oracle Database, 1-10
- adapters
 - Avro, 6-2
 - Oracle NoSQL Database, 6-29
- additional_path.txt file for ODI, 4-4
- aggregate functions for Hive, 8-3
- ALTER SESSION commands, 2-31
- annotations
 - Avro collection, 6-5
 - equal to Oracle Loader for Hadoop configuration properties, 6-26
 - Oracle Database adapter, 6-20
 - Oracle NoSQL Database adapter, 6-38
- Apache Hadoop distribution, 1-3, 1-4, 1-5, 1-12, 1-17
- Apache licenses, 3-40, 3-43, 3-44
- APIs in ORCH, 8-2
- APPEND hint, 2-31
- as.ore.* functions, 8-3
- Avro
 - functions for reading, 6-5
 - functions for writing, 6-7
- Avro array,
 - reading as XML, 6-12
- Avro file adapter, 6-2
 - reading Avro as XML, 6-10
 - writing XML as Avro, 6-14
- Avro files
 - converting text to, 6-8
 - functions for reading, 6-3
 - querying records, 6-9
 - reading, 6-5
 - reading as XML, 6-10
 - writing, 6-7
- Avro license, 3-43
- Avro maps, 6-3
- Avro maps, reading as XML, 6-11
- Avro null values, 6-14
- Avro primitives
 - reading as XML, 6-14
- Avro records, reading as XML, 6-10
- Avro unions, reading as XML, 6-13
- avro:collection-avroxml function, 6-3
- avro:get function, 6-3
- avroxml method, 6-10, 6-14

B

- bagged.clust.R demo, 8-18
- balancing loads in Oracle Loader for Hadoop, 3-23
- bashrc configuration file, 1-18
- batchSize property, 6-45
- bitops R package, 9-59, 9-62
- bzip2 input files, 2-25

C

- cd command
 - executing in HDFS from R, 9-13
- CDH3 distribution, 1-12
- character encoding, 6-39, 6-42
- character methods for Hive, 8-4
- CKM Hive, 4-2, 4-10
- client libraries, 1-12
- clients
 - configuring Hadoop, 1-5, 1-22
- collection annotations
 - Avro, 6-5
- collection function
 - Oracle NoSQL Database adapter, 6-38
- combiners
 - generating key-value pairs for in R, 9-56, 9-57
 - running from R, 9-5, 9-8
- compressed data files, 2-25
- compressed files, 2-26
- compression in R, 9-59
- compression methods, 6-8
- configuration properties
 - for Oracle XQuery for Hadoop, 6-26
 - Oracle NoSQL Database adapter, 6-44
- configuration settings
 - Hadoop client, 1-5, 1-22
 - Oracle Data Integrator agent, 4-5
 - Oracle Data Integrator interfaces, 4-8
 - Oracle Data Integrator Studio, 4-6
 - R MapReduce jobs, 8-10
 - Sqoop utility, 1-18
- configuring a Hadoop client, 1-5, 1-22
- connecting to Oracle Database from R, 9-3
- connection objects
 - reconnecting to Oracle Database with, 9-42
- consistency property, 6-46

- conversion functions in ORCH, 8-3
- copying data
 - from an Oracle database to HDFS using R, 9-27, 9-29
 - from HDFS to an Oracle database using R, 9-25
 - from HDFS to an R data frame, 9-18
 - from local files to HDFS using R, 9-38
- copying files
 - from HDFS to Linux using R, 9-16
- cp command
 - executing in HDFS from R, 9-14
- CREATE SESSION privilege, 1-10
- CREATE TABLE privilege, 1-11
- CREATE_TARG_TABLE option, 4-9, 4-10, 4-11
- CSV files, 2-26, 3-26, 9-11

D

- data compression in R, 9-59
- data decompression in R, 9-62
- data files
 - accessing from R, 9-11
 - copying database objects into using R, 9-27
 - copying into Oracle Database using R, 9-25
 - deleting from HDFS using R, 9-31
 - obtaining size from R, 9-36
 - sampling from R, 9-34
 - uploading local, 9-38
 - See also* files; HDFS files
- data frames
 - copying HDFS data into, 9-18
 - example of mapper output format, 9-6
 - example program using, 8-23, 8-25, 8-27, 8-28, 8-29
 - from Oracle R Enterprise, 9-8
 - input to mappers, 9-8
- data integrity checking, 4-10
- Data Pump files, 2-8
 - XML template, 2-9
- data sources
 - defining for Oracle Data Integrator, 4-3
- data transformations, 4-10
- data type mappings
 - between XQuery and Avro, 6-14
 - between XQuery and Oracle Database, 6-21
- data types
 - Oracle Loader for Hadoop, 3-5
- data validation in Oracle Data Integrator, 4-2, 4-10
- database
 - copying data from using R, 9-29
- database connections
 - disconnecting using R, 9-52
 - from R, 9-41, 9-44, 9-45
 - querying from R, 9-51
 - reconnecting from R, 9-60
- database directories
 - for Oracle SQL Connector for HDFS, 1-7
- database objects
 - copying to HDFS using R, 9-27
- database patches, 1-4, 1-12, 2-8

- database privileges, 1-10
- database system, configuring to run MapReduce jobs, 1-5
- DataServer objects
 - creating for Oracle Data Integrator, 4-3
- debugging in R
 - displaying last error, 9-47
 - identifying version, 9-63
 - redirecting output, 9-50
 - turning off, 9-48
 - turning on, 9-49
- decompression in R, 9-62
- DEFER_TARGET_LOAD option, 4-9
- DELETE_ALL option, 4-11
- DELETE_TEMPORARY_OBJECTS option, 4-9, 4-10, 4-11
- delimited text files
 - XML templates, 2-14
- DelimitedTextInputFormat class, 3-11, 3-30, 3-34
 - Oracle Loader for Hadoop, 3-11
- DelimitedTextOutputFormat class, 3-26
- demo code for ORCH, 8-11
- describe command
 - executing in HDFS using R, 9-15
- dfs.id object
 - obtaining in R, 9-20
- directories
 - accessible by Oracle Data Integrator, 4-3
 - creating in HDFS using R, 9-22
 - current HDFS, 9-30
 - for Oracle Loader for Hadoop output, 4-11
 - listing in HDFS, 9-21
 - ODI Application Adapter for Hadoop home, 1-15
 - Oracle SQL Connector for HDFS home, 1-7
 - R Advanced Analytics examples, 8-19
 - removing using R, 9-32
 - Sqoop home, 1-18
 - See also* database directories; root directory
- disable_directory_link_check access parameter, 2-8
- downloading software, 1-3, 1-4, 1-14, 1-17, 1-18, 1-19, 1-23
- drivers
 - JDBC, 1-18, 3-18, 4-4
 - Oracle Data Integrator agent, 4-5
 - ORACLE_DATAPUMP, 3-21
 - ORACLE_LOADER, 2-18
- DROP_ERROR_TABLE option, 4-10
- dry run execution in R, 9-54
- durability property, 6-44

E

- encoding characters, 6-39, 6-42
- error Something is terribly wrong..., 1-18
- example code for ORCH
 - descriptions, 8-19
- exists function in R, 9-17
- exporting R objects to Hadoop, 9-55
- EXT_TAB_DIR_LOCATION option, 4-11
- external tables

- about, 2-1
- EXTERNAL_VARIABLE_DATA access
 - parameter, 2-8
- EXTERNAL_TABLE option, 4-9
- ExternalTable command
 - syntax, 2-6
- EXTRA_OLH_CONF_PROPERTIES option, 4-11

F

- file formats for Oracle Data Integrator, 4-8
- file identifiers
 - validating in R, 9-40
- file sampling in R, 9-19, 9-37
- file size
 - obtaining using R, 9-36
- file sources
 - defining for Oracle Data Integrator, 4-3
- File to Hive KM, 4-2, 4-8
- FILE_IS_LOCAL option, 4-9
- File-Hive to Oracle (OLH) KM, 4-2, 4-10
- files
 - downloading from HDFS, 9-16
 - obtaining HDFS metadata using R, 9-15
 - testing existence in HDFS from R, 9-17
 - uploading local to HDFS, 9-38
 - See also* data files; HDFS files
- filter1.R example, 8-19
- filter2.R example, 8-19
- filter3.R example, 8-20
- flex fields, 4-4, 4-8
- FLOW_CONTROL option, 4-9
- FLOW_TABLE_OPTIONS option, 4-11
- frame methods for Hive, 8-4
- functions
 - for reading from Oracle NoSQL Database, 6-41
 - for writing to Oracle NoSQL Database, 6-43
 - Oracle NoSQL Database, 6-31
 - reading and writing sequence files, 6-48
 - reading Avro files, 6-5
 - reading from Oracle NoSQL Database, 6-38
 - writing Avro files, 6-7

G

- get function
 - Oracle NoSQL Database adapter, 6-41
- group.apply.R example, 8-20
- gzip input files, 2-25

H

- Hadoop client
 - configuring, 1-5, 1-22
 - installing, 1-5
- hadoop fs command, 1-5
- HADOOP_CLASSPATH
 - for Oracle Data Integrator, 4-6
- HADOOP_HOME
 - for Oracle Data Integrator, 4-5
- HADOOP_HOME environment variable, 1-18

- HADOOP_LIBEXEC_DIR environment
 - variable, 1-18
- hadoop.exec function, 8-10
 - config argument, 8-10
 - example, 8-18, 8-22, 8-25, 9-6
 - syntax description, 9-5
- hadoop.run function, 8-10
 - config argument, 8-10
 - example, 8-15, 8-19, 8-20, 8-22, 8-23, 8-24, 8-26, 8-27, 8-28, 8-29, 9-9
 - syntax description, 9-8
- HDFS client
 - See* Hadoop client
- HDFS commands
 - issuing from R, 9-3
- HDFS data
 - copying in R, 9-3
- HDFS directories
 - acquiring root using R, 9-33
 - changing in R, 9-13
 - changing root using R, 9-35
 - creating a dfs.id object, 9-20
 - creating in R, 9-22
 - deleting using R, 9-31, 9-32
 - identifying current in R, 9-30
 - listing in R, 9-21
- HDFS file identifiers
 - obtaining in R, 9-20
 - validating in R, 9-40
- HDFS file names
 - converting to dfs.id object in R, 9-20
- HDFS files
 - accessing from R, 9-11
 - copying data from Oracle Database, 9-27
 - copying data from the database using R, 9-29
 - copying data to Oracle Database, 9-25
 - copying data to R, 9-18
 - copying from database objects using R, 9-27
 - copying into Hive, 4-8
 - copying into Oracle Database using R, 9-25
 - copying using R, 9-14
 - deleting using R, 9-31
 - downloading to local system, 9-16
 - loading data into an Oracle database, 3-14
 - moving from R, 9-23
 - number of parts, 9-24
 - obtaining size from R, 9-36
 - restrictions in Oracle R Advanced Analytics for Hadoop, 8-2
 - sampling from R, 9-34
 - testing existence in HDFS from R, 9-17
 - uploading local, 9-38
- HDFS metadata
 - obtaining using R, 9-15
- HDFS path
 - changing in R, 9-13
- HDFS root directory
 - changing from R, 9-35
 - obtaining using R, 9-33
- hdfs_datatrans R demo, 8-12

- hdfs_dir R demo, 8-13
- hdfs_putget R demo, 8-13
- hdfs_stream Bash shell script, 1-6
- hdfs.attach function
 - example, 8-13, 9-6, 9-9, 9-40
 - syntax description, 9-11
- hdfs.cd function
 - example, 8-12, 8-13, 8-15, 8-16, 8-17
 - syntax description, 9-13
- hdfs.cp function
 - example, 8-12
 - syntax description, 9-14
- hdfs.cpmv R demo, 8-11
- hdfs.describe function
 - example, 8-12, 8-13
 - syntax example, 9-15
- hdfs.download function
 - example, 8-12, 8-15, 8-20, 8-24
 - syntax description, 9-16
- hdfs.exists function
 - example, 8-12, 8-13, 8-15, 8-20, 8-24
 - syntax description, 9-17
- hdfs.get function
 - example, 8-12, 8-13, 8-15, 8-16, 8-18, 8-19, 8-20, 8-22, 8-23, 8-24, 8-25, 8-27, 8-28
 - syntax description, 9-18
- hdfs.head function
 - syntax description, 9-19
- hdfs.id function
 - example, 8-15, 8-20, 8-24
 - syntax description, 9-20
- hdfs.ls function
 - example, 8-12, 8-13
 - syntax description, 9-21
- hdfs.mkdir function
 - example, 8-12, 8-13, 8-16, 8-17, 8-20, 8-24
 - syntax description, 9-22
- hdfs.mv function
 - example, 8-12
 - syntax description, 9-23
- hdfs.parts function
 - syntax description, 9-24
- hdfs.pull function
 - example, 8-12
 - syntax description, 9-25
- hdfs.push function
 - syntax description, 9-27
- hdfs.put function
 - example, 8-12, 8-13, 8-15, 8-16, 8-18, 8-19, 8-20, 8-22, 8-23, 8-24, 8-25, 8-27, 8-28, 8-29
 - syntax description, 9-29
- hdfs.pwd function
 - example, 8-12, 8-13, 8-15, 8-16, 8-17
 - syntax description, 9-30
- hdfs.rm function
 - example, 8-12, 8-13, 8-15, 8-16
 - syntax example, 9-31
- hdfs.rmdir function
 - example, 8-12, 8-13, 8-15, 8-16, 8-17, 8-20, 8-24
 - syntax description, 9-32
- hdfs.root function
 - example, 8-12, 8-13, 8-15, 8-16, 8-17
 - syntax description, 9-33
- hdfs.sample function
 - example, 8-13, 8-16
 - syntax description, 9-34
- hdfs.setroot function
 - example, 8-12, 8-13, 8-15, 8-16, 8-17
 - syntax description, 9-35
- hdfs.size function
 - example, 8-12
 - syntax description, 9-36
- hdfs.tail function
 - syntax description, 9-37
- hdfs.upload function
 - example, 8-12, 8-15, 8-20, 8-24
 - syntax description, 9-38
- hints for optimizing queries, 2-31
- Hive access from R, 8-3
- Hive application adapters, 4-2, 4-10
- Hive Control Append KM, 4-2, 4-9
- Hive data source for Oracle Data Integrator, 4-3
- Hive data types, support for, 8-5
- Hive database for Oracle Loader for Hadoop, 1-12
- Hive distribution, 1-12
- Hive flex fields, 4-8
- Hive Query Language (HiveQL), 4-2
- Hive tables
 - loading data into (Oracle Data Integrator), 4-8
 - reverse engineering, 4-2, 4-7
 - reverse engineering in Oracle Data Integrator, 4-7
 - XML format, 2-12
- Hive Transform KM, 4-2, 4-10
- hive_aggregate R demo, 8-13
- hive_analysis R demo, 8-13
- hive_basic R demo, 8-14
- hive_binning R demo, 8-14
- hive_columnfns R demo, 8-14
- HIVE_HOME, for Oracle Data Integrator, 4-5
- hive_nulls R demo, 8-14
- hive_pushpull R demo, 8-14
- hive_sequencefile R demo, 8-14
- HiveToAvroInputFormat class, 3-12
- hosts property, 6-45

- I
- IKM File to Hive, 4-2, 4-8
- IKM File-Hive to Oracle (OLH), 4-2, 4-10
- IKM Hive Control Append, 4-2, 4-9, 4-10
- IKM Hive Transform, 4-2, 4-9, 4-10
- IndexedRecord, 3-15
- InputFormat class
 - Oracle Loader for Hadoop, 3-11
- INSERT_UPDATE mode, 4-3
- installation
 - Apache Hadoop, 1-5
 - CDH, 1-5
 - Hadoop client, 1-5
 - Oracle Data Integrator Application Adapter for

- Hadoop, 1-14
- Oracle Loader for Hadoop, 1-12
- Oracle R Advanced Analytics for Hadoop, 1-17
- Oracle SQL Connector for HDFS, 1-4
- Sqoop utility, 1-18
- Instant Client libraries, 1-12
- interface configurations
 - Oracle Data Integrator, 4-8
- is.hdfs.id function
 - example, 8-18
 - syntax description, 9-40
- is.ore.* functions, 8-3

J

- JDBC drivers, 1-18, 3-18, 4-4
- Jellyfish R demo, 8-16
- job names
 - specifying in R, 8-10, 9-5, 9-6, 9-8

K

- key-value pairs
 - generating from R objects, 9-56, 9-57
- kmeans.R demo, 8-18
- kmeans.R example, 8-22
- knowledge modules
 - description, 4-2
 - See also* CKM, IKM, and RKM entries
- kv
 - collection-avroxml function, 6-32
 - collection-binxml function, 6-33, 6-34
 - collection-text function, 6-31, 6-32
 - collection-xml function, 6-32, 6-33
 - get-avroxml function, 6-34
 - get-binxml function, 6-35
 - get-text function, 6-34
 - get-xml function, 6-35
 - key-range function, 6-35
 - put-binxml function, 6-34
 - put-text function, 6-34
 - put-xml function, 6-34
- kvstore property, 6-45

L

- licenses, third-party, 3-40
- linear model R demo, 8-15
- LMF Jellyfish R demo, 8-16
- LMF mahout ALS-WR demo, 8-16
- lm.R example, 8-22
- load balancing
 - in Oracle Loader for Hadoop, 3-23
- loadCI, 3-23
- loading data files into Hive, 4-8
- loading options for Oracle Data Integrator, 4-8
- local files
 - copying into Hive, 4-8
 - downloading from HDFS, 9-16
- LOG_FILE_NAME option, 4-7
- logical methods for Hive, 8-4

- logreg.R example, 8-23

M

- mahout ALS-WR demo, 8-16
- map.df.R example, 8-23
- map.list.R example, 8-23
- mappers
 - configuration settings, 8-10
 - examples, 8-19
 - generating key-value pairs for in R, 9-56, 9-57
 - running from R, 9-5, 9-8
- mapred_basic R demo, 8-15
- mapred_modelBuild R demo, 8-15
- MAPRED_OUTPUT_BASE_DIR option, 4-11
- mapred.config class
 - syntax description, 8-10
 - using, 9-5, 9-8
- MapReduce functions
 - writing in R, 9-3
- MapReduce jobs
 - configuring in R, 8-10
 - exporting R objects to, 9-55
 - running from R, 9-5, 9-8
- MapReduce programs
 - data compression in R, 9-59
 - data decompression in R, 9-62
- MasterPolicy durability, 6-44
- matrix methods for Hive, 8-4
- maxLoadFactor property, 3-23
- mkdir command
 - running in R, 9-22
- model.plot.R example, 8-24
- model.prep.R example, 8-25
- mv command
 - running from R, 9-23

N

- neural network R demo, 8-16
- nilled function, 6-14
- NMF R demo, 8-17
- null values in Avro, 6-14
- numeric methods for Hive, 8-4

O

- OCI Direct Path
 - See* Oracle OCI Direct Path output formats
- ODI_ADDITIONAL_CLASSPATH environment variable, 4-5
- ODI_HIVE_SESSION_JARS, for Oracle Data Integrator, 4-5, 4-6
- ODI_OLH_JARS, for Oracle Data Integrator, 4-6
- OLH_HOME environment variable, 1-13, 1-16, 4-6
- OLH_OUTPUT_MODE option, 4-11
- operating system user permissions, 1-6
- Oracle, 3-27
- Oracle Data Integrator agent
 - configuring, 4-5
 - drivers, 4-5

- Oracle Data Integrator Application Adapter
 - description, 4-1
- Oracle Data Integrator Application Adapter for Hadoop
 - creating models, 4-7
 - data sources, 4-3
 - flex fields, 4-8
 - installing, 1-14
 - loading options, 4-8
 - security, 4-2
 - setting up projects, 4-6
 - topology setup, 4-3
- Oracle Data Integrator Companion CD, 1-15
- Oracle Data Integrator Studio configuration, 4-6
- Oracle Database
 - connecting from R, 9-3
 - copying data from HDFS, 9-25
 - copying data to HDFS, 9-27
 - querying connections from R, 9-51
 - reconnecting from R, 9-60
 - user privileges, 1-10
- Oracle Database access from ORCH, 8-8
- Oracle Database adapter, 6-19
 - configuration properties, 6-26
- Oracle Database connections
 - disconnecting from R, 9-52
 - from R, 9-41, 9-44, 9-45
 - reconnecting from R, 9-60

See also Oracle Database
- Oracle Direct Connector for HDFS
 - pattern-matching characters, 2-26
- Oracle Instant Client libraries, 1-12
- Oracle Loader for Hadoop
 - description, 3-1
 - input formats, 3-14
 - installing, 1-12
 - supported database versions, 1-12
 - using in Oracle Data Integrator, 4-2, 4-6, 4-10
- Oracle Loader for Hadoop configuration properties
 - used by Oracle XQuery for Hadoop, 6-26
- Oracle NoSQL Database
 - writing to, 6-43
- Oracle NoSQL Database Adapter
 - configuration properties, 6-44
- Oracle NoSQL Database adapter, 6-29
 - collection function, 6-38
 - functions for reading, 6-38
 - reading Avro as XML, 6-10
 - writing XML as Avro, 6-14
- Oracle NoSQL Database functions, 6-31
- Oracle OCI Direct Path interface, 3-27
- Oracle OCI Direct Path output formats, 3-27
- Oracle permissions, 1-6
- Oracle R Advanced Analytics for Hadoop
 - description, 8-1
 - installation, 1-17
 - obtaining the ORCH version number, 9-63
- Oracle R Connector for Hadoop
 - access to HDFS files, 9-11
 - alphabetical list of functions, 9-1
 - categorical list of functions, 9-2
 - connecting to Oracle Database, 9-3
 - copying HDFS data, 9-3
 - debugging functions, 9-3
 - HDFS commands issued from, 9-3
 - MapReduce functions, 9-3
- Oracle Software Delivery Cloud, 1-3
- Oracle SQL Connector for HDFS
 - description, 2-1
 - installation, 1-4
 - query optimization, 2-31
- Oracle Technology Network
 - certifications, 1-15
 - downloads, 1-3, 1-18
- Oracle XQuery for Hadoop, 5-1
- ORACLE_DATAPUMP driver, 3-21
- ORACLE_LOADER driver, 2-18
- oracle.hadoop.xquery.kv.config.durability
 - property, 6-44
- oracle.hadoop.xquery.kv.config.requestLimit
 - property, 6-44
- oracle.hadoop.xquery.kv.config.requestTimeout
 - property, 6-45
- oracle.hadoop.xquery.kv.config.socketOpenTimeout
 - property, 6-45
- oracle.hadoop.xquery.kv.config.socketReadTimeout
 - property, 6-45
- oracle.kv.batchSize property, 6-45
- oracle.kv.consistency property, 6-46
- oracle.kv.hosts configuration property, 6-44
- oracle.kv.hosts property, 6-45
- oracle.kv.kvstore configuration property, 6-44
- oracle.kv.kvstore property, 6-45
- oracle.kv.timeout property, 6-46
- orahdfs-*version*/bin directory, 1-7
- orahdfs-*version*.zip file, 1-6, 1-9
- OraLoader, 3-21, 3-25, 5-13
- OraLoaderMetadata utility program, 3-8
- oraloader-*version* directory, 1-13, 1-16
- oraloader-*version*.zip file, 1-6, 1-13, 1-16
- ORCH package
 - installation, 1-17, 1-19
 - version numbers, 9-63

See also Oracle R Advanced Analytics for Hadoop
- orch_lm R demo, 8-15
- orch_lmf_jellyfish R demo, 8-16
- orch_lmf_mahout_als R demo, 8-16
- orch_neural R demo, 8-16
- orch.connect function
 - syntax description, 9-41
- orch.connected function
 - example, 8-12
 - syntax description, 9-44
- orch.dbcon function
 - syntax description, 9-45
- orch.dbg.lasterr function
 - syntax description, 9-47
- orch.dbg.off function
 - syntax description, 9-48
- orch.dbg.on function

- syntax description, 9-49
- orch.dbg.output function
 - syntax description, 9-50
- orch.dbinfo function
 - syntax description, 9-51
- orch.disconnect function
 - syntax description, 9-52
- orch.dryrun function
 - syntax description, 9-54
- orch.evaluate function, 8-9
 - example, 8-16
- orch.export function
 - example, 8-15, 8-18, 8-20, 8-22, 8-23, 8-24, 8-25, 8-28, 8-29
 - syntax description, 9-55
- orch.export.fit function, 8-9
 - example, 8-16
- orch.keyval function
 - example, 8-15, 8-18, 8-19, 8-20, 8-27, 8-28
 - syntax description, 9-56
- orch.keyvals function
 - example, 8-25, 8-26, 8-28, 8-29
 - syntax description, 9-57
- orch.lm function, 8-9
 - example, 8-15
- orch.lmf function, 8-9
 - example, 8-16
- orch.neural function, 8-9
 - example, 8-16
- orch.nmf demo, 8-17
- orch.nmf function, 8-9
- orch.nmf.NMFalgo function, 8-9
 - example, 8-17
- orch.pack function
 - example, 8-15, 8-18, 8-20, 8-23, 8-24
 - syntax description, 9-59
- orch.recommend function, 8-9
 - example, 8-16
- orch.reconnect function
 - syntax description, 9-60
- orch.temp.path function, 9-61
- orch.tgz package, 1-19
- orch.unpack function
 - example, 8-15, 8-18, 8-20, 8-23, 8-26
 - syntax description, 9-62
- orch.version function
 - syntax description, 9-63
- orch.which function
 - See* orch.dbcon function
- ORE functions for Hive, 8-3
- ore functions for Hive, 8-3
- ore.attach function
 - example, 8-13, 8-14
- ore.connect function
 - example, 8-13, 8-14
- ore.create function, 8-7
 - example, 8-14
- ore.drop function
 - example, 8-12, 8-15
- ore.exec function, 8-7

- example, 8-14
- ore.frame objects, 9-27, 9-29
 - See* data frames
- ore.is.connected function
 - example, 8-12
- ore.pull function
 - example, 8-14
- ore.push function
 - example, 8-14
- ore.sync function
 - example, 8-14
- ore.warn.order option, 8-7
- OSCH_BIN_PATH directory, 1-11
- OVERRIDE_INPUTFORMAT option, 4-11
- OVERRIDE_ROW_FORMAT option, 4-9

P

- parallel processing, 1-2, 2-31, 8-10, 8-11, 8-20, 9-24
- partitioning, 3-5
- path, changing in R, 9-13
- pattern-matching characters in Oracle Direct Connector for HDFS, 2-26
- POST_TRANSFORM_DISTRIBUTE option, 4-10
- POST_TRANSFORM_SORT option, 4-10
- PQ_DISTRIBUTE hint, 2-31
- PRE_TRANSFORM_DISTRIBUTE option, 4-10
- PRE_TRANSFORM_SORT option, 4-10
- predict.orch.lm function, 8-9
 - example, 8-15
- preprocessor access parameter, 2-8
- print.orch.lm function, 8-9
- print.summary.orch.lm function, 8-9
- privileges, Oracle Database, 1-10
- program execution, local dry run in R, 9-54
- projects
 - setting up in Oracle Data Integrator, 4-6
- put function
 - Oracle NoSQL Database adapter, 6-43
- pwd command
 - running from R, 9-30

Q

- query optimization for Oracle SQL Connector for HDFS, 2-31

R

- R connector
 - See* Oracle R Connector for Hadoop
- R Distribution, 1-19, 1-23
- R distribution, 1-17, 1-22
- R functions
 - alphabetical listing, 9-1
 - categorical listing, 9-2
 - See also* Oracle R Connector for Hadoop
- R functions for Hive, 8-4
- R packages
 - bitops, 9-59
- R programs

- local execution, 9-54
- random order messages, 8-7
- reading Avro files, 6-5
- RECYCLE_ERRORS option, 4-9
- reducers
 - configuration settings, 8-11
 - examples, 8-19
 - generating key-value pairs for in R, 9-56, 9-57
 - running from R, 9-5, 9-8
- ReplicaAck policy, 6-44
- ReplicaPolicy durability, 6-44
- requestLimit property, 6-44
- requestTimeout property, 6-45
- reverse engineering in Hive, 4-2, 4-7
- reverse-engineering Hive tables, 4-7
- reverse.log file, 4-7
- RKM Hive, 4-2, 4-7
- rIm.R example, 8-26
- rm command
 - issuing from R, 9-31
- rmdir command
 - issuing from R, 9-32
- root directory
 - changing from R, 9-35
 - obtaining using R, 9-33

S

- sampling data
 - from Oracle Loader for Hadoop, 3-23
- sampling HDFS files, 9-19, 9-37
- scripts
 - debugging in R, 9-3
 - snippets, 4-2
- seq
 - collection function, 6-48
 - collection-binxml function, 6-49
 - collection-xml function, 6-48
 - put function, 6-50
 - put functions, 6-49
 - put-binxml function, 6-51, 6-52
 - put-xml function, 6-50, 6-51
- sequence file adapter, 6-47
- sequence file adapter functions, 6-48
- SerDes JAR files, 4-5, 4-6
- snippets in Oracle Data Integrator, 4-2
- socketOpenTimeout property, 6-45
- socketReadTimeout property, 6-45
- software downloads, 1-3, 1-4, 1-14, 1-17, 1-18, 1-19, 1-23
- Something is terribly wrong with Hadoop
 - MapReduce message, 1-18
- split sizes, 6-6
- split.map.R example, 8-27
- split.reduce.R example, 8-27
- SQL*Loader, 3-19
- Sqoop, 8-8
- Sqoop utility
 - installing on a Hadoop client, 1-23
 - installing on a Hadoop cluster, 1-18

- STATIC_CONTROL option, 4-9
- STOP_ON_FILE_NOT_FOUND option, 4-9
- subrange specification, Oracle NoSQL Database adapter, 6-40
- summary.orch.Im function, 8-9
- sum.R example, 8-28

T

- tables
 - copying data from HDFS, 3-1
 - copying data from HDFS to Oracle Database, 9-25
 - copying data from Oracle Database to HDFS, 9-27
- TEMP_DIR option, 4-11
- temporary HDFS path for R, 9-61
- teragen2.xy.R example, 8-29
- teragen.matrix.R example, 8-28
- teragen.xy.R example, 8-29
- terasort.R example, 8-29
- testing functions in R, 8-3
- text files
 - converting to Avro, 6-8
- third-party licenses, 3-40
- timeout property, 6-46
- TRANSFORM_SCRIPT option, 4-10
- TRANSFORM_SCRIPT_NAME option, 4-10
- transforming data
 - in Oracle Data Integrator, 4-1, 4-9
- TRUNCATE option, 4-9, 4-11
- type mappings
 - between XQuery and Avro, 6-14
 - between XQuery and Oracle Database, 6-21

U

- uncompressed files, 2-26
- USE_HIVE_STAGING_TABLE option, 4-11
- USE_LOG option, 4-7
- USE_ORACLE_STAGING_TABLE option, 4-11
- USE_STAGING_TABLE option, 4-9
- userlib directory, 4-6
- UTF-8 encoding, 6-39, 6-42
- UTL_FILE package, 1-11

V

- validating data
 - in Oracle Data Integrator, 4-1, 4-2, 4-9
- vector methods for Hive, 8-4

W

- wildcard characters
 - in resource names, 4-8
 - setting up data sources in ODI using, 4-3
 - support in IKM File To Hive (Load Data), 4-2
- writing Avro files, 6-7

X

XML

- writing as Avro arrays, 6-17
- writing as Avro maps, 6-16
- writing as Avro primitives, 6-18
- writing as Avro records, 6-15
- writing as Avro unions, 6-17

XML template for Data Pump files, 2-9

XML templates

- Data Pump files, 2-9
- delimited text files, 2-14
- Hive tables, 2-12

xml-reference directory, 1-15, 4-3

XQuery

- See* Oracle XQuery for Hadoop

