

Oracle® Big Data Connectors

User's Guide



Release 5 (5.0)

F10954-02

June 2019

The Oracle logo, consisting of a solid red square with the word "ORACLE" in white, uppercase, sans-serif font centered within it.

ORACLE®

Oracle Big Data Connectors User's Guide, Release 5 (5.0)

F10954-02

Copyright © 2011, 2019, Oracle and/or its affiliates.

Primary Author: Dimpi Sarmah, Frederick Kush

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, then the following notice is applicable:

U.S. GOVERNMENT END USERS: Oracle programs (including any operating system, integrated software, any programs embedded, installed or activated on delivered hardware, and modifications of such programs) and Oracle computer documentation or other Oracle data delivered to or accessed by U.S. Government end users are "commercial computer software" or "commercial computer software documentation" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, the use, reproduction, duplication, release, display, disclosure, modification, preparation of derivative works, and/or adaptation of i) Oracle programs (including any operating system, integrated software, any programs embedded, installed or activated on delivered hardware, and modifications of such programs), ii) Oracle computer documentation and/or iii) other Oracle data, is subject to the rights and limitations specified in the license contained in the applicable contract. The terms governing the U.S. Government's use of Oracle cloud services are defined by the applicable contract for such services. 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 Inside 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, Epyc, and the AMD 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 about 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 unless otherwise set forth in an applicable agreement between you and Oracle. 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, except as set forth in an applicable agreement between you and Oracle.

Contents

Preface

Audience	xv
Documentation Accessibility	xv
Related Documents	xv
Text Conventions	xvi
Syntax Conventions	xvi
Changes in Oracle Big Data Connectors 5.0	xvi

Part I Setup

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?	1-3
What is Apache Hadoop?	1-3
Download and Install Oracle Big Data Connectors	1-4
Certified Hadoop Platforms	1-5
Secure Connection to Oracle Database	1-5
Using JDBC SSL	1-5
Using Secure External Java KeyStore and Hadoop credential command	1-6
Oracle SQL Connector for Hadoop Distributed File System Setup	1-7
Software Requirements	1-7
Install and Configure a Hadoop Client on the Oracle Database System	1-8
Install Oracle SQL Connector for HDFS	1-10
Oracle Database Privileges for OSCH Users	1-13
OS-Level Requirements for OSCH Users	1-14
Use Oracle SQL Connector for HDFS on a Secure Hadoop Cluster	1-14
Use OSCH in Oracle SQL Developer	1-15
Oracle Loader for Hadoop Setup	1-15
Software Requirements	1-15
Install Oracle Loader for Hadoop	1-15

Oracle Database Privileges for OLH Users	1-16
Provide Support for Offline Database Mode	1-16
Use Oracle Loader for Hadoop on a Secure Hadoop Cluster	1-17
Oracle Shell for Hadoop Loaders Setup	1-17
Install Oracle Shell for Hadoop Loaders on a Hadoop Node	1-19
Oracle Database Privileges for OHS Users	1-21
Configure OHS to Enable Job Monitoring	1-21
Oracle XQuery for Hadoop Setup	1-23
Software Requirements	1-23
Install Oracle XQuery for Hadoop	1-23
Troubleshoot the File Paths	1-25
Configure Oozie for the Oracle XQuery for Hadoop Action	1-25
Oracle R Advanced Analytics for Hadoop Setup	1-26
Install the Software on Hadoop	1-26
Software Requirements for a Third-Party Hadoop Cluster	1-27
Install Sqoop on a Third-Party Hadoop Cluster	1-27
Install Hive on a Third-Party Hadoop Cluster	1-28
Install R on a Hadoop Client	1-28
Install R on a Third-Party Hadoop Cluster	1-29
Install the ORCH Package on a Third-Party Hadoop Cluster	1-29
Install Additional R Packages	1-30
Provide Remote Client Access to R Users	1-32
Software Requirements for Remote Client Access	1-32
Configure the Server as a Hadoop Client	1-32
Install Sqoop on a Hadoop Client	1-33
Install R on a Hadoop Client	1-33
Install the ORCH Package on a Hadoop Client	1-33
Install the Oracle R Enterprise Client Packages (Optional)	1-33
Oracle Data Integrator	1-33
Oracle Datasource for Apache Hadoop Setup	1-34
Configure HiveServer2	1-34

Part II Oracle Database Connectors

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
Configure Your System for Oracle SQL Connector for HDFS	2-6
Use Oracle SQL Connector for HDFS with Oracle Big Data Appliance and Oracle Exadata	2-7

Use the ExternalTable Command-Line Tool	2-7
About ExternalTable	2-7
ExternalTable Command-Line Tool Syntax	2-7
Create External Tables	2-9
Create External Tables with the ExternalTable Tool	2-9
Create External Tables from Data Pump Format Files	2-10
Required Properties	2-10
Optional Properties	2-11
Defining Properties in XML Files for Data Pump Format Files	2-11
Example	2-12
Create External Tables from Hive Tables	2-12
Hive Table Requirements	2-13
Data Type Mappings	2-13
Required Properties	2-13
Optional Properties	2-14
Defining Properties in XML Files for Hive Tables	2-14
Example	2-15
Creating External Tables from Partitioned Hive Tables	2-16
Create External Tables from Delimited Text Files	2-20
Data Type Mappings	2-20
Required Properties	2-20
Optional Properties	2-21
Defining Properties in XML Files for Delimited Text Files	2-21
Example	2-23
Create External Tables in SQL	2-23
Update External Tables	2-24
ExternalTable Syntax for Publish	2-25
ExternalTable Example for Publish	2-25
Explore External Tables and Location Files	2-25
ExternalTable Syntax for Describe	2-26
ExternalTable Example for Describe	2-26
Drop Database Objects Created by Oracle SQL Connector for HDFS	2-26
ExternalTable Syntax for Drop	2-26
ExternalTable Example for Drop	2-27
More About External Tables Generated by the ExternalTable Tool	2-27
About Configurable Column Mappings	2-27
Default Column Mappings	2-27
All Column Overrides	2-28
One Column Overrides	2-28
Mapping Override Examples	2-28
What Are Location Files?	2-29

Enable Parallel Processing	2-29
Set Up Degree of Parallelism	2-29
Location File Management	2-30
Location File Names	2-31
Configure Oracle SQL Connector for HDFS	2-31
Create a Configuration File	2-31
Oracle SQL Connector for HDFS Configuration Property Reference	2-32
Performance Tips for Querying Data in HDFS	2-45

3 Oracle Loader for Apache Hadoop

What Is Oracle Loader for Hadoop?	3-1
Interfaces to Oracle Loader For Hadoop	3-2
Get Started With Oracle Loader for Hadoop	3-2
Additional Information	3-7
Use Oracle Loader for Hadoop With the Hadoop Command Line Utility	3-8
About the Modes of Operation	3-9
Online Database Mode	3-9
Offline Database Mode	3-12
Create the Target Table	3-14
Supported Data Types for Target Tables	3-14
Supported Partitioning Strategies for Target Tables	3-14
Compression	3-15
Create a Job Configuration File	3-15
Establish Secure Connections to Oracle Database Using SSL and Oracle Wallet	3-17
Use Oracle Wallets	3-17
Use JDBC SSL	3-18
Generate the Target Table Metadata for Offline Database Mode	3-18
About Input Formats	3-18
Delimited Text Input Format	3-19
Complex Text Input Formats	3-20
Hive Table Input Format	3-21
Avro Input Format	3-22
Oracle NoSQL Database Input Format	3-22
Custom Input Formats	3-23
Mapping Input Fields to Target Table Columns	3-24
Automatic Mapping	3-24
Manual Mapping	3-24
Manual Mapping: Examples	3-25
About Output Formats	3-27
JDBC Output Format	3-27

Oracle OCI Direct Path Output Format	3-28
Delimited Text Output Format	3-28
Oracle Data Pump Output Format	3-30
Run a Loader Job	3-31
Specify Hive Input Format JAR Files	3-32
Specify Oracle NoSQL Database Input Format JAR Files	3-32
Job Reporting	3-32
Handling Rejected Records	3-32
Log Rejected Records in Bad Files	3-33
Set a Job Reject Limit	3-33
Balancing Loads When Loading Data into Partitioned Tables	3-33
Use the Sampling Feature	3-33
Tuning Load Balancing	3-33
Tuning Sampling Behavior	3-34
When Does Oracle Loader for Hadoop Use the Sampler's Partitioning Scheme?	3-34
Resolve Memory Issues	3-35
What Happens When a Sampling Feature Property Has an Invalid Value?	3-35
Optimize Communications Between Oracle Engineered Systems	3-35
Oracle Loader for Hadoop Configuration Property Reference	3-36

4 Ease of Use Tools for Oracle Big Data Connectors

Introducing Oracle Shell for Hadoop Loaders	4-1
Configure Oracle Shell for Hadoop Loaders Interface (OHSI)	4-1
Get Started with Oracle Shell for Hadoop Loaders	4-3
Use Oracle SQL Developer With Oracle Big Data Connectors	4-8

Part III Oracle XQuery for Apache Hadoop

5 Using Oracle XQuery for Apache Hadoop

What Is Oracle XQuery for Hadoop?	5-1
Get Started With Oracle XQuery for Hadoop	5-3
Basic Steps	5-3
Example: Hello World!	5-3
About the Oracle XQuery for Hadoop Functions	5-4
About the Adapters	5-4
About Other Modules for Use With Oracle XQuery for Hadoop	5-6
Create 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
Call Custom Java Functions from XQuery	5-8
Access User-Defined XQuery Library Modules and XML Schemas	5-8
XQuery Transformation Examples	5-9
Run Queries	5-14
Oracle XQuery for Hadoop Options	5-15
Generic Options	5-16
About Running Queries Locally	5-16
Run Queries from Apache Oozie	5-17
Use Oozie with Oracle XQuery for Hadoop Action	5-17
Supported XML Elements	5-17
Example: Hello World	5-18
Oracle XQuery for Hadoop Configuration Properties	5-19

6 Oracle XQuery for Apache Hadoop Reference

Avro File Adapter	6-1
Built-in Functions for Reading Avro Files	6-2
avro:collection-avroxml	6-2
avro:get	6-3
Custom Functions for Reading Avro Container Files	6-3
Custom Functions for Writing Avro Files	6-5
Examples of Avro File Adapter Functions	6-6
About Converting Values Between Avro and XML	6-8
Reading Avro as XML	6-8
Writing XML as Avro	6-13
JSON File Adapter	6-17
Built-in Functions for Reading JSON	6-17
json:collection-jsonxml	6-18
json:parse-as-xml	6-18
json:get	6-18
Custom Functions for Reading JSON Files	6-19
Examples of JSON Functions	6-20
JSON File Adapter Configuration Properties	6-21
About Converting JSON Data Formats to XML	6-22
About Converting JSON Objects to XML	6-23
About Converting JSON Arrays to XML	6-23
About Converting Other JSON Types	6-23
Oracle Database Adapter	6-24
Custom Functions for Writing to Oracle Database	6-24

Examples of Oracle Database Adapter Functions	6-28
Oracle Loader for Hadoop Configuration Properties and Corresponding %oracle-property Annotations	6-30
Oracle NoSQL Database Adapter	6-32
Prerequisites for Using the Oracle NoSQL Database Adapter	6-33
Built-in Functions for Reading from and Writing to Oracle NoSQL Database	6-33
kv:collection-text	6-34
kv:collection-avroxml	6-34
kv:collection-xml	6-35
kv:collection-binxml	6-35
kv:collection-tika	6-36
kv:put-text	6-36
kv:put-xml	6-36
kv:put-binxml	6-37
kv:get-text	6-37
kv:get-avroxml	6-37
kv:get-xml	6-37
kv:get-binxml	6-37
kv:get-tika	6-37
kv:key-range	6-38
kv:key-range	6-38
Built-in Functions for Reading from and Writing to Oracle NoSQL Database using Table API	6-38
kv-table:collection-jsontext	6-39
kv-table:get-jsontext	6-39
kv-table:put-jsontext	6-40
Built-in Functions for Reading from and Writing to Oracle NoSQL Database using Large Object API	6-40
kv-lob:get-text	6-41
kv-lob:get-xml	6-41
kv-lob:get-binxml	6-41
kv-lob:get-tika	6-41
kv-lob:put-text	6-41
kv-lob:put-xml	6-42
kv-lob:put-binxml	6-42
Custom Functions for Reading Values from Oracle NoSQL Database	6-42
Custom Functions for Retrieving Single Values from Oracle NoSQL Database	6-45
Custom Functions for Reading Values from Oracle NoSQL Database using Table API	6-46
Custom Functions for Reading Single Row from Oracle NoSQL Database using Table API	6-47
Custom Functions for Retrieving Single Values from Oracle NoSQL Database using Large Object API	6-48

Custom Functions for Writing to Oracle NoSQL Database	6-48
Custom Functions for Writing Values to Oracle NoSQL Database using Table API	6-49
Custom Functions for Writing Values to Oracle NoSQL Database using Large Object API	6-50
Examples of Oracle NoSQL Database Adapter Functions	6-50
Oracle NoSQL Database Adapter Configuration Properties	6-56
Sequence File Adapter	6-58
Built-in Functions for Reading and Writing Sequence Files	6-59
seq:collection	6-59
seq:collection-xml	6-59
seq:collection-binxml	6-60
seq:collection-tika	6-60
seq:put	6-61
seq:put-xml	6-61
seq:put-binxml	6-62
Custom Functions for Reading Sequence Files	6-63
Custom Functions for Writing Sequence Files	6-64
Examples of Sequence File Adapter Functions	6-66
Solr Adapter	6-68
Prerequisites for Using the Solr Adapter	6-68
Configuration Settings	6-68
Example Query Using the Solr Adapter	6-68
Built-in Functions for Loading Data into Solr Servers	6-69
solr:put	6-69
Custom Functions for Loading Data into Solr Servers	6-69
Examples of Solr Adapter Functions	6-70
Solr Adapter Configuration Properties	6-71
Text File Adapter	6-73
Built-in Functions for Reading and Writing Text Files	6-73
text:collection	6-74
text:collection-xml	6-74
text:put	6-75
text:put-xml	6-75
text:trace	6-76
Custom Functions for Reading Text Files	6-76
Custom Functions for Writing Text Files	6-78
Examples of Text File Adapter Functions	6-79
Tika File Adapter	6-82
Built-in Library Functions for Parsing Files with Tika	6-82
tika:collection	6-82
tika:parse	6-83

Custom Functions for Parsing Files with Tika	6-83
Tika Parser Output Format	6-84
Tika Adapter Configuration Properties	6-84
Examples of Tika File Adapter Functions	6-85
XML File Adapter	6-86
Built-in Functions for Reading XML Files	6-86
xmlf:collection (Single Task)	6-86
xmlf:collection-multipart (Single Task)	6-87
xmlf:collection (Multiple Tasks)	6-87
Custom Functions for Reading XML Files	6-88
Examples of XML File Adapter Functions	6-91
Utility Module	6-93
Oracle XQuery Functions for Duration, Date, and Time	6-93
ora-fn:date-from-string-with-format	6-93
ora-fn:date-to-string-with-format	6-94
ora-fn:dateTime-from-string-with-format	6-94
ora-fn:dateTime-to-string-with-format	6-95
ora-fn:time-from-string-with-format	6-96
ora-fn:time-to-string-with-format	6-96
Format Argument	6-97
Locale Argument	6-97
Oracle XQuery Functions for Strings	6-97
ora-fn:pad-left	6-97
ora-fn:pad-right	6-98
ora-fn:trim	6-99
ora-fn:trim-left	6-100
ora-fn:trim-right	6-100
Hadoop Module	6-100
Built-in Functions for Using Hadoop	6-101
oxh:find	6-101
oxh:increment-counter	6-101
oxh:println	6-102
oxh:println-xml	6-102
oxh:property	6-102
Serialization Annotations	6-102

7 Oracle XML Extensions for Hive

What are the XML Extensions for Hive?	7-1
Use the Hive Extensions From the Command Line	7-2
Use the Hive Extensions in HiveServer2	7-3

About the Hive Functions	7-6
Permanently Declaring the Hive Functions	7-6
Create XML Tables	7-7
Hive CREATE TABLE Syntax for XML Tables	7-7
CREATE TABLE Configuration Properties	7-8
CREATE TABLE Examples	7-9
Syntax Example	7-9
Simple Examples	7-10
OpenStreetMap Examples	7-13
Oracle XML Functions for Hive Reference	7-15
Data Type Conversions	7-15
Hive Access to External Files	7-15
Online Documentation of Functions	7-16
xml_exists	7-17
xml_query	7-18
xml_query_as_primitive	7-20
xml_table	7-24

Part IV Oracle R Advanced Analytics for Apache Hadoop

8 Oracle R Advanced Analytics for Apache Hadoop

About Oracle R Advanced Analytics for Hadoop	8-1
Oracle R Advanced Analytics for Hadoop Architecture	8-2
Oracle R Advanced Analytics for Hadoop packages and functions	8-2
Oracle R Advanced Analytics for Hadoop APIs	8-3
Inputs to Oracle R Advanced Analytics for Hadoop	8-4
Access to HDFS Files	8-5
Access to Apache Hive	8-5
ORCH Functions for Hive	8-5
ORE Functions for Hive	8-5
Generic R Functions Supported in Hive	8-6
Support for Hive Data Types	8-7
Usage Notes for Hive Access	8-9
Example: Loading Hive Tables into Oracle R Advanced Analytics for Hadoop	8-10
Access to Oracle Database	8-10
Usage Notes for Oracle Database Access	8-11
Scenario for Using Oracle R Advanced Analytics for Hadoop with Oracle R Enterprise	8-11
Oracle R Advanced Analytics for Hadoop Functions	8-11
Native Analytical Functions	8-12

Using the Hadoop Distributed File System (HDFS)	8-13
Using Apache Hive	8-13
Using Aggregate Functions in Hive	8-14
Making Database Connections	8-14
Copying Data and Working with HDFS Files	8-15
Converting to R Data Types	8-15
Using MapReduce	8-16
Debugging Scripts	8-17
Demos of Oracle R Advanced Analytics for Hadoop Functions	8-18
Security Notes for Oracle R Advanced Analytics for Hadoop	8-19

Part V Oracle DataSource for Apache Hadoop

9 Oracle DataSource for Apache Hadoop (OD4H)

Operational Data, Big Data and Requirements	9-1
Overview of Oracle DataSource for Apache Hadoop (OD4H)	9-1
Opportunity with Hadoop 2.x	9-2
Oracle Tables as Hadoop Data Source	9-2
External Tables	9-3
TBLPROPERTIES	9-4
SERDE PROPERTIES	9-6
List of jars in the OD4H package	9-6
How does OD4H work?	9-6
Create a new Oracle Database Table or Reuse an Existing Table	9-7
Hive DDL	9-7
Create External Tables in Hive	9-8
Features of OD4H	9-9
Performance And Scalability Features	9-9
Splitters	9-10
Choosing a Splitter	9-12
Predicate Pushdown	9-13
Projection Pushdown	9-14
Partition Pruning	9-14
Smart Connection Management	9-15
Security Features	9-15
Improved Authentication	9-16
Use HiveQL with OD4H	9-19
Use Spark SQL with OD4H	9-19

Part VI Appendices

A OraLoaderMetadata Utility

D Oracle Big Data Connectors Accessibility Recommendations

Tips on Using Screen Readers and Braille Displays D-1

Tips on Using Screen Magnifiers D-1

F Recent Change History

Changes in Oracle Big Data Connectors Release 4.12 F-1

B Using Oracle's Hive Storage Handler for Kafka to Create a Hive External Table for Kafka Topics

C Apache License

Apache Licensed Code C-4

E Additional Big Data Connector Resources

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 R Advanced Analytics for Hadoop
- Oracle Datasource for Apache Hadoop
- [Oracle Data Integrator¹](#)

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 that have purchased support 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:

¹ Oracle Big Data Connectors includes a restricted use license for the Oracle Data Integrator when licensed on an Oracle Big Data Appliance. However, additional licensing is required for using it on other Hadoop clusters.

- *Oracle Loader for Hadoop Java API Reference*
- *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 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 Oracle Big Data Connectors 5.0

Oracle Big Data Connectors 5.0 includes the following new features, and software revisions.

About JDBC SSL

JDBC SSL lets you create secure connections to Oracle Database. See [Using JDBC SSL](#)

About Hadoop Credential CLI

Hadoop Credential CLI lets you create a secure keystore file to store database passwords. See [Using Secure External Java KeyStore and Hadoop credential command](#).

Software Updates in This Release

Connector	Version
Oracle SQL Connector for HDFS (OSCH)	5.0.0

Connector	Version
Oracle Loader for Hadoop (OLH)	5.0.1
Oracle Shell for Hadoop Loaders (OHSH)	5.0.1
Oracle R Advanced Analytics for Hadoop (ORAAH)	2.8.1
Oracle DataSource for Apache Hadoop (OD4H)	1.3.1

 **Note:**

Oracle XQuery for Hadoop is not included in this release of Big Data Connectors 5.0. It will be included in a future release.

Recent Change History Prior to Oracle Big Data Connectors 5.0

[Appendix F](#) in this guide describes changes in the previous Oracle Big Data Connector release – 4.12. If you are upgrading from an earlier release, you may find it useful to review this information.

Part I

Setup

Part I contains the following chapter:

- [Getting Started with Oracle Big Data Connectors](#)

1

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](#)
- [Download and Install Oracle Big Data Connectors](#)
- [Oracle SQL Connector for Hadoop Distributed File System Setup](#)
- [Oracle Loader for Hadoop Setup](#)
- [Oracle Shell for Hadoop Loaders Setup](#)
- [Oracle XQuery for Hadoop Setup](#)
- [Oracle R Advanced Analytics for Hadoop Setup](#)
- [Oracle Data Integrator](#)
- [Oracle Datasource for Apache Hadoop Setup](#)

About Oracle Big Data Connectors

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

These are the connectors:

- **Oracle SQL Connector for Hadoop Distributed File System:** 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 Loader for Apache 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 XQuery for Apache 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, Apache Solr, or Oracle Database. An additional XML processing capability is through XML Extensions for Hive.

- **Oracle Shell for Hadoop Loaders:** A helper shell that provides a simple-to-use command line interface to Oracle Loader for Hadoop, Oracle SQL Connector for HDFS, and Copy to Hadoop (a feature of Big Data SQL). It has basic shell features such as command line recall, history, inheriting environment variables from the parent process, setting new or existing environment variables, and performing environmental substitution in the command line.
- **Oracle R Advanced Analytics for Apache Hadoop:** Provides a general computation framework, in which you can use the R language to write your custom logic as mappers or reducers. A collection of R packages provides predictive analytic techniques that run as MapReduce jobs. The code executes in a distributed, parallel manner using the available compute and storage resources on the Hadoop cluster. Oracle R Advanced Analytics for Hadoop includes interfaces to work with Apache Hive tables, the Apache Hadoop compute infrastructure, the local R environment, and Oracle database tables.
- **Oracle Data Integrator:** Extracts, loads, and transforms data from sources such as files and databases into Hadoop and from Hadoop into Oracle or third-party databases. Oracle Data Integrator provides a graphical user interface to utilize the native Hadoop tools and transformation engines such as Hive, HBase, Sqoop, Oracle Loader for Hadoop, and Oracle SQL Connector for Hadoop Distributed File System.
- **Oracle Datasource for Hadoop:** Provides direct, fast, parallel, secure and consistent access to master data in Oracle Database using Hive SQL, Spark SQL, as well as Hadoop APIs that support SerDes, HCatalog, InputFormat and StorageHandler.

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. For details on integrating Oracle Database and Apache Hadoop visit the [Certification Matrix](#).



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 quickly 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 3.0 and later supports 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 system. 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 (CDH5), visit the Oracle Cloudera website.
- For information about Apache Hadoop, visit the Apache Hadoop website.

Download and Install Oracle Big Data Connectors

You can download Oracle Big Data Connectors from Oracle Technology Network or Oracle Software Delivery Cloud. Both sites are cross-browser compatible.

 **Note:**

Oracle Big Data Appliance customers do not need to download Oracle Big Data Connectors from an external source. Oracle Big Data Connectors are included in the Oracle Big Data Appliance deployment bundle. See Enabling and Disabling Oracle Big Data Connectors in the *Oracle Big Data Appliance Owner's Guide*. All other customers should download the software as described here.

To download from Oracle Technology Network:

1. Go to <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. Go to <https://edelivery.oracle.com/>
2. Sign in and accept the Export Restrictions.
3. Type in the product name in the Product field and select the platform:
Product: Oracle Big Data Connectors
Platform: Linux x86-64
4. When Oracle Big Data Connectors appears in the Product List, click **Continue**. The most recent major release of Oracle Big Data Connectors will appear as the selected option.
5. To choose a different release, click **Select Alternate Release** and choose another package from the list. Click **Continue**.

6. Read the Terms and Conditions. Click the checkbox if you accept them, then click **Continue**.
7. On the download site, select the zip files for individual Oracle Big Data Connectors or click **Download All**.

Each download package includes a README file with installation instructions and a set of examples.

Certified Hadoop Platforms

All Oracle Big Data Connectors run on both CDH (Cloudera Distribution Including Apache Hadoop) and HDP (Hortonworks Data Platform).

See the [Oracle Big Data Connectors Certification Matrix](#) for currently supported releases of CDH and HDP.

Note:

Big Data Connectors 5.0 is required to work with CDH 6.x. All components with the exception of Oracle Data Integrator are certified to work with CDH 6.x.

Note:

Oracle Loader for Hadoop 5.0.1 and Oracle SQL Connector for HDFS 5.0.1 are certified to work with Hortonworks 3.0.1.

Secure Connection to Oracle Database

Describes using JDBC SSL and Java KeyStore to securely connect to Oracle Database.

Using JDBC SSL

Follow these steps to connect to the Oracle Database using JDBC SSL:

1. Download the SSL wallet.zip file. For example, see [Download Client Credentials](#) for information on downloading client credentials for Oracle Autonomous Data Warehouse.
2. Copy the wallet.zip file to a directory location on a node in the Hadoop cluster or Hadoop edge node (from where you will run Oracle Loader for Hadoop or Oracle SQL Connector for HDFS or Oracle Shell for Hadoop Loaders). This is your `TNS_ADMIN` directory on the Hadoop node. For example: `/home/oracle/SSL_wallet`.
3. Unzip the wallet.zip file.
4. Set the permissions of the file to be accessible to the OS user who will run Oracle Loader for Hadoop or Oracle SQL Connector for HDFS or Oracle Shell for Hadoop Loaders.

5. In the `sqlnet.ora` file, update the `WALLET_LOCATION` entry so that the `DIRECTORY` points to the directory location of the SSL wallet you created in Step 2.

For Example:

```
WALLET_LOCATION=
    (SOURCE=
      (METHOD=FILE)
      (METHOD_DATA=
        (DIRECTORY=/home/oracle/SSL_wallet/)))
```

6. In the `tnsnames.ora` file, find the name of the TNS entry using TCPS protocol. For example:

```
inst1_ssl = (DESCRIPTION=(ADDRESS=
    (PROTOCOL=tcps)
    (HOST=<hostname>)
    (PORT=1521))
(CONNECT_DATA=(SERVICE_NAME=<service_name>)))
```

7. Create a `connection.properties` file in the `TNS_ADMIN` directory and add the following code:

```
javax.net.ssl.trustStore=<directory_location>/cwallet.sso
javax.net.ssl.trustStoreType=SSO
javax.net.ssl.keyStore=<directory_location>/cwallet.sso
javax.net.ssl.keyStoreType=SSO
```

For example:

```
javax.net.ssl.trustStore=/home/oracle/SSL_wallet/cwallet.sso
javax.net.ssl.trustStoreType=SSO
javax.net.ssl.keyStore=/home/oracle/SSL_wallet/cwallet.sso
javax.net.ssl.keyStoreType=SSO
```

Using Secure External Java KeyStore and Hadoop credential command

Oracle Shell for Hadoop Loaders (OHS) can use a secure external Java KeyStore (JKS) file to store database passwords. The Hadoop credential command can be used to create the Java KeyStore file.

- Use the following syntax to create the keyStore file using the Hadoop credential command:

```
$ hadoop credential create <password alias>
    -value <mypasswd>
    -provider jceks://file/<directory location where the keystore
file will be stored>/<keystorefilename>.jceks
```


For example:

```
$ hadoop credential create oracle_passwd  
-value password  
-provider jceks://file/home/oracle/passwd.jceks
```

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.

On Oracle Big Data Appliance, there is an option to include Oracle Big Data Connectors in the installation. If this was selected in the Configuration Generation Utility, then Oracle SQL Connector for HDFS is already available on the appliance. (See Chapter 4, *Using Oracle Big Data Appliance Configuration Generation Utility*, in the *Big Data Appliance Owner's Guide*.)

This section contains the following topics:

- [Software Requirements](#)
- [Install and Configure a Hadoop Client on the Oracle Database System](#)
- [Install Oracle SQL Connector for HDFS](#)
- [Oracle Database Privileges for OSCH Users](#)
- [OS-Level Requirements for OSCH Users](#)
- [Use Oracle SQL Connector for HDFS on a Secure Hadoop Cluster](#)

Software Requirements

Oracle SQL Connector for HDFS requires the following software:

Hadoop Requirements:

- A certified release of either CDH (Cloudera Distribution Including Apache Hadoop) or HDP (Hadoop Data Platform).
- Java Development Kit (JDK). Consult the distributor of your Hadoop software (Cloudera or Apache) for the recommended version.

For supported CDH and HDP releases, see the [Oracle Big Data Connectors Certification Matrix](#)

Oracle Big Data Appliance already meets these requirements. If you are using CDH or HDP on a commodity server platform, check to ensure that the system meets them.

Oracle Database System and Hadoop Client System Requirements:

- A version of Oracle Database that is currently supported by Oracle.
- The same version of Hadoop that is installed on your Hadoop cluster: CDH5, or Hortonworks Data Platform 2.4.0 and 2.5.0. .

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.

- The same version of the JDK that is installed on your Hadoop cluster.

 **Note:**

Oracle SQL Connector for HDFS requires a Hadoop client on the OS platform of the database system. This is straightforward for Linux systems. Platforms other than Linux require a tarball installation of the Hadoop client. Refer to this Oracle Blog post [Connecting Hadoop with Oracle](#). See the following documents in My Oracle Support for details:

- [Installation Instructions for Oracle SQL Connector for HDFS on Solaris \(Doc ID 2101331.1\)](#)
- [Using Oracle Big Data Connectors with Hadoop clusters on commodity hardware and Oracle Databases on commodity hardware \(Doc ID 2101354.1\)](#)
- [Configuring Oracle SQL Connector for HDFS for Oracle Database systems on IBM AIX \(Doc ID 2152000.1\)](#)

 **See Also:**

[Use Oracle SQL Connector for HDFS on a Secure Hadoop Cluster.](#)

Install and Configure a Hadoop Client on the Oracle Database System

Oracle SQL Connector for HDFS requires a Hadoop client on the Oracle Database System. The Hadoop installation can be minimally configured for Hadoop client use only. The full configuration of Hadoop is not needed. The only parts of Hadoop needed for Oracle SQL Connector for HDFS are the Hadoop JAR files and the configuration files from the Hadoop installation.

 **Note:**

Even if there is a complete Hadoop installation on the Oracle Database system, do not start Hadoop on this system at any time. If Hadoop is running locally, then Oracle SQL Connector for HDFS attempts to connect to it instead of to the Hadoop cluster.

For Oracle RAC systems including Oracle Exadata Database Machine, you must install and configure Oracle SQL Connector for HDFS using identical paths on all systems running Oracle instances.

Add a Hadoop Client for use with Oracle Big Data Appliance

Oracle Big Data Appliance requires that you follow its own system-supported procedures for installing a Hadoop client. If your Hadoop system is an Oracle Big Data Appliance, see Provide Remote Access to CDH in the *Oracle Big Data Appliance*

Software User's Guide. This section describes how to install the CDH client, configure it for use in a Kerberos-secured or non-secured environment, and verify HDFS access.

Add a Hadoop Client for use with Other Hadoop Systems

For connections to Hadoop systems other than Oracle Big Data Appliance, download and install the Hadoop client provide by the Hadoop distributor (Cloudera or Apache). The following example shows how to connect a Hadoop client to a CDH system that is not an Oracle Big Data Appliance. You can use these steps to install the client, configure it for use in Kerberos-secured or a non-secured environment, and test to verify HDFS access.

In this case also you need set up Kerberos access (if Kerberos is installed) and should include a final test to make sure HDFS access is working.

1. Use one of these methods to obtain the files:
 - Download the tarball from the Cloudera tarball downloads page. Check that the Hadoop version in the filename (as in `hadoop-2.5.0-cdh5.2.5.tar.gz`) that matches the version of the Hadoop cluster.
 - Click on the **hdfs** service in Cloudera Manager, and select the action **Download Client Configuration**.
2. Extract the files and copy them to a permanent path of your choice on the database system.
3. Set the `HADOOP_HOME` environment variable to this path and add `HADOOP_HOME/bin` to the `PATH` variable.
4. Ensure that `JAVA_HOME` points to a JDK installation with the version required by the Hadoop installation.
5. If your cluster is secured with Kerberos, then configure the Oracle system to permit Kerberos authentication. See "[Using Oracle SQL Connector for HDFS on a Secure Hadoop Cluster](#)."
6. Test HDFS access from the Oracle Database system:
 - 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:

```
$ hdfs dfs -ls /user
```

You might need to add the directory containing the Hadoop executable file to the 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 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.

7. For an Oracle RAC system, repeat this procedure for every Oracle Database instance.

Set up Hadoop Clients on Additional Systems

You have the option to set up Hadoop clients on other servers in addition to the Oracle Database system. If you do, use the procedure provided in this section and install the same version of CDH or Apache Hadoop consistently.

Install Oracle SQL Connector for HDFS

Follow this procedure to install Oracle SQL Connector for HDFS on the Oracle Database system.

In addition to this required installation on the database system, you can also install Oracle SQL Connector for HDFS on any system configured as a compatible Hadoop client. This will give you the option to create Oracle Database external tables from that node.

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-<version>.zip
Archive:  oraosch-<version>.zip
  extracting:  orahdfs-<version>.zip
  inflating:  README.txt
```

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

```
$ unzip orahdfs-<version>.zip
unzip orahdfs-<version>.zip
Archive:  orahdfs-<version>.zip
  creating:  orahdfs-<version>/
  creating:  orahdfs-<version>/log/
  inflating:  orahdfs-<version>/examples.zip
  creating:  orahdfs-<version>/doc/
  inflating:  orahdfs-<version>/doc/README.txt
  creating:  orahdfs-<version>/jlib/
  inflating:  orahdfs-<version>/jlib/osdt_cert.jar
  inflating:  orahdfs-<version>/jlib/oraclepki.jar
  inflating:  orahdfs-<version>/jlib/osdt_core.jar
  inflating:  orahdfs-<version>/jlib/ojdbc7.jar
  inflating:  orahdfs-<version>/jlib/orahdfs.jar
  inflating:  orahdfs-<version>/jlib/ora-hadoop-common.jar
  creating:  orahdfs-<version>/bin/
  inflating:  orahdfs-<version>/bin/hdfs_stream
  inflating:  orahdfs-<version>/bin/hdfs_stream.cmd
```

The unzipped files have the structure shown in [Example 1-1](#). The `examples.zip` file is not unzipped. Unzip that file later when you want to work with the examples.

4. Open the `orahdfs-<version>/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`.

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 and the account does not already have a Kerberos ticket, then obtain one:

```
$ kinit
```

See [Use Oracle SQL Connector for HDFS on a Secure Hadoop Cluster](#) in this guide for information on acquiring and maintaining Kerberos tickets.

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. For an Oracle RAC system, repeat the previous steps for every Oracle instance, using identical path locations.
8. Log in to Oracle Database and create a database directory for the `orahdfs-<version>/bin` directory where `hdfs_stream` resides. For Oracle RAC systems, this directory must be accessible by all Oracle instances through identical paths.

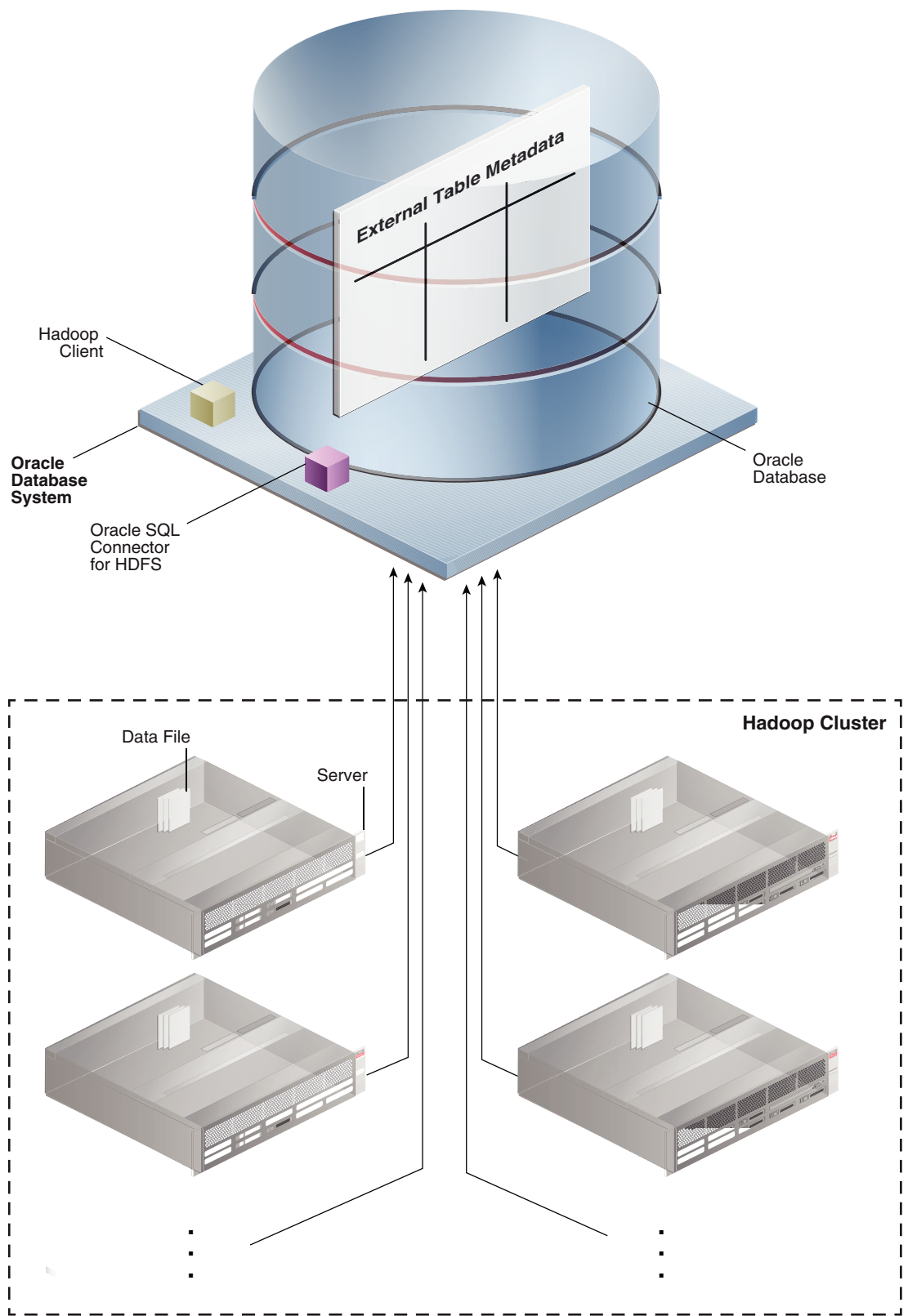
In this example, Oracle SQL Connector for HDFS is installed in `/etc`:

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

9. To support access to Hive tables:
 - a. Ensure that the system is configured as a Hive client.
 - b. 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 your local shell initialization script instead of making a global change to `HADOOP_CLASSPATH`. If there are multiple JAR file paths in `HADOOP_CLASSPATH` ensure that the JARs for the current product are listed first.

The following figure 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



The unzipped files have the structure shown in the following example.

Example 1-1 Structure of the orahdfs Directory

```
orahdfs-<version>
  bin/
    hdfs_stream
    hdfs_stream.cmd
  doc/
    README.txt
  jlib/
    ojdbc7.jar
    oraloader.jar
    ora-hadoop-common.jar
    oraclepki.jar
    orahdfs.jar
    osdt_cert.jar
    osdt_core.jar
  log/
  examples.zip
```

Oracle Database Privileges for OSCH Users

Oracle Database users require these privileges in order to use Oracle SQL Connector for HDFS to create external tables:

- CREATE SESSION
- CREATE TABLE
- CREATE VIEW
- 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. For Oracle RAC systems, this directory must be on a shared disk that all Oracle instances can access.
- A tablespace and quota for copying data into the Oracle database. Optional.

The following example shows the SQL commands granting these privileges to HDFSUSER.

Note:

To query an external table that uses Oracle SQL Connector for HDFS, users need READ privilege for Oracle Database 12c or later and SELECT privilege for older versions of the database.

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, CREATE VIEW 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;

```

OS-Level Requirements for OSCH Users

Wherever Oracle SQL Connector for HDFS is installed (on the Oracle Database system, a Hadoop cluster node, or a separate system set up as a Hadoop client), the OS-level user account that logs in to use OSCH requires access to the shell variable `HADOOP_CLASSPATH`. This variable must include the OSCH path on the Hadoop cluster – `path/orahdfs-<version>/jlib/*`

Set the `HADOOP_CLASSPATH` as shown in the following example, where the OSCH path is prepended to the current `HADOOP_CLASSPATH`. Putting OSCH first gives it precedence over other JARs in the path.

```
$ export HADOOP_CLASSPATH="/etc/orahdfs-<version>/jlib/*:$HADOOP_CLASSPATH"
```

Use 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 behaves like a Hadoop client. On the system where the Oracle database is running, it connects as the OS user of the Oracle process (usually `oracle`). For OSCH to work, this account requires read permission on the files of all OSCH users. On a non-secure cluster these files are world-readable, but on a Kerberos-secured cluster this access requires a Kerberos ticket.

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`. For example, if Kerberos tickets expire every two weeks, then set up a `cron` job to renew the ticket weekly.

Be sure to schedule the `cron` job to run when Oracle SQL Connector for HDFS is not actively being used.

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.

Use OSCH in Oracle SQL Developer

Oracle SQL Developer is a free graphical IDE that includes integration with Oracle Database and Oracle Big Data Connectors (among many other products). It provides wizards to help you access and use Oracle Big Data Connectors. See [Use Oracle SQL Developer With Oracle Big Data Connectors](#) in this guide for instructions on downloading Oracle SQL Developer and configuring it for use with Oracle Big Data Connectors.

Oracle Loader for Hadoop Setup

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

- [Software Requirements](#)
- [Install Oracle Loader for Hadoop](#)
- [Oracle Database Privileges for OLH Users](#)
- [Provide Support for Offline Database Mode](#)
- [Use Oracle Loader for Hadoop on a Secure Hadoop Cluster](#)

Software Requirements

Oracle Loader for Hadoop requires the following software:

- A certified release of CDH or HDP.
See the [Oracle Big Data Connectors Certification Matrix](#)
- A target database system running a version of Oracle Database that is currently supported by Oracle.

Oracle Big Data Appliance already meets these requirements. If you are using CDH or HDP on a commodity server platform, check to ensure that the system meets them.

Install Oracle Loader for Hadoop

Oracle Loader for Hadoop is packaged with the Oracle Database 12c (12.1.0.2 and 12.2.0.1) client libraries and Oracle Instant Client libraries for connecting to Oracle Database 11.2.0.4, 12.1.0.2, or 12.2.0.1.

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.
2. Unzip `oraloader-<version>-h2.x86_64.zip` into a directory on your Hadoop cluster.

A directory named `oraloader-<version>-h2` is created with the following subdirectories along with the `examples.zip` file, which you must unzip yourself.

```
doc
jlib
lib
examples.zip
```

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/*
```

When using OLH, `$OLH_HOME/jlib/*` should always be listed first in `HADOOP_CLASSPATH`. Alternatively, you can avoid conflict with other scripts by defining `HADOOP_CLASSPATH` within a script that uses it.

- To support data loads from Hive tables:

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

See "[oracle.hadoop.xquery.lib.share.](#)"

- To read data from Oracle NoSQL Database Release 2:

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

Oracle Database Privileges for OLH Users

Oracle Database users require these privileges in order to use Oracle Loader for Hadoop to load data into the table:

- `CREATE SESSION` (to connect to the database).
- A tablespace and quota for inserting rows into the table.



Note:

As a OLH user, you must own the table. If you don't own the table, then you need additional privileges to access `DBMS_METADATA`. See [DBMS_METADATA](#)

Provide 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.

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.

- 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. You must use the same version of the software as you installed on the Hadoop cluster.
2. Unzip `oraloader-<version>-h2.x86_64.zip`.
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-<version>-h2/"
```

4. Add the Oracle Loader for Hadoop JAR files to the `HADOOP_CLASSPATH` environment variable. If there are other JAR file paths in `HADOOP_CLASSPATH`, ensure that the Oracle Loader for Hadoop JAR file path is listed first when using Oracle Loader for Hadoop. This example uses the Bash shell syntax:

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

Related Topics

- [About the Modes of Operation](#)

Use 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.

Oracle Big Data Appliance configures Kerberos security automatically as a configuration option.

Oracle Shell for Hadoop Loaders Setup

Oracle Shell for Hadoop Loaders (OSHL) is integrated with Big Data Connectors. It provides a set of declarative commands you can use to load content from Hadoop and Hive to Oracle Database tables using Oracle Loader for Hadoop (OLH) and Oracle SQL Connector for Hadoop Distributed File System (OSCH). It also enables you to

load contents from Oracle Database tables to Hadoop and Hive using Copy to Hadoop.

Prerequisites


OHSB can work with OLH and OSCH to load data into Oracle Database tables from any of these environments:

- A Hadoop client
- An edge node
- A Hadoop node
- The Oracle Database server

OHSB can be set up in any of the environments above (Hadoop client, Hadoop node, edge node, or Oracle Database server). To use OHSB, you need to install the software in only one of these environments.

Each environment has its own prerequisites. Check the relevant column in table below and install any software packages that are missing from the environment where you choose to run OHSB. As the table indicates, JDBC connectivity is required in all cases.

Table 1-1 Prerequisites for Running OHSB

If You Plan to run OHSB From...	The Prerequisites are...
A Hadoop node, a Hadoop client, or an edge node.	<ul style="list-style-type: none"> • SQL*Plus • OLH and OSCH • JDBC access to Oracle Database
<div style="border: 1px solid #0070C0; padding: 10px; background-color: #E6F2FF;"> <p> Note:</p> <p>When Big Data Connectors is licensed on Oracle Big Data Appliance, all of the prerequisite software and OHSB itself are pre-installed.</p> </div>	
The Oracle Database server.	<ul style="list-style-type: none"> • Hadoop and Hive libraries (installed and configured). • OLH and OSCH • JDBC access to Oracle Database

Installing Oracle Shell for Hadoop Loaders

Follow these instructions for setting up Oracle Shell for Hadoop Loaders. The instructions are applicable to set up on a Hadoop client, an edge node, a Hadoop node, or, on the Oracle Database server.

1. Extract the contents of ohsh-*<version>*.zip to a directory on the system where you plan to run OHSB.

The extraction creates a directory named `ohsh-<version>` with a `README.txt` file, the examples package (`examples.zip`) and four subdirectories:

```
README.txt
examples.zip
/bin
/conf
/doc
/jlib
```

You must unzip `examples.zip` yourself.

The directory `ohsh-<version>` is referred to as `<OHS_HOME>` later in these instructions.

2. Follow the instructions contained in `README.txt` to configure Oracle Shell for Hadoop Loaders. Below are instructions to install and configure Oracle Shell for Hadoop Loaders on a Hadoop node.

Install Oracle Shell for Hadoop Loaders on a Hadoop Node

This procedure applies to installation on a Hadoop Node only. See the `OHS README.txt` file for installation on other systems.

Note:

These instructions use placeholders for the absolute paths that you will set as the value of some variables. These are in italic font and are framed in brackets. For example, `<OHS_HOME>` is the path where OHS is installed.

1. Install and set up SQL*Plus if it is not already on the node.
 - Download the Oracle Instant Client for Linux along with the corresponding Instant Client Package for SQL*Plus from the [Oracle Technology Network](http://www.oracle.com/technetwork/topics/linuxx86-64soft-092277.html). Select the client version that matches the version of the Oracle Database.

For example, you can find the client downloads for Oracle Database 12.2.0.1.0 at this address:
<http://www.oracle.com/technetwork/topics/linuxx86-64soft-092277.html>
2. Extract both clients into the same directory (`<SQLPLUS_DIR>`).
3. Copy the `tnsnames.ora` and `sqlnet.ora` files from `$(TNS_ADMIN)` on the Oracle Database host to a directory of your choice on the Hadoop node (`<TNS_ADMIN_DIR>`).

If an Oracle Wallet is created for the Oracle Database host, copy the wallet file to a directory of your choice on the Hadoop node (`<WALLET_LOCATION_DIR>`).
4. Edit `sqlnet.ora` . Set `WALLET_LOCATION` to `<WALLET_LOCATION_DIR>`. Also check to be sure that `sqlnet.wallet_override` is set to "true".

```
WALLET_LOCATION=
(SOURCE=(METHOD=FILE)(METHOD_DATA=
```

```
(DIRECTORY=<WALLET_LOCATION_DIR>)))  
sqlnet.wallet_override=true
```

5. Install OLH and OSCH on the Hadoop node if they are not already installed.

Note that OSCH requires installation and configuration steps on the Oracle Database host as well as on the Hadoop node. For both OLH and OSCH, follow the setup instructions in the *Big Data Connectors User's Guide*.

6. Edit <OHS_HOME>/bin/ohsh_config.sh, to configure the home directories of OHS dependencies

```
export HADOOP_HOME=<HADOOP_CLIENT_KIT>  
export HADOOP_CONF_DIR=<HADOOP_CONF>  
export HIVE_HOME=<HIVE_CLIENT_KIT>  
export HIVE_CONF_DIR=<HIVE_CONF>  
export OLH_HOME=<OLH_HOME>  
export OSCH_HOME=<OSCH_HOME>  
export CP2HADOOP_HOME=<CP2HADOOP_HOME>  
export HS2_HOST_PORT=<HS2_HOST>:<HS2_PORT>  
export HIVE_SESS_VAR_LIST=<semicolon_separated_variable_list>
```

7. If TNS admin and Oracle Wallet are enabled, then also set the following variables:

```
export WALLET_LOCATION="<WALLET_LOCATION_DIR>"  
export TNS_ADMIN="<TNS_ADMIN_DIR>"
```

These values are assigned to OHS tnsadmin and walletlocation defaults at the start of an OHS session. They are used for all TNS and Oracle Wallet authentication in the session.

8. Add <OHS_HOME>/bin to the PATH environment variable.
9. Start an OHS session.

```
$ ohsh
```

Under the banner you should see a list of the kits that were found (OSCH, OLH, and CP2HADOOP).

You can use the show resources command to see what resources are available.

```
ohsh> show resources
```

You will always see the three predefined resources: hadoop0, hive0, and bash0.

 **See Also:**

- Unzip `examples.zip` into the installation directory at `<OHS_HOME>`. The `README.txt` file in the unzipped examples directory shows how to run OHS load methods in the examples.
- In the *Big Data Connectors User's Guide*:
 - Oracle Loader for Hadoop Setup
 - Oracle SQL Connector for Hadoop Distributed File System Setup
- In the Oracle Big Data Connectors blog space:
 - [How to Load Oracle and Hive Tables using OHS \(Part 1 - Introduction\)](#)
 - [How to Load Oracle and Hive Tables using OHS \(Part 2 - OHS Configuration and CLI Usage\)](#)

Oracle Database Privileges for OHS Users

OHS users should have these privileges to load data:

- Privileges required for Oracle Loader for Hadoop when using Oracle Loader for Hadoop to load data. See [Oracle Database Privileges for OLH Users](#).
- Privileges required for Oracle SQL Connector for HDFS when using Oracle SQL Connector for HDFS to load data. See [Oracle Database Privileges for OSCH Users](#).

Configure OHS to Enable Job Monitoring

When OHS jobs are executed, status and other information about the job is recorded into a back-end database. To access information from the OHS command line, you must first configure the connection to the database.

Configuration Steps

Configure the following properties in `conf/smartloader-conf.xml` in order to enable a database instance where job history is stored.

- `oracle.hadoop.smartloader.diagnostics.jobhistory.jdbc.driver`
Specifies the JDBC driver for the supported back-end database type. Currently, `MYSQL` and `ORACLE` are valid values. If this property is not specified, the job history commands fail.

Additional properties must be set. These differ, depending upon which database type is defined as the supported back-end database

- If `jdbc.driver = ORACLE`:
 - `oracle.hadoop.smartloader.diagnostics.jobhistory.jdbc.oracleConnectId`
A TNS entry name defined in the `tnsnames.ora` file.

- `oracle.hadoop.smartloader.diagnostics.jobhistory.jdbc.oracleWalletDir`

The OS directory containing the Oracle Wallet used to connect to an Oracle Database schema through JDBC.

- `oracle.hadoop.smartloader.diagnostics.jobhistory.jdbc.oracleTnsDir`

The file path to a directory on the node where OSH runs. This directory contains SQL*Net configuration files such as `sqlnet.ora` and `tnsnames.ora`. Typically, this is `${ORACLE_HOME}/network/admin`.

 **Note:**

If you are running OSH from a Hadoop client and want to use Oracle Wallet, copy `tnsnames.ora` and the wallet files to any directory on the Hadoop client.

- If `jdbc.driver = MYSQL`:
 - `oracle.hadoop.smartloader.diagnostics.jobhistory.jdbc.mysqlConnectURL`
The URL used to make a JDBC connection to the MySQL database
 - `oracle.hadoop.smartloader.diagnostics.jobhistory.jdbc.mysqlUser`
MySQL user of job history schema
 - `oracle.hadoop.smartloader.diagnostics.jobhistory.jdbc.mysqlPW`
Password of the MySQL user account.

Commands for Monitoring OSH Jobs

After this configuration is completed, you will be able to execute the following OSH commands:

- `ohsh> show job <job_id>`
Shows the detailed information of the job specified by ID.
- `ohsh> show job summary <job_id>`
Shows the performance of the completed job specified by ID.
- `ohsh> show job abstract <job_id>`
Provides a functional description of the job.
- `ohsh> show jobs [failed|running|completed|finished] [extended] [<integer>]`
Shows the last *n* jobs of a particular job status.
 - The first parameter specifies job status. If the status is not specified, all jobs are shown, regardless of job status.
 - The second parameter specifies whether to show details.
 - The third parameter specifies that the last *n* jobs of the specified status should be shown. If *n* is not specified, then all jobs of that status are shown.
- `ohsh> truncate jobs [<integer>]`

Removes the last n jobs from the database. If the integer is not specified, the command removes all jobs

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](#)
- [Install Oracle XQuery for Hadoop](#)
- [Troubleshoot the File Paths](#)
- [Configure Oozie for the Oracle XQuery for Hadoop Action](#)

Software Requirements

Oracle Big Data Appliance Release 4.3 and later releases meet the requirements below. However, if you are installing Oracle XQuery for Hadoop on a third-party cluster, then you must ensure that these components are installed.

- Java 8.x or 7.x.
- A certified release of either CDH (Cloudera's Distribution including Apache Hadoop) or HDP (Hortonworks Data Platform).
- Oracle NoSQL Database 3.x or 2.x to support reading and writing to Oracle NoSQL Database
- Oracle Loader for Hadoop 3.8.0 or greater to support writing tables in Oracle databases

See Also:

For supported CDH and HDP version, see the [Oracle Big Data Connectors Certification Matrix](#)

Install Oracle XQuery for Hadoop

Perform 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-<version>-cdh-<version>.zip
Archive:  oxh-<version>-cdh-<version>.zip
  creating: oxh-<version>-cdh<version>/
  creating: oxh-<version>-cdh<version>/lib/
  creating: oxh-<version>-cdh<version>/oozie/
  creating: oxh-<version>-cdh<version>/oozie/lib/
  inflating: oxh-<version>-cdh<version>/lib/ant-launcher.jar
  inflating: oxh-<version>-cdh<version>/lib/ant.jar
```

You can now run Oracle XQuery for Hadoop.

2. For the fastest execution time, copy the libraries into the Hadoop distributed cache:
 - a. Copy all Oracle XQuery for Hadoop and third-party libraries into an HDFS directory. To use the `-exportliboozie` option to copy the files, see "[Oracle XQuery for Hadoop Options](#)". Alternatively, you can copy the libraries manually using the HDFS command line interface.

If you use Oozie, then use the same folder for all files. See "[Configure Oozie for the Oracle XQuery for Hadoop Action](#)".
 - b. Set the `oracle.hadoop.xquery.lib.share` property or use the `-sharelib` option on the command line to identify the directory for the Hadoop distributed cache.
3. To support data loads into Oracle Database, install Oracle Loader for Hadoop:
 - 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 an archive and a `README` file.
 - b. Unzip the archive into a directory on your Hadoop cluster:

```
unzip oraloader-<version>-h2.x86_64.zip
```

A directory named `oraloader-<version>-h2` is created with the following subdirectories and the `examples.zip` file:

```
doc
jlib
lib
examples.zip
```

Unzip the `example.zip` file yourself.

- c. Create an environment variable named `OLH_HOME` and set it to the installation directory. Do not set `HADOOP_CLASSPATH`.
4. 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.

 **Note:**

Do not add NoSQL Database jar files to a `HADOOP_CLASSPATH`.

5. To support indexing by Apache Solr:
 - a. Ensure that Solr is installed and configured in your Hadoop cluster. Solr is included in Cloudera Search, which is installed automatically on Oracle Big Data Appliance.
 - b. Create a collection in your Solr installation into which you will load documents. To create a collection, use the `solrctl` utility.

 **See Also:**

For the `solrctl` utility, *Cloudera Search User Guide* at

http://www.cloudera.com/content/cloudera-content/cloudera-docs/Search/latest/Cloudera-Search-User-Guide/csug_solrctl_ref.html

- c. Configure Oracle XQuery for Hadoop to use your Solr installation by setting the `OXH_SOLR_MR_HOME` environment variable to the local directory containing `search-mr-<version>.jar` and `search-mr-<version>-job.jar`. For example:

```
$ export OXH_SOLR_MR_HOME="/usr/lib/solr/contrib/mr"
```

 **Note:**

Configure Oracle XQuery for Hadoop and set the `OXH_SOLR_MR_HOME` environment variable to the local directory before using Apache Tika adapter as well.

Troubleshoot the File Paths

If Oracle XQuery for Hadoop fails to find its own or third-party libraries when running queries, first ensure that the environment variables were set correctly during Oracle XQuery for Hadoop installation.

 **Note:**

The `HADOOP_CLASSPATH` environment variable or `-libjars` command line option must not contain either an OXH or third-party library.

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}`.

Related Topics

- [Install Oracle XQuery for Hadoop](#)

Configure Oozie for the Oracle XQuery for Hadoop Action

You can use Apache Oozie workflows to run your queries, as described in "[Run Queries from Apache Oozie](#)". The software is already installed and configured on Oracle Big Data Appliance.

For other Hadoop clusters, you must first configure Oozie to use the Oracle XQuery for Hadoop action. These are the general steps to install the Oracle XQuery for Hadoop action:

1. Modify the Oozie configuration. If you run CDH on third-party hardware, then use Cloudera Manager to change the Oozie server configuration. For other Hadoop installations, edit `oozie-site.htm`.
 - Add `oracle.hadoop.xquery.oozie.OXHActionExecutor` to the value of the `oozie.service.ActionService.executor.ext.classes` property.
 - Add `oxh-action-v1.xsd` to the value of the `oozie.service.SchemaService.wf.ext.schemas` property.
2. Add `oxh-oozie.jar` to the Oozie server class path. For example, in a CDH5 installation, copy `oxh-oozie.jar` to `/var/lib/oozie` on the server.
3. Add all Oracle XQuery for Hadoop dependencies to the Oozie shared library in a subdirectory named `oxh`. You can use the CLI `-exportliboozie` option. See "[Oracle XQuery for Hadoop Options](#)".
4. Restart Oozie for the changes to take effect.

The specific steps depend on your Oozie installation, such as whether Oozie is already installed and which version you are using.

Oracle R Advanced Analytics for Hadoop Setup

An overview of Oracle R Advanced Analytics for Hadoop (ORAAH) is provided in Part IV of this guide .

Release notes, installation instructions, comprehensive reference material, and a list of changes in the current release are published separately on the Oracle Technology Network.

- [ORAAH 2.8.0 Installation Guide](#).
- [Oracle R Advanced Analytics for Hadoop 2.8.0 Release Notes](#) .
- [Oracle R Advanced Analytics for Hadoop 2.8.0 Reference Manual](#).
- [ORAAH 2.8.0 Change List Summary](#).
- [ORAAH 2.8.0 Oracle Formula and Data Preprocessing](#)
- [Supported Features for Apache Hive/Impala in ORAAH 2.8.0](#)

Each ORAAH release is compatible with a number of Oracle Big Data Appliance releases and releases of CDH running on non-Oracle platforms.

For a complete ORAAH compatibility matrix, see Document [2225633.1](#) on My Oracle Support.

Install the Software on Hadoop

Oracle Big Data Appliance supports Oracle R Advanced Analytics for Hadoop without any additional software installation or configuration. However, to use Oracle R Advanced Analytics for Hadoop on a third-party 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:

- A certified release of CDH (Cloudera's Distribution including Apache Hadoop) or HDP (Hadoop Data Platform), or, Apache Hadoop 0.20.2+923.479 or later. See the [Oracle Big Data Connectors Certification Matrix](#).

Complete the instructions provided by the distributor.

- Apache Hive 0.10.0+67 or later

See "[Install Hive on a Third-Party Hadoop Cluster](#)."

- Sqoop 1.3.0+5.95 or later 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 "[Install Sqoop on a Third-Party Hadoop Cluster](#)."

- 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 3.0.1 with all base libraries on all nodes in the Hadoop cluster.

See "[Install R on a Third-Party Hadoop Cluster](#)."

- The ORCH package on each R engine, which must exist on every node of the Hadoop cluster.

See "[Install the ORCH Package on a Third-Party Hadoop Cluster](#)".

- Oracle Loader for Hadoop to support the OLH driver (optional).

See "[Oracle Loader for Hadoop Setup](#)."

Note:

Do not set `HADOOP_HOME` on the Hadoop cluster. CDH5 does not need it, and it interferes with Oracle R Advanced Analytics for Hadoop. 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
```

Install Sqoop on a Third-Party 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.
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`.

Install Hive on a Third-Party 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:
3. **\$ hive -H** usage: `hive -d,--define <key=value>` Variable substitution to apply to hive commands. e.g. `-d A=B` or `--define A=B . . .`
4. If the command fails or you see warnings in the output, then fix the Hive installation.

Install R on a Hadoop Client

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

<https://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/>

Install R on a Third-Party Hadoop Cluster

You can download Oracle R Distribution 3.0.1 and get the installation instructions from the website at

<http://www.oracle.com/technetwork/database/database-technologies/r/r-distribution/downloads/index.html>

Install the ORCH Package on a Third-Party Hadoop Cluster

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

To install the ORCH package:

1. Log in as `root` to the first node of the cluster.
2. Set the environment variables for the supporting software:

```
$ export JAVA_HOME="/usr/lib/jdk7"  
$ export R_HOME="/usr/lib64/R"  
$ export SQOOP_HOME "/usr/lib/sqoop"
```

3. Unzip the downloaded file:

```
$ unzip orch-<version>.zip  
$ unzip orch-linux-x86_64-<version>.zip  
Archive:  orch-linux-x86_64-<version>.zip  
  creating: ORCH<version>/  
  extracting: ORCH<version>/ORCH_<version>_R_x86_64-unknown-linux-gnu.tar.gz  
  inflating: ORCH<version>/ORCHcore_<version>_R_x86_64-unknown-linux-gnu.tar.gz  
  .  
  .  
  .
```

4. Change to the new directory:

```
$ cd ORCH<version>
```

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

```
R --vanilla CMD INSTALL OREbase_<version>_R_x86_64-unknown-linux-gnu.tar.gz  
R --vanilla CMD INSTALL OREstats_<version>_R_x86_64-unknown-linux-gnu.tar.gz  
R --vanilla CMD INSTALL OREmodels_<version>_R_x86_64-unknown-linux-gnu.tar.gz  
R --vanilla CMD INSTALL OREserver_<version>_R_x86_64-unknown-linux-gnu.tar.gz  
R --vanilla CMD INSTALL ORCHcore_<version>_R_x86_64-unknown-linux-gnu.tar.gz  
R --vanilla CMD INSTALL ORCHstats_<version>_R_x86_64-unknown-linux-gnu.tar.gz  
R --vanilla CMD INSTALL ORCH_<version>_R_x86_64-unknown-linux-gnu.tar.gz
```

6. 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_<version>_R_x86_64-unknown-linux-gnu.tar.gz -d /tmp/  
OREserver_<version>_R_x86_64-unknown-linux-gnu.tar.gz
```

```
$ dcli -C " R --vanilla CMD INSTALL /tmp/OREserver_<version>_R_x86_64-unknown-  
linux-gnu.tar.gz"
```

Install 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 *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  
bdalnode01: package libpng-devel is not installed  
bdalnode02: 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. This example uses `dcli`, which is available only on Oracle Big Data Appliance:

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

4. Change to the `yum` directory:
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
bdalnode01: Loaded plugins: rhnplugin, security
bdalnode01: Repository 'bda' is missing name in configuration, using id
bdalnode01: This system is not registered with ULN.
bdalnode01: ULN support will be disabled.
bdalnode01: http://bdalnode01-master.abcd.com/bda/repodata/repomd.xml:
bdalnode01: [Errno 14] HTTP Error 502: notresolvable
bdalnode01: Trying other mirror.
.
.
.
bdalnode01: Running Transaction
bdalnode01: Installing      : libpng-devel                1/2
bdalnode01: Installing      : libpng-devel                2/2

bdalnode01: Installed:
bdalnode01: libpng-devel.i386 2:1.2.10-17.el5_8  libpng-devel.x86_64
2:1.2.10-17.el5_8

bdalnode01: Complete!
bdalnode02: Loaded plugins: rhnplugin, security
.
.
.
```

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

```
# dcli rpm -qi libpng-devel
bdalnode01: 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.abcd.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.  
.  
.  
.
```

Provide 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
- Oracle R distribution 3.0.1 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

Configure 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 CDH5 or Apache Hadoop 2.2.0 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/hdfs dfs -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.

Install Sqoop on a Hadoop Client

Complete the same procedures on the client system for installing and configuring Sqoop as those provided in ["Install Sqoop on a Third-Party Hadoop Cluster"](#).

Install R on a Hadoop Client

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

<https://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/>

Install the ORCH Package on a Hadoop Client

To install ORCH on your Hadoop client system:

1. Download the ORCH package and unzip it on the client system.
2. Change to the installation directory.
3. Run the client script:

```
# ./install-client.sh
```

Install the Oracle R Enterprise Client Packages (Optional)

To support full access to Oracle Database using R, install the Oracle R Enterprise 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

Oracle Data Integrator

For the instructions to set up and use Oracle Data Integrator refer to [Oracle Fusion Middleware Integrating Big Data with Oracle Data Integrator](#).

Note:

Oracle Data Integrator is not supported for CDH 6.x.

Oracle Datasource for Apache Hadoop Setup

Software Requirements

Oracle Datasource for Apache Hadoop requires the following software:

- A target database system running Oracle Database 12c, 11.2.0.4, or earlier Oracle database releases that can be queried with the Oracle JDBC driver for 12c.
Note that Oracle Database 11.2.0.4 and potentially earlier Oracle Database release may work. However, some of the SPLIT patterns have dependencies on Oracle Database 12c and might not provide accurate splits for parallel hadoop jobs when used to query earlier releases of Oracle Database.
- Cloudera's Distribution including Apache Hadoop version 5 (CDH5), Hortonworks Data Platform (HDP) 2.x, or, Apache Hadoop 2.2.0 to 2.6.0.
- Apache Hive 0.13.0, 0.13.1 or 1.1.0 (in order to query data from Oracle Database tables).

Installing Oracle Datasource for Apache Hadoop

Set `HADOOP_CLASSPATH` to include `$OD4H_HOME/jlib/*` in the Hadoop node where you are running the Hive client. Ensure that this is listed first in `HADOOP_CLASSPATH`.

Ensure that the OD4H jars in `$OD4H_HOME/jlib` are accessible to Hive commands using OD4H. You can do this in one of the following ways:

1. Configure HiveServer 2 and make jars available cluster wide. This will enable Hive client beeline and other tools (such as SQL Developer) to work with OD4H.
2. To isolate configuration to a particular session, add jars manually. You can use this method for situations when you use OD4H and do not want the OD4H jars to interfere with other applications.

Add jar files to Hive CLI

To enable jar files to be present locally to a Hive CLI session, add the jar files via Hive as follows:

```
$hive
```

```
hive> Add jar <jar name>
```

```
hive> Add jar <jar name>
```

Configure HiveServer2

Following are the steps to configure HiveServer2:

1. Login to Cloudera Manager
2. Click on **Hive**.

Figure 1-2 Hive UI

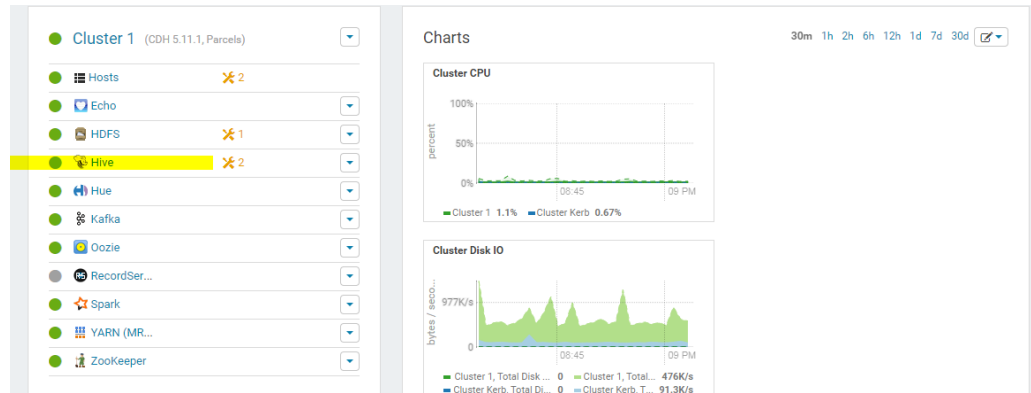
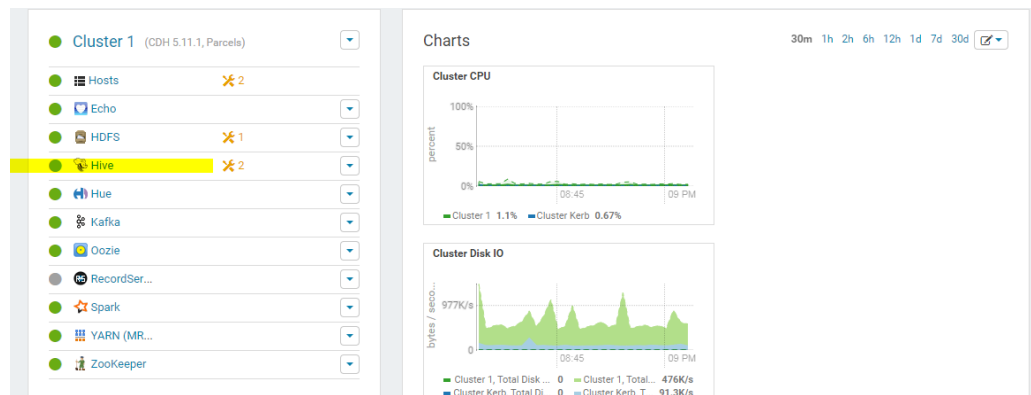
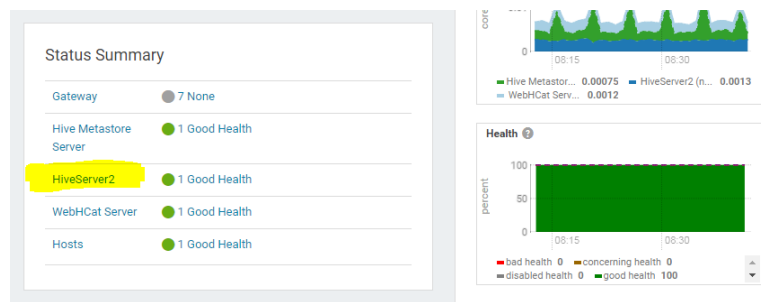


Figure 1-3 Cloudera Manager



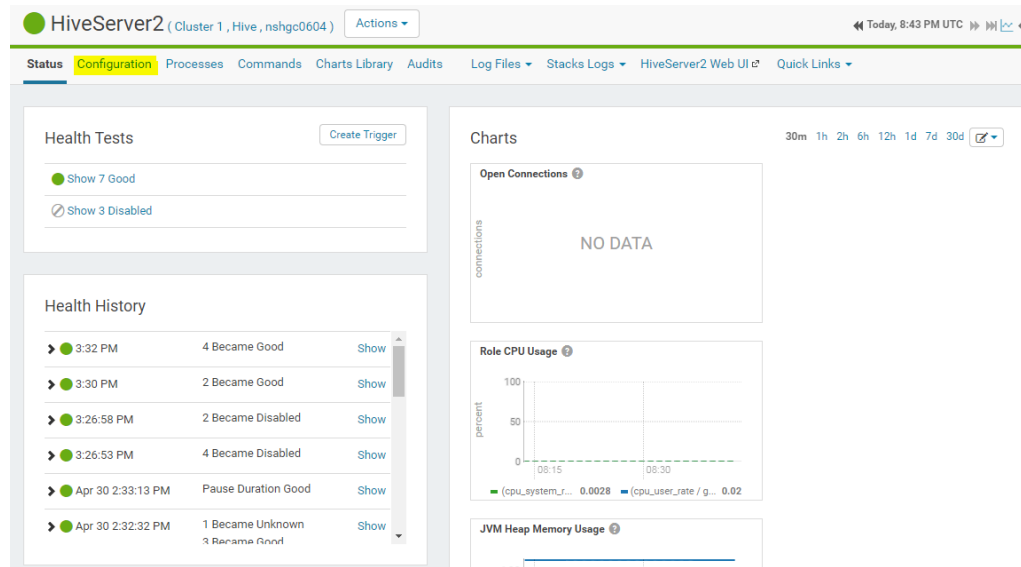
3. Under Status Summary, click **HiveServer2**.

Figure 1-4 Hive Server2



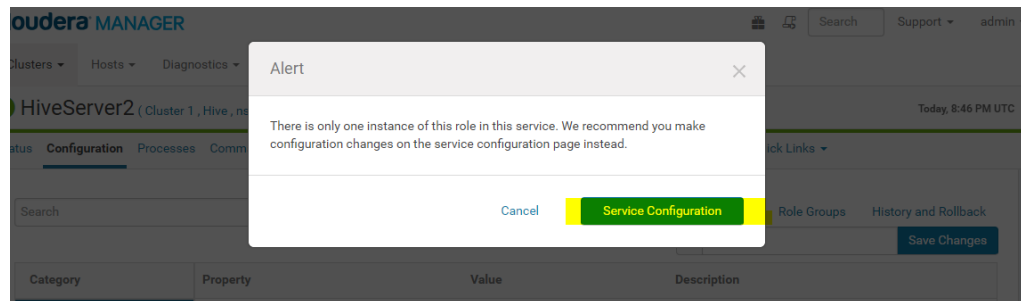
4. Click on **Configuration**.

Figure 1-5 Configuring HiveServer2



5. Click on **Service Configuration**.

Figure 1-6 Service Configuration



6. Type 'AUX' in the search box. Under the property 'Service-Wide/Advanced', add the directory containing jars you want to add in HiveServer2.

Figure 1-7 Adding Jars

The screenshot shows the Hive Configuration page for 'Cluster 1'. The 'Configuration' tab is active, showing a list of properties. A search bar at the top left contains 'AUX'. A yellow warning banner indicates '1 validation warning below'. The table below lists two properties:

Category	Property	Value	Description
Service-Wide / Advanced	Hive Auxiliary JARs Directory	<input type="text" value="/opt/cloudera/parcels/CDH-5.11."/> Reset to empty default value ↗	Directory containing auxiliary JARs used by Hive. This should be a directory location and not a classpath containing one or more JARs. This directory must be created and managed manually on hosts that run the Hive Metastore Server, HiveServer2, or the Hive CLI. The directory location is set in the environment as HIVE_AUX_JARS_PATH and will generally override the hive.aux.jars.path property set in XML files, even if hive.aux.jars.path is set in an advanced configuration snippet.
Service-Wide / Suppressions	Suppress Parameter Validation: Hive Auxiliary JARs Directory	<input type="checkbox"/> default value	Whether to suppress configuration warnings produced by the built-in parameter validation for the Hive Auxiliary JARs Directory parameter.

7. Restart Hive.

Part II

Oracle Database Connectors

This part contains the following chapters:

- [Oracle SQL Connector for Hadoop Distributed File System](#)
- [Oracle Loader for Apache Hadoop](#)

2

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](#)
- [Configure Your System for Oracle SQL Connector for HDFS](#)
- [Use the ExternalTable Command-Line Tool](#)
- [Create External Tables](#)
- [Update External Tables](#)
- [Explore External Tables and Location Files](#)
- [Drop Database Objects Created by Oracle SQL Connector for HDFS](#)
- [More About External Tables Generated by the ExternalTable Tool](#)
- [Configure 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 as delimited text in a new Hive table and then use Oracle SQL Connector for HDFS. Partitioned Hive tables are supported, enabling you to represent a subset of Hive table partitions in Oracle Database, instead of the entire Hive table.

Oracle SQL Connector for HDFS uses external tables and database views 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. Oracle SQL Connector for HDFS creates database views over external tables to support access to partitioned Hive tables. By querying the external tables or views, you can access data stored in HDFS and Hive tables as if that data were stored in tables in an Oracle database.

To create these objects in Oracle Database, 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. The `ExternalTable` command-line tool can be used from either a shell or from SQL Developer.

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 (either all of it or just a selection) if it is queried routinely. Oracle SQL Connector for HDFS functions as a Hadoop client running on the Oracle database and uses the external table preprocessor `hdfs_stream` to access data in HDFS. Oracle Shell for Hadoop Loaders has commands to create an external table and do a bulk load in one step.

 **See Also:**

The following Oracle blog provides information on using Oracle SQL Developer: <https://blogs.oracle.com/bigdataconnectors/move-data-between-apache-hadoop-and-oracle-database-with-sql-developer>

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. Log in to a system where Oracle SQL Connector for HDFS is installed, which can be the Oracle Database system, a node in the Hadoop cluster, or a system set up as a remote client for the Hadoop cluster.
See "[Install and Configure a Hadoop Client on the Oracle Database System.](#)"
2. The first time you use Oracle SQL Connector for HDFS, ensure that the software is configured.
See "[Configure Your System for Oracle SQL Connector for HDFS.](#)" You might also need to edit `hdfs_stream` if your environment is unique. See "[Install Oracle SQL Connector for HDFS](#)".
3. If you are connecting to a secure cluster, then run `kinit` to authenticate yourself.
See "[Use Oracle SQL Connector for HDFS on a Secure Hadoop Cluster.](#)"
4. Create an XML document describing the connections and the data source, unless you are providing these properties in the `ExternalTable` command.
See "[Explore External Tables and Location Files.](#)"
5. Create a shell script containing an `ExternalTable` command.
See "[Use the ExternalTable Command-Line Tool.](#)"
6. Run the shell script.
7. 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.

8. 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.
9. If the data will be queried frequently, then you may want to load it into a database table to improve querying performance. External tables do not have indexes or partitions.

If you want the data to be compressed as it loads into the table, then create the table with the `COMPRESS` option.

10. To delete the Oracle Database objects created by Oracle SQL Connector for HDFS, use the `-drop` command.

See "[Drop Database Objects Created by Oracle SQL Connector for HDFS](#)".

Example 2-1 Accessing HDFS Data Files from Oracle Database

The following illustrates these steps:

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

hadoop jar $OSCH_HOME/jlib/orahdfs.jar \
    oracle.hadoop.exctab.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.nullIfSpecifier</name>
    <value>\N</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.exttab.colMap.TIME_ID.columnType</name>
  <value>TIMESTAMP</value>
</property>
<property>
  <name>oracle.hadoop.exttab.colMap.timestampMask</name>
  <value>YYYY-MM-DD:HH:MI:SS</value>
</property>
<property>
  <name>oracle.hadoop.exttab.colMap.RECOMMENDED.columnType</name>
  <value>NUMBER</value>
</property>
<property>
  <name>oracle.hadoop.exttab.colMap.ACTIVITY_ID.columnType</name>
  <value>NUMBER</value>
</property>
<property>
  <name>oracle.hadoop.exttab.colMap.RATING.columnType</name>
  <value>NUMBER</value>
</property>
<property>
  <name>oracle.hadoop.exttab.colMap.SALES.columnType</name>
  <value>NUMBER</value>
</property>
<property>
  <name>oracle.hadoop.exttab.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 3.4.0 - Production
```

```
Copyright (c) 2011, 2015, 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-20141114064206-5250-1',
  'osch-20141114064206-5250-2',
  'osch-20141114064206-5250-3',
  'osch-20141114064206-5250-4'
)
) PARALLEL REJECT LIMIT UNLIMITED;
```

The following location files were created.

osch-20141114064206-5250-1 contains 1 URI, 12754882 bytes

12754882 hdfs://localhost.localdomain:8020/user/oracle/moviework/data/part-00001

osch-20141114064206-5250-2 contains 1 URI, 438 bytes

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

osch-20141114064206-5250-3 contains 1 URI, 432 bytes

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

osch-20141114064206-5250-4 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 Apr 18 09:24:18 2014

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

Enter password: **password**

Last Successful login time: Thu Apr 17 2014 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.

```
SQL> SELECT movie_id, time_id, recommended, rating FROM movie_facts WHERE rownum < 5;
```

MOVIE_ID	TIME_ID	RECOMMENDED	RATING
205	03-DEC-10 03.14.54.000000000 AM	1	1
77	14-AUG-11 10.46.55.000000000 AM	1	3
116	24-NOV-11 05.43.00.000000000 AM	1	5
141	01-JAN-11 05.17.57.000000000 AM	1	4

Configure Your System for Oracle SQL Connector for HDFS

You can run the `ExternalTable` command-line tool provided with Oracle SQL Connector for HDFS on either the Oracle Database system or the Hadoop cluster:

- For Hive sources, 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, 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 to run the tool. Your system administrator may have set them up for you when creating your account, or may have left that task for you. See "[OS-Level Requirements for OSCH Users](#)".

Setting up the environment variables:

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

```
path/orahdfs-<version>/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-<version>"
```

 **See Also:**

- ["Oracle SQL Connector for Hadoop Distributed File System Setup"](#) 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.

Use Oracle SQL Connector for HDFS with Oracle Big Data Appliance and Oracle Exadata

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. See the [Configuring Oracle Exadata Database Machine for Use with Oracle Big Data Appliance](#) for instructions on configuring Oracle Exadata Database Machine for Use with Oracle Big Data Appliance.

Use 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 ["Create External Tables"](#) 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. .

Related Topics

- [Configure Oracle SQL Connector for HDFS](#)

ExternalTable Command-Line Tool Syntax

This is the full syntax of the `ExternalTable` command-line tool, which is run using the `hadoop` command:

```
hadoop jar OSCH_HOME/jlib/orahdfs.jar \  
oracle.hadoop.exttab.ExternalTable \  

```

```
[-conf config_file]... \  
[-D property=value]... \  
-createTable [--noexecute [--output filename.sql]]  
  | -drop [--noexecute]  
  | -describe  
  | -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-<version>/jlib/orahdfs.jar \  
oracle.hadoop.exttab.ExternalTable \  
.  
.  
.
```

Generic Options and User Commands

-conf *config_file*

Identifies the name of an XML configuration file containing properties needed by the command being executed.

-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.

For partitioned Hive tables, Oracle SQL Connector for HDFS creates external tables, views, and a metadata table. See [Table 2-2](#).

Specify the metadata table name for partitioned Hive tables, or the external table name for all other data sources.

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`.

-drop [--noexecute]

Deletes one or more Oracle Database objects created by Oracle SQL Connector for HDFS to support a particular data source. Specify the metadata table name for partitioned Hive tables, or the external table name for all other data sources. An error occurs if you attempt to drop a table or view that Oracle SQL Connector for HDFS did not create.

Use the `--noexecute` option to list the objects to be deleted.

-describe

Provides information about the Oracle Database objects created by Oracle SQL Connector for HDFS. Use this command instead of `-getDDL` or `-listLocations`.

-publish [--noexecute]

Publishes the data URIs to the location files of an existing external table. Use this command 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 "[Update External Tables](#)."

-listLocations [--details]

Shows the location file content as text. With the `--details` option, this command provides a detailed listing. This command is deprecated in release 3.0. Use `-describe` instead.

-getDDL

Prints the table definition of an existing external table. This command is deprecated in release 3.0. Use `-describe` instead.

Related Topics

- [Configure Oracle SQL Connector for HDFS](#)

**See Also:**

["Syntax Conventions"](#)

Create External Tables

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

Create External Tables with the ExternalTable Tool

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

- [Create External Tables from Data Pump Format Files](#)
- [Create External Tables from Hive Tables](#)
- [Create 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](#)."

To create external tables manually, follow the instructions in "[Create External Tables in SQL](#)."

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]
```

Create 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.

To delete the external tables and location files created by Oracle SQL Connector for HDFS, use the `-drop` command. See "[Drop Database Objects Created by Oracle SQL Connector for HDFS](#)".



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

See "[Configure Oracle SQL Connector for HDFS](#)" for descriptions of the properties used for this data source.

Optional Properties

This property is optional:

- [oracle.hadoop.exctab.logDirectory](#)
- [oracle.hadoop.exctab.createLogFiles](#)
- [oracle.hadoop.exctab.createBadFiles](#)

Defining Properties in XML Files for Data Pump Format Files

The following example 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 "[Create a Configuration File.](#)"

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

The following example 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 clusterwide 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:

```
$ export OSCH_HOME="/opt/oracle/orahdfs-<version>"
$ export HADOOP_CLASSPATH="$OSCH_HOME/jlib/*:$HADOOP_CLASSPATH"
$ 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
```

Create 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. After changes are made to a Hive table, you must use the `ExternalTable` tool to drop the existing external table and create a new one.

To delete the external tables and location files created by Oracle SQL Connector for HDFS, use the `-drop` command. See "[Drop Database Objects Created by Oracle SQL Connector for HDFS](#)".

Hive Table Requirements

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

Oracle SQL Connector for HDFS also supports partitioned Hive tables. In this case Oracle SQL Connector for HDFS creates one or more external tables and database views. See "[Creating External Tables from Partitioned Hive Tables](#)".

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

Data Type Mappings

The following table 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.exctab.hive.columnType.*` properties listed under "[Optional Properties](#)".

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	NUMBER
DECIMAL(<i>p</i> , <i>s</i>)	NUMBER(<i>p</i> , <i>s</i>)
DOUBLE, FLOAT	NUMBER
DATE	DATE with format mask YYYY-MM-DD
TIMESTAMP	TIMESTAMP with format mask YYYY-MM-DD HH24:MI:SS.FF
BOOLEAN	VARCHAR2(5)
CHAR(<i>size</i>)	CHAR(<i>size</i>)
STRING	VARCHAR2(4000)
VARCHAR	VARCHAR2(4000)
VARCHAR(<i>size</i>)	VARCHAR2(<i>size</i>)

Required Properties

These properties are required for Hive table sources:

- `oracle.hadoop.exctab.tableName`
- `oracle.hadoop.exctab.defaultDirectory`
- `oracle.hadoop.exctab.sourceType=hive`
- `oracle.hadoop.exctab.hive.tableName`

- `oracle.hadoop.exctab.hive.databaseName`
- `oracle.hadoop.connection.url`
- `oracle.hadoop.connection.user`

See "[Configure Oracle SQL Connector for HDFS](#)" for descriptions of the properties used for this data source.

Optional Properties

These properties are optional for Hive table sources:

- `oracle.hadoop.exctab.hive.columnType.*`
- `oracle.hadoop.exctab.hive.partitionFilter`
- `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`
- `oracle.hadoop.exctab.createLogFiles`
- `oracle.hadoop.exctab.createBadFiles`
- `oracle.hadoop.exctab.logDirectory`

Defining Properties in XML Files for Hive Tables

The following example 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 "[Create a Configuration File](#)."

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>
<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.partitionFilter</name>
  <value>value</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

This example 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 nonpartitioned 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:

```
$ export OSCH_HOME="/opt/oracle/orahdfs-<version>"
$ export HADOOP_CLASSPATH="$OSCH_HOME/jlib/*:/usr/lib/hive/lib/*:/etc/hive/
conf:$HADOOP_CLASSPATH"

$ 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/my servicename \
-D oracle.hadoop.connection.user=SCOTT \
-createTable
```

 **Note:**

For nonpartitioned Hive tables and other data sources the value for property `oracle.hadoop.exctab.tableName` is the name of the external table.

Creating External Tables from Partitioned Hive Tables

Oracle SQL Connector for HDFS supports partitioned Hive tables, enabling you to query a single partition, a range of partitions, or all partitions. You can represent all Hive partitions or a subset of them in Oracle Database.

 **See Also:**

"[Create External Tables from Hive Tables](#)" for required properties, data type mappings, and other details applicable to all Hive table access using Oracle SQL Connector for HDFS.

Database Objects that Support Access to Partitioned Hive Tables

To support a partitioned Hive table, Oracle SQL Connector for HDFS creates the objects described in the following table.

Table 2-2 Oracle Database Objects for Supporting a Partitioned Hive Table

Database Object	Description	Naming Convention ¹
External Tables	One for each Hive partition	<code>OSCHtable_name_n</code> For example, <code>OSCHDAILY_1</code> and <code>OSCHDAILY_2</code>
Views	One for each external table. Used for querying the Hive data.	<code>table_name_n</code> For example, <code>DAILY_1</code> and <code>DAILY_2</code>

Table 2-2 (Cont.) Oracle Database Objects for Supporting a Partitioned Hive Table

Database Object	Description	Naming Convention ¹
Metadata Table	One for the Hive table. Identifies all external tables and views associated with a particular Hive table. Specify this table when creating, describing, or dropping these database objects.	<i>table_name</i> For example, DAILY

¹ The "_n" suffixed with table name indicates a numeric value.

For example, if a Hive table comprises five partitions, then Oracle SQL Connector for HDFS creates five external tables, five views, and one metadata table in Oracle Database.

To drop the objects described in [Table 2-2](#) and the location files, use the `-drop` command. See "[Drop Database Objects Created by Oracle SQL Connector for HDFS](#)".

 **Note:**

For partitioned Hive tables and other data sources the value for property `oracle.hadoop.exttab.tableName` is the name of the metadata table.

Querying the Metadata Table

The metadata table provides critical information about how to query the Hive table. The following table describes the columns of a metadata table.

Table 2-3 Metadata Table Columns

Column	Description
VIEW_NAME	The Oracle Database view used to access a single Hive table partition. The view contains both Hive table and partition columns.
EXT_TABLE_NAME	An Oracle external table that represents a Hive table partition. The external table contains only the Hive table columns and not the Hive partition columns. To access all the data in a Hive partition, use the corresponding Oracle Database view.
HIVE_TABLE_NAME	The partitioned Hive table being accessed through Oracle Database.
HIVE_DB_NAME	The Hive database where the table resides.
HIVE_PART_FILTER	The Hive partition filter used to select a subset of partitions for access by Oracle Database. A NULL value indicates that all partitions are accessed.

Table 2-3 (Cont.) Metadata Table Columns

Column	Description
Partition Columns	Each column used to partition the Hive table has a separate column in the metadata table. For example, the metadata table has columns for COUNTRY, STATE, and CITY for a Hive table partitioned by a combination of COUNTRY, STATE, and CITY values.

The following SELECT statement queries a metadata table named HIVE_SALES_DATA:

```
SQL> SELECT view_name, ext_table_name, Hive_table_name, \
       hive_db_name, country, city \
       FROM hive_sales_data \
       WHERE state = 'TEXAS';
```

The results of the query identify three views with data from cities in Texas:

VIEW_NAME	EXT_TABLE_NAME	HIVE_TABLE_NAME	HIVE_DB_NAME	COUNTRY	CITY
HIVE_SALES_DATA_1	OSCHHIVE_SALES_DATA_1	hive_sales_data	db_sales	US	AUSTIN
HIVE_SALES_DATA_2	OSCHHIVE_SALES_DATA_2	hive_sales_data	db_sales	US	HOUSTON
HIVE_SALES_DATA_3	OSCHHIVE_SALES_DATA_3	hive_sales_data	db_sales	US	DALLAS

The views include partition column values. Oracle recommends that you use the views while querying a partitioned Hive table, as the external tables do not include the partition column values.

Creating UNION ALL Views for Querying

To facilitate querying, you can create UNION ALL views over the individual partition views. Use the `mkhive_unionall_view.sql` script, which is provided in the `OSCH_HOME/example/sql` directory. To maintain performance, do not create UNION ALL views over more than 50 to 100 views (depending on their size).

To use `mkhive_unionall_view.sql`, use the following syntax:

```
@mkhive_unionall_view[.sql] table schema view predicate
```

MKHIVE_UNIONALL_VIEW Script Parameters

table

The name of the metadata table in Oracle Database that represents a partitioned Hive table. Required.

schema

The owner of the metadata table. Optional; defaults to your schema.

view

The name of the UNION ALL view created by the script. Optional; defaults to `table_ua`.

predicate

A WHERE condition used to select the partitions in the Hive table to include in the UNION ALL view. Optional; defaults to all partitions.

Example 2-6 Union All Views for Partitioned Hive Tables

The following example creates a UNION ALL view named HIVE_SALES_DATA_UA, which accesses all partitions listed in the HIVE_SALES_DATA metadata table:

```
SQL> @mkhive_unionall_view.sql HIVE_SALES_DATA null null null
```

This example creates a UNION ALL view named ALL_SALES, which accesses all partitions listed in the HIVE_SALES_DATA metadata table:

```
SQL> @mkhive_unionall_view.sql HIVE_SALES_DATA null ALL_SALES null
```

The next example creates a UNION ALL view named TEXAS_SALES_DATA, which accesses the rows of all partitions where STATE = 'TEXAS'.

```
SQL> @mkhive_unionallview.sql HIVE_SALES_DATA null TEXAS_SALES_DATA '(STATE =
''TEXAS'')'
```

Error Messages

table name too long, max limit length

Cause: The names generated for the database objects exceed 30 characters.

Action: Specify a name that does not exceed 24 characters in the [oracle.hadoop.exctab.tableName](#) property. Oracle SQL Connector for HDFS generates external table names using the convention `OSCHtable_name_n`. See [Table 2-2](#).

table/view names containing string table_name found in schema schema_name

Cause: An attempt was made to create external tables for a partitioned Hive table, but the data objects already exist.

Action: Use the `hadoop -drop` command to drop the existing tables and views, and then retry the `-createTable` command. If this solution fails, then you might have "dangling" objects. See "[Dropping Dangling Objects](#)".

Dropping Dangling Objects

Always use Oracle SQL Connector for HDFS commands to manage objects created by the connector to support partitioned Hive tables. Dangling objects are caused when you use the SQL `drop table` command to drop a metadata table instead of the `-drop` command. If you are unable to drop the external tables and views for a partitioned Hive table, then they are dangling objects.

Notice the schema and table names in the error message generated when you attempted to drop the objects, and use them in the following procedure.

To drop dangling database objects:

1. Open a SQL session with Oracle Database, and connect as the owner of the dangling objects.
2. Identify the location files of the external table by querying the `ALL_EXTERNAL_LOCATIONS` and `ALL_EXTERNAL_TABLES` data dictionary views:

```
SELECT a.table_name, a.directory_name, a.location \
FROM all_external_locations a, all_external_tables b \
WHERE a.table_name = b.table_name AND a.table_name \
```

```
LIKE 'OSCHtable%' AND a.owner='schema';
```

In the `LIKE` clause of the previous syntax, replace *table* and *schema* with the appropriate values.

In the output, the location file names have an `osch-` prefix, such as `osch-20140408014604-175-1`.

3. Identify the external tables by querying the `ALL_EXTERNAL_TABLES` data dictionary view:

```
SELECT table_name FROM all_external_tables \
WHERE table_name \
LIKE 'OSCHtable%' AND owner=schema;
```

4. Identify the database views by querying the `ALL_VIEWS` data dictionary view:

```
SELECT view_name FROM all_views
WHERE view_name
LIKE 'table%' AND owner='schema';
```

5. Inspect the tables, views, and location files to verify that they are not needed, using commands like the following:

```
DESCRIBE schema.table;
SELECT * FROM schema.table;
```

```
DESCRIBE schema.view;
SELECT * FROM schema.view;
```

6. Delete the location files, tables, and views that are not needed, using commands like the following:

```
EXECUTE utl_file.fremove('directory', 'location_file');
```

```
DROP TABLE schema.table;
DROP VIEW schema.view;
```

Create 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](#)."

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 "[Configure Oracle SQL Connector for HDFS](#)" 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.column_name.nullIfSpecifier`
- `oracle.hadoop.exctab.colMap.timestampMask`
- `oracle.hadoop.exctab.colMap.column_name.timestampMask`
- `oracle.hadoop.exctab.colMap.timestampTZMask`
- `oracle.hadoop.exctab.colMap.column_name.timestampTZMask`
- `oracle.hadoop.exctab.createLogFiles`
- `oracle.hadoop.exctab.createBadFiles`
- `oracle.hadoop.exctab.logDirectory`
- `oracle.hadoop.exctab.nullIfSpecifier`

Defining Properties in XML Files for Delimited Text Files

This example 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 "[Create a Configuration File](#)."

Example 2-7 XML Template with Properties for a Delimited Text File

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

<!-- Required Properties -->

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

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

  <property>
    <name>oracle.hadoop.exttab.columnCount</name>
    <value>value</value>
  </property>
  <property>
    <name>oracle.hadoop.exttab.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.exttab.colMap.TYPE</name>
    <value>value</value>
  </property>
  <property>
    <name>oracle.hadoop.exttab.recordDelimiter</name>
    <value>value</value>
  </property>
  <property>
    <name>oracle.hadoop.exttab.fieldTerminator</name>
    <value>value</value>
  </property>
  <property>
    <name>oracle.hadoop.exttab.initialFieldEncloser</name>
    <value>value</value>
  </property>

```

```

</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

This example creates an external table named SALES_DT_XTAB from delimited text files.

Example 2-8 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:

```
$ export OSCH_HOME="/opt/oracle/orahdfs-<version>"
$ export HADOOP_CLASSPATH="$OSCH_HOME/jlib/*:$HADOOP_CLASSPATH"

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

Create 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 the previous step 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.

Update External Tables

The `-publish` option provides a way to update the location files (that point to HDFS data paths) of existing Oracle external tables based on text, Data Pump, or Hive sources.

The `ExternalTable` command line tool with the `-createTable` option creates the external table and related metadata in Oracle Database. It also populates the external table's location files with the Universal Resource Identifiers (URIs) of the data files in HDFS.

To “publish” updates to the location files of existing external tables, use `ExternalTable` with the `-publish` option. This operation updates the location files of external tables with new URIs of HDFS data paths, and adds external tables and views for new partitions when the source is a partitioned Hive table

Use `-publish` in order to:

- Publish new data into an already existing external table.
`-createTable` takes a snapshot of the HDFS or Hive source at the time of external table creation. However the source may change later. The `-publish` option enables you to update the existing external table from the source.
- Publish new data when the source is a partitioned Hive table.
When new partitions are added to a source that is a partitioned Hive table, the `-publish` option enables you to add new database objects required to access the new partitions. This option detects the new Hive partitions and creates the additional external tables and views for the partitions and updates the metadata table created with the `-createTable` command.
- Populate an external table that you created manually using the `-createTable` command with the `--noexecute` option.

Note:

The `publish` option now fully supports partitioned Hive tables. It is no longer necessary to use `-drop` and `-createTable` as a workaround to update external tables derived from partitioned Hive tables.

See Also:

[Location File Management](#)

ExternalTable Syntax for Publish

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



See Also:

["ExternalTable Command-Line Tool Syntax"](#)

ExternalTable Example for Publish

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

Example 2-9 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.exctab.ExternalTable \  
-D oracle.hadoop.exctab.tableName=SALES_DP_XTAB \  
-D oracle.hadoop.exctab.sourceType=datapump \  
-D oracle.hadoop.exctab.dataPaths=hdfs:/user/scott/data/ \  
-D oracle.hadoop.connection.url=jdbc:oracle:thin:@/myhost:1521/my servicename \  
-D oracle.hadoop.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.

Explore External Tables and Location Files

The `-describe` command is a debugging and diagnostic utility that prints the definition of an existing external table. It also 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.exctab.tableName`
- The JDBC connection properties; see ["Connections using url, user, and password Properties."](#)

ExternalTable Syntax for Describe

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



See Also:

"ExternalTable Command-Line Tool Syntax"

ExternalTable Example for Describe

This example shows the command syntax to describe the external tables and location files associated with SALES_DP_XTAB.

Example 2-10 Exploring External Tables and Location Files

```
$ export HADOOP_CLASSPATH="OSCH_HOME/jlib/*"
$ hadoop jar OSCH_HOME/jlib/orahdfs.jar oracle.hadoop.exctab.ExternalTable \
-D oracle.hadoop.exctab.tableName=SALES_DP_XTAB \
-D oracle.hadoop.connection.url=jdbc:oracle:thin:@//myhost:1521/myservicename \
-D oracle.hadoop.connection.user=scott -describe
```

Drop Database Objects Created by Oracle SQL Connector for HDFS

The `-drop` command deletes the database objects created by Oracle SQL Connector for HDFS. These objects include external tables, location files, and views. If you delete objects manually, problems can arise as described in "[Dropping Dangling Objects](#)".

The `-drop` command only deletes objects created by Oracle SQL Connector for HDFS. Oracle recommends that you always use the `-drop` command to drop objects created by Oracle SQL Connector for HDFS.

These properties are required to use this command:

- `oracle.hadoop.exctab.tableName`. For partitioned Hive tables, this is the name of the metadata table. For other data source types, this is the name of the external table.
- The JDBC connection properties; see "[Connections using url, user, and password Properties](#)."

ExternalTable Syntax for Drop

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

**See Also:**

["ExternalTable Command-Line Tool Syntax"](#)

ExternalTable Example for Drop

This example shows the command syntax to drop the database objects associated with SALES_DP_XTAB.

Example 2-11 Dropping Database Objects

```
$ 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.connection.url=jdbc:oracle:thin:@//myhost:1521/myservicename \
-D oracle.hadoop.connection.user=scott -drop
```

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.

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.exctab.hive.*` property settings.

- `oracle.hadoop.exctab.colMap.columnLength`
- `oracle.hadoop.exctab.colMap.columnType`
- `oracle.hadoop.exctab.colMap.dateMask`
- `oracle.hadoop.exctab.colMap.fieldLength`
- `oracle.hadoop.exctab.colMap.timestampMask`
- `oracle.hadoop.exctab.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.exctab.colMap.column_name.columnLength`
- `oracle.hadoop.exctab.colMap.column_name.columnType`
- `oracle.hadoop.exctab.colMap.column_name.dateMask`
- `oracle.hadoop.exctab.colMap.column_name.fieldLength`
- `oracle.hadoop.exctab.colMap.column_name.timestampMask`
- `oracle.hadoop.exctab.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.exctab.tableName=MOVIE_FACT_EXT_TAB_TXT
oracle.hadoop.exctab.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.exctab.colMap.TIME_ID.columnType=TIMESTAMP
oracle.hadoop.exctab.colMap.RECOMMENDED.columnType=NUMBER
oracle.hadoop.exctab.colMap.ACTIVITY_ID.columnType=NUMBER
oracle.hadoop.exctab.colMap.RATING.columnType=NUMBER
oracle.hadoop.exctab.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                                 V          VARCHAR2(4000)
MOVIE_ID                                V          VARCHAR2(4000)
GENRE_ID                                V          VARCHAR2(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.exttab.colMap.CUST_ID.columnLength=12
oracle.hadoop.exttab.colMap.MOVIE_ID.columnLength=12
oracle.hadoop.exttab.colMap.GENRE_ID.columnLength=12

```

All columns now have custom data types:

```

SQL> DESCRIBE movie_facts_ext
Name                                     Null?    Type
-----
CUST_ID                                 VARCHAR2(12)
MOVIE_ID                                VARCHAR2(12)
GENRE_ID                                 VARCHAR2(12)
TIME_ID                                 TIMESTAMP(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.

Enable 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 determines the number of child processes started by the external table during a table read, which is known as the **degree of parallelism** or **DOP**.

Set Up Degree of Parallelism

Ideally, you can decide to run at a particular degree of parallelism and create a number of location files that are a multiple of the degree of parallelism, as described in the following procedure.

To set up parallel processing for maximum performance:

1. Identify the maximum DOP that your Oracle DBA will permit you to use when running Oracle SQL Connector for HDFS.

When loading a huge amount of data into an Oracle database, you should also work with the DBA to identify a time when the maximum resources are available.

2. Create a number of location files that is a small multiple of the DOP. For example, if the DOP is 8, then you might create 8, 16, 24, or 32 location files.

3. Create a number of HDFS files that are about the same size and a multiple of the number of location files. For example, if you have 32 location files, then you might create 128, 1280, or more HDFS files, depending on the amount of data and the minimum HDFS file size.
4. Set the DOP for the data load, using either the `ALTER SESSION` command or hints in the SQL `SELECT` statement.

This example sets the DOP to 8 using `ALTER SESSION`:

```
ALTER SESSION FORCE PARALLEL DML PARALLEL 8;  
ALTER SESSION FORCE PARALLEL QUERY PARALLEL 8;
```

The next example sets the DOP to 8 using the `PARALLEL` hint:

```
INSERT /*+ parallel(my_db_table,8) */ INTO my_db_table \  
  SELECT /*+ parallel(my_hdfs_external_table,8) */ * \  
  FROM my_hdfs_external_table;
```

An `APPEND` hint in the SQL `INSERT` statement can also help improve performance.

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.exctab.locationFileCount` configuration property. See "[Connections using url, user, and password Properties](#)".

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.

Configure 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.

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](#)".

Create 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>
```

The following example shows a configuration file. See "[Oracle SQL Connector for HDFS Configuration Property Reference](#)" for descriptions of these properties.

Example 2-12 Configuration File for Oracle SQL Connector for HDFS

```
<?xml version="1.0"?>
<configuration>
  <property>
    <name>oracle.hadoop.exttab.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.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](#)
- [Connections using url, user, and password Properties](#)

General Properties

Property	Description
oracle.hadoop.exctab.badFileFormatUsePercentA	<p>Indicates whether an external table bad file name contains '%a'. This is an optional property for the <code>-createTable</code> command.</p> <p>Valid values: TRUE,FALSE</p> <p>Default value: FALSE</p> <p>The default bad file name is <code><external_table_name>_%p.bad</code>. If the value is TRUE, this generates deterministic bad file names of the form <code><external_table_name>_%a.bad</code>. For example: 'mytable_000.bad', and 'mytable_001.bad'.</p> <p>If .bad files exist then diagnostic external tables can be created over the .bad files to review the rejected rows. No .bad files are generated if there are no rejected rows.</p>

See Also:

- [oracle.hadoop.exctab.logFileFormatUsePercentA](#)
- [BADFILE | NOBADFILE in Oracle Database Utilities.](#)

Property	Description
oracle.hadoop.exctab.colMap.columnLength	<p>Specifies the length of all external table columns of type CHAR, VARCHAR2, NCHAR, NVARCHAR2, and RAW. Optional.</p> <p>Default Value: The maximum length allowed by the column type</p> <p>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.</p> <p>Valid values: Integer</p>
oracle.hadoop.exctab.colMap.columnType	<p>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.</p> <p>Default value: VARCHAR2 for text; see Table 2-1 for Hive</p> <p>Valid values: The following Oracle data types are supported:</p> <p>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</p> <p>* RAW binary data in delimited text files must be encoded in hexadecimal.</p>
oracle.hadoop.exctab.colMap.dateMask	<p>Specifies the format mask used in the <i>date_format_spec</i> clause of the external table for all DATE columns. This clause indicates that a character data field contains a date in the specified format.</p> <p>Default value: The default globalization format mask, which is set by the NLS_DATE_FORMAT database parameter</p> <p>Valid values: A datetime format model as described in <i>Oracle Database SQL Language Reference</i>. However, it cannot contain quotation marks.</p>
oracle.hadoop.exctab.colMap.fieldLength	<p>Sets the character buffer length used by the ORACLE_LOADER access driver for all CLOB columns. The value is used in the <i>field_list</i> clause of the external table definition, which identifies the fields in the data file and their data types.</p> <p>Default value: 4000 bytes</p> <p>Valid values: Integer</p>

Property	Description
<code>oracle.hadoop.exctab.colMap.timestampMask</code>	<p>Specifies the format mask used in the <i>date_format_spec</i> clause of the external table for all <code>TIMESTAMP</code> and <code>TIMESTAMP WITH LOCAL TIME ZONE</code> columns. This clause indicates that a character data field contains a timestamp in the specified format.</p> <p>Default value: The default globalization format mask, which is set by the <code>NLS_TIMESTAMP_FORMAT</code> database parameter</p> <p>Valid values: A datetime format model as described in <i>Oracle Database SQL Language Reference</i>. However, it cannot contain quotation marks.</p>
<code>oracle.hadoop.exctab.colMap.timestampTZMask</code>	<p>Specifies the format mask used in the <i>date_format_spec</i> clause of the external table for all <code>TIMESTAMP WITH TIME ZONE</code> columns. This clause indicates that a character data field contains a timestamp in the specified format.</p> <p>Default value: The default globalization format mask, which is set by the <code>NLS_TIMESTAMP_TZ_FORMAT</code> database parameter</p> <p>Valid values: A datetime format model as described in the <i>Oracle Database SQL Language Reference</i>. However, it cannot contain quotation marks.</p>
<code>oracle.hadoop.exctab.colMap.column_name.columnLength</code>	<p>Specifies the length of all external table columns of type <code>CHAR</code>, <code>VARCHAR2</code>, <code>NCHAR</code>, <code>NVARCHAR2</code>, and <code>RAW</code>. Optional.</p> <p>Default Value: The value of <code>oracle.hadoop.exctab.colMap.columnLength</code>; if that property is not set, then the maximum length allowed by the data type</p> <p>Valid values: Integer</p>
<code>oracle.hadoop.exctab.colMap.column_name.columnType</code>	<p>Overrides the data type mapping for <i>column_name</i>. Optional.</p> <p>The <i>column_name</i> is case-sensitive. It must exactly match the name of a column in a Hive table or a column listed in <code>oracle.hadoop.exctab.columnNames</code>.</p> <p>Default value: The value of <code>oracle.hadoop.exctab.colMap.columnType</code>; if that property is not set, then the default data type identified in Table 2-1</p> <p>Valid values: See <code>oracle.hadoop.exctab.colMap.columnType</code></p>
<code>oracle.hadoop.exctab.colMap.column_name.dateMask</code>	<p>Overrides the format mask for <i>column_name</i>. Optional.</p> <p>The <i>column_name</i> is case-sensitive. It must exactly match the name of a column in a Hive table or a column listed in <code>oracle.hadoop.exctab.columnNames</code>.</p> <p>Default value: The value of <code>oracle.hadoop.exctab.colMap.dateMask</code>.</p> <p>Valid values: A datetime format model as described in the <i>Oracle Database SQL Language Reference</i>. However, it cannot contain quotation marks.</p>

Property	Description
<code>oracle.hadoop.exctab.colMap.column_name.fieldLength</code>	<p>Overrides the character buffer length used by the <code>ORACLE_LOADER</code> access driver for <code>column_name</code>. This property is especially useful for <code>CLOB</code> and extended data type columns. Optional.</p> <p>The <code>column_name</code> 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.</p> <p>Default value: Oracle SQL Connector for HDFS sets the default field lengths as shown in the following table.</p>

Table 2-4 Field Length Calculations

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

Valid values: Integer

`oracle.hadoop.exctab.colMap.column_name.nullIfSpecifier` This property is applied to a column identified by `column_name` in an external table. Optional. Overrides the property `oracle.hadoop.exctab.nullIfSpecifier`.

Type: string

Valid values: same as for the property `oracle.hadoop.exctab.nullIfSpecifier`.

Default values: none.

This property applies only to Delimited Text sources.

`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 *Oracle Database SQL Language Reference*. However, it cannot contain quotation marks.

`oracle.hadoop.exctab.columnCount` Specifies 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.

Property	Description
oracle.hadoop.exctab.columnNames	<p>Specifies 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 <code>oracle.hadoop.exctab.columnCount</code> property.</p> <p>The column names are read as SQL identifiers: unquoted values are capitalized, and double-quoted values stay exactly as entered.</p> <p>The <code>-createTable</code> command uses this property when <code>oracle.hadoop.exctab.sourceType=text</code>.</p> <p>You must set either this property or oracle.hadoop.exctab.columnCount when creating an external table from delimited text files.</p>

oracle.hadoop.exctab.dataCompressionCodec

 **Notice:**

This property is deprecated. OSCH discovers the compression code of each file at runtime. The dataset can contain both compressed and uncompressed files and can also contain files compressed with different codecs.

Specifies 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 any 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

Property	Description
oracle.hadoop.exctab.data Paths	<p>Specifies 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 the following table. This property is required for the <code>-createTable</code> and <code>-publish</code> commands using Data Pump or delimited text files. The property is ignored for Hive data sources.</p> <p>For example, to select all files in <code>/data/s2/</code>, and only the CSV files in <code>/data/s7/</code>, <code>/data/s8/</code>, and <code>/data/s9/</code>, enter this expression:</p> <pre>/data/s2/,/data/s[7-9]/*.csv</pre> <p>The external table accesses the data contained in all listed files and all files in listed directories. These files compose a single data set.</p> <p>The data set can contain compressed files or uncompressed files, but not both.</p>

Table 2-5 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.data PathFilter Specifies 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. In Oracle RAC, this directory must be on a shared directory accessible by all Oracle instances.

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.

Property	Description
oracle.hadoop.exctab.fieldTerminator	<p>Specifies the field terminator for an external table when <code>oracle.hadoop.exctab.sourceType=text</code>. Optional.</p> <p>Default value: , (comma)</p> <p>Valid values: A string in one of the following formats:</p> <ul style="list-style-type: none"> One or more regular printable characters; it cannot start with <code>\u</code>. For example, <code>\t</code> represents a tab. One or more encoded characters in the format <code>\uHHHH</code>, where <code>HHHH</code> is a big-endian hexadecimal representation of the character in UTF-16. For example, <code>\u0009</code> represents a tab. The hexadecimal digits are case insensitive. <p>Do not mix the two formats.</p>
oracle.hadoop.exctab.hive.columnType.*	<p>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.</p> <p>When Hive <code>TIMESTAMP</code> column is mapped to an Oracle <code>TIMESTAMP</code> column, then the format mask is <code>YYYY-MM-DD H24:MI:SS.FF</code>. When a Hive <code>STRING</code> column is mapped to an Oracle <code>TIMESTAMP</code> 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.</p> <p>Default values: The following table lists the Hive column type properties and their default values.</p> <p>Valid values: See the valid values for oracle.hadoop.exctab.colMap.columnType.</p>

Table 2-6 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	<p>Specifies the name of a Hive database that contains the input data table.</p> <p>The <code>-createTable</code> command requires this property when <code>oracle.hadoop.exctab.sourceType=hive</code>.</p>
---	--

Property	Description
oracle.hadoop.exctab.hive.partitionFilter	<p data-bbox="552 262 1464 325">Specifies a valid HiveQL expression that is used to filter the source Hive table partitions. This property is ignored if the table is not partitioned.</p> <p data-bbox="552 325 1464 367">Type: String</p> <p data-bbox="552 367 1464 409">Default value: None. All partitions of the Hive table are mapped to external tables.</p> <p data-bbox="552 409 1464 451">Valid values: A valid HiveQL expression.</p> <p data-bbox="552 451 1464 556">Description: Specifies a valid HiveQL expression that is used to filter the source Hive table partitions. This property is ignored if the Hive table is not partitioned. Including other columns does not raise an error, but unintended consequences can result. Oracle recommends that you exclude other columns.</p> <p data-bbox="552 556 1464 598">The expression must conform to the following restrictions:</p> <ul data-bbox="552 598 1464 913" style="list-style-type: none"> • Selects partitions and not individual records inside the partitions. • Does not include columns that are not used to partition the table, because they might cause unintended consequences. • Does not include subqueries. • Does not include user-defined functions (UDFs). Built-in functions are supported. • Does not support Hive variable name spaces (such as <code>env:</code>, <code>system:</code>, <code>hiveconf:</code>, and <code>hivevar:</code>) because Hive variable expansion is disabled when OSCH processes this string. Expand all variables in Hive CLI before setting this property. For example: <pre data-bbox="552 945 1464 1050">CREATE VIEW view_name AS SELECT * from database.table_name WHERE expression; DESCRIBE FORMATTED view_name;</pre> <p data-bbox="552 1081 1464 1144">The View Original Text field contains the query with all variables expanded. Copy the where clause, starting after where.</p> <p data-bbox="552 1144 1464 1312">Since all variable expansions are resolved at the Hadoop level, define any Hadoop variables used in the expression using generic options (<code>-D</code> and <code>-conf</code>). Use the Hive CLI to test the expression and ensure that it returns the expected results. The following examples assume a source table defined with this command:</p> <pre data-bbox="552 1344 1464 1449">CREATE TABLE t(c string) PARTITIONED BY (p1 string, p2 int, p3 boolean, p4 string, p5 timestamp);</pre> <p data-bbox="552 1480 1464 1522">Example 1: Nested Expressions</p> <pre data-bbox="552 1554 1464 1596">p1 like 'abc%' or (p5 >= '2010-06-20' and p5 <= '2010-07-03')</pre> <p data-bbox="552 1627 1464 1669">Example 2: Built-in Functions</p> <pre data-bbox="552 1701 1464 1743">year(p5) = 2014</pre> <p data-bbox="552 1774 1464 1816">Example 3: Bad Usage: Columns That Are Not Used to Partition the Table</p> <p data-bbox="552 1816 1464 1873">These examples show that using <code>c</code>, a column that is not used to partition the table, is unnecessary and can cause unexpected results.</p>

Property	Description
	<p>This example is equivalent to <code>p2 > 35:</code></p> <pre>p2 > 35 and c like 'abc%'</pre> <p>This example loads all partitions. All partitions could contain <code>c like 'abc%',</code> so partitions are filtered out:</p> <pre>p2 > 35 or c like 'abc%'</pre>
oracle.hadoop.exctab.hive.refreshTables	<p>Only applies when the source is a Hive partitioned table, is ignored otherwise. This property specifies whether the <code>-publish</code> operation should refresh HDFS data paths in existing external tables when adding new external tables and views for new Hive partitions in the source.</p> <p>Set this property to <code>TRUE</code> to enable refresh. Note that enabling such a refresh can slow down the <code>-publish</code> operation. If the existing Hive partitions in the source table have not changed, set the property to <code>FALSE</code>.</p> <p>Default value: <code>FALSE</code></p>
oracle.hadoop.exctab.hive.tableName	<p>Specifies the name of an existing Hive table.</p> <p>The <code>-createTable</code> command requires this property when <code>oracle.hadoop.exctab.sourceType=hive</code>.</p>
oracle.hadoop.exctab.hive.deleteObsoleteTables	<p>Specifies whether the <code>-publish</code> operation should drop the views and external tables that do not map to any partition in the partitioned Hive table. The property only applies when the source is a Hive-partitioned table and is otherwise ignored. This property is also ignored if the original <code>-createTable</code> operation for the Hive partitioned source table included the <code>oracle.hadoop.exctab.hive.partitionFilter</code> property</p> <p>Set this property to <code>TRUE</code> to enable dropping obsolete objects.</p> <p>Default value: <code>FALSE</code></p>
oracle.hadoop.exctab.initialFieldEncloser	<p>Specifies the initial field encloser for an external table created from delimited text files. Optional.</p> <p>Default value: <code>null</code>; no enclosers are specified for the external table definition.</p> <p>The <code>-createTable</code> command uses this property when <code>oracle.hadoop.exctab.sourceType=text</code>.</p> <p>Valid values: A string in one of the following formats:</p> <ul style="list-style-type: none"> • One or more regular printable characters; it cannot start with <code>\u</code>. • One or more encoded characters in the format <code>\uHHHH</code>, where <code>HHHH</code> is a big-endian hexadecimal representation of the character in UTF-16. The hexadecimal digits are case insensitive. <p>Do not mix the two formats.</p>

Property	Description
oracle.hadoop.exctab.locationFileCount	<p>Specifies the desired number of location files for the external table. Applicable only to non-Data-Pump files.</p> <p>Default value: 4</p> <p>This property is ignored if the data files are in Data Pump format. Otherwise, the number of location files is the lesser of:</p> <ul style="list-style-type: none"> The number of data files The value of this property <p>At least one location file is created.</p> <p>See "Enable Parallel Processing" for more information about the number of location files.</p>
oracle.hadoop.exctab.logDirectory	<p>Specifies a database directory where log files and bad 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 <code>_%p.log</code>.</p> <p>This is an optional property for the <code>-createTable</code> command.</p> <p>These are the default file name extensions:</p> <ul style="list-style-type: none"> Log files: <code>log</code> Bad files: <code>bad</code> <p>Valid values: An existing Oracle directory object name.</p> <p>Unquoted names are changed to uppercase. Quoted names are not changed. The following table provides examples of how values are transformed.</p>

Table 2-7 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>
<code>"my_log_dir": "sales_tab_%p.log"</code>	<code>my_log_dir/sales_tab_%p.log</code>


oracle.hadoop.exctab.nullIfSpecifier	<p>Specifies the NULLIF clause of the external table definition. Optional.</p> <p>This property is applied to all the columns in an external table.</p> <p>Type: string</p> <p>Valid values: a string in following formats:</p> <ul style="list-style-type: none"> One or more regular, printable characters, for example: <code>\N</code> One or more encoded characters in the format <code>\uHHHH</code>, where HHHH is a big-endian hexadecimal representation of the character in UTF-16. For example: <code>\u000A</code> represents a newline. The hexadecimal digits are case insensitive. <p>Default values: none.</p> <p>This property applies only to Delimited Text sources.</p>
---	---



See Also:

[Example 2-1](#) . This example shows how to use `nullIfSpecifier` when accessing HDFS data files From Oracle Database.

Property	Description
oracle.hadoop.exctab.preprocessorDirectory	<p>Specifies the database directory for the preprocessor. The file-system directory must contain the <code>hdfs_stream</code> script.</p> <p>Default value: <code>OSCH_BIN_PATH</code></p> <p>The preprocessor directory is used in the <code>PREPROCESSOR</code> clause of the external table.</p>
oracle.hadoop.exctab.preprocessorScript	<p>Specifies the name of the preprocessor script for the external table.</p> <p>Default value: <code>hdfs_stream</code></p> <p>The preprocessor script name is used in the <code>PREPROCESSOR</code> clause of the external table. This property is required only for Oracle Database running on Microsoft Windows platforms and is optional for all other Oracle Database platforms. On Microsoft Windows, the value must be set to <code>hdfs_stream.cmd</code>.</p>
oracle.hadoop.exctab.recordDelimiter	<p>Specifies the record delimiter for an external table created from delimited text files. Optional.</p> <p>Default value: <code>\n</code></p> <p>The <code>-createTable</code> command uses this property when oracle.hadoop.exctab.sourceType=text.</p> <p>Valid values: A string in one of the following formats:</p> <ul style="list-style-type: none"> • One or more regular printable characters; it cannot start with <code>\u</code>. • One or more encoded characters in the format <code>\uHHHH</code>, where <code>HHHH</code> is a big-endian hexadecimal representation of the character in UTF-16. The hexadecimal digits are case insensitive. <p>Do not mix the two formats.</p>
oracle.hadoop.exctab.sourceType	<p>Specifies the type of source files. The <code>-createTable</code> and <code>-publish</code> operations require the value of this property.</p> <p>Default value: <code>text</code></p> <p>Valid values: <code>datapump</code>, <code>hive</code>, or <code>text</code></p>
oracle.hadoop.exctab.stringSizes	<p>Indicates whether the lengths specified for character strings are bytes or characters. This value is used in the <code>STRING SIZES ARE IN</code> clause of the external table. Use characters when loading multibyte character sets. See <i>Oracle Database Utilities</i>.</p> <p>Default value: <code>BYTES</code></p> <p>Valid values: <code>BYTES</code> or <code>CHARACTERS</code></p>
oracle.hadoop.exctab.createLogFiles	<p>Specifies whether the log files should be created when the external tables are queried. Oracle recommends enabling log file creation during development and disabling log file creation during production for best performance.</p> <p>Default value: <code>TRUE</code></p> <p>Log files are created by default. To stop creating log files you must drop the table, set this property to <code>FALSE</code>, and then recreate the table. Use the <code>-drop</code> and <code>-createTable</code> commands to drop and recreate the table.</p>
oracle.hadoop.exctab.printVerbose	<p>Specifies whether to print verbose reports on the console during a <code>-publish</code> operation. Set the value to <code>TRUE</code> to see verbose reports on the console. This property should only be used for debugging.</p> <p>Default value: <code>FALSE</code></p>

Property	Description
oracle.hadoop.exctab.createBadFiles	<p>Specifies whether bad files should be created when the external tables are queried. Bad files contain information on rows with bad data. Bad files are created only when there is bad data. Oracle recommends creating bad files.</p> <p>Default value: TRUE</p> <p>Bad files are created by default. To stop creating bad files you must drop the table, set this property to <code>FALSE</code>, and then recreate the table. Use the <code>-drop</code> and <code>-createTable</code> commands to drop and recreate the table.</p> <p>This property applies only to Hive and Delimited Text sources.</p>
oracle.hadoop.exctab.logFileFormatUsePercentA	<p>Indicates whether an external table log file name contains '%a'. This is an optional property for the <code>-createTable</code> command.</p> <p>Valid values: TRUE,FALSE</p> <p>Default value: FALSE</p> <p>The default log file name is <code><external_table_name>_%p.log</code>. If the value is <code>TRUE</code>, this generates deterministic log file names of the form <code><external_table_name>_%a.log</code>. For example: 'mytable_000.log', and 'mytable_001.log'.</p> <p>Diagnostic external tables can be created over the <code>.log</code> files in order to determine whether the load was successful or to review the reason for the rejected rows.</p>
<div style="border: 1px solid #0070c0; padding: 10px; background-color: #e6f2ff;">  See Also: <ul style="list-style-type: none"> • oracle.hadoop.exctab.badFileFormatUsePercentA • LOGFILE NOLOGFILE in <i>Oracle Database Utilities</i>. </div>	
oracle.hadoop.exctab.tableName	<p>Specifies the metadata table for partitioned Hive tables or schema-qualified name of the external table for all other data sources, in this format:</p> <p><i>schemaName.tableName</i></p> <p>If you omit <i>schemaName</i>, then the schema name defaults to the connection user name.</p> <p>Default value: none</p> <p>Required property for all operations.</p>
oracle.hadoop.exctab.trailingFieldEncloser	<p>Specifies the trailing field encloser for an external table created from delimited text files. Optional.</p> <p>Default value: null; defaults to the value of oracle.hadoop.exctab.initialFieldEncloser</p> <p>The <code>-createTable</code> command uses this property when oracle.hadoop.exctab.sourceType=text.</p> <p>Valid values: A string in one of the following formats:</p> <ul style="list-style-type: none"> • One or more regular printable characters; it cannot start with <code>\u</code>. • One or more encoded characters in the format <code>\uHHHH</code>, where <i>HHHH</i> is a big-endian hexadecimal representation of the character in UTF-16. The hexadecimal digits are case insensitive. <p>Do not mix the two formats.</p>

Connections using url, user, and password Properties

The url, user, and password properties provide a distinct connection method. Do not mix these properties with those required for a connection using an Oracle Wallet.

Property	Description
<code>oracle.hadoop.connection.url</code>	<p>Specifies the database connection string in the thin-style service name format:</p> <pre>jdbc:oracle:thin:@//host_name:port/service_name</pre> <p>If you are unsure of the service name, then enter this SQL command as a privileged user:</p> <pre>SQL> show parameter service</pre> <p>This property takes precedence over all other connection properties.</p> <p>Default value: Not defined Valid values: A string</p>
<code>oracle.hadoop.connection.user</code>	<p>Specifies an Oracle database log-in name. The <code>externalTable</code> tool prompts for a password if the <code>oracle.hadoop.connection.password</code> is not specified .</p> <p>Default value: Not defined Valid values: A string</p>
<code>oracle.hadoop.connection.password</code>	<p>Password for the Oracle Database user. Oracle recommends that you do not use this property to store a clear text password outside of non-sensitive test or demo environments. You can force a password prompt/response by excluding the password property from the connection. In that case, the <code>externalTable</code> tool prompts for the password. If you require a connection with no prompt/response, use the Oracle Wallet connection method described in the next section instead.</p> <p>Default value: Not defined. Valid values: A string</p>

Connections Using Oracle Wallet

When using Oracle Wallet as an external password store, set the properties shown in the following table.

Table 2-8 Properties Required for Connections Using Oracle Wallet

Property	Description
<code>oracle.hadoop.connection.tnsEntryName</code>	<p>Specifies a TNS entry name defined in the <code>tnsnames.ora</code> file.</p> <p>This property is used with the <code>oracle.hadoop.connection.tns_admin</code> property.</p> <p>Default value: Not defined Valid values: A string</p>

Table 2-8 (Cont.) Properties Required for Connections Using Oracle Wallet

Property	Description
<code>oracle.hadoop.connection.tns_admin</code>	<p>Specifies the directory that contains the <code>tnsnames.ora</code> 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 <code>oracle.net.tns_admin</code> property. When both are set, this property takes precedence over <code>oracle.net.tns_admin</code>.</p> <p>This property must be set when using Oracle Wallet as an external password store.</p> <p>Default value: The value of the Java <code>oracle.net.tns_admin</code> system property</p> <p>Valid values: A string</p>
<code>oracle.hadoop.connection.wallet_location</code>	<p>Specifies a file path to an Oracle Wallet directory where the connection credential is stored.</p> <p>Default value: Not defined</p> <p>Valid values: A string</p>

 **Tip:**

Connections using Oracle Wallet can accommodate many TNS entries and are therefore recommended over those using the user, password and url properties which are restricted to a single machine/port/servicename combination.

For a simple step-by-step demonstration, see the posting [Using Oracle SQL Connector for HDFS with Oracle Wallet](#) in the [Connecting Hadoop With Oracle](#) blog.

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.

See [My Oracle Support Document 2111850.1](#) for additional details and examples for improving performance.

3

Oracle Loader for Apache Hadoop

This chapter explains how to use Oracle Loader for Apache Hadoop (Oracle Loader for Hadoop) to load 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](#)
- [Get Started With Oracle Loader for Hadoop](#)
- [Use Oracle Loader for Hadoop With the Hadoop Command Line Utility](#)
 - [Create the Target Table](#)
 - [Create a Job Configuration File](#)
 - [Establish Secure Connections to Oracle Database Using SSL and Oracle Wallet](#)
 - [About Input Formats](#)
 - [Mapping Input Fields to Target Table Columns](#)
 - [About Output Formats](#)
 - [Run a Loader Job](#)
 - [Handling Rejected Records](#)
 - [Balancing Loads When Loading Data into Partitioned Tables](#)
 - [Optimize Communications Between Oracle Engineered Systems](#)
- [Oracle Loader for Hadoop Configuration Property Reference](#)

What Is Oracle Loader for Hadoop?

Oracle Loader for Hadoop is an efficient and high-performance loader for fast loading of data from a Hadoop cluster into a table in an Oracle database. It pre-partitions 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. Off-loading 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.

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.

Interfaces to Oracle Loader For Hadoop

There are three ways to use Oracle Loader for Hadoop:

- **Oracle Shell for Hadoop Loaders (OHS)**
OHS is the preferred way to use Oracle Loader for Hadoop. It includes a CLI (whose simple command syntax can also be scripted) for moving data between Hadoop and Oracle Database using various resources, including OLH.
- **Oracle SQL Developer**
Oracle SQL Developer is a free graphical IDE that includes integration with Oracle Database and Oracle Big Data Connectors (among many other products). It provides wizards to help you access and use Oracle Big Data Connectors.
- **The `hadoop` command-line utility**
On the command line, you provide the job details with the configuration settings. You typically provide these settings in a job configuration file. You can consider this option, if you need to use a feature not supported by your preferred UI (OHS or SQL Developer). For most use cases, OHS or SQL Developer is sufficient.

See Also:

[Use Oracle SQL Developer With Oracle Big Data Connectors](#) in this guide provides instructions for downloading Oracle SQL Developer and configuring it for use with Oracle Big Data Connectors.

Get Started With Oracle Loader for Hadoop

These instructions show how to use Oracle Loader for Hadoop through OHS.

Before You Start

This is what you need to know before using OLH to load an Oracle Database table with data stored in Hadoop:

- The password of the database schema you are connecting to (which is implied by the database connection URL) or any other relevant database credentials information.
- The name of the Oracle Database table.
- The source of the data stored in Hadoop (either a path to an HDFS directory or the name of a Hive table).
- The preferred method for loading. Choose either JDBC or direct path. Direct path load is faster, but requires a partitioned target table. JDBC does not.

About Resources

In OHSB, the term *resources* refers to the interfaces that OHSB presents for defining the data source, destination, and command language. Four types of resources are available:

- Hadoop resources – for executing HDFS commands and use HDFS as a source or destination.
- Hive resources – for executing Hive commands and specifying Hive as a source or destination.
- JDBC resources – for making JDBC connections to a database.
- SQL resources – for executing SQL commands in a database schema.

Two resources are created upon OHSB startup:

- `hive0` – enables access to the default Hive database.
- `hadoop0` – enables access to HDFS.

Within a session, you can create SQL and JDBC resources as well as additional Hive resources (for example, to connect to other Hive databases). Assign a meaningful name to a resource, as in the example below where the names `sql10` and `ora_mydatabase` clearly indicate the type of resource.

Where resources are invoked in the commands below, the percent sign (%) prefix identifies a resource name.

Loading an Oracle Database Table

1. Start an OHSB session.

```
$ ohsh  
ohsh>
```

2. Create the following resources:

- **SQL resource:**

```
ohsh> create sqlplus resource sql0 connectid="<database connection  
url>"
```

At the prompt, enter the database password.

- **SQL resource when using JDBC SSL:**

You can use a JDBC SSL connection to connect to Oracle Database (for example, to connect to an Oracle Autonomous Database). See [Using JDBC SSL](#) to download the client credentials and identify the TNS entry in the `tnsnames.ora` file.

For example:

```
ohsh> create sqlplus resource sql_ssl connectid="inst1_ssl"
```

`inst1_ssl` is the TNS entry for the JDBC SSL connection.

At the prompt, enter the database password.

– **SQL resource when using Java KeyStore:**

You can use Java KeyStore to store your password and you won't be prompted for username and password. Add this to the scripts you develop to load data. See [Using Secure External Java KeyStore and Hadoop credential command](#) to create a Java KeyStore.

For example:

```
ohsh> create sqlplus resource sql_cs user=oracle
passwordalias=oracle_passwd
provider="jceks://file/home/oracle/passwd.jceks"
connectid="inst1"
```

• **JDBC resource:**

A JDBC resource name that indicates the target schema is recommended.

```
ohsh> create jdbc resource ora_mydatabase
connectid="<database connection url>"
```

At the prompt, enter the database password.

– **JDBC resource when using JDBC SSL:**

You can use a JDBC SSL connection to connect to Oracle Database (for example, to connect to an Oracle Autonomous Database).

For example:

```
ohsh> create jdbc resource ora_mydatabase_ssl
connectiondir=/home/oracle/ssl_client_wallet
connectid="inst1_ssl"
```

inst1_ssl is the TNS entry for the JDBC SSL connection.

At the prompt, enter the database password.

See [Using JDBC SSL](#) to download the client credentials and identify the TNS entry in the `tnsnames.ora` file.

– **JDBC resource when using Java KeyStore:**

You can use Java KeyStore to store your password and you won't be prompted for username and password. Add this to the scripts you develop to load data. See [Using Secure External Java KeyStore and Hadoop credential command](#) to create a Java KeyStore.

For example:

```
ohsh> create jdbc resource ora_mydatabase_cs
connectiondir=oracle
passwordalias=oracle_passwd
provider="jceks://file/home/oracle/passwd.jceks"
connectid="inst1"
```

• **Additional Hive resources (if required):**

The default Hive resource `hive0` connects to the default database in Hive. If you want to connect to another Hive database, create another resource:

```
ohsh> create hive resource hive_mydatabase
connectionurl="jdbc:hive2:///<Hive database name>
```

3. Use the `load` command to load files from HDFS into a target table in the Oracle database.

The following command loads data from a delimited text file in HDFS `<HDFS path>` into the target table in Oracle Database using the direct path option.

```
ohsh> load oracle table ora_mydatabase:<target table in the Oracle
database> from path hadoop0:/user/<HDFS path> using directpath
```

Note:

The default direct path method is the fastest way to load a table. However, it requires partitioned target table. Direct path is always recommended for use with partition tables. Use the JDBC option to load into a non-partitioned target table.

If the command does not explicitly state the load method, then OSHH automatically uses the appropriate method. If the target Oracle table is partitioned, then by default, OSHH uses direct path (i.e. Oracle OCI). If the Oracle table is not partitioned, it uses JDBC.

4. After loading, check the number of rows.

You can do this conveniently from the OSHH command line:

```
ohsh> %sql0 select count(*) from <target table in Oracle Database>
```

Loading a Hive Table Into an Oracle Database Table

You can use OSHH to load a Hive table into a target table in an Oracle database. The command below shows how to do this using the direct path method.

```
ohsh> load oracle table ora_mydatabase:<target table in Oracle Database>
from hive table hive0:<Hive table name>
```

Note that if the target table is partitioned, then OSHH uses direct path automatically. You do not need to enter `using directpath` explicitly in the command.

If the target table is non-partitioned, then specify the JDBC method instead:

```
ohsh> load oracle table ora_mydatabase:<target table in Oracle Database>
from hive table hive0:<Hive table name> using jdbc
```

 **Note:**

The `load` command assumes that the column names in the Hive table and in the Oracle Database table are identically matched. If they do not match, then use OHS `loadermap`.

Using OHS Loadermaps

The simple load examples in this section assume the following:

- Where we load data from a text file in Hadoop into an Oracle Database table, the declared order of columns in the target table maps correctly to the physical ordering of the delimited text fields in the file.
- Where we load Hive tables in to Oracle Database tables, the Hive and Oracle Database column names are identically matched.

However, in less straightforward cases where the column names (or the order of column names and delimited text fields) do not match, use the OHS `loadermap` construct to correct these mismatches.

You can also use a loadermap to specify a subset of target columns to load into table or in the case of a load from a text file, specify the format of a field in the load.

Loadermaps are not covered in this introduction.

Performance Tuning Oracle Loader for Hadoop in OHS

Aside from network bandwidth, two factors can have significant impact on Oracle Loader for Hadoop performance. You can tune both in OHS.

- Degree of parallelism

The degree of parallelism affects performance when Oracle Loader for Hadoop runs in Hadoop. For the default method (direct path), parallelism is determined by the number of reducer tasks. The higher the number of reducer tasks, the faster the performance. The default value is 4. To set the number of tasks:

```
ohsh> set reducetasks 18
```

For the JDBC option, parallelism is determined by the number of map tasks and the optimal number is determined automatically. However, remember that if the target table is partitioned, direct path is faster than JDBC.

- Load balancing

Performance is best when the load is balanced evenly across reduce tasks. The load is detected by sampling. Sampling is always enabled by default for loads using the JDBC and the default copy method.

Debugging in OHS

Several OHS settings control the availability of debugging information:

- `outputlevel`

The `outputlevel` is set to `minimal` by default. Set it to `verbose` in order to return a stack trace when a command fails:

```
ohsh> set outputlevel verbose
```

- `logbadrecords`

```
ohsh> set logbadrecords true
```

This is set to `true` by default.

These log files are informative for debugging:

- Oracle Loader for Hadoop log files.

```
/user/<username>/smartloader/jobhistory/oracle/<target table  
schema>/<target table name>/<OHS job ID>/_olh
```

- Log files generated by the map and reduce tasks.

Other OHS Properties That are Useful for Oracle Loader for Hadoop

You can set these properties on the OHS command line or in a script.

- `dateformat`

```
ohsh> set dateformat "yyyy-MM-dd HH:mm:ss"
```

The syntax for this command is dictated by the Java date format.

- `rejectlimit`

The number of rows that can be rejected before the load of a delimited text file fails.

- `fieldterminator`

The field terminator in loads of delimited text files.

- `hadooptnsadmin`

Location of an Oracle TNS admin directory in the Hadoop cluster

- `hadoopwalletlocation`

Location of the Oracle Wallet directory in the Hadoop cluster.

Additional Information

Using the `exttab` (External Table) Method to Load Data

A third option to load data from Hadoop into Oracle Database is `exttab`.



Note:

The `exttab` option is available in on-premises deployments of OHS only. It is not available in Oracle cloud services.

In the `exttab`, data is loaded via external tables. OHS creates the external table using Oracle SQL Connector for HDFS, and then uses a `Create table as Select` statement to load the data into the target table:

```
ohsh> load oracle table ora_mydatabase:<target table in Oracle Database>  
from hive table hive0:<Hive table name> using exttab
```

Learning Resources

These OHS blog entries can help you get started.

- [How to Load Oracle and Hive Tables with OHS \(Part 3 - Loading Oracle Tables\)](#)
- [How to Load Oracle and Hive tables using OHS \(Part 5 - Using "loadermap" when loading Oracle tables\)](#)
- [How to Load Oracle and Hive tables using OHS \(Part 6 - Using the "etl" method for loading Oracle tables\)](#)
- [Oracle Loader for Hadoop and Performance Tuning](#)

See the [Java™ Platform, Standard Edition 7 API Specification](#) for documentation on the `SimpleDateFormat` class.

Use Oracle Loader for Hadoop With the Hadoop Command Line Utility

Perform 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](#)."
2. Connect to Oracle Database and create the target table.
See "[Create the Target Table](#)."
3. Establish a secure connection to the Oracle Database.
See "[Establish Secure Connections to Oracle Database Using SSL and Oracle Wallet](#)."
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 ["Create a Job Configuration File."](#)
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 ."](#)
 8. Create a shell script to run the Oracle Loader for Hadoop job.
See ["Run a Loader Job."](#)
 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."](#)
 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 [Oracle SQL Connector for Hadoop Distributed File System .](#)
 - 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"](#) or ["About DataPumpOutputFormat."](#)
 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](#)
 - [Optimize Communications Between Oracle Engineered Systems](#)

About the Modes of Operation

Oracle Loader for Hadoop operates in two modes:

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

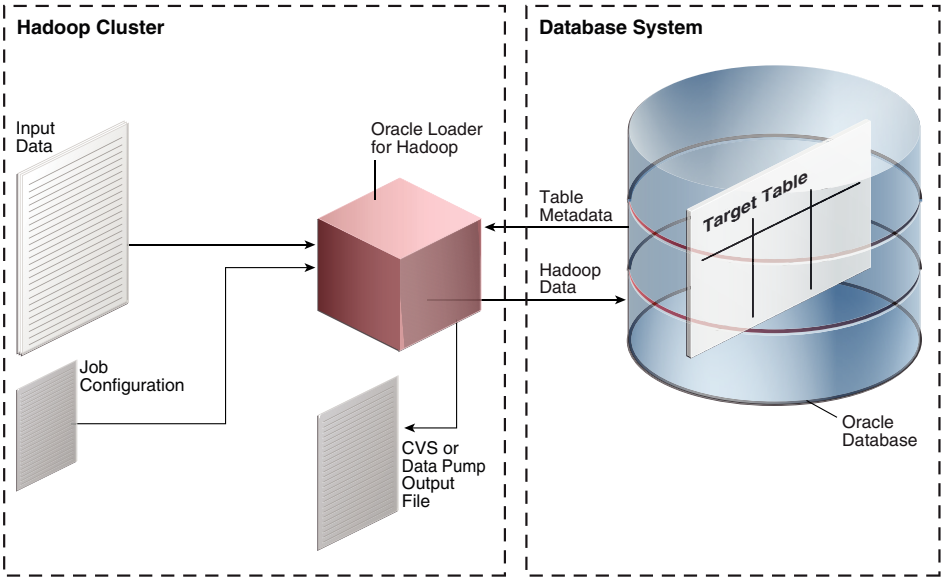
Online Database Mode

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.

The following figure shows the relationships among elements in online database mode.

Figure 3-1 Online Database Mode

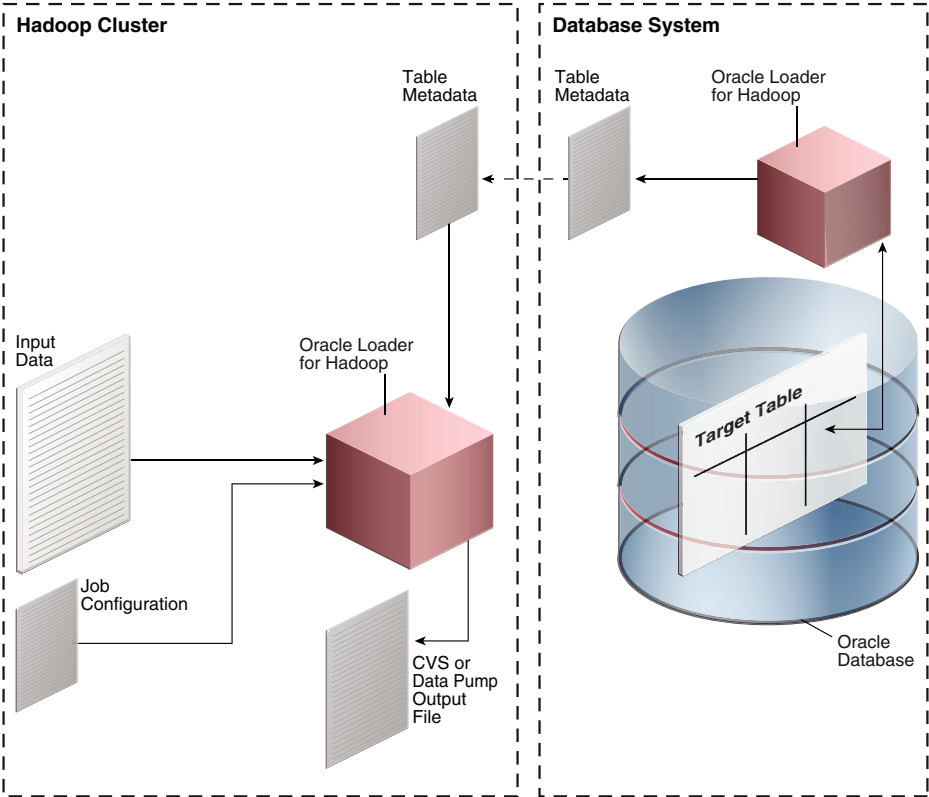


Offline Database Mode

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.

The following figure 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



Create 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

Compression

Oracle Loader for Hadoop does not compress data. Compressing data during load is defined by the table and database properties. To load data into a compressed table define the table and database properties accordingly.

Create 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](#)".

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

- How to secure connection to Oracle Database.
See "[Establish Secure Connections to Oracle Database Using SSL and Oracle Wallet](#)."
- The format of the input data.
See "[About Input Formats](#)."
- The format of the output data.

See "[About Output Formats](#)."

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 "[Run a Loader Job](#)."

You can use any text or XML editor to create the file. The following example 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.job.inputformat.class</name>
  <value>oracle.hadoop.loader.lib.input.DelimitedTextInputFormat</value>
</property>

<property>
  <name>mapreduce.input.fileinputformat.inputdir</name>
  <value>/user/oracle/moviedemo/session/*0000</value>
</property>

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

<property>
  <name>oracle.hadoop.loader.input.fieldNames</name>
  <value>SESSION_ID,TIME_IDDATE,CUST_ID,DURATION_SESSION,NUM_RATED,DURATION_RATED,NUM_C
  OMPLETED,DURATION_COMPLETED,TIME_TO_FIRST_START,NUM_STARTED,NUM_BROWSED,DURATION_BROW
  SED,NUM_LISTED,DURATION_LISTED,NUM_INCOMPLETE,NUM_SEARCHED</value>
</property>

<property>
  <name>oracle.hadoop.loader.defaultDateFormat</name>
  <value>yyyy-MM-dd:HH:mm:ss</value>
</property>

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

<property>
  <name>mapreduce.output.fileoutputformat.outputdir</name>
  <value>temp_out_session</value>
</property>

<!--                               Table information                               -->
<property>
  <name>oracle.hadoop.loader.loaderMap.targetTable</name>
  <value>movie_sessions_tab</value>
</property>
```

```

<!--          Connection information          -->

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

<property>
  <name>TCPPOINT</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> Note: Protect this file with 600 permissions since it has password
in clear text.</description>
</property>

</configuration>

```

Establish Secure Connections to Oracle Database Using SSL and Oracle Wallet

Learn about establishing secure connections to Oracle Database.

This section describes how to create and use an Oracle Wallet or the JDBC SSL protocol to create and establish highly secure connections to Oracle Database.

Use Oracle Wallets

Oracle Wallet is a secure software container that stores authentication and signing credentials. 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`

Use JDBC SSL

SSL is a widely used industry standard protocol that provides secure communication over a network. SSL provides authentication, data encryption, and data integrity. SSL is required when connecting to Oracle Cloud services such as Oracle Autonomous Data Warehouse Cloud Service.

Connect to Oracle Database Using JDBC SSL

Learn to connect to Oracle Database using JDBC SSL.

1. Follow the instructions in [Using JDBC SSL](#) to connect to Oracle Database.
2. Add the following properties in the job configuration file:

```
<property>
<name>oracle.hadoop.loader.connection.directory</name>
<value><directory_location></value>
<description>For example: /home/oracle/SSL_wallet</description>
</property>

<property>
<name>oracle.hadoop.loader.connection.tns_admin</name>
<value><directory_location></value>
<description>For example: /home/oracle/SSL_wallet</description>
</property>

<property>
<name>oracle.hadoop.loader.connection.tnsEntryName</name>
<value><tns_entry></value>
<description>The TNS Entry must use the tcps protocol in order to
activate SSL in the JDBC Thin driver.For example,
  inst1_ssl.</description>
</property>
```

Generate the Target Table Metadata for Offline Database Mode

Learn to generate 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 learn more about the OraLoaderMetadata utility, see [OraLoaderMetadata Utility](#).

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](#)."

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

A null replaces any empty-string token, whether enclosed or unenclosed.

`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.

Hadoop automatically decompresses compressed text files when they are read.

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.job.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. Hadoop automatically decompresses compressed files when they are read.



See Also:

Java Platform Standard Edition 6 Java Reference for more information about `java.util.regex` at

<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.job.inputformat.class` to

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

About HiveToAvroInputFormat

For nonpartitioned tables, `HiveToAvroInputFormat` imports the entire table, which is all files in the Hive table directory.

For partitioned tables, `HiveToAvroInputFormat` imports one or more of the partitions. You can either load or skip a partition. However, you cannot partially load a partition.

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](#)."

`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](#)".

Note:

This input format does not support Hive tables using quoted identifiers for column names. See [HIVE-6013](#)

Also note that `HiveToAvroInputFormat` does not enforce the *SQL Standard Based Hive Authorization*. For more information, see <https://cwiki.apache.org/confluence/display/Hive/SQL+Standard+Based+Hive+Authorization>.

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`

Optional Configuration Properties

To specify a subset of rows in the input Hive table to load, use the following property:

- `oracle.hadoop.loader.input.hive.rowFilter`

Avro Input Format

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

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

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

Configuration Properties

None

Oracle NoSQL Database Input Format

To load data from Oracle NoSQL Database, set `mapreduce.job.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/javadoc/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](#)."

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.job.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 "[Log Rejected Records in Bad Files](#)."

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](#)."

InputFormat Source Code Example

Oracle Loader for Hadoop provides the source code for an `InputFormat` example.

In order to access the examples, unzip file `examples.zip`, which is in `$OLH_HOME`. You can find the `InputFormat` example in the `examples/jsrc` directory.

The sample format loads data from a simple, comma-separated value (CSV) file. To use this input format, add `$OLH_HOME/examples/oraloader-examples.jar` to `HADOOP_CLASSPATH` and specify `oracle.hadoop.loader.examples.CSVInputFormat` as the value of `mapreduce.job.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. For example: When you load from a Hive table, the names of the Oracle target table columns exactly match the names of the Hive table columns.
- 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 identical to 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.

 **Note:**

Manual Mapping is particularly useful when different date columns have different formats.

Manual Mapping: Examples

The following are the examples of manual mapping:

Configuration File `conf.xml` when loading from a text file

When you load delimited text from text files on HDFS, use `F0`, `F1`, ... to refer to columns in text files. In this example, `F0` maps to `EMPLOYEE_ID` in the `columnNames` property, `F1` maps to `LAST_NAME`, and so on.

```
<?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>
  <property>
    <name>oracle.hadoop.loader.loaderMap.EMAIL.field</name>
    <value>F2</value>
  </property>
</configuration>
```

```

    <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>

```

Configuration File conf.xml when loading from a Hive table

```

<?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>EMPLOYEE_ID</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>LAST_NAME</value>
  </property>
  <property>
    <name>oracle.hadoop.loader.loaderMap.LAST_NAME.format</name>
    <value></value>
  </property>
  <property>
    <name>oracle.hadoop.loader.loaderMap.EMAIL.field</name>
    <value>EMAIL</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>HIRE_DATE</value>
  </property>

```



```

</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>JOB_ID</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.job.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.job.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.job.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](#)."

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.job.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 [Oracle SQL Connector for Hadoop Distributed File System](#).

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 `${mapreduce.output.fileoutputformat.outputdir}/_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`

The following example shows a sample SQL*Loader control file that might be generated by an output task.

Example 3-2 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](#)."

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.job.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 [Oracle SQL Connector for Hadoop Distributed File System](#).

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 `$`
`{mapreduce.output.fileoutputformat.outputdir}/_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

Run 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.

Unzip the `examples.zip` file in `$OLH_HOME` in order to use the `oraloader-examples.jar` file in the instructions below.

Basic Options

-conf job_config.xml

Identifies the job configuration file. See "[Create a Job Configuration File.](#)"

-libjars

Identifies the JAR files for the input format.

- When using the example input format, specify `$OLH_HOME/jlib/oraloader-examples.jar`. (You will first need to set up the example for use as described in [InputFormat Source Code Example.](#))
- 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 the JAR 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 uses a built-in input format and a job configuration file named `MyConf.xml`.

```
HADOOP_CLASSPATH="$OLH_HOME/jlib/*:$OLH_HOME/examples/oraloader-examples.jar:$HADOOP_CLASSPATH"

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

Specify Hive Input Format JAR Files

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

```
HADOOP_CLASSPATH="$OLH_HOME/jlib/*:hive_home/lib/*:hive_conf_dir:$HADOOP_CLASSPATH"
```

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) 5.8:

```
# hadoop jar $OLH_HOME/jlib/oraloader.jar oracle.hadoop.loader.OraLoader \
-conf MyConf.xml \
-libjars hive-exec-1.1.0-cdh5.8.0.jar, hive-metastore-1.1.0-cdh5.8.0.jar,
libfb303-0.9.3.jar
```

Specify 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 `${mapreduce.output.fileoutputformat.outputdir}/_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

Log 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.

Set 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.

Use 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`.

Partition report identifies the keys that are assigned to the various mappers. This report is saved in XML for the sampler to use; it does not contain information of use to you. The report is named `${mapreduce.output.fileoutputformat.outputdir}/_balancer/orabalancer_report.xml`. It is only generated for sampled jobs. This xml file contains the information about how to assign map output to different reducers, as well as the sampling statistics.

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.

Resolve 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](#)."

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.

Optimize 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. Set this Hadoop configuration property for the child task JVMs:

```
-D mapred.child.java.opts="-Doracle.net.SDP=true -Djava.net.preferIPv4Stack=true"
```

Note:

This Hadoop configuration property can be either added to the OLH command line or set in the configuration file.

3. Configure standard Ethernet communications. In the job configuration file, set `oracle.hadoop.loader.connection.url` using this syntax:

```
jdbc:oracle:thin:@(DESCRIPTION=(ADDRESS_LIST=
  (ADDRESS=(PROTOCOL=TCP)(HOST=hostName)(PORT=portNumber))
  (CONNECT_DATA=(SERVICE_NAME=serviceName)))
```

4. Configure the Oracle listener on Exadata to support the SDP protocol and bind it to a specific port address (such as 1522). In the job configuration file, specify the listener address as the value of `oracle.hadoop.loader.connection.oci_url` using this syntax:

```
(DESCRIPTION=(ADDRESS=(PROTOCOL=SDP)
(HOST=hostName) (PORT=portNumber))
(CONNECT_DATA=(SERVICE_NAME=serviceName)))
```

Replace *hostName*, *portNumber*, and *serviceName* with the appropriate values to identify the SDP listener 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.

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

Property	Description
<code>mapreduce.job.name</code>	<p>Type: String</p> <p>Default Value: OraLoader</p> <p>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.</p>

Property	Description
<code>mapreduce.input.fileinputformat.inputdir</code>	<p>Type: String</p> <p>Default Value: Not defined</p> <p>Description: A comma-separated list of input directories.</p>
<code>mapreduce.job.inputformat.class</code>	<p>Type: String</p> <p>Default Value: Not defined</p> <p>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 <code>InputFormat</code> class:</p> <ul style="list-style-type: none"> <code>oracle.hadoop.loader.lib.input.AvroInputFormat</code> <code>oracle.hadoop.loader.lib.input.DelimitedTextInputFormat</code> <code>oracle.hadoop.loader.lib.input.HiveToAvroInputFormat</code> <code>oracle.hadoop.loader.lib.input.RegexInputFormat</code> <code>oracle.kv.hadoop.KVAvroInputFormat</code> <p>See "About Input Formats" for descriptions of the built-in input formats.</p>
<code>mapreduce.output.fileoutputformat.outputdir</code>	<p>Type: String</p> <p>Default Value: Not defined</p> <p>Description: A comma-separated list of output directories, which cannot exist before the job runs. Required.</p>
<code>mapreduce.job.outputformat.class</code>	<p>Type: String</p> <p>Default Value: Not defined</p> <p>Description: Identifies the output type. The values can be:</p> <ul style="list-style-type: none"> <code>oracle.hadoop.loader.lib.output.DataPumpOutputFormat</code> Writes data records into binary format files that can be loaded into the target table using an external table. <code>oracle.hadoop.loader.lib.output.DelimitedTextOutputFormat</code> Writes data records to delimited text format files such as comma-separated values (CSV) format files. <code>oracle.hadoop.loader.lib.output.JDBCOutputFormat</code> Inserts rows into the target table using a JDBC connection. <code>oracle.hadoop.loader.lib.output.OCIOutputFormat</code> Inserts rows into the target table using the Oracle OCI Direct Path interface. <p>See "About Output Formats."</p>

Property	Description
<code>mapreduce.job.reduces</code>	<p>Type: Integer</p> <p>Default Value: 1</p> <p>Description: The number of reduce tasks used by the Oracle Loader for Hadoop job. The default value of 1 does not support parallel processing, therefore performance improves when the value is increased to support multiple parallel data loads. Choose a value that provides an ample, but not excessive, number of reduce tasks for the job. At a point dictated by the available resources, an excessive increase in the number of reduce tasks result in diminishing improvements, while potentially degrading the performance of other jobs.</p>

OraLoader Configuration Properties

Property	Description
<code>oracle.hadoop.loader.badRecordFlushInterval</code>	<p>Type: Integer</p> <p>Default Value: 500</p> <p>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.</p> <p>The oracle.hadoop.loader.logBadRecords property must be set to <code>true</code> for a flush interval to take effect.</p>
<code>oracle.hadoop.loader.compressionFactors</code>	<p>Type: Decimal</p> <p>Default Value: BASIC=5.0,OLTP=5.0,QUERY_LOW=10.0,QUERY_HIGH=10.0,ARCHIVE_LOW=10.0,ARCHIVE_HIGH=10.0</p> <p>Description: These values are used by Oracle Loader for Hadoop when sampling is enabled and the target table is compressed. They are the compression factors of the target table. For best performance, the values of this property should match the compression factors of the target table. The values are a comma-delimited list of <i>name=value</i> pairs. The names must be one of the following keywords:</p> <p>ARCHIVE_HIGH ARCHIVE_LOW BASIC OLTP QUERY_HIGH QUERY_LOW</p>
<code>oracle.hadoop.loader.connection.defaultExecuteBatch</code>	<p>Type: Integer</p> <p>Default Value: 100</p> <p>Description: The number of records inserted in one trip to the database. It applies only to <code>JDBCOutputFormat</code> and <code>OCIOutputFormat</code>.</p> <p>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.</p> <p>A value less than 1 sets the property to the default value.</p>

Property	Description
<code>oracle.hadoop.loader.connection.oci_url</code>	<p>Type: String</p> <p>Default Value: Value of <code>oracle.hadoop.loader.connection.url</code></p> <p>Description: The database connection string used by <code>OCIOutputFormat</code>. 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.</p> <pre>(DESCRIPTION=(ADDRESS_LIST=(ADDRESS=(PROTOCOL=SDP)(HOST=myhost)(PORT=1521))) (CONNECT_DATA=(SERVICE_NAME=my_db_service_name)))</pre> <p>This connection string does not require a "jdbc:oracle:thin:@" prefix. All characters up to and including the first at-sign (@) are removed.</p>
<code>oracle.hadoop.loader.connection.password</code>	<p>Type: String</p> <p>Default Value: Not defined</p> <p>Description: Password for the connecting user. Oracle recommends that you do not store your password in clear text. Use an Oracle wallet instead.</p>
<code>oracle.hadoop.loader.connection.sessionTimeZone</code>	<p>Type: String</p> <p>Default Value: LOCAL</p> <p>Description: Alters the session time zone for database connections. Valid values are:</p> <ul style="list-style-type: none"> • <code>[+ -]hh:mm</code>: Hours and minutes before or after Coordinated Universal Time (UTC), such as <code>-5:00</code> for Eastern Standard Time • LOCAL: The default time zone of the JVM • <code>time_zone_region</code>: A valid JVM time zone region, such as EST (for Eastern Standard Time) or America/New_York <p>This property also determines the default time zone for input data that is loaded into <code>TIMESTAMP WITH TIME ZONE</code> and <code>TIMESTAMP WITH LOCAL TIME ZONE</code> database column types.</p>
<code>oracle.hadoop.loader.connection.cluster.tns_admin</code>	<p>Type: String</p> <p>Default Value: Not defined.</p> <p>Description: The TNS admin location on the cluster node if it is different from the client side location. By default, the client-side TNS admin location is the same as the location on cluster nodes and it is specified by oracle.hadoop.loader.connection.tns_admin. It is invalid to specify this property without specifying <code>oracle.hadoop.loader.connection.tns_admin</code>.</p>
<code>oracle.hadoop.loader.connection.directory</code>	<p>Type: String</p> <p>Default Value: Not defined</p> <p>Description: File path to a directory on each node of the Hadoop cluster.</p>

Property	Description
<code>oracle.hadoop.loader.connection.tns_admin</code>	<p>Type: String</p> <p>Default Value: Not defined</p> <p>Description: File path to a directory on each node of the Hadoop cluster, which contains SQL*Net configuration files such as <code>sqlnet.ora</code> and <code>tnsnames.ora</code>. Set this property so that you can use TNS entry names in database connection strings.</p> <p>You must set this property when using an Oracle wallet as an external password store (as Oracle recommends). See oracle.hadoop.loader.connection.wallet_location.</p>
<code>oracle.hadoop.loader.connection.tnsEntryName</code>	<p>Type: String</p> <p>Default Value: Not defined</p> <p>Description: A TNS entry name defined in the <code>tnsnames.ora</code> file. Use this property with oracle.hadoop.loader.connection.tns_admin.</p>
<code>oracle.hadoop.loader.connection.url</code>	<p>Type: String</p> <p>Default Value: Not defined</p> <p>Description: The URL of the database connection. This property overrides all other connection properties.</p> <p>If an Oracle wallet is configured as an external password store (as Oracle recommends), then the property value must start with the <code>jdbc:oracle:thin:@</code> driver prefix, and the database connection string must exactly match the credential in the wallet. See oracle.hadoop.loader.connection.wallet_location.</p> <p>The following examples show valid values of connection URLs:</p> <ul style="list-style-type: none"> Oracle Net Format: <pre> jdbc:oracle:thin:@(DESCRIPTION=(ADDRESS_LIST= (ADDRESS=(PROTOCOL=TCP)(HOST=myhost) (PORT=1521))) (CONNECT_DATA=(SERVICE_NAME=example_service_name))) </pre> TNS Entry Format: <pre> jdbc:oracle:thin:@myTNSEntryName </pre> Thin Style: <pre> jdbc:oracle:thin:@//myhost:1521/my_db_service_name </pre>
<code>oracle.hadoop.loader.connection.user</code>	<p>Type: String</p> <p>Default Value: Not defined</p> <p>Description: A database user name. This property requires that you also set oracle.hadoop.loader.connection.password. However, Oracle recommends that you use an Oracle wallet to store your password. Do not store it in clear text.</p> <p>When using online database mode, you must set either this property or oracle.hadoop.loader.connection.wallet_location.</p>

Property	Description
<code>oracle.hadoop.loader.connection.wallet_location</code>	<p>Type: String</p> <p>Default Value: Not defined</p> <p>Description: File path to an Oracle wallet directory on each node of the Hadoop cluster, where the connection credentials are stored.</p> <p>When using an Oracle wallet, you must also set the following properties:</p> <ul style="list-style-type: none"> <code>oracle.hadoop.loader.connection.tns_admin</code> <code>oracle.hadoop.loader.connection.url</code> or <code>oracle.hadoop.loader.connection.tnsEntryName</code>
<code>oracle.hadoop.loader.connection.cluster.wallet_location</code>	<p>Type: String</p> <p>Default Value: Not defined.</p> <p>Description: The wallet location on the cluster node if it is different from the client-side location.</p> <p>By default, the client-side wallet location is the same as the location on cluster node and it is specified by <code>oracle.hadoop.loader.connection.wallet_location</code>.</p> <p>It is invalid to specify this property without specifying <code>oracle.hadoop.loader.connection.wallet_location</code>.</p>
<code>oracle.hadoop.loader.defaultDateFormat</code>	<p>Type: String</p> <p>Default Value: <code>yyyy-MM-dd HH:mm:ss</code></p> <p>Description: Parses an input field into a DATE column using a <code>java.text.SimpleDateFormat</code> pattern and the default locale. If the input file requires different patterns for different fields, then use the manual mapping properties. See "Manual Mapping."</p>
<code>oracle.hadoop.loader.enableSorting</code>	<p>Type: Boolean</p> <p>Default Value: <code>true</code></p> <p>Description: Controls whether output records within each reducer group are sorted. Use the <code>oracle.hadoop.loader.sortKey</code> property to identify the columns of the target table to sort by. Otherwise, Oracle Loader for Hadoop sorts the records by the primary key.</p>
<code>oracle.hadoop.loader.enforceClasspath</code>	<p>Type: Boolean</p> <p>Default Value: <code>true</code></p> <p>To prevent mismatched versions of its JARs from being added to the classpath, Oracle Loader for Hadoop checks that its internal classes are loaded from <code>\${oracle.hadoop.loader.olh_home}/jlib/jars</code>.</p> <p>To disable this check, set the property to <code>false</code>.</p>
<code>oracle.hadoop.loader.extTabDirectoryName</code>	<p>Type: String</p> <p>Default Value: <code>OLH_EXTTAB_DIR</code></p> <p>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.</p> <p>This property applies only to <code>DelimitedTextOutputFormat</code> and <code>DataPumpOutputFormat</code>.</p>

Property	Description
oracle.hadoop.loader.input.fieldNames	<p>Type: String</p> <p>Default Value: F0,F1,F2,...</p> <p>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" for loading only selected fields.</p> <p>The names are used to create the Avro schema for the record, so they must be valid JSON name strings.</p> <p>This property applies to <code>DelimitedTextInputFormat</code> and <code>RegexInputFormat</code> only.</p>
oracle.hadoop.loader.input.fieldTerminator	<p>Type: String</p> <p>Default Value: , (comma)</p> <p>Description: A character that indicates the end of an input field for <code>DelimitedTextInputFormat</code>. The value can be either a single character or <code>\uHHHH</code>, where <code>HHHH</code> is the character's UTF-16 encoding.</p>
oracle.hadoop.loader.input.hive.databaseName	<p>Type: String</p> <p>Default Value: Not defined</p> <p>Description: The name of the Hive database where the input table is stored</p>

Property	Description
oracle.hadoop.loader.input.hive.partitionFilter	<p data-bbox="633 268 1458 300">Type: String</p> <p data-bbox="633 310 1458 342">Default Value: Not defined</p> <p data-bbox="633 352 1458 573">Description: A valid HiveQL expression that is used to filter the source Hive table partitions for <code>HiveToAvroInputFormat</code>. The expression must contain <i>only</i> partition columns. Including other columns does not raise an error, but unintended consequences can result. Oracle recommends that you not include other columns. If the value is not set, then Oracle Loader for Hadoop loads the data from all partitions of the source Hive table. This property is ignored if the table is not partitioned. It is also ignored if <code>oracle.hadoop.loader.input.hive.rowFilter</code> is set.</p> <p data-bbox="633 583 1458 615">The expression must conform to the following restrictions:</p> <ul data-bbox="633 625 1458 961" style="list-style-type: none"> • Selects partitions and not individual records inside the partitions. • Does not include columns that are not used to partition the table, because they might cause unintended consequences. • Does not include subqueries. • Does not include user-defined functions (UDFs), which are not supported; built-in functions are supported. • Resolves all variable expansions at the Hadoop level. Hive variable name spaces (such as <code>env:</code>, <code>system:</code>, <code>hiveconf:</code>, and <code>hivevar:</code>) have no meaning. Oracle Loader for Hadoop sets <code>hive.variable.substitute</code> to <code>false</code>, which disables Hive variable expansion. You can choose between these expansion methods: <p data-bbox="633 972 1458 1024">Expand all variables before setting this property: In the Hive CLI, use the following commands:</p> <pre data-bbox="698 1066 1209 1161">CREATE VIEW <i>view_name</i> AS SELECT * from <i>database.table_name</i> WHERE <i>expression</i>; DESCRIBE FORMATTED <i>view_name</i>;</pre> <p data-bbox="633 1203 1458 1266">The View Original Text field contains the query with all variables expanded. Copy the where clause, starting after where.</p> <p data-bbox="633 1276 1458 1371">Define all variables in Oracle Loader for Hadoop: In the <code>hadoop</code> command to run Oracle Loader for Hadoop, use the generic options (<code>-D</code> and <code>-conf</code>).</p> <p data-bbox="633 1381 1458 1434">You can use the Hive CLI to test the expression and ensure that it returns the expected results.</p> <p data-bbox="633 1444 1458 1476">The following examples assume a source table defined with this command:</p> <pre data-bbox="649 1518 1347 1602">CREATE TABLE t(c string) PARTITIONED BY (p1 string, p2 int, p3 boolean, p4 string, p5 timestamp);</pre> <p data-bbox="633 1654 1458 1686">Example 1: Nested Expressions</p> <pre data-bbox="649 1717 1274 1772">p1 like 'abc%' or (p5 >= '2010-06-20' and p5 <= '2010-07-03')</pre>


 **Note:**

This property is deprecated. Use `oracle.hadoop.loader.input.hive.rowFilter` instead.

Property	Description
	Example 2: Built-in Functions
	<code>year(p5) = 2014</code>
	Example 3: Bad Usage: Columns That Are Not Used to Partition the Table
	These examples show that using <code>c</code> , a column that is not used to partition the table, is unnecessary and can cause unexpected results.
	This example is equivalent to <code>p2 > 35</code> :
	<code>p2 > 35 and c like 'abc%'</code>
	This example loads all partitions. All partitions could contain <code>c like 'abc %'</code> , so partitions are filtered out:
	<code>p2 > 35 or c like 'abc%'</code>

Property	Description
oracle.hadoop.loader.input.hive.rowFilter	<p>Type: String</p> <p>Default Value: Not defined</p> <p>Description: A valid HiveQL expression that is used to filter the rows of the source Hive table for HiveToAvroInputFormat. If this value is not set (default), Oracle Loader for Hadoop attempts to use the value of <code>oracle.hadoop.loader.input.hive.partitionFilter</code> (provided the table is partitioned). Otherwise, Oracle Loader for Hadoop loads the entire source hive table.</p> <p>The expression must conform to the following restrictions:</p> <ul style="list-style-type: none"> • Does not include subqueries. • Does not include user-defined functions (UDFs), which are not supported; built-in functions are supported. • Resolves all variable expansions at the Hadoop level. Hive variable name spaces (such as <code>env:</code>, <code>system:</code>, <code>hiveconf:</code>, and <code>hivevar:</code>) have no meaning. Oracle Loader for Hadoop sets <code>hive.variable.substitute</code> to <code>false</code>, which disables Hive variable expansion. You can choose between these expansion methods: <ul style="list-style-type: none"> – Expand all variables before setting this property: In the Hive CLI, use the following commands: <pre data-bbox="747 898 1258 989">CREATE VIEW <i>view_name</i> AS SELECT * from <i>database.table_name</i> WHERE <i>expression</i>; DESCRIBE FORMATTED <i>view_name</i>;</pre> <p>The View Original Text field contains the query with all variables expanded. Copy the expression within the WHERE clause. (Do not include the WHERE keyword itself.)</p> – Define all variables in Oracle Loader for Hadoop. In the Hadoop command to run Oracle Loader for Hadoop, use the generic options (<code>-D</code> and <code>-conf</code>). <p>In both cases you can use the Hive CLI to test the expression and ensure that it returns the expected results. The following examples assume a source table defined with this command:</p> <pre data-bbox="651 1350 1347 1440">CREATE TABLE t(c string) PARTITIONED BY (p1 string, p2 int, p3 boolean, p4 string, p5 timestamp);</pre> <p>Example #1: nested expressions</p> <pre data-bbox="651 1556 1279 1614">c like 'abc%' and (p5 <= '2010-06-20' and p5 <= '2010-07-03')</pre> <p>Example #2: built-in functions</p> <pre data-bbox="651 1734 865 1761">year(p5) = 2013)</pre> <p>Oracle recommends that you turn on <code>hive.optimize.index.filter</code> when importing a subset of rows from a native Hive table (a table that is not managed by a storage handler). This is known to help input formats such as ORC and PARQUET, however there are several caveats:</p>

Property	Description
	<ul style="list-style-type: none"> The property must be set with a <code>-D</code> (using <code>-conf</code> will not work). Alternatively, the property can be set in <code>hive-site.xml</code>. This does not work for ORC tables in Hive 0.12.
oracle.hadoop.loader.input.hive.tableName	<p>Type: String</p> <p>Default Value: Not defined</p> <p>Description: The name of the Hive table where the input data is stored.</p>
oracle.hadoop.loader.input.initialFieldEncloser	<p>Type: String</p> <p>Default Value: Not defined</p> <p>Description: A character that indicates the beginning of a field. The value can be either a single character or <code>\uHHHH</code>, where <code>HHHH</code> is the character's UTF-16 encoding. To restore the default setting (no encloser), enter a zero-length value. A field encloser cannot equal the terminator or white-space character defined for the input format.</p> <p>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 enclosers are not set, then the parser reads each field as an unenclosed token.</p> <p>If you set this property but not oracle.hadoop.loader.input.trailingFieldEncloser, then the same value is used for both properties.</p>
oracle.hadoop.loader.input.regexCaseInsensitive	<p>Type: Boolean</p> <p>Default Value: <code>false</code></p> <p>Description: Controls whether pattern matching is case-sensitive. Set to <code>true</code> to ignore case, so that "string" matches "String", "STRING", "string", "StRiNg", and so forth. By default, "string" matches only "string".</p> <p>This property is the same as the <code>input.regex.case.insensitive</code> property of <code>org.apache.hadoop.hive.contrib.serde2.RegexSerDe</code>.</p>

Property	Description
oracle.hadoop.loader.input.regexPattern	<p>Type: String</p> <p>Default Value: Not defined</p> <p>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 <i>not</i> applied repeatedly to a line of input text.</p> <p>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.</p> <p>This property is the same as the <code>input.regex</code> property of <code>org.apache.hadoop.hive.contrib.serde2.RegexSerDe</code>.</p> <div data-bbox="857 709 1458 1054" style="border: 1px solid #0070c0; padding: 10px; margin-top: 10px;"> <p> See Also:</p> <p>For descriptions of regular expressions and capturing groups, the entry for <code>java.util.regex</code> in the <i>Java Platform Standard Edition 6 API Specification</i> at http://docs.oracle.com/javase/6/docs/api/java/util/regex/Pattern.html</p> </div>
oracle.hadoop.loader.input.trailingFieldEncloser	<p>Type: String</p> <p>Default Value: The value of oracle.hadoop.loader.input.initialFieldEncloser</p> <p>Description: Identifies a character that marks the end of a field. The value can be either a single character or <code>\uHHHH</code>, where <code>HHHH</code> is the character's UTF-16 encoding. For no trailing encloser, enter a zero-length value.</p> <p>A field encloser cannot be the terminator or a white-space character defined for the input format.</p> <p>If the trailing field encloser 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 <code>' '</code> (two single quotes) to load <code>'</code> (one single quote).</p> <p>If you set this property, then you must also set oracle.hadoop.loader.input.initialFieldEncloser.</p>

Property	Description
oracle.hadoop.loader.loadByPartition	<p>Type: Boolean</p> <p>Default Value: true</p> <p>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. <code>DelimitedTextOutputFormat</code> and <code>DataPumpOutputFormat</code> generate multiple files, and each file contains the records from one partition. For <code>DelimitedTextOutputFormat</code>, this property also controls whether the <code>PARTITION</code> keyword appears in the generated control files for <code>SQL*Loader</code>. <code>OCIOutputFormat</code> requires partitioned tables. If you set this property to <code>false</code>, then <code>OCIOutputFormat</code> turns it back on. For the other output formats, you can set <code>loadByPartition</code> to <code>false</code>, so that Oracle Loader for Hadoop handles a partitioned table as if it were nonpartitioned.</p>
oracle.hadoop.loader.loaderMap.columnNames	<p>Type: String</p> <p>Default Value: Not defined</p> <p>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.</p> <p>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.</p>
oracle.hadoop.loader.loaderMap.column_name.field	<p>Type: String</p> <p>Default Value: Normalized column name</p> <p>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.</p> <p>You must set oracle.hadoop.loader.loaderMap.columnNames, or this property is ignored.</p>
oracle.hadoop.loader.loaderMap.column_name.format	<p>Type: String</p> <p>Default Value: Not defined</p> <p>Description: Specifies the data format of the data being loaded into the column identified in the property name. Use a <code>java.text.SimpleDateFormat</code> pattern for a date format or regular expression patterns for text. Optional.</p> <p>You must set oracle.hadoop.loader.loaderMap.columnNames, or this property is ignored.</p>

Property	Description
<code>oracle.hadoop.loader.loaderMap.targetTable</code>	<p>Type: String</p> <p>Default Value: Not defined</p> <p>Description: A schema-qualified name for the table to be loaded. This property takes precedence over oracle.hadoop.loader.loaderMapFile.</p> <p>To load a subset of columns, set the oracle.hadoop.loader.loaderMap.columnNames property. With <code>columnNames</code>, 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 <code>columnNames</code>.</p>
<code>oracle.hadoop.loader.loaderMapFile</code>	<p>Loader maps are deprecated starting with Release 2.3. The <code>oracle.hadoop.loader.loaderMap.*</code> configuration properties replace loader map files. See "Manual Mapping."</p>
<code>oracle.hadoop.loader.log4j.propertyPrefix</code>	<p>Type: String</p> <p>Default Value: <code>log4j.logger.oracle.hadoop.loader</code></p> <p>Description: Identifies the prefix used in Apache log4j properties loaded from its configuration file.</p> <p>Oracle Loader for Hadoop enables you to specify log4j properties in the <code>hadoop</code> command using the <code>-conf</code> and <code>-D</code> options. For example:</p> <pre>-D log4j.logger.oracle.hadoop.loader.OraLoader=DEBUG -D log4j.logger.oracle.hadoop.loader.metadata=INFO</pre> <p>All configuration properties starting with this prefix are loaded into log4j. They override the settings for the same properties that log4j loaded from <code>{log4j.configuration}</code>. The overrides apply to the Oracle Loader for Hadoop job driver, and its map and reduce tasks.</p> <p>The configuration properties are copied to log4j with RAW values; any variable expansion is done for log4j. Any configuration variables to be used in the expansion must also start with this prefix.</p>
<code>oracle.hadoop.loader.logBadRecords</code>	<p>Type: Boolean</p> <p>Default Value: <code>false</code></p> <p>Description: Controls whether Oracle Loader for Hadoop logs bad records to a file.</p> <p>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.</p>

Property	Description
oracle.hadoop.loader.logRetentionPolicy	<p>Type: String</p> <p>Default Value: ALWAYS</p> <p>Description: Specifies when Oracle Loader for Hadoop logs should be generated/retained at the end of a job. Valid values are:</p> <ul style="list-style-type: none"> • ALWAYS – logs are generated and retained at the end of all jobs. • NEVER – logs are never retained. • ON_ERROR – logs are discarded unless an error is identified. <p>The following situations are considered errors for the purposes of logRetentionPolicy:</p> <ul style="list-style-type: none"> • Any non-zero exit code. • Any input record parse error (See oracle.hadoop.loader.rejectLimit and oracle.hadoop.loader.logBadRecords). • Any rejected rows (for OCIOutputFormat or JDBCOutputFormat). <p>The following files are covered by this property:</p> <ul style="list-style-type: none"> • \${mapreduce.output.fileoutputformat.outputdir}/_olh/* with the exception of *.ctl and *.sql files. • \${mapreduce.output.fileoutputformat.outputdir}/_balancer/
oracle.hadoop.loader.olh_home	<p>Type: String</p> <p>Default Value: Value of the OLH_HOME environment variable</p> <p>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.</p>
oracle.hadoop.loader.olhcachePath	<p>Type: String</p> <p>Default Value: \${mapreduce.output.fileoutputformat.outputdir}/../olhcache</p> <p>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.</p> <p>The distributed cache is a facility for caching large, application-specific files and distributing them efficiently across the nodes in a cluster.</p>



See Also:

The description of `org.apache.hadoop.filecache.DistributedCache` in the Java documentation at <http://hadoop.apache.org/>

Property	Description
<code>oracle.hadoop.loader.output.degreeOfParallelism</code>	<p>Type: String</p> <p>Default Value: Not defined</p> <p>Description: If set, the value of this property controls either:</p> <ul style="list-style-type: none"> The number of reduce tasks (if Oracle Loader for Hadoop triggers a map-reduce job), or, The number of map tasks (if Oracle Loader for Hadoop triggers a map-only job). <p>If this property is not set, then the value of <code>\${mapreduce.job.reduces}</code> is used for the number of reduce tasks.</p> <ul style="list-style-type: none"> If the value for <code>\${mapreduce.job.reduces}</code> was set explicitly for the job (set through a <code>-D</code>, a <code>-conf</code>, or set programmatically), then that value will be used as is. If the value for <code>\${mapreduce.job.reduces}</code> was not set explicitly for the job (e.g. set in one of the <code>*-{site default}.xml</code> configuration files), then the value is considered to be a cluster-wide setting and is limited to a maximum value of 64. You can avoid this by explicitly setting <code>oracle.hadoop.loader.output.degreeOfParallelism</code> or <code>mapreduce.job.reduces</code> explicitly. <p>This property provides a unified way to limit the number of database connections made by <code>OCIOutputFormat</code> and <code>JDBCOutputFormat</code>. However, the property is enforced on all output formats in order to facilitate debugging scenarios. For example, you can replace <code>OCIOutputFormat</code> with <code>DelimitedTextOutputFormat</code> in order to see what data is being processed by a particular reduce task.</p>
<code>oracle.hadoop.loader.output.dirpathBufsize</code>	<p>Type: Integer</p> <p>Default Value: 131072 (128 KB)</p> <p>Description: Sets the size in bytes of the direct path stream buffer for <code>OCIOutputFormat</code>. Values are rounded up to the next multiple of 8 KB.</p>
<code>oracle.hadoop.loader.output.escapeEnclosers</code>	<p>Type: Boolean</p> <p>Default Value: false</p> <p>Description: Controls whether the embedded trailing enclosure character is handled as literal text (that is, escaped). Set this property to <code>true</code> when a field may contain the trailing enclosure character as part of the data value. See oracle.hadoop.loader.output.trailingFieldEncloser.</p>
<code>oracle.hadoop.loader.output.fieldTerminator</code>	<p>Type: String</p> <p>Default Value: , (comma)</p> <p>Description: A character that indicates the end of an output field for <code>DelimitedTextInputFormat</code>. The value can be either a single character or <code>\uHHHH</code>, where <code>HHHH</code> is the character's UTF-16 encoding.</p>

Property	Description
oracle.hadoop.loader.output.granuleSize	<p>Type: Integer</p> <p>Default Value: 10240000</p> <p>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.</p>
	<div style="border: 1px solid #0070c0; padding: 10px; background-color: #e6f2ff;"> <p> See Also:</p> <p><i>Oracle Database Utilities</i> for more information about the ORACLE_DATAPUMP access driver.</p> </div>
oracle.hadoop.loader.output.initialFieldEncloser	<p>Type: String</p> <p>Default Value: Not defined</p> <p>Description: A character generated in the output to identify the beginning of a field. The value must be either a single character or <code>\uHHHH</code>, where <code>HHHH</code> is the character's UTF-16 encoding. A zero-length value means that no enclosers are generated in the output (default value).</p> <p>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 <code>true</code>.</p> <p>If you set this property, then you must also set oracle.hadoop.loader.output.trailingFieldEncloser.</p>
oracle.hadoop.loader.output.trailingFieldEncloser	<p>Type: String</p> <p>Default Value: Value of oracle.hadoop.loader.output.initialFieldEncloser</p> <p>Description: A character generated in the output to identify the end of a field. The value must be either a single character or <code>\uHHHH</code>, where <code>HHHH</code> is the character's UTF-16 encoding. A zero-length value means that there are no enclosers (default value).</p> <p>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 <code>true</code>.</p> <p>If you set this property, then you must also set oracle.hadoop.loader.output.initialFieldEncloser.</p>
oracle.hadoop.loader.rejectLimit	<p>Type: Integer</p> <p>Default Value: 1000</p> <p>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.</p> <p>If <code>mapreduce.map.speculative</code> is <code>true</code> (the default), then the number of rejected records may be inflated temporarily, causing the job to stop prematurely.</p> <p>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.</p>

Property	Description
<code>oracle.hadoop.loader.sampler.enableSampling</code>	<p>Type: Boolean</p> <p>Default Value: <code>true</code></p> <p>Description: Controls whether the sampling feature is enabled. Set this property to <code>false</code> to disable sampling.</p> <p>Even when <code>enableSampling</code> is set to <code>true</code>, 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.</p>
<code>oracle.hadoop.loader.sampler.hintMaxSplitSize</code>	<p>Type: Integer</p> <p>Default Value: 1048576 (1 MB)</p> <p>Description: Sets the Hadoop <code>mapred.max.split.size</code> property for the sampling process; the value of <code>mapred.max.split.size</code> does not change for the job configuration. A value less than 1 is ignored.</p> <p>Some input formats (such as <code>FileInputFormat</code>) use this property as a hint to determine the number of splits returned by <code>getSplits</code>. Smaller values imply that more chunks of data are sampled at random, which results in a better sample.</p> <p>Increase this value for data sets with tens of terabytes of data, or if the input format <code>getSplits</code> method throws an out-of-memory error.</p> <p>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 <code>getSplits</code> to run out of memory by returning too many splits.</p> <p>The <code>org.apache.hadoop.mapreduce.lib.input.FileInputFormat</code> method always returns splits at least as large as the minimum split size setting, regardless of the value of this property.</p>
<code>oracle.hadoop.loader.sampler.hintNumMapTasks</code>	<p>Type: Integer</p> <p>Default Value: 100</p> <p>Description: Sets the value of the Hadoop <code>mapred.map.tasks</code> configuration property for the sampling process; the value of <code>mapred.map.tasks</code> does not change for the job configuration. A value less than 1 is ignored.</p> <p>Some input formats (such as <code>DBInputFormat</code>) use this property as a hint to determine the number of splits returned by the <code>getSplits</code> method. Higher values imply that more chunks of data are sampled at random, which results in a better sample.</p> <p>Increase this value for data sets with more than a million rows, but remember that extremely large values can cause <code>getSplits</code> to run out of memory by returning too many splits.</p>

Property	Description
oracle.hadoop.loader.sampler.loadCI	<p>Type: Decimal</p> <p>Default Value: 0.95</p> <p>Description: The statistical confidence indicator for the maximum reducer load factor.</p> <p>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.</p> <p>See oracle.hadoop.loader.sampler.maxLoadFactor.</p>
oracle.hadoop.loader.sampler.maxHeapBytes	<p>Type: Integer</p> <p>Default Value: -1</p> <p>Description: Specifies in bytes the maximum amount of memory available to the sampler.</p> <p>Sampling stops when one of these conditions is true:</p> <ul style="list-style-type: none">• 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	<p>Type: Float</p> <p>Default Value: 0.05 (5%)</p> <p>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.</p> <p>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.</p> <p>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 $(\text{assigned_load} - \text{ideal_load}) / \text{ideal_load}$. If load balancing is successful, the job runs within the maximum load factor at the specified confidence.</p> <p>See oracle.hadoop.loader.sampler.loadCI.</p>

Property	Description
oracle.hadoop.loader.sampler.maxSamplesPct	<p>Type: Float</p> <p>Default Value: 0.01 (1%)</p> <p>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.</p> <p>This property accepts a range of 0 to 1 (0% to 100%). A negative value disables it.</p> <p>Sampling stops when one of these conditions is true:</p> <ul style="list-style-type: none"> 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	<p>Type: Integer</p> <p>Default Value: 5</p> <p>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 <code>minSplits</code>, then the sampler reads from all the input splits.</p> <p>A number less than or equal to 0 is the same as a value of 1.</p>
oracle.hadoop.loader.sampler.numThreads	<p>Type: Integer</p> <p>Default Value: 5</p> <p>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.</p> <p>Set the value based on the processor and memory resources available on the node where you start the Oracle Loader for Hadoop job.</p>
oracle.hadoop.loader.sortKey	<p>Type: String</p> <p>Default Value: Not defined</p> <p>Description: A comma-delimited list of column names that forms a key for sorting output records within a reducer group.</p> <p>The column names can be quoted or unquoted identifiers:</p> <ul style="list-style-type: none"> A quoted identifier begins and ends with double quotation marks ("). An unquoted identifier is converted to uppercase before use.
oracle.hadoop.loader.tableMetadataFile	<p>Type: String</p> <p>Default Value: Not defined</p> <p>Description: Path to the target table metadata file. Set this property when running in offline database mode.</p> <p>Use the <code>file://</code> syntax to specify a local file, for example:</p> <pre>file:///home/jdoe/metadata.xml</pre> <p>To create the table metadata file, run the <code>OraLoaderMetadata</code> utility. See "OraLoaderMetadata Utility."</p>
oracle.hadoop.loader.targetTable	<p>Deprecated. Use oracle.hadoop.loader.loaderMap.targetTable.</p>

Oracle NoSQL Database Configuration Properties

Property	Description
<code>oracle.kv.kvstore</code>	<p>Type: String</p> <p>Default Value: Not defined</p> <p>Description: The name of the KV store with the source data.</p>
<code>oracle.kv.hosts</code>	<p>Type: String</p> <p>Default Value: Not defined</p> <p>Description: An array of one or more <i>hostname:port</i> pairs that identify the hosts in the KV store with the source data. Separate multiple pairs with commas.</p>
<code>oracle.kv.batchSize</code>	<p>Type: Key</p> <p>Default Value: Not defined</p> <p>Description: The desired number of keys for <code>KVAvroInputFormat</code> to fetch during each network round trip. A value of zero (0) sets the property to a default value.</p>
<code>oracle.kv.parentKey</code>	<p>Type: String</p> <p>Default Value: Not defined</p> <p>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 <code>KVAvroInputFormat</code> returns all keys in the store.</p>
<code>oracle.kv.subRange</code>	<p>Type: KeyRange</p> <p>Default Value: Not defined</p> <p>Description: Further restricts the returned values to a particular child under the parent key specified by oracle.kv.parentKey.</p>
<code>oracle.kv.depth</code>	<p>Type: Depth</p> <p>Default Value: PARENT_AND_DESCENDENTS</p> <p>Description: Restricts the returned values to a particular hierarchical depth under the value of oracle.kv.parentKey. The following keywords are valid values:</p> <ul style="list-style-type: none"> • 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.

Property	Description
oracle.kv.consistency	<p>Type: Consistency</p> <p>Default Value: NONE_REQUIRED</p> <p>Description: The consistency guarantee for reading child key-value pairs. The following keywords are valid values:</p> <ul style="list-style-type: none"> • 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	<p>Type: Long</p> <p>Default Value:</p> <p>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.</p>
oracle.kv.formatterClass	<p>Type: String</p> <p>Default Value: Not defined</p> <p>Description: Specifies the name of a class that implements the <code>AvroFormatter</code> interface to format <code>KeyValueVersion</code> instances into Avro <code>IndexedRecord</code> strings.</p> <p>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.</p>

4

Ease of Use Tools for Oracle Big Data Connectors

Oracle Big Data Connectors are a powerful toolset for data interchange between Hadoop and Oracle Database. Oracle Shell for Hadoop Loaders is an ease-of-use tool for using Oracle Big Data Connectors.

Introducing Oracle Shell for Hadoop Loaders

What is Oracle Shell for Hadoop Loaders?

Oracle Shell for Hadoop Loaders (OHSB) is a helper shell that provides an easy-to-use command line interface to Oracle Loader for Apache Hadoop, Oracle SQL Connector for HDFS, and Copy to Hadoop. It has basic shell features such as command line recall, history, inheriting environment variables from the parent process, setting new or existing environment variables, and performing environmental substitution in the command line.

The core functionality of Oracle Shell for Hadoop Loaders includes the following:

- Defining named external resources with which Oracle Shell for Hadoop Loaders interacts to perform loading tasks.
- Setting default values for load operations.
- Running load commands.
- Delegating simple pre and post load tasks to the Operating System, HDFS, Hive and Oracle. These tasks include viewing the data to be loaded, and viewing the data in the target table after loading.

See Also:

The examples directory in the OHSB kit contains many examples that define resources and load data using Oracle Shell for Hadoop Loaders. Unzip `<OHSB_KIT>/examples.zip` and see `<OHSB_KIT>/examples/README.txt` for a description of the examples and instructions on how to run OHSB load methods.

Configure Oracle Shell for Hadoop Loaders Interface (OHSB)

Confirm the location of the OHSB install and set `OHSB_HOME`. For example:

1. Set `OHSB_HOME`.

```
$ export OHSB_HOME=/opt/oracle/ohsh-5.0.0
```


2. Edit \$OHS_HOME/bin/ohsh_config.sh to set the following environment variables:

- a.**
- OLH_HOME: Installation location of Oracle Loader for Hadoop. For example:

```
export OLH_HOME=/opt/oracle/oraloader-5.0.0-h2
```

- b.**
- HS2_HOST_PORT: Hiveserver2 server and port information. For example:

```
export HS2_HOST_PORT = <server>:<port>
```

- c.**
- HADOOP_CONF_DIR: Hadoop conf directory. For example:

```
export HADOOP_CONF_DIR=/etc/hadoop/conf
```

- d.**
- HIVE_CONF_DIR: Hive conf directory. For example:

```
export HIVE_CONF_DIR=/etc/hive/conf
```

- e.**
- TNS_ADMIN: To identify the Oracle Database.

- If you are using a standard JDBC connection instead of SSL or Oracle Wallet, then you need to copy `tnsnames.ora` and `sqlnet.ora` from `$TNS_ADMIN` on the database host to a `<directory_location>` on the system where you plan to run OHS.

```
export TNS_ADMIN=<directory_location>
```

- If you are using Oracle Wallet to store database credentials, then create the wallet file on the database host and copy it to a `<wallet_location>` on the system where you plan to run OHS. In addition to `TNS_ADMIN`, set `WALLET_LOCATION_DIR` to the location of the Wallet file.

```
export WALLET_LOCATION=<wallet_location>
```

 **Note:**

Make sure that `sqlnet.ora` has wallet enabled and points to the correct wallet location. For example:

```
WALLET_LOCATION=
(SOURCE=(METHOD=FILE)
(METHOD_DATA=(DIRECTORY=<wallet_location>)))
sqlnet.wallet_override=true
```

- If you are using SSL, `TNS_ADMIN` is the directory where the SSL client wallet files reside. For example:

```
export TNS_ADMIN=/home/oracle/SSL_wallet
```

- f. `SQLCLIENT_HOME`: Database client installation directory. For example:

```
export SQLCLIENT_HOME=/usr/lib/oracle/12.1/client64
```

- g. Set `HIVE_SESS_VAR_LIST` for any required Hive session variables. For example:

- To connect to HiveServer2 with Kerberos authentication:

```
export HIVE_SESS_VAR_LIST="principal=<The server principal of HiveServer2>"
```

The Hive principal is specified by the configuration property `hive.server2.authentication.kerberos.principal` in `hive-site.xml`.

- To connect to HiveServer2 running in HTTP mode:

```
export HIVE_SESS_VAR_LIST="transportMode=http; httpPath=<The HTTP endpoint>"
```

- To connect to HiveServer2 when SSL is enabled:

```
export HIVE_SESS_VAR_LIST="ssl=true; sslTrustStore=<Trust Store path>; trustStorePassword=<Trust Store password>"
```

3. Add `$OHS_HOME/bin/ohsh` to your `$PATH` variable.

```
$ export PATH=$PATH:$OHS_HOME/bin/ohsh
```

Get Started with Oracle Shell for Hadoop Loaders

Starting an OHS Interactive Session

To start an interactive session, enter `ohsh` on the command line. This brings you to the OHS shell (if you have OHS in your path):

```
$ ohsh  
ohsh>
```

You can execute OHS commands in this shell (using the OHS syntax). You can also execute commands for Beeline/Hive, Hadoop, Bash, and SQL*Plus. For non-OHS commands, you add a delegation operator prefix ("%") to the name of the resource used to execute the command. For example:

```
ohsh> %bash0 ls -l
```

Scripting OHS

You can also script the same commands that work in the CLI. The `ohsh` command provides three parameters for working with scripts.

- `ohsh -i <filename>.ohsh`

The `-i` parameter tells OSH to initialize an interactive session with the commands in the script before the prompt appears. This is a useful way to set up the required session resources and automate other preliminary tasks before you start working within the shell.

```
$ ohsh -i initresources.ohsh
```

- `ohsh -f <filename>.ohsh`

The `ohsh` command with the `-f` parameter starts a non-interactive session and runs the commands in the script.

```
$ ohsh -f myunattendedjobs.ohsh
```

- `ohsh -i -f <filename>.ohsh`

You can use `-i` and `-f` together to initialize a non-interactive session and then run another script in the session.

```
$ ohsh -i mysetup.ohsh -f myunattendedjobs.ohsh
```

- `ohsh -c`

This command dumps all Hadoop configuration properties that an OSH session inherits at start up.

Working With OSH Resources

A resource is some named entity that OSH interacts with. For example: a Hadoop cluster is a resource, as is a JDBC connection to an Oracle database, a Hive database, a SQL session with an Oracle database, and a Bash shell on the local OS.

OSH provides two default resources at start up: `hive0` (to connect to the default Hive database) and `hadoop0`.

- Using `hive0` resource to execute a Hive command:

```
ohsh> %hive0 show tables
```

You can create additional Hive resources to connect to other Hive databases.

- Using the `hadoop0` resource to execute a Hadoop command:

```
ohsh> %hadoop0 fs -ls
```

Within an interactive or scripted session, you can create instances of additional resources, such as SSL, SQL, and JDBC. You need to create these three resources in order to connect to Oracle Database through OSH.

- **Creating a SQL resource:**

```
ohsh> create sqlplus resource sql0 connectid="bigdatalite.localdomain:  
1521/orcl"
```



Note:

At the prompt, enter database username and password.

- **Creating a JDBC resource:**

```
ohsh> create jdbc resource jdbc0 connectid=<database connection URL>
```



Note:

At the prompt, enter database username and password.

- **Showing resources:**

```
ohsh> show resources
```

This command lists default resources and any additional resources created within the session.

Creating Resources Using JDBC SSL to Connect Oracle Database

You need to create these resources in order to connect to Oracle Database using SSL.

- **Creating a SSL resource** . See [Using JDBC SSL](#) to download the SSL client wallet and complete the configuration steps.
- **Creating a SQL resource:**

```
ohsh> create sqlplus resource sql_ssl connectid="<tns entry name for the SSL connection>"
```

For example:

```
ohsh> create sqlplus resource sql_ssl connectid="inst1_ssl"
```

inst1_ssl is the TNS entry for the JDBC SSL connection



Note:

At the prompt, enter database username and password.

- **Creating a JDBC resource:**

```
ohsh> create jdbc resource jdbc_ssl  
connectiondir=<SSL wallet directory location>  
connectid="<tns entry name for the SSL connection>"
```

For example:

```
ohsh> create jdbc resource ora_mydatabase_cs
connectiondir=/home/oracle/ssl_client_wallet
connectid="inst1_ssl"
```

inst1_ssl is the TNS entry for the JDBC SSL connection

 **Note:**

At the prompt, enter database username and password.

Creating Resources Using Secure Java KeyStore

You can store passwords in the secure Java KeyStore. If you use the Java KeyStore, then you won't be prompted for the username and password. You can also add this to the scripts you develop to load data.

- **Creating a Java KeyStore.** See [Using Secure External Java KeyStore and Hadoop credential command](#)
- **Creating a SQL resource Using a Java KeyStore:**

```
ohsh> create sqlplus resource sql_ssl_cs user=<username>
passwordalias=<password alias>
provider="<provider_path>"
connectid="<tns entry name for the SSL connection>"
```

For example:

```
ohsh> create sqlplus resource sql_ssl_cs user=oracle
passwordalias=oracle_passwd
provider="jceks://file/home/oracle/passwd.jceks"
connectid="inst1"
```

- **Creating a SQL resource Using Java KeyStore with JDBC SSL connection:**

```
ohsh> create sqlplus resource sql_ssl_cs user=<username>
passwordalias=<password alias>
provider="<provider_path>"
connectiondir=<SSL wallet directory location>
connectid="<tns entry name for the SSL connection>"
```

For example:

```
ohsh> create sqlplus resource sql_ssl_cs user=oracle
passwordalias=oracle_passwd
provider="jceks://file/home/oracle/passwd.jceks"
connectiondir=/home/oracle/ssl_client_wallet
connectid="inst1_ssl"
```

- **Creating a JDBC resource Using a Java KeyStore:**

```
ohsh> create jdbc resource jdbc_ssl_cs user=<username>  
passwordalias=<password alias>  
provider="<provider_path>"  
connectid="<tns entry name for the SSL connection>"
```

For example:

```
ohsh> create jdbc resource jdbc_ssl_cs user=oracle  
passwordalias=oracle_passwd  
provider="jceks://file/home/oracle/passwd.jceks"  
connectid="inst1"
```

- **Creating a JDBC resource Using Java KeyStore with JDBC SSL connection:**

```
ohsh> create jdbc resource jdbc_ssl_cs user=<username>  
passwordalias=<password alias>  
provider="<provider_path>"  
connectiondir=<SSL wallet directory location>  
connectid="<tns entry name for the SSL connection>"
```

For example:

```
ohsh> create jdbc resource jdbc_ssl_cs user=oracle  
passwordalias=oracle_passwd  
provider="jceks://file/home/oracle/passwd.jceks"  
connectiondir=/home/oracle/ssl_client_wallet  
connectid="inst1_ssl"
```

Getting Help

The OSH shell provides online help for all commands.

To get a list of all OSH commands:

```
ohsh> help
```

To get help on a specific command, enter `help`, followed by the command:

```
ohsh> help show
```

The table below describes the help categories available.

Help Command	Description
help load	Describes load commands for Oracle and Hadoop tables.
help set	Shows help for setting defaults for load operations. It also describes what load methods are impacted by a particular setting.
help show	Shows help for inspecting default settings.

Help Command	Description
help shell	Shows shell-like commands.
help resource	Show commands for creating and dropping named resources.

Use Oracle SQL Developer With Oracle Big Data Connectors

Oracle SQL Developer is an essentially a graphical version SQL*Plus. Among its features is a user-friendly interface to tools for moving data between Hadoop and Oracle Database. It includes support for Copy to Hadoop, Oracle Loader for Hadoop, and Oracle SQL Connector for Hadoop Distributed File System (as well as Oracle Big Data SQL). There are wizards in the interface to assist with use of all of of these tools.

Follow these steps to set up Oracle SQL Developer to work with Oracle Big Data Connectors.

1. Download and install Oracle SQL Developer.
2. Download the Hive JDBC Drivers.
3. Add the new Hive JDBC Drivers to Oracle SQL Developer.
4. Set environment variables required for Oracle Big Data Connectors.
5. Set up the necessary connections.

After you have installed the drivers, configured the environment, and created connections between Oracle Database and Hadoop, you can start using Oracle Big Data Connectors from within Oracle SQL Developer.

Downloading and Installing Oracle SQL Developer

Install SQL Developer. Release 4.2 and later includes support for *Copy To Hadoop*.

The installation is simple. Just download the package and extract it.

1. Go to the [Oracle SQL Developer download site](#) on the Oracle Technology Network (OTN).
2. Accept the license agreement and download the version that is appropriate for your platform.
3. Extract the downloaded ZIP file to your local drive.

You can extract to any folder name.

See *Installing and Getting Started with SQL Developer* in the *Oracle SQL Developer User's Guide* for further installation and configuration details.

Downloading and Installing the Hive JDBC Drivers for Cloudera Enterprise

To connect Oracle SQL Developer to Hive in the Hadoop environment, you need to download and install the Hive JDBC drivers for Cloudera Enterprise. These drivers are not included in the Oracle SQL Developer download package.

 **Note for HDP Users:**

At this time, SQL Developer 4.2 requires the Cloudera JDBC drivers for Hive. However, these drivers appear to work against Hortonworks clusters as well. HDP users should test to determine if these drivers meet their needs.

1. Download the latest Cloudera JDBC drivers for Hive from the [Cloudera](#) website to any local directory.

You can search for “cloudera hive jdbc drivers download” on the Cloudera website to locate the available driver packages.

You are prompted to select the driver version, OS, and OS version (32/64 bit). At this time, the latest drive version is 2.5.18. You can choose the newest version available.

2. Unzip the archive:

```
unzip hive_jdbc_<version>.zip
```

3. View the extracted content. Notice that under the top-level folder there are multiple ZIP files. Each is for a different JDBC version. For this setup, only JDBC 4.0 is usable. Select the **JDBC4_** ZIP file (JDBC4_<version>.zip).

 **Important:**

Choose *only* the **JDBC4_** ZIP file, which contains the drivers for JDBC 4.0. This is the only compatible version. The drivers in other packages, such as JDBC41_*, are not compatible with SQL Developer 4.2 and will return errors upon connection attempts.

4. Unzip the JDBC4 archive to a target directory that is accessible to Oracle SQL Developer, for example, `./home/oracle/jdbc` :

```
# unzip Cloudera_HiveJDBC4_<version>.zip -d /home/oracle/jdbc/
```

The extracted content should be similar to this:

```
Cloudera_HiveJDBC4_2.5.18.1050\Cloudera-JDBC-Driver-for-Apache-Hive-Install-Guide.pdf
Cloudera_HiveJDBC4_2.5.18.1050\Cloudera-JDBC-Driver-for-Apache-Hive-Release-Notes.pdf
Cloudera_HiveJDBC4_2.5.18.1050\commons-codec-1.3.jar
Cloudera_HiveJDBC4_2.5.18.1050\commons-logging-1.1.1.jar
Cloudera_HiveJDBC4_2.5.18.1050\HiveJDBC4.jar
Cloudera_HiveJDBC4_2.5.18.1050\hive_metastore.jar
Cloudera_HiveJDBC4_2.5.18.1050\hive_service.jar
Cloudera_HiveJDBC4_2.5.18.1050\httpclient-4.1.3.jar
Cloudera_HiveJDBC4_2.5.18.1050\httpcore-4.1.3.jar
Cloudera_HiveJDBC4_2.5.18.1050\libfb303-0.9.0.jar
Cloudera_HiveJDBC4_2.5.18.1050\libthrift-0.9.0.jar
Cloudera_HiveJDBC4_2.5.18.1050\log4j-1.2.14.jar
```



```

Cloudera_HiveJDBC4_2.5.18.1050\out.txt
Cloudera_HiveJDBC4_2.5.18.1050\ql.jar
Cloudera_HiveJDBC4_2.5.18.1050\slf4j-api-1.5.11.jar
Cloudera_HiveJDBC4_2.5.18.1050\slf4j-log4j12-1.5.11.jar
Cloudera_HiveJDBC4_2.5.18.1050\TCLIServiceClient.jar
Cloudera_HiveJDBC4_2.5.18.1050\zookeeper-3.4.6.jar

```

Adding the new Hive JDBC Drivers to Oracle SQL Developer

Next, start up SQL Developer and copy all of the extracted driver files into “Third Party JDBC Drivers” in the **Preferences** window.

1. Navigate to the folder where you downloaded and extracted Oracle SQL Developer.
2. Click the `sqldeveloper` subfolder. Then, click `sqldeveloper.exe` in this folder.
3. In the SQL Developer menu bar, select **Tools>Preferences**.
4. In the file explorer of the **Preferences** window, expand **Database** and then click **Third Party JDBC Drivers**.
5. Click **Add Entry**.
6. Navigate to the folder where you sent the files extracted from `Cloudera_HiveJDBC4_<version>.zip`. Copy all of the JAR files from the ZIP extraction into this window and then click **OK**.
7. Restart Oracle SQL Developer.

Setting up Environment Variables for Using Oracle Big Data Connectors With Oracle SQL Developer

SQL Developer with Oracle Big Data Connectors requires the user to make an SSH connection from SQL Developer to a Hadoop client, Hadoop node, or Hadoop edge node. The home directory of this account requires a specific environment file for each of the Big Data Connectors it runs



Note:

If you want to do *staged copies* in Copy to Hadoop, then Copy to Hadoop requires an additional SSH connection to the `oracle` OS account on the database system.

You must create and populate the environment files. The following table provides the exact filenames and the content you must add to each file. The file must be readable by the account using the Big Data Connector.

Environment File Name	Contents
-----------------------	----------

For Copy to Hadoop:

```
#!/bin/bash
# Environment file for Copy to Hadoop
```

.sqldev_cp2hadoop_env

```
export CP2HADOOP_HOME=<Parent directory of the directory containing Copy to Hadoop JARs>
# On Oracle Big Data Appliance, the Copy to Hadoop JARs are in /opt/oracle/bigdatasql/bdcell-12.1
export HADOOP_CLASSPATH=${CP2HADOOP_HOME}/jlib/*
# -----
# If using Oracle Wallet, add the following four variables:
export WALLET_LOCATION=<Location of the Oracle Wallet files>
#For example: export WALLET_LOCATION=/home/${USER}/wallet
export TNS_ADMIN=<Like WALLET_LOCATION, this also points to the location of the Oracle Wallet files>
export CLUSTER_WALLET_LOCATION=${WALLET_LOCATION}
export CLUSTER_TNS_ADMIN=${TNS_ADMIN}
```

For Oracle Loader for Hadoop:

```
#!/bin/bash
# Environment file for Oracle Loader for Hadoop
```

.sqldev_olh_env

```
export HIVE_HOME=<For example: /opt/cloudera/parcels/CDH/lib/hive>
export HIVE_CONF_DIR=<For example: /etc/hive/conf>
export OLH_HOME=<For example (on Oracle Big Data Appliance): /opt/oracle/olh>
export HADOOP_CLASSPATH=${OLH_HOME}/jlib/*:${HIVE_CONF_DIR}:${HIVE_HOME}/lib/*
export OLH_LIB_JARS=${HIVE_HOME}/lib/hive-exec.jar,${HIVE_HOME}/lib/hive-metastore.jar,${HIVE_HOME}/lib/libfb303-0.9.2.jar
# -----
# If using Oracle Wallet, add the following four variables:
export WALLET_LOCATION=<Location of the Oracle Wallet files>
export TNS_ADMIN=<Same path as WALLET_LOCATION>
export CLUSTER_WALLET_LOCATION=${WALLET_LOCATION}
export CLUSTER_TNS_ADMIN=${TNS_ADMIN}
```

Environment File Name	Contents
For Oracle SQL Connector for HDFS:	#!/bin/bash # Environment file for Oracle SQL Connector for HDFS
.sqldev_osch_env	<pre> export HIVE_HOME=<For example: /opt/cloudera/parcels/CDH/lib/hive> export HIVE_CONF_DIR=<For example: /etc/hive/conf> export OSCH_HOME=<For example (on Oracle Big Data Appliance): /opt/oracle/osch> export HADOOP_CLASSPATH=\${OSCH_HOME}/jlib/*:\${HIVE_CONF_DIR}:\${HIVE_HOME}/lib/* # ----- # If using Oracle Wallet, add the following four variables: export WALLET_LOCATION=<Location of the Oracle Wallet files> export TNS_ADMIN=<Same path as WALLET_LOCATION> export CLUSTER_WALLET_LOCATION=\${WALLET_LOCATION} export CLUSTER_TNS_ADMIN=\${TNS_ADMIN} </pre>

Setting Up Secure Connections for Oracle Big Data Connectors

See [Apache Hadoop Connectors Support in SQL Developer in the *Oracle SQL Developer User's Guide*](#) for instructions on how to create SSH connections required for Oracle Big Data Connectors access to Hadoop.

Part III

Oracle XQuery for Apache Hadoop

This part contains the following chapters:

- [Using Oracle XQuery for Apache Hadoop](#)
- [Oracle XQuery for Apache Hadoop Reference](#)
- [Oracle XML Extensions for Hive](#)

5

Using Oracle XQuery for Apache Hadoop

This chapter explains how to use Oracle XQuery for Apache Hadoop (Oracle XQuery for Hadoop) to extract and transform large volumes of semistructured data. It contains the following sections:

- [What Is Oracle XQuery for Hadoop?](#)
- [Get Started With Oracle XQuery for Hadoop](#)
- [About the Oracle XQuery for Hadoop Functions](#)
- [Create an XQuery Transformation](#)
- [Run Queries](#)
- [Run Queries from Apache Oozie](#)
- [Oracle XQuery for Hadoop Configuration Properties](#)

What Is Oracle XQuery for Hadoop?

Oracle XQuery for Hadoop is a transformation engine for semistructured 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 an 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, such as 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.

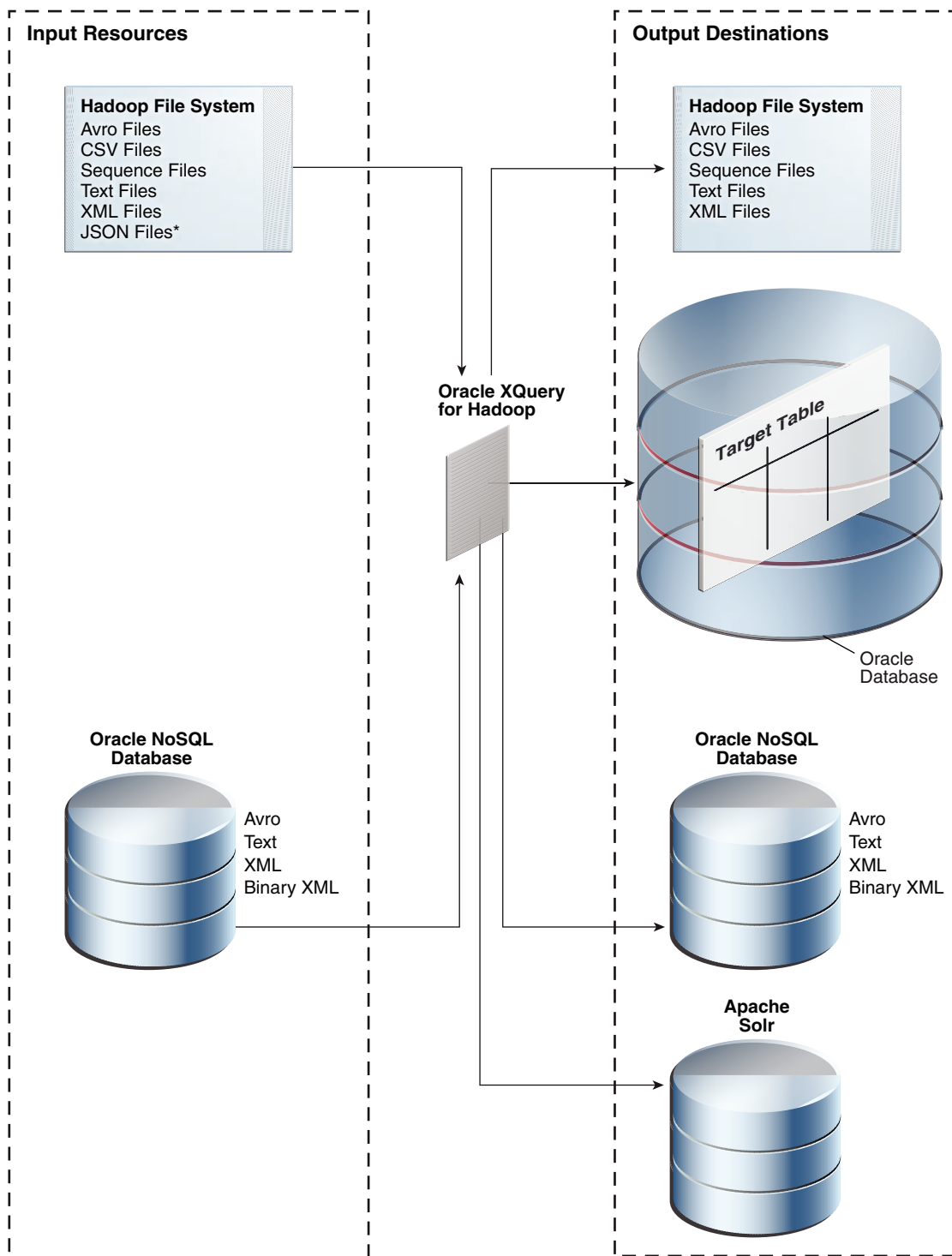
Oracle XQuery for Hadoop also provides extensions to Apache Hive to support massive XML files.

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, which enables Oracle XQuery for Hadoop to:

- Load data efficiently into Oracle Database using Oracle Loader for Hadoop.
- Provide read and write support to Oracle NoSQL Database.

The following figure provides an overview of the data flow using Oracle XQuery for Hadoop.

Figure 5-1 Oracle XQuery for Hadoop Data Flow



* Parallel processing of a single JSON file is not supported. See the JSON File Adapter.

Get 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:

- [W3schools XQuery Tutorial](#)
- [XQuery 3.1: An XML Query Language](#)

Basic Steps

Perform the following basic steps when using Oracle XQuery for Apache Hadoop:

1. The first time you use Oracle XQuery for Apache Hadoop, ensure that the software is installed and configured.
See "[Oracle XQuery for Hadoop Setup](#)."
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 Apache Hadoop functions. It can use various adapters for input and output.
See "[About the Oracle XQuery for Hadoop Functions](#)" and "[Create an XQuery Transformation](#)."
4. Execute the XQuery transformation.
See "[Run Queries](#)."

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
13/11/21 02:41:57 INFO hadoop.xquery: OXH: Oracle XQuery for Hadoop 4.2.0
((build 4.2.0-cdh5.0.0-mr1 @mr2). Copyright (c) 2014, Oracle. All rights
reserved.
13/11/21 02:42:01 INFO hadoop.xquery: Submitting map-reduce job "oxh:hello.xq#0"
id="3593921f-c50c-4bb8-88c0-6b63b439572b.0", inputs=[hdfs://
bigdatalite.localdomain:8020/user/oracle/hello.txt], output=myout
.
.
.
```

5. Check the output file:

```
$ hdfs dfs -cat ./myout/part-m-00000
Hello World!
```

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.

Other commonly used functions are also included in Oracle XQuery for Hadoop.

About the Adapters

Following are brief descriptions of the Oracle XQuery for Hadoop adapters.

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.

JSON File Adapter

The JSON file adapter provides access to JSON files stored in HDFS. It contains a collection function for reading JSON files, and a group of helper functions for parsing JSON data directly. You must use another adapter to write the output.

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 "[Software Requirements](#)" for the supported versions of Oracle Database.

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 Table, Avro, XML, binary XML, or text. This adapter includes collection, get, and put functions.

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.

Solr Adapter

The Solr adapter provides functions to create full-text indexes and load them into Apache Solr servers.

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 file adapter extends the support for JSON objects stored in text files.

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.

Related Topics

- [Avro File Adapter](#)
- [JSON File Adapter](#)
- [Oracle Database Adapter](#)
- [Oracle NoSQL Database Adapter](#)
- [Sequence File Adapter](#)
- [Solr Adapter](#)
- [Text File Adapter](#)
- [XML File Adapter](#)

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.

Hadoop Functions

The Hadoop module is a group of functions that are specific to Hadoop.

Duration, Date, and Time Functions

This group of functions parse duration, date, and time values.

String-Processing Functions

These functions add and remove white space that surrounds data values.

Related Topics

- [About XQuery Language Support](#)
- [Hadoop Module](#)
- [Oracle XQuery Functions for Duration, Date, and Time](#)
You can manipulate durations, dates, and times in XQuery using Oracle XQuery functions.
- [Oracle XQuery Functions for Strings](#)
You can manipulate strings in XQuery using Oracle XQuery functions.

Create 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](#)
- [Call Custom Java Functions from XQuery](#)
- [Access 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 XQuery FLWOR expression "For, Let, Where, Order by, Return" expression.

- 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 [Oracle XQuery for Apache Hadoop Reference](#) for the `collection` functions.
- Each top-level FLWOR expression can have optional `let`, `where`, and `group by` clauses. Other types of clauses are invalid, such as `order by`, `count`, and `window` clauses.
- Each top-level FLWOR expression must return one or more results from calling an Oracle XQuery for Hadoop `put` function. See [Oracle XQuery for Apache Hadoop Reference](#) 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:

- "FLWOR Expressions" in [XQuery 3.1: An XML Query Language](#)
- For a description of updating expressions, "Extensions to XQuery 1.0" in [W3C XQuery Update Facility 1.0](#)

About XQuery Language Support

Oracle XQuery for Hadoop supports W3C XQuery 3.1, except for the following:

- FLWOR window clause
- FLWOR count clause
- namespace constructors
- `fn:parse-ietf-date`
- `fn:transform`
- higher order XQuery functions

For the language, see W3C [XQuery 3.1: An XML Query Language](#) .

For the functions, see W3C [XPath and XQuery Functions and Operators](#) .

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 5-7](#).

Call Custom Java Functions from XQuery

Oracle XQuery for Hadoop is extensible with custom external functions implemented in the Java language. A Java implementation must be a static method with the parameter and return types as defined by the *XQuery API for Java (XQJ)* specification.

A 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 5-8](#) 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>

Access 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` or `import schema` statement.
- Specify the library module or XML schema file in the `-files` command line option when calling Oracle XQuery for Hadoop.

For an example of using user-defined XQuery library modules and XML schemas, see [Example 5-9](#).

 **See Also:**

"Location URIs" in *XQuery 3.1: An XML Query Language*

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 5-1 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 5-2 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 5-3 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 colons (:).

```
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 5-4 Left Outer Joins

This example is similar to the second query in [Example 5-3](#), 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 5-5 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 5-6 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 5-7 Accessing Auxiliary Input Data

The next query is an alternative version of the second query in [Example 5-3](#), 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."](#)

```
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 5-8 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](#).

Example 5-9 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
```

Example 5-10 Filtering Dirty Data Using a Try/Catch Expression

The XQuery try/catch expression can be used to broadly handle cases where input data is in an unexpected form, corrupted, or missing. The next query finds reads an input file, ages.txt, that contains a username followed by the user's age.

```

USER      AGE
-----
john      45
kelly
laura     36
phil      OLD!

```

Notice that the first two lines of this file contain header text and that the entries for Kelly and Phil have missing and dirty age values. For each user in this file, the query writes out the user name and whether the user is over 40 or not.

```

import module "oxh:text";

for $line in text:collection("ages.txt")
let $split := fn:tokenize($line, "\s+")
return
  try {
    let $user := $split[1]
    let $age := $split[2] cast as xs:integer
    return
      if ($age gt 40) then
        text:put($user || " is over 40")
      else
        text:put($user || " is not over 40")
  } catch * {
    text:trace($err:code || " : " || $line)
  }

```

The query generates an output text file that contains the following lines:

```

john is over 40
laura is not over 40

```

The query also generates a trace* file that contains the following lines:

```

err:FORG0001 : USER      AGE
err:XPTY0004 : -----
err:XPTY0004 : kelly
err:FORG0001 : phil      OLD!

```

Run Queries

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] [-sharelib hdfs_dir][-skiperrors] [-version]
```

Oracle XQuery for Hadoop Options

query.xq

Identifies the XQuery file. See ["Create an XQuery Transformation."](#)

-clean

Deletes all files from the output directory before running the query. If you use the default directory, Oracle XQuery for Hadoop always cleans the directory, even when this option is omitted.

-exportliboozie directory

Copies Oracle XQuery for Hadoop dependencies to the specified directory. Use this option to add Oracle XQuery for Hadoop to the Hadoop distributed cache and the Oozie shared library. External dependencies are also copied, so ensure that environment variables such as `KVHOME`, `OLH_HOME`, and `OXH_SOLR_MR_HOME` are set for use by the related adapters (Oracle NoSQL Database, Oracle Database, and Solr).

-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. The default output directory is `/tmp/oxh-user_name/output`.

See ["About the Oracle XQuery for Hadoop Functions"](#) 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.

-sharelib hdfs_dir

Specifies the HDFS folder location containing Oracle XQuery for Hadoop and third-party libraries.

-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 these configuration properties:

[oracle.hadoop.xquery.skiperrors.counters](#)

[oracle.hadoop.xquery.skiperrors.max](#)

[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. Oracle XQuery for Hadoop 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](#)."

When you work with the Oracle Database or Oracle NoSQL Database adapters, you can set various job properties in this file. See "[Oracle Loader for Hadoop Configuration Properties and Corresponding %oracle-property Annotations](#)" and "[Oracle NoSQL Database Adapter Configuration Properties](#)".

-D *property=value*

Identifies a configuration property. See "[Oracle XQuery for Hadoop Configuration Properties](#)."

-files

Specifies a comma-delimited list of files that are added to the distributed cache. See "[Accessing Data in the Hadoop Distributed Cache](#)."



See Also:

For full descriptions of the generic options, go to

http://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-common/CommandsManual.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!](#)" 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!
```

Run Queries from Apache Oozie

Apache Oozie is a workflow tool that enables you to run multiple MapReduce jobs in a specified order and, optionally, at a scheduled time. Oracle XQuery for Hadoop provides an Oozie action node that you can use to run Oracle XQuery for Hadoop queries from an Oozie workflow.

Use Oozie with Oracle XQuery for Hadoop Action

Follow these steps to execute your queries in an Oozie workflow:

1. The first time you use Oozie with Oracle XQuery for Hadoop, ensure that Oozie is configured correctly. See "[Configure Oozie for the Oracle XQuery for Hadoop Action](#)".
2. Develop your queries in Oracle XQuery for Hadoop the same as always.
3. Create a workflow XML file like the one shown in [Example 5-11](#). You can use the XML elements listed in "[Supported XML Elements](#)".
4. Set the Oozie job parameters. The following parameter is required:

```
oozie.use.system.libpath=true
```

See [Example 5-13](#).

5. Run the job using syntax like the following:

```
oozie job -name http://example.com:11000/oozie -config filename -run
```

See Also:

"Oozie Command Line Usage" in the *Apache Oozie Command Line Interface Utilities* at

https://oozie.apache.org/docs/4.0.0/DG_CommandLineTool.html#Oozie_Command_Line_Usage

Supported XML Elements

The Oracle XQuery for Hadoop action extends Oozie's Java action. It supports the following optional child XML elements with the same syntax and semantics as the Java action:

- archive
- configuration
- file
- job-tracker
- job-xml
- name-node
- prepare

 **See Also:**

The Java action description in the Oozie Specification at

https://oozie.apache.org/docs/4.0.0/WorkflowFunctionalSpec.html#a3.2.7_Java_Action

In addition, the Oracle XQuery for Hadoop action supports the following elements:

- `script`: The location of the Oracle XQuery for Hadoop query file. Required.
The query file must be in the workflow application directory. A relative path is resolved against the application directory.
Example: `<script>myquery.xq</script>`
- `output`: The output directory of the query. Required.
The output element has an optional `clean` attribute. Set this attribute to `true` to delete the output directory before the query is run. If the output directory already exists and the `clean` attribute is either not set or set to `false`, an error occurs. The output directory cannot exist when the job runs.
Example: `<output clean="true">/user/jdoe/myoutput</output>`

Any error raised while running the query causes Oozie to perform the error transition for the action.

Example: Hello World

This example uses the following files:

- `workflow.xml`: Describes an Oozie action that sets two configuration values for the query in `hello.xq`: an HDFS file and the string `World!`
The HDFS input file is `/user/jdoe/data/hello.txt` and contains this string:
`Hello`
See [Example 5-11](#).
- `hello.xq`: Runs a query using Oracle XQuery for Hadoop.
See [Example 5-12](#).
- `job.properties`: Lists the job properties for Oozie. See [Example 5-13](#).

To run the example, use this command:

```
oozie job -oozie http://example.com:11000/oozie -config job.properties -run
```

After the job runs, the `/user/jdoe/myoutput` output directory contains a file with the text "Hello World!"

Example 5-11 The workflow.xml File for Hello World

This file is named `/user/jdoe/hello-oozie-oxh/workflow.xml`. It uses variables that are defined in the `job.properties` file.

```

<workflow-app xmlns="uri:oozie:workflow:0.4" name="oxh-helloworld-wf">
  <start to="hello-node"/>
  <action name="hello-node">
    <oxh xmlns="oxh:oozie-action:v1">
      <job-tracker>${jobTracker}</job-tracker>
      <name-node>${nameNode}</name-node>

      <!--
        The configuration can be used to parameterize the query.
      -->
      <configuration>
        <property>
          <name>myinput</name>
          <value>${nameNode}/user/jdoe/data/src.txt</value>
        </property>
        <property>
          <name>mysuffix</name>
          <value> World!</value>
        </property>
      </configuration>

      <script>hello.xq</script>

      <output clean="true">${nameNode}/user/jdoe/myoutput</output>

    </oxh>
    <ok to="end"/>
    <error to="fail"/>
  </action>
  <kill name="fail">
    <message>OXH failed: [${wf:errorMessage(wf:lastErrorNode())}]</message>
  </kill>
  <end name="end"/>
</workflow-app>

```

Example 5-12 The hello.xq File for Hello World

This file is named `/user/jdoe/hello-oozie-oxh/hello.xq`.

```

import module "oxh:text";

declare variable $input := oxh:property("myinput");
declare variable $suffix := oxh:property("mysuffix");

for $line in text:collection($input)
return
  text:put($line || $suffix)

```

Example 5-13 The job.properties File for Hello World

```

oozie.wf.application.path=hdfs://example.com:8020/user/jdoe/hello-oozie-oxh
nameNode=hdfs://example.com:8020
jobTracker=hdfs://example.com:8032
oozie.use.system.libpath=true

```

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 "[Run Queries](#)."



See Also:

Hadoop documentation for job configuration files at

<http://wiki.apache.org/hadoop/JobConfFile>

Property	Description																		
oracle.hadoop.xquery.lib.share	<p>Type: String</p> <p>Default Value: Not defined.</p> <p>Description: Identifies an HDFS directory that contains the libraries for Oracle XQuery for Hadoop and third-party software. For example:</p> <pre>http://path/to/shared/folder</pre> <p>All HDFS files must be in the same directory. Alternatively, use the <code>-sharelib</code> option on the command line.</p> <p>Pattern Matching: You can use pattern matching characters in a directory name. If multiple directories match the pattern, then the directory with the most recent modification timestamp is used.</p> <p>To specify a directory name, use alphanumeric characters and, optionally, any of the following special, pattern matching characters:</p> <table border="1"> <thead> <tr> <th>Pattern</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>?</td> <td>Matches any one character.</td> </tr> <tr> <td>*</td> <td>Matches zero or more characters.</td> </tr> <tr> <td>[abc]</td> <td>Matches one character from character set {a,b,c}.</td> </tr> <tr> <td>[a-b]</td> <td>Matches one character from the character range from a to b. Character a must be less than or equal to character b.</td> </tr> <tr> <td>[^a]</td> <td>Matches one character that is not from the a character set or range. The carat (^) must follow the opening bracket immediately (no spaces).</td> </tr> <tr> <td>\c</td> <td>Removes (escapes) any special meaning of character c.</td> </tr> <tr> <td>{ab,cd}</td> <td>Matches a string from the string set {ab, cd}.</td> </tr> <tr> <td>{ab,c{de,fh}}</td> <td>Matches a string from the string set {ab, cde, cfh}.</td> </tr> </tbody> </table> <p>Oozie libraries: The value <code>oxh:oozie</code> expands automatically to <code>/user/{oozie,user}/share/lib/{oxh,*/oxh*}</code>, which is a common search path for supported Oozie versions. The <i>user</i> is the current user name. However, the Oracle XQuery for Hadoop Oozie action ignores this setting when running queries, because all libraries are preinstalled in HDFS.</p>	Pattern	Description	?	Matches any one character.	*	Matches zero or more characters.	[abc]	Matches one character from character set {a,b,c}.	[a-b]	Matches one character from the character range from a to b. Character a must be less than or equal to character b.	[^a]	Matches one character that is not from the a character set or range. The carat (^) must follow the opening bracket immediately (no spaces).	\c	Removes (escapes) any special meaning of character c.	{ab,cd}	Matches a string from the string set {ab, cd}.	{ab,c{de,fh}}	Matches a string from the string set {ab, cde, cfh}.
Pattern	Description																		
?	Matches any one character.																		
*	Matches zero or more characters.																		
[abc]	Matches one character from character set {a,b,c}.																		
[a-b]	Matches one character from the character range from a to b. Character a must be less than or equal to character b.																		
[^a]	Matches one character that is not from the a character set or range. The carat (^) must follow the opening bracket immediately (no spaces).																		
\c	Removes (escapes) any special meaning of character c.																		
{ab,cd}	Matches a string from the string set {ab, cd}.																		
{ab,c{de,fh}}	Matches a string from the string set {ab, cde, cfh}.																		
oracle.hadoop.xquery.output	<p>Type: String</p> <p>Default Value: <code>/tmp/oxh-user_name/output</code>. The <i>user_name</i> is the name of the user running Oracle XQuery for Hadoop.</p> <p>Description: Sets the output directory for the query. This property is equivalent to the <code>-output</code> command line option. See "Oracle XQuery for Hadoop Options."</p>																		

Property	Description
<code>oracle.hadoop.xquery.scratch</code>	<p>Type: String</p> <p>Default Value: <code>/tmp/oxh-user_name/scratch</code>. The <code>user_name</code> is the name of the user running Oracle XQuery for Hadoop.</p> <p>Description: Sets the HDFS temp directory for Oracle XQuery for Hadoop to store temporary files.</p>
<code>oracle.hadoop.xquery.timezone</code>	<p>Type: String</p> <p>Default Value: Client system time zone</p> <p>Description: The XQuery implicit time zone, which is used in a comparison or arithmetic operation when a date, time, or datetime value does not have a time zone. The value must be in the format described by the Java <code>TimeZone</code> class. See the <code>TimeZone</code> class description in <i>Java 7 API Specification</i> at http://docs.oracle.com/javase/7/docs/api/java/util/TimeZone.html</p>
<code>oracle.hadoop.xquery.skiperrors</code>	<p>Type: Boolean</p> <p>Default Value: <code>false</code></p> <p>Description: Set to <code>true</code> to turn on error recovery, or set to <code>false</code> to stop processing when an error occurs. This property is equivalent to the <code>-skiperrors</code> command line option.</p>
<code>oracle.hadoop.xquery.skiperrors.counters</code>	<p>Type: Boolean</p> <p>Default Value: <code>true</code></p> <p>Description: Set to <code>true</code> to group errors by error code, or set to <code>false</code> to report all errors in a single counter.</p>
<code>oracle.hadoop.xquery.skiperrors.max</code>	<p>Type: Integer</p> <p>Default Value: Unlimited</p> <p>Description: Sets the maximum number of errors that a single MapReduce task can recover from.</p>
<code>oracle.hadoop.xquery.skiperrors.log.max</code>	<p>Type: Integer</p> <p>Default Value: 20</p> <p>Description: Sets the maximum number of errors that a single MapReduce task logs.</p>
<code>log4j.logger.oracle.hadoop.xquery</code>	<p>Type: String</p> <p>Default Value: Not defined</p> <p>Description: Configures the <code>log4j</code> logger for each task with the specified threshold level. Set the property to one of these values: <code>OFF</code>, <code>FATAL</code>, <code>ERROR</code>, <code>WARN</code>, <code>INFO</code>, <code>DEBUG</code>, or <code>ALL</code>. If this property is not set, then Oracle XQuery for Hadoop does not configure <code>log4j</code>.</p>

6

Oracle XQuery for Apache Hadoop Reference

This chapter describes the adapters available in Oracle XQuery for Apache Hadoop:

- [Avro File Adapter](#)
- [JSON File Adapter](#)
- [Oracle Database Adapter](#)
- [Oracle NoSQL Database Adapter](#)
- [Sequence File Adapter](#)
- [Solr Adapter](#)
- [Text File Adapter](#)
- [Tika File Adapter](#)
- [XML File Adapter](#)
- [Serialization Annotations](#)

This chapter also describes several other library modules:

- [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](#)
- [Examples of Avro File Adapter Functions](#)
- [About Converting Values Between Avro and XML](#)

 **Note:****Additional Configuration Steps for HDP Users**

Oracle XQuery for Hadoop has been verified to run on both Cloudera's CDH5 and Hortonwork's HDP 2.3.3. However, to run queries that write to Avro container files in HDP 2.3.2, you must change the OXH classpath definition to use avro-mapred-1.7.4-hadoop2.jar.

1. Download the JAR from the Apache archive <https://archive.apache.org/dist/avro/avro-1.7.4/java/>
2. In \$OXH_HOME/lib/oxh-lib.xml locate the following path tag:

```
<path id="oracle.hadoop.xquery.avro.lib">
  <fileset dir="{oracle.hadoop.loader.olh_home}/jlib"
erroronmissingdir="false">
    <include name="avro-mapred*.jar"/>
  </fileset>
</path>
```

Replace the path tag above with the following revision. [DIRECTORY] in this example is a placeholder. Replace it with the directory path to the JAR.

```
<path id="oracle.hadoop.xquery.avro.lib">
  <fileset dir="[DIRECTORY]">
    <include name="avro-mapred-1.7.4-hadoop2.jar"/>
  </fileset>
</path>
```

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 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."](#)

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 you omit the \$map parameter, 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](#)."

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](#)."

%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 it is 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](#)."

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.](#)"

%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](#)."

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

These examples use the following text file in HDFS:

```
mydata/ages.txt
```

```
john,45
kelly,36
laura,
mike,27
```

Example 6-1 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 6-2 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 6-1](#) 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 enables XQuery to query them. 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">Example Inc</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 Example Inc. 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 $person/properties/oxh:entry[@key eq "employer"] eq "Example Inc"
return
  text:put($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 $person/properties/avro:get("employer") eq "Example Inc"
return
  text:put($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

The following table 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/fn: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">Example Inc</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, then 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 of 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>
```

JSON File Adapter

The JSON file adapter provides access to JSON files stored in HDFS. It also contains functions for working with JSON data 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 a single JSON file in parallel is not currently supported. A set of JSON files can be processed in parallel, with sequential processing of each file.

The JSON module is described in the following topics:

- [Built-in Functions for Reading JSON](#)
- [Custom Functions for Reading JSON Files](#)
- [Examples of JSON Functions](#)
- [JSON File Adapter Configuration Properties](#)
- [About Converting JSON Data Formats to XML](#)

Built-in Functions for Reading JSON

To use the built-in functions in your query, you must import the JSON file adapter as follows:

```
import module "oxh:json";
```

The JSON module contains the following functions:

- [json:collection-jsonxml](#)
- [json:parse-as-xml](#)

- [json:get](#)

As of Big Data Connectors Release 4.9, Oracle XQuery for Hadoop also supports XQuery 3.1 including the standard facilities for processing JSON, including: `fn:parse-json`, `fn:json-to-xml`, and `fn:xml-to-json`

**See Also:**

[XPath and XQuery Functions and Operators 3.1](#)

json:collection-jsonxml

Accesses a collection of JSON files in HDFS. Multiple files can be processed concurrently, but each individual file is parsed by a single task.

The JSON file adapter automatically decompresses files compressed with a Hadoop-supported compression codec.

Signature

```
json:collection-jsonxml($uris as xs:string*) as element()* external;
```

Parameters

`$uris`: The JSON file URIs

Returns

XML elements that model the JSON values. See "[About Converting JSON Data Formats to XML](#)."

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. See "[About Converting JSON Data Formats to XML](#)."

json:get

Retrieves an entry from a JSON object modeled as XML. See "[About Converting JSON Data Formats to XML](#)."

Signature

```
json:get($key as xs:string?, $obj as node()) as element(oxh:entry)?
```

```
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 `$input` not present, then the behavior is identical to calling the two-argument function using the context item for `$obj`. See the Notes.

Notes

These function calls 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](#)."

Custom Functions for Reading JSON Files

You can use the following annotations to define functions that read collections of JSON files in HDFS. These annotations provide additional functionality that is not available using the built-in functions.

Signature

Custom functions for reading JSON files must have the following signature:

```
declare %json:collection("jsonxml") [additional annotations]  
    function local:myFunctionName($uris as xs:string*) as element()* external;
```

Annotations

%json:collection("jsonxml")

Declares the collection function. The annotation parameter must be `jsonxml`.

%output:encoding("charset")

Identifies the text encoding of the input files.

The valid encodings are those supported by the JVM. If this annotation is omitted, then the encoding is automatically detected from the JSON file as UTF-8, UTF-16 big-endian serialization (BE) or little-endian serialization (LE), or UTF-32 (BE or LE).

For better performance, omit the encoding annotation if the actual file encoding is specified by JSON Request for Comment 4627, Section 3 "Encoding," on the Internet Engineering Task Force (IETF) website at <http://www.ietf.org/rfc/rfc4627.txt>

Parameters

\$uris as xs:string*

Lists the JSON file URIs. Required.

Returns

A collection of XML elements. Each element models the corresponding JSON value. See "[About Converting JSON Data Formats to XML.](#)"

Examples of JSON Functions

Example 6-3

This example uses the following JSON text files stored in HDFS:

```
mydata/users1.json
[
  { "user" : "john", "full name" : "John Doe", "age" : 45 },
  { "user" : "kelly", "full name" : "Kelly Johnson", "age" : 32 }
]
```

```
mydata/users2.json
[
  { "user" : "laura", "full name" : "Laura Smith", "age" : null },
  { "user" : "phil", "full name" : "Phil Johnson", "age" : 27 }
]
```

The following query selects names of users whose last name is Johnson from users1.json and users2.json

```
import module "oxh:text";
import module "oxh:json";

for $user in json:collection-jsonxml("mydata/users*.json")/oxh:item
let $fullname := $user/json:get("full name")
where tokenize($fullname, "\s+")[2] eq "Johnson"
return
  text:put-text($fullname)
```

This query generates text files that contain the following lines:

```
Phil Johnson
Kelly Johnson
```

The remaining 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 6-4

The following query selects the names of users that are older than 30 from `users-json.txt`:

```
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 $user/json:get("age") gt 30
return
  text:put($user/json:get("full name"))
```

This query generates text files that contain the following lines:

```
John Doe
Kelly Johnson
```

Example 6-5

The next query selects the names of employees that have a null value for age from `users-json.txt`:

```
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 $user/json:get("age")/nilled()
return
  text:put($user/json:get("full name"))
```

This query generates a text file that contains the following line:

```
Laura Smith
```

JSON File 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.

The following configuration properties are equivalent to the Jackson parser options with the same names. You can enter the option name in either upper or lower case. For example,

```
oracle.hadoop.xquery.json.parser.ALLOW_BACKSLASH_ESCAPING_ANY_CHARACTER
and
oracle.hadoop.xquery.json.parser.allow_backslash_escaping_any_character
```

are equal.

oracle.hadoop.xquery.json.parser.ALLOW_BACKSLASH_ESCAPING_ANY_CHARACTER

Type: Boolean

Default Value: `false`

Description: Enables any character to be escaped with a backslash (`\`). Otherwise, only the following characters can be escaped: quotation mark (`"`), slash (`/`), backslash

(\), backspace, form feed (f), new line (n), carriage return (r), horizontal tab (t), and hexadecimal representations (`unnnn`)

oracle.hadoop.xquery.json.parser.ALLOW_COMMENTS

Type: Boolean

Default Value: `false`

Description: Allows Java and C++ comments (`/*` and `//`) within the parsed text.

oracle.hadoop.xquery.json.parser.ALLOW_NON_NUMERIC_NUMBERS

Type: Boolean

Default Value: `false`

Description: Allows Not a Number (NaN) tokens to be parsed as floating number values.

oracle.hadoop.xquery.json.parser.ALLOW_NUMERIC_LEADING_ZEROS

Type: Boolean

Default Value: `false`

Description: Allows integral numbers to start with zeroes, such as 00001. The zeros do not change the value and can be ignored.

oracle.hadoop.xquery.json.parser.ALLOW_SINGLE_QUOTES

Type: Boolean

Default Value: `false`

Description: Allow single quotes (`'`) to delimit string values.

oracle.hadoop.xquery.json.parser.ALLOW_UNQUOTED_CONTROL_CHARS

Type: Boolean

Default Value: `false`

Description: Allows JSON strings to contain unquoted control characters (that is, ASCII characters with a decimal value less than 32, including the tab and line feed).

oracle.hadoop.xquery.json.parser.ALLOW_UNQUOTED_FIELD_NAMES

Type: Boolean

Default Value: `false`

Description: Allows unquoted field names, which are allowed by Javascript but not the JSON specification.

Related Topics

- [Run Queries](#)

About Converting JSON Data Formats to XML

This section describes how JSON data formats are converted to XML. It contains the following topics:

- [About Converting JSON Objects to XML](#)
- [About Converting JSON Arrays to XML](#)
- [About Converting Other JSON Types](#)

As of Big Data Connectors Release 4.9, Oracle XQuery for Hadoop also supports XQuery 3.1 including the standard facilities for processing JSON, including: `fn:parse-json`, `fn:json-to-xml`, and `fn:xml-to-json`

**See Also:**[XPath and XQuery Functions and Operators 3.1](#)

About Converting JSON Objects to XML

JSON objects are similar to Avro maps and are converted to the same XML structure. See "[Reading Maps](#)."

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](#)."

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 shown in the following table.

Table 6-2 JSON Type Conversions

JSON	XML
null	An empty (nilled) element
true/false	xs:boolean
number	xs:decimal
string	xs:string

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 Step 3 of "[Install Oracle XQuery for Hadoop](#)." 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](#)
- [Examples of Oracle Database Adapter Functions](#)
- [Oracle Loader for Hadoop Configuration Properties and Corresponding %oracle-property Annotations](#)



See Also:

"[Software Requirements](#)" for the versions of Oracle Database that Oracle Loader for Hadoop supports

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](#)."
- `oci`: Writes to an Oracle database table using an Oracle Call Interface (OCI) connection.
See "[Oracle OCI Direct Path Output Format](#)."
- `datapump`: Creates Data Pump files and associated scripts in HDFS for subsequent loading by another utility.
See "[Oracle Data Pump Output Format](#)."
- `text`: Creates delimited text files and associated scripts in HDFS.
See "[Delimited Text Output Format](#)."

For Oracle XQuery for Hadoop 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](#)."

%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](#)." Optional.

This annotation enables 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. 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. "[Oracle Loader for Hadoop Configuration Properties and Corresponding %oracle-property Annotations](#)" explains this relationship in detail.

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 the following table. 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.

The following table 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 6-3 Data Type Mappings Between Oracle Database and XQuery

Database Type	XQuery Type
VARCHAR2	<code>xs:string</code> Limited by the VARCHAR2 maximum size of 4000 bytes.
CHAR	<code>xs:string</code> Limited by the CHAR maximum size of 2000 bytes.
NVARCHAR2	<code>xs:string</code> Limited by the NVARCHAR2 maximum size of 4000 bytes.
NCHAR	<code>xs:string</code> Limited by the NCHAR maximum size of 2000 bytes.
DATE	<code>xs:dateTime</code> Limited to the range of January 1, 4712 BC, to December 31, 9999 CE. If a time zone is specified in the <code>xs:dateTime</code> value, then the time zone information is dropped. Fractional seconds are also dropped. A time value of 24:00:00 is not valid.
TIMESTAMP	<code>xs:dateTime</code> Limited to the range of January 1, 4712 BC, to December 31, 9999 CE. If a time zone is specified in the <code>xs:dateTime</code> 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	<code>xs:dateTime</code> 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 ."

Table 6-3 (Cont.) Data Type Mappings Between Oracle Database and XQuery

Database Type	XQuery Type
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. "
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 Loader for Hadoop Configuration Properties and Corresponding %oracle-property Annotations](#)."

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](#)" and "[Oracle Data Pump Output Format](#)."

Examples of Oracle Database Adapter Functions

These examples use the 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 use 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
```

This SQL command creates the VISITS table in the Oracle database:

```
CREATE TABLE visits (time TIMESTAMP, name VARCHAR2(15), page VARCHAR2(15), code
NUMBER)
```

Example 6-6 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. For anonymous access, the user name is missing, therefore 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.

The example uses a clear-text user name and password, which is insecure but acceptable in a development environment. Oracle recommends that you use a wallet instead for security, especially in a production application. 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 "[Use Oracle Wallets](#)."

```
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("mydata/*visits*.log")
let $split := fn:tokenize($line, "\s*,\s*")
return
  if (count($split) > 3) then
    local:myPut($split[1], $split[2], $split[3], $split[4])
  else
    local:myPut($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 6-7 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("mydata/*visits*.log")
let $split := fn:tokenize($line, "\s*,\s*")
where $split[2] eq 'john'
return
  local:myPut($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 Loader for Hadoop Configuration Properties and Corresponding %oracle-property Annotations

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, which are listed in the last column of the table below.

You can specify these properties with the generic `-conf` and `-D hadoop` command-line options in Oracle XQuery for Hadoop. Properties specified using this method apply to all Oracle Database adapter put functions in your query. See "[Run Queries](#)" and especially "[Generic Options](#)" for more information about the `hadoop` command-line options.

Alternatively, you can specify these properties as Oracle Database adapter put function annotations with the `%oracle-property` prefix. These annotations are listed in the second column of the table below. Annotations apply only to the particular Oracle Database adapter put function that contains them in its declaration.

For example, you can set the target table to `VISITS` by adding the following lines to the configuration file, and identifying the configuration file with the `-conf` option:

```
<property>
  <name>oracle.hadoop.loader.targetTable</name>
  <value>visits</value>
</property>
```

You can also set the target table to `VISITS` with the `-D` option, using the same Oracle Loader for Hadoop property:

```
-D oracle.hadoop.loader.targetTable=visits
```

Both methods set the target table to `VISITS` for all Oracle Database adapter put functions in your query.

Alternatively, 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')
```

This flexibility is provided for convenience. For example, if a query has multiple Oracle Database adapter put functions, each writing to a different table in the same database, then the most convenient way to specify the necessary information is like this:

- Use the [oracle.hadoop.loader.connection.url](#) property in the configuration file to specify the database connection URL. Then identify the configuration file using the `-conf` option. This option sets the same database connection URL for all Oracle Database adapter put functions in your query.

- Set a different table name using the `%oracle-property:targetTable` annotation in each Oracle Database adapter put function declaration.

The following table identifies the Oracle Loader for Hadoop properties and their equivalent Oracle XQuery for Hadoop annotations by functional category. Oracle XQuery for Hadoop supports only the Oracle Loader for Hadoop properties listed in this table.

Table 6-4 Configuration Properties and Corresponding %oracle-property Annotations

Category	Property	Annotation
Connection	<code>oracle.hadoop.loader.connection.defaultExecuteBatch</code>	<code>%oracle-property:connection.defaultExecuteBatch</code>
Connection	<code>oracle.hadoop.loader.connection.oci_url</code>	<code>%oracle-property:connection.oci_url</code>
Connection	<code>oracle.hadoop.loader.connection.password</code>	<code>%oracle-property:connection.password</code>
Connection	<code>oracle.hadoop.loader.connection.sessionTimeZone</code>	<code>%oracle-property:connection.sessionTimeZone</code>
Connection	<code>oracle.hadoop.loader.connection.tns_admin</code>	<code>%oracle-property:connection.tns_admin</code>
Connection	<code>oracle.hadoop.loader.connection.tnsEntryName</code>	<code>%oracle-property:connection.tnsEntryName</code>
Connection	<code>oracle.hadoop.loader.connection.url</code>	<code>%oracle-property:connection.url</code>
Connection	<code>oracle.hadoop.loader.connection.user</code>	<code>%oracle-property:connection.user</code>
Connection	<code>oracle.hadoop.loader.connection.wallet_location</code>	<code>%oracle-property:connection.wallet_location</code>
General	<code>oracle.hadoop.loader.badRecordFlushInterval</code>	<code>%oracle-property:badRecordFlushInterval</code>
General	<code>oracle.hadoop.loader.compressionFactors</code>	<code>%oracle-property:compressionFactors</code>
General	<code>oracle.hadoop.loader.enableSorting</code>	<code>%oracle-property:enableSorting</code>
General	<code>oracle.hadoop.loader.extTabDirectoryName</code>	<code>%oracle-property:extTabDirectoryName</code>
General	<code>oracle.hadoop.loader.loadByPartition</code>	<code>%oracle-property:loadByPartition</code>
General	<code>oracle.hadoop.loader.logBadRecords</code>	<code>%oracle-property:logBadRecords</code>
General	<code>oracle.hadoop.loader.rejectLimit</code>	<code>%oracle-property:rejectLimit</code>
General	<code>oracle.hadoop.loader.sortKey</code>	<code>%oracle-property:sortKey</code>
General	<code>oracle.hadoop.loader.tableMetadataFile</code>	<code>%oracle-property:tableMetadataFile</code>
General	<code>oracle.hadoop.loader.targetTable</code>	<code>%oracle-property:targetTable</code>
Output	<code>oracle.hadoop.loader.output.dirpathBufsize</code>	<code>%oracle-property:dirpathBufsize</code>
Output	<code>oracle.hadoop.loader.output.escapeEnclosers</code>	<code>%oracle-property:output.escapeEnclosers</code>
Output	<code>oracle.hadoop.loader.output.fieldTerminator</code>	<code>%oracle-property:output.fieldTerminator</code>
Output	<code>oracle.hadoop.loader.output.granuleSize</code>	<code>%oracle-property:output.granuleSize</code>

Table 6-4 (Cont.) Configuration Properties and Corresponding %oracle-property Annotations

Category	Property	Annotation
Output	oracle.hadoop.loader.output.initialFieldEncloser	%oracle-property:output.initialFieldEncloser
Output	oracle.hadoop.loader.output.trailingFieldEncloser	%oracle-property:output.trailingFieldEncloser
Sampler	oracle.hadoop.loader.sampler.enableSampling	%oracle-property:sampler.enableSampling
Sampler	oracle.hadoop.loader.sampler.hintMaxSplitSize	%oracle-property:sampler.hintMaxSplitSize
Sampler	oracle.hadoop.loader.sampler.hintNumMapTasks	%oracle-property:sampler.hintNumMapTask
Sampler	oracle.hadoop.loader.sampler.loadCI	%oracle-property:sampler.loadCI
Sampler	oracle.hadoop.loader.sampler.maxHeapBytes	%oracle-property:sampler.maxHeapBytes
Sampler	oracle.hadoop.loader.sampler.maxLoadFactor	%oracle-property:sampler.maxLoadFactor
Sampler	oracle.hadoop.loader.sampler.maxSamplesPct	%oracle-property:sampler.maxSamplesPct
Sampler	oracle.hadoop.loader.sampler.minSplits	%oracle-property:sampler.minSplits
Sampler	oracle.hadoop.loader.sampler.numThreads	%oracle-property:sampler.numThreads

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](#)
- [Built-in Functions for Reading from and Writing to Oracle NoSQL Database using Table API](#)
- [Custom Functions for Reading Values from Oracle NoSQL Database](#)
- [Custom Functions for Retrieving Single Values from Oracle NoSQL Database](#)
- [Custom Functions for Reading Values from Oracle NoSQL Database using Table API](#)
- [Custom Functions for Reading Single Row from Oracle NoSQL Database using Table API](#)
- [Custom Functions for Retrieving Single Values from Oracle NoSQL Database using Large Object API](#)
- [Custom Functions for Writing to Oracle NoSQL Database](#)

- [Custom Functions for Writing Values to Oracle NoSQL Database using Table API](#)
- [Custom Functions for Writing Values to Oracle NoSQL Database using Large Object API](#)
- [Examples of Oracle NoSQL Database Adapter Functions](#)
- [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.
- The `OXH_SOLR_MR_HOME` environment variable to the local directory containing `search-mr-<version>.jar` and `search-mr-<version>-job.jar`, only when Tika parser is invoked. That is, only when `kv:collection-tika()` or `kv:get-tika()` functions are invoked or, `%kv:collection('tika')` or `%kv:get('tika')` annotations are used with external functions.

You can set the configuration properties using either the `-D` or `-conf` options in the `hadoop` command when you run the query. See "[Run Queries](#)."

This example sets `KVHOME` and uses the `hadoop -D` option 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
```

This example sets `OXH_SOLR_MR_HOME` environment variable when the Tika parser is invoked:

```
$ export OXH_SOLR_MR_HOME=/usr/lib/solr/contrib/mr
```

Note:

The `HADOOP_CLASSPATH` environment variable or `-libjars` command line option must not contain NoSQL DB jars.

See "[Oracle NoSQL Database Adapter Configuration Properties](#)."

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:collection-tika](#)
- [kv:put-text](#)
- [kv:put-xml](#)
- [kv:put-binxml](#)
- [kv:get-text](#)
- [kv:get-avroxml](#)
- [kv:get-xml](#)
- [kv:get-binxml](#)
- [kv:get-tika](#)
- [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;
```

```
declare %kv:collection("text") function
    kv:collection-text($parent-key as xs:string?, $depth as xs:int?) as xs:string*
external;
```

```
declare %kv:collection("text") function
    kv:collection-text($parent-key as xs:string?) as xs:string* external;
```

Parameters

See "[Parameters](#)." Omitting `$subrange` is the same as specifying `$subrange()`. Likewise, omitting `$depth` is the same as specifying `$depth()`.

Returns

One string for each value

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](#)."

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;
```

```
declare %kv:collection("avroxml") function
  kv:collection-avroxml($parent-key as xs:string?, $depth as xs:int?) as element()*
external;
```

```
declare %kv:collection("avroxml") function
  kv:collection-avroxml($parent-key as xs:string?) as element()* external;
```

Parameters

See "[Parameters](#)." Omitting `$subrange` is the same as specifying `$subrange()`. Likewise, omitting `$depth` is the same as specifying `$depth()`.

Returns

One XML element for each Avro record

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;
```

```
declare %kv:collection("xml") function
  kv:collection-xml($parent-key as xs:string?, $depth as xs:int?) as document-
node()* external;
```

```
declare %kv:collection("xml") function
  kv:collection-xml($parent-key as xs:string?) as document-node()* external;
```

Parameters

See "[Parameters](#)." Omitting `$subrange` is the same as specifying `$subrange()`. Likewise, omitting `$depth` is the same as specifying `$depth()`.

Returns

One XML document for each value.

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;
```

```
declare %kv:collection("binxml") function
    kv:collection-binxml($parent-key as xs:string?, $depth as xs:int?) as document-
node()* external;
```

```
declare %kv:collection("binxml") function
    kv:collection-binxml($parent-key as xs:string?) as document-node()* external;
```

Parameters

See "[Parameters](#)." Omitting `$subrange` is the same as specifying `$subrange()`. Likewise, omitting `$depth` is the same as specifying `$depth()`.

Returns

One XML document for each value.

kv:collection-tika

Uses Tika to parse the specified value when invoked and returns as a document node.

Signature

```
declare %kv:collection("tika") function
kv:collection-tika($parent-key as xs:string?, $depth as xs:int?, $subrange as
xs:string?) $contentType as xs:string?) as document-node()* external;
```

Parameters

See "[Parameters](#)." Omitting `$subrange` is the same as specifying `$subrange()`. Likewise, omitting `$depth` is the same as specifying `$depth()`.

Returns

One document node for each value.

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 ."](#)

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.

Signature

```
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;
```

kv:get-tika

Obtains the value associated with the key. The value is parsed as byte array and returned as a document node.

Signature

```
declare %kv:get("tika") function
  kv:get-tika($key as xs:string, $contentType as xs:string?) as document-node()?
external;
```

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.

Built-in Functions for Reading from and Writing to Oracle NoSQL Database using Table API

To use the built-in functions in your query, you must have declared the name space and imported the module as follows:

```
declare namespace kv-table = "oxh:kv-table";
import module "oxh:kv-table";
```

The Oracle NoSQL Database through Table API module contains the following functions:

- [kv-table:collection-jsontext](#)
- [kv-table:get-jsontext](#)
- [kv-table:put-jsontext](#)

kv-table:collection-jsontext

These functions iterate over all or a subset of rows stored in a single table in the NoSQL Database. Each row is returned in a form of a JSON string.

Signature

```

declare %kv-table:collection-jsontext("jsontext") function
    kv-table:collection-jsontext($tableName as xs:string) as xs:string*

declare %kv-table:collection("jsontext") function
    kv-table:collection-jsontext($tableName as xs:string, $primaryKeyJsonValue as
xs:string?) as xs:string*

declare %kv-table:collection("jsontext") function
    kv-table:collection-jsontext($tableName as xs:string, $primaryKeyJsonValue as
xs:string?, $fieldRangeJsonValue as xs:string?) as xs:string*

```

Parameters

`$tableName` as xs:string – name of the table in NoSQL Database

`$primaryKeyJsonValue` as xs:string? – a partial primary key specified as JSON text

See Also:

<http://docs.oracle.com/cd/NOSQL/html/GettingStartedGuideTables/primaryshardkeys.html#partialprimarykeys>

`$fieldRangeJsonValue` as xs:string? – field range for a remaining field of the given primary key specified as JSON text

```

{
  "name": "fieldname",
  "start": "startVal",
  "startInclusive": true|false,
  "end" : "endVal",
  "endInclusive": true|false
}

```

Returns

JSON value of each row

Use "[json:parse-as-xml](#)" function to parse JSON string into an XML document

kv-table:get-jsontext

This function reads a single row stored in a table in NoSQL Database. The row is returned in a form of a JSON string. If the row is not found, then an empty sequence is returned.

Signature

```
declare %kv-table:get("jsontext") function
  kv-table:get-jsontext($tableName as xs:string, $primaryKeyJsonValue as xs:string)
  as xs:string?
```

Parameters

`$tableName` as xs:string – name of the table in NoSQL Database

`$primaryKeyJsonValue` as xs:string? – a full primary key specified as JSON text



See Also:

<http://docs.oracle.com/cd/NOSQL/html/GettingStartedGuideTables/primaryshardkeys.html#primarykeys>

Returns

JSON value of the row or an empty sequence, if the row is not found.

Use ["json:parse-as-xml"](#) function to parse JSON string into an XML document

kv-table:put-jsontext

This function writes a row into NoSQL Database using its Table API

Signature

```
declare %kv-table:put("jsontext") function
  kv-table:put-jsontext($tableName as xs:string, $jsonValue as xs:string);
```

Parameters

`$tableName` as xs:string – name of the table in NoSQL Database

`$jsonValue` as xs:string – row specified as JSON text

Built-in Functions for Reading from and Writing to Oracle NoSQL Database using Large Object API

To use the built-in functions in your query you must have declared the name space and imported the module as follows:

```
declare namespace kv-lob = "oxh:kv-lob";
import module "oxh:kv-lob";
```

The Oracle NoSQL Database through Large Object API module contains the following functions:

- [kv-lob:get-text](#)
- [kv-lob:get-xml](#)
- [kv-lob:get-binxml](#)

- [kv-lob:get-tika](#)
- [kv-lob:put-text](#)
- [kv-lob:put-xml](#)
- [kv-lob:put-binxml](#)

kv-lob:get-text

Obtains the value associated with the key. The value is decoded as UTF-8 and returned as a string.

Signature

```
declare %kv-lob:get("text")
function kv-lob:get-text($key as xs:string) as xs:string?
```

kv-lob:get-xml

Obtains the value associated with the key. The value is read as a sequence of bytes and parsed as XML.

Signature

```
declare %kv-lob:get("xml")
function kv-lob:get-xml($key as xs:string) as document-node()?
```

kv-lob:get-binxml

Obtains the value associated with the key. The value is read as XDK binary XML and returned as an XML document. See *Oracle XML Developer's Kit Programmer's Guide*.

Signature

```
declare %kv-lob:get("binxml")
function kv-lob:get-binxml($key as xs:string) as document-node()?
```

kv-lob:get-tika

Obtains the value associated with the key. The value is parsed as byte array and returned as a document node.

Signature

```
declare %kv-lob:get("tika")
function kv-lob:get-tika($key as xs:string) as document-node()?
```

```
declare %kv-lob:get("tika")
function kv-lob:get-tika($key as xs:string, $contentType as xs:string?) as document-
node()?
```

kv-lob:put-text

Writes a key-value pair. The \$value is encoded as UTF-8.

Signature

```
declare %kv-lob:put("text")
function kv-lob:put-text($key as xs:string, $value as xs:string)
```

kv-lob:put-xml

Writes a key/value pair. The `$xml` is serialized and encoded as UTF-8.

Signature

```
declare %kv-lob:put("xml")
function kv-lob:put-xml($key as xs:string, $document as node())
```

kv-lob:put-binxml

Puts a key/value pair. The `$xml` is encoded as XDK binary XML.

Signature

```
declare %kv-lob:put("binxml")
function kv-lob:put-binxml($key as xs:string, $document as node())
```

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"|"tika"]) [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("tika") [additional annotations]
    function local:myFunctionName($parent-key as xs:string?, $depth as
xs:int?, $subrange as xs:string?, $contentType as xs:string?) as document-node()*
external;
```

Annotations**%kv:collection("method")**

Declares the NoSQL Database collection function. Required. The *method* parameter is one of the following values:

- `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](#)."

- `binxml`: Each value is read as XDK binary XML and returned as an XML document.
- `text`: Each value is decoded using the character set specified by the `%output:encoding` annotation.
- `tika`: Each value is parsed by Tika, and returned as a document node.
- `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/schemaevolution.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 JVM. 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, *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 appear 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"|"tika"]) [additional annotations]
    function local:myFunctionName($key as xs:string) as document-node()?

declare %kv:get(["tika"]) [additional annotations]
    function local:myFunctionName($key as xs:string $contentType 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:

- **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 ."](#)
- **binxml**: The value is read as XDK binary XML and returned as an XML document.
- **text**: The value is decoded using the character set specified by the `%output:encoding` annotation.
- **tika**: Each value is parsed by Tika, and returned as a document node.
- **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/schemaevolution.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 JVM. 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 Reading Values from Oracle NoSQL Database using Table API

You can use the following functions to read values from Oracle NoSQL Database using Table API. These annotations provide additional functionality that is not available using the built-in functions.

Signature

Custom functions for reading collections of NoSQL values using Table API must have one of the following signatures:

```
declare %kv-table:collection("jsontext")
function local:myFunctionName($tableName as xs:string) as xs:string* external;
```

```
declare %kv-table:collection("jsontext")
function local:myFunctionName($tableName as xs:string, $primaryKeyJsonValue as
xs:string?) as xs:string* external;
```



```
declare %kv-table:collection("jsontext")
function local:myFunctionName($tableName as xs:string, $primaryKeyJsonValue as
xs:string?, $fieldRangeJsonValue as xs:string?) as xs:string* external;
```

Annotations

%kv-table:collection("jsontext")

Declares the collection function that uses Table API.

 **Note:**

`jsontext` is the only supported and required annotation value.

Parameters

Same as "[Parameters](#)."

Returns

Same as "[Returns](#)."

Custom Functions for Reading Single Row from Oracle NoSQL Database using Table API

You can use the following functions to read single row from Oracle NoSQL Database using Table API. These annotations provide additional functionality that is not available using the built-in functions.

Signature

Custom functions to read single row from Oracle NoSQL Database using Table API must have one of the following signatures:

```
declare %kv-table:get("jsontext")
function local:myFunctionName($tableName as xs:string, $primaryKeyJsonValue as
xs:string?) as xs:string? external;
```

Annotations

%kv-table:get("jsontext")

Declares the get function that uses Table API.

 **Note:**

`jsontext` is the only supported and required annotation value.

Parameters

Same as "[Parameters](#)."

Returns

Same as "[Returns](#)."

Custom Functions for Retrieving Single Values from Oracle NoSQL Database using Large Object API

You can use the following functions to read values from Oracle NoSQL Database using Large Object API. These annotations provide additional functionality that is not available using the built-in functions.

Signature

Custom functions for reading single values using Large Object API must have one of the following signatures:

```
declare %kv-lob:get("text") [additional annotations]  
function local:myFunctionName($key as xs:string) as xs:string? external;
```

```
declare %kv-lob:get(["xml"|"binxml"|"tika"]) [additional annotations]  
function local:myFunctionName($key as xs:string) as document-node()?
```

```
declare %kv-lob:get(["tika"]) [additional annotations]  
function local:myFunctionName($key as xs:string $contentType as xs:string?) as  
document-node()?
```

Annotations

%kv-lob:get("method")

Declares the NoSQL Database get function that uses Large Object API. Required. Supported method parameters are `binxml`, `text`, `tika`, and `xml` – same as in `%kv:get("method")`.



Note:

`avroxml` method is not supported with Large Object API.

%kv-lob: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. Same as `%kv:key()`.

%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 JVM. This annotation currently only applies to the text method. For XML files, the document encoding declaration is used, if it is available.

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

Annotation	Description
%kv:put("method")	<p>Declares the NoSQL Database module put function. Required.</p> <p>The <i>method</i> determines how the value is stored. It must be one of the following values:</p> <ul style="list-style-type: none"> text: <i>\$value</i> is serialized and encoded using the character set specified by the <code>%output:encoding</code> annotation. avroxml: <i>\$xml</i> is mapped to an instance of the Avro record specified by the <code>%avro:schema-kv</code> annotation. See "Writing XML as Avro." binxml: <i>\$xml</i> is encoded as XDK binary XML. xml: <i>\$xml</i> is serialized and encoded using the character set specified by the <code>%output:encoding</code> annotation. You can specify other XML serialization parameters using <code>%output:*</code>.
%avro:schema-kv("schema-name")	<p>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.</p> <p>For example: <code>%avro:schema-kv("org.example.PersonRecord")</code></p>
%output:*	<p>A standard XQuery serialization parameter for the output method (text or XML) specified in <code>%kv:put</code>. See "Serialization Annotations."</p>

Custom Functions for Writing Values to Oracle NoSQL Database using Table API

You can use the following annotations to define functions that write to Oracle NoSQL Database using Table API.

Signature

Custom functions for writing rows using Table API must have one of the following signatures:

```

declare %kv-table:put("jsontext")
function local:myFunctionName($tableName as xs:string, $jsonValue as xs:string?)
external;

```

Annotations

%kv-table:put("jsontext")

Declares the put function that uses Table API.



Note:

jsontext is the only supported and required annotation value.

Parameters

Same as "Parameters."

Custom Functions for Writing Values to Oracle NoSQL Database using Large Object API

You can use the following annotations to define functions that write to Oracle NoSQL Database using Large Object API.

Signature

Custom functions for writing values using Large Object API must have one of the following signatures:

```
declare %kv-lob:put("text")
function local:myFunctionName($key as xs:string, $value as xs:string) external;
```

```
declare %kv-lob:put(["xml"|"binxml"])
function local:myFunctionName($key as xs:string, $xml as node()) external;
```

Annotations

%kv-lob:put("method")

Declares the NoSQL Database put function. Required. Supported method parameters are binxml, text, and xml – same as in "%kv:put("method")"



Note:

avroxml method is not supported with Large Object API.

%output:*

A standard XQuery serialization parameter for the output method (text or XML) specified in %kv-lob:put. See "[Serialization Annotations](#)."

Examples of Oracle NoSQL Database Adapter Functions

Example 6-8 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 colons (:).

```
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 6-9 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

```

Example 6-10 Storing XML in NoSQL Database

The following query uses the XML files shown in [Example 6-24](#) of "Examples of XML File Adapter Functions" as input. It writes each comment element as an Oracle NoSQL Database value:

```

import module "oxh:xmlf";
import module "oxh:kv";

for $comment in xmlf:collection("mydata/comments*.xml")/comments/comment
let $key := "/comments/" || $comment/@id
return
  kv:put-xml($key, $comment)

```

The query writes the five `comment` elements as XML values in Oracle NoSQL Database.

For very large XML files, modify the query as follows to improve performance and disk space consumption:

- Use the following `for` clause, which causes each XML file to be split and processed in parallel by multiple tasks:


```
for $comment in xmlf:collection("mydata/comments*.xml", "comment")
```
- In the return clause, use `kv:put-binxml` instead of `kv:put-xml` to store the values as binary XML instead of plain text.

Use the `kv:collection-xml` function to read the values in the database. For example:

```

import module "oxh:text";
import module "oxh:kv";

```

```

for $comment in kv:collection-xml("/comments")/comment
return
  text:put($comment/@id || " " || $comment/@user)

```

The query creates text files that contain the following lines:

```

12345 john
23456 john
54321 mike
56789 kelly
87654 mike

```

Example 6-11 Storing XML as Avro in Oracle NoSQL Database

This example converts the XML values to Avro before they are stored.

Add the following Avro schema to Oracle NoSQL Database:

```

{
  "type": "record",
  "name": "Comment",
  "namespace": "com.example",
  "fields" : [
    {"name": "cid", "type": "string"},
    {"name": "user", "type": "string"},
    {"name": "content", "type": "string"},
    {"name": "likes", "type" : { "type" : "array", "items" : "string" } }
  ]
}

```

The following query writes five comment elements as Avro values in Oracle NoSQL Database:

```

import module "oxh:xm1f";
import module "oxh:kv";

declare %kv:put("avroxml") %avro:schema-kv("com.example.Comment")
  function local:put-comment($key as xs:string, $value as node()) external;

for $comment in xm1f:collection("mydata/comments*.xml", "comment")
let $key := "/comments/" || $comment/@id
let $value :=
  <comment>
    <cid>{$comment/@id/data()}</cid>
    <user>{$comment/@user/data()}</user>
    <content>{$comment/@text/data()}</content>
    <likes>{
      for $like in $comment/like
      return <oxh:item>{$like/@user/data()}</oxh:item>
    }</likes>
  </comment>
return
  local:put-comment($key, $value)

```

Use the `kv:collection-avroxml` function to read the values in the database. For example:

```

import module "oxh:text";
import module "oxh:kv";

for $comment in kv:collection-avroxml("/comments")

```

```
return
  text:put($comment/cid || " " || $comment/user || " " || count($comment/likes/*))
```

The query creates text files that contain the following lines:

```
12345 john 0
23456 john 2
54321 mike 1
56789 kelly 2
87654 mike 0
```

Example 6-12 Reading and writing data using Oracle NoSQL Database Table API

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 colons (:).

```
mydata/users.txt
john:John Doe:45
kelly:Kelly Johnson:32
laura:Laura Smith:
phil:Phil Johnson:27
```

Let us create a table called users in NoSQL DB as follows:

```
CREATE TABLE users (id STRING, name STRING, age INTEGER, PRIMARY KEY (id));
```

The first query stores users age into this table.

```
import module "oxh:text";
import module "oxh:kv-table";

for $line in text:collection("mydata/users.txt")
let $split := tokenize($line, ":")
let $id := $split[1]
let $name := $split[2]
let $age := $split[3]
where string-length($age) gt 0
let $row :=
'{' ||
  '"id":"' || $id || ',' ||
  '"name":"' || $name || ',' ||
  '"age":"' || $age ||
'}'

return
  kv-table:put-jsontext("users", $row)
```

After running this query the table contains the following records:

Id	name	age
john	John Doe	45
kelly	Kelly Johnson	32
phil	Phil Johnson	27

The second query reads row from the table and returns ids of users whose name ends with *Johnson*.


```

import module "oxh:text ";
import module "oxh:json";
import module "oxh:kv-table";

for $row in kv-table:collection("users")
let $user := json:parse-as-xml($row)
let $id := $user/json:get("id")
let $name := $user/json:get("name")
where ends-with($name, "Johnson")

return text:put($id)

```

The query creates a text file that contains the following lines:

```

kelly
phil

```

Example 6-13 Reading data using Oracle NoSQL Database Large Object API

Assuming Oracle NoSQL Database contains the following information:

1. Table userImages

```

CREATE TABLE userImages (imageFileName STRING, imageVersion STRING,
imageDescription INTEGER, PRIMARY KEY (imageFileName))

```

imageFileName	imageVersion	imageDescription
IMG_001.JPG	1	Sunrise
IMG_002.JPG	1	Sunrise

2. Key/Value data loaded with Large Object API where:

- Key is the lob/imageFileName/image.lob
- Value is a JPEG image data that contains geolocation metadata in EXIF format

The following query extracts that metadata and converts it to CSV format as imageFileName, latitude, and longitude.

```

import module "oxh:kv-table";
import module "oxh:kv-lob";
import module "oxh:tika";
import module "oxh:json";
import module "oxh:text ";

for $row in kv-table:collection("userImages")

let $imageFileName := json:parse-as-xml($row)/json:get("imageFileName")
let $imageKey := "lob/" || $imageFileName || "/image.lob"
let $doc := kv-lob:get-tika($imageKey, "image/jpeg")
let $lat := $doc/tika:metadata/tika:property[@name eq "GPS Latitude"]
let $lon := $doc/tika:metadata/tika:property[@name eq "GPS Longitude"]
where exists($lat) and exists($lon)

return text:put($imageFileName || "," || $lat || "," || $lon)

```

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 "[Run Queries](#)."

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.

Property	Description
<code>oracle.hadoop.xquery.kv.config.durability</code>	<p>Type: String</p> <p>Default Value: NO_SYNC, NO_SYNC, SIMPLE_MAJORITY</p> <p>Description: Defines the durability characteristics associated with <code>%kv:put</code> operations. The value consists of three parts, which you specify in order and separate with commas (,):</p> <p><i>MasterPolicy, ReplicaPolicy, ReplicaAck</i></p> <ul style="list-style-type: none"> <p><i>MasterPolicy:</i> The synchronization policy used when committing a transaction to the master database. Set this part to one of the following constants:</p> <p>NO_SYNC: Do not write or synchronously flush the log on a transaction commit.</p> <p>SYNC: Write and synchronously flush the log on a transaction commit.</p> <p>WRITE_NO_SYNC: Write but do not synchronously flush the log on a transaction commit.</p> <p><i>ReplicaPolicy:</i> 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 <i>MasterPolicy</i>.</p> <p><i>ReplicaAck:</i> The acknowledgment policy used to obtain transaction acknowledgments from the replica databases. Set this part to one of the following constants:</p> <p>ALL: All replicas must acknowledge that they have committed the transaction.</p> <p>NONE: No transaction commit acknowledgments are required, and the master does not wait for them.</p> <p>SIMPLE_MAJORITY: A simple majority of replicas (such as 3 of 5) must acknowledge that they have committed the transaction.</p>

Property	Description
<code>oracle.hadoop.xquery.kv.config.requestLimit</code>	<p>Type: Comma-separated list of integers 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: <i>maxActiveRequests</i>, <i>requestThresholdPercent</i>, <i>nodeLimitPercent</i></p> <ul style="list-style-type: none"> • <i>maxActiveRequests</i>: 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. • <i>requestThresholdPercent</i>: The percentage of <i>maxActiveRequests</i> at which requests are limited. • <i>nodeLimitPercent</i>: The maximum number of active requests that can be associated with a node when the number of active requests exceeds the threshold specified by <i>requestThresholdPercent</i>.
<code>oracle.hadoop.xquery.kv.config.requestTimeout</code>	<p>Type: Long Default Value: 5000 ms Description: Configures the request timeout period in milliseconds. The value must be greater than zero (0).</p>
<code>oracle.hadoop.xquery.kv.config.socketOpenTimeout</code>	<p>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).</p>
<code>oracle.hadoop.xquery.kv.config.socketReadTimeout</code>	<p>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.</p>
<code>oracle.kv.batchSize</code>	<p>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.</p>

Property	Description
<code>oracle.kv.consistency</code>	<p>Type: Consistency</p> <p>Default Value: NONE_REQUIRED</p> <p>Description: The consistency guarantee for reading child key-value pairs. The following keywords are valid values:</p> <ul style="list-style-type: none"> 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.
<code>oracle.kv.hosts</code>	<p>Type: String</p> <p>Default Value: Not defined</p> <p>Description: An array of one or more <code>hostname:port</code> pairs that identify the hosts in the KV store with the source data. Separate multiple pairs with commas.</p>
<code>oracle.kv.kvstore</code>	<p>Type: String</p> <p>Default Value: Not defined</p> <p>Description: The name of the KV store with the source data.</p>
<code>oracle.kv.timeout</code>	<p>Type: Long</p> <p>Default Value: Not defined</p> <p>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.</p>
<code>oracle.hadoop.xquery.kv.config.LOBSuffix</code>	<p>Type: String</p> <p>Default Value: .lob</p> <p>Description: Configures the default suffix associated with LOB keys.</p>
<code>oracle.hadoop.xquery.kv.config.LOBTimeout</code>	Necessary or FYI? Also, hard-coded link.
<code>oracle.hadoop.xquery.kv.config.readZones</code>	<p>Type: Comma separated list of strings</p> <p>Default Value: Not defined</p> <p>Description: Sets the zones in which nodes must be located to be used for read operations.</p>
<code>oracle.hadoop.xquery.kv.config.security</code>	<p>Type: String</p> <p>Default Value: Not defined</p> <p>Description: Configures security properties for the client.</p>

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](#)
- [Examples of Sequence File Adapter Functions](#)

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:collection-tika](#)
- [seq:put](#)
- [seq:put-xml](#)
- [seq:put-binxml](#)

For examples, see "[Examples of Sequence File Adapter Functions](#)."

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.

Returns

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. See "[Tika Parser Output Format](#)."

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](#)."

seq:collection-tika

Uses Tika to parse the sequence files in the HDFS. The values in the sequence files must be either `org.apache.hadoop.io.Text` or `org.apache.hadoop.io.BytesWritable`. For each value a document node returned produced by Tika.

Signature

```
declare %seq:collection("tika") function
  seq:collection-tika($uris as xs:string*) as document-node()* external;
declare %seq:collection("tika") function
  seq:collection-tika($uris as xs:string*, $contentType as xs:string?) as document-
node()* external;
```

Parameters

`$uris`: The sequence file URIs. The values in the sequence files must be either `org.apache.hadoop.io.Text` or `org.apache.hadoop.io.BytesWritable`. Tika library automatically detects character encoding. Alternatively, the encoding can be passed in `$contentType` parameter as `charset` attribute.

`$contentType`: Specifies the media type of the content to parse, and may have the *charset* attribute.

Returns

One document node for each value in each file.

seq:put

Writes either the string value or both the key and string value of a key-value pair to a sequence file in the output directory of the query.

This function writes the keys and values as `org.apache.hadoop.io.Text`.

When the function is called without the `$key` parameter, it 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($key as xs:string, $value as xs:string) external;

declare %seq:put("text") function
    seq:put($value as xs:string) external;
```

Parameters

`$key`: The key of a key-value pair

`$value`: The value of a key-value pair

Returns

`empty-sequence()`

Notes

The values are spread across one or more sequence files. 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 ["Run Queries."](#)

seq:put-xml

Writes either an XML value or a key and XML value to a sequence file in the output directory of the query.

This function writes the keys and values as `org.apache.hadoop.io.Text`.

When the function is called without the `$key` parameter, it 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($key as xs:string, $xml as node()) external;
```

```
declare %seq:put("xml") function
    seq:put-xml($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 values are spread across one or more sequence files. 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 ["Run Queries."](#)

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`.

When the function is called without the `$key` parameter, it 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($key as xs:string, $xml as node()) external;

declare %seq:put("binxml") function
    seq:put-binxml($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 ["Run Queries."](#)

You can use the `seq:collection-binxml` function to read the files created by this function. See ["seq:collection-binxml."](#)

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"|"tika"]) [additional annotations]
    function local:myFunctionName($uris as xs:string*) as document-node()* external;
declare %seq:collection(["tika"]) [additional annotations]
    function local:myFunctionName($uris as xs:string*, $contentType 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`. Bytes are decoded using the character set specified by the `%output:encoding` annotation. 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.
- **tika**: The values in the sequence files must be either `org.apache.hadoop.io.Text` or `org.apache.hadoop.io.BytesWritable`. The values are parsed by Tika and returned by the function.

%output:encoding("charset")

Specifies the character encoding of the input values. The valid encodings are those supported by the JVM. 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.

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](#)."

**See Also:**

[SequenceFile](#) at Apache's Hadoop Wiki.

"The Influence of Serialization Parameters" sections for XML and text output methods in [XSLT and XQuery Serialization 3.1](#)

Examples of Sequence File Adapter Functions

These examples 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 6-14

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 6-15

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."](#)

Example 6-16

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)
```

Example 6-17

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 query creates a text file that contains the following line:

```
<comment id="12345" user="john" text="It is raining :( "/>
```

Solr Adapter

This adapter provides functions to create full-text indexes and load them into Apache Solr servers. These functions call the Solr `org.apache.solr.hadoop.MapReduceIndexerTool` at run time to generate a full-text index on HDFS and optionally merge it into Solr servers. You can declare and use multiple custom put functions supplied by this adapter and the built-in put function within a single query. For example, you can load data into different Solr collections or into different Solr clusters.

This adapter is described in the following topics:

- [Prerequisites for Using the Solr Adapter](#)
- [Built-in Functions for Loading Data into Solr Servers](#)
- [Custom Functions for Loading Data into Solr Servers](#)
- [Examples of Solr Adapter Functions](#)
- [Solr Adapter Configuration Properties](#)

Prerequisites for Using the Solr Adapter

The first time that you use the Solr adapter, ensure that Solr is installed and configured on your Hadoop cluster as described in "[Install Oracle XQuery for Hadoop](#)".

Configuration Settings

Your Oracle XQuery for Hadoop query must use the following configuration properties or the equivalent annotation:

- `oracle.hadoop.xquery.solr.loader.zk-host`
- `oracle.hadoop.xquery.solr.loader.collection`

If the index is loaded into a live set of Solr servers, then this configuration property or the equivalent annotation is also required:

- `oracle.hadoop.xquery.solr.loader.go-live`

You can set the configuration properties using either the `-D` or `-conf` options in the `hadoop` command when you run the query. See "[Run Queries](#)" and "[Solr Adapter Configuration Properties](#)".

Example Query Using the Solr Adapter

This example sets `OXH_SOLR_MR_HOME` and uses the `hadoop -D` option in a query to set the configuration properties:

```
$ export OXH_SOLR_MR_HOME=/usr/lib/solr/contrib/mr
$ hadoop jar $OXH_HOME/lib/oxh.jar -D oracle.hadoop.xquery.solr.loader.zk-host=/solr
-D oracle.hadoop.xquery.solr.loader.collection=collection1 -D
oracle.hadoop.xquery.solr.loader.go-live=true ./myquery.xq -output ./myoutput
```

Built-in Functions for Loading Data into Solr Servers

To use the built-in functions in your query, you must import the Solr module as follows:

```
import module "oxh:solr";
```

The Solr module contains the following functions:

- `solr:put`

The `solr` prefix is bound to the `oxh:solr` namespace by default.

`solr:put`

Writes a single document to the Solr index.

This document XML format is specified by Solr at

<https://wiki.apache.org/solr/UpdateXmlMessages>

Signature

```
declare %solr:put function  
    solr:put($value as element(doc)) external;
```

Parameters

`$value`: A single XML element named `doc`, which contains one or more `field` elements, as shown here:

```
<doc>  
<field name="field_name_1">field_value_1</field>  
    .  
    .  
    .  
<field name="field_name_N">field_value_N</field>  
</doc>
```

Returns

A generated index that is written into the `output_dir/solr-put` directory, where `output_dir` is the query output directory

Custom Functions for Loading Data into Solr Servers

You can use the following annotations to define functions that generate full-text indexes and load them into Solr.

Signature

Custom functions for generating Solr indexes must have the following signature:

```
declare %solr:put [additional annotations]  
    function local:myFunctionName($value as node()) external;
```

Annotations

%solr:put

Declares the solr put function. Required.

%solr:file(directory_name)

Name of the subdirectory under the query output directory where the index files will be written. Optional, the default value is the function local name.

%solr-property:property_name(value)

Controls various aspects of index generation. You can specify multiple %solr-property annotations.

These annotations correspond to the command-line options of `org.apache.solr.hadoop.MapReduceIndexerTool`. Each `MapReduceIndexerTool` option has an equivalent Oracle XQuery for Hadoop configuration property and a %solr-property annotation. Annotations take precedence over configuration properties. See "[Solr Adapter Configuration Properties](#)" for more information about supported configuration properties and the corresponding annotations.



See Also:

For more information about `MapReduceIndexerTool` command line options, see *Cloudera Search User Guide* at

http://www.cloudera.com/content/cloudera-content/cloudera-docs/Search/latest/Cloudera-Search-User-Guide/csug_mapreduceindexertool.html

Parameters

`$value`: An element or a document node conforming to the Solr XML syntax. See "[solr:put](#)" for details.

Examples of Solr Adapter Functions

Example 6-18 Using the Built-in solr:put Function

This example uses the following HDFS text file. The file contains user profile information such as user ID, full name, and age, separated by colons (:).

```
mydata/users.txt
john:John Doe:45
kelly:Kelly Johnson:32
laura:Laura Smith:
phil:Phil Johnson:27
```

The first query creates a full-text index searchable by name.

```
import module "oxh:text";
import module "oxh:solr";
for $line in text:collection("mydata/users.txt")
let $split := fn:tokenize($line, ":")
let $id := $split[1]
let $name := $split[2]
return solr:put(
<doc>
```



```

<field name="id">{ $id }</field>
<field name="name">{ $name }</field>
</doc>
)

```

The second query accomplishes the same result, but uses a custom put function. It also defines all configuration parameters by using function annotations. Thus, setting configuration properties is not required when running this query.

```

import module "oxh:text";
declare %solr:put %solr-property:go-live %solr-property:zk-host("/solr") %solr-
property:collection("collection1")
function local:my-solr-put($doc as element(doc)) external;
for $line in text:collection("mydata/users.txt")
let $split := fn:tokenize($line, ":")
let $id := $split[1]
let $name := $split[2]
return local:my-solr-put(
<doc>
<field name="id">{ $id }</field>
<field name="name">{ $name }</field>
</doc>
)

```

Solr Adapter Configuration Properties

The Solr adapter configuration properties correspond to the Solr `MapReduceIndexerTool` options.

`MapReduceIndexerTool` is a MapReduce batch job driver that creates Solr index shards from input files, and writes the indexes into HDFS. It also supports merging the output shards into live Solr servers, typically a SolrCloud.

You can specify these properties with the generic `-conf` and `-D hadoop` command-line options in Oracle XQuery for Hadoop. Properties specified using this method apply to all Solr adapter put functions in your query. See "[Run Queries](#)" and especially "[Generic Options](#)" for more information about the `hadoop` command-line options.

Alternatively, you can specify these properties as Solr adapter put function annotations with the `%solr-property` prefix. These annotations are identified in the property descriptions. Annotations apply only to the particular Solr adapter put function that contains them in its declaration.

See Also:

For discussions about how Solr uses the `MapReduceIndexerTool` options, see the *Cloudera Search User Guide* at

http://www.cloudera.com/content/cloudera-content/cloudera-docs/Search/latest/Cloudera-Search-User-Guide/csug_mapreduceindexertool.html

Property	Overview
<code>oracle.hadoop.xquery.solr.loader.collection</code>	<p>Type: String</p> <p>Default Value: Not defined</p> <p>Equivalent Annotation: %solr-property:collection</p> <p>Description: The SolrCloud collection for merging the index, such as <code>mycollection</code>. Use this property with oracle.hadoop.xquery.solr.loader.go-live and oracle.hadoop.xquery.solr.loader.zk-host. Required as either a property or an annotation.</p>
<code>oracle.hadoop.xquery.solr.loader.fair-scheduler-pool</code>	<p>Type: String</p> <p>Default Value: Not defined</p> <p>Equivalent Annotation: %solr-property:fair-scheduler-pool</p> <p>Description: The name of the fair scheduler pool for submitting jobs. The job runs using fair scheduling instead of the default Hadoop scheduling method. Optional.</p>
<code>oracle.hadoop.xquery.solr.loader.go-live</code>	<p>Type: String values true or false</p> <p>Default Value: false</p> <p>Equivalent Annotation: %solr-property:go-live</p> <p>Description: Set to <code>true</code> to enable the final index to merge into a live Solr cluster. Use this property with oracle.hadoop.xquery.solr.loader.collection and oracle.hadoop.xquery.solr.loader.zk-host. Optional.</p>
<code>oracle.hadoop.xquery.solr.loader.go-live-threads</code>	<p>Type: Integer</p> <p>Default Value: 1000</p> <p>Equivalent Annotation: %solr-property:go-live-threads</p> <p>Description: The maximum number of live merges that can run in parallel. Optional.</p>
<code>oracle.hadoop.xquery.solr.loader.log4j</code>	<p>Type: String</p> <p>Default Value:</p> <p>Equivalent Annotation: %solr-property:log4j</p> <p>Description: The relative or absolute path to the <code>log4j.properties</code> configuration file on the local file system. For example, <code>/path/to/log4j.properties</code>. Optional.</p> <p>This file is uploaded for each MapReduce task.</p>
<code>oracle.hadoop.xquery.solr.loader.mappers</code>	<p>Type: String</p> <p>Default Value: -1</p> <p>Equivalent Annotation: %solr-property:mappers</p> <p>Description: The maximum number of mapper tasks that Solr uses. A value of <code>-1</code> enables the use of all map slots available on the cluster.</p>

Property	Overview
<code>oracle.hadoop.xquery.solr.loader.max-segments</code>	<p>Type: String</p> <p>Default Value: 1</p> <p>Equivalent Annotation: <code>%solr-property:max-segments</code></p> <p>Description: The maximum number of segments in the index generated by each reducer.</p>
<code>oracle.hadoop.xquery.solr.loader.reducers</code>	<p>Type: String</p> <p>Default Value: -1</p> <p>Equivalent Annotation: <code>%solr-property:reducers</code></p> <p>Description: The number of reducers to use:</p> <ul style="list-style-type: none"> -1: Uses all reduce slots available on the cluster. -2: Uses one reducer for each Solr output shard. This setting disables the MapReduce M-tree merge algorithm, which typically improves scalability.
<code>oracle.hadoop.xquery.solr.loader.zk-host</code>	<p>Type: String</p> <p>Default Value: Not defined</p> <p>Equivalent Annotation: <code>%solr-property:zk-host</code></p> <p>Description: The address of a ZooKeeper ensemble used by the SolrCloud cluster. Specify the address as a list of comma-separated <i>host:port</i> pairs, each corresponding to a ZooKeeper server. For example, <code>127.0.0.1:2181,127.0.0.1:2182,127.0.0.1:2183/solr</code>. Optional.</p> <p>If the address starts with a slash (/), such as <code>/solr</code>, then Oracle XQuery for Hadoop automatically prefixes the address with the ZooKeeper connection string.</p> <p>This property enables Solr to determine the number of output shards to create and the Solr URLs in which to merge them. Use this property with oracle.hadoop.xquery.solr.loader.collection and oracle.hadoop.xquery.solr.loader.golive. Required as either a property or an annotation.</p>

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 Adapter 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](#)."

text:collection

Accesses a collection of text files in HDFS. The files can be compressed using a Hadoop-supported compression codec. They are automatically decompressed when read.

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;

declare %text:collection("text") function
    function text:collection($uris as xs:string*, $delimiter as xs:string?) as
    xs:string* external;
```

Parameters

`$uris`: The text file URIs.

`$delimiter`: A custom delimiter on which the file is split. The default is the newline character.

Returns

One string value for each file segment identified by the delimiter; for the default delimiter, a string value for each line in each file

text:collection-xml

Accesses a collection of text files in HDFS. The files can be compressed using a Hadoop-supported compression codec. They are automatically decompressed when read.

The files might be split up and processed in parallel by multiple tasks. Each delimited section of each file is parsed as an XML document and returned by the function. Therefore, each segment must fully contain a single XML document, and any delimit characters in the XML must be escaped with XML character references. By default, the delimiter is a new line.

Signature

```
declare %text:collection("xml") function
    text:collection-xml($uris as xs:string*) as document-node()* external;
```

```
declare %text:collection("xml") function
    text:collection-xml($uris as xs:string*, $delimiter as xs:string?) as document-
node()* external;
```

Parameters

`$uris`: The text file URIs.

`$delimiter`: A custom delimiter on which the file is split. The default is the newline character.

Returns

One string value for each file segment identified by the delimiter; for the default delimiter, a string value 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 ["Run Queries."](#)

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 that 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 ["Run Queries."](#)

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()
```

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.

The input files can be compressed with a Hadoop-supported compression codec. They are automatically decompressed when read.

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*, $delimiter as xs:string?) as
    xs:string* external;
```

```
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*, $delimiter as xs:string?) as
    document-node()* 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.

%text:split("delimiter")

Specifies a custom delimiter for splitting the input files. The default delimiter is the newline character.

Do not combine this annotation with the `$delimiter` parameter. To specify a custom delimiter, use either this annotation or the `$delimiter` parameter.

%text: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:

```
%text:split-max(1024)
%text:split-max("1024")
%text:split-max("1K")
```

%text: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:

```
%text:split-min(1024)
%text:split-min("1024")
%text:split-min("1K")
```

Parameters

\$uris as xs:string*

Lists the HDFS file URIs. The files can be uncompressed or compressed with a Hadoop-supported codec. Required.

\$delimiter as xs:string?

A custom delimiter on which the input text files are split. The default delimiter is a new line. Do not combine this parameter with the `%text:split` annotation.

Returns

`xs:string*` for the `text` method

`document-node()*` for the `xml` method

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 process ensures that the resulting XML document is one text line with no line breaks.

%text:compress("codec")

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)

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](#)."

UTF-8 is currently the only supported character encoding.

Examples of Text File Adapter Functions

Example 6-19 Using Built-in Functions to Query Text Files

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])
```

This 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
```

Example 6-20 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 contain the following lines:

```
about.html,2
contact.html,1
index.html,6
```

Example 6-21 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 /index.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 /index.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 6-20](#). 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 /index.html HTTP/1.1
192.0.2.1,GET /aobut.html HTTP/1.1
```

 **See Also:**

- *XPath and XQuery Functions and Operators 3.0* specification for information about the `fn:tokenize` and `fn:analyze-string` functions:
[fn:tokenize](#)
[fn:analyze-string](#)
- For information about the Apache Common log format:
<http://httpd.apache.org/docs/current/logs.html>

Tika File Adapter

This adapter provides functions to parse files stored in HDFS in various formats using Apache Tika library. It is described in the following topics:

- [Built-in Library Functions for Parsing Files with Tika](#)
- [Custom Functions for Parsing Files with Tika](#)
- [Tika Parser Output Format](#)
- [Tika Adapter Configuration Properties](#)
- [Examples of Tika File Adapter Functions](#)

Built-in Library Functions for Parsing Files with Tika

To use the built-in functions in your query, you must import the Tika file module as follows:

```
import module "oxh:tika";
```

The Tika file module contains the following functions:

For examples, see "[Examples of Tika File Adapter Functions](#)."

tika:collection

Parses files stored in HDFS in various formats and extracts the content or metadata from them.

Signature

```
declare %tika:collection function
  tika:collection($uris as xs:string*) as document-node()* external;

declare %tika:collection function
  function tika:collection($uris as xs:string*, $contentType as xs:string?) as
  document-node()* external;
```

Parameters

`$uris`: The HDFS file URIs.

`$contentType`: Specifies the media type of the content to parse, and may have the *charset* attribute. When the parameter is specified, then it defines both type and encoding. When not specified, then Tika will attempt to auto-detect values from the file extension. Oracle recommends you to specify the parameter.

Returns

Returns a document node for each value. See ["Tika Parser Output Format"](#).

tika:parse

Parses the data given to it as an argument. For example, it can parse an html fragment within an XML or JSON document.

Signature

```
declare function
  tika:parse($data as xs:string?, $contentType as xs:string?) as document-node()*
external;
```

Parameters

`$data`: The value to be parsed.

`$contentType`: Specifies the media type of the content to parse, and may have the *charset* attribute. When the parameter is specified, then it defines both type and encoding. When not specified, then Tika will attempt to auto-detect values from the file extension. Oracle recommends you to specify the parameter.

Returns

Returns a document node for each value. See ["Tika Parser Output Format"](#).

Custom Functions for Parsing Files with Tika

You can use the following annotations to define functions to parse files in HDFS with Tika. These annotations provide additional functionality that is not available using the built-in functions.

Signature

Custom functions for reading HDFS files must have one of the following signatures:

```
declare %tika:collection [additional annotations]
  function local:myFunctionName($uris as xs:string*, $contentType as xs:string?) as
document-node()* external;
declare %tika:collection [additional annotations]
  function local:myFunctionName($uris as xs:string*) as document-node()* external;
```

Annotations

%tika:collection(["method"])

Identifies an external function to be implemented by Tika file adapter. Required. The optional *method* parameter can be one of the following values:

- `tika`: Each line in the tika file is returned as `document-node()`. Default.

%output:media-type

Declares the file content type. It is a MIME type and must not have the *charset* attribute as per XQuery specifications. Optional.

%output:encoding

Declares the file character set. Optional.

**Note:**

`%output:media-type` and `%output:encoding` annotations specify the content type or encoding when the `$contentType` parameter is not explicitly provided in the signature.

Parameters**\$uris as xs:string***

Lists the HDFS file URIs. Required.

\$contentType as xs:string?

The file content type. It may have the *charset* attribute.

Returns

`document-node()` * with two root elements. See "[Tika Parser Output Format](#)".

Tika Parser Output Format

The result of Tika parsing is a document node with two root elements:

- Root element #1 is an XHTML content produced by Tika.
- Root element #2 is the document metadata extracted by Tika.

The format of the root elements look like these:

Root element #1

```
<html xmlns="http://www.w3.org/1999/xhtml">
...textual content of Tika HTML...
</html>
```

Root element #2

```
<tika:metadata xmlns:tika="oxh:tika">
  <tika:property name="Name_1">VALUE_1</tika:property>
  <tika:property name="NAME_2">VALUE_2</tika:property>
</tika:metadata>
```

Tika Adapter Configuration Properties

The following Hadoop properties control the behavior of Tika adapter:

oracle.hadoop.xquery.tika.html.asis

Type: Boolean

Default Value: false.

Description: When this is set to TRUE, then all the HTML elements are omitted during parsing. When this is set to FALSE, then only the safe elements are omitted during parsing.

oracle.hadoop.xquery.tika.locale

Type:Comma-separated list of strings

Default Value:Not Defined.

Description:Defines the locale to be used by some Tika parsers such as Microsoft Office document parser. Only three strings are allowed: language, country, and variant. The strings country and variant are optional. When locale is not defined, then the system locale is used. When the strings are defined it must correspond to the `java.util.Locale` specification format mentioned in <http://docs.oracle.com/javase/7/docs/api/java/util/Locale.html> and the locale can be constructed as follows:

- If only language is specified, then the locale is constructed from the language.
- If the language and country are specified, then the locale is constructed from both language and country
- If language, country, and variant are specified, then the locale is constructed from language, country, and variant.

Examples of Tika File Adapter Functions

Example 6-22 Using Built-in Functions to Index PDF documents with Cloudera Search

This example query uses Tika to parse PDF files into HTML form and then add the HTML documents into Solr's full-text index.

```
*bigdata*.pdf
```

The following query indexes the HDFS files:

```
import module "oxh:tika";
import module "oxh:solr";

for $doc in tika:collection("*bigdata*.pdf")
let $docid := data($doc/*:meta[@name eq "resourceName"]/@content)[1]
let $body := $doc/*:body[1]
return
  solr:put(
    <doc>
      <field name="id">{ $docid }</field>
      <field name="text">{ string($body) }</field>
      <field name="content">{ serialize($doc/*:html) }</field>
    </doc>
  )
```

The HTML representation of the documents is added to Solr index and they become searchable. Each document Id in the index is the file name.

Example 6-23 Using Built-in Functions to Index HTML documents with Cloudera Search

This example query uses sequence files and Tika to parse, where key is an URL and value is a html.

```
import module "oxh:tika";
import module "oxh:solr";
import module "oxh:seq";

for $doc in seq:collection-tika("/path/to/seq/files/*")
let $docid := document-uri($doc)
let $body := $doc//*[body[1]]
return
  solr:put(
    <doc>
      <field name="id">{ $docid }</field>
      <field name="text">{ string($body) }</field>
      <field name="content">{ serialize($doc/*.html) }</field>
    </doc>
  )
```

The HTML representation of the documents is added to Solr index and they become searchable. Each document Id in the index is the file name.

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](#)
- [Examples of XML File Adapter Functions](#)

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 \(Single Task\)](#)
- [xmlf:collection-multipart \(Single Task\)](#)
- [xmlf:collection \(Multiple Tasks\)](#)

See "[Examples of XML File Adapter Functions](#)."

xmlf:collection (Single Task)

Accesses a collection of XML documents in HDFS. Multiple files can be processed concurrently, but each individual file is parsed by a single task.

This function automatically decompresses files compressed with a Hadoop-supported codec.

 **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-multipart (Single Task)

Accesses a collection of XML documents in HDFS. Multiple files can be processed concurrently, but each individual file is parsed by a single task. This function is the same as *xmlf:collection* except that each file may contain multiple well-formed XML documents concatenated together.

This function automatically decompresses files compressed with a Hadoop-supported codec. For example, a file containing multiple XML documents could be compressed using GZIP and then accessed directly by this function.

Signature

```
declare %xmlf:collection("multipart")
  function xmlf:collection($uris as xs:string*) as document-node()* external;
```

Parameters**\$uris**

The XML file URIs.

Returns

One or more XML documents for each file.

xmlf:collection (Multiple Tasks)

Accesses a collection of XML documents in HDFS. The files might be split and processed by multiple tasks simultaneously, which enables very large XML files to be processed efficiently. The function returns only elements that match a specified name.

This function does not automatically decompress files. It only supports XML files that meet certain requirements. See "[Restrictions on Splitting XML Files](#)."

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

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(["xml"|"multipart"]) [additional annotations]
    function local:myFunctionName($uris as xs:string*) as node()* external;

declare %xmlf:collection("xml") [additional annotations]
    function local:myFunctionName($uris as xs:string*, $names as xs:anyAtomicType+)
as element()* external;
```

Annotations

%xmlf:collection

Declares the collection function. Required.

The method parameter is one of the following values:

- `xml`: Each value is parsed as XML
- `multipart`: Each value (or, file) may contain a concatenation of multiple well-formed XML documents. This method cannot be used with parallel XML parsing. (See *xmlf:split* and the two-argument function signature.)

%xmlf:split("element-name1"[,... "element-nameN"])

Specifies the element names used for parallel XML parsing. You can use this annotation instead of the `$names` argument.

When this annotation is specified, only the single-argument version of the function is allowed. This restriction 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 JVM. 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")
```

%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)
%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 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 backward 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.

Examples of XML File Adapter Functions

Example 6-24 Using Built-in Functions to Query XML Files

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 query 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

```

Example 6-25 Writing a Custom Function to Query 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

```

Example 6-26 Accessing Compressed, Multipart XML Files

Assume that files comments1.xml, comments2.xml, and comments3.xml from example 5-24 are concatenated together and compressed using GZIP to create a single file named comments.xml.gz. For example:

```

cat comments1.xml comments2.xml comments3.xml | gzip > comments.xml.gz

```

The following query accesses this multipart, compressed XML file:

```

import module "oxh:text"; import module "oxh:xmlf";
for $comment in xmlf:collection-multipart("comments.xml.gz")/comments/
comment
return
  text:put($comment/@id || ", " || $comment/@user)

```

The query creates a text file that contains the following lines:

```

12345,john
56789,kelly

```

```
54321,mike
87654,mike
23456, john
```

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:

- [Oracle XQuery Functions for Duration, Date, and Time](#)
- [Oracle XQuery Functions for Strings](#)

Oracle XQuery Functions for Duration, Date, and Time

You can manipulate durations, dates, and times in XQuery using Oracle XQuery functions.

The Oracle XQuery functions are in namespace `http://xmlns.oracle.com/xdk/xquery/function`. Namespace prefix `ora-fn` is predeclared, and the module is automatically imported.

`ora-fn:date-from-string-with-format`

This Oracle XQuery function returns a new date value from a string according to a given pattern.

Signature

```
ora-fn:date-from-string-with-format($format as xs:string?,
                                   $dateString as xs:string?,
                                   $locale as xs:string*)
as xs:date?

ora-fn:date-from-string-with-format($format as xs:string?,
                                   $dateString as xs:string?)
as xs:date?
```

Parameters

`$format`: The pattern; see [Format Argument](#)

`$dateString`: An input string that represents a date

`$locale`: A one- to three-field value that represents the locale; see [Locale Argument](#)

Example

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-to-string-with-format

This Oracle XQuery function returns a date string with a given pattern.

Signature

```
ora-fn:date-to-string-with-format($format as xs:string?,  
                                $date as xs:date?,  
                                *$locale as xs:string?)  
                                as xs:string?
```

```
ora-fn:date-to-string-with-format($format as xs:string?,  
                                $date as xs:date?)  
                                as xs:string?
```

Parameters

\$format: The pattern; see [Format Argument](#)

\$date: The date

\$locale: A one- to three-field value that represents the locale; see [Locale Argument](#)

Example

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:dateTime-from-string-with-format

This Oracle XQuery function returns a new date-time value from an input string, according to a given pattern.

Signature

```
ora-fn:dateTime-from-string-with-format($format as xs:string?,  
                                       $dateTimeString as xs:string?,  
                                       $locale as xs:string?)  
                                       as xs:dateTime?
```

```
ora-fn:dateTime-from-string-with-format($format as xs:string?,  
                                       $dateTimeString as xs:string?)  
                                       as xs:dateTime?
```

Parameters

\$format: The pattern; see [Format Argument](#)

\$dateTimeString: The date and time

\$locale: A one- to three-field value that represents the locale; see [Locale Argument](#)

Examples

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-to-string-with-format

This Oracle XQuery function returns a date and time string with a given pattern.

Signature

```
ora-fn:dateTime-to-string-with-format($format as xs:string?,  
                                       $dateTime as xs:dateTime?,  
                                       $locale as xs:string?)  
as xs:string?
```

```
ora-fn:dateTime-to-string-with-format($format as xs:string?,  
                                       $dateTime as xs:dateTime?)  
as xs:string?
```

Parameters

\$format: The pattern; see [Format Argument](#)

\$dateTime: The date and time

\$locale: A one- to three-field value that represents the locale; see [Locale Argument](#)

Examples

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:time-from-string-with-format

This Oracle XQuery function returns a new time value from an input string, according to a given pattern.

Signature

```
ora-fn:time-from-string-with-format($format as xs:string?,  
    $timeString as xs:string?,  
    $locale as xs:string?)  
    as xs:time?
```

```
ora-fn:time-from-string-with-format($format as xs:string?,  
    $timeString as xs:string?)  
    as xs:time?
```

Parameters

`$format`: The pattern; see [Format Argument](#)

`$timeString`: The time

`$locale`: A one- to three-field value that represents the locale; see [Locale Argument](#)

Example

This example returns 9:45:22 PM in the current time zone:

```
ora-fn:time-from-string-with-format("HH.mm.ss", "21.45.22")
```

The next example returns 8:07:22 PM in the current time zone:

```
fn-bea:time-from-string-with-format("hh:mm:ss a", "8:07:22 PM")
```

ora-fn:time-to-string-with-format

This Oracle XQuery function returns a time string with a given pattern.

Signature

```
ora-fn:time-to-string-with-format($format as xs:string?,  
    $time as xs:time?,  
    $locale as xs:string?)  
    as xs:string?
```

```
ora-fn:time-to-string-with-format($format as xs:string?, $time as  
xs:time?) as xs:string?
```

Parameters

`$format`: The pattern; see [Format Argument](#)

`$time`: The time

`$locale`: A one- to three-field value that represents the locale; see [Locale Argument](#)

Examples

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"))
```

Format Argument

The `$format` argument identifies the various fields that compose a date or time value.

Locale Argument

The `$locale` represents a specific geographic, political, or cultural region.

It is 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.

Oracle XQuery Functions for Strings

You can manipulate strings in XQuery using Oracle XQuery functions.

The Oracle XQuery functions are in namespace `http://xmlns.oracle.com/xdk/xquery/function`. Namespace prefix `ora-fn` is predeclared, and the module is automatically imported.

ora-fn:pad-left

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. The default padding character is a space (ASCII 32).

Signature

```
ora-fn:pad-left($str as xs:string?,  
               $size as xs:integer?,  
               $pad as xs:string?)  
as xs:string?
```

```
ora-fn:pad-left($str as xs:string?,
               $size as xs:integer?)
               as xs:string?
```

Parameters

`$str`: The input string

`$size`: The desired fixed length, which is obtained by adding padding characters to `$str`

`$pad`: The padding character

If either argument is an empty sequence, then the function returns an empty sequence.

Examples

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 example returns only "ab" because the input string exceeds the specified fixed length:

```
ora-fn:pad-left("abcd", 2, "01")
```

This example prefixes spaces to the string up to the specified maximum of six characters. The returned string has a prefix of 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-right

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. The default padding character is a space (ASCII 32).

Signature

```
ora-fn:pad-right($str as xs:string?,
                 $size as xs:integer?,
                 $pad as xs:string?)
                 as xs:string?
```

```
ora-fn:pad-right($str as xs:string?,
                 $size as xs:integer?)
                 as xs:string?
```

Parameters

`$str`: The input string

`$size`: The desired fixed length, which is obtained by adding padding characters to `$str`

`$pad`: The padding character

If either argument is an empty sequence, then the function returns an empty sequence.

Examples

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")
```

This example returns only "ab" because the input string exceeds the specified fixed length:

```
ora-fn:pad-right("abcd", 2, "01")
```

This example appends spaces to the string up to the specified maximum of six characters. The returned string has a suffix of 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:trim

Removes any leading or trailing white space from a string.

Signature

```
ora-fn:trim($input as xs:string?) as xs:string?
```

Parameters

`$input`: The string to trim. If `$input` is an empty sequence, then the function returns an empty sequence. Other data types trigger an error.

Example

This example returns the string "abc":

```
ora-fn:trim(" abc ")
```

ora-fn:trim-left

Removes any leading white space.

Signature

```
ora-fn:trim-left($input as xs:string?) as xs:string?
```

Parameters

`$input`: The string to trim. If `$input` is an empty sequence, then the function returns an empty sequence. Other data types trigger an error.

Example

This example removes the leading spaces and returns the string "abc":

```
ora-fn:trim-left(" abc ")
```

ora-fn:trim-right

Removes any trailing white space.

Signature

```
ora-fn:trim-right($input as xs:string?) as xs:string?
```

Parameters

`$input`: The string to trim. If `$input` is an empty sequence, then the function returns an empty sequence. Other data types trigger an error.

Example

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.

The Hadoop module is described in the following topic:

- Hadoop Functions

Built-in Functions for Using Hadoop

The following functions are built in to Oracle XQuery for Hadoop:

- [oxh:find](#)
- [oxh:increment-counter](#)
- [oxh:println](#)
- [oxh:println-xml](#)
- [oxh:property](#)

oxh:find

Returns a sequence of file paths that match a pattern.

Signature

```
oxh:find($pattern as xs:string?) as xs:string*
```

Parameters

\$pattern: The file pattern to search for

See Also:

For the file pattern, the `globStatus` method in the Apache Hadoop API at

[http://hadoop.apache.org/docs/current/api/org/apache/hadoop/fs/FileSystem.html#globStatus\(org.apache.hadoop.fs.Path\)](http://hadoop.apache.org/docs/current/api/org/apache/hadoop/fs/FileSystem.html#globStatus(org.apache.hadoop.fs.Path))

oxh:increment-counter

Increments a user-defined MapReduce job counter. The default increment is one (1).

Signature

```
oxh:increment-counter($groupName as xs:string, $counterName as xs:string, $value as xs:integer)
```

```
oxh:increment-counter($groupName as xs:string, $counterName as xs:string)
```

Parameters

\$groupName: The group of counters that this counter belongs to.

\$counterName: The name of a user-defined counter

\$value: The amount to increment the counter

oxh:println

Prints a line of text to `stdout` of the Oracle XQuery for Hadoop client process. Use this function when developing queries.

Signature

```
declare %updating function oxh:println($arg as xs:anyAtomicType?)
```

Parameters

`$arg`: A value to add to the output. A cast operation first converts it to `string`. An empty sequence is handled the same way as an empty string.

Example

This example prints the values of `data.txt` to `stdout`:

```
for $i in text:collection("data.txt")
return oxh:println($i)
```

oxh:println-xml

Prints a line of text or XML to `stdout` of the Oracle XQuery for Hadoop client process. Use this function when developing queries and printing nodes of an XML document.

Signature

```
declare %updating function oxh:println-xml($arg as item())?
```

Parameters

`$arg`: A value to add to the output. The input item is converted into a text as defined by XSLT 2.0 and XQuery 1.0 Serialization specifications. An empty sequence is handled the same way as an empty string.

oxh:property

Returns the value of a Hadoop configuration property.

Signature

```
oxh:property($name as xs:string?) as xs:string?
```

Parameters

`$name`: 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`: Any encoding supported by the JVM

- `normalization-form`: none, NFC, NFD, NFKC, NFKD

Serialization parameters supported for the `xml` output method, using any values permitted by the XQuery specification:

- `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*, at locations like the following:

http://www.w3.org/TR/xslt-xquery-serialization/#XML_DOCTYPE

http://www.w3.org/TR/xslt-xquery-serialization/#XML_CDATA-SECTION-ELEMENTS

7

Oracle XML Extensions for Hive

This chapter explains how to use the XML extensions for Apache Hive provided with Oracle XQuery for Hadoop. The chapter contains the following sections:

- [What are the XML Extensions for Hive?](#)
- [Use the Hive Extensions From the Command Line](#)
- [About the Hive Functions](#)
- [Create XML Tables](#)
- [Oracle XML Functions for Hive Reference](#)

What are the XML Extensions for Hive?

The XML Extensions for Hive provide 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 3.1
- XQuery Update Facility 1.0 (transform expressions)
- XPath 3.1
- XML Schema 1.0
- XML Namespaces

The XML extensions have two components:

- XML InputFormat and SerDe for creating XML tables
See "[Create XML Tables.](#)"
- XML function library
See "[About the Hive Functions.](#)"

Use the Hive Extensions From the Command Line

To enable the Oracle XQuery for Hadoop extensions, use the `--auxpath` and `-i` arguments when starting Hive:

```
$ hive --auxpath \
$OXH_HOME/hive/lib/oxh-hive.jar,\
$OXH_HOME/hive/lib/oxh-mapreduce.jar,\
$OXH_HOME/hive/lib/oxquery.jar,\
$OXH_HOME/hive/lib/xqjapi.jar,\
$OXH_HOME/hive/lib/apache-xmlbeans.jar,\
$OXH_HOME/hive/lib/woodstox-core-asl-*.jar,\
$OXH_HOME/hive/lib/stax2-api-*.jar \
-i $OXH_HOME/hive/init.sql
```

Note:

On the Oracle BigDataLite VM, `HIVE_AUX_JARS_PATH` contains the Hive extensions by default and hence specifying `--auxpath` is unnecessary.

The first time you use the extensions, verify that they are accessible. The following procedure creates a table named `SRC`, loads one row into it, and calls the `xml_query` function.

To verify that the extensions are accessible:

1. Log in to a server in the Hadoop cluster where you plan to work.
2. Start the Hive command-line interface (CLI):

```
$ hive --auxpath \
$OXH_HOME/hive/lib/oxh-hive.jar,\
$OXH_HOME/hive/lib/oxh-mapreduce.jar,\
$OXH_HOME/hive/lib/oxquery.jar,\
$OXH_HOME/hive/lib/xqjapi.jar,\
$OXH_HOME/hive/lib/apache-xmlbeans.jar,\
$OXH_HOME/hive/lib/woodstox-core-asl-*.jar,\
$OXH_HOME/hive/lib/stax2-api-*.jar \
-i $OXH_HOME/hive/init.sql
```

The `init.sql` file contains the `CREATE TEMPORARY FUNCTION` statements that declare the XML functions.

3. Call an Oracle XQuery for Hadoop function for Hive. This example calls the `xml_query` function to parse an XML string:

```
hive> SELECT xml_query("x/y", "<x><y>123</y><z>456</z></x>");
.
.
.
["123"]
```

If the extensions are accessible, then the query returns ["123"], as shown in the example.

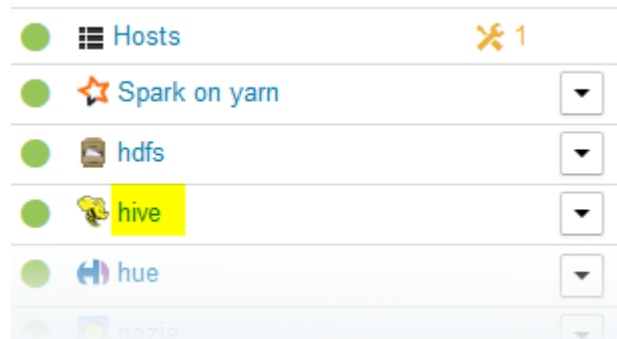
Use the Hive Extensions in HiveServer2

To enable the XML Extensions for Hive in HiveServer2, you must add the following JARs to the environment variable `HIVE_AUX_JARS_PATH` and to the configuration property `hive.aux.jars.path`.

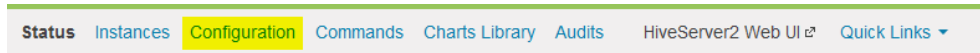
```
$OXH_HOME/hive/lib/woodstox-core-asl-*.jar
$OXH_HOME/hive/lib/apache-xmlbeans.jar
$OXH_HOME/hive/lib/oxh-hive.jar
$OXH_HOME/hive/lib/oxh-mapreduce.jar
$OXH_HOME/hive/lib/stax2-api-*.jar
$OXH_HOME/hive/lib/xqjapi.jar
$OXH_HOME/hive/lib/oxquery.jar
```

If you are using Cloudera's distribution including Apache Hadoop (on Oracle Big Data Appliance or commodity hardware) you can add the JARs as follows:

1. In Cloudera Manager, click **hive**:



2. Click **Configuration** to go to the Configuration tab.



3. In the search field, enter `hive_aux`. Then, add the following text after whatever value is in **Gateway Client Environment Advanced Configuration Snippet (Safety Valve) for hive-env.sh**. Be sure to include the leading comma. It is advisable to save the text of the original path before editing this field in case you need to revert at a later time.

```
, /opt/oracle/oxh-*/hive/lib/woodstox-core-asl-oxh-*.jar, /opt/oracle/oxh-*/hive/lib/apache-xmlbeans.jar, /opt/oracle/oxh-*/hive/lib/oxh-hive.jar, /opt/oracle/oxh-*/hive/lib/oxh-mapreduce.jar, /opt/oracle/oxh-*/hive/lib/stax2-api-*.jar, /opt/oracle/oxh-*/hive/lib/xqjapi.jar, /opt/oracle/oxh-*/hive/lib/oxquery.jar
```

Note that the JAR version numbers in the paths above are wildcarded (*). Replace the wildcards with the correct JAR versions for the OXH release you are working with.

hive_aux

Hive Auxiliary JARs Directory hive (Service-Wide) ↵
/opt/oracle/bigdatasql/bdcell-12.1/lib

Gateway Client Environment Advanced Configuration Snippet (Safety Valve) for hive-env.sh Gateway Default Group ↵

```
HIVE_AUX_JARS_PATH=$HIVE_AUX_JARS_PATH,/opt/cloudera/parcels/CDH/lib/hive/lib/hive-contrib.jar,/opt/oracle/oxh-4.8.0-1.cdh5.0.0/hive/lib/orai18n-utility.jar, /opt/oracle/oxh-4.8.0-1.cdh5.0.0/hive/lib/woodstox-core-ast-4.2.0.jar,/opt/oracle/oxh-4.8.0-1.cdh5.0.0/hive/lib/apache-xmbeans.jar,/opt/oracle/oxh-4.8.0-1.cdh5.0.0/hive/lib/oxh-hive.jar,/opt/oracle/oxh-4.8.0-1.cdh5.0.0/hive/lib/oxh-mapreduce.jar, /opt/oracle/oxh-4.8.0-1.cdh5.0.0/hive/lib/orai18n-mapping.jar,/opt/oracle
```

- Search for `hive_aux_jars_path`. Add the following text after whatever value is in **Hive Client Advanced Configuration Snippet (Safety Valve) for hive-site.xml**. Be sure to add the text to the **value** field in the **Gateway Default Group** panel:

hive_aux_jars_path

Hive Auxiliary JARs Directory hive (Service-Wide) ↵
/opt/oracle/bigdatasql/bdcell-12.1/lib

Hive Service Advanced Configuration Snippet (Safety Valve) for hive-site.xml hive (Service-Wide) ↵

Name

Value

Description

Final

+

Hive Client Advanced Configuration Snippet (Safety Valve) for hive-site.xml Gateway Default Group ↵

Name

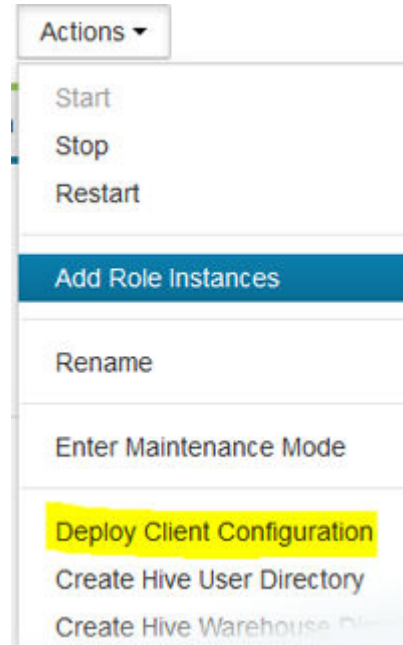
Value

Description

Final

- Click **Save Changes**.

6. Click **Actions** and then click **Deploy Client Configuration**.



7. After the service restarts, check the status to ensure that it is working correctly.
8. To verify that the extensions are accessible, use Beeline:
- a. Start Beeline:

```
$ beeline -i $OXH_HOME/hive/init.sql
```

 **Note:**

You can use `init.sql` as shown above to temporarily declare the functions `xml_query`, `xml_table`, et cetera. You can also enable the functions permanently as described in [Permanently Declaring the Hive Functions](#).

- b. At the Beeline CLI, enter the following. (Fill in the placeholders `host` and `port` with the host and port of `hiveserver2`.)

```
beeline> !connect jdbc:hive2: //[host]:[port]
```

When prompted for a username and password, you can press **Enter**.

- c. All the XML extension functions should be created when you connected to Hive. Try the following command to test if they are working correctly:

```
beeline> SELECT xml_query("x/y", "<x><y>123</y><z>456</z></x>");
```

The results returned should be similar to this example:

```
+-----+
|  _c0  |
```

```
+-----+
| ["123"] |
+-----+
```

If this test is successful, then you are ready to use the Oracle XQuery for Hadoop functions for Hive in HiveServer2.

See Also:

- [Permanently Declaring the Hive Functions.](#)

You can use `init.sql` to temporarily declare the functions, as shown in previous examples. You can also permanently declare them.

About the Hive Functions

The Oracle XQuery for Hadoop extensions enable you to query XML strings in Hive tables and XML file resources in the Hadoop distributed cache. These are the functions:

- `xml_query`: Returns the result of a query as an array of `STRING` values.
- `xml_query_as_primitive`: Returns the result of a query as a Hive primitive value. Each Hive primitive data type has a separate function named for it.
- `xml_exists`: Tests if the result of a query is empty
- `xml_table`: Maps an XML value to zero or more table rows, and enables nested repeating elements in XML to be mapped to Hive table rows.

See "[Oracle XML Functions for Hive Reference.](#)"

Permanently Declaring the Hive Functions

In the examples in the previous section, `$OXH_HOME/hive/init.sql` is used to temporarily declare the XML extensions for Hive functions. However, as an alternative, you can permanently declare the functions so that `init.sql` is not needed. Use the following commands to permanently declare the functions.

```
CREATE FUNCTION xml_query AS 'oracle.hadoop.xquery.hive.OXMLQueryUDF' ;
CREATE FUNCTION xml_query_as_bigint AS
'oracle.hadoop.xquery.hive.OXMLQueryBigintUDF' ;
CREATE FUNCTION xml_query_as_int AS
'oracle.hadoop.xquery.hive.OXMLQueryIntUDF' ;
CREATE FUNCTION xml_query_as_smallint AS
'oracle.hadoop.xquery.hive.OXMLQuerySmallintUDF' ;
CREATE FUNCTION xml_query_as_tinyint AS
'oracle.hadoop.xquery.hive.OXMLQueryTinyintUDF' ;
CREATE FUNCTION xml_query_as_float AS
'oracle.hadoop.xquery.hive.OXMLQueryFloatUDF' ;
CREATE FUNCTION xml_query_as_double AS
'oracle.hadoop.xquery.hive.OXMLQueryDoubleUDF' ;
```

```
CREATE FUNCTION xml_query_as_boolean AS
'oracle.hadoop.xquery.hive.OXMLQueryBooleanUDF' ;
CREATE FUNCTION xml_query_as_string AS
'oracle.hadoop.xquery.hive.OXMLQueryStringUDF' ;
CREATE FUNCTION xml_exists AS 'oracle.hadoop.xquery.hive.OXMLExists' ;
CREATE FUNCTION xml_table AS 'oracle.hadoop.xquery.hive.OXMLTableUDTF' ;
```

Create 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](#)." Otherwise, you might get errors or incorrect results.

Hive CREATE TABLE Syntax for XML Tables

The following is the basic syntax of the Hive `CREATE TABLE` statement for 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)
```

Parameters

Parameter	Description
columns	All column types in an XML table must be one of the Hive primitive types given in " Data Type Conversions ."
configuration	Any of the properties described in " CREATE TABLE Configuration Properties ." Separate multiple properties with commas.

Note:

Inserting data into XML tables is not supported.

CREATE TABLE Configuration Properties

Use these configuration properties in the [configuration](#) parameter of the `CREATE TABLE` command.

oxh-default-namespace

Sets the default namespace for expressions in the table definition and for XML parsing. The value is a URI.

This example defines the default namespace:

```
"oxh-default-namespace" = "http://example.com/foo"
```

oxh-charset

Specifies the character encoding of the XML files. The supported encodings are UTF-8 (default), ISO-8859-1, and US-ASCII.

All XML files for the table must share the same character encoding. Any encoding declarations in the XML files are ignored.

This example defines the character set:

```
"oxh-charset" = "ISO-8859-1"
```

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.

Check the 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.

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. See "[Syntax Example](#)."

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-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-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 example, 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"
```

CREATE TABLE Examples

This section includes the following examples:

- [Syntax Example](#)
- [Simple Examples](#)
- [OpenStreetMap Examples](#)

Syntax Example

This example shows how to map XML elements to column names.

Example 7-1 Basic Column Mappings

In the following table definition, 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 a 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" = "./bar",
  "oxh-column.zip" = "./zip"
)
```

Example 7-2 Conditional Column Mappings

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 if the `zip` element contains a nonnumeric value:

```
"oxh-column.zip" = "
  if (./zip castable as xs:int) then
    xs:int(./zip)
  else
    -1
"
```

Example 7-3 Default Column Mappings

The following two table definitions are equivalent. Table Definition 2 relies on the default mappings for the `BAR` and `ZIP` columns.

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"
)
```

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 (`comments.xml`) 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 7-4 Creating a Table

The following Hive CREATE TABLE command creates a table named `COMMENTS` with a row for each comment containing the user names, text, and number of likes:

```
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.

```
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 7-5 Querying an XML Column

This CREATE TABLE command is like the previous example, 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](#)" 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 7-6 Generating XML in a Single String Column

This command 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](#)" 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
```

OpenStreetMap 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, go to <http://www.openstreetmap.org/export>
- For information about the OpenStreetMap XML format, go to http://wiki.openstreetmap.org/wiki/OSM_XML

Example 7-7 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 7-8

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.

```

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;
```

See Also:

For a description of the OpenStreetMap elements and attributes, go to

<http://wiki.openstreetmap.org/wiki/Elements>

Oracle XML Functions for Hive Reference

This section describes the Oracle XML Extensions for Hive. It describes the following commands and functions:

- [xml_exists](#)
- [xml_query](#)
- [xml_query_as_primitive](#)
- [xml_table](#)

Data Type Conversions

This table 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

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 examples of accessing the distributed cache, see [Example 7-15](#) for `xml_query`, [Example 7-22](#) for `xml_query_as_primitive`, and [Example 7-31](#) for `xml_table`.
- For information about the default base URI, see *XQuery 3.1: An XML Query Language* at <http://www.w3.org/TR/xquery/#dt-base-uri>

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"]
.
.
.
```

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. 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. 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.
- `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 "[Data Type Conversions](#)."

Return Value

`true` if the result of the query is not empty; `false` if the result is empty or the query raises a dynamic error

Notes

The first dynamic error raised by a query is logged, but subsequent errors are suppressed.

Examples

Example 7-9 STRING Binding

This example parses and binds the input XML string to the initial context item of the query `x/y`:

```
Hive> SELECT xml_exists("x/y", "<x><y>123</y></x>") FROM src LIMIT 1;
.
.
.
true
```

Example 7-10 STRUCT Binding

This example defines two query variables, `$data` and `$value`:

```
Hive> 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 7-11 Error Logging

In this example, an error is written to the log, because the input XML is invalid:

```
hive> SELECT xml_exists("x/y", "<x><y>123</invalid></x>") FROM src LIMIT 1;
.
.
.
false
```

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 7-15](#).

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 7-12](#).
- `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 "[Data Type Conversions](#)" and [Example 7-13](#).

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 7-12 Using a `STRING` Binding

This example parses and binds the input XML string to the initial context item of the query `x/y`:

```
hive>
SELECT xml_query("x/y", "<x><y>hello</y><z/><y>world</y></x>")
FROM src LIMIT 1;
.
.
.
["hello", "world"]
```

Example 7-13 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 "[Data Type Conversions](#)."

```
hive>
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 7-14 Obtaining Serialized XML

This example uses the `fn:serialize` function to return serialized XML:

```
hive>
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 7-15 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 this value:

```
<x><y>hello</y><z/><y>world</y></x>

hive> ADD FILE test.xml;
Added resource: test.xml
hive> SELECT xml_query("fn:doc('test.xml')/x/y", NULL) FROM src LIMIT 1;
.
.
.
["hello", "world"]
```

Example 7-16 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:

```
hive> SELECT xml_query("x/y", "<x><y>hello</y></invalid") FROM src LIMIT 1;
.
.
.
[]
```

xml_query_as_primitive

Returns the result of a query as a Hive primitive value. Each Hive primitive data type has a separate function named for it:

- `xml_query_as_string`
- `xml_query_as_boolean`
- `xml_query_as_tinyint`
- `xml_query_as_smallint`
- `xml_query_as_int`
- `xml_query_as_bigint`
- `xml_query_as_double`
- `xml_query_as_float`

Signature

```
xml_query_as_primitive (
  STRING query,
  {STRUCT | STRING} bindings,
) as primitive
```

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 7-15](#).

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 7-17](#).
- `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 "[Data Type Conversions](#)" and [Example 7-18](#).

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

A Hive primitive value, which is the first item returned by the query, converted to an atomic value. If the result of the query is empty, then the return value is `NULL`.

Examples

Example 7-17 Using a STRING Binding

This example parses and binds the input XML string to the initial context item of the query `x/y`:

```
hive> 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:

```
hive> SELECT xml_query_as_int("x/y", "<x><y>123</y></x>") FROM src LIMIT 1;
.
```

```

.
.
123

hive> SELECT xml_query_as_double("x/y", "<x><y>12.3</y></x>") FROM src LIMIT 1;
.
.
12.3

hive> SELECT xml_query_as_boolean("x/y", "<x><y>>true</y></x>") FROM src LIMIT 1;
.
.
true

```

Example 7-18 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 ["Data Type Conversions."](#)

```

hive>
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 7-19 Returning Multiple Query Results

This example returns only the first item (hello) from the query. The second item (world) is discarded.

```

hive> SELECT xml_query_as_string("x/y", "<x><y>hello</y><z/><y>world</y></x>") FROM
src LIMIT 1;
.
.
hello

```

Example 7-20 Returning Empty Query Results

This example returns `NULL` because the result of the query is empty:

```

hive> SELECT xml_query_as_string("x/foo", "<x><y>hello</y><z/><y>world</y></x>")
FROM src LIMIT 1;
.
.
NULL

```

Example 7-21 Obtaining Serialized XML

These examples use the `fn:serialize` function to return complex XML elements as a `STRING` value:

```
hive> 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>"

hive> 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 7-22 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 this value:

```
<x><y>hello</y><z/><y>world</y></x>

Hive> ADD FILE test.xml;
Added resource: test.xml
Hive> SELECT xml_query_as_string("fn:doc('test.xml')/x/y[1]", NULL) FROM src LIMIT 1;
.
.
.
hello
```

Example 7-23 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:

```
Hive> 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:

```
Hive> SELECT xml_query_as_float("x/y", "<x><y>foo</y></x>") FROM src LIMIT 1;
.
.
.
NULL
```


xml_table

A user-defined 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 7-26](#).

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 7-24](#).
- `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 "[Data Type Conversions](#)."

columns

The XQuery or XPath expressions that define the columns of the generated rows. Optional.

The value is a `STRUCT` that contains the additional XQuery expressions. 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 7-25](#).

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 "[Hive CREATE TABLE Syntax for XML Tables](#)".

Examples

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 7-24 Using a STRING Binding

The query "x/y" returns two <y> elements, therefore two table rows are generated. Because there are two column expressions (".z", ".w"), each row has two columns.

```
hive> 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 7-25 Using the Columns Argument

The following two queries are equivalent. The first query explicitly specifies the value of the *columns* argument:

```
hive> 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(".")`:

```

hive> SELECT xml_table(
        "x/y",
        "<x><y>hello</y><y>world</y></x>"
    ) AS (y)
FROM src;
.
.
.
hello
world

```

Example 7-26 Using the Namespaces Argument

This example specifies the optional *namespaces* argument, which identifies an *ns* prefix and a URI of `http://example.org`.

```

hive> 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 7-27 Querying a Hive Table of XML Documents

This example queries a table named `COMMENTS3`, which has a single column named `XML_STR` of type `STRING`. It contains these three rows:

```

hive> SELECT xml_str 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 following query shows how to extract the user, text, and number of likes from the `COMMENTS3` table.

```

hive> SELECT t.id, t.usr, t.likes
FROM comments3 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
```

Example 7-28 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 the previous example for the `COMMENTS` table.

```
> SELECT t.i, t.u, t.l
       FROM comments3 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 7-29 Mapping Optional Nested XML Elements to Table Rows

This example is a slight modification of the previous example that produces a row even when a comment has no likes. See [Example 7-27](#) for the `COMMENTS` table.

```
> SELECT t.i, t.u, t.l
       FROM comments3 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 7-30 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:

```
hive> 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 7-31 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 this string:

```
<x><y>hello</y><z/><y>world</y></x>
```

```
hive> ADD FILE test.xml;
Added resource: test.xml
hive> SELECT xml_table("fn:doc('test.xml')/x/y", NULL) AS y FROM src;
.
.
.
hello
world
```

Part IV

Oracle R Advanced Analytics for Apache Hadoop

This part contains the following chapter:

- [Oracle R Advanced Analytics for Apache Hadoop](#)

8

Oracle R Advanced Analytics for Apache 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](#)
- [Oracle R Advanced Analytics for Hadoop Functions](#)
- [Demos of Oracle R Advanced Analytics for Hadoop Functions](#)
- [Security Notes for Oracle R Advanced Analytics for Hadoop](#)

Note:

Oracle R Advanced Analytics for Apache Hadoop (Oracle R Advanced Analytics for Hadoop) was previously called Oracle R Connector for Hadoop or ORCH. ORCH is still mentioned in this document and in the product for backward compatibility.

About Oracle R Advanced Analytics for Hadoop

Oracle R Advanced Analytics for Hadoop provides:

- A general computation framework, in which you can use the R language to write your custom logic as mappers or reducers. The code executes in a distributed, parallel manner using the available compute and storage resources on the Hadoop cluster.
- An R interface to manipulate Hive tables, which is similar to the transparency layer of Oracle R Enterprise but with a restricted set of functionality.
- A set of pre-packaged parallel-distributed algorithms.
- Support for Apache Spark, with which you can execute predictive analytics functions on a Hadoop cluster using YARN to dynamically form a Spark cluster or on a dedicated stand-alone Spark cluster. You can switch on or off Spark execution using `spark.connect()` and `spark.disconnect()` functions.
- The ability to use Spark to execute neural network analytical function (`orch.neural`), for significantly improved performance over MapReduce execution.

Oracle R Advanced Analytics for Hadoop Architecture

Oracle R Advanced Analytics for Hadoop:

- is built upon Hadoop streaming, a utility that is a part of Hadoop distribution and allows creation and execution of Map or Reduce jobs with any executable or script as mapper or reducer.
- is designed for R users to work with Hadoop cluster in a client-server configuration. Client configurations must conform to the requirements of the Hadoop distribution that Oracle R Advanced Analytics for Hadoop is deployed in.
- uses command line interfaces to HDFS and HIVE to communicate from client nodes to Hadoop clusters.
- builds the logic required to transform an input stream of data into R data frame object to be readily consumed by user-provided mapper and reducer functions written into R.
- allows R users to move data from an Oracle Database table or view into Hadoop as an HDFS file, using the Sqoop utility. Similarly data can be moved back from an HDFS file into Oracle Database, using the Sqoop utility or Oracle Loader for Hadoop, depending on the size of data being moved and security requirements
- support's R's binary RData representation for input and output, for performance sensitive analytic workloads. Conversion utilities from delimiter separated representation to and from RData representation is available as part of Oracle R Advanced Analytics for Hadoop.
- includes a Hadoop Abstraction Layer (HAL) which manages the similarities and differences across various Hadoop distributions. ORCH will auto-detect the Hadoop version at startup.

Oracle R Advanced Analytics for Hadoop packages and functions

Oracle R Advanced Analytics for Hadoop includes a collection of R packages that provides:

- Interfaces to work with the:
 - Apache Hive tables
 - Apache Hadoop compute infrastructure
 - local R environment
 - Oracle Database tables
 - -Proprietary binary RData representations
 - Apache Spark RDD objects
- Predictive analytic techniques for:
 - linear regression
 - generalized linear models
 - neural networks
 - matrix completion using low rank matrix factorization
 - nonnegative matrix factorization

- k-means clustering
- principal components analysis
- multivariate analysis

ORAAH 2.6 introduces full stack of predictive modeling algorithms on Spark. This includes integration of many Spark MLLib capabilities, including Linear Model techniques (Linear Regression, LASSO, Ridge Regression), as well as GLM, SVM, k-Means, Gaussian Mixture clustering, Decision Trees, Random Forests and Gradient Boosted Trees, PCA and SVD. Existing ORAAH custom Spark algorithms are enhanced with the addition of Linear Models and Stepwise capability for both LM and GLM.

While these techniques have R interfaces, Oracle R Advanced Analytics for Hadoop implement them in either Java or R as distributed, parallel MapReduce jobs, thereby leveraging all nodes of your Hadoop cluster.

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 Database instances
- Manipulate Hive data transparently from R
- Execute R programs as Hadoop MapReduce jobs and return the results to any of those locations
 - With Oracle R Advanced Analytics for Hadoop, MapReduce jobs can be submitted from R for both non-cluster (local) execution and Hadoop cluster execution
 - When Oracle R Enterprise and Oracle R Advanced Analytics for Hadoop are used together on a database server, you can schedule database jobs using the DBMS_SCHEDULER to execute scripts containing ORCH functions

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 "[Oracle R Advanced Analytics for Hadoop Functions](#)".



See Also:

The R Project website at <http://www.r-project.org/>

Inputs to Oracle R Advanced Analytics for Hadoop

Oracle R Advanced Analytics for Hadoop can work with delimited text files resident in an HDFS directory, HIVE tables, or binary RData representations of data. If the input data to an Oracle R Advanced Analytics for Hadoop orchestrated map-reduce computation does not reside in HDFS, a copy of the data in HDFS is created automatically prior to launching the computation.

Before Oracle R Advanced Analytics for Hadoop can work with delimited text files it determines metadata associated with the files and captures the same in a file stored alongside of the data files. This file is named `__ORCHMETA__`. The metadata contains information such as:

- If the file contains key(s), then the delimiter that is the key separator
- The delimiter that is the value separator
- Number and data types of columns in the file
- Optional names of columns
- Dictionary information for categorical columns
- Other Oracle R Advanced Analytics for Hadoop-specific system data

Oracle R Advanced Analytics for Hadoop runs an automatic metadata discovery procedure on HDFS objects as part of `hdfs.attach()` invocation to create the metadata file. When working with HIVE tables, `__ORCHMETA__` file is created automatically from the HIVE table definition2.

Oracle R Advanced Analytics for Hadoop can optionally convert input data into R's binary RData representation for I/O performance that is on par with a pure Java based map-reduce implementation.

Oracle R Advanced Analytics for Hadoop captures row streams from HDFS files and delivers them formatted as a data frame object (or optionally matrix, vector, or list objects generated from the data frame object or AS IS, if RData representation is used) to the mapped function written in R. To accomplish this, Oracle R Advanced Analytics for Hadoop must recognize the tokens and data types of the tokens that become columns of a data frame. Oracle R Advanced Analytics for Hadoop uses R's facilities to parse and interpret tokens in input row streams. If missing values are not represented using R's "NA" token, they can be explicitly identified by the `na.strings` argument of `hdfs.attach()`.

Delimited text files with the same key and value separator are preferred over files with a different key delimiter and value delimiter. The Read performance of files with the same key and value delimiter is roughly 2x better than that of files with different key and value delimiter.

The key delimiter and value delimiter can be specified through the `key.sep` and `val.sep` arguments of `hdfs.attach()` or when running a MapReduce job for its output HDFS data.

Binary RData representation is the most performance efficient representation of input data in Oracle R Advanced Analytics for Hadoop. When possible, users are encouraged to use this binary data representation for performance sensitive analytics.

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.

All delimiters are supported, and key and value delimiters can be different.

You can also convert a delimited file into binary format, using the Rdata representation from R, for the best I/O performance.

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.

ORCH Functions for Hive

ORCH provides these conversion functions to help you move data between HDFS and Hive:

```
hdfs.toHive  
hdfs.fromHive
```

ORE Functions for Hive

You can connect to Hive and analyze and transform Hive table objects using R functions that have an `ore` prefix, such as `ore.connect`. If you are also using Oracle R Enterprise, then you will 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.

Note:

For information about requirements and instructions to set up and use Oracle R Enterprise, refer to Oracle R Enterprise library at: https://docs.oracle.com/cd/E83411_01/index.htm.

For example, the `ore.connect(type="HIVE")` establishes a connection with the default HIVE database `ore.hiveOptions(dbname='dbtmp')` and allows you to change the default database, while `ore.showHiveOptions()` allows you to examine the current default HIVE database.

See [Table 8-7](#) for a list of ORE `as.ore.*` and `is.ore.*` functions.

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`.

The following example shows simple data preparation and processing.

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, you will receive 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>
```

2. Identify the column with an unsupported data type:

```
hive> describe table_name;
```

3. View the data in the column:

```
hive> select column_name from table_name;
```

4. Create a table for the converted data, using only supported data types.

5. Copy the data into the new table, using an appropriate conversion tool.

The first example below shows the conversion of an array. The other two examples 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"
```

The following example uses automatic conversion of the `timestamp` data type into string. The data is stored in a table named `t5` with a column named `tstmp`.

Example 8-3 Converting a Timestamp Column

```
hive> select * from t5;

hive> create table t6 (timestmp string);
hive> insert into table t6 SELECT tstamp from t5;
```

The following example 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

The following example 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 included in 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` data type.
- 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.

Oracle R Advanced Analytics for Hadoop Functions

The Oracle R Advanced Analytics for Hadoop functions are described in R Help topics. This section groups them into functional categories and provides brief descriptions.

- [Native Analytical Functions](#)
- [Using the Hadoop Distributed File System \(HDFS\)](#)
- [Using Apache Hive](#)

- [Using Aggregate Functions in Hive](#)
- [Making Database Connections](#)
- [Copying Data and Working with HDFS Files](#)
- [Converting to R Data Types](#)
- [Using MapReduce](#)
- [Debugging Scripts](#)

Native Analytical Functions

The following table describes the native analytic functions.

Table 8-1 Functions for Statistical Analysis

Function	Description
<code>orch.cor</code>	Generates a correlation matrix with a Pearson's correlation coefficients.
<code>orch.cov</code>	Generates a covariance matrix.
<code>orch.getXlevels</code>	Creates a list of factor levels that can be used in the <code>xlev</code> argument of a <code>model.matrix</code> call. It is equivalent to the <code>.getXlevels</code> function in the <code>stats</code> package.
<code>orch.glm</code>	Fits and uses generalized linear models on data stored in HDFS.
<code>orch.kmeans</code>	Perform k-means clustering on a data matrix that is stored as a file in HDFS.
<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.princomp</code>	Analyzes the performance of principal component.
<code>orch.recommend</code>	Computes the top n items to be recommended for each user that has predicted ratings based on the input <code>orch.mahout.lmf.as1</code> model.
<code>orch.sample</code>	Provides the reservoir sampling.
<code>orch.scale</code>	Performs scaling.

Using the Hadoop Distributed File System (HDFS)

The following table describes the functions that execute HDFS commands from within the R environment.

Table 8-2 Functions for Using HDFS

Function	Description
<code>hdfs.cd</code>	Sets the default HDFS path.
<code>hdfs.cp</code>	Copies an HDFS file from one location to another.
<code>hdfs.describe</code>	Returns the metadata associated with a file in HDFS.
<code>hdfs.exists</code>	Verifies that a file exists in HDFS.
<code>hdfs.head</code>	Copies a specified number of lines from the beginning of a file in HDFS.
<code>hdfs.id</code>	Converts an HDFS path name to an R <code>dfs.id</code> object.
<code>hdfs.ls</code>	Lists the names of all HDFS directories containing data in the specified path.
<code>hdfs.mkdir</code>	Creates a subdirectory in HDFS relative to the current working directory.
<code>hdfs.mv</code>	Moves an HDFS file from one location to another.
<code>hdfs.parts</code>	Returns the number of parts composing a file in HDFS.
<code>hdfs.pwd</code>	Identifies the current working directory in HDFS.
<code>hdfs.rm</code>	Removes a file or directory from HDFS.
<code>hdfs.rmdir</code>	Deletes a directory in HDFS.
<code>hdfs.root</code>	Returns the HDFS root directory.
<code>hdfs.setroot</code>	Sets the HDFS root directory.
<code>hdfs.size</code>	Returns the size of a file in HDFS.
<code>hdfs.tail</code>	Copies a specified number of lines from the end of a file in HDFS.

Using Apache Hive

The following table describes the functions available in Oracle R Advanced Analytics for Hadoop for use with Hive. .

Table 8-3 Functions for Using Hive

Function	Description
<code>hdfs.fromHive</code>	Converts a Hive table to a HDFS identifier in ORCH.
<code>hdfs.toHive</code>	Converts an HDFS object identifier to a Hive table represented by an <code>ore.frame</code> object.
<code>ore.create</code>	Creates a database table from a <code>data.frame</code> or <code>ore.frame</code> object.
<code>ore.drop</code>	Drops a database table or view.
<code>ore.get</code>	Retrieves the specified <code>ore.frame</code> object.

Table 8-3 (Cont.) Functions for Using Hive

Function	Description
<code>ore.pull</code>	Copies data from a Hive table to an R object.
<code>ore.push</code>	Copies data from an R object to a Hive table.
<code>ore.recode</code>	Replaces the values in an <code>ore.vector</code> object.

Related Topics

- [ORE Functions for Hive](#)

Using Aggregate Functions in Hive

The following table describes the aggregate functions from the OREstats package that Oracle R Advanced Analytics for Hadoop supports for use with Hive data.

Table 8-4 Oracle R Enterprise Aggregate Functions

Function	Description
<code>aggregate</code>	Splits the data into subsets and computes summary statistics for each subset.
<code>fivenum</code>	Returns Tukey's five-number summary (minimum, lower hinge, median, upper hinge, and maximum) for the input data.
<code>IQR</code>	Calculates an interquartile range.
<code>median</code>	Calculates a sample median.
<code>quantile</code>	Generates sample quantiles that correspond to the specified probabilities.
<code>sd</code>	Calculates the standard deviation.
<code>var</code> ¹	Calculates the variance.

¹ For vectors only

Making Database Connections

The following table describes the functions for establishing a connection to Oracle Database.

Table 8-5 Functions for Using Oracle Database

Function	Description
<code>orch.connect</code>	Establishes a connection to Oracle Database.
<code>orch.connected</code>	Checks whether Oracle R Advanced Analytics for Hadoop is connected to Oracle Database.
<code>orch.dbcon</code>	Returns a connection object for the current connection to Oracle Database, excluding the authentication credentials.
<code>orch.dbinfo</code>	Displays information about the current connection.

Table 8-5 (Cont.) Functions for Using Oracle Database

Function	Description
<code>orch.disconnect</code>	Disconnects the local R session from Oracle Database.
<code>orch.reconnect</code>	Reconnects to Oracle Database with the credentials previously returned by <code>orch.disconnect</code> .

Copying Data and Working with HDFS Files

The following table describes the functions for copying data between platforms, including R data frames, HDFS files, local files, and tables in an Oracle database.

Table 8-6 Functions for Copying Data

Function	Description
<code>hdfs.attach</code>	Copies data from an unstructured data file in HDFS into the R 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.
<code>hdfs.download</code>	Copies a file from HDFS to the local file system.
<code>hdfs.get</code>	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 <code>val1</code> and <code>val2</code> are assigned.
<code>hdfs.pull</code>	Copies data from HDFS into an Oracle database. This operation requires authentication by Oracle Database. See <code>orch.connect</code> .
<code>hdfs.push</code>	Copies data from an Oracle database to HDFS. This operation requires authentication by Oracle Database. See <code>orch.connect</code> .
<code>hdfs.put</code>	Copies data from an R in-memory object (<code>data.frame</code>) to HDFS. All data attributes, like column names and data types, are stored as metadata with the data.
<code>hdfs.sample</code>	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.
<code>hdfs.upload</code>	Copies a file from the local file system into HDFS.
<code>is.hdfs.id</code>	Indicates whether an R object contains a valid HDFS file identifier.

Converting to R Data Types

The following table describes functions for converting and testing data types. The Oracle R Enterprise `OREbase` package provides these functions.

Table 8-7 Functions for Converting and Testing Data Types

Function	Description
<code>as.ore</code>	Coerces an in-memory R object to an ORE object.
<code>as.ore.character</code>	Coerces an in-memory R object to an ORE character object.
<code>as.ore.date</code>	Coerces an in-memory R object to an ORE date object.
<code>as.ore.datetime</code>	Coerces an in-memory R object to an ORE datetime object.
<code>as.ore.difftime</code>	Coerces an in-memory R object to an ORE difftime object.
<code>as.ore.factor</code>	Coerces an in-memory R object to an ORE factor object.
<code>as.ore.frame</code>	Coerces an in-memory R object to an ORE frame object.
<code>as.ore.integer</code>	Coerces an in-memory R object to an ORE integer object.
<code>as.ore.list</code>	Coerces an in-memory R object to an ORE list object.
<code>as.ore.logical</code>	Coerces an in-memory R object to an ORE logical object.
<code>as.ore.matrix</code>	Coerces an in-memory R object to an ORE matrix object.
<code>as.ore.numeric</code>	Coerces an in-memory R object to an ORE numeric object.
<code>as.ore.object</code>	Coerces an in-memory R object to an ORE object.
<code>as.ore.vector</code>	Coerces an in-memory R object to an ORE vector object.
<code>is.ore</code>	Tests whether the specified value is an object of a particular Oracle R Enterprise class.
<code>is.ore.character</code>	Tests whether the specified value is a character.
<code>is.ore.date</code>	Tests whether the specified value is a date.
<code>is.ore.datetime</code>	Tests whether the specified value is a datetime type.
<code>is.ore.difftime</code>	Tests whether the specified value is a difftime type.
<code>is.ore.factor</code>	Tests whether the specified value is a factor.
<code>is.ore.frame</code>	Tests whether the specified value is a frame.
<code>is.ore.integer</code>	Tests whether the specified value is an integer.
<code>is.ore.list</code>	Tests whether the specified value is a list.
<code>is.ore.logical</code>	Tests whether the specified value is a logical type.
<code>is.ore.matrix</code>	Tests whether the specified value is a matrix.
<code>is.ore.numeric</code>	Tests whether the specified value is numeric.
<code>is.ore.object</code>	Tests whether the specified value is an object.
<code>is.ore.vector</code>	Tests whether the specified value is a vector.

Using MapReduce

The following table describes functions that you use when creating and running MapReduce programs.

Table 8-8 Functions for Using MapReduce

Function	Description
<code>hadoop.exec</code>	Starts the Hadoop engine and sends the mapper, reducer, and combiner R functions for execution. You must load the data into HDFS first.
<code>hadoop.jobs</code>	Lists the running jobs, so that you can evaluate the current load on the Hadoop cluster.
<code>hadoop.run</code>	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 <code>hadoop.run</code> first copies the data there.
<code>orch.dryrun</code>	Switches the execution platform between the local host and the Hadoop cluster. No changes in the R code are required for a dry run.
<code>orch.export</code>	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.
<code>orch.keyval</code>	Outputs key-value pairs in a MapReduce job.
<code>orch.keyvals</code>	Outputs a set of key-value pairs in a MapReduce job.
<code>orch.pack</code>	Compresses one or more in-memory R objects that the mappers or reducers must write as the values in key-value pairs.
<code>orch.tempPath</code>	Sets the path where temporary data is stored.
<code>orch.unpack</code>	Restores the R objects that were compressed with a previous call to <code>orch.pack</code> .
<code>orch.create.parttab</code>	Enables partitioned Hive tables to be used with ORCH MapReduce framework.

Debugging Scripts

The following table lists the functions available to help you debug your R program scripts.

Table 8-9 Functions for Debugging Scripts

Function	Description
<code>orch.dbg.lasterr</code>	Returns the last error message.
<code>orch.dbg.off</code>	Turns off debugging mode.
<code>orch.dbg.on</code>	Turns on debugging mode, which prints out the interactions between Hadoop and Oracle R Advanced Analytics for Hadoop including the R commands.
<code>orch.dbg.output</code>	Directs the output from the debugger.
<code>orch.version</code>	Identifies the version of the ORCH package.
<code>orch.debug</code>	Enables R style debugging of MapReduce R scripts.

Demos of Oracle R Advanced Analytics for Hadoop Functions

Oracle R Advanced Analytics for Hadoop provides an extensive set of demos, which you can access in the same way as any other R demos.

The `demo` function lists the functions available in ORCH:

```
R> demo(package="ORCH")
Demos in package 'ORCH':

hdfs_cpmv           ORCH's copy and move APIs
hdfs_datatrans     ORCH's HDFS data transfer APIs
hdfs_dir           ORCH's HDFS directory manipulation APIs
hdfs_putget        ORCH's get and put API usage
hive_aggregate     Aggregation in HIVE
hive_analysis      Basic analysis & data processing operations
hive_basic         Basic connectivity to HIVE storage
hive_binning       Binning logic
hive_columnfns     Column function
hive_nulls         Handling of NULL in SQL vs. NA in R
.
.
.
```

To run a demo from this list, use this syntax:

```
demo("demo_name", package="ORCH")
```

For example, this package runs the Hive binning demo:

```
R> demo("hive_binning", package = "ORCH")

demo('hive_binning', package = 'ORCH')

demo(hive_binning)
---- ~~~~~

> #
> #   ORACLE R CONNECTOR FOR HADOOP DEMOS
> #
> #   Name: hive_binning.R
> #   Description: Demonstrates binning logic in R
> #
> #
.
.
.
```

If an error occurs, 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*
# hdfs dfs -rm -r /tmp/orch*
```


Upon completion run these:

1. `hadoop.exec` to cleanup or remove all empty `part` files and Hadoop log files.
2. `hadoop.run` to allow overwriting of HDFS objects with the same name.

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 the Help topic for `orch.connect`.

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.

Part V

Oracle DataSource for Apache Hadoop

This part describes Oracle DataSource for Apache Hadoop (OD4H) storage handler for Oracle Database. It contains the following chapters:

- [Oracle DataSource for Apache Hadoop \(OD4H\)](#)

9

Oracle DataSource for Apache Hadoop (OD4H)

Oracle DataSource for Apache Hadoop (OD4H) allows direct, fast, parallel, secure and consistent access to master data in Oracle Database using Spark SQL via Hive metastore. This chapter discusses Oracle DataSource for Apache Hadoop (OD4H) in the following sections:

- [Operational Data, Big Data and Requirements](#)
- [Overview of Oracle DataSource for Apache Hadoop \(OD4H\)](#)
- [How Does OD4H Work?](#)
- [Features of OD4H](#)
- [Using Hive SQL with OD4H](#)
- [Using Spark SQL with OD4H](#)
- [Writing Back To Oracle Database](#)

Operational Data, Big Data and Requirements

The common data architecture in most companies nowadays generally comprises of the following components:

- Oracle Database(s) for operational, transactional, and master data, that is shared business object such as customers, products, employees and so on
- Big Data

Hadoop applications such as Master Data Management (MDM), Events processing, and others, need access to data in both Hadoop storages (such as HDFS and NoSQL Database as a landing point for weblogs, and so on) and Oracle Database (as the reliable and auditable source of truth). There are two approaches to process such data that reside in both Hadoop storage and Oracle Database:

- ETL Copy using tools such as Oracle's Copy to BDA
- Direct Access using Oracle Big Data SQL and Oracle DataSource for Apache Hadoop (OD4H).

In this chapter, we will discuss Oracle DataSource for Apache Hadoop (OD4H).

Overview of Oracle DataSource for Apache Hadoop (OD4H)

Oracle DataSource for Apache Hadoop (OD4H) is the storage handler for Oracle Database that uses HCatalog and InputFormat.

This section discusses the following concepts:

- [Opportunity with Hadoop 2.x](#)

- [Oracle Tables as Hadoop Data Source](#)
- [External Tables](#)

Opportunity with Hadoop 2.x

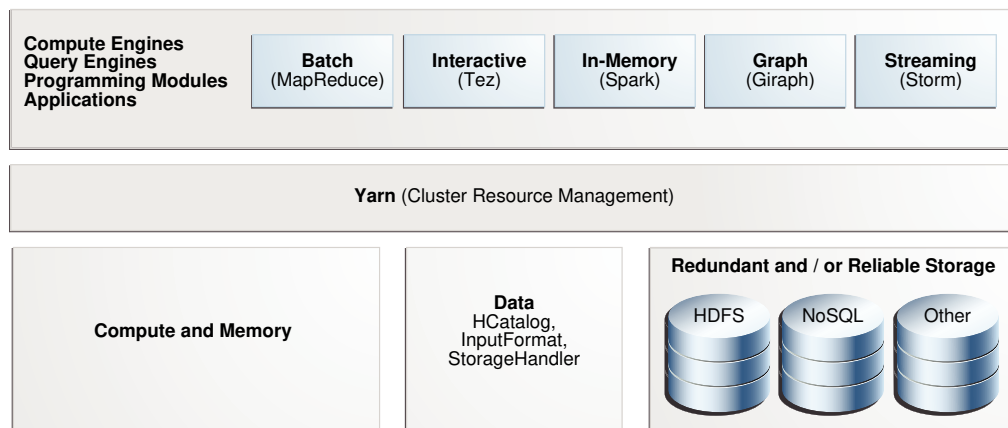
Hadoop 2.x architecture decouples compute engines from cluster resources management and storages. It enables:

- A variety of SQL query engines. For instance, Hive SQL, Spark SQL, Big Data SQL, and so on.
- A variety of programmatic compute engines. For instance, MapReduce, Pig, Storm, Solr, Cascading, and so on.
- Elastic allocation of compute resources (CPU, memory) through YARN.
- A variety of data stores such as HDFS, NoSQL, as well as remote storages through HCatalog, InputFormat, OutputFormat and StorageHandler interfaces.

Oracle DataSource for Apache Hadoop (OD4H) is the storage handler for Oracle Database that uses HCatalog and InputFormat.

Following is an illustration of Hadoop 2.0 Architecture:

Figure 9-1 Hadoop 2.0 Architecture



Oracle Tables as Hadoop Data Source

OD4H enables current and ad-hoc querying. This makes querying data faster and more secure. You can query data directly and retrieve only the data that you need, when you need it.

OD4H also provides Oracle's end-to-end security. This includes Identity Management, Column Masking, and Label and Row Security.

OD4H also furnishes direct access for Hadoop and Spark APIs such as Pig, MapReduce and others.

External Tables

External Tables turn Oracle tables into Hadoop and/or Spark datasources. The DDL for declaring External Tables is as follows:

```
CREATE[TEMPORARY] EXTERNAL TABLE [IF NOT EXISTS] [db_name.]table_name
[(col_name data_type [COMMENTcol_comment],...)]
[COMMENT table_comment]
STORED BY 'oracle.hcat.osh.OracleStorageHandler' [WITHSERDEPROPERTIES(...)]
[TBLPROPERTIES (property_name=property_value,...)]
```

```
data_type
|SMALLINT
|INT
|BIGINT
|BOOLEAN
|FLOAT
|DOUBLE
|STRING
|BINARY
|TIMESTAMP
|DECIMAL
|DECIMAL(precision,scale)
|VARCHAR
|CHAR
```

See Also:

Refer the following link for Hive External Table syntax <https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DDL#LanguageManualDDL-CreateTable>

Note:

Oracle supports only primitive types.

The following table shows the mappings between Oracle and Hive types.

Oracle Data Type	Hive Data Type
NUMBER	INT when the scale is 0 and the precision is less than 10. BIGINT when the scale is 0 and precision is less than 19. DECIMAL when the scale is greater than 0 or the precision is greater than 19.
CLOB	STRING

NCLOB	
BINARY_DOUBLE	DOUBLE
BINARY_FLOAT	FLOAT
BLOB	BINARY
CHAR	CHAR
NCHAR	
VARCHAR2	VARCHAR
NVARCHAR2	
ROWID	BINARY
UROWID	
DATE	TIMESTAMP
TIMESTAMP	TIMESTAMP
TIMESTAMPtz	Unsupported
TIMESTAMPtz	
RAW	BINARY

The properties of external tables can be described as follows:

TBLPROPERTIES

Property	Use
oracle.hcat.osh.columns.mapping	Comma separated list to specify mapping between Hive columns and Oracle table columns. All external tables using OracleStorageHandler must define this.
mapreduce.jdbc.url	Connection URL to connect to the database
mapreduce.jdbc.username	Connection user name to connect to the database
mapreduce.jdbc.password	Connection password to connect to the database
mapreduce.jdbc.input.table.name	Oracle table name
mapreduce.jdbc.input.conditions	To be used for querying the database. Must be used for query pushdown.
mapreduce.jdbc.input.query	To be used for querying the database. Query should be used only when a subset of the columns is selected.
mapreduce.jdbc.input.orderby	ORDER BY clause to be specified for pushing ordering to the database.

Property	Use
oracle.hcat.osh.splitterKind	To be used to specify how OracleStorageHandler must create splits, so that they are a good match for the physical structure of the target table in Oracle Database. The splitter kind applicable could be SINGLE_SPLITTER, PARTITION_SPLITTER, ROW_SPLITTER, BLOCK_SPLITTER.
oracle.hcat.osh.rowsPerSplit	Used only when ROW_SPLITTER splitterKind is applied on the table. Represents Number of rows per split for LIMIT_RANGE splitter. Default is 1000
oracle.hcat.osh.authentication	Authentication method used to connect to Oracle Database. Can be SIMPLE (default), ORACLE_WALLET, KERBEROS
sun.security.krb5.principal	Kerberos principal. Used only when KERBEROS authentication is applied.
oracle.hcat.osh.kerb.callback	Callback for Kerberos authentication. Used only when Kerberos authentication is applied.
oracle.hcat.osh.maxSplits	Maximum number of splits for any splitter kind
oracle.hcat.osh.useChunkSplitter	Use chunk based ROW_SPLITTER and BLOCK_SPLITTER that use DBMS_PARALLEL_EXECUTE package to divide table into chunks that will map to hadoop splits. The default value is set to 'true'.
oracle.hcat.osh.chunkSQL	Used by CUSTOM_SPLITTER to create splits. The SQL string should be a SELECT statement that returns range of each chunk must have two columns: start_id and end_id The columns must be of ROWID type.
oracle.hcat.osh.useOracleParallelism	When configured, parallel queries will be executed while fetching rows from Oracle. Default value: 'false'
oracle.hcat.osh.fetchSize	JDBC fetchsize for generated select queries used to fetch rows. Default value: 10 (set by Oracle JDBC Driver)

 **Note:**

In addition to the above, any JDBC connection properties (oracle.jdbc.* and oracle.net.*) can be specified as TBLPROPERTIES. They will be used while establishing connection to Oracle Database using JDBC driver.

 **Note:**

Oracle DataSource for Apache Hadoop (OD4H) works with Oracle View and Oracle Tables.

SERDE PROPERTIES

Property	Use
oracle.hcat.osh.columns.mapping	All external tables using OracleStorageHandler must define this. Its a comma separated list to specify mapping between hive columns (specified in create table) and oracle table columns. WITHSERDEPROPERTIES also enables the external table definition to refer only to select columns in the actual Oracle table. In other words, not all columns from the Oracle table need to be part of the Hive external table. The ordering of oracle columns in the mapping is the same as ordering of hive columns specified in create table.

List of jars in the OD4H package

Oracle DataSource for Apache Hadoop (OD4H) contains the following list of jars.

OD4H consists of the following list of jars.

Table 9-1 List of jars in OD4H

Name of JAR	Use
osh.jar	Contains OracleStorageHandler Implementation
ojdbc8.jar	An OD4H specific JDBC driver (which is optimized with internal calls), used by Spark or Hadoop tasks to connect to the database.
ucp.jar	For creating connection pools in OracleStorageHandler
oraclepki103.jar, osdt_core.jar, osdt_cert.jar, osdt_jce.jar	For Oracle Wallet authentication
orai18n.jar	Oracle Globalization Support
xdb.jar	Oracle XDB jar

How does OD4H work?

Oracle DataSource for Apache Hadoop (OD4H) does not require creating a new table. You can start working with OD4H using the following steps:

1. Create a new Oracle table, or, reuse an existing table.
2. Create the Hive DDL for creating the external table referencing the Oracle Table.
3. Issue HiveSQL, SparkSQL, or other Spark/Hadoop queries and API calls.

The following sections show how to create a new Oracle Database Table, and a Hive DDL:

- [Create a New Oracle Database Table](#)

- [Hive DDL](#)
- [Creating External Table in Hive](#)

Create a new Oracle Database Table or Reuse an Existing Table

Here is an illustration of a partitioned Oracle table that we will use to demo how partition pruning works:

```
1. CREATE TABLE EmployeeData ( Emp_ID NUMBER,
    First_Name VARCHAR2(20),
    Last_Name VARCHAR2(20),
    Job_Title VARCHAR2(40),
    Salary NUMBER)
PARTITION BY RANGE (Salary)
( PARTITION salary_1 VALUES LESS THAN (60000)
  TABLESPACE tsa
, PARTITION salary_2 VALUES LESS THAN (70000)
  TABLESPACE tsb
, PARTITION salary_3 VALUES LESS THAN (80000)
  TABLESPACE tsc
, PARTITION salary_4 VALUES LESS THAN (90000)
  TABLESPACE tsd
, PARTITION salary_5 VALUES LESS THAN (100000)
  TABLESPACE tse
);
```

Note:

You can use this syntax for table creation, in the following examples listed in this Book.

2. Issue queries from Hive, Spark, or any other Hadoop models (including joins with local Hive Tables.)

Hive DDL

In this example, we will associate two Hive external tables to the same Oracle table, using two different split patterns:

- SIMPLE_SPLITTER
- PARTITION_SPLITTER

Note:

It is possible that the external table has fewer columns than the base Oracle table.
Since columns can have different names, use `TBLPROPERTY` for mapping with the base table.

In the following examples, we are using the following variables:

```

connection_string = jdbc:oracle:thin:@localhost:1521/<servicename>

oracle_user=od4h

oracle_pwd=od4h

```

The following command creates a Hive external table with the default split pattern, that is SIMPLE_SPLITTER.

```

CREATE EXTERNAL TABLE EmployeeDataSimple (
  Emp_ID int,
  First_Name string,
  Last_Name string,
  Job_Title string,
  Salary int
)
STORED BY 'oracle.hcat.osh.OracleStorageHandler'
WITH SERDEPROPERTIES (
  'oracle.hcat.osh.columns.mapping' =
  'Emp_ID,First_Name,Last_Name,Job_Title,Salary')
TBLPROPERTIES (
  'mapreduce.jdbc.url' = '${hiveconf:jdbc:oracle:thin:@localhost:1521/<servicename>}',
  'mapreduce.jdbc.username' = '${hiveconf:od4h}',
  'mapreduce.jdbc.password' = '${hiveconf:od4h}',
  'mapreduce.jdbc.input.table.name' = 'EmployeeData'
);

```

The following example creates a Hive external table using PARTITION_SPLITTER.

```

DROP TABLE EmployeeDataPartitioned;
CREATE EXTERNAL TABLE EmployeeDataPartitioned (
  Emp_ID int,
  First_Name string,
  Last_Name string,
  Job_Title string,
  Salary int
)
STORED BY 'oracle.hcat.osh.OracleStorageHandler'
WITH SERDEPROPERTIES (
  'oracle.hcat.osh.columns.mapping' =
  'Emp_ID,First_Name,Last_Name,Job_Title,Salary')
TBLPROPERTIES (
  'mapreduce.jdbc.url' = '${hiveconf:jdbc:oracle:thin:@localhost:1521/<servicename>}',
  'mapreduce.jdbc.username' = '${hiveconf:od4h}',
  'mapreduce.jdbc.password' = '${hiveconf:od4h}',
  'mapreduce.jdbc.input.table.name' = 'EmployeeData',
  'oracle.hcat.osh.splitterKind' = 'PARTITIONED_TABLE'
);

```

Create External Tables in Hive

You can create an external table in Hive in the following way:

```

DROP TABLE employees;

CREATE EXTERNAL TABLE employees (
  EMPLOYEE_ID INT,
  FIRST_NAME STRING,
  LAST_NAME STRING,
  SALARY DOUBLE,

```

```
HIRE_DATE    TIMESTAMP,  
JOB_ID       STRING  
)  
  
STORED BY 'oracle.hcat.osh.OracleStorageHandler'  
  
WITH SERDEPROPERTIES (  
  'oracle.hcat.osh.columns.mapping' =  
'employee_id,first_name,last_name,salary,hire_date,job_id')  
  
TBLPROPERTIES (  
  'mapreduce.jdbc.url' = 'jdbc:oracle:thin:@localhost:1521:orcl',  
  'mapreduce.jdbc.username' = 'hr',  
  'mapreduce.jdbc.password' = 'hr',  
  'mapreduce.jdbc.input.table.name' = 'EMPLOYEES'  
);
```

 **Note:**

Ensure that `ucp.jar`, `ojdbc8.jar` and `osh.jar` are present in the Hive CLASSPATH, for using OD4H. This is pre-configured on BDA. .

To learn more about CLASSPATH and other Hive configuration properties, refer the following sources:

<https://cwiki.apache.org/confluence/display/Hive/LanguageManual+Cli>

<https://cwiki.apache.org/confluence/display/Hive/Configuration+Properties>

For Cloudera distribution, refer to the steps for Cloudera Manager here:

https://www.cloudera.com/documentation/enterprise/5-14-x/topics/cm_mc_hive_udf.html For other distributions, refer to the respective documentation on adding additional jars to Hive/HiveServer2 environment.

Features of OD4H

The following topics discuss features of OD4H.

- [Performance and Scalability Features](#)
- [Security Features](#)
- [Using Hive SQL with OD4H](#)
- [Using Spark SQL with OD4H](#)

Performance And Scalability Features

Following sections discuss the performance and scalability features of OD4H:

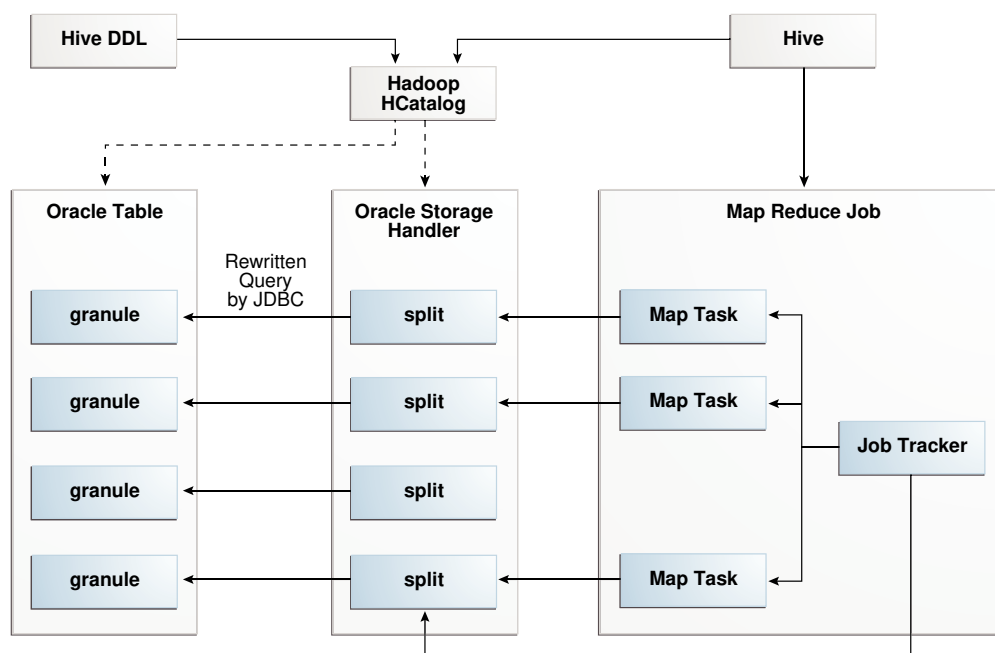
- [Splitters](#)
- [Predicate Pushdown](#)
- [Projection Pushdown](#)
- [Partition Pruning](#)
- [Smart Connection Management](#)

HCatalog stores table metadata from Hive DDL, HiveSQL, Spark SQL and others, then use this metadata while submitting queries.

The Oracle table is divided into granules determined by the `splitterKind` property. These granules are then read into a split by `OracleStorageHandler`, by submitting generated queries.

`OracleStorageHandler` will not have to test all possible query types if the query plan determines which splits need to be scanned.

Figure 9-2 OD4H in a Nutshell



Splitters

While executing a query on a Hive external table through OD4H, the underlying Oracle table is dynamically divided into granules, which correspond to splits on the Hadoop side. Each split is processed by a single map task. With the help of the `ORACLE_SPLITTER_KIND` property, you can specify how the splits are created. This ensures that the splits are a good match for the physical structure of the target table in Oracle Database.

The different kinds of splitters available are:

SINGLE_SPLITTER

Creates one split for the table. Use `SINGLE_SPLITTER` where a single task is sufficient to process the query against the entire table.

ROW_SPLITTER

Limits the number of rows per Split. The default number of rows is 1000. You can specify number of rows by setting the `oracle.hcat.osh.rowsPerSplit` property. The

default value of `oracle.hcat.osh.maxSplits` is 1 when `ROW_SPLITTER` is used. You can increase this value to enable parallel reads.

Based on the values provided in the `rowsPerSplit` property, OD4H will divide tables into splits. If the number of splits obtained is higher than the `maxSplits`, then `maxSplits` property will be used. The rows per split will be divided accordingly.

 **Note:**

`oracle.hcat.osh.rowsPerSplit` is used only by `ROW_SPLITTER` and not any other splitter kind.

BLOCK_SPLITTER

Creates splits based on underlying storage of data blocks. With Block Splitter, you can specify the maximum number of splits to be generated. The default value of `oracle.hcat.osh.maxSplits` is 1, when `BLOCK_SPLITTER` is used. You can increase this value to enable parallel reads. `BLOCK_SPLITTER` requires `SELECT` privilege on the `SYS.DBA.EXTENTS` table, granted to the schema containing the Oracle target table. In the event that this permission does not exist, OD4H will use `SINGLE_SPLITTER`.

 **Note:**

The actual number of splits under `BLOCK_SPLITTER` may be lesser than the value specified in the `oracle.hcat.osh.maxSplits` property. Do not use `BLOCK_SPLITTER` on partitioned tables or Index Organized tables.

 **Note:**

For `ROW_SPLITTER` and `BLOCK_SPLITTER` types, use `oracle.hcat.osh.useChunkSplitter` to specify splitting mechanism. The default property value is `true`. This enables creating chunks corresponding to splits using the `DBMS_PARALLEL_EXECUTE` package. When the property value is `false`, custom SQL is generated for splitting. Since `DBMS_PARALLEL_EXECUTE` can only be used for tables and not views, if `mapreduce.jdbc.input.table.name` points to a view and not a table, then `oracle.hcat.osh.useChunkSplitter` should be set to `false`.

PARTITION_SPLITTER

Creates one split per partition. `PARTITION_SPLITTER` is used by default when the table is partitioned. You can override this setting by specifying `ROW_SPLITTER` in table properties. With `PARTITION_SPLITTER`, the default value of `oracle.hcat.osh.maxSplits` table property is 64.

Following is an illustration of ROW_SPLITTER:

```
DROP TABLE employees;

CREATE EXTERNAL TABLE employees (
  EMPLOYEE_ID INT,
  FIRST_NAME  STRING,
  LAST_NAME   STRING,
  SALARY      DOUBLE,
  HIRE_DATE   TIMESTAMP,
  JOB_ID      STRING
)
STORED BY 'oracle.hcat.osh.OracleStorageHandler'

WITH SERDEPROPERTIES (
  'oracle.hcat.osh.columns.mapping' =
  'employee_id,first_name,last_name,salary,hire_date,job_id')

TBLPROPERTIES (
  'mapreduce.jdbc.url' = 'jdbc:oracle:thin:@localhost:1521:orcl',
  'mapreduce.jdbc.username' = 'hr',
  'mapreduce.jdbc.password' = 'hr',
  'mapreduce.jdbc.input.table.name' = 'EMPLOYEES',
  'oracle.hcat.osh.splitterKind' = 'ROW_SPLITTER',
  'oracle.hcat.osh.rowsPerSplit' = '1500'
);
```

CUSTOM_SPLITTER

Use CUSTOM_SPLITTER if you want to provide a custom split generation mechanism. You can do this using CUSTOM_SPLITTER through oracle.hcat.osh.splitterKind property and a SELECT statement that emits ROWIDs corresponding to start and end of each split in oracle.hcat.osh.chunkSQL.

Choosing a Splitter

SINGLE_SPLITTER is used by default if no splitter is specified in the table properties for Hive external table, and the target Oracle table is not partitioned.

For an unpartitioned table, the default value of oracle.hcat.osh.maxSplits will be 1. For partitioned table, the default value of the same will be 64, and the default splitter will be PARTITION_SPLITTER. The default for maxSplits is set to limit the number of connections to the Oracle server. To increase this limit, you must increase the value of oracle.hcat.osh.maxSplits explicitly in hive table properties.

Use the following guidelines while choosing a splitter kind for a hive external table:

Splitter Kind	Use
SINGLE_SPLITTER	When no parallelism is required.
PARTITION_SPLITTER	Used by default when target table is partitioned

Splitter Kind	Use
BLOCK_SPLITTER	When Oracle user has <code>SELECT</code> privilege on <code>SYS.DBA_EXTENTS</code> , and target table is not partitioned.
ROW_SPLITTER	When Oracle user does not have <code>SELECT</code> privilege on <code>SYS.DBA_EXTENTS</code> .
CUSTOM_SPLITTER	For fine grain control over generated splits.

Predicate Pushdown

Predicate Pushdown is an optimization technique, in which you push predicates (`WHERE` condition) down to be evaluated by Oracle Database at the time of querying. This minimizes the amount of data fetched from Oracle Database to Hive, while performing a query.

Set the configuration property `hive.optimize.ppd` to either `true` or `false` for enabling Predicate Pushdown. The default value on `hive-1.1.0` is set to `true`. Hence, Predicate Pushdown is always performed, unless you want to disable it.

Note:

OD4H does not push down all possible predicates. It considers only the part of the execution plan pertaining to Oracle table declared as external table. OD4H also rewrites sub-queries for the Oracle SQL engine and each split task. At present conditions involving operators `>`, `=`, `<` and `!=` in a single condition over a column (e.g. `key > 10`) or a combination of multiple conditions separated by `AND` (e.g. `key > 10 AND key < 20 AND key != 17`) are pushed down.

Another option to reduce the amount of data fetched from the Oracle Database is to specify a condition at the time of table creation, using `TBLPROPERTY` `mapreduce.jdbc.input.conditions`. For instance:

```
mapreduce.jdbc.input.conditions = 'key > 10 OR key = 0'.
```

This will restrict the rows fetched from Oracle Database whenever any query is performed based on the condition specified. The external table that gets created, is analogous to a view on Oracle Database. This approach is only useful when you want to push down complex predicates that cannot be analyzed and automatically pushed down by OD4H.

 **Note:**

Due to incompatibilities between date and timestamp representation in Hive and Oracle, these columns are not pushed down by default in a query. You can enable this with certain limitations by setting the tableproperty `oracle.hcat.datetime.pushdown` to `true`. When set to `true`, the date representation in the query should be in the form `YYYY-MM-DD` and timestamp should be in the form `"YYYY-MM-DD HH:MM:SS"` with no decimal places. No other date or timestamp representation is supported when `oracle.hcat.datetime.pushdown` is set to `true`.

Projection Pushdown

Projection Pushdown is an optimization technique that fetches only the required columns from Oracle Database when a query is performed. If you want to fetch all columns during a query (not recommended), you can disable it by setting the `hive.io.file.read.all.columns` connection property to `true`. On Hive-1.1.0, this property is `false` by default.

Partition Pruning

If you refer to Employee Data Partition table, the partitions irrelevant to the query are removed from the partition list. This is done by executing an explain plan on the query to obtain the list of partitions and sub-partitions that are relevant to the query.

Table level partition pruning uses table level predicate pushdown, on the other hand partition pruning at the query level uses query level predicate pushdown.

Partition pruning is active when a `SELECT` query is run, in which the `WHERE` clause uses the partitioning key. Following is an example of partition pruning:

To query the partition, where salary is in the above range and prune other partitions, perform the following:

Hive External Table:

```
CREATE EXTERNAL TABLE EmployeeDataPartitioned (
  Emp_ID int,
  First_Name string,
  Last_Name string,
  Job_Title string,
  Salary int
)
STORED BY 'oracle.hcat.osh.OracleStorageHandler'
WITH SERDEPROPERTIES (
  'oracle.hcat.osh.columns.mapping' =
  'Emp_ID,First_Name,Last_Name,Job_Title,Salary')
TBLPROPERTIES (
  'mapreduce.jdbc.url' = '${hiveconf:connection_string}',
  'mapreduce.jdbc.username' = '${hiveconf:oracle_user}',
  'mapreduce.jdbc.password' = '${hiveconf:oracle_pwd}',
  'mapreduce.jdbc.input.table.name' = 'EmployeeData',
  'oracle.hcat.osh.oosKind' = 'PARTITIONED_TABLE'
);
```


The following `SELECT` statement shows how to query the partition, where salary is between 72000 to 78000, and prunes other partitions:

```
select * from EmployeeDataPartitioned where salary > 72000 and salary < 78000;
```

Smart Connection Management

Connection Caching

Each map task runs in its own JVM. Each JVM in turn caches a single connection to the Oracle database that you can reuse within the same query. The Mapper checks the cache before establishing a new connection and caching is not done once the query has completed executing.

Oracle RAC Awareness

JDBC and UCP are aware of various Oracle RAC instances. This can be used to split queries submitted to JDBC. The StorageHandler will depend on listener for load balancing.

Handling Logon Storms

Hadoop allows you to limit the number of mappers attempting to connect to the Database. Hadoop allows you to limit the number of mappers attempting to connect to the Database using `oracle.hcat.osh.maxSplits`. This parameter controls the degree of concurrency. However, subsequent tasks of the same query are guaranteed to query their table granule as per the System Commit Number (SCN) of the query. This ensures consistency of the result sets.

Database Resident Connection Pooling (DRCP)

It is recommended to configure DRCP for OD4H, and limit the maximum number of concurrent connections to the Oracle Database from OD4H.

Configuring Database Resident Connection Pooling

To configure DRCP, use the following steps:

1. Login as SYSDBA.
2. Start the default pool, `SYS_DEFAULT_CONNECTION_POOL` using `DBMS_CONNECTION_POOL.START_POOL` with the default settings.

You can use `DBMS_CONNECTION_POOL.MINSIZE` and `DBMS_CONNECTION_POOL.MAXSIZE` with the default settings.

See Also:

For more information on configuring DRCP Oracle Database Administrator's Guide

Security Features

Following are the security features of OD4H:

Improved Authentication

OD4H uses Oracle JDBC driver for connecting to Oracle Database. It provides all authentication methods supported by Oracle JDBC. OD4H supports authentication through use of basic authentication (username and password), Oracle Wallet, and Kerberos. You can specify the authentication to be used for a table created in Hive, through the `oracle.hcat.osh.authentication` table property. This is useful only for strong authentication.

- Kerberos
- Oracle Wallet
- Basic Authentication



Note:

Oracle recommends using strong authentication such as Kerberos.

The various authentication processes are described with examples as follows:

1. Kerberos

Uses Kerberos credentials of the Hadoop engine process. This principal should have access to the table.



See Also:

[Oracle Database JDBC Developer's Guide](#) for information on configuring database for Kerberos and details of client parameters

You can enable Kerberos configuration on Hive, by adding to `hive-env.sh` the following:

```
export HADOOP_OPTS="$HADOOP_OPTS -Djava.security.krb5.conf=<path to
kerberos configuration>
```

To enable child JVMs to use Kerberos configuration, edit the `mapred-site.xml` to include the following property on all nodes of the cluster:

```
<property><name>mapred.child.java.opts</name> <value>-
Djava.security.krb5.conf=<path to kerberos configuration></value></
property>
```

Enable these configurations on BDA using Cloudera manager..

Following is an illustration of Kerberos authentication:

```
CREATE EXTERNAL TABLE kerb_example (
id DECIMAL,
```

```

name STRING,
salary DECIMAL
)
STORED BY 'oracle.hcat.osh.OracleStorageHandler'
WITH SERDEPROPERTIES (
    'oracle.hcat.osh.columns.mapping' = 'id,name,salary')
TBLPROPERTIES (
'mapreduce.jdbc.url' =
'jdbc:oracle:thin:@(DESCRIPTION=(ADDRESS=(PROTOCOL=tc)
(HOST=adc*****.xxxxxx.com)(PORT=5521))(CONNECT_DATA=
(SERVICE_NAME=project_name.xxx.rdbms.xxxx.com))',
'mapreduce.jdbc.input.table.name' = 'kerb_example',
'mapreduce.jdbc.username' = 'CLIENT@xxxxxx.COM',
'oracle.hcat.osh.authentication' = 'KERBEROS',
'oracle.net.kerberos5_cc_name' = '/tmp/krb5cc_xxxxx',
'java.security.krb5.conf' = '/home/user/kerberos/krb5.conf',
'oracle.hcat.osh.kerb.callback' = 'KrbCallbackHandler',
'sun.security.krb5.principal' = 'CLIENT@xxxxxx.COM'
);

```

The path specified in `oracle.security.krb5.conf` should be accessible to all nodes of the cluster. These paths should also match with the path of the corresponding properties in Oracle Database `sqlnet.ora`. The `keytab` path provided in `sqlnet.ora` should also be accessible from all nodes of the cluster.

If `sun.security.krb5.principal` is not specified, OD4H will attempt to authenticate using default principal in Credential Cache specified by the `oracle.net.kerberos5_cc_name` property.

Note:

The callback will be called only if the principal cannot be authenticated using a ticket obtained from the credential cache specified in `oracle.net.kerberos5_cc_name` property.

A simple callback handler class is described as follows (The callback class must be available to the hive classpath):

```

class KrbCallbackHandler
    implements CallbackHandler{

@Override
public void handle(Callback[] callbacks) throws IOException,
    UnsupportedCallbackException{
    for (int i = 0; i < callbacks.length; i++){
        if (callbacks[i] instanceof PasswordCallback){
            PasswordCallback pc = (PasswordCallback)callbacks[i];
            System.out.println("set password to 'welcome'");
            pc.setPassword((new String("welcome")).toCharArray());
        } else if (callbacks[i] instanceof NameCallback) {
            ((NameCallback)callbacks[i]).setName("client@xxxxxx.COM");
        }else{
            throw new UnsupportedCallbackException(callbacks[i],
                "Unrecognized Callback");
        }
    }
}

```

```

    }
  }
}

```

2. Oracle Wallet

The wallet should be available in the OS environment of each engine process. Following is an illustration of how to add Wallet authentication:

```

CREATE EXTERNAL TABLE wallet_example (
  id DECIMAL,
  name STRING,
  salary DECIMAL
)
STORED BY 'oracle.hcat.osh.OracleStorageHandler'
WITH SERDEPROPERTIES (
  'oracle.hcat.osh.columns.mapping' = 'id,name,salary')
TBLPROPERTIES (
  'mapreduce.jdbc.url' = 'jdbc:oracle:thin:@inst1',
  'mapreduce.jdbc.input.table.name' = 'wallet_example',
  'oracle.hcat.osh.authentication' = 'ORACLE_WALLET',
  'oracle.net.tns_admin' = '/scratch/user/view_storage/user_project6/
work',
  'oracle.net.wallet_location' = '/scratch/user/view_storage/
user_project6/work'
);

```

Note:

The paths specified in `oracle.net.tns_admin` and `oracle.net.wallet_location` should be accessible from all nodes of the cluster.

See Also:

Managing the Secure External Password Store for Password Credentials section in the *Oracle Database Security Guide*.

3. Basic Authentication (for demo purposes only)

This is stored in HCatalog `TBLPROPERTIES` or supplied on HiveQL `SELECT` statement.

When Basic Authentication is used, the username and password for Oracle Schema is specified in Hive external Table properties.

Note:

Oracle does not recommend this in the production environment, since the password is stored in clear in HCatalog.

Use HiveQL with OD4H

HiveQL is a SQL like language provided by Hive. It can be used to query hive external tables created using OD4H.

You can run the Resource Manager web interface in your browser (<http://bigdatalite.localdomain:8088/cluster>), to track the status of a running query on BDA.

You can also see the logs of a query in Cloudera Manager, which also indicates the actual query sent to Oracle Database corresponding to your query on HiveQL. Hive and OD4H use slf4j framework for logging. You can control logging level for OD4H related classes using logging configuration techniques of Hive.

Use Spark SQL with OD4H

Spark SQL enables relational queries expressed in SQL and HiveSQL to be executed using Spark. Spark SQL allows you to mix SQL queries with programmatic data manipulations supported by RDDs (Resilient Distributed Datasets) in Java, Python and Scala, with a single application.

Spark SQL enables you to submit relational queries using SQL or HiveQL. You can also use it to query external tables created using OD4H.

Perform the following steps to configure Spark-SQL on BigDataLite-4.2 VM, before running queries:

1. Add `ojdbc7.jar` and `osh.jar` to `CLASSPATH` in `/usr/lib/spark/bin/compute-classpath.sh`

```
CLASSPATH="$CLASSPATH:/opt/oracle/od4h/lib/osh.jar"  
CLASSPATH="$CLASSPATH:/opt/oracle/od4h/lib/ojdbc7.jar"
```

2. Edit `SPARK_HOME` in `/usr/lib/spark/conf/spark-env.sh`

```
export SPARK_HOME=/usr/lib/spark:/etc/hive/conf
```

3. You will need to specify additional environment variables in `/usr/lib/spark/conf/spark-env.sh`.

The Hive related variables that need to be added are marked in bold. The file already contains Hadoop related environment variables.

```
export DEFAULT_HADOOP=/usr/lib/hadoop  
export DEFAULT_HIVE=/usr/lib/hive  
export DEFAULT_HADOOP_CONF=/etc/hadoop/conf  
export DEFAULT_HIVE_CONF=/etc/hive/conf  
export HADOOP_HOME=${HADOOP_HOME:-$DEFAULT_HADOOP}  
export HADOOP_HDFS_HOME=${HADOOP_HDFS_HOME:-${HADOOP_HOME}/../hadoop-hdfs}  
export HADOOP_MAPRED_HOME=${HADOOP_MAPRED_HOME:-${HADOOP_HOME}/../hadoop-mapreduce}  
export HADOOP_YARN_HOME=${HADOOP_YARN_HOME:-${HADOOP_HOME}/../hadoop-yarn}  
export HADOOP_CONF_DIR=${HADOOP_CONF_DIR:-$DEFAULT_HADOOP_CONF}  
export HIVE_CONF_DIR=${HIVE_CONF_DIR:-$DEFAULT_HIVE_CONF}
```

```
CLASSPATH="$CLASSPATH:$HIVE_CONF_DIR"  
CLASSPATH="$CLASSPATH:$HADOOP_CONF_DIR"
```

```
if [ "x" != "x$YARN_CONF_DIR" ]; then
    CLASSPATH="$CLASSPATH:$YARN_CONF_DIR"
fi

# Let's make sure that all needed hadoop libs are added properly
CLASSPATH="$CLASSPATH:$HADOOP_HOME/client/*"
CLASSPATH="$CLASSPATH:$HIVE_HOME/lib/*"
CLASSPATH="$CLASSPATH:${HADOOP_HOME}/bin/hadoop classpath)"
```

Once configured, you can run some sample queries on spark SQL using scripts included in `demo:/shell/*QuerySpark.sh`. By default, Spark prints queries on the console. To modify this behavior you can edit the spark logging configuration file `/usr/lib/spark/conf/log4j.properties`.

The log printed by OracleRecordReader shows the actual query sent to Oracle Database, as follows:

```
15/03/18 10:36:08 INFO OracleRecordReader: Reading records from Oracle Table
using Query: SELECT FIRST_NAME, LAST_NAME, EMP_ID FROM EmployeeData
```

Writing Back to Oracle Database

In the typical use case for OD4H, you store the result sets of Hive or Spark SQL queries back to Oracle Database. OD4H implements `OutputFormat` to enable you to write back to an Oracle Database table from Hadoop.

After the data is inserted into an Oracle Database table, you can then use your favorite business intelligence tools for further data mining

The following query is from the OD4H demo code samples. It demonstrates writing back to an external table called `EmployeeBonusReport`.

Example 9-1 Writing Hive or Spark Result Sets Back to Oracle Database

```
INSERT INTO EmployeeBonusReport
    SELECT EmployeeDataSimple.First_Name,
    EmployeeDataSimple.Last_Name,
    EmployeeBonus.bonus
    FROM EmployeeDataSimple JOIN EmployeeBonus ON
        (EmployeeDataSimple.Emp_ID=EmployeeBonus.Emp_ID)
    WHERE salary > 70000 and bonus > 7000"
```

Part VI

Appendices

This section contains the following appendices.

- [OraLoaderMetadata Utility](#)
- [Using Oracle's Hive Storage Handler for Kafka to Create a Hive External Table for Kafka Topics](#)
- [Oracle Big Data Connectors Accessibility Recommendations](#)
- [Apache License](#)
- [Additional Big Data Connector Resources](#)

A

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:

```
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.

The following example shows how to store the target table metadata in an XML file.

Example A-1 Generating Table Metadata

Run the `OraLoaderMetadata` utility:


```
$ java -cp '/tmp/oraloader-<version>-h2/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 <version> - Production
```

```
Copyright (c) 2011, 2015, 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 <version> - Production

Copyright (c) 2011, 2016, 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>
  .
  .
  .
```

D

Oracle Big Data Connectors Accessibility Recommendations

This section provides some tips on using screen readers and screen magnifiers with these tools.

Tips on Using Screen Readers and Braille Displays

The following are some suggestions on using text-to-speech output and braille displays.

- Use a character mode based terminal such as Putty or Cygwin. Do not use an X-Windows-based VNC.
- In the settings of the terminal software, set the cursor type to "block" cursor, not blinking or flashing.
- The output of the certain commands can generate a significant amount of information and might spill off the terminal window, and the virtual window or braille display. In those, cases consider piping the output of a command through `more` in order to break the output into pages. You can then use the space bar key to page through the output.
- A few recommended screen reader settings include the following (JAWS is used here just as an example):
 - Set the JAWS cursor to "All". Use the key combination of `Insert + s` until you hear "All".
 - You may need to turn off virtual cursor. If you are using JAWS, you can do this using the key combination of `Insert + z`.
 - Use the virtual window to capture text. If you are using JAWS, you can do this using the key combination of `Insert + Alt + w`.

Tips on Using Screen Magnifiers

Examples of screen magnifiers include ZoomText, MAGic, and SuperNova.

- Screen magnifiers can support both character-based terminals and X-Window-based VNC.
- If you are using the screen reader function of the screen magnifier (ZoomText screen reader), then you should use a character-based terminal as described above.
- If you are using a VNC, decide your preference for a window display, for example, TWM or ICE. A display setting for ICE can be done with the following:

```
vncserver -geometry 1600*950 :2
```

1600*950 specifies the display size, and :2 specifies the VNC display number.

F

Recent Change History

Changes in Oracle Big Data Connectors Release 4.12

The following are changes in Oracle Big Data Connectors for Release 4.12.

Software Updates in This Release

Connector	Version
Oracle SQL Connector for HDFS (OSCH)	3.8.2
Oracle Loader for Hadoop (OLH)	3.9.2
Oracle Shell for Hadoop Loaders (OHS)	1.3.2
Oracle XQuery for Hadoop (OXH)	4.9.1
Oracle R Advanced Analytics for Hadoop (ORAAH)	2.8.0
Oracle DataSource for Apache Hadoop (OD4H)	1.3.1
Oracle Data Integrator (ODI)	12.2.1.2.6

New Commands for Monitoring OHS Jobs

OHS includes four new commands to monitor jobs and a command to remove jobs.

- `ohsh> show job <job_id>`
Shows detailed information about the job specified by ID.
- `ohsh> show job summary <job_id>`
Shows the performance of the completed job specified by ID.
- `ohsh> show job abstract <job_id>`
Provides a functional description of the job .
- `ohsh> show jobs [failed|running|completed|finished] [extended] [<integer>]`
Shows the last *n* jobs of a particular job status.
 - The first parameter specifies job status. If the status is not specified, all jobs are shown, regardless of job status.
 - The second parameter specifies whether to show details.
 - The third parameter specifies that the last *n* jobs of the specified status should be shown. If *n* is not specified, then all jobs of that status are shown.
- `ohsh> truncate jobs [<integer>]`
Removes the last *n* jobs from the database. If the integer is not specified, the command removes all jobs

 **See Also:**

- See the OSH help for descriptions of all OSH commands:

```
ohsh> help
```

- When OSH is installed in on-premises environments (outside of Oracle Big Data cloud services), edits to the `smartloader-conf.xml` configuration file are required in order to enable these commands.

Other Changes

In earlier releases, usage examples for each of the Oracle Big Data Connectors are automatically installed into an `examples` directory under the home directory for the connector. In this release, the installation zip file for each connector includes an `examples.zip` file which you can unpack when you are ready to start working with the examples.

B

Using Oracle's Hive Storage Handler for Kafka to Create a Hive External Table for Kafka Topics

The Hive storage handler for Kafka enables Hive (as well as Oracle Big Data SQL) to query Kafka topics.

To provide access to Kafka data, you create a Hive external table over the Kafka topics. The Oracle Big Data SQL storage handler that enables Hive to read the Kafka data format is `oracle.hadoop.kafka.hive.KafkaStorageHandler`.

You can use this storage handler to create external Hive tables backed by data residing in Kafka. Big Data SQL can then query the Kafka data through the external Hive tables.

The Hive DDL is demonstrated by the following example, where `topic1` and `topic2` are two topics in Kafka broker whose keys are serialized by Kafka's String serializer and whose values are serialized by kafka's Long serializer.

```
CREATE EXTERNAL TABLE test_table
row format serde 'oracle.hadoop.kafka.hive.KafkaSerDe'
stored by 'oracle.hadoop.kafka.hive.KafkaStorageHandler'
tblproperties('oracle.kafka.table.key.type'='string',
              'oracle.kafka.table.value.type'='long',
              'oracle.kafka.bootstrap.servers'='nshgc0602:9092',
              'oracle.kafka.table.topics'='topic1,topic2');
```

The example below shows the resulting Hive table. The Kafka key, value, offset, topic name, and partitionid are mapped to Hive columns. You can explicitly designate the offset for each topic/partition pair through a WHERE clause in your Hive query.

```
hive> describe test_table;
OK
topic          string          from deserializer
partitionid    int            from deserializer
key            string         from deserializer
value          bigint        from deserializer
offset         bigint        from deserializer
timestamptype  smallInt      from deserializer
timestamp      timestamp     from deserializer
Time taken: 0.084 seconds, Fetched: 7 row(s)
```

The content of the table is a snapshot of the Kafka topics when the Hive query is executed. When new data is inserted into the Kafka topics, you can use the offset column or the timestamp column to track the changes to the topic. The offsets are per

topic/partition. For example, the following query will return new messages after the specified offsets in the where clause for each topic/partition:

```
hive> select * from test_table where (topic="topic1" and partitoinid=0 and
offset > 199) or (topic="topic1" and partitionid=1 and offset > 198) or
(topic="topic2" and partitionid=0 and offset > 177) or (topic="topic2" and
partitionid=1 and offset > 176);
```

You need to keep track of the offsets for all topic/partition. For example, you can use an Oracle table to store these offsets. A more convenient way to keep track of new data is using the timestamp column. You can query data after a specific time point using the following query:

```
hive> select * from test_table where timestamp > '2017-07-12 11:30:00';
```

See the Property Reference section below for descriptions of all table properties

Property Reference

Table B-1 Table Properties of Hive Storage Handler for Kafka

Property Name	Requirement	Description
oracle.kafka.table.topics	Required	A comma-separated list of Kafka topics. Each Kafka topic name must consist of only letters (uppercase and lowercase), numbers, <code>.</code> (dot), <code>_</code> (underscore), and <code>-</code> (minus). The maximum length for each topic name is 249. These topics must have the same serialization mechanisms. The resulting Hive table consists of records from all the topics listed here. A Hive column "topic" will be added and it will be set to the topic name for each record.
oracle.kafka.bootstrap.servers	Required	This property will be translated to the "bootstrap.servers" property for the underlying Kafka consumer. The consumer makes use of all servers, irrespective of which servers are specified here for bootstrapping. This list only impacts the initial hosts used to discover the full set of servers. This list should be in the form <code>host1:port1,host2:port2,...</code> . Since these servers are just used for the initial connection to discover the full cluster membership (which may change dynamically), this list need not contain the full set of servers. For availability reasons, you may want to list more than one server.

Table B-1 (Cont.) Table Properties of Hive Storage Handler for Kafka

Property Name	Requirement	Description
<code>oracle.kafka.table.key.type</code>	Optional	<p>The key type for your record. If unset, then the key part of the Kafka record will be ignored in the Hive row. Only values of "string", "integer", "long", "double", "avro", "avro_confluent" are supported. "string", "integer", "double" and "long" correspond to the built-in primitive serialization types supported by Kafka. If this property is one of these primitive types, then the Kafka key for each record will be mapped to one single Hive Column. If this property is set to "avro" or "avro_confluent", then <code>oracle.kafka.table.key.schema</code> is required. The Kafka key for each record will be deserialized into an Avro Object. If the Avro schema is of record type then each first level field of the record will be mapped to a single Hive column. If the Avro schema is not of Record Type, then it will be mapped to a single Hive Column named "key".</p> <p>The difference between "avro" and "avro_confluent" is that the wire format for the serialization is slightly different. For "avro", the entire bytes array of the key consists of the bytes of avro serialization. For "avro_confluent", the bytes array consists of a magic byte, a version number, then the bytes of avro serialization of the key.</p>
<code>oracle.kafka.table.value.type</code>	Optional	<p>The value type of your record. If unset, then the value part of Kafka record will be ignored in the Hive row. Use of this property is similar to use of <code>oracle.kafka.table.key.type</code>. The difference between them is: when the Avro Schema for Kafka value is not of record type. The whole Avro object will be mapped to a single Hive Column named "value" instead of "key".</p>
<code>oracle.kafka.table.key.writer.schema</code>	Optional	<p>An optional writer schema for the Kafka key's Avro serialization. It's required when the reader schema for the key is different from the schema in which the keys are written to Kafka brokers. It must be the exact schema in which Kafka keys are serialized.</p>
<code>oracle.kafka.table.key.schema</code>	Required when "oracle.kafka.table.key.type" is "avro" or "avro_confluent"	<p>The JSON string for the Kafka key's Avro reader schema. It doesn't need to be exactly the same as the Kafka key's writer Avro schema. As long as the reader schema is compatible with the Kafka key or the converted object from the converter, it is valid. This enables you to rename Hive columns and choose what fields to keep from the Kafka key in the Hive row. If the schema in this property is different from the schema in which the Kafka keys are serialized, then <code>oracle.kafka.table.key.writer.schema</code> is required.</p>
<code>oracle.kafka.table.value.writer.schema</code>	Optional	<p>An optional writer schema for the Kafka value's Avro serialization. Its use is similar to <code>oracle.kafka.table.key.writer.schema</code>.</p>

Table B-1 (Cont.) Table Properties of Hive Storage Handler for Kafka

Property Name	Requirement	Description
oracle.kafka.table.value.schema	Required when "oracle.kafka.table.value.type" is "avro" or "avro_confluent"	The JSON string for the Kafka value's Avro reader schema. Its use is similar to <code>oracle.kafka.table.key.schema</code> .
oracle.kafka.table.extra.columns	Optional, default to "true"	A boolean flag to control whether to include extra Kafka columns: <code>partitionid, offset, timestamptype</code> .
oracle.kafka.chop.partition	Optional, default to false	A Boolean flag to control whether to chop Kafka partitions into smaller chunks. This is useful when the number of Kafka partitions is small and the size of each Kafka partition is large.
oracle.kafka.partition.chunk.size	Optional	When <code>oracle.kafka.chop.partition</code> is true, this property controls the number of Kafka records in each partition chunk. It should be set a value estimated by $(\text{Ideal size of a split}) / (\text{Average size of a Kafka record})$. For example, if the ideal size of a split is 256 MB and the average size of s Kafka record is 256 Bytes, then this property should be set to 1000000.

C

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. (Don't 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 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):

E

Additional Big Data Connector Resources

The following are useful resources for learning about and using Oracle Big Data Connectors.

Oracle Big Data Connector Downloads

The Oracle Technology Network (OTN) provides downloads of the latest versions as well as earlier versions of the [Oracle Big Data Connectors](#).

Oracle Big Data Connector Blogs and Community Forums

[Oracle Blogs](#) includes a number of postings on Oracle Big Data Connectors under the topic *Connecting Hadoop with Oracle*, including the following.

- [Oracle Shell for Hadoop Loaders \(OHSB\)](#), an introduction to OHSB.
- [Copy to Hadoop with OHSB](#), some OHSB examples using the Oracle Big Data Lite VM.
- [Using Oracle SQL Connector for HDFS with Oracle Wallet](#), a simple step-by-step demonstration of how to use Oracle SQL Connector for HDFS with a client-side Oracle Wallet.
- [Oracle SQL Connector for HDFS and Oracle Database System Platforms](#), a post about OSCH support for the various Oracle Database server platforms.

The [Oracle Datasource for Apache Hadoop Community Forum](#) provides blog posts on OD4H and a discussion forum for OD4H users.

Index

Symbols

%*
 put annotation, [5-7](#)
%annotations, [6-5](#), [6-76](#), [6-78](#)
%ora-java
 binding annotation, [5-8](#)
%output annotation, [6-48](#)
%output encoding annotation, [6-83](#)
%output media-type annotation, [6-83](#)
%updating annotation, [5-7](#)

A

access privileges, Oracle Database, [1-13](#)
adapters
 Avro, [6-1](#)
 Oracle NoSQL Database, [6-32](#)
 sequence file, [6-58](#)
 text file, [6-73](#)
 tika, [6-82](#)
 XML file, [6-86](#)
aggregate functions for Hive, [8-14](#)
ALLOW_BACKSLASH_ESCAPING_ANY_CHAR
 ACTER property, [6-21](#)
ALLOW_COMMENTS property, [6-21](#)
ALLOW_NON_NUMERIC_NUMBERS property,
 [6-21](#)
ALLOW_NUMERIC_LEADING_ZEROS property,
 [6-21](#)
ALLOW_SINGLE_QUOTES property, [6-21](#)
ALLOW_UNQUOTED_CONTROL_CHARS
 property, [6-21](#)
ALLOW_UNQUOTED_FIELD_NAMES property,
 [6-21](#)
ALTER SESSION commands, [2-45](#)
analytic functions in R, [8-12](#)
annotations
 Avro collection, [6-3](#)
 equal to Oracle Loader for Hadoop
 configuration properties, [6-30](#)
 for writing to Oracle NoSQL Database, [6-48](#)
 Oracle Database adapter, [6-24](#)
 Oracle NoSQL Database adapter, [6-42](#)
 parsing tika files, [6-83](#)

annotations (*continued*)
 reading from Oracle NoSQL Database, [6-45](#)
 reading sequence files, [6-63](#)
 reading text files, [6-76](#)
 reading XML files, [6-88](#)
 writing text files, [6-78](#)
Apache Hadoop distribution, [1-3](#), [1-27](#)
Apache licenses, [C-4](#)
avro
 compress annotation, [6-5](#)
 file annotation, [6-5](#)
 put annotation, [6-5](#)
 schema annotation, [6-5](#)
 schema-file annotation, [6-5](#)
 schema-kv annotation, [6-5](#), [6-42](#), [6-45](#), [6-48](#)
Avro
 annotations for reading, [6-3](#)
 annotations for writing, [6-5](#)
Avro array,
 reading as XML, [6-11](#)
Avro file adapter, [6-1](#)
 examples, [6-6](#)
 reading Avro as XML, [6-8](#)
 writing XML as Avro, [6-13](#)
Avro files
 collection annotations, [6-3](#)
 collection function, [6-3](#)
 converting text to, [6-6](#)
 functions for reading, [6-2](#)
 output file name, [6-5](#)
 put functions, [6-5](#)
 querying records, [6-6](#)
 reading, [6-3](#)
 reading as XML, [6-8](#)
 writing, [6-5](#)
Avro maps, [6-3](#)
Avro maps, reading as XML, [6-10](#)
Avro null values, [6-13](#)
Avro primitives
 reading as XML, [6-12](#)
Avro reader schema, [6-4](#), [6-6](#), [6-46](#)
Avro records, reading as XML, [6-8](#)
Avro unions, reading as XML, [6-11](#)
avro((colon))collection-avroxml function, [6-2](#)
avro((colon))get function, [6-3](#)

avroxml method, [6-8](#), [6-13](#)

B

balancing loads in Oracle Loader for Hadoop, [3-33](#)

batchSize property, [6-56](#)

bzip2 input files, [2-32](#)

C

character encoding, [6-42](#), [6-45](#)

character methods for Hive, [8-6](#)

client libraries, [1-15](#)

clients

configuring Hadoop, [1-32](#)

coersing data types in R, [8-15](#)

collection annotation

text files, [6-76](#)

tika files, [6-83](#)

collection annotations

Avro, [6-3](#)

collection function (XQuery)

description, [5-4](#)

collection functions

Oracle NoSQL Database adapter, [6-42](#)

sequence files, [6-63](#)

text files, [6-76](#)

tika files, [6-83](#)

collection-tika function, [6-36](#), [6-60](#)

columnCount property (OSCH), [2-32](#)

columnLength property (OSCH), [2-32](#)

columnNames property (OSCH), [2-32](#)

columnType property (OSCH), [2-32](#)

compressed data files, [2-32](#)

compressed files, [2-37](#)

compression

data in database tables, [2-3](#)

sequence files, [6-64](#)

compression codec, [6-5](#)

compression methods

Avro output files, [6-6](#)

CompressionCodec property (OSCH), [2-32](#)

configuration properties

for Oracle XQuery for Hadoop, [6-30](#)

JSON file adapter, [6-21](#)

Oracle NoSQL Database adapter, [6-56](#)

Oracle XQuery for Hadoop, [5-19](#)

configuration settings

Hadoop client, [1-32](#)

Sqoop utility, [1-27](#)

configuring a Hadoop client, [1-32](#)

connecting to Oracle Database from R, [8-14](#)

consistency property, [6-56](#)

CREATE TABLE

configuration properties, [7-8](#)

examples, [7-9](#)

syntax, [7-7](#)

CREATE TABLE privilege, [1-13](#)

createBadFiles property, [2-32](#)

createLogFiles property, [2-32](#)

CSV files, [2-37](#), [3-37](#)

D

Data Pump files, [2-10](#)

XML template, [2-11](#)

data type mappings

between XQuery and Avro, [6-12](#)

between XQuery and Oracle Database, [6-26](#)

Oracle Database and XQuery, [6-24](#)

data type mappings, Hive (OSCH), [2-32](#)

data type testing in R, [8-15](#)

data types

Oracle Loader for Hadoop, [3-14](#)

database directories

for Oracle SQL Connector for HDFS, [1-11](#)

database patches, [1-15](#), [2-10](#)

database privileges, [1-13](#)

database system, configuring to run MapReduce jobs, [1-8](#)

database tables

writing using Oracle XQuery for Hadoop, [6-24](#)

databaseName property, Hive (OSCH), [2-32](#)

dataCompressionCodec property (OSCH), [2-32](#)

dataPathFilter property (OSCH), [2-32](#)

dataPaths property (OSCH), [2-32](#)

dateMask property (OSCH), [2-32](#)

defaultDirectory property (OSCH), [2-32](#)

deflate compression, [6-5](#)

delimited text files

XML templates, [2-21](#)

DelimitedTextInputFormat class, [3-19](#), [3-42](#)

Oracle Loader for Hadoop, [3-20](#)

delimiter

for splitting text files, [6-76](#)

Direct Connector for HDFS, [2-37](#)

directories, [1-11](#)

default HDFS for XQuery, [5-19](#)

Oracle SQL Connector for HDFS home, [1-11](#)

Sqoop home, [1-28](#)

Directory property (OSCH), [2-32](#)

disable_directory_link_check access parameter, [2-10](#)

distributed cache

accessing from Oracle XQuery for Hadoop, [5-7](#)

downloading software, [1-4](#), [1-27–1-29](#), [1-33](#)

drivers

- JDBC, [1-28](#), [3-27](#)
- ORACLE_DATAPUMP, [3-31](#)
- ORACLE_LOADER, [2-27](#)

durability property, [6-56](#)

E

encoding characters, [6-42](#), [6-45](#)

external tables

- about, [2-1](#)

ExternalTable command

- syntax, [2-7](#)

F

fieldLength property (OSCH), [2-32](#)

fieldTerminator property (OSCH), [2-32](#)

file paths

- locating in XQuery, [6-101](#)

FLWOR requirements, [5-7](#)

fn

- nilled function, [6-9](#), [6-10](#)

frame methods for Hive, [8-6](#)

functions

- for writing to Oracle NoSQL Database, [6-48](#)
- Oracle NoSQL Database, [6-34](#), [6-38](#), [6-40](#)
- parsing tika files, [6-82](#), [6-83](#)
- reading and writing sequence files, [6-59](#)
- reading and writing text files, [6-73](#)
- reading Avro files, [6-3](#)
- reading from Oracle NoSQL Database, [6-42](#), [6-45](#)
- reading JSON files, [6-17](#)
- reading sequence files, [6-63](#)
- reading text files, [6-76](#)
- reading XML files, [6-86](#), [6-88](#)
- writing Avro files, [6-5](#)
- writing sequence files, [6-64](#)
- writing text files, [6-78](#)

G

get function

- Oracle NoSQL Database adapter, [6-45](#)

get-tika function, [6-37](#)

gzip input files, [2-32](#)

H

Hadoop client

- configuring, [1-32](#)
- installing, [1-8](#)

HADOOP_HOME environment variable, [1-27](#)

HADOOP_LIBEXEC_DIR environment variable, [1-27](#)

HDFS commands

- issuing from R, [8-13](#)

HDFS data

- copying in R, [8-15](#)

HDFS directories

- creating in R, [8-13](#)

HDFS directory, [5-19](#)

HDFS files

- loading data into an Oracle database, [3-23](#)
- restrictions in Oracle R Advanced Analytics for Hadoop, [8-5](#)

hdfs_stream Bash shell script, [1-10](#)

Hive access from R, [8-5](#)

Hive access in R, [8-13](#)

Hive data type mappings (OSCH), [2-32](#)

Hive data types, support for, [8-7](#)

Hive JAR files for Oracle Loader for Hadoop, [3-32](#)

Hive tables

- XML format, [2-14](#)

hive.columnType property (OSCH), [2-32](#)

hive.databaseName property (OSCH), [2-32](#)

hive.partitionFilter property, [2-32](#)

hive.tableName property, [2-32](#)

HiveToAvroInputFormat class, [3-21](#), [3-32](#)

Hortonworks Data Platform distribution, [1-7](#)

hosts property, [6-56](#)

I

initialFieldEncloser property, [2-32](#)

InputFormat class

- Oracle Loader for Hadoop, [3-20](#)

installation

- Hadoop client, [1-8](#)
- Oracle Data Integrator Application Adapter for Hadoop, [1-33](#)
- Oracle Loader for Hadoop, [1-15](#)
- Oracle R Advanced Analytics for Hadoop, [1-26](#)
- Oracle Shell for Hadoop Loaders Setup, [1-17](#)
- Oracle SQL Connector for HDFS, [1-7](#)
- Sqoop utility, [1-27](#)

installation instructions, [1-1](#)

Instant Client libraries, [1-15](#)

J

JDBC drivers, [1-28](#), [3-27](#)

json

- get function, [6-18](#)
- parse-as-xml function, [6-18](#)

JSON data formats
 converting to XML, [6-22](#)
 JSON file adapter
 configuration properties, [6-21](#)
 JSON files
 reading, [6-17](#)
 JSON module, [6-17](#)
 examples, [6-20](#)

K

kv
 collection annotation, [6-42](#)
 collection-avroxml function, [6-34](#)
 collection-binxml function, [6-35](#)
 collection-text function, [6-34](#)
 collection-xml function, [6-35](#)
 get annotation, [6-45](#)
 get-avroxml function, [6-37](#)
 get-binxml function, [6-37](#)
 get-text function, [6-37](#)
 get-xml function, [6-37](#)
 key annotation, [6-42](#), [6-45](#)
 key-range function, [6-38](#)
 put annotation, [6-48](#)
 put-binxml function, [6-37](#)
 put-text function, [6-36](#)
 put-xml function, [6-36](#)
 kv-lob
 get-binxml, [6-41](#)
 get-text, [6-41](#)
 get-tika, [6-41](#)
 get-xml, [6-41](#)
 put-binxml, [6-42](#)
 put-text, [6-41](#)
 put-xml, [6-42](#)
 kv-table
 collection-jsontext, [6-39](#)
 KVAvroInputFormat class, [3-32](#)
 kvstore property, [6-56](#)

L

load balancing
 in Oracle Loader for Hadoop, [3-33](#)
 LOBSuffixproperty, [6-56](#)
 LOBTimeout property, [6-56](#)
 locationFileCount property, [2-32](#)
 log4j.logger.oracle.hadoop.xquery property, [5-19](#)
 logDirectory property, [2-32](#)
 logical methods for Hive, [8-6](#)

M

mapping
 JSON to XML, [6-23](#)
 mappings
 Oracle Database and XQuery data types, [6-24](#)
 mappings, Hive to Oracle Database (OSCH), [2-32](#)
 MapReduce functions
 writing in R, [8-16](#)
 MasterPolicy durability, [6-56](#)
 matrix methods for Hive, [8-6](#)

N

nilled elements, [6-9](#)
 nilled function, [6-13](#)
 null values in Avro, [6-13](#)
 numeric methods for Hive, [8-6](#)

O

OCI Direct Path, [3-38](#)
 OHS, [1-17](#)
 operating system user permissions, [1-11](#)
 ora-java
 binding annotation, [5-8](#)
 oracle
 columns annotation, [6-24](#)
 put annotation, [6-24](#)
 Oracle Data Integrator Application Adapter for Hadoop
 installing, [1-33](#)
 Oracle Database
 annotations for writing, [6-24](#)
 connecting from R, [8-14](#)
 put function, [6-24](#)
 user privileges, [1-13](#)
 Oracle Database access from ORCH, [8-10](#)
 Oracle Database adapter, [6-24](#)
 configuration properties, [6-30](#)
 examples, [6-28](#)
 Oracle Database Adapter
 using Oracle Loader for Hadoop, [6-24](#)
 Oracle Direct Connector for HDFS, [2-37](#)
 Oracle Exadata Database Machine
 installing a Hadoop client, [1-8](#)
 Oracle Instant Client libraries, [1-15](#)
 Oracle Loader for Hadoop
 description, [3-1](#)
 input formats, [3-23](#)
 installing, [1-15](#)
 supported database versions, [1-15](#)

- Oracle NoSQL Database
 - annotations for writing, [6-48](#)
- Oracle NoSQL Database adapter, [6-32](#)
 - annotations for reading, [6-42](#)
 - collection function, [6-42](#)
 - get function, [6-45](#)
 - reading Avro as XML, [6-8](#)
 - writing XML as Avro, [6-13](#)
- Oracle NoSQL Database Adapter
 - configuration properties, [6-56](#)
 - examples, [6-50](#)
- Oracle NoSQL Database functions, [6-34](#), [6-38](#), [6-40](#)
- Oracle OCI Direct Path, [3-37](#), [3-38](#)
- Oracle permissions, [1-11](#)
- Oracle R Advanced Analytics for Hadoop
 - categorical list of functions, [8-11](#)
 - connecting to Oracle Database, [8-14](#)
 - copying HDFS data, [8-15](#)
 - debugging functions, [8-17](#)
 - description, [1-2](#), [8-2](#)
 - HDFS commands issued from, [8-13](#)
 - installation, [1-26](#)
 - MapReduce functions, [8-16](#)
- Oracle RAC systems, installing a Hadoop client, [1-8](#)
- Oracle Shell for Hadoop Loaders Setup
 - installing, [1-17](#)
- Oracle Software Delivery Cloud, [1-4](#)
- Oracle SQL Connector for HDFS
 - description, [2-1](#)
 - installation, [1-7](#)
 - pattern-matching characters, [2-37](#)
 - query optimization, [2-45](#)
- Oracle Technology Network
 - downloads, [1-4](#), [1-28](#)
- Oracle XQuery for Hadoop, [5-1](#)
 - accessing the distributed cache, [5-7](#)
 - accessing user-defined XQuery library modules and XML schemas, [5-8](#)
 - basic transformation examples, [5-9](#)
 - calling custom Java external functions, [5-8](#)
 - configuration properties, [5-19](#)
 - configuring Oracle NoSQL Database server, [6-33](#)
 - description, [5-1](#)
 - error logging levels, [5-19](#)
 - error recovery setting, [5-19](#)
 - hadoop command, [5-14](#)
 - JSON module, [6-17](#)
 - Oracle NoSQL Database adapter, [6-32](#)
 - output directory, [5-19](#)
 - running queries, [5-14](#)
 - running queries locally, [5-16](#)
 - sequence file adapter, [6-58](#)
- Oracle XQuery for Hadoop (*continued*)
 - temp directory, [5-19](#)
 - text file adapter, [6-73](#)
 - tika adapter, [6-82](#)
 - time zone, [5-19](#)
 - XML file adapter, [6-86](#)
- Oracle XQuery for Hadoop adapters
 - overview, [5-4](#)
- Oracle XQuery for Hadoop modules
 - overview, [5-6](#)
- ORACLE_DATAPUMP driver, [3-31](#)
- ORACLE_LOADER driver, [2-27](#)
- oracle-property annotation, [6-24](#)
- oracle.hadoop.exctab.colMap.column_name.nullIf Specifier property, [2-32](#)
- oracle.hadoop.exctab.createBadFiles property, [2-32](#)
- oracle.hadoop.exctab.createLogFiles property, [2-32](#)
- oracle.hadoop.exctab.hive.tableName property, [2-32](#)
- oracle.hadoop.exctab.initialFieldEncloser property, [2-32](#)
- oracle.hadoop.exctab.locationFileCount property, [2-32](#)
- oracle.hadoop.exctab.logDirectory property, [2-32](#)
- oracle.hadoop.exctab.nullIfSpecifier property, [2-32](#)
- oracle.hadoop.exctab.preprocessorDirectory property, [2-32](#)
- oracle.hadoop.exctab.preprocessorScript, [2-32](#)
- oracle.hadoop.exctab.recordDelimiter property, [2-32](#)
- oracle.hadoop.exctab.sourceType property, [2-32](#)
- oracle.hadoop.exctab.stringSizes property, [2-32](#)
- oracle.hadoop.exctab.tableName property, [2-32](#)
- oracle.hadoop.xquery.* properties, [5-19](#)
- oracle.hadoop.xquery.json.parser.*, [6-21](#)
- oracle.hadoop.xquery.kv property, [6-56](#)
- oracle.hadoop.xquery.kv.config.durability property, [6-56](#)
- oracle.hadoop.xquery.kv.config.requestLimit property, [6-56](#)
- oracle.hadoop.xquery.kv.config.requestTimeout property, [6-56](#)
- oracle.hadoop.xquery.kv.config.socketOpenTime out property, [6-56](#)
- oracle.hadoop.xquery.kv.config.socketReadTime out property, [6-56](#)
- oracle.hadoop.xquery.lib.share property, [5-19](#)
- oracle.hadoop.xquery.tika.html.asis property, [6-84](#)
- oracle.hadoop.xquery.tika.locale property, [6-84](#)
- oracle.kv.batchSize property, [6-56](#)
- oracle.kv.consistency property, [6-56](#)

oracle.kv.hosts configuration property, [6-56](#)
 oracle.kv.hosts property, [6-56](#)
 oracle.kv.kvstore configuration property, [6-56](#)
 oracle.kv.kvstore property, [6-56](#)
 oracle.kv.timeout property, [6-56](#)
 orahdfs-version.zip file, [1-10](#)
 orahdfs-version/bin directory, [1-11](#)
 OraLoader, [3-36](#)
 oraloader-<version>.zip file, [1-24](#)
 oraloader-version directory, [1-16](#), [1-24](#)
 oraloader-version.zip, [1-17](#)
 oraloader-version.zip file, [1-10](#), [1-15](#)
 ORCH package
 installation, [1-27](#), [1-29](#)
 orch.tgz package, [1-29](#)
 ORE functions for Hive, [8-5](#)
 ore.create function, [8-9](#)
 ore.exec function, [8-9](#)
 ore.warn.order option, [8-9](#)
 OSCH_BIN_PATH directory, [1-13](#)
 output
 encoding annotation, [6-42](#), [6-45](#), [6-63](#), [6-88](#)
 parameter annotation, [6-78](#)
 output annotation, [6-64](#)
 output directory for Oracle XQuery for Hadoop,
 [5-19](#)
 oxh
 find function, [6-101](#)
 increment-counter function, [6-101](#)
 println function, [6-102](#)
 println-xml function, [6-102](#)
 property function, [6-102](#)
 oxh-charset property, [7-8](#)
 oxh-column property, [7-8](#)
 oxh-default-namespace property, [7-8](#)
 oxh-elements property, [7-8](#)
 oxh-entity.name property, [7-8](#)
 oxh-namespace.prefix property, [7-8](#)
 OXMLSerDe, [7-7](#)

P

parallel processing, [1-3](#), [2-45](#)
 parsing options for JSON files, [6-21](#)
 parsing tika files, [6-82](#)
 partitioning, [3-14](#)
 PathFilter property (OSCH), [2-32](#)
 Paths property (OSCH), [2-32](#)
 pattern matching, [5-19](#)
 pattern matching (OSCH), [2-32](#)
 pattern-matching characters in Oracle SQL
 Connector for HDFS, [2-37](#)
 preprocessor access parameter, [2-10](#)
 preprocessorDirectory property, [2-32](#)
 privileges, Oracle Database, [1-13](#)

put function (XQuery)
 description, [5-4](#)
 put functions
 Oracle NoSQL Database adapter, [6-48](#)
 sequence files, [6-64](#)
 text files, [6-78](#)

Q

queries
 running in Oracle XQuery for Hadoop, [5-14](#)
 running locally in Oracle XQuery for Hadoop,
 [5-16](#)
 query optimization for Oracle SQL Connector for
 HDFS, [2-45](#)

R

R data types, converting and testing, [8-15](#)
 R distribution, [1-27](#), [1-32](#)
 R Distribution, [1-28](#), [1-29](#), [1-33](#)
 R functions
 categorical listing, [8-11](#)
 R functions for Hive, [8-6](#)
 random order messages, [8-9](#)
 reading Avro files, [6-3](#)
 reading sequence files, [6-59](#)
 reading text files, [6-73](#)
 readZones property, [6-56](#)
 recordDelimiter property, [2-32](#)
 records, rejected, [3-33](#)
 rejected records, [3-33](#)
 ReplicaAck policy, [6-56](#)
 ReplicaPolicy durability, [6-56](#)
 requestLimit property, [6-56](#)
 requestTimeout property, [6-56](#)

S

sampling data
 from Oracle Loader for Hadoop, [3-33](#)
 scripts
 debugging in R, [8-17](#)
 security property, [6-56](#)
 seq
 collection annotation, [6-63](#)
 collection function, [6-59](#)
 collection-binxml function, [6-60](#)
 collection-xml function, [6-59](#)
 compress annotation, [6-64](#)
 file annotation, [6-64](#)
 key annotation, [6-63](#)
 put annotation, [6-64](#)
 put functions, [6-61](#)

- seq (*continued*)
 - put-binxml function, [6-62](#)
 - put-xml function, [6-61](#)
 - split-max annotation, [6-63](#)
 - split-min annotation, [6-63](#)
 - sequence file adapter, [6-58](#)
 - annotations for writing, [6-64](#)
 - collection function, [6-63](#)
 - examples, [6-66](#)
 - sequence file adapter functions, [6-59](#)
 - sequence files
 - compression, [6-64](#)
 - output file name, [6-64](#)
 - reading, [6-63](#)
 - split size, [6-64](#)
 - writing, [6-64](#)
 - serialization parameter, [6-49](#), [6-78](#)
 - serialization parameters, [6-102](#)
 - skiperrors property for Oracle XQuery for Hadoop, [5-19](#)
 - skiperrors.counters property, [5-19](#)
 - skiperrors.log.max property, [5-19](#)
 - skiperrors.max property, [5-19](#)
 - snappy compression, [6-5](#)
 - socketOpenTimeout property, [6-56](#)
 - socketReadTimeout property, [6-56](#)
 - software downloads, [1-4](#), [1-27–1-29](#), [1-33](#)
 - sourceType property, [2-32](#)
 - split size
 - for Avro files, [6-4](#)
 - sequence files, [6-64](#)
 - text files, [6-76](#)
 - split sizes, [6-4](#)
 - splitting XML files, [6-90](#)
 - SQL*Loader, [3-29](#)
 - Sqoop, [8-11](#)
 - Sqoop utility
 - installing on a Hadoop client, [1-33](#)
 - installing on a Hadoop cluster, [1-27](#)
 - stringSizes property, [2-32](#)
 - subrange specification, Oracle NoSQL Database adapter, [6-44](#)
- ## T
-
- tables
 - compression in database, [2-3](#)
 - copying data from HDFS, [3-1](#)
 - writing to Oracle Database, [6-24](#)
 - temp directory, setting for Oracle XQuery for Hadoop, [5-19](#)
 - text
 - collection annotation, [6-76](#)
 - collection function, [6-74](#)
 - collection-xml function, [6-74](#)
 - text (*continued*)
 - compress annotations, [6-78](#)
 - file annotation, [6-78](#)
 - put annotation, [6-78](#)
 - put function, [6-75](#)
 - put-xml function, [6-75](#)
 - split annotation, [6-76](#)
 - split-max annotation, [6-76](#)
 - trace function, [6-76](#)
 - text file adapter, [6-73](#)
 - collection function, [6-76](#)
 - put function, [6-78](#)
 - text files
 - converting to Avro, [6-6](#)
 - delimiter, [6-76](#)
 - reading, [6-76](#)
 - reading and writing, [6-73](#)
 - split size, [6-76](#)
 - writing, [6-78](#)
 - tika
 - %output encoding annotation, [6-83](#)
 - %output media-type annotation, [6-83](#)
 - collection annotation, [6-83](#)
 - collection function, [6-82](#)
 - helper function, [6-83](#)
 - parse function, [6-83](#)
 - parse textual data, [6-83](#)
 - tika adapter, [6-82](#)
 - tika file adapter
 - collection function, [6-83](#)
 - parsing, [6-82](#)
 - tika files
 - parsing, [6-83](#)
 - time zones in XQuery, [6-27](#)
 - timeout property, [6-56](#)
 - timestampMask property (OSCH), [2-32](#)
 - timestampTZMask property (OSCH), [2-32](#)
 - timezone property for Oracle XQuery for Hadoop, [5-19](#)
 - type mappings
 - between XQuery and Avro, [6-12](#)
 - between XQuery and Oracle Database, [6-26](#)
- ## U
-
- uncompressed files, [2-37](#)
 - updating functions, [5-7](#)
 - UTF-8 encoding, [6-42](#), [6-45](#)
 - UTL_FILE package, [1-13](#)
- ## V
-
- vector methods for Hive, [8-6](#)

W

wildcards, [5-19](#)
writing Avro files, [6-5](#)
writing sequence files, [6-59](#)
writing text files, [6-73](#)
writing to Oracle tables, [6-24](#)

X

XML

writing as Avro arrays, [6-15](#)
writing as Avro maps, [6-15](#)
writing as Avro primitives, [6-17](#)
writing as Avro records, [6-13](#)
writing as Avro unions, [6-16](#)
XML file adapter, [6-86](#)
examples, [6-91](#)
XML files
reading, [6-86](#), [6-88](#)
restrictions on splitting, [6-90](#)
XML schemas
accessing user-defined, [5-8](#)
XML template for Data Pump files, [2-11](#)
XML templates
Data Pump files, [2-11](#)
delimited text files, [2-21](#)
Hive tables, [2-14](#)

XML templates (*continued*)

XML_EXISTS function, [7-17](#)
XML_QUERY function, [7-18](#)
XML_QUERY_AS_primitive function, [7-20](#)
XML_TABLE function, [7-24](#)
xmlf
collection annotation, [6-88](#)
collection functions, [6-86](#)
split annotation, [6-88](#)
split-entity annotation, [6-88](#)
split-max annotation, [6-88](#)
split-min annotation, [6-76](#), [6-88](#)
split-namespace annotation, [6-88](#)
XQuery, [5-1](#)
XQuery library modules
accessing user-defined, [5-8](#)
XQuery specification support, [5-7](#)
XQuery transformations
requirements, [5-6](#)
xquery.output property, [5-19](#)
xquery.scratch property, [5-19](#)
xquery.skiperrors property, [5-19](#)
xquery.skiperrors.counters property, [5-19](#)
xquery.skiperrors.log.max property, [5-19](#)
xquery.skiperrors.max property, [5-19](#)
xquery.timezone property, [5-19](#)
xsi
nil attribute, [6-9](#)