# Oracle® Fusion Middleware

# Integrating Big Data with Oracle Data Integrator

14 *c* (14.1.2.0.0)
G10247-01
December 2024

Oracle Fusion Middleware Integrating Big Data with Oracle Data Integrator, 14 *c* (14.1.2.0.0)

G10247-01

Primary Author: Oracle Corporation

# Contents

## Preface

Audience ix

Documentation Accessibility ix

Related Documents ix

Conventions x

## 1 Big Data Integration with Oracle Data Integrator

1.1 Overview of Hadoop Data Integration 1-1

1.2 Big Data Knowledge Modules Matrix 1-2

## 2 Hadoop Data Integration Concepts

2.1 Hadoop Data Integration with Oracle Data Integrator 2-1

2.2 Generate Code in Different Languages with Oracle Data Integrator 2-1

2.3 Leveraging Apache Oozie to execute Oracle Data Integrator Projects 2-2

2.4 Oozie Workflow Execution Modes 2-2

2.5 Lambda Architecture 2-3

## 3 Setting Up the Environment for Integrating Big Data

3.1 Configuring Big Data technologies using the Big Data Configurations Wizard 3-1

- 3.1.1 General Settings 3-3
- 3.1.2 HDFS Data Server Definition 3-4
- 3.1.3 HBase Data Server Definition 3-4
- 3.1.4 Kafka Data Server Definition 3-5
- 3.1.5 Kafka Data Server Properties 3-6

3.2 Creating and Initializing the Hadoop Data Server 3-6

- 3.2.1 Hadoop Data Server Definition 3-6
- 3.2.2 Hadoop Data Server Properties 3-8

3.3 Creating a Hadoop Physical Schema 3-14

3.4 Configuring the Oracle Data Integrator Agent to Execute Hadoop Jobs 3-14

3.5 Configuring Oracle Loader for Hadoop 3-15

3.6 Configuring Oracle Data Integrator to Connect to a Secure Cluster 3-15

3.7 Configuring Oracle Data Integrator Studio for Executing Hadoop Jobs on the Local Agent 3-20

## 4 Integrating Hadoop Data

4.1 Integrating Hadoop Data 4-1
4.2 Setting Up File Data Sources 4-2
4.3 Setting Up HDFS Data Sources 4-3
4.4 Setting Up Hive Data Sources 4-3
4.5 Setting Up HBase Data Sources 4-4
4.6 Setting Up Kafka Data Sources 4-5
4.7 Setting Up Cassandra Data Sources 4-6
4.8 Importing Hadoop Knowledge Modules 4-6
4.9 Creating ODI Models and Data Stores to represent Hive, HBase and Cassandra Tables, and HDFS Files 4-7
4.9.1 Creating a Model 4-7
4.9.2 Reverse-Engineering Hive Tables 4-7
4.9.3 Reverse-Engineering HBase Tables 4-11
4.9.4 Reverse-Engineering HDFS Files 4-11
4.9.5 Reverse-Engineering Cassandra Tables 4-12
4.9.6 Reverse-Engineering Support for Kafka 4-12
4.10 Password Handling in Hadoop 4-13
4.11 Loading Data from Files into Hive 4-14
4.12 Loading Data from Hive to Files 4-14
4.13 Loading Data from HBase into Hive 4-14
4.14 Loading Data from Hive into HBase 4-15
4.15 Loading Data from an SQL Database into Hive, HBase, and File using SQOOP 4-15
4.16 Loading Data from an SQL Database into Hive using SQOOP 4-16
4.17 Loading Data from an SQL Database into HDFS File using SQOOP 4-16
4.18 Loading Data from an SQL Database into HBase using SQOOP 4-17
4.19 Validating and Transforming Data Within Hive 4-17
4.20 Loading Data into an Oracle Database from Hive and File 4-18
4.21 Loading Data into an SQL Database from Hbase, Hive, and File using SQOOP 4-18
4.22 Loading Data from Kafka to Spark Processing Engine 4-19

## 5 Executing Oozie Workflows

5.1 Executing Oozie Workflows with Oracle Data Integrator 5-1
5.2 Setting Up and Initializing the Oozie Runtime Engine 5-1
5.2.1 Oozie Runtime Engine Definition 5-2
5.2.2 Oozie Runtime Engine Properties 5-3
5.3 Creating a Logical Oozie Engine 5-3
5.4 Executing or Deploying an Oozie Workflow 5-4

5.5 Auditing Hadoop Logs 5-4
5.6 Userlib jars support for running ODI Oozie workflows 5-5

## 6 Using Query Processing Engines to Generate Code in Different Languages

6.1 Query Processing Engines Supported by Oracle Data Integrator 6-1
6.2 Setting Up Hive Data Server 6-2
6.2.1 Hive Data Server Definition 6-2
6.2.2 Hive Data Server Connection Details 6-2
6.3 Creating a Hive Physical Schema 6-3
6.4 Setting Up Pig Data Server 6-3
6.4.1 Pig Data Server Definition 6-3
6.4.2 Pig Data Server Properties 6-4
6.5 Creating a Pig Physical Schema 6-5
6.6 Setting Up Spark Data Server 6-5
6.6.1 Spark Data Server Definition 6-5
6.6.2 Spark Data Server Properties 6-6
6.7 Creating a Spark Physical Schema 6-7
6.8 Generating Code in Different Languages 6-7

## 7 Working with Spark

7.1 Spark Usage 7-1
7.1.1 Creating a Spark Mapping 7-1
7.1.2 Pre-requisites for handling Avro and Delimited files in Spark Mappings 7-1
7.2 Spark Design Considerations 7-2
7.2.1 Batch or Streaming 7-3
7.2.2 Resilient Distributed Datasets (RDD) or DataFrames 7-3
7.2.3 Infer Schema Knowledge Module Option 7-3
7.2.4 Expression Syntax 7-4
7.3 Spark Streaming Support 7-6
7.3.1 Spark Checkpointing 7-7
7.3.2 Spark Windowing and Stateful Aggregation 7-7
7.3.3 Spark Repartitioning and Caching 7-8
7.3.4 Configuring Streaming Support 7-9
7.3.4.1 Spark Streaming DataServer Properties 7-9
7.3.4.2 Extra Spark Streaming Data Properties 7-10
7.3.5 Executing Mapping in Streaming Mode 7-11
7.4 Switching between RDD and DataFrames in ODI 7-12
7.5 Components that do not support DataFrame Code Generation 7-12

7.6 Adding Customized Code in the form of a Table Function 7-12

## 8 Working with Unstructured Data

8.1 Working with Unstructured Data 8-1

## 9 Working with Complex Datatypes and HDFS File Formats

9.1 HDFS File Formats 9-1
9.2 Working with Complex Datatypes in Mappings 9-2
9.3 Hive Complex Datatypes 9-3
9.3.1 Using Flatten for Complex Types in Hive Mappings 9-3
9.4 Cassandra Complex Datatypes 9-5
9.4.1 How ODI deals with Cassandra Lists and User Defined Types 9-6
9.5 Loading Data from HDFS File to Hive 9-8
9.6 Loading Data from HDFS File to Spark 9-9

## A Hive Knowledge Modules

A.1 LKM SQL to Hive SQOOP A-2
A.2 LKM SQL to File SQOOP Direct A-3
A.3 LKM SQL to HBase SQOOP Direct A-5
A.4 LKM File to SQL SQOOP A-6
A.5 LKM Hive to SQL SQOOP A-8
A.6 LKM HBase to SQL SQOOP A-9
A.7 LKM HDFS File to Hive Load Data A-11
A.8 LKM HDFS File to Hive Load Data (Direct) A-11
A.9 IKM Hive Append A-11
A.10 IKM Hive Incremental Update A-12
A.11 LKM File to Hive LOAD DATA A-12
A.12 LKM File to Hive LOAD DATA Direct A-13
A.13 LKM HBase to Hive HBASE-SERDE A-15
A.14 LKM Hive to HBase Incremental Update HBASE-SERDE Direct A-15
A.15 LKM Hive to File Direct A-16
A.16 XKM Hive Sort A-16
A.17 LKM File to Oracle OLH-OSCH A-16
A.18 LKM File to Oracle OLH-OSCH Direct A-19
A.19 LKM Hive to Oracle OLH-OSCH A-21
A.20 LKM Hive to Oracle OLH-OSCH Direct A-24
A.21 RKM Hive A-27
A.22 RKM HBase A-28
A.23 IKM File to Hive (Deprecated) A-29

A.24 LKM HBase to Hive (HBase-SerDe) [Deprecated] A-31
A.25 IKM Hive to HBase Incremental Update (HBase-SerDe) [Deprecated] A-32
A.26 IKM SQL to Hive-HBase-File (SQOOP) [Deprecated] A-32
A.27 IKM Hive Control Append (Deprecated) A-34
A.28 CKM Hive A-35
A.29 IKM Hive Transform (Deprecated) A-35
A.30 IKM File-Hive to Oracle (OLH-OSCH) [Deprecated] A-37
A.31 IKM File-Hive to SQL (SQOOP) [Deprecated] A-40

## B Pig Knowledge Modules

B.1 LKM File to Pig B-1
B.2 LKM Pig to File B-3
B.3 LKM HBase to Pig B-4
B.4 LKM Pig to HBase B-5
B.5 LKM Hive to Pig B-6
B.6 LKM Pig to Hive B-6
B.7 LKM SQL to Pig SQOOP B-7
B.8 XKM Pig Aggregate B-8
B.9 XKM Pig Distinct B-9
B.10 XKM Pig Expression B-9
B.11 XKM Pig Filter B-9
B.12 XKM Pig Flatten B-9
B.13 XKM Pig Join B-9
B.14 XKM Pig Lookup B-10
B.15 XKM Pig Pivot B-10
B.16 XKM Pig Set B-10
B.17 XKM Pig Sort B-10
B.18 XKM Pig Split B-10
B.19 XKM Pig Subquery Filter B-10
B.20 XKM Pig Table Function B-10
B.21 XKM Pig Unpivot B-10

## C Spark Knowledge Modules

C.1 LKM File to Spark C-2
C.2 LKM Spark to File C-3
C.3 LKM Hive to Spark C-5
C.4 LKM Spark to Hive C-5
C.5 LKM HDFS to Spark C-7
C.6 LKM Spark to HDFS C-7
C.7 LKM Kafka to Spark C-8

C.8 LKM Spark to Kafka C-9
C.9 LKM SQL to Spark C-9
C.10 LKM Spark to SQL C-10
C.11 LKM Spark to Cassandra C-11
C.12 RKM Cassandra C-12
C.13 XKM Spark Aggregate C-12
C.14 XKM Spark Distinct C-13
C.15 XKM Spark Expression C-13
C.16 XKM Spark Filter C-13
C.17 XKM Spark Input Signature and Output Signature C-13
C.18 XKM Spark Join C-13
C.19 XKM Spark Lookup C-14
C.20 XKM Spark Pivot C-14
C.21 XKM Spark Set C-15
C.22 XKM Spark Sort C-15
C.23 XKM Spark Split C-15
C.24 XKM Spark Table Function C-15
C.25 IKM Spark Table Function C-16
C.26 XKM Spark Unpivot C-16

D Component Knowledge Modules

D.1 XKM Oracle Flatten D-1
D.2 XKM Oracle Flatten XML D-1
D.3 XKM Spark Flatten D-2
D.4 XKM Jagged D-2

E Considerations, Limitations, and Issues

E.1 Considerations, Limitations, and Issues E-1

# Preface

This manual describes how to develop Big Data integration projects using Oracle Data Integrator.

This preface contains the following topics:.

- Audience
- Documentation Accessibility
- Related Documents
- Conventions

## Audience

This document is intended for anyone interested in using Oracle Data Integrator (ODI) to develop Big Data integration projects. It provides conceptual information about the Big Data related features and functionality of ODI and also explains how to use the ODI graphical user interface to create integration projects.

## 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 in the *Oracle Data Integrator Library*.

- *Release Notes for Oracle Data Integrator*
- *Understanding Oracle Data Integrator*
- *Developing Integration Projects with Oracle Data Integrator*
- *Administering Oracle Data Integrator*
- *Installing and Configuring Oracle Data Integrator*
- *Upgrading Oracle Data Integrator*
- *Application Adapters Guide for Oracle Data Integrator*
- *Developing Knowledge Modules with Oracle Data Integrator*

- *Connectivity and Knowledge Modules Guide for Oracle Data Integrator Developer's Guide*
- *Migrating From Oracle Warehouse Builder to Oracle Data Integrator*
- *Oracle Data Integrator Tools Reference*
- *Data Services Java API Reference for Oracle Data Integrator*
- *Open Tools Java API Reference for Oracle Data Integrator*
- *Getting Started with SAP ABAP BW Adapter for Oracle Data Integrator*
- *Java API Reference for Oracle Data Integrator*
- *Getting Started with SAP ABAP ERP Adapter for Oracle Data Integrator*
- *Oracle Data Integrator 12c Online Help*, which is available in ODI Studio through the JDeveloper Help Center when you press **F1** or from the main menu by selecting **Help**, and then **Search** or **Table of Contents**.

## Conventions

The following text conventions are used in this document:

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

1

# Big Data Integration with Oracle Data Integrator

This chapter provides an overview of Big Data integration using Oracle Data Integrator. It also provides a compatibility matrix of the supported Big Data technologies.
This chapter includes the following sections:

- Overview of Hadoop Data Integration
- Big Data Knowledge Modules Matrix

## 1.1 Overview of Hadoop Data Integration

Oracle Data Integrator combined with Hadoop, can be used to design the integration flow to process huge data from non-relational data sources.

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

You can use Oracle Data Integrator to design the 'what' of an integration flow and assign knowledge modules to define the 'how' of the flow in an extensible range of mechanisms. The 'how' is whether it is Oracle, Teradata, Hive, Spark, Pig, etc.

Employing familiar and easy-to-use tools and preconfigured knowledge modules (KMs), Oracle Data Integrator lets you to do the following:

- Reverse-engineer non-relational and relational data stores like Hive, HBase, and Cassandra.

  For more information, see Creating ODI Models and Data Stores to represent Hive, HBase and Cassandra Tables, and HDFS Files.
- Load data into Hadoop directly from Files or SQL databases.

  For more information, see Integrating Hadoop Data.
- Validate and transform data within Hadoop with the ability to make the data available in various forms such as Hive, HBase, or HDFS.

  For more information, see Validating and Transforming Data Within Hive.
- Load the processed data from Hadoop into Oracle database, SQL database, or Files.

  For more information, see Integrating Hadoop Data.
- Execute integration projects as Oozie workflows on Hadoop.

  For more information, see Executing Oozie Workflows with Oracle Data Integrator.
- Audit Oozie workflow execution logs from within Oracle Data Integrator.

  For more information, see Auditing Hadoop Logs.
- Generate code in different languages for Hadoop, such as HiveQL, Pig Latin, or Spark Python.

  For more information, see Generating Code in Different Languages

## 1.2 Big Data Knowledge Modules Matrix

Big Data Knowledge Modules Matrix depicts the Big Data Loading and Integration KMs that are provided by Oracle Data Integrator.

Depending on the source and target technologies, you can use the KMs shown in the following table in your integration projects. You can also use a combination of these KMs. For example, to read data from SQL into Spark, you can load the data from SQL into Spark first using `LKM SQL to Spark`, and then use `LKM Spark to HDFS` to continue.

The Big Data knowledge modules that start with LKM File for example, LKM File to SQL SQOOP support both OS File and HDFS File, as described in this matrix. We provide additional KMs, starting with LKM HDFS to Spark, LKM HDFS File to Hive. These support HDFS files only, unlike the other KMs, however, they have additional capabilities, for example, Complex Data can be described in an HDFS data store and used in a mapping using the flatten component.

The following table shows the Big Data Loading and Integration KMs that Oracle Data Integrator provides to integrate data between different source and target technologies.

**Table 1-1 Big Data Loading and Integration Knowledge Modules**

| Source | Target | Knowledge Module |
|---|---|---|
| OS File | HDFS File | NA |
| | Hive | LKM File to Hive LOAD DATA Direct |
| | HBase | NA |
| | Pig | LKM File to Pig |
| | Spark | LKM File to Spark |
| SQL | HDFS File | LKM SQL to File SQOOP Direct |
| | Hive | LKM SQL to Hive SQOOP |
| | HBase | LKM SQL to HBase SQOOP Direct |
| | Pig | LKM SQL to Pig SQOOP |
| | Spark | LKM SQL to Spark |
| HDFS | Kafka | NA |
| HDFS | Spark | LKM HDFS to Spark |
| HDFS File | OS File | NA |
| | SQL | LKM File to SQL SQOOP<br>LKM File to Oracle OLH-OSCH Direct |
| | HDFS File | NA |
| | Hive | LKM File to Hive LOAD DATA Direct<br>LKM HDFS File to Hive Load Data<br>LKM HDFS File to Hive Load Data (Direct) |
| | HBase | NA |
| | Pig | LKM File to Pig |
| | Spark | LKM HDFS to Spark |
| Hive | OS File | LKM Hive to File Direct |

**Table 1-1 (Cont.) Big Data Loading and Integration Knowledge Modules**

| Source | Target | Knowledge Module |
|---|---|---|
| | SQL | LKM Hive to SQL SQOOP<br>LKM Hive to Oracle OLH-OSCH Direct |
| | HDFS File | LKM Hive to File Direct |
| | Hive | IKM Hive Append<br>IKM Hive Incremental Update |
| | HBase | LKM Hive to HBase Incremental Update HBASE-SERDE Direct |
| | Pig | LKM Hive to Pig |
| | Spark | LKM Hive to Spark |
| HBase | OS File | NA |
| | SQL | LKM HBase to SQL SQOOP |
| | HDFS File | NA |
| | Hive | LKM HBase to Hive HBASE-SERDE |
| | HBase | NA |
| | Pig | LKM HBase to Pig |
| | Spark | NA |
| Pig | OS File | LKM Pig to File |
| | HDFS File | LKM Pig to File |
| | Hive | LKM Pig to Hive |
| | HBase | LKM Pig to HBase |
| | Pig | NA |
| | Spark | NA |
| Spark | OS File | LKM Spark to File |
| | SQL | LKM Spark to SQL |
| | HDFS File | LKM Spark to File<br>LKM Spark to HDFS |
| | Hive | LKM Spark to Hive |
| | HBase | NA |
| | Pig | NA |
| | Spark | IKM Spark Table Function |
| | Kafka | LKM Spark to Kafka |
| | Cassandra | LKM Spark to Cassandra |

The following table shows the Big Data Reverse Engineering KMs provided by ODI.

**Table 1-2 Big Data Reverse-Engineering Knowledge Modules**

| Technology | Knowledge Module |
|---|---|
| HBase | RKM HBase |
| Hive | RKM Hive |

Table 1-2 (Cont.) Big Data Reverse-Engineering Knowledge Modules

| Technology | Knowledge Module |
|---|---|
| **Cassandra** | RKM Cassandra |

# 2 Hadoop Data Integration Concepts

The chapter provides an introduction to the basic concepts of Hadoop Data integration using Oracle Data Integrator.
This chapter includes the following sections:

- Hadoop Data Integration with Oracle Data Integrator
- Generate Code in Different Languages with Oracle Data Integrator
- Leveraging Apache Oozie to execute Oracle Data Integrator Projects
- Oozie Workflow Execution Modes
- Lambda Architecture

## 2.1 Hadoop Data Integration with Oracle Data Integrator

When you implement a big data processing scenario, the first step is to load the data into Hadoop. The data source is typically in Files or SQL databases.

When the data has been aggregated, condensed, or processed into a smaller data set, you can load it into an Oracle database, other relational database, HDFS, HBase, or Hive for further processing and analysis. Oracle Loader for Hadoop is recommended for optimal loading into an Oracle database.

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

A Kafka cluster consists of one to many Kafka brokers handling and storing messages. Messages are organized into topics and physically broken down into topic partitions. Kafka producers connect to a cluster and feed messages into a topic. Kafka consumers connect to a cluster and receive messages from a topic. All messages on a specific topic need not have the same message format, it is a good practice to use only a single message format per topic. Kafka is integrated into ODI as a new technology.

For more information, see Integrating Hadoop Data .

## 2.2 Generate Code in Different Languages with Oracle Data Integrator

Oracle Data Integrator can generate code for multiple languages. For Big Data, this includes HiveQL, Pig Latin, Spark RDD, and Spark DataFrames.

The style of code is primarily determined by the choice of the data server used for the staging location of the mapping.

It is recommended to run Spark applications on yarn. Following this recommendation ODI only supports yarn-client and yarn-cluster mode execution and has introduced a run-time check.

In case you are using any other Spark deployment modes, which is not supported in ODI, the following dataserver property must be added to the Spark dataserver:

```
odi.spark.enableUnsupportedSparkModes = true
```

For more information about generating code in different languages and the Pig and Spark component KMs, see the following:

- Pig Knowledge Modules .
- Spark Knowledge Modules .
- Using Query Processing Engines to Generate Code in Different Languages.

## 2.3 Leveraging Apache Oozie to execute Oracle Data Integrator Projects

Apache Oozie is a workflow scheduler system that helps you orchestrate actions in Hadoop. It is a server-based Workflow Engine specialized in running workflow jobs with actions that run Hadoop MapReduce jobs.

Implementing and running Oozie workflow requires in-depth knowledge of Oozie.

However, Oracle Data Integrator does not require you to be an Oozie expert. With Oracle Data Integrator you can easily define and execute Oozie workflows.

Oracle Data Integrator enables automatic generation of an Oozie workflow definition by executing an integration project (package, procedure, mapping, or scenario) on an Oozie engine. The generated Oozie workflow definition is deployed and executed into an Oozie workflow system. You can also choose to only deploy the Oozie workflow to validate its content or execute it at a later time.

Information from the Oozie logs is captured and stored in the ODI repository along with links to the Oozie UIs. This information is available for viewing within ODI Operator and Console.

For more information, see Executing Oozie Workflows.

## 2.4 Oozie Workflow Execution Modes

You can execute Oozie workflows through Task and Session modes. Task mode is the default mode for Oozie workflow execution.

ODI provides the following two modes for executing the Oozie workflows:

- **TASK**

  Task mode generates an Oozie action for every ODI task. This is the default mode.

  The task mode cannot handle the following:

  - KMs with scripting code that spans across multiple tasks.
  - KMs with transactions.
  - KMs with file system access that cannot span file access across tasks.
  - ODI packages with looping constructs.
- **SESSION**

Session mode generates an Oozie action for the entire session.

ODI automatically uses this mode if any of the following conditions is true:

- Any task opens a transactional connection.
- Any task has scripting.
- A package contains loops.

  Loops in a package are not supported by Oozie engines and may not function properly in terms of execution and/or session log content retrieval, even when running in SESSION mode.

> **Note:**
>
> This mode is recommended for most of the use cases.

By default, the Oozie Runtime Engines use the Task mode, that is, the default value of the `OOZIE_WF_GEN_MAX_DETAIL` property for the Oozie Runtime Engines is **TASK**.

You can configure an Oozie Runtime Engine to use Session mode, irrespective of whether the conditions mentioned above are satisfied or not. To force an Oozie Runtime Engine to generate session level Oozie workflows, set the `OOZIE_WF_GEN_MAX_DETAIL` property for the Oozie Runtime Engine to **SESSION**.

For more information, see Oozie Runtime Engine Properties.

## 2.5 Lambda Architecture

Lambda architecture is a data-processing architecture designed to handle massive quantities of data by taking advantage of both batch and stream processing methods.

Lambda architecture has the following layers:

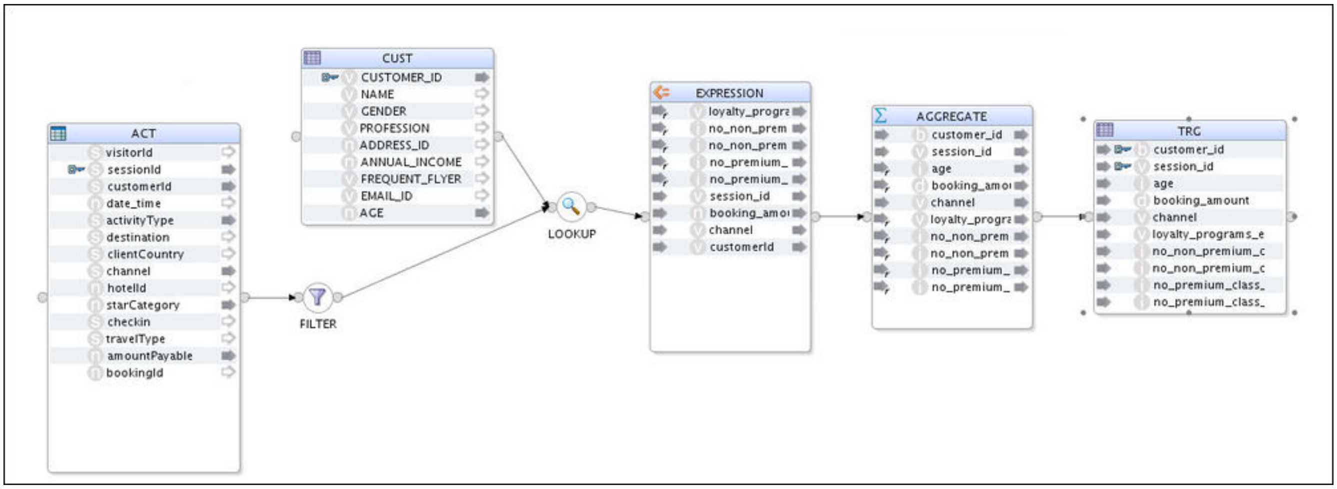
1. **Batch Layer**: In this layer, fresh data is loaded into the system at regular intervals with perfect accuracy containing complete detail. New data is processed with all available data when generating views.
2. **Speed Layer**: In this layer, real-time data is streamed into the system for immediate availability at the expense of total completeness (which is resolved in the next Batch run).
3. **Serving Layer**: In this layer, views are built to join the data from the Batch and Speed layers.

With ODI Mappings, you can create a single Logical Mapping for which you can provide multiple Physical Mappings. This is ideal for implementing Lambda Architecture with ODI.

### Logical Mapping

In the following figure, the ACT datastore is defined as a Generic data store. The same applies to the TRG node. These can be created by copying and pasting from a reverse-engineered data store with the required attributes.

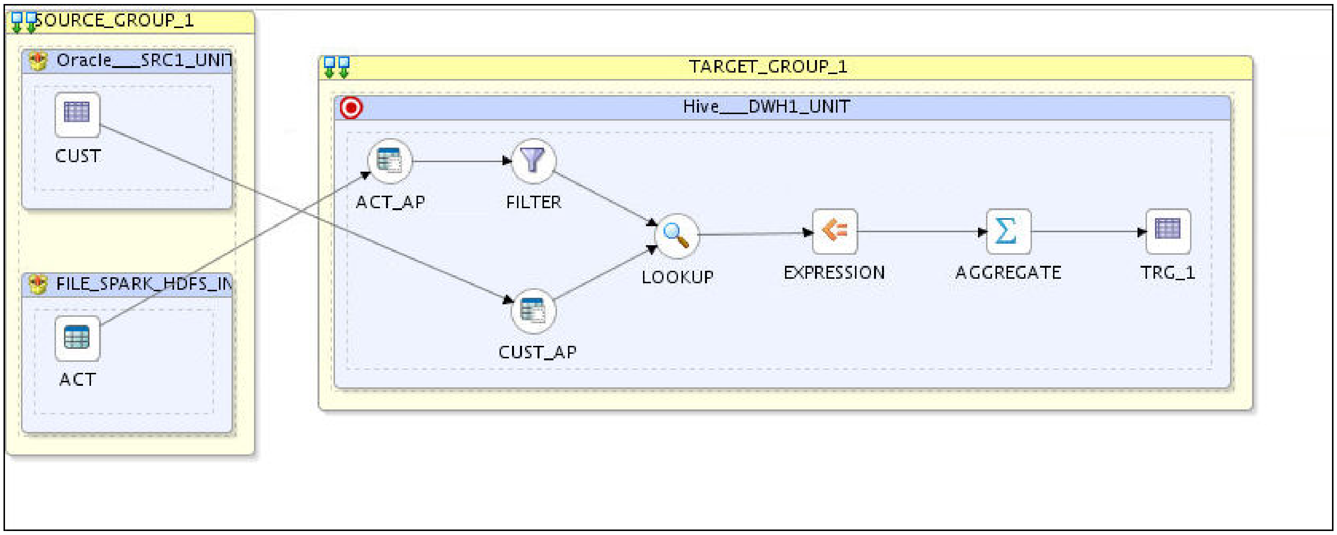
**Figure 2-1 Logical Mapping in Lambda Architecture**



### Batch Physical Mapping

As seen in the figure below, for Batch Physical Mapping, ACT has a File Data Store and TRG_1 has a Hive Data Store.

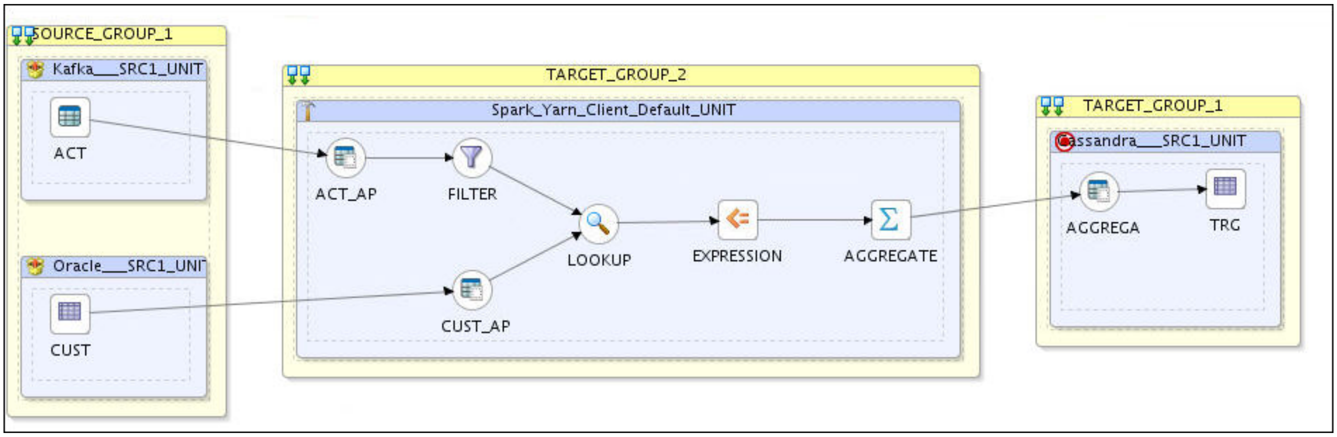
**Figure 2-2 Batch Physical Mapping in Lambda Architecture**



### Streaming Physical Mapping

As seen in the figure below, Streaming Physical Mapping has a Kafka Data Store for ACT and a Cassandra Data Store for TRG.

**Figure 2-3 Streaming Physical Mapping in Lambda Architecture**



Any changes made to the logical mapping will be synced to each physical mapping, thus reducing the complexity of Lambda architecture implementations.

3

# Setting Up the Environment for Integrating Big Data

This chapter provides information on the steps you need to perform to set up the environment to integrate Big Data.
This chapter includes the following sections:

- Configuring Big Data technologies using the Big Data Configurations Wizard
- Creating and Initializing the Hadoop Data Server
- Creating a Hadoop Physical Schema
- Configuring the Oracle Data Integrator Agent to Execute Hadoop Jobs
- Configuring Oracle Loader for Hadoop
- Configuring Oracle Data Integrator to Connect to a Secure Cluster
- Configuring Oracle Data Integrator Studio for Executing Hadoop Jobs on the Local Agent

## 3.1 Configuring Big Data technologies using the Big Data Configurations Wizard

The Big Data Configurations wizard provides a single entry point to set up multiple Hadoop technologies. You can quickly create data servers, physical schema, logical schema, and set a context for different Hadoop technologies such as Hadoop File System or HDFS, HBase, Oozie, Spark, Hive, Pig, etc

The default metadata for different distributions, such as properties, host names, port numbers, etc., and default values for environment variables are pre-populated for you. This helps you to easily create the data servers along with the physical and logical schema, without having in-depth knowledge about these technologies.

After all the technologies are configured, you can validate the settings against the data servers to test the connection status.

> **Note:**
>
> If you do not want to use the **Big Data Configurations** wizard, you can set up the data servers for the Big Data technologies manually using the information mentioned in the subsequent sections.

To run the Big Data Configurations Wizard:

1. In ODI Studio, select **File** and click **New...** or

   Select **Topology** tab — **Topology** Menu — **Big Data Configurations**.
2. In the **New Gallery** dialog, select **Big Data Configurations** and click **OK**.

The **Big Data Configurations** wizard appears.

3. In the **General Settings** panel of the wizard, specify the required options.

   See General Settings for more information.

4. Click **Next**.

   Data server panel for each of the technologies you selected in the **General Settings** panel will be displayed.

5. In the **Hadoop** panel of the wizard, do the following:
   - Specify the options required to create the Hadoop data server.

     See Hadoop Data Server Definition for more information.
   - In **Properties** section, click the + icon to add any data server properties.
   - Select a logical schema, physical schema, and a context from the appropriate drop-down lists.
6. Click **Next**.
7. In the **HDFS** panel of the wizard, do the following:
   - Specify the options required to create the HDFS data server.

     See HDFS Data Server Definition for more information.
   - In the **Properties** section, click + icon to add any data server properties.
   - Select a logical schema, physical schema, and a context from the appropriate drop-down lists.
8. Click **Next**.
9. In the **HBase** panel of the wizard, do the following:
   - Specify the options required to create the HBase data server.

     See HBase Data Server Definition for more information.
   - In the **Properties** section, click + icon to add any data server properties.
   - Select a logical schema, physical schema, and a context from the appropriate drop-down lists.
10. In the **Spark** panel of the wizard, do the following:
    - Specify the options required to create the Spark data server.

      See Spark Data Server Definition for more information.
    - In the **Properties** section, click + icon to add any data server properties.
    - Select a logical schema, physical schema, and a context from the appropriate drop-down lists.
11. Click **Next**.
12. In the **Kafka** panel of the wizard, do the following:
    - Specify the options required to create the Kafka data server.

      See Kafka Data Server Definition for more information.
    - In the **Properties** section, click + icon to add any data server properties.
    - Select a logical schema, physical schema, and a context from the appropriate drop-down lists.
13. Click **Next**.

14. In the **Pig** panel of the wizard, do the following:
    - Specify the options required to create the Pig data server.

      See Pig Data Server Definition for more information.
    - In the **Properties** section, click + icon to add any data server properties.
    - Select a logical schema, physical schema, and a context from the appropriate drop-down lists.
15. Click **Next**.
16. In the **Hive** panel of the wizard, do the following:
    - Specify the options required to create the Hive data server.

      See Hive Data Server Definition for more information.
    - In the **Properties** section, click + icon to add any data server properties.
    - Select a logical schema, physical schema, and a context from the appropriate drop-down lists.
17. Click **Next**.
18. In the **Oozie** panel of the wizard, do the following:
    - Specify the options required to create the Oozie run-time engine.

      See Oozie Runtime Engine Definition for more information.
    - Under **Properties** section, review the data server properties that are listed.

      **Note:** You cannot add new properties or remove listed properties. However, if required, you can change the value of listed properties.

      See Oozie Runtime Engine Properties for more information.
    - Select an existing logical agent and context or enter new names for the logical agent and context.
19. Click **Next**.
20. In the **Validate all settings** panel, click **Validate All Settings** to initialize operations and validate the settings against the data servers to ensure the connection status.
21. Click **Finish**.

## 3.1.1 General Settings

The following table describes the options that you need to set on the **General Settings** panel of the Big Data Configurations wizard.

**Table 3-1 General Settings Options**

| Option | Description |
|---|---|
| **Prefix** | Specify a prefix. This prefix is attached to the data server name, logical schema name, and physical schema name. |
| **Distribution** | Select a distribution, either **Manual** or **Cloudera Distribution for Hadoop (CDH) <version>**. |
| **Base Directory** | Specify the directory location where CDH is installed. This base directory is automatically populated in all other panels of the wizard.<br>**Note:** This option appears only if the distribution is other than **Manual**. |

**Table 3-1 (Cont.) General Settings Options**

| Option | Description |
|---|---|
| **Distribution Type** | Select a distribution type, either **Normal** or **Kerberized**. |
| **Technologies** | Select the technologies that you want to configure.<br>**Note:** Data server creation panels are displayed only for the selected technologies. |

Configuring Big Data technologies using the Big Data Configurations Wizard.

## 3.1.2 HDFS Data Server Definition

The following table describes the options that you must specify to create a HDFS data server.

> **Note:**
>
> Only the fields required or specific for defining a HDFS data server are described.

**Table 3-2 HDFS Data Server Definition**

| Option | Description |
|---|---|
| **Name** | Type a name for the data server. This name appears in Oracle Data Integrator. |
| **User/Password** | HDFS currently does not implement User/Password security. Leave this option blank. |
| **Hadoop Data Server** | Hadoop data server that you want to associate with the HDFS data server. |
| **Additional Classpath** | Specify additional jar files to the classpath if needed. |

## 3.1.3 HBase Data Server Definition

The following table describes the options that you must specify to create an HBase data server.

**Note:** Only the fields required or specific for defining a HBase data server are described.

**Table 3-3 HBase Data Server Definition**

| Option | Description |
|---|---|
| **Name** | Type a name for the data server. This name appears in Oracle Data Integrator. |
| **HBase Quorum** | ZooKeeper Quorum address in hbase-site.xml . For example, `localhost:2181`. |
| **User/Password** | HBase currently does not implement User/Password security. Leave these fields blank. |
| **Hadoop Data Server** | Hadoop data server that you want to associate with the HBase data server. |

**Table 3-3 (Cont.) HBase Data Server Definition**

| Option | Description |
|---|---|
| **Additional Classpath** | Specify any additional classes/jar files to be added.<br>The following classpath entries will be built by the Base Directory value:<br>• `/usr/lib/hbase/*`<br>• `/usr/lib/hbase/lib` |

Configuring Big Data technologies using the Big Data Configurations Wizard.

## 3.1.4 Kafka Data Server Definition

The following table describes the options that you must specify to create a Kafka data server.

> **Note:**
>
> Only the fields required or specific for defining a Kafka data server are described.

**Table 3-4 Kafka Data Server Definition**

| Option | Description |
|---|---|
| **Name** | Type a name for the data server. |
| **User/Password** | User name with its password. |
| **Hadoop Data Server** | Hadoop data server that you want to associate with the Kafka data server.<br>If Kafka is not running on the Hadoop server, then there is no need to specify a Hadoop Data Server. This option is useful when Kafka runs on its own server. |
| **Additional Classpath** | Specify any additional classes/jar files to be added.<br>The following classpath entries will be built by the Base Directory value:<br>• `/opt/cloudera/parcels/CDH/lib/kafka/libs/*`<br>If required, you can add more additional classpaths.<br>If Kafka is not running on the Hadoop server, then specify the absolute path of Kafka libraries in this field.<br>**Note:**<br>This field appears only when you are creating the Kafka Data Server using the Big Data Configuration wizard. |

### 3.1.5 Kafka Data Server Properties

The following table describes the Kafka data server properties that you need to add on the Properties tab when creating a new Kafka data server.

**Table 3-5 Kafka Data Server Properties**

| Key | Value |
|---|---|
| **metadata.broker.list** | This is a comma separated list of Kafka metadata brokers. Each broker is defined by `hostname:port`. The list of brokers can be found in the `server.properties` file, typically located in `/etc/kafka/conf`. |
| **oracle.odi.prefer.dataserver.packages** | Retrieves the topic and message from Kafka server. The address is **scala, kafka, oracle.odi.kafka.client.api.impl, org.apache.log4j**. |
| **security.protocol** | Protocol used to communicate with brokers. Valid values are: **PLAINTEXT**, **SSL**, **SASL_PLAINTEXT**, and **SASL_SSL**. |
| **zookeeper.connect** | Specifies the ZooKeeper connection string in the form `hostname:port`, where host and port are the host and port of a ZooKeeper server. To allow connecting through other ZooKeeper nodes when a ZooKeeper machine is down you can also specify multiple hosts in the form `hostname1:port1,hostname2:port2,hostname3:port3`. |

## 3.2 Creating and Initializing the Hadoop Data Server

Configure the Hadoop Data Server Definitions and Properties, to create and initialize Hadoop Data Server.

To create and initialize the Hadoop data server:

1. Click the Topology tab.
2. In the Physical Architecture tree, under Technologies, right-click **Hadoop** and then click **New Data Server**.
3. In the Definition tab, specify the details of the Hadoop data server.

   See Hadoop Data Server Definition for more information.
4. In the Properties tab, specify the properties for the Hadoop data server.

   See Hadoop Data Server Properties for more information.
5. Click **Initialize** to initialize the Hadoop data server.

   Initializing the Hadoop data server creates the structure of the ODI Master repository and Work repository in HDFS.
6. Click **Test Connection** to test the connection to the Hadoop data server.

### 3.2.1 Hadoop Data Server Definition

The following table describes the fields that you must specify on the Definition tab when creating a new Hadoop data server.

**Note:** Only the fields required or specific for defining a Hadoop data server are described.

**Table 3-6 Hadoop Data Server Definition**

| Field | Description |
|---|---|
| **Name** | Name of the data server that appears in Oracle Data Integrator. |
| **Data Server** | Physical name of the data server. |
| **User/Password** | Hadoop user with its password.<br>If password is not provided, only simple authentication is performed using the username on HDFS and Oozie. |
| **Authentication Method** | Select one of the following authentication methods:<br>• Simple Username Authentication (unsecured)<br>• Kerberos Principal Username/Password (secured)<br>• Kerberos Credential Ticket Cache (secured)<br>**Note:**<br>The following link helps determine if the Hadoop cluster is secured:<br>https://www.cloudera.com/documentation/cdh/5-0-x/CDH5-Security-Guide/cdh5sg_hadoop_security_enable.html |
| **HDFS Node Name URI** | URI of the HDFS node name.<br>`hdfs://localhost:8020` |
| **Resource Manager/Job Tracker URI** | URI of the resource manager or the job tracker.<br>`localhost:8032` |
| **ODI HDFS Root** | Path of the ODI HDFS root directory.<br>`/user/<login_username>/odi_home`. |
| **Additional Class Path** | Specify additional classpaths.<br>Add the following additional classpaths:<br>• `/usr/lib/hadoop/*`<br>• `/usr/lib/hadoop/client/*`<br>• `/usr/lib/hadoop/lib/*`<br>• `/usr/lib/hadoop-hdfs/*`<br>• `/usr/lib/hadoop-mapreduce/*`<br>• `/usr/lib/hadoop-yarn/*`<br>• `/usr/lib/hbase/lib/*`<br>• `/usr/lib/hive/lib/*`<br>• `/usr/lib/oozie/lib/*`<br>• `/etc/hadoop/conf/`<br>• `/etc/hbase/conf`<br>• `/etc/hive/conf`<br>• `/opt/oracle/orahdfs/jlib/*` |

Creating and Initializing the Hadoop Data Server

Configuring Big Data technologies using the Big Data Configurations Wizard.

## 3.2.2 Hadoop Data Server Properties

The following table describes the properties that you can configure in the Properties tab when defining a new Hadoop data server.

> **Note:**
>
> By default, only the `oracle.odi.prefer.dataserver.packages` property is displayed. Click the + icon to add the other properties manually.
>
> These properties can be inherited by other Hadoop technologies, such as Hive or HDFS. To inherit these properties, you must select the configured Hadoop data server when creating data server for other Hadoop technologies.

**Table 3-7 Hadoop Data Server Properties Mandatory for Hadoop and Hive**

| Property Group | Property | Description/Value |
|---|---|---|
| General | HADOOP_HOME | Location of Hadoop dir. For example, `/usr/lib/hadoop` |
| User Defined | HADOOP_CONF | Location of Hadoop configuration files such as core-default.xml, core-site.xml, and hdfs-site.xml. For example, `/home/shared/hadoop-conf` |
| Hive | HIVE_HOME | Location of Hive dir. For example, `/usr/lib/hive` |
| User Defined | HIVE_CONF | Location of Hive configuration files such as hive-site.xml. For example, `/home/shared/hive-conf` |
| General | HADOOP_CLASSPATH | `$HIVE_HOME/lib/hive-metastore-*.jar:$HIVE_HOME/lib/libthrift-*.jar:$HIVE_HOME/lib/libfb*.jar:$HIVE_HOME/lib/hive-exec-*.jar:$HIVE_CONF` |
| General | HADOOP_CLIENT_OPTS | `-Dlog4j.debug -Dhadoop.root.logger=INFO,console -Dlog4j.configuration=file:/etc/hadoop/conf.cloudera.yarn/log4j.properties` |

**Table 3-7 (Cont.) Hadoop Data Server Properties Mandatory for Hadoop and Hive**

| Property Group | Property | Description/Value |
|---|---|---|
| Hive | HIVE_SESSION_JARS | `$HIVE_HOME/lib/hive-contrib-*.jar:<ODI library directory>/wlhive.jar`<br>• Actual path of `wlhive.jar` can be determined under ODI installation home.<br>• Include other JAR files as required, such as custom SerDes JAR files. These JAR files are added to every Hive JDBC session and thus are added to every Hive MapReduce job.<br>• List of JARs is separated by ":", wildcards in file names must not evaluate to more than one file.<br>• Follow the steps for Hadoop Security models, such as Apache Sentry, to allow the Hive ADD JAR call used inside ODI Hive KMs:<br>– Define the environment variable HIVE_SESSION_JARS as empty.<br>– Add all required jars for Hive in the global Hive configuration hive-site.xml. |

**Table 3-8 Hadoop Data Server Properties Mandatory for HBase (In addition to base Hadoop and Hive Properties)**

| Property Group | Property | Description/Value |
|---|---|---|
| HBase | HBASE_HOME | Location of HBase dir. For example, `/usr/lib/hbase` |
| General | HADOOP_CLASSPATH | `$HBASE_HOME/lib/hbase-*.jar:$HIVE_HOME/lib/hive-hbase-handler*.jar:$HBASE_HOME/hbase.jar` |

Table 3-8 (Cont.) Hadoop Data Server Properties Mandatory for HBase (In addition to base Hadoop and Hive Properties)

| Property Group | Property | Description/Value |
|---|---|---|
| Hive | HIVE_SESSION_JARS | `$HBASE_HOME/hbase.jar:$HBASE_HOME/lib/hbase-sep-api-*.jar:$HBASE_HOME/lib/hbase-sep-impl-*hbase*.jar:/$HBASE_HOME/lib/hbase-sep-impl-common-*.jar:/$HBASE_HOME/lib/hbase-sep-tools-*.jar:$HIVE_HOME/lib/hive-hbase-handler-*.jar` |

**Note:**

Follow the steps for Hadoop Security models, such as Apache Sentry, to allow the Hive ADD JAR call used inside ODI Hive KMs:

**Table 3-8 (Cont.) Hadoop Data Server Properties Mandatory for HBase (In addition to base Hadoop and Hive Properties)**

| Property Group | Property | Description/Value |
|---|---|---|
| | | - Define the environment variable HIVE_SESSION_JARS as empty. |

**Table 3-8 (Cont.) Hadoop Data Server Properties Mandatory for HBase (In addition to base Hadoop and Hive Properties)**

| Property Group | Property | Description/Value |
|---|---|---|
| | | - Add all required jars for Hive in the global Hive configuratio |

**Table 3-8 (Cont.) Hadoop Data Server Properties Mandatory for HBase (In addition to base Hadoop and Hive Properties)**

| Property Group | Property | Description/Value |
|---|---|---|
| | | nhive-site.xml. |

**Table 3-9 Hadoop Data Server Properties Mandatory for Oracle Loader for Hadoop (In addition to base Hadoop and Hive properties)**

| Property Group | Property | Description/Value |
|---|---|---|
| OLH/OSCH | OLH_HOME | Location of OLH installation. For example, `/u01/connectors/olh` |
| OLH/OSCH | OLH_FILES | `usr/lib/hive/lib/hive-contrib-1.1.0-cdh5.5.1.jar` |
| OLH/OSCH | ODCH_HOME | Location of OSCH installation. For example, `/u01/connectors/osch` |
| General | HADOOP_CLASSPATH | `$OLH_HOME/jlib/*:$OSCH_HOME/jlib/*` |
| OLH/OSCH | OLH_JARS | Comma-separated list of all JAR files required for custom input formats, Hive, Hive SerDes, and so forth, used by Oracle Loader for Hadoop. All filenames have to be expanded without wildcards.<br>For example:<br>`$HIVE_HOME/lib/hive-metastore-0.10.0-cdh4.5.0.jar,$HIVE_HOME/lib/libthrift-0.9.0-cdh4-1.jar,$HIVE_HOME/lib/libfb303-0.9.0.jar` |

**Table 3-9 (Cont.) Hadoop Data Server Properties Mandatory for Oracle Loader for Hadoop (In addition to base Hadoop and Hive properties)**

| Property Group | Property | Description/Value |
|---|---|---|
| OLH/OSCH | OLH_SHAREDLIBS (deprecated) | `$OLH_HOME/lib/libolh12.so,$OLH_HOME/lib/libclntsh.so.12.1,$OLH_HOME/lib/libnnz12.so,$OLH_HOME/lib/libociei.so,$OLH_HOME/lib/libclntshcore.so.12.1,$OLH_HOME/lib/libons.so` |

**Table 3-10 Hadoop Data Server Properties Mandatory for SQOOP (In addition to base Hadoop and Hive properties)**

| Property Group | Property | Description/Value |
|---|---|---|
| SQOOP | SQOOP_HOME | Location of Sqoop directory. For example, `/usr/lib/sqoop` |
| SQOOP | SQOOP_LIBJARS | Location of the SQOOP library jars. For example, `usr/lib/hive/lib/hive-contrib.jar` |

Creating and Initializing the Hadoop Data Server

## 3.3 Creating a Hadoop Physical Schema

To create a physical schema for Hadoop, first create a logical schema for the same using the standard procedure.

Create a Hadoop physical schema using the standard procedure, as described in the Creating a Physical Schema section in *Administering Oracle Data Integrator*.

Create for this physical schema a logical schema using the standard procedure, as described in the Creating a Logical Schema section in *Administering Oracle Data Integrator* and associate it in a given context.

## 3.4 Configuring the Oracle Data Integrator Agent to Execute Hadoop Jobs

You must configure the Oracle Data Integrator agent to execute Hadoop jobs.

For information on creating a physical agent, see the Creating a Physical Agent section in *Administering Oracle Data Integrator*.

To configure the Oracle Data Integrator agent:

1. If the ODI agent is not installed on one of the Hadoop Cluster nodes, then you must install the Hadoop Client libraries on that computer.

For instructions on setting up a remote Hadoop client in Oracle Big Data Appliance, see the Providing Remote Client Access to CDH section in the *Oracle Big Data Appliance Software User's Guide* .

2. Install Hive on your Oracle Data Integrator agent computer.
3. Install SQOOP on your Oracle Data Integrator agent computer.
4. Set the base properties for Hadoop and Hive on your ODI agent computer.

   These properties must be added as Hadoop data server properties. For more information, see Hadoop Data Server Properties.
5. If you plan to use HBase features, set the properties on your ODI agent computer. You must set these properties in addition to the base Hadoop and Hive properties.

   These properties must be added as Hadoop data server properties. For more information, see Hadoop Data Server Properties.

## 3.5 Configuring Oracle Loader for Hadoop

If you want to use Oracle Loader for Hadoop, you must install and configure Oracle Loader for Hadoop on your Oracle Data Integrator agent computer.

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.

To install and configure Oracle Loader for Hadoop:

1. Install Oracle Loader for Hadoop on your Oracle Data Integrator agent computer.

   See the Installing Oracle Loader for Hadoop section in *Oracle Big Data Connectors User's Guide*.
2. To use Oracle SQL Connector for HDFS (`OLH_OUTPUT_MODE=DP_OSCH` or `OSCH`), you must first install it.

   See the Oracle SQL Connector for Hadoop Distributed File System Setup section in *Oracle Big Data Connectors User's Guide*.
3. Set the properties for Oracle Loader for Hadoop on your ODI agent computer. You must set these properties in addition to the base Hadoop and Hive properties.

   These properties must be added as Hadoop data server properties. For more information, see Hadoop Data Server Properties.

## 3.6 Configuring Oracle Data Integrator to Connect to a Secure Cluster

To run the Oracle Data Integrator agent on a Hadoop cluster that is protected by Kerberos authentication, you must configure a Kerberos-secured cluster.

To use a Kerberos-secured cluster:

1. Log in to the node of the Oracle Big Data Appliance, where the Oracle Data Integrator agent runs.
2. Set the environment variables by using the following commands. The user name in the following example is oracle. Substitute the appropriate values for your appliance:

   ```
   $ export KRB5CCNAME=Kerberos-ticket-cache-directory
   ```

```
$ export KRB5_CONFIG=Kerberos-configuration-file
```

```
$ export HADOOP_OPTS="$HADOOP_OPTS -
Djavax.xml.parsers.DocumentBuilderFactory=com.sun.org.apache.xerces.internal.
jaxp.DocumentBuilderFactoryImpl -Djava.security.krb5.conf=Kerberos-
configuration-file"
```

In this example, the configuration files are named krb5* and are located in /tmp/oracle_krb/:

```
$ export KRB5CCNAME=/tmp/oracle_krb/krb5cc_1000
```

```
$ export KRB5_CONFIG=/tmp/oracle_krb/krb5.conf
```

```
$ export HADOOP_OPTS="$HADOOP_OPTS -
Djavax.xml.parsers.DocumentBuilderFactory=com.sun.org.apache.xerces.internal.
jaxp.DocumentBuilderFactoryImpl -Djava.security.krb5.conf=/tmp/oracle_krb/
krb5.conf"
```

3. Generate a new Kerberos ticket for the oracle user. Use the following command, replacing realm with the actual Kerberos realm name.

   ```
   $ kinit oracle@realm
   ```

4. **ODI Studio:** To set the VM for ODI Studio, add AddVMoption in `odi.conf` in the same folder as `odi.sh`.

   **Kerberos configuration file location:**

   ```
   AddVMOption -Djava.security.krb5.conf=/etc/krb5.conf
   AddVMOption -Dsun.security.krb5.debug=true
   AddVMOption -Dsun.security.krb5.principal=odidemo
   ```

5. Redefine the JDBC connection URL, using syntax like the following:

**Table 3-11 Kerberos Configuration for Dataserver**

| Technology | Configuration | Example |
|---|---|---|
| Hadoop | No specific configuration to be done, general settings is sufficient. | |

**Table 3-11 (Cont.) Kerberos Configuration for Dataserver**

| Technology | Configuration | Example |
|---|---|---|
| Hive | `$MW_HOME/oracle_common/modules/datadirect/JDBCDriverLogin.conf`<br>**ODI Studio Configuration**:<br>Add the following content to `bin/odi.conf` file:<br>`AddVMOption -Djava.security.auth.login.config=<ORACLE_HOME>/oracle_common/modules/datadirect/JDBCDriverLogin.conf`<br>**ODI J2EE Agent Configuration**:<br>Add the following content to `<DOMAIN_HOME>/bin/setDomainEnv.sh`: file:<br>`export KRB5_CONFIG=/etc/krb5.conf`<br>`export KRB5CCNAME=/tmp/krb5cc_500`<br>`export ODI_ADDITIONAL_JAVA_OPTIONS="-Djava.security.krb5.conf=/etc/krb5.conf`<br>`-Djava.security.auth.login.config=/scratch/Oracle/Middleware/Oracle_Home/oracle_common/modules/datadirect/JDBCDriverLogin.conf`<br>`-Dsun.security.jgss.debug=true -Dsun.security.krb5.debug=true`<br>`-Djava.security.krb5.realm=SHARED.BDA.COM`<br>`-Djava.security.krb5.kdc=scaj43bda05.us.oracle.com:88`<br>`-Djavax.security.auth.useSubjectCredsOnly=false"`<br>`JAVA_PROPERTIES="${JAVA_PROPERTIES} ${WLP_JAVA_PROPERTIES} ${ODI_ADDITIONAL_JAVA_OPTIONS}"`<br>`export JAVA_PROPERTIES`<br>`JAVA_OPTIONS="${JAVA_OPTIONS} ${JAVA_PROPERTIES}"`<br>`export JAVA_OPTIONS` | **Example of configuration file**<br>`JDBC_DRIVER_01 {`<br>`com.sun.security.auth.module.Krb5LoginModule required`<br>`debug=true`<br>`useTicketCache=true`<br>`ticketCache="/tmp/krb5cc_500"`<br>`doNotPrompt=true`<br>`;`<br>`};`<br>**Example of Hive URL**<br>`jdbc:weblogic:hive://<hostname>:10000;DatabaseName=default;AuthenticationMethod=kerberos;ServicePrincipalName=<username>/<fully.qualified.domain.name>@<YOUR-REALM>.COM` |

**Table 3-11 (Cont.) Kerberos Configuration for Dataserver**

| Technology | Configuration | Example |
|---|---|---|
| HBase | `export HBASE_HOME=/scratch/fully.qualified.domain.name/etc/hbase/conf`<br>`export HBASE_CONF_DIR = $HBASE_HOME/conf`<br>`export HBASE_OPTS ="-Djava.security.auth.login.config=$HBASE_CONF_DIR/hbase-client.jaas"`<br>`export HBASE_MASTER_OPTS ="-Djava.security.auth.login.config=$HBASE_CONF_DIR/hbase-server.jaas"`<br><br>**ODI Studio Configuration**:<br>`AddVMOption -Djava.security.auth.login.config=$HBASE_CONF_DIR/hbase-client.jaas"` | **Example of Hbase configuration file:**<br><br>`hbase-client.jaas`<br>`Client {`<br>`com.sun.security.auth.module.Krb5LoginModule required`<br>`useKeyTab=false`<br>`useTicketCache=true;`<br>`};` |
| Spark | Spark Kerberos configuration is done through spark submit parameters<br><br>`--principal // define principle name`<br>`--keytab     // location of keytab file` | **Example of spark-submit command:**<br><br>`spark-submit --master yarn --py-files  /tmp/pyspark_ext.py --executor-memory 1G --driver-memory 512M --executor-cores 1 --driver-cores 1 --num-executors 2 --principal fully.qualified.domain.name@YOUR-REALM.com --keytab /tmp/fully.qualified.domain.name.tab --queue default /tmp/New_Mapping_Physical.py` |

**Table 3-11 (Cont.) Kerberos Configuration for Dataserver**

| Technology | Configuration | Example |
|---|---|---|
| Kafka | Kafka Kerberos configuration is done through kafka-client.jaas file: The configuration file is placed in Kafka configuration folder. | **Example of Kafka configuration file:**<br>`KafkaClient {`<br>`com.sun.security.auth.module.Krb5LoginModule required`<br>`useKeyTab=false`<br>`useTicketCache=true`<br>`ticketCache="/tmp/krb5cc_1500"`<br>`serviceName="kafka";`<br>`};`<br>The location of Kafka configuration file is set in ODI Studio VM option<br>`AddVMOption -Djava.security.auth.login.config="/etc/kafka-jaas.conf"` |
| Pig/Oozie | Pig and Ooize will extend the Kerberos configuration of linked Hadoop data server and does not require specific configuration. | |

For more information on these properties and settings, see "HiveServer2 Security Configuration" in the CDH5 Security Guide at the following URL:

`https://www.cloudera.com/documentation/cdh/5-0-x/CDH5-Security-Guide/cdh5sg_hiveserver2_security.html`

6. Renew the Kerberos ticket for the Oracle user on a regular basis to prevent disruptions in service.
7. Download the unlimited strength JCE security jars.

   For instructions about managing Kerberos on Oracle Big Data Appliance, see the About Accessing a Kerberos-Secured Cluster section in *Oracle Big Data Appliance Software User's Guide* .

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

Perform the following configuration steps to execute Hadoop jobs on the local agent of Oracle Data Integrator Studio.

For executing Hadoop jobs on the local agent of an Oracle Data Integrator Studio installation, follow the configuration steps in Configuring the Oracle Data Integrator Agent to Execute Hadoop Jobs with the following change:

Copy the following Hadoop client jar files to the local machines.

```
/usr/lib/hadoop/*.jar
/usr/lib/hadoop/lib/*.jar
/usr/lib/hadoop/client/*.jar
/usr/lib/hadoop-hdfs/*.jar
/usr/lib/hadoop-mapreduce/*.jar
/usr/lib/hadoop-yarn/*.jar
/usr/lib/oozie/lib/*.jar
/usr/lib/hive/*.jar
/usr/lib/hive/lib/*.jar
/usr/lib/hbase/*.jar
/usr/lib/hbase/lib/*.jar
```

Add the above classpaths in the `additional_path.txt` file under the `userlib` directory.

For example:

Linux: `$USER_HOME/.odi/oracledi/userlib` directory.

Windows: `C:\Users\<USERNAME>\AppData\Roaming\odi\oracledi\userlib` directory

# 4 Integrating Hadoop Data

This chapter provides information about the steps you need to perform to integrate Hadoop data.
This chapter includes the following sections:

- Integrating Hadoop Data
- Setting Up File Data Sources
- Setting Up HDFS Data Sources
- Setting Up Hive Data Sources
- Setting Up HBase Data Sources
- Setting Up Kafka Data Sources
- Setting Up Cassandra Data Sources
- Importing Hadoop Knowledge Modules
- Creating ODI Models and Data Stores to represent Hive, HBase and Cassandra Tables, and HDFS Files
- Password Handling in Hadoop
- Loading Data from Files into Hive
- Loading Data from Hive to Files
- Loading Data from HBase into Hive
- Loading Data from Hive into HBase
- Loading Data from an SQL Database into Hive, HBase, and File using SQOOP
- Loading Data from an SQL Database into Hive using SQOOP
- Loading Data from an SQL Database into HDFS File using SQOOP
- Loading Data from an SQL Database into HBase using SQOOP
- Validating and Transforming Data Within Hive
- Loading Data into an Oracle Database from Hive and File
- Loading Data into an SQL Database from Hbase, Hive, and File using SQOOP
- Loading Data from Kafka to Spark Processing Engine

## 4.1 Integrating Hadoop Data

To integrate Hadoop data, set up data sources, import Hadoop knowledge modules, create oracle data integrator models and design mappings to load, validate, and transform Hadoop data.

The following table summarizes the steps for integrating Hadoop data.

**Table 4-1 Integrating Hadoop Data**

| Step | Description |
|---|---|
| Set Up Data Sources | Set up the data sources to create the data source models. You must set up File, Hive, HDFS, and HBase data sources.<br>See Setting Up File Data Sources<br>See Setting Up Hive Data Sources<br>See Setting Up HBase Data Sources<br>See Setting Up Kafka Data Sources<br>See Setting Up Cassandra Data Sources<br>See Setting Up HDFS Data Sources |
| Import Hadoop Knowledge Modules | Import the Hadoop KMs into Global Objects or a project.<br>See Importing Hadoop Knowledge Modules |
| Create Oracle Data Integrator Models | Reverse-engineer the Hive and HBase models to create Oracle Data Integrator models.<br>See Creating ODI Models and Data Stores to represent Hive, HBase and Cassandra Tables, and HDFS Files |
| Configure Hadoop Credential Provider | Configure Hadoop Credential Provider and define the password.<br>See Password Handling in Hadoop. |
| Integrate Hadoop Data | Design mappings to load, validate, and transform Hadoop data.<br>See Loading Data from Files into Hive<br>See Loading Data from HBase into Hive<br>See Loading Data from Hive into HBase<br>See Loading Data from an SQL Database into Hive, HBase, and File using SQOOP<br>See Validating and Transforming Data Within Hive<br>See Loading Data into an Oracle Database from Hive and File<br>See Loading Data into an SQL Database from Hbase, Hive, and File using SQOOP<br>See Loading Data from Kafka to Spark Processing Engine<br>See Loading Data from HDFS File to Hive<br>See Loading Data from HDFS File to Spark<br>See Loading Data from Hive to Files |

## 4.2 Setting Up File Data Sources

To setup file data sources, you need to create a data server object under File technology along with a physical and logical schema for every directory to be accessed.

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

**To define a data source:**

1. Create a Data Server object under File technology.
2. Create a Physical Schema object for every directory to be accessed.
3. Create a Logical Schema object for every directory to be accessed.
4. Create a Model for every Logical Schema.

5. Create one or more data stores for each different type of file and wildcard name pattern.
6. HDFS Files are now created using the HDFS Technology as seen in Setting Up HDFS Data Sources. However, for backward compatibility, there are some Big Data File Knowledge Modules that support HDFS Files. To define HDFS files, you must select **HDFS File** and define the Hadoop DataServer reference. Alternatively, you can create a Data Server object under File technology by entering the HDFS name node in the field JDBC URL and leave the JDBC Driver name empty. For example:

   ```
   hdfs://bda1node01.example.com:8020
   ```

   Test Connection is not supported for this Data Server configuration.

Integrating Hadoop Data

## 4.3 Setting Up HDFS Data Sources

To setup HDFS data sources, you need to create a data server object under HDFS technology along with a physical and logical schema for every directory to be accessed.

This topic provides steps in Oracle Data Integrator that are required for connecting to a HDFS system.

1. Create a Data Server object under HDFS technology.

   > **Note:**
   >
   > HDFS data server should reference an existing Hadoop data server.

2. Create a Physical Schema object for every directory to be accessed.
3. Create a Logical Schema object for every directory to be accessed.
4. Create a Model for every Logical Schema
5. Create one or more data stores for each different type of file.

   The definition tab has a Resource Name field that enables you to specify which file or files it represents. If wildcards are used, the files must have the same schema and be of the same format (all JSON or all Avro).
6. Select the appropriate **Storage Format** and the **Schema File**.

   The contents of the schema are displayed.
7. Select the **Attributes Tab** to either enter, or reverse-engineer the Attributes from the supplied schema.

## 4.4 Setting Up Hive Data Sources

To setup Hive data sources, you need to create a data server object under Hive technology. Oracle Data Integrator connects to Hive by using JDBC.

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

Oracle Data Integrator connects to Hive by using JDBC.

**To set up a Hive data source:**

1. Create a Data Server object under Hive technology.

2. Set the following locations under JDBC:

   JDBC Driver: `weblogic.jdbc.hive.HiveDriver`

   JDBC URL: `jdbc:weblogic:hive://<host>:<port>[; property=value[;...]]`

   For example, `jdbc:weblogic:hive://localhost:10000;DatabaseName=default;User=default;Password=default`

   > **Note:**
   >
   > Usually User ID and Password are provided in the respective fields of an ODI Data Server. In case where a Hive user is defined without password, you must add `password=default` as part of the JDBC URL and the password field of Data Server shall be left blank.

3. Set the following under on the definition tab of the data server:

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

4. Ensure that the Hive server is up and running.
5. Test the connection to the Data Server.
6. Create a Physical Schema. Enter the name of the Hive schema in both schema fields of the Physical Schema definition.
7. Create a Logical Schema object.
8. Import RKM Hive into Global Objects or a project.
9. Create a new model for Hive Technology pointing to the logical schema.
10. Perform a custom reverse-engineering operation using RKM Hive.

Reverse-engineered Hive table populates the attribute and storage tabs of the data store.

Integrating Hadoop Data

## 4.5 Setting Up HBase Data Sources

To setup HBase data sources, you need to create a data server object under HBase technology along with a physical and logical schema object.

The following steps in Oracle Data Integrator are required for connecting to a HBase system.

**To set up a HBase data source:**

1. Create a Data Server object under HBase technology.

   JDBC Driver and URL are not available for data servers of this technology.

2. Set the following under on the definition tab of the data server:

   HBase Quorum: Quorum of the HBase installation. For example: `zkhost1.example.com,zkhost2.example.com,zkhost3.example.com`

3. Ensure that the HBase server is up and running.

> **Note:**
>
> You cannot test the connection to the HBase Data Server.

4. Create a Physical Schema.
5. Create a Logical Schema object.
6. Import RKM HBase into Global Objects or a project.
7. Create a new model for HBase Technology pointing to the logical schema.
8. Perform a custom reverse-engineering operation using RKM HBase.

> **Note:**
>
> Ensure that the HBase tables contain some data before performing reverse-engineering. The reverse-engineering operation does not work if the HBase tables are empty.

At the end of this process, the HBase Data Model contains all the HBase tables with their columns and data types.

Integrating Hadoop Data

## 4.6 Setting Up Kafka Data Sources

To setup kafka data sources, you need to create a data server object under Kafka technology along with a physical and logical schema object. Create one or more data sources for each different topic and then test the connection to the Data Server.

This following procedure describes how to connect to a Kafka system in Oracle Data Integrator.

1. Create a Data Server object under Kafka technology.

   For information on creating a Kafka data server, see Kafka Data Server Definition and Kafka Data Server Properties.
2. Create a Physical Schema object.
3. Create a Logical Schema object.
4. Create a Model for every Logical Schema
5. Create one or more data stores for each different topic.

   Resource Name in the definition tab of data store indicates the Kafka topic . Kafka topic name can be either entered by the user or selected from the list of available Kafka topics in the Kafka cluster. There are two ways to load data from Kafka topics which are receiver-based and direct and LKM Kafka to Spark supports both approaches.
6. Test the connection to the Data Server.

   For information on Kafka Integration, see Hadoop Data Integration with Oracle Data Integrator.

The Kafka data model contains all the Kafka topics with their columns and data types.

## 4.7 Setting Up Cassandra Data Sources

To setup Cassandra data sources, you need to create a data server object under Casssandra technology. Oracle Data Integrator connects to Cassandra by using JDBC.

This following procedure describes how to connect to a Cassandra system in Oracle Data Integrator.

1. Create a Data Server object under Cassandra technology.
2. Set the following locations under JDBC:

   Add the Cassandra JDBC Driver to the Driver List.
   JDBC Driver: `weblogic.jdbc.cassandra.CassandraDriver`

   JDBC URL: `jdbc:weblogic:cassandra://<host>:<port>[;property=value[:...]]`

   For example, `jdbc:weblogic:cassandra://cassandra.example.com:9042;KeyspaceName=mykeyspace`

   > **Note:**
   >
   > Latest driver uses the binary protocol and hence uses default port 9042.

3. Ensure that the Cassandra server is up and running.
4. Test the connection to the Data Server.
5. Create a Physical Schema object.
6. Create a Logical Schema object.
7. Import RKM Cassandra into Global Objects or a project.
8. Create a Model for every Logical Schema
9. Perform a custom reverse-engineering operation using RKM Cassandra.

## 4.8 Importing Hadoop Knowledge Modules

Unlike other built-in Big Data Knowledge Modules, you need to import RKMs and CKMs into your project or as global objects before you use them.

Most of the Big Data Knowledge Modules are built-in the product. The exceptions are the RKMs and CKMs, and these will need to be imported into your project or as global objects before you use them. They are:

- CKM Hive
- RKM Hive
- RKM HBase
- RKM Cassandra

Integrating Hadoop Data

# 4.9 Creating ODI Models and Data Stores to represent Hive, HBase and Cassandra Tables, and HDFS Files

You must create ODI models to hold the data stores that represent HDFS files or Hive, HBase and Cassandra tables. The reverse-engineering process creates Hive, HBase and Cassandra data stores for the corresponding Hive, HBase and Cassandra tables. You can use these data stores as source or target in your mappings.

This section contains the following topics:

- Creating a Model
- Reverse-Engineering Hive Tables
- Reverse-Engineering HBase Tables
- Reverse-Engineering HDFS Files
- Reverse-Engineering Cassandra Tables
- Reverse-Engineering Support for Kafka

## 4.9.1 Creating a Model

To create a model that is based on the technology hosting Hive, HBase, Cassandra, or HDFS and on the logical schema created when you configured the Hive, HBase, Cassandra, HDFS or File connection, follow the standard procedure described in *Developing Integration Projects with Oracle Data Integrator*.

For backward compatibility, the Big Data LKMs reading from Files (LKM File to Hive LOAD DATA), also support reading from HDFS, however the source data store must be from a file model. If reading from HDFS, it is preferable to use KMs like the LKM HDFS to File LOAD DATA . In this case, the source data store must be from an HDFS model.

## 4.9.2 Reverse-Engineering Hive Tables

RKM Hive is used to reverse-engineer Hive tables and views. To perform a customized reverse-engineering of Hive tables with RKM Hive, follow the usual procedures, as described in *Developing Integration Projects with Oracle Data Integrator*. This topic details information specific to Hive tables.

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

For more information about RKM Hive, see RKM Hive.

Hive data stores contain a storage tab allowing you to see how data is stored and formatted within Hive. If the Hive table has been reverse-engineered, then these fields will be automatically populated. If you created this data store from the beginning, with the intention of creating the table when running a mapping (using create target table), then you can choose how the data is formatted by editing these fields.

The target Hive table is created based on the data provided in the Storage and Attribute panels of the Hive data store as shown in Table 4-2 and Table 4-3.

**Table 4-2 Hive Data Store Storage Panel Properties**

| Property | Description |
|---|---|
| **Table Type** | Select one of the following as the type of Hive table to be created:<br>• **Managed**<br>• **External**<br>• **<Undefined>** |
| **Storage Type** | Select one of the following as the type of Data storage:<br>• **Native**<br>• **Non-Native**<br>• **<Undefined>** |
| **Row Format** | This property appears when **Native** is selected as the Storage Type.<br>Select one of the following as the Row Format:<br>• **Built-In**<br>• **Delimited**<br>• **SerDe**<br>• **<Undefined>** |
| **Record Separator** | This property appears when **Delimited** is selected as the Row Format.<br>Fill in the following fields:<br>• **Fields Terminated By**<br>• **Fields Escaped By**<br>• **Collection Items Terminated By**<br>• **Map Keys Terminated By**<br>• **Lines Terminated By**<br>• **File Null Value** |
| **SerDe** | This property appears when **SerDe** is selected as the Row Format.<br>Fill in the **SerDe Class** field. |

Table 4-2 (Cont.) Hive Data Store Storage Panel Properties

| Property | Description |
|---|---|
| **Storage Format** | This longer Storage Format section appears when **Native** is selected as the Storage Type.<br>It contains the following properties:<br>• **Predefined File Format**<br>• **Input Format** (appears when **INPUTFORMAT** is selected as the Predefined File Format.)<br>• **Output Format** (appears when **INPUTFORMAT** is selected as the Predefined File Format.)<br>• **Location** (appears when **External** is selected as the Table Type.)<br>Select one of the following as the Predefined File Format:<br>• **INPUTFORMAT**<br>• **SEQUENCEFILE**<br>• **PARQUET**<br>• **TEXTFILE**<br>• **ORC**<br>• **JSON**<br>• **RCFILE**<br>• **AVRO**<br>• **<Undefined>** |
| **Storage Handler** | This property appears when **Non-Native** is selected as the Storage Type.<br>Fill in the **Storage Handler Class** field. |
| **Storage Format** | This shorter Storage Format section appears when **Non-Native** is selected as the Storage Type.<br>Fill in the **Location** field. |

Table 4-3 Hive Data Store Attribute Panel Properties

| Property | Description |
|---|---|
| **Order** | Order in which attributes are sequenced. |
| **Name** | Name of the attribute. |
| **Type** | Data type of the attribute. |

**Table 4-3 (Cont.) Hive Data Store Attribute Panel Properties**

| Property | Description |
|---|---|
| **Data Format** | Data Format of the attribute.<br>**Note:**<br>This field is only used for attributes with a data type of "Complex". The content is populated during reverse-engineering and will contain a definition of the Complex Type. |
| **Length** | Physical length of the attribute. |
| **Scale** | Scale of the numeric attribute. |
| **Not Null** | Specifies if the attribute can be null or not. |
| **SCD Behavior** | This is not used for Hive data stores. |
| **Partition By** | Select if it is a partition column. |
| **Cluster By** | Select if it is a bucketed column. |
| **Sort By** | Select to sort data on this column within the bucket.<br>**Note:**<br>You must set the position of this column in the SORTED BY clause. The column whose Sort By value is smaller will get the higher priority. For example, consider three columns, C1 with Sort By = 5, C2 with Sort By = 2, C3 with Sort By = 8. The SORTED BY clause will be SORTED BY (C2, C1, C3). |
| **Sort Direction** | Select to sort data in the ascending (ASC) or descending (DESC) order. |

The data provided above can also be used to create a Hive DDL when the CREATE_TARG_TABLE option is selected in the LKMs and IKMs.

To fully use the Hive format and storage information, one or more of the following KMs must be used:

- IKM Hive Append
- IKM Hive Incremental Update
- LKM File to Hive LOAD DATA Direct
- LKM HDFS File to Hive LOAD DATA Direct

## 4.9.3 Reverse-Engineering HBase Tables

RKM HBase is used to reverse-engineer HBase tables. To perform a customized reverse-engineering of HBase tables with RKM HBase, follow the usual procedures, as described in *Developing Integration Projects with Oracle Data Integrator*. This topic details information specific to HBase tables.

The reverse-engineering process creates the data stores for the corresponding HBase table. You can use the data stores as either a source or a target in a mapping.

> **Note:**
>
> Ensure that the HBase tables contain some data before performing reverse-engineering. The reverse-engineering operation does not work if the HBase tables are empty.

For more information about RKM HBase, see RKM HBase.

## 4.9.4 Reverse-Engineering HDFS Files

HDFS files are represented using data stores based on HDFS technology. The HDFS data stores contain the storage format (JSON, Delimited, etc.), attributes, datatypes, and datatype properties.

In previous versions of ODI, File technology was used to represent HDFS Files, but the storage format information was specified in the mappings. If you have existing mappings that use Knowledge Modules such as LKM File to Hive or LKM File to Spark, then you should continue to represent your HDFS files with File technology.

> **Note:**
>
> The preferred method of representing HDFS files is by using the HDFS technology.

### Reverse-Engineering HDFS Files into HDFS Data Stores

To reverse-engineer an HDFS file, perform the following steps:

1. Create a HDFS data store.
2. From the Storage Tab, select the Storage Format from the **Storage Format** drop-down list and specify the complete path of the schema file in the **Schema File** field.

   The schema file should be located in the local file system.

3. Click **Reverse Engineer** operation from the Attributes Tab of the HDFS data store.

> **Note:**
>
> - There is no need to import an RKM into the project.
> - HDFS reverse-engineering requires a Schema (JSON, Parquet, or Avro), hence HDFS files with a Delimited format cannot be reverse-engineered.

For more information, see the Reverse-engineer a File Model section in *Connectivity and Knowledge Modules Guide for Oracle Data Integrator Developer's Guide* .

Creating ODI Models and Data Stores to represent Hive, HBase and Cassandra Tables, and HDFS Files

**Reverse-Engineering HDFS Files in File Data Stores**

HDFS files can be reverse-engineered like regular files. To reverse-engineer HDFS files, you must copy them to your File System and follow the same process as that to reverse-engineer regular files.

> **Note:**
>
> If the file is large for your local File System, retrieve the first N records from HDFS and place them in a local file.

## 4.9.5 Reverse-Engineering Cassandra Tables

RKM Cassandra is used to reverse-engineer Cassandra tables. To perform a customized reverse-engineering of Cassandra tables with RKM Cassandra, follow the usual procedures, as described in *Developing Integration Projects with Oracle Data Integrator*.

The reverse-engineering process creates the data stores for the corresponding Cassandra table. For more information about RKM Cassandra, see RKM Cassandra.

## 4.9.6 Reverse-Engineering Support for Kafka

Reverse-engineering for Kafka is very similar to reverse-engineering HDFS files.

Create a model based on Kafka technology. Create a data store in that model as mentioned below:

1. Go to the Definition panel and enter **Name** and **Resource Name**.
2. Go to the Storage panel, select the **Storage Format** and specify the path of the **Schema File**.

   The Schema File has to be locally accessible.

> **Note:**
>
> The Storage Format can be AVRO, JSON, or PARQUET. The DELIMITED Storage Format is not supported for reverse-engineering. Use Data Store Editor to create a Kafka data store with DELIMITED Storage format.

3. Go to the Attribute panel and click **Reverse Engineer**.

   All the attributes specified in the Schema File are listed here.

## 4.10 Password Handling in Hadoop

Before using LKM SQL to Spark, LKM Spark to SQL, and LKM Spark to Cassandra, the Hadoop Credential Provider has to be configured and the password has to be defined.

To use these KMs, it is mandatory to follow the below procedure:

1. **Configure the Hadoop Credential Provider**.

   JDBC connection passwords are stored using the Hadoop Credential API. This requires the Hadoop cluster to be configured with at least one Credential Provider.

   Below is an example:

   ```
   <property>
   <name>hadoop.security.credential.provider.path</name>
   <value>user:///,jceks://file/tmp/test.jceks,jceks://hdfs@cluster1-ns/my/path/test.jceks</value>
   </property>
   ```

   > **Note:**
   >
   > The property in the example above should be defined in core-site.xml or its equivalent.

   For the proper configuration applicable to your system and security configuration/needs, see CredentialProvider API Guide.

2. **Create a password alias in Hadoop Credential Provider**.

   Once the Hadoop cluster is configured, you must create a credential for each password that Spark will be using to connect to the SQL source or target. ODI will assume the following format for credential alias names:

   ```
   odi.<user_name>.<dataserver_name>.password
   ```

   The `user_name` and `dataserver_name` are obtained from the ODI topology DataServer properties.

   The example below shows the creation of a password alias in Hadoop Credential Provider where the user name is `oracle` and dataserver is `Hadoop_CDH5_5`.

   ```
   hadoop credential create odi.oracle.Hadoop_CDH5_5.password
   ```

## 4.11 Loading Data from Files into Hive

To load data from files into Hive, create data stores for local and HDFS files and create a mapping after which you can select the option LKM file to Hive Load data, to load data from flat files to Hive.

The LKM File to Hive KMs support loading data from HDFS Files and, also local Files. However, if you are using HDFS files, the preferred way is to use the HDFS KMs, as described in Loading Data from HDFS into Hive.

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

   For information on reverse-engineering and configuring local file data sources, see *Connectivity and Knowledge Modules Guide for Oracle Data Integrator Developer's Guide*.
2. Create a mapping using the file data store as the source and the corresponding Hive table as the target.
3. Use the LKM File to Hive LOAD DATA or the LKM File to Hive LOAD DATA Direct knowledge module specified in the physical diagram of the mapping.

   These integration knowledge modules load data from flat files into Hive, replacing or appending any existing data.

For more information about the KMs, see the following sections:

- LKM File to Hive LOAD DATA
- LKM File to Hive LOAD DATA Direct

## 4.12 Loading Data from Hive to Files

To load data from Hive tables to local file system or HDFS files, create data store for the Hive tables and create a mapping after which you can select the option LKM Hive to File Direct Knowledge module, to load data from Hive to flat files.

To load data from Hive tables to a local file system or a HDFS file:

1. Create a data store for the Hive tables that you want to load in flat files.

   For information about reverse-engineering and configuring Hive data sources, see Setting Up Hive Data Sources.
2. Create a mapping using the Hive data store as the source and the corresponding File data source as the target.
3. Use the LKM Hive to File Direct knowledge module, specified in the physical diagram of the mapping.

   This integration knowledge module loads data from Hive into flat Files.

   For more information about LKM Hive to File Direct, see LKM Hive to File Direct.

## 4.13 Loading Data from HBase into Hive

To load data from HBase table into Hive, create data store for the HBase table and create a mapping after which you can select the option LKM HBase to Hive HBASE-SERDE knowledge module, to load data from HBase table into Hive.

To load data from an HBase table into Hive:

1. Create a data store for the HBase table that you want to load in Hive.

   For information about reverse-engineering and configuring HBase data sources, see Setting Up HBase Data Sources.
2. Create a mapping using the HBase data store as the source and the corresponding Hive table as the target.
3. Use the LKM HBase to Hive HBASE-SERDE knowledge module, specified in the physical diagram of the mapping.

   This knowledge module provides read access to an HBase table from Hive.

For more information about LKM HBase to Hive HBASE-SERDE, see LKM HBase to Hive HBASE-SERDE.

## 4.14 Loading Data from Hive into HBase

To load data from Hive to HBase table, create data store for the Hive tables and create a mapping after which you can select the option LKM Hive to HBase Incremental Update HBASE-SERDE Direct knowledge module, to load data from Hive table into HBase.

To load data from a Hive table into HBase:

1. Create a data store for the Hive tables that you want to load in HBase.

   For information about reverse-engineering and configuring Hive data sources, see Setting Up Hive Data Sources.
2. Create a mapping using the Hive data store as the source and the corresponding HBase table as the target.
3. Use the LKM Hive to HBase Incremental Update HBASE-SERDE Direct knowledge module, specified in the physical diagram of the mapping.

   This integration knowledge module loads data from Hive into HBase and supports inserting new rows and, also updating existing data.

For more information about LKM Hive to HBase Incremental Update HBASE-SERDE Direct, see LKM Hive to HBase Incremental Update HBASE-SERDE Direct.

## 4.15 Loading Data from an SQL Database into Hive, HBase, and File using SQOOP

To load data from an SQL Database into Hive, HBase, and File using SQOOP create a data store for the SQL source and create a mapping after which you can select the option IKM SQL to Hive-HBase-File (SQOOP) knowledge module, to load data from a SQL source into Hive, HBase, or Files target using SQOOP.

To load data from an SQL Database into a Hive, HBase, and File target:

1. Create a data store for the SQL source that you want to load into Hive, HBase, or File target.

   For information about reverse-engineering and configuring SQL data sources, see *Connectivity and Knowledge Modules Guide for Oracle Data Integrator Developer's Guide*.
2. Create a mapping using the SQL source data store as the source and the corresponding HBase table, Hive table, or HDFS files as the target.

3. Use the IKM SQL to Hive-HBase-File (SQOOP) knowledge module, specified in the physical diagram of the mapping.

   > **Note:**
   >
   > The IKM SQL to Hive-HBase-File (SQOOP) is not seeded and has to be manually imported.

   This integration knowledge module loads data from a SQL source into Hive, HBase, or Files target. It uses SQOOP to load the data into Hive, HBase, and File targets. SQOOP uses parallel JDBC connections to load the data.

For more information about IKM SQL to Hive-HBase-File (SQOOP), see IKM SQL to Hive-HBase-File (SQOOP) [Deprecated].

## 4.16 Loading Data from an SQL Database into Hive using SQOOP

To load data from an SQL Database into Hive using SQOOP create a data store for the SQL source and create a mapping after which you can select the option LKM SQL to Hive SQOOP knowledge module, to load data from a SQL source into Hive using SQOOP.

To load data from an SQL Database into a Hive target:

1. Create a data store for the SQL source that you want to load into Hive target.

   For information about reverse-engineering and configuring SQL data sources, see *Connectivity and Knowledge Modules Guide for Oracle Data Integrator Developer's Guide*.
2. Create a mapping using the SQL source data store as the source and the corresponding Hive table as the target.
3. Use the LKM SQL to Hive SQOOP knowledge module, specified in the physical diagram of the mapping.

   This KM loads data from a SQL source into Hive. It uses SQOOP to load the data into Hive. SQOOP uses parallel JDBC connections to load the data.

For more information about LKM SQL to Hive SQOOP, see LKM SQL to Hive SQOOP.

## 4.17 Loading Data from an SQL Database into HDFS File using SQOOP

To load data from an SQL Database into a HDFS File target:

1. Create a data store for the SQL source that you want to load into HDFS File target.

   For information about reverse-engineering and configuring SQL data sources, see *Connectivity and Knowledge Modules Guide for Oracle Data Integrator Developer's Guide*.
2. Create a mapping using the SQL source data store as the source and the corresponding HDFS files as the target.
3. Use the LKM SQL to File SQOOP Direct knowledge module, specified in the physical diagram of the mapping.

This integration knowledge module loads data from a SQL source into HDFS Files target. It uses SQOOP to load the data into File targets. SQOOP uses parallel JDBC connections to load the data.

For more information about IKM SQL to Hive-HBase-File (SQOOP), see IKM SQL to Hive-HBase-File (SQOOP) [Deprecated].

## 4.18 Loading Data from an SQL Database into HBase using SQOOP

To load data from an SQL Database into a HBase target:

1. Create a data store for the SQL source that you want to load into HBase target.

   For information about reverse-engineering and configuring SQL data sources, see *Connectivity and Knowledge Modules Guide for Oracle Data Integrator Developer's Guide*.
2. Create a mapping using the SQL source data store as the source and the corresponding HBase table as the target.
3. Use the LKM SQL to HBase SQOOP Direct knowledge module, specified in the physical diagram of the mapping.

   This integration knowledge module loads data from a SQL source into HBase target. It uses SQOOP to load the data into HBase targets. SQOOP uses parallel JDBC connections to load the data.

For more information about LKM SQL to HBase SQOOP Direct, see LKM SQL to HBase SQOOP Direct.

## 4.19 Validating and Transforming Data Within Hive

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

> **Note:**
>
> IKM Hive Control Append, CKM Hive, and IKM Hive Transform have to be imported.

- IKM Hive Control Append

  For more information, see IKM Hive Append.
- IKM Hive Append

  For more information, see IKM Hive Append.
- IKM Hive Incremental Update

  For more information, see IKM Hive Incremental Update.
- CKM Hive

  For more information, see CKM Hive.
- IKM Hive Transform

  For more information, see IKM Hive Transform (Deprecated).

## 4.20 Loading Data into an Oracle Database from Hive and File

Use the knowledge modules listed in the following table to load data from an HDFS file or Hive source into an Oracle database target using Oracle Loader for Hadoop.

**Table 4-4 Knowledge Modules to load data into Oracle Database**

| Knowledge Module | Use To... |
|---|---|
| IKM File-Hive to Oracle (OLH-OSCH) | Load data from an HDFS file or Hive source into an Oracle database target using Oracle Loader for Hadoop.<br>For more information, see IKM File-Hive to Oracle (OLH-OSCH) [Deprecated].<br>**Note:** This KM has to be imported. |
| LKM File to Oracle OLH-OSCH | Load data from an HDFS file into an Oracle staging table using Oracle Loader for Hadoop.<br>For more information, see LKM File to Oracle OLH-OSCH. |
| LKM File to Oracle OLH-OSCH Direct | Load data from an HDFS file into an Oracle database target using Oracle Loader for Hadoop.<br>For more information, see LKM File to Oracle OLH-OSCH Direct. |
| LKM Hive to Oracle OLH-OSCH | Load data from a Hive source into an Oracle staging table using Oracle Loader for Hadoop.<br>For more information, see LKM Hive to Oracle OLH-OSCH. |
| LKM Hive to Oracle OLH-OSCH Direct | Load data from a Hive source into an Oracle database target using Oracle Loader for Hadoop.<br>For more information, see LKM Hive to Oracle OLH-OSCH Direct. |

## 4.21 Loading Data into an SQL Database from Hbase, Hive, and File using SQOOP

Use the knowledge modules listed in the following table to load data from a HDFS file, HBase source, or Hive source into an SQL database target using SQOOP.

**Table 4-5    Knowledge Modules to load data into SQL Database**

| Knowledge Module | Use To... |
|---|---|
| IKM File-Hive to SQL (SQOOP) | Load data from an HDFS file or Hive source into an SQL database target using SQOOP.<br>For more information, see IKM File-Hive to SQL (SQOOP) [Deprecated].<br>**Note:**<br>This KM has to be imported. |
| LKM HBase to SQL SQOOP | Load data from an HBase source into an SQL database target using SQOOP.<br>For more information, see LKM HBase to SQL SQOOP. |
| LKM File to SQL SQOOP | Load data from an HDFS file into an SQL database target using SQOOP.<br>For more information, see LKM File to SQL SQOOP. |
| LKM Hive to SQL SQOOP | Load data from a Hive source into an SQL database target using SQOOP.<br>For more information, see LKM Hive to SQL SQOOP. |

## 4.22 Loading Data from Kafka to Spark Processing Engine

Loading data from Kafka to Spark.

1. Create a data store for the Kafka tables that you want to load in Spark.

   For configuring Kafka data sources, see Setting Up Kafka Data Sources.

2. Create a mapping using the Kafka data store as the source and the File/HDFS/SQL/Hive/Kafka data store as the target. Use Spark Python Physical Schema as the staging location.

   For more information, see Creating a Spark Physical Schema.

3. Use the Storage function KM option with the value `createStream` for a receiver-based connection or the value `createDirectStream` for a direct connection as specified in the physical diagram of the mapping.

   Set the `zookeeper.connect` and `metadata.broker.list` Kafka data server properties for the appropriate connection.

   This knowledge module loads data from Kafka into the Spark processing engine. You can use other knowledge modules to load data from Spark into File/HDFS/SQL/Hive/Kafka.

> **Note:**
>
> Every Kafka source in an ODI mapping allocates a Spark executor. A Spark Kafka mapping hangs if the number of available executors is low. The number of executors must be atleast n+1 where n is the number of Kafka sources in the mapping. For additional information, refer to Spark Documentation.

For more information about LKM Kafka to Spark, see LKM Kafka to Spark.

5

# Executing Oozie Workflows

This chapter provides information about how to set up the Oozie Engine and explains how to execute Oozie Workflows using Oracle Data Integrator. It also explains how to audit Hadoop logs.
This chapter includes the following sections:

- Executing Oozie Workflows with Oracle Data Integrator
- Setting Up and Initializing the Oozie Runtime Engine
- Creating a Logical Oozie Engine
- Executing or Deploying an Oozie Workflow
- Auditing Hadoop Logs
- Userlib jars support for running ODI Oozie workflows

## 5.1 Executing Oozie Workflows with Oracle Data Integrator

To execute oozie workflows with oracle data integrator, setup the Oozie runtime engine, execute or deploy an Oozie workflow and then audit the Hadoop Logs.

The following table summarizes the steps you need to perform to execute Oozie Workflows with Oracle Data Integrator.

**Table 5-1 Executing Oozie Workflows**

| Step | Description |
|---|---|
| Set up the Oozie runtime engine | Set up the Oozie runtime engine to configure the connection to the Hadoop data server where the Oozie engine is installed. This Oozie runtime engine is used to execute ODI Design Objects or Scenarios on the Oozie engine as Oozie workflows.<br>See Setting Up and Initializing the Oozie Runtime Engine . |
| Execute or deploy an Oozie workflow | Run the ODI Design Objects or Scenarios using the Oozie runtime engine created in the previous step to execute or deploy an Oozie workflow.<br>See Executing or Deploying an Oozie Workflow. |
| Audit Hadoop Logs | Audit the Hadoop Logs to monitor the execution of the Oozie workflows from within Oracle Data Integrator.<br>See Auditing Hadoop Logs. |

## 5.2 Setting Up and Initializing the Oozie Runtime Engine

Before you set up the Oozie runtime engine, ensure that the Hadoop data server where the Oozie engine is deployed is available in the topology. The Oozie engine must be associated with this Hadoop data server.

To set up the Oozie runtime engine:

1. In the Topology Navigator, right-click the **Agents Tree** node in the Physical Architecture navigation tree and click **New Oozie Engine**.
2. In the Definition tab, specify the values in the fields for the defining the Oozie runtime engine.

   See Oozie Runtime Engine Definition for the description of the fields.
3. In the Properties tab, specify the properties for the Oozie Runtime Engine.

   See Oozie Runtime Engine Properties for the description of the properties.
4. Click **Test** to test the connections and configurations of the actual Oozie server and the associated Hadoop data server.
5. Click **Initialize** to initialize the Oozie runtime engine.

   Initializing the Oozie runtime engine deploys the log retrieval workflows and coordinator workflows to the HDFS file system and starts the log retrieval coordinator and workflow jobs on the actual Oozie server. The log retrieval flow and coordinator for a repository and oozie engine will have the names `OdiRetrieveLog_<EngineName>_<ReposId>_F` and `OdiLogRetriever_<EngineName>_<ReposId>_C` respectively.

   It also deploys the ODI libraries and classes.
6. Click **Save**.

Executing Oozie Workflows with Oracle Data Integrator

## 5.2.1 Oozie Runtime Engine Definition

The following table describes the fields that you need to specify on the Definition tab when defining a new Oozie runtime engine. An Oozie runtime engine models an actual Oozie server in a Hadoop environment.

**Table 5-2 Oozie Runtime Engine Definition**

| Field | Values |
|---|---|
| **Name** | Name of the Oozie runtime engine that appears in Oracle Data Integrator. |
| **Host** | Name or IP address of the machine on which the Oozie runtime agent has been launched. |
| **Port** | Listening port used by the Oozie runtime engine. Default Oozie port value is 11000. |
| **Web application context** | Name of the web application context. Type `oozie` as the value of this field, as required by the Oozie service process running in an Hadoop environment. |
| **Protocol** | Protocol used for the connection. Possible values are `http` or `https`. Default is `http`. |
| **Hadoop Server** | Name of the Hadoop server where the oozie engine is installed. This Hadoop server is associated with the oozie runtime engine. |
| **Poll Frequency** | Frequency at which the Hadoop audit logs are retrieved and stored in ODI repository as session logs.<br>The poll frequency can be specified in seconds (s), minutes (m), hours (h), days (d), and years (d). For example, 5m or 4h. |

**Table 5-2 (Cont.) Oozie Runtime Engine Definition**

| Field | Values |
|---|---|
| **Lifespan** | Time period for which the Hadoop audit logs retrieval coordinator stays enabled to schedule audit logs retrieval workflows.<br>Lifespan can be specified in minutes (m), hours (h), days (d), and years (d). For example, 4h or 2d. |
| **Schedule Frequency** | Frequency at which the Hadoop audit logs retrieval workflow is scheduled as an Oozie Coordinator Job.<br>Schedule workflow can be specified in minutes (m), hours (h), days (d), and years (d). For example, 20m or 5h. |

Setting Up and Initializing the Oozie Runtime Engine

Configuring Big Data technologies using the Big Data Configurations Wizard

## 5.2.2 Oozie Runtime Engine Properties

The following table describes the properties that you can configure on the Properties tab when defining a new Oozie runtime engine.

**Table 5-3 Oozie Runtime Engine Properties**

| Field | Values |
|---|---|
| **OOZIE_WF_GEN_MAX_DETAIL** | Limits the maximum detail (session level or fine-grained task level) allowed when generating ODI Oozie workflows for an Oozie engine.<br>Set the value of this property to TASK to generate an Oozie action for every ODI task or to SESSION to generate an Oozie action for the entire session. |

Setting Up and Initializing the Oozie Runtime Engine

Configuring Big Data technologies using the Big Data Configurations Wizard

# 5.3 Creating a Logical Oozie Engine

To create a logical oozie agent:

1. In Topology Navigator, right-click the **Agents** node in the Logical Architecture navigation tree.
2. Select **New Logical Oozie Engine.**
3. Fill in the **Name**.
4. For each Context in the left column, select an existing Physical Agent in the right column. This Physical Agent is automatically associated to the Logical Oozie Engine in this context.
5. From the **File** menu, click **Save**.

Setting Up and Initializing the Oozie Runtime Engine

## 5.4 Executing or Deploying an Oozie Workflow

You can run an ODI design-time object such as a Mapping or a runtime object such as a Scenario using an Oozie Workflow. When running the ODI design object or scenario, you can choose to only deploy the Oozie workflow without executing it.

> **Note:**
>
> To enable SQOOP logging when executing an Oozie workflow, add the below property to the data server – `HADOOP_CLIENT_OPTS="-Dlog4j.debug -Dhadoop.root.logger=INFO,console -Dlog4j.configuration=file:/etc/hadoop/conf.cloudera.yarn/log4j.properties"`

To execute an ODI Oozie workflow:

1. From the Projects menu of the Designer navigator, right-click the mapping that you want to execute as an Oozie workflow and click **Run**.
2. From the **Logical Agent** drop-down list, select the Oozie runtime engine.
3. Click **OK**.

   The Information dialog appears.
4. Check if the session started and click **OK** on the Information dialog.

To deploy an ODI Oozie workflow:

1. From the Load Plans and Scenarios menu of the Designer navigator, right-click the scenario that you want to deploy as an Oozie workflow and click **Run**.
2. From the **Logical Agent** drop-down list, select the Oozie runtime engine.
3. Select **Deploy Only** to process the scenario, generate the Oozie workflow, and deploy it to HDFS.
4. Click **OK**.

   The Information dialog appears.
5. Check if the session started and click **OK** on the Information dialog.

Executing Oozie Workflows with Oracle Data Integrator

## 5.5 Auditing Hadoop Logs

When the ODI Oozie workflows are executed, log information is retrieved and captured according to the frequency properties on the Oozie runtime engine. This information relates to the state, progress, and performance of the Oozie job.

You can retrieve the log data of an active Oozie session by clicking the **Retrieve Log Data** in the Operator menu. Also, you can view information regarding the oozie session in the oozie webconsole or the MapReduce webconsole by clicking the URL available in the Definition tab of the Session Editor.

The Details tab in the Session Editor, Session Step Editor, and Session Task Editor provides a summary of the oozie and MapReduce job.

Executing Oozie Workflows with Oracle Data Integrator

## 5.6 Userlib jars support for running ODI Oozie workflows

Support of userlib jars for ODI Oozie workflows allows a user to copy jar files into a userlib HDFS directory, which is referenced by ODI Oozie workflows that are generated and submitted with the `oozie.libpath` property.

This avoids replicating the `libs/jars` in each of the workflow app's lib HDFS directory. The userlib directory is located in HDFS in the following location:

```
<ODI HDFS Root>/odi_<version>/userlib
```

Executing Oozie Workflows with Oracle Data Integrator

6

# Using Query Processing Engines to Generate Code in Different Languages

This chapter describes how to set up the query processing engines that are supported by Oracle Data Integrator to generate code in different languages.
This chapter includes the following sections:

- Query Processing Engines Supported by Oracle Data Integrator
- Setting Up Hive Data Server
- Creating a Hive Physical Schema
- Setting Up Pig Data Server
- Creating a Pig Physical Schema
- Setting Up Spark Data Server
- Creating a Spark Physical Schema
- Generating Code in Different Languages

## 6.1 Query Processing Engines Supported by Oracle Data Integrator

Hadoop provides a framework for parallel data processing in a cluster. There are different languages that provide a user front-end. Oracle Data Integrator supports the following query processing engines to generate code in different languages:

- **Hive**

  The Apache Hive warehouse software facilitates querying and managing large datasets residing in distributed storage. Hive provides a mechanism to project structure onto this data and query the data using a SQL-like language called HiveQL.

- **Pig**

  Pig is a high-level platform for creating MapReduce programs used with Hadoop. The language for this platform is called Pig Latin.

- **Spark**

  Spark is a fast and general processing engine compatible with Hadoop data. It can run in Hadoop clusters through YARN or Spark's standalone mode, and it can process data in HDFS, HBase, Cassandra, Hive, and any Hadoop Input Format.

To generate code in these languages, you need to set up Hive, Pig, and Spark data servers in Oracle Data Integrator. These data servers are to be used as the staging area in your mappings to generate HiveQL, Pig Latin, or Spark code.

Generate Code in Different Languages with Oracle Data Integrator

# 6.2 Setting Up Hive Data Server

To set up the Hive data server:

1. Click the Topology tab.
2. In the Physical Architecture tree, under Technologies, right-click Hive and then click **New Data Server**.
3. In the Definition tab, specify the details of the Hive data server.

   See Hive Data Server Definition for more information.
4. In the JDBC tab, specify the Hive data server connection details.

   See Hive Data Server Connection Details for more information.
5. Click **Test Connection** to test the connection to the Hive data server.

## 6.2.1 Hive Data Server Definition

The following table describes the fields that you need to specify on the Definition tab when creating a new Hive data server.

**Note:** Only the fields required or specific for defining a Hive data server are described.

**Table 6-1 Hive Data Server Definition**

| Field | Description |
|---|---|
| **Name** | Name of the data server that appears in Oracle Data Integrator. |
| **Data Server** | Physical name of the data server. |
| **User/Password** | Hive user with its password. |
| **Metastore URI** | Hive Metastore URIs: for example, `thrift://BDA:10000`. |
| **Hadoop Data Server** | Hadoop data server that you want to associate with the Hive data server. |
| **Additional Classpath** | Additional classpaths. |

Setting Up Hive Data Server

Configuring Big Data technologies using the Big Data Configurations Wizard

## 6.2.2 Hive Data Server Connection Details

The following table describes the fields that you need to specify on the JDBC tab when creating a new Hive data server.

**Note:** Only the fields required or specific for defining a Hive data server are described.

**Table 6-2 Hive Data Server Connection Details**

| Field | Description |
|---|---|
| **JDBC Driver** | `Apache Hive DataDirect Driver`<br>Use this JDBC driver to connect to the Hive Data Server. |

**Table 6-2 (Cont.) Hive Data Server Connection Details**

| Field | Description |
|---|---|
| **JDBC URL** | `jdbc:weblogic:hive://<host>:<port>[; property=value[;...]]`<br>For example, `jdbc:weblogic:hive://localhost:10000;DatabaseName=default;User=default;Password=default`<br>**Kerberized**: `jdbc:weblogic:hive://<host>:<port>;DatabaseName=<value>;AuthenticationMethod=kerberos;ServicePrincipalName=<value>`<br>For example, `jdbc:weblogic:hive://localhost:10000;DatabaseName=default;AuthenticationMethod=kerberos;ServicePrincipalName=hive` |

Setting Up Hive Data Server

# 6.3 Creating a Hive Physical Schema

Create a Hive physical schema using the standard procedure, as described in the Creating a Physical Schema section in *Administering Oracle Data Integrator*.

Create for this physical schema a logical schema using the standard procedure, as described in the Creating a Logical Schema section in *Administering Oracle Data Integrator* and associate it in a given context.

Setting Up Hive Data Server

# 6.4 Setting Up Pig Data Server

To set up the Pig data server:

1. Click the Topology tab.
2. In the Physical Architecture tree, under Technologies, right-click Pig and then click **New Data Server**.
3. In the Definition tab, specify the details of the Pig data server.

   See Pig Data Server Definition for more information.
4. In the Properties tab, add the Pig data server properties.

   See Pig Data Server Properties for more information.
5. Click **Test Connection** to test the connection to the Pig data server.

## 6.4.1 Pig Data Server Definition

The following table describes the fields that you need to specify on the Definition tab when creating a new Pig data server.

**Note:** Only the fields required or specific for defining a Pig data server are described.

**Table 6-3 Pig Data Server Definition**

| Field | Description |
|---|---|
| **Name** | Name of the data server that will appear in Oracle Data Integrator. |
| **Data Server** | Physical name of the data server. |
| **Process Type** | Choose one of the following:<br>• **Local Mode**<br>Select to run the job in local mode.<br>In this mode, pig scripts located in the local file system are executed. MapReduce jobs are not created.<br>• **MapReduce Mode**<br>Select to run the job in MapReduce mode.<br>In this mode, pig scripts located in the HDFS are executed. MapReduce jobs are created.<br>**Note:** If this option is selected, the Pig data server must be associated with a Hadoop data server. |
| **Hadoop Data Server** | Hadoop data sever that you want to associate with the Pig data server.<br>**Note:** This field is displayed only when the **MapReduce Mode** option is set to Process Type. |
| **Additional Classpath** | Specify additional classpaths.<br>Add the following additional classpaths:<br>Local Mode<br>• `/<dir name>/pig/pig.jar`<br>MapReduce Mode<br>• `/etc/hbase/conf`<br>• `/usr/lib/pig/lib`<br>• `/usr/lib/pig/pig-0.12.0-cdh<version>.jar`<br>Replace <version> with the Cloudera version you have. For example, `/usr/lib/pig/pig-0.12.0-cdh5.10.0.jar`.<br>• `/usr/lib/hive-hcatalog/share/hcatalog`<br>• `/usr/lib/hbase/lib`<br>• `/usr/lib/hbase`<br>For pig-hcatalog-hive, add the following classpath in addition to the ones mentioned above:<br>`/usr/lib/hive-hcatalaog/share/hcatalog` |
| **User/Password** | Pig user with its password. |

Setting Up Pig Data Server

Configuring Big Data technologies using the Big Data Configurations Wizard

## 6.4.2 Pig Data Server Properties

The following table describes the Pig data server properties that you need to add on the Properties tab when creating a new Pig data server.

**Table 6-4 Pig Data Server Properties**

| Key | Value |
|---|---|
| **hive.metastore.uris** | `thrift://bigdatalite.localdomain:9083` |
| **pig.additional.jars** | `/usr/lib/hive-hcatalog/share/hcatalog/*.jar:/usr/lib/hive/` |
| **hbase.defaults.for.version.skip** | Set to true to skip the **hbase.defaults.for.version** check. Set this boolean to true to avoid seeing the **RuntimException** issue. |
| **hbase.zookeeper.quorum** | Quorum of the HBase installation. For example, `localhost:2181`. |

Setting Up Pig Data Server

# 6.5 Creating a Pig Physical Schema

Create a Pig physical schema using the standard procedure, as described in the Creating a Physical Schema section in *Administering Oracle Data Integrator*.

Create for this physical schema a logical schema using the standard procedure, as described in the Creating a Logical Schema section in *Administering Oracle Data Integrator* and associate it in a given context.

Setting Up Pig Data Server

# 6.6 Setting Up Spark Data Server

To set up the Spark data server:

1. Click the Topology tab.
2. In the Physical Architecture tree, under Technologies, right-click Spark Python and then click **New Data Server**.
3. In the Definition tab, specify the details of the Spark data server.

   See Spark Data Server Definition for more information.
4. In the Properties tab, specify the properties for the Spark data server.

   See Spark Data Server Properties for more information.
5. Click **Test Connection** to test the connection to the Spark data server.

> **Note:**
>
> The test connection button is disabled because Spark and Pig are not testable.

## 6.6.1 Spark Data Server Definition

The following table describes the fields that you need to specify on the Definition tab when creating a new Spark Python data server.

**Note:** Only the fields required or specific for defining a Spark Python data server are described.

**Table 6-5 Spark Data Server Definition**

| Field | Description |
|---|---|
| **Name** | Name of the data server that will appear in Oracle Data Integrator. |
| **Master Cluster (Data Server**) | Physical name of the master cluster or the data server. |
| **User/Password** | Spark data server or master cluster user with its password. |
| **Hadoop DataServer** | Hadoop data server that you want to associate with the Spark data server.<br>**Note:** This field appears only when you are creating the Spark Data Server using the Big Data Configurations wizard. |
| **Additional Classpath** | The following additional classpaths are added by default:<br>• `/usr/lib/spark/*`<br>• `/usr/lib/spark/lib/*`<br>If required, you can add more additional classpaths.<br>**Note:** This field appears only when you are creating the Spark Data Server using the Big Data Configuration wizard. |

Setting Up Spark Data Server

Configuring Big Data technologies using the Big Data Configurations Wizard

## 6.6.2 Spark Data Server Properties

The following table describes the properties that you can configure on the Properties tab when defining a new Spark data server.

**Note:** Other than the properties listed in the following table, you can add Spark configuration properties on the Properties tab. The configuration properties that you add here are applied when mappings are executed. For more information about the configuration properties, refer to the Spark documentation available at the following URL:

http://spark.apache.org/docs/latest/configuration.html

**Table 6-6 Spark Data Server Properties**

| Property | Description |
|---|---|
| **archives** | Comma separated list of archives to be extracted into the working directory of each executor. |
| **deploy-mode** | Whether to launch the driver program locally (client) or on one of the worker machines inside the cluster (cluster). |
| **driver-class-path** | Classpath entries to pass to the driver. Jar files added with --jars are automatically included in the classpath. |
| **driver-cores** | Number of cores used by the driver in Yarn Cluster mode. |
| **driver-java-options** | Extra Java options to pass to the driver. |
| **driver-library-path** | Extra library path entries to pass to the driver. |
| **driver-memory** | Memory for driver, for example, 1000M, 2G. The default value is **512M**. |
| **executor-cores** | Number of cores per executor. The default value is 1 in YARN mode, or all available cores on the worker in standalone mode. |
| **executor-memory** | Memory per executor, for example, 1000M, 2G. The default value is **1G**. |

**Table 6-6 (Cont.) Spark Data Server Properties**

| Property | Description |
|---|---|
| **jars** | Comma-separated list of local jars to include on the driver and executor classpaths. |
| **num-executors** | Number of executors to launch. The default value is **2**. |
| **odi-execution-mode** | ODI execution mode, either **SYNC** or **ASYNC**. |
| **properties-file** | Path to a file from which to load extra properties. If not specified, this will look for `conf/spark-defaults.conf`. |
| **py-files** | Additional python file to execute. |
| **queue** | The YARN queue to submit to. The default value is **default**. |
| **spark-home-dir** | Home directory of Spark installation. |
| **spark-web-port** | Web port of Spark UI. The default value is **1808**. |
| **spark-work-dir** | Working directory of ODI Spark mappings that stores the generated python file. |
| **supervise** | If configured, restarts the driver on failure (Spark Standalone mode). |
| **total-executor-cores** | Total cores for all executors (Spark Standalone mode). |
| **yarn-web-port** | Web port of yarn, the default value is **8088**. |
| **principal** | Kerberized User name. |
| **keytab** | Kerberized Password. |

Setting Up Spark Data Server

Configuring Big Data technologies using the Big Data Configurations Wizard

## 6.7 Creating a Spark Physical Schema

Create a Spark physical schema using the standard procedure, as described in the Creating a Physical Schema section in *Administering Oracle Data Integrator*.

Create for this physical schema a logical schema using the standard procedure, as described in the Creating a Logical Schema section in *Administering Oracle Data Integrator* and associate it in a given context.

Setting Up Spark Data Server

## 6.8 Generating Code in Different Languages

Oracle Data Integrator can generate code for multiple languages. For Big Data, this includes HiveQL, Pig Latin, Spark RDD, and Spark DataFrames. The style of code is primarily determined by the choice of the data server used for the staging location of the mapping.

Before you generate code in these languages, ensure that the Hive, Pig, and Spark data servers are set up.

For more information see the following sections:

Setting Up Hive Data Server

Setting Up Pig Data Server

Setting Up Spark Data Server

**To generate code in different languages:**

1. Open your mapping.
2. To generate HiveQL code, run the mapping with the default staging location (Hive).
3. To generate Pig Latin or Spark code, go to the Physical diagram and do one of the following:
   a. To generate Pig Latin code, set the **Execute On Hint** option to use the Pig data server as the staging location for your mapping.
   b. To generate Spark code, set the **Execute On Hint** option to use the Spark data server as the staging location for your mapping.
4. Execute the mapping.

Query Processing Engines Supported by Oracle Data Integrator

Generate Code in Different Languages with Oracle Data Integrator

7

# Working with Spark

This chapter describes the various concepts involved in working with Spark.

This chapter includes the following sections:

- Spark Usage
- Spark Design Considerations
- Spark Streaming Support
- Switching between RDD and DataFrames in ODI
- Components that do not support DataFrame Code Generation
- Adding Customized Code in the form of a Table Function

## 7.1 Spark Usage

To use Spark engines, a Staging Execution Unit must be created in the Physical Mapping and the EU execution location must be set to Spark Schema.

### 7.1.1 Creating a Spark Mapping

To create a Spark mapping, ensure the Spark Logical and Physical Schemas are already created, and follow the procedure below:

1. Select **Mappings** > **New Mapping**.
2. Drag the `file_src` and `hdfs_tgt` Data Stores from the Models tree onto the Logical Diagram.
3. Link the mapping connectors together and choose map columns by position.

   This will map the columns.
4. Set the **Staging Location Hint** to your Spark Logical Schema.
5. Go to the Physical Diagram and select the white space on the canvas. Ensure that the **Optimization Context** is set to the correct Context for running against your cluster, and that the **Preset Staging Location** is set to Spark.
6. Click the `SPARKLS_STAGING_NODE` node and set the **Loading Knowledge Module** to LKM File to Spark.
7. Click the `FIL_AP` node in the Target Group and set the **Loading Knowledge Module** to LKM Spark to File.
8. Click the `HDF` node and ensure that the **Integration Knowledge Module** is set to <Default>.

### 7.1.2 Pre-requisites for handling Avro and Delimited files in Spark Mappings

You must install external libraries before using Spark mappings with Avro or Delimited files.

### Avro files

For using Avro files in Spark mappings, the Avro `.egg` file has to be added to the ODI installation. The `.egg` file for Avro cannot be downloaded directly, and has to be generated from the Avro package.

To add the Avro `.egg` file to the ODI installation:

1. **Generate the .egg file from the Avro Package**

   a. Download `avro-1.8.0.tar.gz` from avro 1.8.2 : Python Package Index or Apache Avro™ Releases.

   b. Unzip it, and install the Avro Python library as shown below.

   ```
   $ tar xvf avro-1.8.1.tar.gz
   $ cd avro-1.8.1
   $ sudo python setup.py install
   Installed /usr/lib/python2.6/site-packages/avro-_AVRO_VERSION_-py2.6.egg
   Processing dependencies for avro===-AVRO-VERSION-
   Finished processing dependencies for avro===-AVRO-VERSION-
   ```

   The `avro-_AVRO_VERSION_-py2.6.egg` file can now be found in the Python installed directory.

   For more information, see Apache Avro™ 1.8.0 Getting Started (Python).

2. **Copy the `.egg` file to a specific location in ODI**

   For ODI Agent, copy the `.egg` file to `$DOMAIN_HOME_PROPERTY/lib/spark`.

   For ODI Studio, copy the `.egg` file to `$HOME/.odi/oracledi/userlib/spark`.

### Delimited files

For using Delimited files in Spark mappings, external jar files must be added to the ODI installation.

To add the CSV jar files to the ODI installation:

1. **Download the CSV jar files**

   Download the following jar files from their corresponding links:

   - `spark-csv_2.10-1.5.0.jar` from spark-csv
   - `commons-csv-1.2.jar` from Commons CSV – Download Apache Commons CSV

2. **Copy the jar file to a specific location in ODI**

   For ODI Agent, copy the jar files to `$DOMAIN_HOME_PROPERTY/lib/spark`.

   For ODI Studio, copy the jar files to `$HOME/.odi/oracledi/userlib/spark`.

## 7.2 Spark Design Considerations

If you have chosen to use Spark as your Transformation Engine, you must take the following design decisions before writing your Mappings:

- Batch or Streaming
- Resilient Distributed Datasets (RDD) or DataFrames
- Infer Schema Knowledge Module Option

- Expression Syntax

## 7.2.1 Batch or Streaming

Spark supports two modes of operation — Batch and Streaming. In Streaming mode, you can ingest data from Kafka Topics, or Files/HDFS Files added to a specified location. To get the most out of Streaming, see Spark Checkpointing and Spark Windowing and Stateful Aggregation.

To set the Streaming flag, select **Physical Design**, click the blank canvas, and select the **Streaming** checkbox on the property panel. If the Streaming flag is not set, the mappings will execute in Batch mode (default).

## 7.2.2 Resilient Distributed Datasets (RDD) or DataFrames

Spark has more than one set of APIs that can be used to transform data. Resilient Distributed Datasets (RDD) and DataFrames are APIs that ODI can generate code for.

**Resilient Distributed Datasets (RDD)**

RDDs are the primary data abstraction in Apache Spark. They are fault tolerant (Resilient) collections of partitioned data (Dataset) with data residing on multiple nodes in a cluster (Distributed). The data resides in memory and is cached where necessary. RDDs are read-only, but can have transformations applied that will create other RDDs. Lazy evaluation is used, where the data is only available or transformed when triggered. RDD partitions are the unit of parallelism.

**DataFrames**

A DataFrame is a read-only distributed collection of data, that (unlike RDDs) is organized into named columns. The abstraction level is higher, making the processing of large datasets even easier, such as in allowing the use of SparkSQL queries. DataFrames are built on top of the Spark SQL engine, allowing for much better performance and space optimization.

> **Note:**
>
> If Streaming is used, RDD is the only option available.

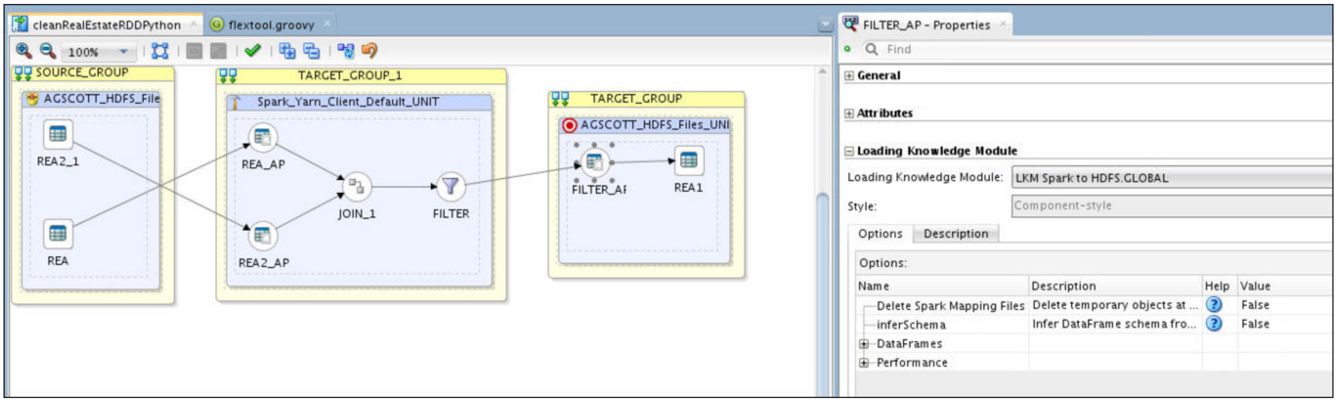
## 7.2.3 Infer Schema Knowledge Module Option

Spark can infer or deduce the Schema of a dataset by looking at the data. Although this can be advantageous, there are some circumstances where datatypes may not be mapped as expected. If this happens, there is an inferSchema option on applicable Spark KMs that can be set to False, turning off this functionality. If you see runtime errors related to datatype mismatches, you may need to adjust the value of the Infer Schema option. This option can be set on reading or writing LKMs.

> **Note:**
>
> Spark uses this option only while creating DataFrames. If inferSchema is set to False, ODI will generate a schema definition based on mapping data store metadata and this structure will be used to create DataFrame API.

The Infer Schema option can be seen in the figure below.

**Figure 7-1    Physical Mapping with InferSchema KM Option**



Loading Knowledge Module: LKM Spark to HDFS.GLOBAL

Style: Component-style

Options:

| Name | Description | Help | Value |
|---|---|---|---|
| Delete Spark Mapping Files | Delete temporary objects at ... | ? | False |
| inferSchema | Infer DataFrame schema fro... | ? | False |
| ⊞ DataFrames | | | |
| ⊞ Performance | | | |

## 7.2.4 Expression Syntax

When you need to write expressions, for example, in a Join or Filter condition, or an Attribute expression, you have options that can be used for the Expression Syntax. If you have decided to have ODI generate RDD code, then your expressions must be written in Python. If, however, you have decided to generate DataFrames, then you can choose to write your expressions in SQL or Python. You can specify your chosen syntax by setting `SQL_EXPRESSIONS` to True/False.

The combinations of possible code generation style are:

- RDD with Python expressions
- DataFrames with Python expressions
- DataFrames with SQL expressions

Since Python expressions are defined differently in RDD and DataFrames, the Python syntax for these two styles of code generation can be different. Therefore, not all Python expressions will work for both RDD and DataFrame code generation styles.

### RDD with Python expressions

For information on the syntax and functions that can be used in Python expressions, see The Python Standard Library.

### DataFrames with Python expressions

For information on the list of Python functions available to Column objects, see pyspark.sql.Column.

### DataFrames with SQL expressions

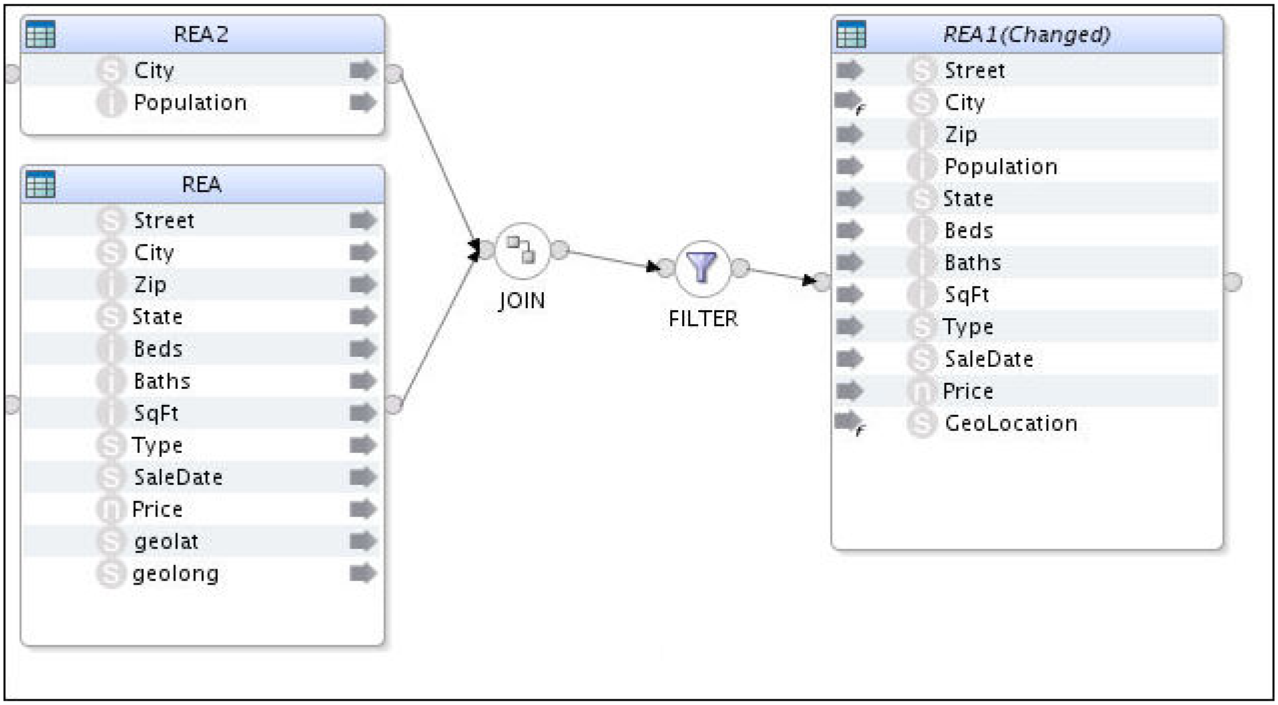
The generic SQL functions and operators can be viewed in the Expression editor on selecting generic SQL language.

Consider an example that shows multiple expressions being used in mappings.

### Mapping Description

In this example, a source (`REA`) containing Real Estate Transactions is combined with a second source (`REA2`) containing City and Population data. A filter is then applied to select only large transactions. This creates a target file (REA1) which contains the combined and filtered information as shown in the figure below.

**Figure 7-2 Mapping with Multiple Expressions**



### Mapping Definitions

The mapping is defined as follows:

- **JOIN**: Joins the Real Estate Transaction Table (REA) with the City Population table (REA2) using City as the Key. The City names in REA1 are in uppercase, whereas in REA2 they are in lowercase.
- **FILTER**: Selects rows that have a price greater or equal to $500,000, and ignores transactions where Type is Multi-Family.
- **City**: City names should be in lowercase except for the first letter of each word.
- **GeoLocation**: This should be in the form "<longitude>, <latitude>". The quotes should be included in the string.
- **Population**: Rounds off to the nearest thousand.

### Mapping Expressions for each Codegen Style

The following table describes the mapping expressions for each codegen style.

**Table 7-1 Mapping Expressions for each Codegen Style**

| Mapping Expression for the Codegen Style | RDD with Python Expressions | DataFrames with Python Expressions | DataFrames with SQL Expressions |
|---|---|---|---|
| **Join Condition** | REA.City == (REA2.City).upper() | REA.City == upper(REA2.City) | REA.City = UPPER(REA2.City) |
| **Filter Syntax** | REA.Type<>'Multi-Family'<br>and REA.Price >=500000 | REA.Type<>'Multi-Family' and REA.Price >=500000 | REA.Type<>'Multi-Family' and REA.Price >=500000 |
| **Target Column Syntax** | # City - note: this only capitalizes the first word!<br>(REA.City).capitalize()<br># GeoLocation<br>REA.geolat + ", " + REA.geolong<br># Population<br>int(round(REA2.Population,-3)) | # City<br>initcap(lower(REA.City))<br># GeoLocation<br>concat(REA.geolat ,lit(", "),REA.geolong)<br># Population<br>round(REA2.Population,-3) | -- City<br>INITCAP(LOWER(REA.City))<br>-- GeoLocation<br>CONCAT(REA.geolat,', ', REA.geolong)<br># Population<br>ROUND(REA2.Population,-3) |

### Importing Libraries

As you'll see from this example, not all the same built-in functions are available across these differing styles. In this case, the `initcap` built-in function is not available in RDD. The `capwords()` function does what is required, but it requires import statements to be added to the script. The Spark EKM has a multi line option called customPythonImports that lets you specify the Import Statements for the script, thereby allowing extra functions to be available in the expressions.

To contain the list of imports, the customPythonImports EKM option will be written as

```
from string import *
from time import localtime
```

The Target Column expression would then be written as

```
#City
capwords(REA.City)
```

## 7.3 Spark Streaming Support

This section provides information about streaming modes of operation on data sets. It also provides information on Checkpointing.

> **Note:**
>
> Spark 2.0 Streaming using Kafka is no longer supported for Data Platforms such as Cloudera CDH 6.0, Hortonworks 3.1 and later.

This section includes the following sub-sections:

- Spark Checkpointing
- Spark Windowing and Stateful Aggregation
- Spark Repartitioning and Caching
- Configuring Streaming Support
- Executing Mapping in Streaming Mode

## 7.3.1 Spark Checkpointing

A streaming application must operate 24/7 and hence should be resilient to failures. Spark Streaming needs to checkpoint information to a fault tolerant storage system so that it can recover from failures.

Checkpointing is enabled for applications recovering from failures of the driver running the application. Checkpointing only ensures that the Spark application will restart from where it left if a checkpoint is found.

For additional information on checkpointing, refer to Spark Streaming Programming Guide.

## 7.3.2 Spark Windowing and Stateful Aggregation

Spark's Windowing feature allows aggregation (and other transformations) to be applied not just to the current RDD, but also include data from several previous RDDs (window duration).

The Spark KMs support batch and, also streaming transformations. While the Python code for non-streaming operates on RDD or DataFrame objects, the streaming code works on DStream objects. Aggregation in batch mode is simple: there is a single set of input records (RDD), which are aggregated to form the output data, which is then written into some target. In streaming mode the continuously incoming data is discretized into a flow of RDDs. By default each RDD is aggregated independently.

Spark windowing works well for calculating things like running sum or running averages. But it comes with two restrictions:

- Older RDDs must be retained
- Data falling into the window is recalculated for every new RDD.

This is the reason why windowing is not suitable for aggregation across an entire data stream. This can only be achieved by stateful aggregation.

Windowing enabled KMs have the following optional KM Options:

- **Window Duration**: Duration of window defined in number of batch intervals.
- **Sliding Interval**: Interval at which the window operation is performed defined in number of batch intervals.

Windowing is supported by:

- XKM Spark Aggregation
- XKM Spark Join
- XKM Spark Set
- XKM Spark Distinct

For additional information, refer to Spark Streaming Programming Guide.

### Stateful Aggregation

When data must be aggregated across all data of a stream, stateful aggregation is required. In stateful aggregation Spark builds called state stream containing the aggregated values for all keys. For every incoming RDD this state is updated, for example aggregated sums are updated based on new incoming data.

By default a state stream will output all stored values for every incoming RDD. This is useful in case the stream output is a file and the file is expected to always hold the entire set of aggregate values.

Stateful processing is supported by:

- XKM Spark Aggregate
- XKM Spark Lookup

## 7.3.3 Spark Repartitioning and Caching

### Caching

In ODI, the Spark caching mechanism is leveraged by providing two additional Spark base KM options.

- **Cache data**: If this option set to true a storage invocation is added to the generated pyspark code of the component.
- **Cache storage level**: This option is hidden if cache data is set to false.

### Repartitioning

If the source is a HDFS file, the number of partitions is initially determined by the data block of the source HDFS system. The platform resource is not fully used if the platform that runs the Spark application has more available slots for running tasks than the number of partitions loaded. In such cases, the RDD.repartition() api can be used to change the number of partitions.

Repartitioning can be done in any step of the whole process, even immediately after data is loaded from source or after processing the filter component. ODI has Spark base KM options which let you decide whether and where to do repartitioning.

- **Repartition**: If this option is set to true, repartition is applied after the transformation of component.
- **Level of Parallelism**: Number of partitions and the default is 0. When the default value is set, spark.default.parallelism will be used to invoke the repartition() function.
- **Sort Partitions**: If this option is set to true, partitions are sorted by key and the key is defined by a Lambda function.
- **Partitions Sort Order**: Ascending or descending. Default is ascending.
- **Partition Keys**: User defined partition keys represented as a comma separated column list.
- **Partition Function**: User defined partition Lambda function. Default value is a pyspark defined hash function `portable_hash`, which simply computes a hash base on the entire RDD row.

## 7.3.4 Configuring Streaming Support

Configuring Streaming Support is performed in two parts:

1. **Topology**
   1. Click the Topology tab.
   2. In the Physical Architecture tree, under Technologies, right-click Spark Python and then click **New Data Server**.
   3. In the Definition tab, specify the details of the Spark data server.

      See Spark Data Server Definition for more information.
   4. In the Properties tab, specify the properties for the Spark data server.

      See Spark Data Server Properties for more information.
   5. Click **Test Connection** to test the connection to the Spark data server.
2. **Mapping Design**
   - To edit your mapping, select **Physical Design**, click the blank canvas, and select the **Streaming** checkbox on the property panel.

     ODI generates code that allows the mapping to run in Streaming mode, instead of Batch mode.

### 7.3.4.1 Spark Streaming DataServer Properties

Provides the Spark Technology-specific streaming properties that are default for the Spark Execution Unit properties.

**Table 7-2 Spark Streaming DataServer Properties**

| Key | Value |
|---|---|
| spark.checkpointingBaseDir | This property defines the base directory for checkpointing. Every mapping under this base directory will create a sub-directory.<br>Example: hdfs://cluster-ns1/user/oracle/spark/checkpoints |
| spark.checkpointingInterval | Displays the time in seconds |
| spark.restartFromCheckpoint | • If set to true, the Spark Streaming application will restart from an existing checkpoint.<br>• If set to false, the Spark Streaming application will ignore any existing checkpoints.<br>• If there is no checkpoint, it will start normally. |
| spark.batchDuration | Displays the duration in seconds of a streaming interval. |
| spark.rememberDuration | Displays the time in seconds and sets the Spark Streaming context to remember RDDs for this duration. |
| spark.checkpointing | Enables Spark checkpointing. |
| spark.streaming.timeout | Displays the timeout in seconds before stopping a Streaming application.<br>Default is 60. |

**Table 7-2 (Cont.) Spark Streaming DataServer Properties**

| Key | Value |
|---|---|
| odi-execution-mode | • **SYNCHRONOUS**: Spark application is submitted and monitored through `OdiOSCommand.`<br>• **ASYNCHRONOUS**: Spark application is submitted asynchronously through `OdiOSCommand` and then monitored through Spark REST APIs. |
| spark.ui.enabled | Enables the Spark Live REST API.<br>**Note:**<br>Set to true for asynchronous execution. |
| spark.eventLog.enabled | Enables Spark event logs. This allows the logs to be accessible by the Spark History Server.<br>**Note:**<br>Set to true for asynchronous execution. |
| principal | Kerberized User name. |
| keytab | The location of the keytab file that contains pairs of kerberos principal and encrypted keys.<br>Example: /tmp/oracle.keytab |
| odi.spark.enableUnsupportedSparkModes | This check is introduced, as only yarn-client and yarn-cluster are supported. |

## 7.3.4.2 Extra Spark Streaming Data Properties

Provides the extra spark streaming properties that are specific to Spark technology that are added to the asynchronous Spark execution unit.

**Table 7-3 Extra Spark Streaming Properties**

| Key | Value |
|---|---|
| spark-webui-startup-polling-retries | Maximum number of retries while waiting for the Spark WebUI to come-up. |
| spark-webui-startup-polling-interval | Displays the time in seconds between retries. |
| spark-webui-startup-polling-persist-after-retries | |

**Table 7-3 (Cont.) Extra Spark Streaming Properties**

| Key | Value |
|---|---|
| spark-webui-rest-timeout | Timeout in second used for REST calls on Spark WebUI. |
| spark-webui-polling-interval | Time in seconds between two polls on the Spark WebUI. |
| spark-webui-polling-persist-after-retries | |
| spark-history-server-rest-timeout | Timeout in seconds used for REST calls on Spark History Server. |
| spark-history-server-polling-retries | Maximum number of retries while waiting for the Spark History Server to make the Spark Event Logs available. |
| spark-history-server-polling-interval | Time in seconds between retries. |
| spark-history-server-polling-persist-after-retries | |
| spark-submit-shutdown-polling-retries | Maximum number of retries while waiting for the spark-submit OS process to complete. |
| spark-submit-shutdown-polling-interval | Time in seconds between retries. |
| spark-submit-shutdown-polling-persist-after-retries | |

## 7.3.5 Executing Mapping in Streaming Mode

This topic provides the steps to enable executing the mapping in the streaming mode. Streaming needs checkpointing information for a fault-tolerant storage system to recover from failures.

1. To enable streaming support, see Configuring Streaming Support.
2. In physical design of the mapping, select staging execution unit, and enable checkpointing options on the EKM. Enable checkpointing by setting the value of `spark.checkpointing` to True and set the Checkpointing directory in the `spark.checkpointingBaseDir` property.

   Every mapping will have its unique checkpointing directory.
3. Execute the mapping and set the context for physical design.

> **Note:**
>
> In the **User Interface Designer** by default, the `Last executed physical design in the mapping execution dialog` is pre-selected.

## 7.4 Switching between RDD and DataFrames in ODI

You can switch between generating DataFrame code and RDD code by setting the EKM option `spark.useDataFrames` to either True or False on the Spark Execution Unit.

## 7.5 Components that do not support DataFrame Code Generation

Some components do not support DataFrame code generation. If even a single mapping component does not support DataFrames, a validation error is shown (asking you to set the Spark Execution Unit property spark.useDataFrames to false) and you will need to switch back to RDD.

The following components do not support DataFrame code generation:

- Pivot
- Unpivot
- Input Signature
- Output Signature

## 7.6 Adding Customized Code in the form of a Table Function

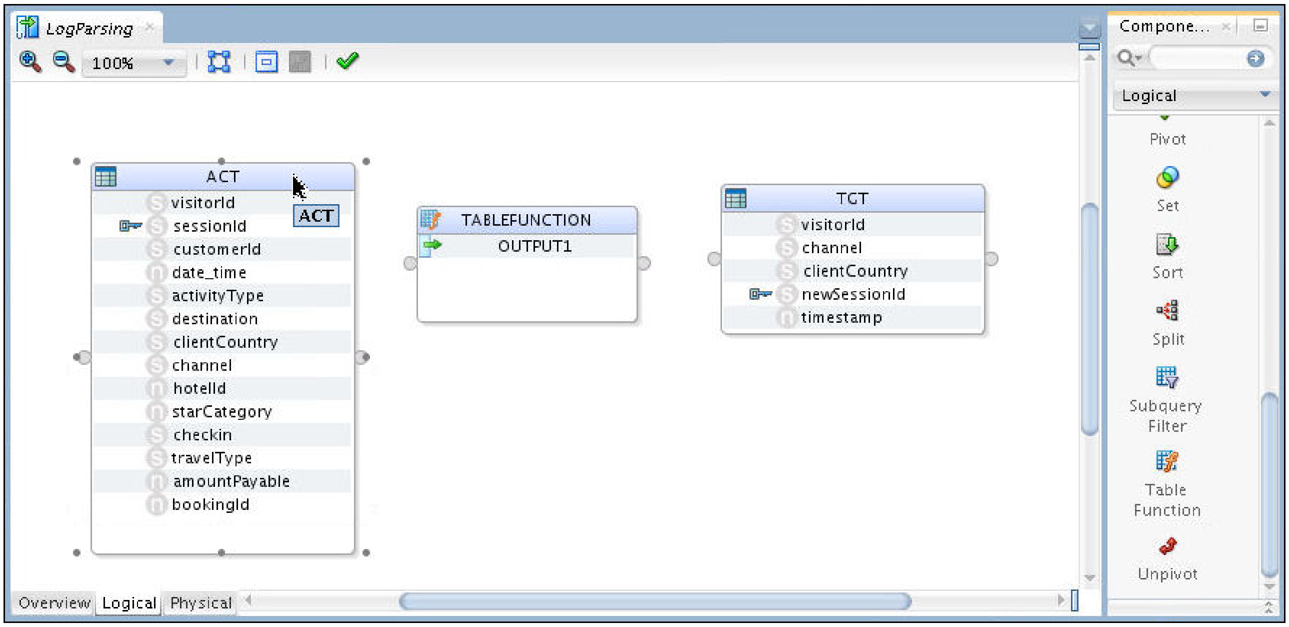
The TableFunction component allows you to add your own code segment into the mapping in the form of a reference to an external script, or some inline code.

Consider an example where the TABLEFUNCTION component is utilized to parse and transform a source log file. The mapping will produce a target file with data as-is from the source file, modified data, and new data such as timestamps.

To build the mapping containing a table function and add input and output attributes to it, follow the below procedure:

1. Create a mapping by adding the source and target data stores along with a table function component named 'TABLEFUNCTION'.

**Figure 7-3 Mapping with Source, Target, and Table Function**

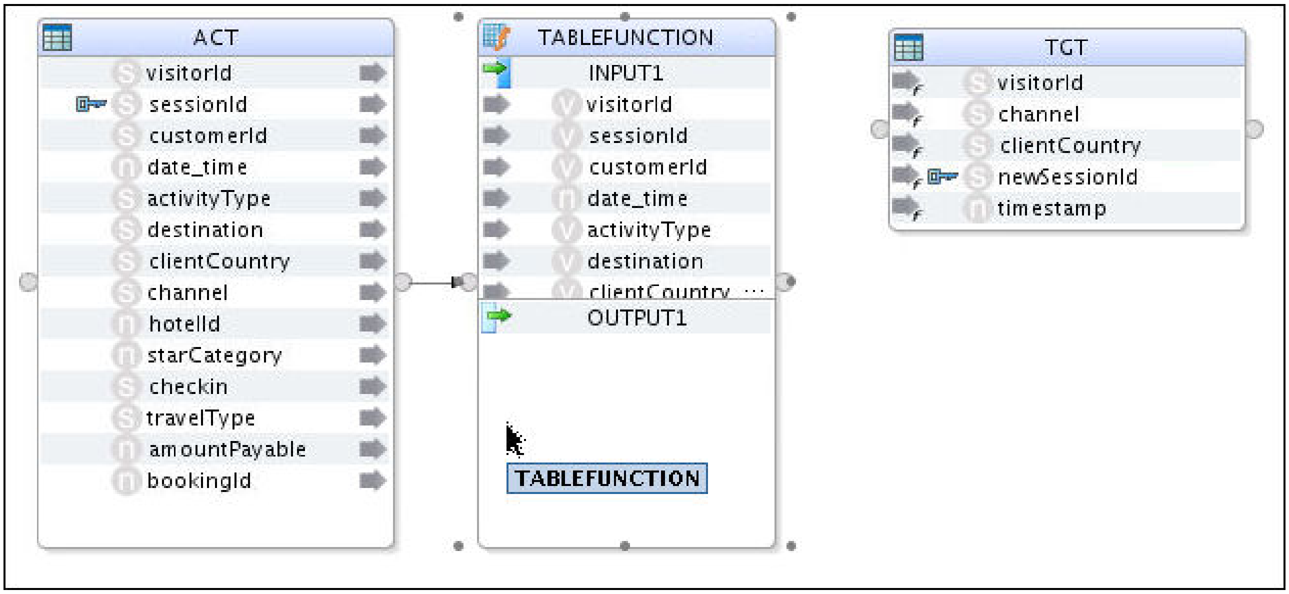


2. Connect the source data store's output connector to the input connector of the TABLEFUNCTION component.

   Input attributes will now be added directly to TABLEFUNCTION.

> **Note:**
>
> - An input group 'INPUT1' is created automatically containing all the attributes from the source data store as shown in the figure below.
> - For each additional source data store, a new input group will be added.

**Figure 7-4 Input Group added to TABLEFUNCTION**

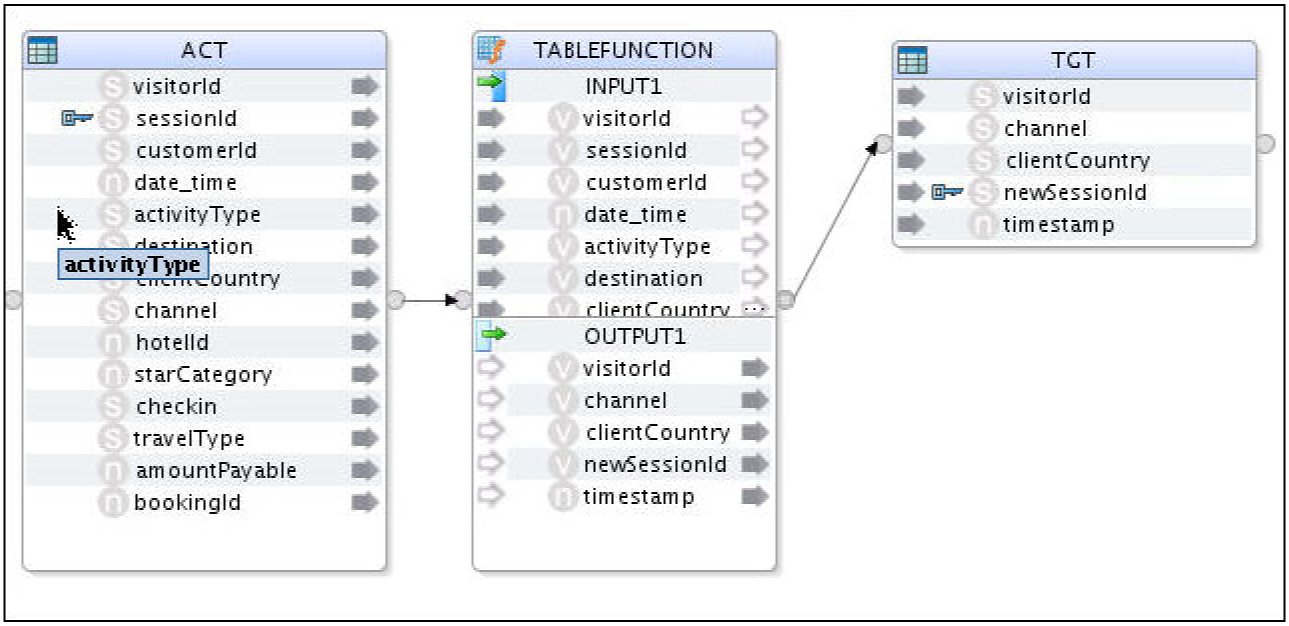


3. Connect the target data store’s input connector to the output connector of the TABLEFUNCTION component.

   Output attributes will now be added directly to TABLEFUNCTION.

> **Note:**
>
> - An output group 'OUTPUT1' is created automatically containing all the attributes from the target data store as shown in the figure below.
> - The output attributes in 'OUTPUT1' can be renamed or deleted.
> - The expression for each output attribute will be set grammatically by the script embedded in the TABLEFUNCTION component and doesn’t need to be set individually.

**Figure 7-5 Mapping with Source, Target, and Table Function connected**



Configure the mapping by following the procedure below:

1. Go to the Logical tab and select **Spark-Local_Default** as the Staging Location Hint.
2. Go to the Physical tab. Under Extract Options, specify the script to use for the TABLEFUNCTION component by entering `/tmp/xkmtf.py` as the value for the SPARK_SCRIPT_FILE KM option. The `xmktf.py` script contains the following content:

```
import sys
import datetime

#get the upstream object using the input connector point name
upstream=sys.argv[0]['INPUT1']

#A value must be calculated for every TF output attribute
TABLEFUNCTION = upstream.map(lambda input:Row(**{"visitorId":input.visitorId,
"channel":input.channel, "clientCountry":input.clientCountry,
"newSessionId":'Prefix'+input.sessionId, "timeStamp":now.strftime("%Y-%m-%d
%H:%M")}))
```

Here, the input group 'INPUT1' of the TABLEFUNCTION component is passed through sys.argv to the Spark-Python script `xkmtf.py`.

Alternatively, you can directly specify the script to use for the TABLEFUNCTION component by entering the following content as the value for the SPARK_SCRIPT KM option:

```
import datetime

now = datetime.datetime.now()

#A value must be calculated for every TF output attribute
TABLEFUNCTION = ACT.map(lambda input:Row(**{"visitorId":input.visitorId,
"channel":input.channel, "clientCountry":input.clientCountry,
"newSessionId":'Prefix'+input.sessionId, "timeStamp":now.strftime("%Y-%m-%d
%H:%M")}))
```

There are two types of Spark Scripts for TableFunction:

- External TableFunction Script
- Inline TableFunction Script

### External TableFunction Script

This can be dynamically executed from within ODI mapping code. If necessary, use sys.argv to send in RDDs/DataFrames for processing with the external script.

For example, consider a TableFunction component inserted with the following properties:

- Name – TABLEFUNCTION
- Input connector - INPUT1
- Input fields - IN_ATTR_1 and IN_ATTR_2
- Output attributes - OUT_ATTR_1, OUT_ATTR_2, and OUT_ATTR_3

As seen in the external script below, the upstream RDD/DataStream object is obtained using the input connector point name. The resulting RDD/DStream is then calculated, where a value is calculated for every TableFunction output attribute name.

```
import sys
import datetime
upstream=sys.argv[0]['INPUT1']
now = datetime.datetime.now()
TABLEFUNCTION = upstream.map(lambda input:Row(**{"OUT_ATTR_1":input.sessionId,
"OUT_ATTR_2":input.customerId, "OUT_ATTR_3":now.strftime("%Y-%m-%d %H:%M")}))
```

To dynamically execute this external script, ODI generates the following mapping code. The result of the external script execution is stored as TABLEFUNCTION.

```
sys.argv=[dict(INPUT1=ACT)]
execfile('/tmp/xkmtf_300.py')
TABLEFUNCTION = TABLEFUNCTION.toDF(...)
```

### Inline TableFunction Script

In inline mode, the actual TableFunction script is stored as an XKM option. You don’t need to use sys.argv to send in any source objects for processing the script.

As seen in the internal script below, the result of the external script execution is directly referenced.

```
ACT=ACT.filter("ACT_customerId = '5001'")
TABLEFUNCTION = ACT.toDF(...)
```

# 8 Working with Unstructured Data

This chapter provides an overview of the Jagged component and the Flatten component. These components help you to process unstructured data.
This chapter includes the following section:

- Working with Unstructured Data

## 8.1 Working with Unstructured Data

Oracle Data Integrator provides a Jagged component that can process unstructured data. Source data from sources such as social media or e-commerce businesses is represented in a key-value free format. Using the jagged component, this data can be transformed into structured entities that can be loaded into database tables.

For more information using the Jagged component and KMs associated with it, see the following sections:

- Creating Jagged Components in *Developing Integration Projects with Oracle Data Integrator*.
- XKM Jagged.

# 9 Working with Complex Datatypes and HDFS File Formats

This chapter provides an overview of extended data format support and complex type support.

This chapter includes the following sections:

- HDFS File Formats
- Working with Complex Datatypes in Mappings
- Hive Complex Datatypes
- Cassandra Complex Datatypes
- Loading Data from HDFS File to Hive
- Loading Data from HDFS File to Spark

## 9.1 HDFS File Formats

### Supported Formats

ODI can read and write HDFS file data in a variety of formats. The HDFS file formats supported are Json, Avro, Delimited, and Parquet. The format is specified on the Storage Tab of the HDFS data store. When you reverse-engineer Avro, JSON, or Parquet files, you are required to supply a Schema in the Storage Tab. The reverse-engineer process will only use the Schema, and not access the HDFS files themselves. Delimited HDFS files cannot be reverse-engineered, the Attributes (in the Attributes tab of the HDFS data store) will have to be added manually and the parameters, such as field separator should be defined on the Storage Tab.

If you are loading Avro files into Hive, then you will need to copy the Avro Schema file (.avsc) into the same HDFS location as the Avro HDFS files (using the same file name that you specified for the Schema in the Storage Panel).

### Complex Types

JSON, Avro, and Parquet formats can contain complex data types, such as array or Object. During the Reverse-Engineering phase, the Datatype field for these Attributes is set to "Complex" and the definition of the complex type is stored in the Data Format field for the Attribute. The Syntax of this definition is the same as Avro uses for its Schema definitions. This information is used by ODI in the Mapping Editor when the flatten component is added to the Mapping.

**Table 9-1 HDFS File Formats**

| File Format | Reverse-Engineer | Complex Type Support | Load into Hive | Load into Spark | Write from Spark |
|---|---|---|---|---|---|
| Avro | Yes (Schema required) | Yes | Yes (Schema required) | Yes (Batch mode only) | Yes |

**Table 9-1 (Cont.) HDFS File Formats**

| File Format | Reverse-Engineer | Complex Type Support | Load into Hive | Load into Spark | Write from Spark |
|---|---|---|---|---|---|
| Delimited | No | No | Yes | Yes | Yes |
| JSON | Yes (Schema required) | Yes | Yes | Yes | Yes |
| Parquet | Yes (Schema required) | Yes | Yes | Yes (Batch mode only) | Yes (Batch and Streaming) |

**Table 9-2 Complex Types**

| Avro | Json | Hive | Parquet |
|---|---|---|---|
| Record | Object | Struct | Record |
| enum | NA | NA | enum |
| array | array | array | array |
| map | NA | map | map |
| union | NA | union | union |
| fixed | NA | NA | fixed |

## 9.2 Working with Complex Datatypes in Mappings

Provides information on working with complex, nested, and user defined metadata that drives the Flatten component.

Oracle Data Integrator provides a Flatten component that can process input data with a Complex structure and produce a flattened representation of the same data using standard data types. The input data may be in various formats, such as a Hive table or a JSON HDFS file.

When you add a Flatten component into a Mapping, you choose the attribute to Flatten from the component upstream.

The Flatten components for Spark and Hive have some advanced usability features that do not exist in the other implementations. Namely, once you've chosen the attribute to flatten from the upstream node, the flattened attributes will be created automatically. The reason this is possible is that the Reverse-Engineering process for Hive and HDFS capture the Complex Type definition in the Attribute's "Data Format" property. You can view this property in the Attribute tab of the Hive or HDFS Data Store. The Flatten component's Collection and Structure properties are also set automatically based on the Attribute definition. That leaves just the "Include Nulls" property to be set manually, based on whether null complex data should be processed. Some technologies, particularly Spark, can drop records containing null complex attributes.

**Table 9-3 Properties for Flatten Component**

| Flatten Property | Description | Automatically detected for Hive and HDFS (if reverse-engineering was used) |
|---|---|---|
| Include Nulls | Indicates whether null complex data should be processed. | No |
| Collection | Indicates whether the Complex Type attribute is a collection such as an array. | Yes |
| Structure | Indicates whether the Complex Type is an object, record, or structure, and not just a collection of scalar types. | Yes |

Each Flatten component can flatten only one Complex Type attribute. You can chain Flatten components together to flatten more than one attribute, or where nested datatypes are concerned, to access the next level of nesting.

For more information using the Flatten component and the KMs associated with it, see the following sections:

- Creating Flatten Components in *Developing Integration Projects with Oracle Data Integrator*.
- XKM Oracle Flatten.

The example in Using Flatten for Complex Types in Hive Mappings shows an example of chaining Flatten components. The `Flatten_Director` Complex Type Attribute is set to the upstream `MOVIE_DIRECTOR` attribute from the `MOV` node. At that point, `NAME` and `AGE` are created in `Flatten_Director` automatically. `Flatten_Ratings` follows `Flatten_Director` and uses `Flatten_Director.RATINGS` as the Complex Type Attribute, after which `rating` and `info` attributes are automatically added.

# 9.3 Hive Complex Datatypes

Hive has the following complex data types:

- Arrays
- Maps
- Structs
- Union

## 9.3.1 Using Flatten for Complex Types in Hive Mappings

The Flatten component is used to handle Complex Types in Hive mappings.

Consider the JSON snippet below which is a source with two Complex Types:

- A MOVIE_DIRECTOR field which is a structure consisting of Name and Age.
- A RATINGS field which is an array of ratings, with each rating comprising a rating and an info field.

```
{"MOVIE_ID":11,"MOVIE_NAME":"The Lobster","MOVIE_DIRECTOR":{"NAME":"Yorgos
Lanthimos","AGE":43},"RATINGS":[{"rating":7,"info":"x"},{"rating":5,"info":"x"}]}
{"MOVIE_ID":12,"MOVIE_NAME":"Green Room","MOVIE_DIRECTOR":{"NAME":"Jeremy
Saulnier","AGE":40},"RATINGS":[{"rating":4,"info":"x"},{"rating":3,"info":"x"}]}
{"MOVIE_ID":13,"MOVIE_NAME":"Louder Than Bombs","MOVIE_DIRECTOR":{"NAME":"Joachin
Trier","AGE":42},"RATINGS":[{"rating":1,"info":"x"},{"rating":2,"info":"x"}]}
...
```

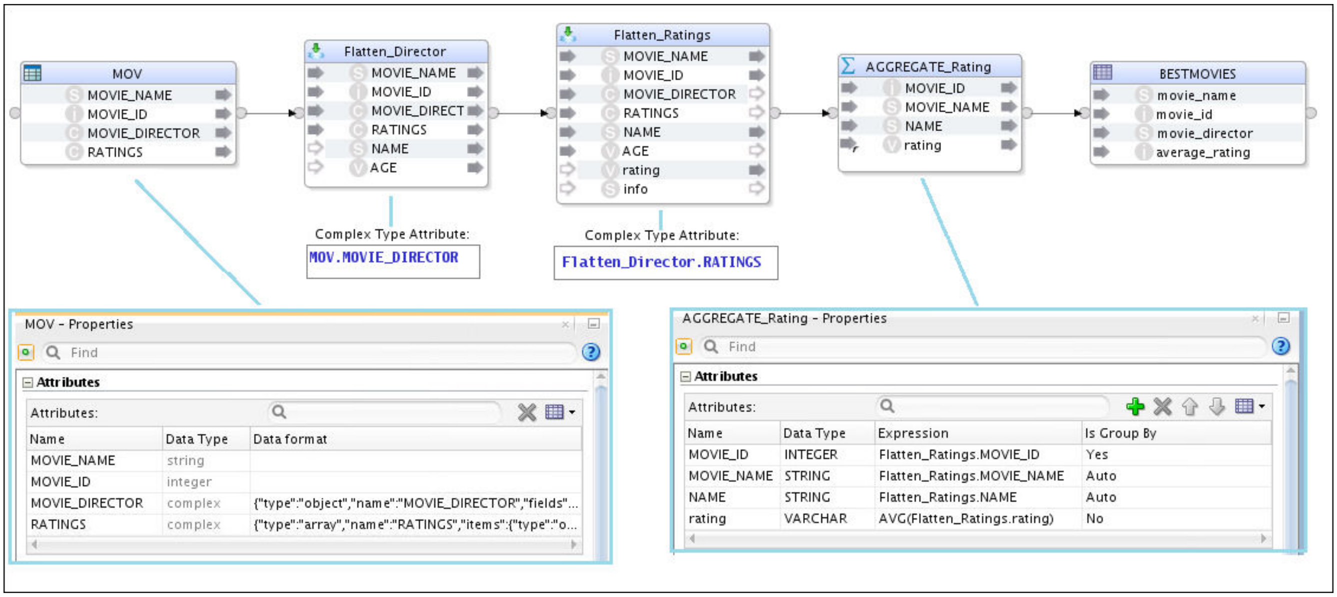
The Hive table that is to be populated requires the NAME to be extracted from the MOVIE_DIRECTOR complex structure along with the average of the rating values from the RATINGS array.

To accomplish this, a mapping is required which flattens the

- MOVIE_DIRECTOR field so that the NAME can be extracted.
- RATINGS array so that the average of the individual ratings for each row can be calculated.

The mapping is as shown in the figure below.

**Figure 9-1 Mapping to flatten Complex Types**



Complex Type Attribute: MOV.MOVIE_DIRECTOR

Complex Type Attribute: Flatten_Director.RATINGS

MOV - Properties

Attributes:

| Name | Data Type | Data format |
|---|---|---|
| MOVIE_NAME | string | |
| MOVIE_ID | integer | |
| MOVIE_DIRECTOR | complex | {"type":"object","name":"MOVIE_DIRECTOR","fields"... |
| RATINGS | complex | {"type":"array","name":"RATINGS","items":{"type":"o... |

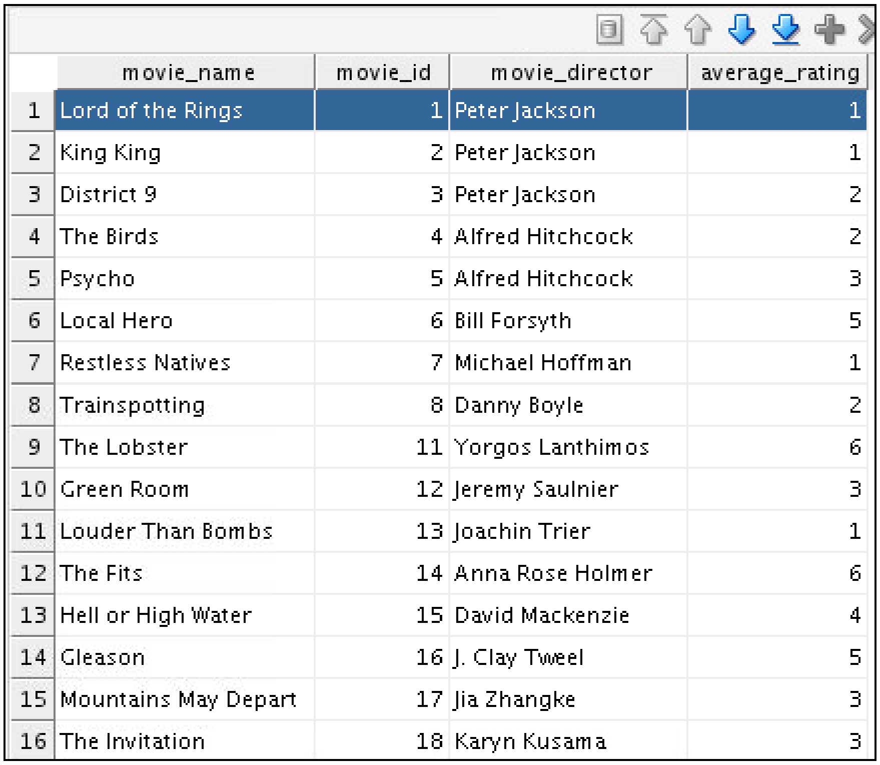
AGGREGATE_Rating - Properties

Attributes:

| Name | Data Type | Expression | Is Group By |
|---|---|---|---|
| MOVIE_ID | INTEGER | Flatten_Ratings.MOVIE_ID | Yes |
| MOVIE_NAME | STRING | Flatten_Ratings.MOVIE_NAME | Auto |
| NAME | STRING | Flatten_Ratings.NAME | Auto |
| rating | VARCHAR | AVG(Flatten_Ratings.rating) | No |

The populated Hive table appears as shown in the figure below.

**Figure 9-2    Populated Hive Table**



| | movie_name | movie_id | movie_director | average_rating |
|---|---|---|---|---|
| 1 | Lord of the Rings | 1 | Peter Jackson | 1 |
| 2 | King King | 2 | Peter Jackson | 1 |
| 3 | District 9 | 3 | Peter Jackson | 2 |
| 4 | The Birds | 4 | Alfred Hitchcock | 2 |
| 5 | Psycho | 5 | Alfred Hitchcock | 3 |
| 6 | Local Hero | 6 | Bill Forsyth | 5 |
| 7 | Restless Natives | 7 | Michael Hoffman | 1 |
| 8 | Trainspotting | 8 | Danny Boyle | 2 |
| 9 | The Lobster | 11 | Yorgos Lanthimos | 6 |
| 10 | Green Room | 12 | Jeremy Saulnier | 3 |
| 11 | Louder Than Bombs | 13 | Joachin Trier | 1 |
| 12 | The Fits | 14 | Anna Rose Holmer | 6 |
| 13 | Hell or High Water | 15 | David Mackenzie | 4 |
| 14 | Gleason | 16 | J. Clay Tweel | 5 |
| 15 | Mountains May Depart | 17 | Jia Zhangke | 3 |
| 16 | The Invitation | 18 | Karyn Kusama | 3 |

# 9.4 Cassandra Complex Datatypes

Cassandra has the following complex data types:

- Map
- Set
- List
- Tuple
- User-Defined Type

### Map

A map is a set of key-value pairs, where the keys are unique. The map is sorted by its keys.

### Set

A set is a collection of unique values. The set is sorted based on the values.

### List

A list is a collection of non-unique values which are ordered by their position in the list.

### Tuple

A Tuple comprises fixed-length sets of typed positional fields. It can accommodate 32768 fields, and can be used as an alternative to a User-Defined Type.

**User-Defined Type**

User-Defined Type (UDT) is a complex data type that can be created, updated, and deleted.

Cassandra can be used with LKM Spark to Cassandra and generic SQL KMs.

The Apache Cassandra DataDirect JDBC Driver handles Complex Types differently compared to the Hive JDBC Driver. Due to this, mappings that use Cassandra Complex Types are written slightly differently compared to Hive mappings. To demonstrate this, consider a table defined in Cassandra with the use of UDTs and Lists.

- You can access UDTs through the Apache Cassandra DataDirect JDBC Driver because the JDBC Driver flattens the UDTs and projects the scalar types as regular columns of the table. This negates the need to use the Flatten Component on the mapping. You will see that the extra columns have been flattened automatically in the Cassandra Data Stores, and can be used directly in mappings.
- You can access collections of values (Lists in Cassandra; Arrays in Hive) through the Apache Cassandra DataDirect JDBC Driver. The Apache Cassandra DataDirect JDBC Driver normalizes the structure and projects the collection type through a child table. When you reverse-engineer the original table, additional data stores will be created for the collections. The mapping then needs to join these two tables.
- You cannot access Nested Types in ODI.

## 9.4.1 How ODI deals with Cassandra Lists and User Defined Types

This is an example that shows how ODI deals with Cassandra Lists and User Defined Types.

Consider a schema, `movieRating2` containing a UDT and a List:

- A `movie_director` attribute which is a UDT consisting of Name and Age.
- A `ratings` attribute which is a list of integers.

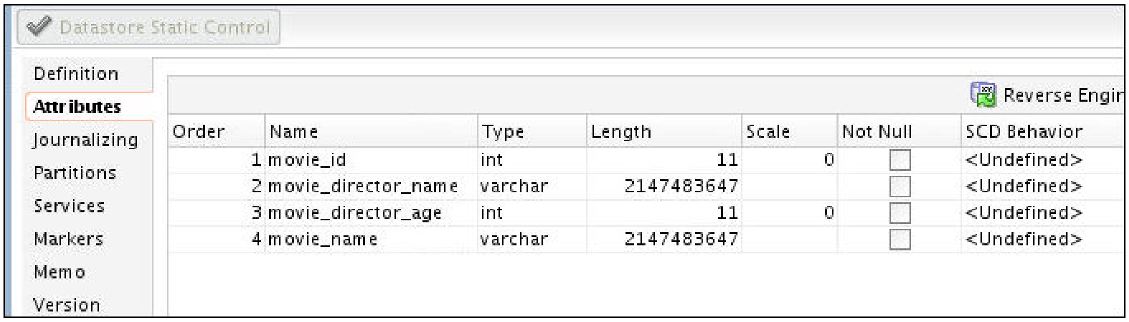
```
create type director_object (name text, age int);

create table movieRating2 (movie_name text, movie_id int PRIMARY KEY, movie_director
frozen<director_object>, ratings list<int>);

INSERT INTO movierating2 (movie_id, movie_name, movie_director, ratings)
VALUES (1,'Lord of the Rings',('Peter Jackson',32),[1,2]);
INSERT INTO movierating2 (movie_id, movie_name, movie_director, ratings)
VALUES (2,'King Kong',('Peter Jackson',32),[1,2]);
INSERT INTO movierating2 (movie_id, movie_name, movie_director, ratings)
VALUES (3,'District 9',('Peter Jackson',32),[1,3]);
INSERT INTO movierating2 (movie_id, movie_name, movie_director, ratings)
VALUES (4,'The Birds',('Alfred Hitchcock',140),[1,4]);
INSERT INTO movierating2 (movie_id, movie_name, movie_director, ratings)
VALUES (5,'Psycho',('Alfred Hitchcock',140),[1,2,8]);
INSERT INTO movierating2 (movie_id, movie_name, movie_director, ratings)
VALUES (6,'Local Hero',('Bill Forsyth',56),[1,9]);
INSERT INTO movierating2 (movie_id, movie_name, movie_director, ratings)
VALUES (7,'Restless Natives',('Michael Hoffman',45),[1]);
INSERT INTO movierating2 (movie_id, movie_name, movie_director, ratings)
VALUES (8,'Trainspotting',('Danny Boyle',12),[1,4]);
```

On reverse-engineering the `movierating2` table in ODI, it appears as shown in the figure below.

**Figure 9-3 Reverse-engineered `movierating2` table**

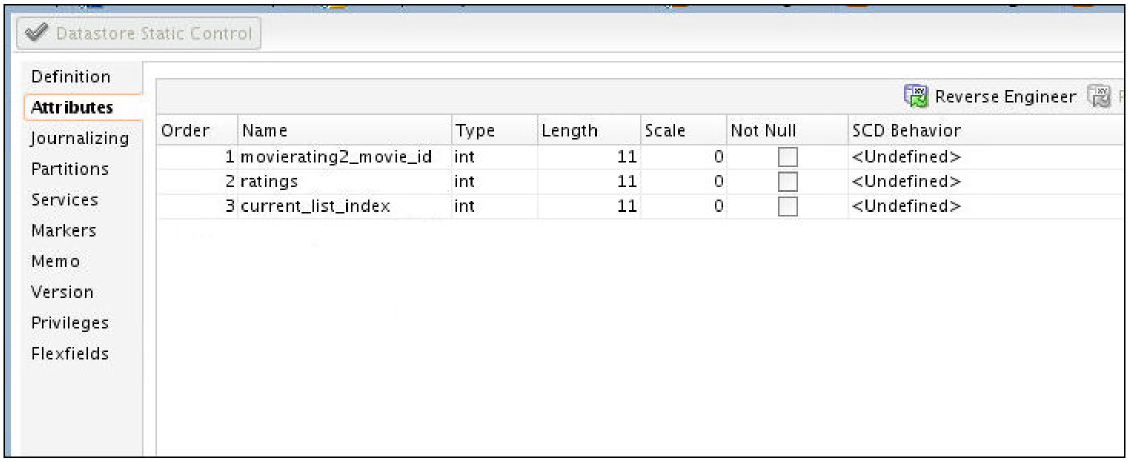


| Order | Name | Type | Length | Scale | Not Null | SCD Behavior |
|---|---|---|---|---|---|---|
| 1 | movie_id | int | 11 | 0 | □ | <Undefined> |
| 2 | movie_director_name | varchar | 2147483647 | | □ | <Undefined> |
| 3 | movie_director_age | int | 11 | 0 | □ | <Undefined> |
| 4 | movie_name | varchar | 2147483647 | | □ | <Undefined> |

This table does not contain the `ratings` attribute. However, the JDBC driver exposes a virtual table called `movierating2_ratings`.

On reverse-engineering this virtual table in ODI, it appears as shown in the figure below.

**Figure 9-4 Reverse-engineered `movierating2_ratings` table**



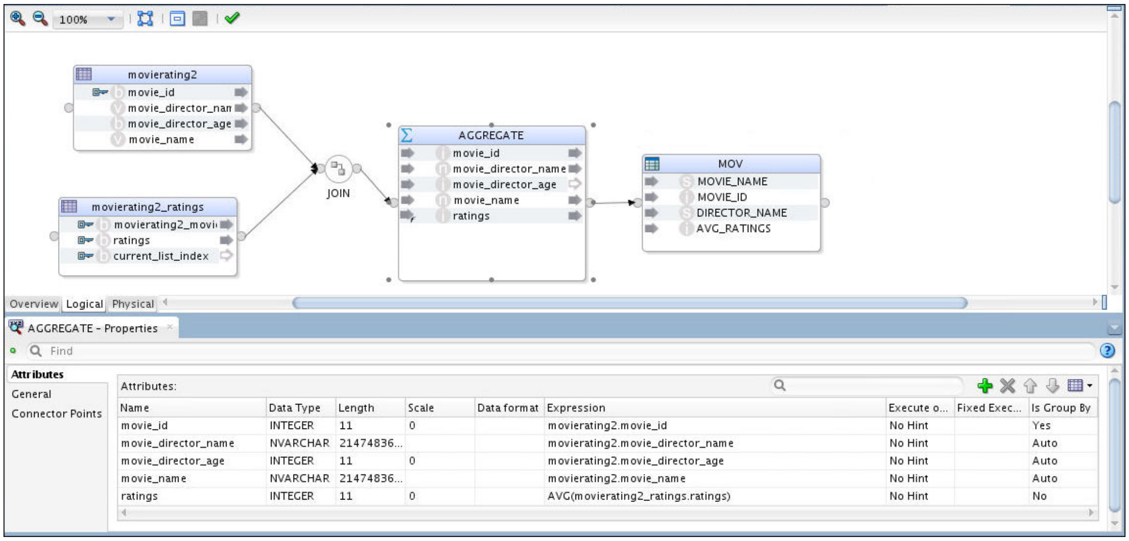
| Order | Name | Type | Length | Scale | Not Null | SCD Behavior |
|---|---|---|---|---|---|---|
| 1 | movierating2_movie_id | int | 11 | 0 | □ | <Undefined> |
| 2 | ratings | int | 11 | 0 | □ | <Undefined> |
| 3 | current_list_index | int | 11 | 0 | □ | <Undefined> |

In this example, the target HDFS file requires that the name is extracted from the `movie_director` UDT along with the average of the values from the `ratings` list.

To accomplish this, a mapping is required which joins the `movierating2` and `movierating2_ratings` tables, and averages the `ratings` for each movie.

The mapping is as shown in the figure below:

**Figure 9-5 Mapping to join `movierating2` and `movierating2_ratings` tables**



| Name | Data Type | Length | Scale | Data format | Expression | Execute o... | Fixed Exec... | Is Group By |
|---|---|---|---|---|---|---|---|---|
| movie_id | INTEGER | 11 | 0 | | movierating2.movie_id | No Hint | | Yes |
| movie_director_name | NVARCHAR | 21474836... | | | movierating2.movie_director_name | No Hint | | Auto |
| movie_director_age | INTEGER | 11 | 0 | | movierating2.movie_director_age | No Hint | | Auto |
| movie_name | NVARCHAR | 21474836... | | | movierating2.movie_name | No Hint | | Auto |
| ratings | INTEGER | 11 | 0 | | AVG(movierating2_ratings.ratings) | No Hint | | No |

The key point here is that for Cassandra Complex Types, the Flatten component is not required to access the complex fields in the mapping. You'll notice that the similar Hive mapping in Using Flatten for Complex Types in Hive Mappings is designed differently.

After running the mapping, the target HDFS file looks like this:

```
hdfs dfs -cat AvgRating.json/part-r-00000-4984bb6c-dacb-4ce1-a474-dc5641385e9f
{"MOVIE_NAME":"District 9","MOVIE_ID":3,"DIRECTOR_NAME":"Peter Jackson","AVG_RATINGS":2}
{"MOVIE_NAME":"Lord of the Rings","MOVIE_ID":1,"DIRECTOR_NAME":"Peter
Jackson","AVG_RATINGS":1}
{"MOVIE_NAME":"The Birds","MOVIE_ID":4,"DIRECTOR_NAME":"Alfred
Hitchcock","AVG_RATINGS":2}
{"MOVIE_NAME":"Restless Natives","MOVIE_ID":7,"DIRECTOR_NAME":"Michael
Hoffman","AVG_RATINGS":1}

hdfs dfs -cat AvgRating.json/part-r-00001-4984bb6c-dacb-4ce1-a474-dc5641385e9f
{"MOVIE_NAME":"Psycho","MOVIE_ID":5,"DIRECTOR_NAME":"Alfred Hitchcock","AVG_RATINGS":3}
{"MOVIE_NAME":"Trainspotting","MOVIE_ID":8,"DIRECTOR_NAME":"Danny Boyle","AVG_RATINGS":2}
{"MOVIE_NAME":"King Kong","MOVIE_ID":2,"DIRECTOR_NAME":"Peter Jackson","AVG_RATINGS":1}
{"MOVIE_NAME":"Local Hero","MOVIE_ID":6,"DIRECTOR_NAME":"Bill Forsyth","AVG_RATINGS":5}
```

# 9.5 Loading Data from HDFS File to Hive

Provides the steps to load data from HDFS file to Hive load data.

1. Create a HDFS Data Model.
2. Create a HDFS Data Store.

   See HDFS Data Server Definition for additional information.
3. In the Storage panel, set the **Storage Format**.

   A Schema is required for all except for delimited.

> **Note:**
>
> - If the Row format is set to Delimited, set the **Fields Terminated By**, **Collection Items Terminated By**, and **Map Keys Terminated By**.
> - If the HDFS file is Avro, then the Avro schema must exist in the same HDFS directory as the HDFS files.

4. Create a mapping with HDFS file as source and Hive file as target.
5. Use the LKM HDFS File to Hive Load Data and IKM Hive specified in the physical diagram of the mapping.

> **Note:**
>
> Refer to Reverse-Engineering Hive Tables for information on Reverse-Engineering.

## 9.6 Loading Data from HDFS File to Spark

Provides the steps to load data from HDFS file to Spark.

1. Create a Data Model for complex file.
2. Create a HIVE table Data Store.
3. In the Storage panel, set the **Storage Format**.
4. Create a mapping with HDFS file as source and target.
5. Use the LKM HDFS to Spark or LKM Spark to HDFS specified in the physical diagram of the mapping.

> **Note:**
>
> For AVRO format, you can specify the schema file location. Refer to Reverse-Engineering Hive Tables for information on Reverse-Engineering. There are two ways of loading Avro file to Spark either with AVSC file or without AVSC file.

A

# Hive Knowledge Modules

This appendix provides information about the Hive knowledge modules.

This appendix includes the following sections:

- LKM SQL to Hive SQOOP
- LKM SQL to File SQOOP Direct
- LKM SQL to HBase SQOOP Direct
- LKM File to SQL SQOOP
- LKM Hive to SQL SQOOP
- LKM HBase to SQL SQOOP
- IKM Hive Append
- LKM File to Hive LOAD DATA
- LKM File to Hive LOAD DATA Direct
- LKM HBase to Hive HBASE-SERDE
- LKM Hive to HBase Incremental Update HBASE-SERDE Direct
- LKM Hive to File Direct
- XKM Hive Sort
- LKM File to Oracle OLH-OSCH
- LKM File to Oracle OLH-OSCH Direct
- LKM Hive to Oracle OLH-OSCH
- LKM Hive to Oracle OLH-OSCH Direct
- RKM Hive
- RKM HBase
- IKM File to Hive (Deprecated)
- LKM HBase to Hive (HBase-SerDe) [Deprecated]
- IKM Hive to HBase Incremental Update (HBase-SerDe) [Deprecated]
- IKM SQL to Hive-HBase-File (SQOOP) [Deprecated]
- IKM Hive Control Append (Deprecated)
- CKM Hive
- IKM Hive Transform (Deprecated)
- IKM File-Hive to Oracle (OLH-OSCH) [Deprecated]
- IKM File-Hive to SQL (SQOOP) [Deprecated]

# A.1 LKM SQL to Hive SQOOP

This KM integrates data from a JDBC data source into Hive.

1. Create a Hive staging table.
2. Create a SQOOP configuration file, which contains the upstream query.
3. Execute SQOOP to extract the source data and import into Hive
4. Drop the Hive staging table.

This is a direct load LKM and will ignore any of the target IKM.

The following table descriptions the options for LKM SQL to Hive SQOOP.

**Table A-1 LKM SQL to Hive SQOOP**

| Option | Description |
|---|---|
| DELETE_TEMPORARY_OBJECTS | Delete temporary objects at end of mapping.<br>Set this option to NO, to retain temporary objects (tables, files and scripts) after integration. Useful for debugging. Default: true. |
| SQOOP_PARALLELISM | Number of SQOOP parallel mappers<br>Specifies the degree of parallelism. More precisely the number of mappers.<br>Number of mapper processes used for extraction.<br>When SQOOP_PARALLELISM > 1, SPLIT_BY must be defined. |
| SPLIT_BY | Target column name for splitting the source data.<br>Specifies the unqualified target column name to be used for splitting the source data into n chunks for parallel extraction, where n is SQOOP_PARALLELISM.<br>To achieve equally sized data chunks the split column should contain homogeneously distributed values.<br>For calculating the data chunk boundaries a query similar to SELECT MIN(EMPNO), MAX(EMPNO) from EMPLOYEE EMP is used. To avoid an extra full table scan the split column should be backed by an index. |
| BOUNDARY_QUERY | Query to retrieve min/max value for calculating data chunks using SPLIT_BY column.<br>For splitting the source data into chunks for parallel extraction the minimum and maximum value of the split column is retrieved (KM option SPLIT-BY). In certain situations this may not be the best boundaries or not the most performant way to retrieve the boundaries. In such cases this KM option can be set to a SQL query returning one row with two columns, lowest value and highest value to be used for split-column. This range will be divided into SQOOP_PARALLELISM chunks for parallel extraction.<br>Example for hard-coded ranges for an Oracle source:<br>SELECT 1000, 2000 FROM DUAL<br>For preserving context independence regular table names should be inserted through odiRef.getObjectName calls.<br>For example:<br>SELECT MIN(EMPNO), MAX(EMPNO) FROM <%=odiRef.getObjectName(EMP")%>" |

**Table A-1 (Cont.) LKM SQL to Hive SQOOP**

| Option | Description |
|---|---|
| TEMP_DIR | Local directory for temporary files.<br>Directory used for storing temporary files like squoop script, stdout and stderr redirects.<br>Leave blank to use system's default temp dir (<?=System.getProperty(java.io.tmp")?>)". |
| MAPRED_OUTPUT_BASE_DIR | MapReduce Output Directory.<br>This option specifies an hdfs directory, where SQOOP will create subdirectories for temporary files. A subdirectory called like the work table will be created here to hold the temporary data. |
| USE_GENERIC_JDBC_CONNECTOR | Use SQOOP's generic JDBC connector?<br>For certain technologies SQOOP provides specific connectors. These connectors take care of SQL-dialects and optimize performance. When there is a connector for the respective target technology, this connector should be used. If not, the generic JDBC connector may provide a solution. |
| EXTRA_HADOOP_CONF_PROPERTIES | Optional generic Hadoop properties.<br>Extra optional properties for SQOOP file: section Hadoop properties. |
| EXTRA_SQOOP_CONF_PROPERTIES | Optional SQOOP properties.<br>Extra optional properties for SQOOP file: section SQOOP properties. |
| EXTRA_SQOOP_CONNECTOR_CONF_PROPERTIES | Optional SQOOP connector properties.<br>Extra optional properties for SQOOP file: section SQOOP connector properties. |

## A.2 LKM SQL to File SQOOP Direct

This KM extracts data from a JDBC data source into an HDFS file

It executes the following steps:

1. Create a SQOOP configuration file, which contains the upstream query.
2. Execute SQOOP to extract the source data and store it as an HDFS file

This is a direct load LKM and must be used without any IKM.

> **Note:**
>
> The entire target directory will be removed before extraction.

The following table descriptions the options for LKM SQL to File SQOOP Direct.

**Table A-2 LKM SQL to File SQOOP Direct**

| Option | Description |
|---|---|
| DELETE_TEMPORARY_OBJECTS | Delete temporary objects at end of mapping.<br>Set this option to NO, to retain temporary objects (tables, files and scripts) after integration. Useful for debugging. Default: true. |
| SQOOP_PARALLELISM | Number of SQOOP parallel mappers<br>Specifies the degree of parallelism. More precisely the number of mappers.<br>Number of mapper processes used for extraction.<br>When SQOOP_PARALLELISM > 1, SPLIT_BY must be defined. |
| SPLIT_BY | Target column name for splitting the source data.<br>Specifies the unqualified target column name to be used for splitting the source data into n chunks for parallel extraction, where n is SQOOP_PARALLELISM.<br>To achieve equally sized data chunks the split column should contain homogeneously distributed values.<br>For calculating the data chunk boundaries a query similar to SELECT MIN(EMPNO), MAX(EMPNO) from EMPLOYEE EMP is used. To avoid an extra full table scan the split column should be backed by an index. |
| BOUNDARY_QUERY | Query to retrieve min/max value for calculating data chunks using SPLIT_BY column.<br>For splitting the source data into chunks for parallel extraction the minimum and maximum value of the split column is retrieved (KM option SPLIT-BY). In certain situations this may not be the best boundaries or not the most performant way to retrieve the boundaries. In such cases this KM option can be set to a SQL query returning one row with two columns, lowest value and highest value to be used for split-column. This range will be divided into SQOOP_PARALLELISM chunks for parallel extraction.<br>Example for hard-coded ranges for an Oracle source:<br>SELECT 1000, 2000 FROM DUAL<br>For preserving context independence regular table names should be inserted through odiRef.getObjectName calls.<br>For example:<br>SELECT MIN(EMPNO), MAX(EMPNO) FROM <%=odiRef.getObjectName(EMP")%>" |
| TEMP_DIR | Local directory for temporary files.<br>Directory used for storing temporary files like squoop script, stdout and stderr redirects.<br>Leave blank to use system's default temp dir (<?=System.getProperty(java.io.tmp")?>)". |
| MAPRED_OUTPUT_BASE_DIR | MapReduce Output Directory.<br>This option specifies an hdfs directory, where SQOOP will create subdirectories for temporary files. A subdirectory called like the work table will be created here to hold the temporary data. |
| USE_GENERIC_JDBC_CONNECTOR | Use SQOOP's generic JDBC connector?<br>For certain technologies SQOOP provides specific connectors. These connectors take care of SQL-dialects and optimize performance. When there is a connector for the respective target technology, this connector should be used. If not, the generic JDBC connector may provide a solution. |

**Table A-2 (Cont.) LKM SQL to File SQOOP Direct**

| Option | Description |
|---|---|
| EXTRA_HADOOP_CONF_PROPERTIES | Optional generic Hadoop properties.<br>Extra optional properties for SQOOP file: section Hadoop properties. |
| EXTRA_SQOOP_CONF_PROPERTIES | Optional SQOOP properties.<br>Extra optional properties for SQOOP file: section SQOOP properties. |
| EXTRA_SQOOP_CONNECTOR_CONF_PROPERTIES | Optional SQOOP connector properties.<br>Extra optional properties for SQOOP file: section SQOOP connector properties. |

# A.3 LKM SQL to HBase SQOOP Direct

This KM extacts data from a JDBC data source and imports the data into HBase.

It executes the following steps:

1. Create a SQOOP configuration file, which contains the upstream query.
2. Execute SQOOP to extract the source data and import into HBase.

This is a direct load LKM and must be used without any IKM.

The following table descriptions the options for LKM SQL to HBase SQOOP Direct.

**Table A-3 LKM SQL to HBase SQOOP Direct**

| Option | Description |
|---|---|
| CREATE_TARG_TABLE | Create target table?<br>Check this option, to create the target table. |
| TRUNCATE | Replace existing target data?<br>Set this option to true, to replace any existing target table content with the new data. |
| DELETE_TEMPORARY_OBJECTS | Delete temporary objects at end of mapping.<br>Set this option to NO, to retain temporary objects (tables, files and scripts) after integration. Useful for debugging. Default: true. |
| SQOOP_PARALLELISM | Number of SQOOP parallel mappers<br>Specifies the degree of parallelism. More precisely the number of mappers.<br>Number of mapper processes used for extraction.<br>When SQOOP_PARALLELISM > 1, SPLIT_BY must be defined. |
| SPLIT_BY | Target column name for splitting the source data.<br>Specifies the unqualified target column name to be used for splitting the source data into n chunks for parallel extraction, where n is SQOOP_PARALLELISM.<br>To achieve equally sized data chunks the split column should contain homogeneously distributed values.<br>For calculating the data chunk boundaries a query similar to SELECT MIN(EMPNO), MAX(EMPNO) from EMPLOYEE EMP is used. To avoid an extra full table scan the split column should be backed by an index. |

**Table A-3 (Cont.) LKM SQL to HBase SQOOP Direct**

| Option | Description |
|---|---|
| BOUNDARY_QUERY | Query to retrieve min/max value for calculating data chunks using SPLIT_BY column.<br>For splitting the source data into chunks for parallel extraction the minimum and maximum value of the split column is retrieved (KM option SPLIT-BY). In certain situations this may not be the best boundaries or not the most performant way to retrieve the boundaries. In such cases this KM option can be set to a SQL query returning one row with two columns, lowest value and highest value to be used for split-column. This range will be divided into SQOOP_PARALLELISM chunks for parallel extraction.<br>Example for hard-coded ranges for an Oracle source:<br>SELECT 1000, 2000 FROM DUAL<br>For preserving context independence regular table names should be inserted through odiRef.getObjectName calls.<br>For example:<br>SELECT MIN(EMPNO), MAX(EMPNO) FROM <%=odiRef.getObjectName(EMP")%>" |
| TEMP_DIR | Local directory for temporary files.<br>Directory used for storing temporary files like squoop script, stdout and stderr redirects.<br>Leave blank to use system's default temp dir (<?=System.getProperty(java.io.tmp")?>)". |
| MAPRED_OUTPUT_BASE_DIR | MapReduce Output Directory.<br>This option specifies an hdfs directory, where SQOOP will create subdirectories for temporary files. A subdirectory called like the work table will be created here to hold the temporary data. |
| USE_GENERIC_JDBC_CONNECTOR | Use SQOOP's generic JDBC connector?<br>For certain technologies SQOOP provides specific connectors. These connectors take care of SQL-dialects and optimize performance. When there is a connector for the respective target technology, this connector should be used. If not, the generic JDBC connector may provide a solution. |
| EXTRA_HADOOP_CONF_PROPERTIES | Optional generic Hadoop properties.<br>Extra optional properties for SQOOP file: section Hadoop properties. |
| EXTRA_SQOOP_CONF_PROPERTIES | Optional SQOOP properties.<br>Extra optional properties for SQOOP file: section SQOOP properties. |
| EXTRA_SQOOP_CONNECTOR_CONF_PROPERTIES | Optional SQOOP connector properties.<br>Extra optional properties for SQOOP file: section SQOOP connector properties. |

## A.4 LKM File to SQL SQOOP

This KM integrates data from HDFS files into a JDBC target.

It executes the following steps:

1. Create a SQOOP configuration file
2. Load data using SQOOP into a work table on RDBMS

3. Drop the work table.

The following table descriptions the options for LKM File to SQL SQOOP.

**Table A-4 LKM File to SQL SQOOP**

| Option | Description |
|---|---|
| SQOOP_PARALLELISM | Number of SQOOP parallel mappers.<br>Specifies the degree of parallelism. More precisely the number of mappers.<br>Number of mapper processes used for extraction.<br>When SQOOP_PARALLELISM > 1, SPLIT_BY must be defined. |
| WORK_TABLE_OPTIONS | Work table options.<br>Use this option to override standard technology specific work table options. When left blank, these options values are used.<br>Oracle: NOLOGGING<br>DB2 UDB: NOT LOGGED INITIALLY<br>Teradata: no fallback, no before journal, no after journal |
| TERADATA_WORK_TABLE_TYPE | Teradata work table type.<br>Use SET or MULTISET table for work table. |
| TERADATA_OUTPUT_METHOD | Teradata Load Method.<br>Specifies the way the Teradata Connector will load the data. Valid values are:<br>• batch.insert: multiple JDBC connections using batched prepared statements (simplest to start with)<br>• multiple.fastload: multiple FastLoad connections<br>• internal.fastload: single coordinated FastLoad connections (most performant)<br>Please see Cloudera's Teradata Connectors User Guide for more details. |
| TEMP_DIR | Local directory for temporary files.<br>Directory used for storing temporary files like squoop script, stdout and stderr redirects.<br>Leave blank to use system's default temp dir (<?=System.getProperty(java.io.tmp")?>)". |
| MAPRED_OUTPUT_BASE_DIR | MapReduce Output Directory.<br>This option specifies an hdfs directory, where SQOOP will create subdirectories for temporary files. A subdirectory called like the work table will be created here to hold the temporary data. |
| USE_GENERIC_JDBC_CONNECTOR | Use SQOOP's generic JDBC connector?<br>For certain technologies SQOOP provides specific connectors. These connectors take care of SQL-dialects and optimize performance. When there is a connector for the respective target technology, this connector should be used. If not, the generic JDBC connector may provide a solution. |
| EXTRA_HADOOP_CONF_PROPERTIES | Optional generic Hadoop properties.<br>Extra optional properties for SQOOP file: section Hadoop properties. |
| EXTRA_SQOOP_CONF_PROPERTIES | Optional SQOOP properties.<br>Extra optional properties for SQOOP file: section SQOOP properties. |

**Table A-4 (Cont.) LKM File to SQL SQOOP**

| Option | Description |
|---|---|
| EXTRA_SQOOP_CONNECTOR_CONF_PROPERTIES | Optional SQOOP connector properties.<br>Extra optional properties for SQOOP file: section SQOOP connector properties. |

# A.5 LKM Hive to SQL SQOOP

This KM integrates data from Hive into a JDBC target.

It executes the following steps:

1. Unload data into HDFS
2. Create a SQOOP configuration file
3. Load data using SQOOP into a work table on RDBMS
4. Drop the work table

The following table descriptions the options for LKM Hive to SQL SQOOP.

**Table A-5 LKM Hive to SQL SQOOP**

| Option | Description |
|---|---|
| DELETE_TEMPORARY_OBJECTS | Delete temporary objects at end of mapping.<br>Set this option to NO, to retain temporary objects (tables, files and scripts) after integration. Useful for debugging. |
| SQOOP_PARALLELISM | Number of SQOOP parallel mappers.<br>Specifies the degree of parallelism. More precisely the number of mappers.<br>Number of mapper processes used for extraction.<br>When SQOOP_PARALLELISM > 1, SPLIT_BY must be defined. |
| WORK_TABLE_OPTIONS | Work table options.<br>Use this option to override standard technology specific work table options. When left blank, these options values are used.<br>Oracle: NOLOGGING<br>DB2 UDB: NOT LOGGED INITIALLY<br>Teradata: no fallback, no before journal, no after journal |
| TERADATA_WORK_TABLE_TYPE | Teradata work table type.<br>Use SET or MULTISET table for work table. |
| TERADATA_OUTPUT_METHOD | Teradata Load Method.<br>Specifies the way the Teradata Connector will load the data. Valid values are:<br>• batch.insert: multiple JDBC connections using batched prepared statements (simplest to start with)<br>• multiple.fastload: multiple FastLoad connections<br>• internal.fastload: single coordinated FastLoad connections (most performant)<br>Please see Cloudera's Teradata Connectors User Guide for more details. |

**Table A-5 (Cont.) LKM Hive to SQL SQOOP**

| Option | Description |
|---|---|
| TEMP_DIR | Local directory for temporary files.<br>Directory used for storing temporary files like squoop script, stdout and stderr redirects.<br>Leave blank to use system's default temp dir (<?=System.getProperty(java.io.tmp")?>)". |
| MAPRED_OUTPUT_BASE_DIR | MapReduce Output Directory.<br>This option specifies an hdfs directory, where SQOOP will create subdirectories for temporary files. A subdirectory called like the work table will be created here to hold the temporary data. |
| USE_GENERIC_JDBC_CONNECTOR | Use SQOOP's generic JDBC connector?<br>For certain technologies SQOOP provides specific connectors. These connectors take care of SQL-dialects and optimize performance. When there is a connector for the respective target technology, this connector should be used. If not, the generic JDBC connector may provide a solution. |
| EXTRA_HADOOP_CONF_PROPERTIES | Optional generic Hadoop properties.<br>Extra optional properties for SQOOP file: section Hadoop properties. |
| EXTRA_SQOOP_CONF_PROPERTIES | Optional SQOOP properties.<br>Extra optional properties for SQOOP file: section SQOOP properties. |
| EXTRA_SQOOP_CONNECTOR_CONF_PROPERTIES | Optional SQOOP connector properties.<br>Extra optional properties for SQOOP file: section SQOOP connector properties. |

# A.6 LKM HBase to SQL SQOOP

This KM integrates data from HBase into a JDBC target.

It executes the following steps:

1. Create a SQOOP configuration file
2. Create a Hive table definition for the HBase table
3. Unload data from Hive (HBase) using SQOOP into a work table on RDBMS
4. Drop the work table.

The following table descriptions the options for LKM HBase to SQL SQOOP.

**Table A-6 LKM HBase to SQL SQOOP**

| Option | Description |
|---|---|
| DELETE_TEMPORARY_OBJECTS | Delete temporary objects at end of mapping.<br>Set this option to NO, to retain temporary objects (tables, files and scripts) after integration. Useful for debugging. Default: true. |
| HIVE_STAGING_LSCHEMA | Logical schema name for Hive-HBase-SerDe table.<br>The unloading from HBase data is done through Hive. This KM option defines the Hive database, which will be used for creating the Hive HBase-SerDe table for unloading the HBase data. |

**Table A-6 (Cont.) LKM HBase to SQL SQOOP**

| Option | Description |
|---|---|
| SQOOP_PARALLELISM | Number of SQOOP parallel mappers.<br>Specifies the degree of parallelism. More precisely the number of mappers.<br>Number of mapper processes used for extraction.<br>When SQOOP_PARALLELISM > 1, SPLIT_BY must be defined. |
| WORK_TABLE_OPTIONS | Work table options.<br>Use this option to override standard technology specific work table options. When left blank, these options values are used.<br>Oracle: NOLOGGING<br>DB2 UDB: NOT LOGGED INITIALLY<br>Teradata: no fallback, no before journal, no after journal |
| TERADATA_WORK_TABLE_TYPE | Teradata work table type.<br>Use SET or MULTISET table for work table. |
| TERADATA_OUTPUT_METHOD | Teradata Load Method.<br>Specifies the way the Teradata Connector will load the data. Valid values are:<br>• batch.insert: multiple JDBC connections using batched prepared statements (simplest to start with)<br>• multiple.fastload: multiple FastLoad connections<br>• internal.fastload: single coordinated FastLoad connections (most performant)<br>Please see Cloudera's Teradata Connectors User Guide for more details. |
| TEMP_DIR | Local directory for temporary files.<br>Directory used for storing temporary files like squoop script, stdout and stderr redirects.<br>Leave blank to use system's default temp dir (<?=System.getProperty(java.io.tmp")?>)". |
| MAPRED_OUTPUT_BASE_DIR | MapReduce Output Directory.<br>This option specifies an hdfs directory, where SQOOP will create subdirectories for temporary files. A subdirectory called like the work table will be created here to hold the temporary data. |
| USE_GENERIC_JDBC_CONNECTOR | Use SQOOP's generic JDBC connector?<br>For certain technologies SQOOP provides specific connectors. These connectors take care of SQL-dialects and optimize performance. When there is a connector for the respective target technology, this connector should be used. If not, the generic JDBC connector may provide a solution. |
| EXTRA_HADOOP_CONF_PROPERTIES | Optional generic Hadoop properties.<br>Extra optional properties for SQOOP file: section Hadoop properties. |
| EXTRA_SQOOP_CONF_PROPERTIES | Optional SQOOP properties.<br>Extra optional properties for SQOOP file: section SQOOP properties. |
| EXTRA_SQOOP_CONNECTOR_CONF_PROPERTIES | Optional SQOOP connector properties.<br>Extra optional properties for SQOOP file: section SQOOP connector properties. |

## A.7 LKM HDFS File to Hive Load Data

This KM will load data only from HDFS file into Hive. The file can be in the format of JSON, Avro, Parquet, Delimited with complex data.

**Table A-7 LKM HDFS File to Hive Load Data**

| Option | Description |
|---|---|
| STOP_ON_FILE_NOT_FOUND | This checkbox option defines whether the KM should stop, if no input file is found. |
| OVERRIDE_ROW_FORMAT | This option allows to override the entire Hive row format definition of the staging table or the target table. |
| DELETE_TEMPORARY_OBJECTS | Set this option to No, to retain the temporary objects (tables, files and scripts) post integration. |

## A.8 LKM HDFS File to Hive Load Data (Direct)

This KM will load data only from HDFS file into Hive Data Direct directly into hive target table, bypassing the staging table for better performance.

**Table A-8 LKM HDFS to Hive Load Data (Direct)**

| Option | Description |
|---|---|
| STOP_ON_FILE_NOT_FOUND | This checkbox option defines whether the KM should stop, if no input file is found. |
| OVERRIDE_ROW_FORMAT | This option allows to override the entire Hive row format definition of the staging table or the target table. |
| DELETE_TEMPORARY_OBJECTS | Set this option to No, to retain the temporary objects (tables, files and scripts) post integration. |
| CREATE_TARG_TABLE | Create target table?<br>Check this option, to create the target table. |
| TRUNCATE | Replace existing target data?<br>Set this option to true, to replace any existing target table content with the new data. |

## A.9 IKM Hive Append

This KM integrates data into a Hive target table in append or replace (truncate) mode.

The following table descriptions the options for IKM Hive Append.

**Table A-9 IKM Hive Append**

| Option | Description |
|---|---|
| CREATE_TARG_TABLE | Create target table.<br>Check this option if you wish to create the target table. |

**Table A-9 (Cont.) IKM Hive Append**

| Option | Description |
|---|---|
| TRUNCATE | Replace all target table data.<br>Set this option to true, if you wish to replace the target table content with the new data. |

> **Note:**
>
> If there is a column containing a Complex Type in the target Hive table, this must not be left unmapped. Hive does not allow setting null values to complex columns.

## A.10 IKM Hive Incremental Update

This IKM integrates data incrementally into a Hive target table. The KM should be assigned on Hive target node.

Target data store integration type needs to be defined as Incremental Update to get this KM on the list of available KMs for assignment.

**Table A-10 IKM Hive Incremental Update**

| Option | Description |
|---|---|
| CREATE_TARG_TABLE | Create target table.<br>Select this option to create the target table. |
| TRUNCATE | Replace all target table data.<br>Set this option to true, to replace the target table content with the new data. |

## A.11 LKM File to Hive LOAD DATA

Integration from a flat file staging area to Hive using Hive's LOAD DATA command.

This KM executes the following steps:

1. Create a flow table in Hive
2. Declare data files to Hive (LOAD DATA command)
3. Load data from Hive staging table into target table

The KM can handle filename wildcards (*, ?).">

The following table describes the options for LKM File to Hive LOAD DATA.

**Table A-11 LKM File to Hive LOAD DATA**

| Option | Description |
|---|---|
| DELETE_TEMPORARY_OBJECTS | Delete temporary objects at end of mapping.<br>Set this option to NO, if you wish to retain temporary objects (tables, files and scripts) after integration. Useful for debugging. |

**Table A-11 (Cont.) LKM File to Hive LOAD DATA**

<table>
<tr><th>Option</th><th>Description</th></tr>
<tr><td>EXTERNAL_TABLE</td><td>Preserve file in original location?<br>Defines whether to declare the target/staging table as externally managed.<br>Default: false<br>For non-external tables Hive manages all data files. That is, it will *move* any data files into <hive.metastore.warehouse.dir>/<table_name>. For external tables Hive does not move or delete any files. It will load data from the location given by the ODI schema.<br>If EXTERNAL_TABLE is set to true:<br>All files in the directory given by the physical data schema will be loaded. So any filename or wildcard information from the source data store's resource name will be ignored.<br>The directory structure and file names must follow Hives directory organization for tables, for example, for partitioning and clustering.<br>The directory and its files must reside in HDFS.<br>No Hive LOAD-DATA-statements are submitted and thus loading of files to a specific partition (using a target-side expression) is not possible.</td></tr>
<tr><td>FILE_IS_LOCAL</td><td>Is this a local file?<br>Defines whether the source file is to be considered local (= outside of the current Hadoop cluster).<br>Default: true<br>If FILE_IS_LOCAL is set to true, the data file(s) are copied into the Hadoop cluster first.<br>If FILE_IS_LOCAL is set to false, the data file(s) are moved into the Hadoop cluster and therefore will no longer be available at their source location. If the source file is already in HDFS, FILE_IS_LOCAL=false results in just a file rename and therefore very fast operation. This option only applies, if EXTERNAL_TABLE is set to false.</td></tr>
<tr><td>STOP_ON_FILE_NOT_FOUND</td><td>Stop if no input file was found?<br>This checkbox option defines whether the KM should stop, if no input file has been found.</td></tr>
<tr><td>OVERRIDE_ROW_FORMAT</td><td>Custom row format clause.<br>This option allows to override the entire Hive row format definition of the staging table (in case USE_STAGE_TABLE is set to true) or the target table (in case USE_STAGE_TABLE is set to false). It contains the text to be used for row format definition.<br>Example for reading Apache Combined WebLog files:<br>ROW FORMAT SERDE 'org.apache.hadoop.hive.contrib.serde2.RegexSerDe' <EOL>WITH SERDEPROPERTIES ( <EOL> input.regex" = "([^ ]*) ([^ ]*) ([^ ]*) (-|\\[[^\\]]*\\]) ([^ \"]*|\"[^\"]*\") (-|[0-9]*) (-|[0-9]*) (\".*?\") (\".*?\") (\".*?\")"</td></tr>
</table>

## A.12 LKM File to Hive LOAD DATA Direct

Direct integration from a flat file into Hive without any staging using Hive's LOAD DATA command.

This is a direct load LKM and must be used without any IKM.

The KM can handle filename wildcards (*, ?).

The following table describes the options for LKM File to Hive LOAD DATA Direct.

**Table A-12 LKM File to Hive LOAD DATA Direct**

| Option | Description |
|---|---|
| CREATE_TARG_TABLE | Create target table.<br>Check this option if you wish to create the target table. |
| TRUNCATE | Replace all target table data.<br>Set this option to true, if you wish to replace the target table content with the new data. |
| DELETE_TEMPORARY_OBJECTS | Delete temporary objects at end of mapping.<br>Set this option to NO, if you wish to retain temporary objects (tables, files and scripts) after integration. Useful for debugging. |
| EXTERNAL_TABLE | Preserve file in original location?<br>Defines whether to declare the target/staging table as externally managed.<br>Default: false<br>For non-external tables Hive manages all data files. That is, it will *move* any data files into <hive.metastore.warehouse.dir>/<table_name>. For external tables Hive does not move or delete any files. It will load data from the location given by the ODI schema.<br>If EXTERNAL_TABLE is set to true:<br>All files in the directory given by the physical data schema will be loaded. So any filename or wildcard information from the source data store's resource name will be ignored.<br>The directory structure and file names must follow Hives directory organization for tables, for example, for partitioning and clustering.<br>The directory and its files must reside in HDFS.<br>No Hive LOAD-DATA-statements are submitted and thus loading of files to a specific partition (using a target-side expression) is not possible. |
| FILE_IS_LOCAL | Is this a local file?<br>Defines whether the source file is to be considered local (= outside of the current Hadoop cluster).<br>Default: true<br>If FILE_IS_LOCAL is set to true, the data file(s) are copied into the Hadoop cluster first.<br>If FILE_IS_LOCAL is set to false, the data file(s) are moved into the Hadoop cluster and therefore will no longer be available at their source location. If the source file is already in HDFS, FILE_IS_LOCAL=false results in just a file rename and therefore very fast operation. This option only applies, if EXTERNAL_TABLE is set to false. |
| STOP_ON_FILE_NOT_FOUND | Stop if no input file was found?<br>This checkbox option defines whether the KM should stop, if no input file has been found. |

**Table A-12 (Cont.) LKM File to Hive LOAD DATA Direct**

<table>
<tr><th>Option</th><th>Description</th></tr>
<tr><td>OVERRIDE_ROW_FORMAT</td><td>Custom row format clause.<br>This option allows to override the entire Hive row format definition of the staging table (in case USE_STAGE_TABLE is set to true) or the target table (in case USE_STAGE_TABLE is set to false). It contains the text to be used for row format definition.<br>Example for reading Apache Combined WebLog files:<br>ROW FORMAT SERDE 'org.apache.hadoop.hive.contrib.serde2.RegexSerDe' <EOL>WITH SERDEPROPERTIES ( <EOL> input.regex" = "([^ ]*) ([^ ]*) ([^ ]*) (-|\\[[^\\]]*\\]) ([^ \"]*|\"[^\"]*\") (-|[0-9]*) (-|[0-9]*) (\".*?\") (\".*?\") (\".*?\")"</td></tr>
</table>

# A.13 LKM HBase to Hive HBASE-SERDE

This LKM provides read access to a HBase table from the Hive.

This is achieved by defining a temporary load table definition on Hive which represents all relevant columns of the HBase source table.

# A.14 LKM Hive to HBase Incremental Update HBASE-SERDE Direct

This LKM loads data from Hive into HBase and supports inserting new rows and, also updating existing data.

This is a direct load LKM and must be used without any IKM.

The following table describes the options for LKM Hive to HBase Incremental Update HBASE-SERDE Direct.

**Table A-13 LKM Hive to HBase Incremental Update HBASE-SERDE Direct**

| Option | Description |
|---|---|
| CREATE_TARG_TABLE | Create target table.<br>Check this option to create the target table. |
| TRUNCATE | Replace all target table data.<br>Set this option to true, to replace the target table content with the new data. |
| HBASE_WAL | Disable Write-Ahead-Log.<br>HBase uses a Write-Ahead-Log to protect against data loss. For better performance, WAL can be disabled. This setting applies to all Hive commands executed later in this session. |
| DELETE_TEMPORARY_OBJECTS | Delete temporary objects at end of mapping.<br>Set this option to NO, to retain temporary objects (tables, files and scripts) after integration. Useful for debugging. |

## A.15 LKM Hive to File Direct

This LKM unloads data from Hive into flat files.

This is a direct load LKM and must be used without any IKM.

The following table describes the options for LKM Hive to File Direct.

**Table A-14 LKM Hive to File Direct**

| Option | Description |
|---|---|
| FILE_IS_LOCAL | Is this a local file?<br>Defines whether the target file is to be considered local (outside of the current Hadoop cluster). |
| STORED_AS | File format.<br>Defines whether the target file is to be stored as plain text file (TEXTFILE) or compressed (SEQUENCEFILE). |

## A.16 XKM Hive Sort

This XKM sorts data using an expression.

The following table describes the options for XKM Hive Sort.

**Table A-15 XKM Hive Sort**

| Option | Description |
|---|---|
| SORT_MODE | Select the mode the SORT operator will generate code for. |

## A.17 LKM File to Oracle OLH-OSCH

This KM integrates data from an HDFS file into an Oracle staging table using Oracle Loader for Hadoop (OLH) and/or Oracle SQL Connector for Hadoop (OSCH).

The KM can handle filename wildcards (*, ?).

The following table describes the options for LKM File to Oracle OLH-OSCH.

**Table A-16 LKM File to Oracle OLH-OSCH**

| Option | Description |
|---|---|
| DELETE_TEMPORARY_OBJECTS | Delete temporary objects at end of mapping.<br>Set this option to NO, to retain temporary objects (tables, files and scripts) after integration. Useful for debugging. |

Table A-16 (Cont.) LKM File to Oracle OLH-OSCH

| Option | Description |
|---|---|
| OLH_OUTPUT_MODE | How to transfer data into Oracle?<br>This option specifies how to load the Hadoop data into Oracle. Permitted values are JDBC, OCI, DP_COPY\|DP_OSCH, and OSCH.<br>• JDBC output mode: The data is inserted using several direct insert JDBC connections.<br>In very rare cases JDBC mode may result in duplicate records in target table due to Hadoop trying to restart tasks.<br>• OCI output mode: The data is inserted using several direct insert OCI connections in direct path mode.<br>For direct loading (no C$ table), the target table must be partitioned. For standard loading, FLOW_TABLE_OPTIONS must explicitly specify partitioning: for example, PARTITION BY HASH(COL1) PARTITIONS 4".<br>In very rare cases OCI mode may result in duplicate records in target table due to Hadoop trying to restart tasks.<br>• DP_COPY output mode: OLH creates several DataPump export files. These files are transferred by a "Hadoop fs -copyToLocal" command to the local path specified by EXT_TAB_DIR_LOCATION. - Please note that the path must be accessible by the Oracle Database engine. Once the copy job is complete. |
| REJECT_LIMIT | Max number of errors for OLH/EXTTAB.<br>Enter the maximum number of errors allowed in the file. Examples: UNLIMITED to except all errors. Integer value (10 to allow 10 rejections).<br>This value is used in OLH job definitions and, also in external table definitions. |
| EXT_TAB_DIR_LOCATION | Directory for ext tab data files.<br>File system path of the external table.<br>**Note:**<br>• Only applicable, if OLH_OUTPUT_MODE = DP_* or OSCH<br>• For OLH_OUTPUT_MODE = DP_*: this path must be accessible both from the ODI agent and from the target database engine.<br>• For OLH_OUTPUT_MODE = DP_*: the name of the external directory object is the I$ table name.<br>• For OLH_OUTPUT_MODE = DP_COPY: ODI agent will use hadoop-fs command to copy dp files into this directory.<br>• For OLH_OUTPUT_MODE = DP_*\|OSCH: this path will contain any external table log/bad/dsc files.<br>• ODI agent will remove any files from this directory during clean up before launching OLH/OSCH. |
| WORK_TABLE_OPTIONS | Option for Flow table creation.<br>Use this option to specify the attributes for the integration table at create time and used for increasing performance.<br>This option is set by default to NOLOGGING.<br>This option may be left empty. |

**Table A-16 (Cont.) LKM File to Oracle OLH-OSCH**

| Option | Description |
|---|---|
| OVERRIDE_INPUTFORMAT | Class name of InputFormat.<br>By default the InputFormat class is derived from the source Data Store/Technology (DelimitedTextInputFormat or HiveToAvroInputFormat). This option allows the user to specify the class name of a custom InputFormat.<br>Default: <empty>.<br>Cannot be used with OLH_OUTPUT_MODE=OSCH.<br>For example, for reading custom file formats like web log files the OLH RegexInputFormat can be used by assigning the value: oracle.hadoop.loader.lib.input.RegexInputFormat<br>See KM option EXTRA_OLH_CONF_PROPERTIES for details on how to specify the regular expression. |
| EXTRA_OLH_CONF_PROPERTIES | Optional extra OLH properties.<br>Allows adding extra parameters to OLH. For example, for changing the default OLH date format:<br><property><br><name>oracle.hadoop.loader.defaultDateFormat</name><br><value>yyyy-MM-dd HH:mm:ss</value><br></property><br>Particularly when using custom InputFormats (see KM option OVERRIDE_INPUTFORMAT for details) the InputFormat may require additional configuration parameters. These are provided in the OLH configuration file. This KM option allows adding extra properties to the OLH configuration file. Default: <empty><br>Cannot be used with OLH_OUTPUT_MODE=OSCH<br>Example (loading apache weblog file format):<br>When OLH RegexInputFormat is used for reading custom file formats, this KM option specified the regular expression and other parsing details:<br><property><br><name>oracle.hadoop.loader.input.regexPattern</name><br><value>([^ ]*) ([^ ]*) ([^ ]*) (-\|\[[^\]]*\]) ([^ \]*\|\"[^\"]*\") (-\|[0-9]*) (-\|[0-9]*) (\".*?\") (\".*?\") (\".*?\")</value><br><description>RegEx for Apache WebLog format</description><br></property>" |
| MAPRED_OUTPUT_BASE_DIR | MapReduce Output Directory.<br>This option specifies an hdfs directory, where SQOOP will create subdirectories for temporary files. A subdirectory called like the work table will be created here to hold the temporary data. |
| TEMP_DIR | Local directory for temporary files.<br>Directory used for storing temporary files like squoop script, stdout, and stderr redirects.<br>Leave blank to use system's default temp dir (<?=System.getProperty(java.io.tmp")?>)". |

# A.18 LKM File to Oracle OLH-OSCH Direct

This KM integrates data from an HDFS file into an Oracle target using Oracle Loader for Hadoop (OLH) and/or Oracle SQL Connector for Hadoop (OSCH)

The KM can handle filename wildcards (*, ?).

This is a direct load LKM (no staging) and must be used without any IKM.

The following table describes the options for LKM File to Oracle OLH-OSCH Direct.

**Table A-17 LKM File to Oracle OLH-OSCH Direct**

| Option | Description |
|---|---|
| CREATE_TARG_TABLE | Create target table.<br>Check this option to create the target table. |
| TRUNCATE | Replace all target table data.<br>Set this option to true, to replace the target table content with the new data. |
| DELETE_ALL | Delete all rows.<br>Set this option to true, to replace the target table content with the new data. |
| DELETE_TEMPORARY_OBJECTS | Delete temporary objects at end of mapping.<br>Set this option to NO, to retain temporary objects (tables, files and scripts) after integration. Useful for debugging. |
| OLH_OUTPUT_MODE | How to transfer data into Oracle?<br>This option specifies how to load the Hadoop data into Oracle. Permitted values are JDBC, OCI, DP_COPY\|DP_OSCH, and OSCH.<br>• JDBC output mode: The data is inserted using several direct insert JDBC connections.<br>In very rare cases JDBC mode may result in duplicate records in target table due to Hadoop trying to restart tasks.<br>• OCI output mode: The data is inserted using several direct insert OCI connections in direct path mode.<br>For direct loading (no C$ table), the target table must be partitioned. For standard loading, FLOW_TABLE_OPTIONS must explicitly specify partitioning: for example, PARTITION BY HASH(COL1) PARTITIONS 4".<br>In very rare cases OCI mode may result in duplicate records in target table due to Hadoop trying to restart tasks.<br>• DP_COPY output mode: OLH creates several DataPump export files. These files are transferred by a "Hadoop fs -copyToLocal" command to the local path specified by EXT_TAB_DIR_LOCATION. - Please note that the path must be accessible by the Oracle Database engine. Once the copy job is complete. |
| REJECT_LIMIT | Max number of errors for OLH/EXTTAB.<br>Enter the maximum number of errors allowed in the file. Examples: UNLIMITED to except all errors. Integer value (10 to allow 10 rejections).<br>This value is used in OLH job definitions and, also in external table definitions. |

**Table A-17 (Cont.) LKM File to Oracle OLH-OSCH Direct**

| Option | Description |
|---|---|
| EXT_TAB_DIR_LOCATION | Directory for ext tab data files.<br>File system path of the external table.<br>**Note:**<br>• Only applicable, if OLH_OUTPUT_MODE = DP_* or OSCH<br>• For OLH_OUTPUT_MODE = DP_*: this path must be accessible both from the ODI agent and from the target database engine.<br>• For OLH_OUTPUT_MODE = DP_*: the name of the external directory object is the I$ table name.<br>• For OLH_OUTPUT_MODE = DP_COPY: ODI agent will use hadoop-fs command to copy dp files into this directory.<br>• For OLH_OUTPUT_MODE = DP_*\|OSCH: this path will contain any external table log/bad/dsc files.<br>• ODI agent will remove any files from this directory during clean up before launching OLH/OSCH. |
| WORK_TABLE_OPTIONS | Option for Flow table creation.<br>Use this option to specify the attributes for the integration table at create time and used for increasing performance.<br>This option is set by default to NOLOGGING.<br>This option may be left empty. |
| OVERRIDE_INPUTFORMAT | Class name of InputFormat.<br>By default the InputFormat class is derived from the source Data Store/Technology (DelimitedTextInputFormat or HiveToAvroInputFormat). This option allows the user to specify the class name of a custom InputFormat.<br>Default: <empty>.<br>Cannot be used with OLH_OUTPUT_MODE=OSCH.<br>For example, for reading custom file formats like web log files the OLH RegexInputFormat can be used by assigning the value: oracle.hadoop.loader.lib.input.RegexInputFormat<br>See KM option EXTRA_OLH_CONF_PROPERTIES for details on how to specify the regular expression. |

**Table A-17 (Cont.) LKM File to Oracle OLH-OSCH Direct**

| Option | Description |
|---|---|
| EXTRA_OLH_CONF_PROPERTIES | Optional extra OLH properties.<br>Allows adding extra parameters to OLH. For example, for changing the default OLH date format:<br><property><br><name>oracle.hadoop.loader.defaultDateFormat</name><br><value>yyyy-MM-dd HH:mm:ss</value><br></property><br>Particularly when using custom InputFormats (see KM option OVERRIDE_INPUTFORMAT for details) the InputFormat may require additional configuration parameters. These are provided in the OLH configuration file. This KM option allows adding extra properties to the OLH configuration file. Default: <empty><br>Cannot be used with OLH_OUTPUT_MODE=OSCH<br>Example (loading apache weblog file format):<br>When OLH RegexInputFormat is used for reading custom file formats, this KM option specified the regular expression and other parsing details:<br><property><br><name>oracle.hadoop.loader.input.regexPattern</name><br><value>([^ ]*) ([^ ]*) ([^ ]*) (-\|\[[^\]]*\]) ([^ \]*\|\"[^\"]*\") (-\|[0-9]*) (-\|[0-9]*) (\".*?\") (\".*?\") (\".*?\")</value><br><description>RegEx for Apache WebLog format</description><br></property>" |
| MAPRED_OUTPUT_BASE_DIR | MapReduce Output Directory.<br>This option specifies an hdfs directory, where SQOOP will create subdirectories for temporary files. A subdirectory called like the work table will be created here to hold the temporary data. |
| TEMP_DIR | Local directory for temporary files.<br>Directory used for storing temporary files like squoop script, stdout, and stderr redirects.<br>Leave blank to use system's default temp dir (<?=System.getProperty(java.io.tmp")?>)". |

## A.19 LKM Hive to Oracle OLH-OSCH

This KM integrates data from a Hive query into an Oracle staging table using Oracle Loader for Hadoop (OLH) and/or Oracle SQL Connector for Hadoop (OSCH).

The following table describes the options for LKM Hive to Oracle OLH-OSCH.

**Table A-18 LKM Hive to Oracle OLH-OSCH**

| Option | Description |
|---|---|
| USE_HIVE_STAGING_TABLE | Use intermediate Hive staging table?<br>By default the Hive source data materializes in a Hive staging table before extraction by OLH. If USE_HIVE_STAGING_TABLE is set to false, OLH directly accesses the Hive source data.<br>USE_HIVE_STAGING_TABLE=0 is only possible, if all these conditions are true.<br>• Only a single source table<br>• No transformations, filters, joins.<br>• No datasets<br>• USE_HIVE_STAGING_TABLE=0 provides better performance by avoiding an extra data transfer step. |
| DELETE_TEMPORARY_OBJECTS | Delete temporary objects at end of mapping.<br>Set this option to NO, to retain temporary objects (tables, files and scripts) after integration. Useful for debugging. |
| OLH_OUTPUT_MODE | How to transfer data into Oracle?<br>This option specifies how to load the Hadoop data into Oracle. Permitted values are JDBC, OCI, DP_COPY\|DP_OSCH, and OSCH.<br>• JDBC output mode: The data is inserted using several direct insert JDBC connections.<br>In very rare cases JDBC mode may result in duplicate records in target table due to Hadoop trying to restart tasks.<br>• OCI output mode: The data is inserted using several direct insert OCI connections in direct path mode.<br>For direct loading (no C$ table), the target table must be partitioned. For standard loading, FLOW_TABLE_OPTIONS must explicitly specify partitioning: for example, PARTITION BY HASH(COL1) PARTITIONS 4".<br>In very rare cases OCI mode may result in duplicate records in target table due to Hadoop trying to restart tasks.<br>• DP_COPY output mode: OLH creates several DataPump export files. These files are transferred by a "Hadoop fs -copyToLocal" command to the local path specified by EXT_TAB_DIR_LOCATION. The path must be accessible by the Oracle Database engine. Once the copy job is complete. |
| REJECT_LIMIT | Max number of errors for OLH/EXTTAB.<br>Enter the maximum number of errors allowed in the file. Examples: UNLIMITED to except all errors. Integer value (10 to allow 10 rejections).<br>This value is used in OLH job definitions and, also in external table definitions. |

**Table A-18 (Cont.) LKM Hive to Oracle OLH-OSCH**

| Option | Description |
|---|---|
| EXT_TAB_DIR_LOCATION | Directory for ext tab data files.<br>File system path of the external table.<br>**Note:**<br>• Only applicable, if OLH_OUTPUT_MODE = DP_* or OSCH<br>• For OLH_OUTPUT_MODE = DP_*: this path must be accessible both from the ODI agent and from the target database engine.<br>• For OLH_OUTPUT_MODE = DP_*: the name of the external directory object is the I$ table name.<br>• For OLH_OUTPUT_MODE = DP_COPY: ODI agent will use hadoop-fs command to copy dp files into this directory.<br>• For OLH_OUTPUT_MODE = DP_*\|OSCH: this path will contain any external table log/bad/dsc files.<br>• ODI agent will remove any files from this directory during clean up before launching OLH/OSCH. |
| WORK_TABLE_OPTIONS | Option for Flow table creation.<br>Use this option to specify the attributes for the integration table at create time and used for increasing performance.<br>This option is set by default to NOLOGGING.<br>This option may be left empty. |
| OVERRIDE_INPUTFORMAT | Class name of InputFormat.<br>By default the InputFormat class is derived from the source Data Store/Technology (DelimitedTextInputFormat or HiveToAvroInputFormat). This option allows the user to specify the class name of a custom InputFormat.<br>Default: <empty>.<br>Cannot be used with OLH_OUTPUT_MODE=OSCH.<br>For example, for reading custom file formats like web log files the OLH RegexInputFormat can be used by assigning the value: oracle.hadoop.loader.lib.input.RegexInputFormat<br>See KM option EXTRA_OLH_CONF_PROPERTIES for details on how to specify the regular expression. |

**Table A-18 (Cont.) LKM Hive to Oracle OLH-OSCH**

| Option | Description |
|---|---|
| EXTRA_OLH_CONF_PROPERTIES | Optional extra OLH properties.<br>Allows adding extra parameters to OLH. For example, for changing the default OLH date format:<br><property><br><name>oracle.hadoop.loader.defaultDateFormat</name><br><value>yyyy-MM-dd HH:mm:ss</value><br></property><br>Particularly when using custom InputFormats (see KM option OVERRIDE_INPUTFORMAT for details) the InputFormat may require additional configuration parameters. These are provided in the OLH configuration file. This KM option allows adding extra properties to the OLH configuration file. Default: <empty><br>Cannot be used with OLH_OUTPUT_MODE=OSCH<br>Example (loading apache weblog file format):<br>When OLH RegexInputFormat is used for reading custom file formats, this KM option specified the regular expression and other parsing details:<br><property><br><name>oracle.hadoop.loader.input.regexPattern</name><br><value>([^ ]*) ([^ ]*) ([^ ]*) (-\|\[[^\]]*\]) ([^ \]*\|\"[^\"]*\") (-\|[0-9]*) (-\|[0-9]*) (\".*?\") (\".*?\") (\".*?\")</value><br><description>RegEx for Apache WebLog format</description><br></property>" |
| MAPRED_OUTPUT_BASE_DIR | MapReduce Output Directory.<br>This option specifies an hdfs directory, where SQOOP will create subdirectories for temporary files. A subdirectory called like the work table will be created here to hold the temporary data. |
| TEMP_DIR | Local directory for temporary files.<br>Directory used for storing temporary files like squoop script, stdout, and stderr redirects.<br>Leave blank to use system's default temp dir (<?=System.getProperty(java.io.tmp")?>)". |

# A.20 LKM Hive to Oracle OLH-OSCH Direct

This KM integrates data from a Hive query into an Oracle target using Oracle Loader for Hadoop (OLH) and/or Oracle SQL Connector for Hadoop (OSCH)

This is a direct load LKM and must be used without any IKM.

The following table describes the options for LKM Hive to Oracle OLH-OSCH.

**Table A-19 LKM Hive to Oracle OLH-OSCH Direct**

| Option | Description |
|---|---|
| CREATE_TARG_TABLE | Create target table.<br>Check this option to create the target table. |

**Table A-19 (Cont.) LKM Hive to Oracle OLH-OSCH Direct**

| Option | Description |
|---|---|
| TRUNCATE | Replace all target table data.<br>Set this option to true, to replace the target table content with the new data. |
| DELETE_ALL | Delete all rows.<br>Set this option to true, to replace the target table content with the new data. |
| USE_HIVE_STAGING_TABLE | Use intermediate Hive staging table?<br>By default the Hive source data materializes in a Hive staging table before extraction by OLH. If USE_HIVE_STAGING_TABLE is set to false, OLH directly accesses the Hive source data.<br>USE_HIVE_STAGING_TABLE=0 is only possible, if all these conditions are true.<br>• Only a single source table<br>• No transformations, filters, joins.<br>• No datasets<br>• USE_HIVE_STAGING_TABLE=0 provides better performance by avoiding an extra data transfer step. |
| DELETE_TEMPORARY_OBJECTS | Delete temporary objects at end of mapping.<br>Set this option to NO, to retain temporary objects (tables, files and scripts) after integration. Useful for debugging. |
| OLH_OUTPUT_MODE | How to transfer data into Oracle?<br>This option specifies how to load the Hadoop data into Oracle. Permitted values are JDBC, OCI, DP_COPY\|DP_OSCH, and OSCH.<br>• JDBC output mode: The data is inserted using several direct insert JDBC connections.<br>In very rare cases JDBC mode may result in duplicate records in target table due to Hadoop trying to restart tasks.<br>• OCI output mode: The data is inserted using several direct insert OCI connections in direct path mode.<br>For direct loading (no C$ table), the target table must be partitioned. For standard loading, FLOW_TABLE_OPTIONS must explicitly specify partitioning: For example, PARTITION BY HASH(COL1) PARTITIONS 4".<br>In very rare cases OCI mode may result in duplicate records in target table due to Hadoop trying to restart tasks.<br>• DP_COPY output mode: OLH creates several DataPump export files. These files are transferred by a "Hadoop fs -copyToLocal" command to the local path specified by EXT_TAB_DIR_LOCATION. - Please note that the path must be accessible by the Oracle Database engine. Once the copy job is complete. |
| REJECT_LIMIT | Max number of errors for OLH/EXTTAB.<br>Enter the maximum number of errors allowed in the file. Examples: UNLIMITED to except all errors. Integer value (10 to allow 10 rejections).<br>This value is used in OLH job definitions and, also in external table definitions. |

**Table A-19 (Cont.) LKM Hive to Oracle OLH-OSCH Direct**

| Option | Description |
|---|---|
| EXT_TAB_DIR_LOCATION | Directory for ext tab data files.<br>File system path of the external table.<br>**Note:**<br>• Only applicable, if OLH_OUTPUT_MODE = DP_* or OSCH<br>• For OLH_OUTPUT_MODE = DP_*: this path must be accessible both from the ODI agent and from the target database engine.<br>• For OLH_OUTPUT_MODE = DP_*: the name of the external directory object is the I$ table name.<br>• For OLH_OUTPUT_MODE = DP_COPY: ODI agent will use hadoop-fs command to copy dp files into this directory.<br>• For OLH_OUTPUT_MODE = DP_*\|OSCH: this path will contain any external table log/bad/dsc files.<br>• ODI agent will remove any files from this directory during clean up before launching OLH/OSCH. |
| WORK_TABLE_OPTIONS | Option for Flow table creation.<br>Use this option to specify the attributes for the integration table at create time and used for increasing performance.<br>This option is set by default to NOLOGGING.<br>This option may be left empty. |
| OVERRIDE_INPUTFORMAT | Class name of InputFormat.<br>By default the InputFormat class is derived from the source Data Store/Technology (DelimitedTextInputFormat or HiveToAvroInputFormat). This option allows the user to specify the class name of a custom InputFormat.<br>Default: <empty>.<br>Cannot be used with OLH_OUTPUT_MODE=OSCH.<br>For example, for reading custom file formats like web log files the OLH RegexInputFormat can be used by assigning the value: oracle.hadoop.loader.lib.input.RegexInputFormat<br>See KM option EXTRA_OLH_CONF_PROPERTIES for details on how to specify the regular expression. |

**Table A-19 (Cont.) LKM Hive to Oracle OLH-OSCH Direct**

| Option | Description |
|---|---|
| EXTRA_OLH_CONF_PROPERTIES | Optional extra OLH properties.<br>Allows adding extra parameters to OLH. For example, for changing the default OLH date format:<br><property><br><name>oracle.hadoop.loader.defaultDateFormat</name><br><value>yyyy-MM-dd HH:mm:ss</value><br></property><br>Particularly when using custom InputFormats (see KM option OVERRIDE_INPUTFORMAT for details) the InputFormat may require additional configuration parameters. These are provided in the OLH configuration file. This KM option allows adding extra properties to the OLH configuration file. Default: <empty><br>Cannot be used with OLH_OUTPUT_MODE=OSCH<br>Example (loading apache weblog file format):<br>When OLH RegexInputFormat is used for reading custom file formats, this KM option specified the regular expression and other parsing details:<br><property><br><name>oracle.hadoop.loader.input.regexPattern</name><br><value>([^ ]*) ([^ ]*) ([^ ]*) (-\|\[[^\]]*\]) ([^ \]*\|\"[^\"]*\") (-\|[0-9]*) (-\|[0-9]*) (\".*?\") (\".*?\") (\".*?\")</value><br><description>RegEx for Apache WebLog format</description><br></property>" |
| MAPRED_OUTPUT_BASE_DIR | MapReduce Output Directory.<br>This option specifies an hdfs directory, where SQOOP will create subdirectories for temporary files. A subdirectory called like the work table will be created here to hold the temporary data. |
| TEMP_DIR | Local directory for temporary files.<br>Directory used for storing temporary files like squoop script, stdout, and stderr redirects.<br>Leave blank to use system's default temp dir (<?=System.getProperty(java.io.tmp")?>)". |

## A.21 RKM Hive

RKM Hive reverses these metadata elements:

- Hive tables and views as data stores.

  Specify the reverse mask in the Mask field, and then select the tables and views to reverse. The Mask field in the Reverse Engineer tab filters reverse-engineered objects based on their names. The Mask field cannot be empty and must contain at least the percent sign (%).
- Hive columns as attributes with their data types.
- Information about buckets, partitioning, clusters, and sort columns are set in the respective flex fields in the data store or column metadata.

## A.22 RKM HBase

RKM HBase reverses these metadata elements:

- HBase tables as data stores.

  Specify the reverse mask in the Mask field, and then select the tables to reverse. The Mask field in the Reverse Engineer tab filters reverse-engineered objects based on their names. The Mask field cannot be empty and must contain at least the percent sign (%).
- HBase columns as attributes with their data types.
- HBase unique row key as attribute called `key`.

> **Note:**
>
> This RKM uses the `oracle.odi.km` logger for logging. You can enable logging by changing log level for `oracle.odi.km` logger to TRACE:16 in `ODI-logging-config.xml` as shown below:
>
> ```
> <logger name="oracle.odi.km" level="TRACE:16" useParentHandlers="true"/>
> <logger name="oracle.odi.studio.message.logger.proxy" level="TRACE:16"
> useParentHandlers="false"/>
> ```
>
> For more information about logging configuration in ODI, see the Runtime Logging for ODI components section in *Administering Oracle Data Integrator*.

The following table describes the options for RKM HBase.

**Table A-20 RKM HBase Options**

| Option | Description |
|---|---|
| `SCAN_MAX_ROWS` | Specifies the maximum number of rows to be scanned during reversing of a table. The default value is `10000`. |
| `SCAN_START_ROW` | Specifies the key of the row to start the scan on. By default the scan will start on the first row. The row key is specified as a Java expressions returning an instance of `org.apache.hadoop.hbase.util.Bytes`. Example: `Bytes.toBytes(?EMP000001?)`. |
| `SCAN_STOP_ROW` | Specifies the key of the row to stop the scan on? By default the scan will run to the last row of the table or up to `SCAN_MAX_ROWS` is reached. The row key is specified as a Java expressions returning an instance of `org.apache.hadoop.hbase.util.Bytes`. Example: `Bytes.toBytes(?EMP000999?)`.<br>Only applies if `SCAN_START_ROW` is specified. |
| `SCAN_ONLY_FAMILY` | Restricts the scan to column families, whose name match this pattern. SQL-LIKE wildcards percentage (`%`) and underscore (`_`) can be used. By default all column families are scanned. |

## A.23 IKM File to Hive (Deprecated)

**Note:** This KM is deprecated and only used for backward compatibility.

IKM File to Hive (Load Data) supports:

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

The following table describes the options for IKM File to Hive (Load Data). See the knowledge module for additional details.

**Table A-21 IKM File to Hive Options**

| Option | Description |
|---|---|
| `CREATE_TARG_TABLE` | Check this option, if you wish to create the target table. In case `USE_STAGING_TABLE` is set to `false`, the data will only be read correctly, if the target table definition, particularly the row format and file format details, are correct. |
| `TRUNCATE` | Set this option to true, if you wish to replace the target table/partition content with the new data. Otherwise the new data will be appended to the target table. If `TRUNCATE` and `USE_STAGING_TABLE` are set to `false`, all source file names must be unique and must not collide with any data files already loaded into the target table. |
| `FILE_IS_LOCAL` | Defines whether the source file is to be considered local (outside of the current Hadoop cluster). If this option is set to `true`, the data file(s) are copied into the Hadoop cluster first. The file has to be accessible by the Hive server through the local or shared file system. If this option is set to `false`, the data file(s) are moved into the Hadoop cluster and therefore will no longer be available at their source location. If the source file is already in HDFS, setting this option is set to `false` results in just a file rename, and therefore the operation is very fast.<br>This option only applies, if `EXTERNAL_TABLE` is set to `false`. |

**Table A-21 (Cont.) IKM File to Hive Options**

| Option | Description |
|---|---|
| EXTERNAL_TABLE | Defines whether to declare the target/staging table as externally managed. For non-external tables Hive manages all data files. That is, it will move any data files into <hive.metastore.warehouse.dir>/<table_name>. For external tables Hive does not move or delete any files. It will load data from the location given by the ODI schema.<br>If this option is set to true:<br>• All files in the directory given by the physical data schema will be loaded. So any filename or wildcard information from the source data store's resource name will be ignored.<br>• The directory structure and file names must several Hives directory organization for tables, for example, for partitioning and clustering.<br>• The directory and its files must reside in HDFS.<br>• No Hive LOAD-DATA-statements are submitted and thus loading of files to a specific partition (using a target-side expression) is not possible. |
| USE_STAGING_TABLE | Defines whether an intermediate staging table will be created.<br>A Hive staging table is required if:<br>• Target table is partitioned, but data spreads across partitions<br>• Target table is clustered<br>• Target table (partition) is sorted, but input file is not<br>• Target table is already defined and target table definition does not match the definition required by the KM<br>• Target column order does not match source file column order<br>• There are any unmapped source columns<br>• There are any unmapped non-partition target columns<br>• The source is a fixed length file and the target has non-string columns<br>In case none of the above is true, this option can be turned off for better performance. |
| DELETE_TEMPORARY_OBJECTS | Removes temporary objects, such as tables, files, and scripts after integration. Set this option to No if you want to retain the temporary files, which might be useful for debugging. |
| DEFER_TARGET_LOAD | Defines whether the file(s), which have been declared to the staging table should be loaded into the target table now or during a later execution. Permitted values are START, NEXT, END or <empty>.<br>This option only applies if USE_STAGE_TABLE is set to true.<br>The typical use case for this option is when there are multiple files and each of them requires data redistribution/sorting and the files are gathered by calling the interface several times. For example, the interface is used in a package, which retrieves (many small) files from different locations and the location, stored in an Oracle Data Integrator variable, is to be used in a target partition column. In this case the first interface execution will have DEFER_TARGET_LOAD set to START, the next interface executions will have DEFER_TARGET_LOAD set to NEXT and set to END for the last interface. The interfaces having DEFER_TARGET _LOAD set to START/NEXT will just load the data file into HDFS (but not yet into the target table) and can be executed in parallel to accelerate file upload to cluster. |

**Table A-21 (Cont.) IKM File to Hive Options**

<table>
<tr><th>Option</th><th>Description</th></tr>
<tr><td>OVERRIDE_ROW_FORMAT</td><td>Allows to override the entire Hive row format definition of the staging table (in case USE_STAGE_TABLE is set to true) or the target table (in case USE_STAGE_TABLE is set to false). It contains the text to be used for row format definition.Example for reading Apache Combined WebLog files:<br>ROW FORMAT SERDE 'org.apache.hadoop.hive.contrib.serde2.RegexSerDe' WITH SERDEPROPERTIES ( "input.regex" = "([^ ]*) ([^ ]*) ([^ ]*) (-|\\[[^\\]]*\\]) ([^ \"]*|\"[^\"]*\") (-|[0-9]*) (-|[0-9]*) (\".*?\") (\".*?\") (\".*?\")", "output.format.string" = "%1$s %2$s %3$s %4$s %5$s %6$s %7$s %8$s %9$s %10$s" ) STORED AS TEXTFILE<br>The list of columns in the source data store must match the list of input groups in the regular expression (same number of columns and appropriate data types). If USE_STAGE_TABLE is set to false, the number of target columns must match the number of columns returned by the SerDe, in the above example, the number of groups in the regular expression. The number of source columns is ignored (At least one column must be mapped to the target.). All source data is mapped into the target table structure according to the column order, the SerDe's first column is mapped to the first target column, the SerDe's second column is mapped to the second target column, and so on. If USE_STAGE_TABLE is set to true, the source data store must have as many columns as the SerDe returns columns. Only data of mapped columns will be transferred.</td></tr>
<tr><td>STOP_ON_FILE_NOT_FOUND</td><td>Defines whether the KM should stop, if input file is not found.</td></tr>
<tr><td>HIVE_COMPATIBILE</td><td>Specifies the Hive version compatibility. The values permitted for this option are 0.7 and 0.8.<br>• 0.7: Simulates the append behavior. Must be used for Hive 0.7 (CDH3).<br>• 0.8: Uses Hive's append feature, which provides better performance. Requires Hive 0.8 (CDH4) or later.</td></tr>
</table>

## A.24 LKM HBase to Hive (HBase-SerDe) [Deprecated]

**Note:** This KM is deprecated and only used for backward compatibility.

LKM HBase to Hive (HBase-SerDe) supports:

- A single source HBase table.

The following table describes the options for LKM HBase to Hive (HBase-SerDe). See the knowledge module for additional details.

**Table A-22 LKM HBase to Hive (HBase-SerDe) Options**

| Option | Description |
|---|---|
| DELETE_TEMPORARY_OBJECTS | Deletes temporary objects such as tables, files, and scripts post data integration. Set this option to NO, to retain the temporary objects, which might be useful for debugging. |

## A.25 IKM Hive to HBase Incremental Update (HBase-SerDe) [Deprecated]

**Note:** This KM is deprecated and only used for backward compatibility.

IKM Hive to HBase Incremental Update (HBase-SerDe) supports:

- Filters, Joins, Datasets, Transformations and Aggregations in Hive
- Inline views generated by IKM Hive Transform
- Inline views generated by IKM Hive Control Append

The following table describes the options for IKM Hive to HBase Incremental Update (HBase-SerDe). See the knowledge module for additional details.

**Table A-23 IKM Hive to HBase Incremental Update (HBase-SerDe) Options**

| Option | Description |
|---|---|
| `CREATE_TARG_TABLE` | Creates the HBase target table. |
| `TRUNCATE` | Replaces the target table content with the new data. If this option is set to `false`, the new data is appended to the target table. |
| `DELETE_TEMPORARY_OBJECTS` | Deletes temporary objects such as tables, files, and scripts post data integration. Set this option to `NO`, to retain the temporary objects, which might be useful for debugging. |
| `HBASE_WAL` | Enables or disables the Write-Ahead-Log (WAL) that HBase uses to protect against data loss. For better performance, WAL can be disabled. |

## A.26 IKM SQL to Hive-HBase-File (SQOOP) [Deprecated]

**Note:** This KM is deprecated and only used for backward compatibility.

IKM SQL to Hive-HBase-File (SQOOP) supports:

- Mappings on staging
- Joins on staging
- Filter expressions on staging
- Datasets
- Lookups
- Derived tables

The following table describes the options for IKM SQL to Hive-HBase-File (SQOOP). See the knowledge module for additional details.

**Table A-24 IKM SQL to Hive-HBase-File (SQOOP) Options**

| Option | Description |
|---|---|
| `CREATE_TARG_TABLE` | Creates the target table. This option is applicable only if the target is Hive or HBase. |

**Table A-24 (Cont.) IKM SQL to Hive-HBase-File (SQOOP) Options**

| Option | Description |
|---|---|
| TRUNCATE | Replaces any existing target table content with the new data. For Hive and HBase targets, the target data is truncated. For File targets, the target directory is removed. For File targets, this option must be set to true. |
| SQOOP_PARALLELISM | Specifies the degree of parallelism. More precisely the number of mapper processes used for extraction.<br>If SQOOP_PARALLELISM option is set to greater than 1, SPLIT_BY option must be defined. |
| SPLIT_BY | Specifies the target column to be used for splitting the source data into n chunks for parallel extraction, where n is SQOOP_PARALLELISM. To achieve equally sized data chunks the split column should contain homogeneously distributed values. For calculating the data chunk boundaries a query similar to SELECT MIN(EMP.EMPNO), MAX(EMP.EMPNO) from EMPLOYEE EMP is used. To avoid an extra full table scan the split column should be backed by an index. |
| BOUNDARY_QUERY | For splitting the source data into chunks for parallel extraction the minimum and maximum value of the split column is retrieved (KM option SPLIT-BY). In certain situations this may not be the best boundaries or not the most optimized way to retrieve the boundaries. In such cases this KM option can be set to a SQL query returning one row with two columns, lowest value and highest value to be used for split-column. This range will be divided into SQOOP_PARALLELISM chunks for parallel extraction. Example for hard-coded ranges for an Oracle source:<br>SELECT 1000, 2000 FROM DUAL<br>For preserving context independence, regular table names should be inserted through odiRef.getObjectName calls. For example:<br>SELECT MIN(EMPNO), MAX(EMPNO) FROM <%=odiRef.getObjectName("EMP")%> |
| TEMP_DIR | Specifies the directory used for storing temporary files, such as sqoop script, stdout and stderr redirects. Leave this option blank to use system's default temp directory:<br><?=System.getProperty("java.io.tmp")?> |
| MAPRED_OUTPUT_BASE_DIR | Specifies an hdfs directory, where SQOOP creates subdirectories for temporary files. A subdirectory called like the work table will be created here to hold the temporary data. |
| DELETE_TEMPORARY_OBJECTS | Deletes temporary objects such as tables, files, and scripts after data integration. Set this option to NO, to retain the temporary objects, which might be useful for debugging. |
| USE_HIVE_STAGING_TABLE | Loads data into the Hive work table before loading into the Hive target table. Set this option to false to load data directly into the target table.<br>Setting this option to false is only possible, if all these conditions are true:<br>• All target columns are mapped<br>• Existing Hive table uses standard hive row separators (\n) and column delimiter (\01)<br>Setting this option to false provides better performance by avoiding an extra data transfer step.<br>This option is applicable only if the target technology is Hive. |

**Table A-24 (Cont.) IKM SQL to Hive-HBase-File (SQOOP) Options**

| Option | Description |
|---|---|
| USE_GENERIC_JDBC_CONNECTOR | Specifies whether to use the generic JDBC connector if a connector for the target technology is not available.<br>For certain technologies SQOOP provides specific connectors. These connectors take care of SQL-dialects and optimize performance. When there is a connector for the respective target technology, this connector should be used. If not, the generic JDBC connector can be used. |
| EXTRA_HADOOP_CONF_PROPERTIES | Optional generic Hadoop properties. |
| EXTRA_SQOOP_CONF_PROPERTIES | Optional SQOOP properties. |
| EXTRA_SQOOP_CONNECTOR_CONF_PROPERTIES | Optional SQOOP connector properties. |

# A.27 IKM Hive Control Append (Deprecated)

**Note:** This KM is deprecated and only used for backward compatibility.

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

The following table describes the options for IKM Hive Control Append.

**Table A-25 IKM Hive Control Append Options**

| Option | Description |
|---|---|
| FLOW_CONTROL | Activates flow control. |
| RECYCLE_ERRORS | Recycles data rejected from a previous control. |
| STATIC_CONTROL | Controls the target table after having inserted or updated target data. |
| CREATE_TARG_TABLE | Creates the target table. |
| TRUNCATE | Replaces the target table content with the new data. Setting this option to `true` provides better performance. |
| DELETE_TEMPORARY_OBJECTS | Removes the temporary objects, such as tables, files, and scripts after data integration. Set this option to `NO`, to retain the temporary objects, which might be useful for debugging. |
| HIVE_COMPATIBILE | Specifies the Hive version compatibility. The values permitted for this option are 0.7 and 0.8.<br>• 0.7: Simulates the append behavior. Must be used for Hive 0.7 (CDH3).<br>• 0.8: Uses Hive's append feature, which provides better performance. Requires Hive 0.8 (CDH4) or later. |

## A.28 CKM Hive

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

The following table describes the options for this check knowledge module.

**Table A-26 CKM Hive Options**

| Option | Description |
|---|---|
| `DROP_ERROR_TABLE` | Drops error table before execution. When this option is set to `YES`, the error table will be dropped each time a control is performed on the target table. This means that any rejected records, identified and stored during previous control operations, will be lost. Otherwise previous rejects will be preserved. In addition to the error table, any table called `<error table>_tmp` will also be dropped. |
| `HIVE_COMPATIBILE` | Specifies the Hive version compatibility. The values permitted for this option are 0.7 and 0.8.<br>• 0.7: Simulates the append behavior. Must be used for Hive 0.7 (CDH3).<br>• 0.8: Uses Hive's append feature, which provides better performance. Requires Hive 0.8 (CDH4) or later. |

## A.29 IKM Hive Transform (Deprecated)

**Note:** This KM is deprecated and only used for backward compatibility.

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

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

The following table describes the options for this integration knowledge module.

**Table A-27 IKM Hive Transform Options**

| Option | Description |
|---|---|
| `CREATE_TARG_TABLE` | Creates the target table. |
| `DELETE_TEMPORARY_OBJECTS` | Removes the temporary objects, such as tables, files, and scripts post data integration. Set this option to `NO`, to retain the temporary objects, which might be useful for debugging. |

**Table A-27    (Cont.) IKM Hive Transform Options**

| Option | Description |
|---|---|
| `TRANSFORM_SCRIPT_NAME` | Defines the file name of the transformation script. This transformation script is used to transform the input data into the output structure. Both local and HDFS paths are supported, for example:<br>Local script location: `file:///tmp/odi/script1.pl`<br>HDFS script location: `hdfs://namenode:nnPort/tmp/odi/script1.pl`<br>Ensure that the following requirements are met:<br>• The path/file must be accessible by both the ODI agent and the Hive server. Read access for the Hive server is required as it is the Hive server, which executes the resulting MR job invoking the script.<br>• If `TRANSFORM_SCRIPT` is set (ODI creates the script file during mapping execution), the path/file must be writable for the ODI agent, as it is the ODI agent, which writes the script file using the HDFS Java API.<br>When the KM option `TRANSFORM_SCRIPT` is set, the following paragraphs provide some configuration help:<br>• **For HDFS script locations:**<br>The script file created is owned by the ODI agent user and receives the group of the owning directory. See *Hadoop Hdfs Permissions Guide* for more details. The standard configuration to cover the above two requirements for HDFS scripts is to ensure that the group of the HDFS script directory includes the ODI agent user (let's assume oracle) and, also the Hive server user (let's assume hive). Assuming that the group hadoop includes oracle and hive, the sample command below adjusts the ownership of the HDFS script directory:<br>`logon as hdfs user hdfs dfs -chown oracle:hadoop /tmp/odi/myscriptdir`<br>• **For local script locations:**<br>The script file created is owned by the ODI agent user and receives the ODI agent user's default group, unless SGID has been set on the script directory. If the sticky group bit has been set, the file will be owned by the group of the script directory instead. The standard configuration to cover the above two requirements for local scripts is similar to the HDFS configuration by using the SGID:<br>`chown oracle:hadoop /tmp/odi/myscriptdir chmod g+s /tmp/odi/myscriptdir` |
| `TRANSFORM_SCRIPT` | Defines the transformation script content. This transformation script is then used to transform the input data into the output structure. If left blank, the file given in `TRANSFORM_SCRIPT_NAME` must already exist. If not blank, the script file is created.<br>Script example (1-to-1 transformation): `#! /usr/bin/csh -f cat`<br>All mapped source columns are spooled as tab separated data into this script through stdin. This unix script then transforms the data and writes out the data as tab separated data on stdout. The script must provide as many output columns as there are target columns. |

**Table A-27 (Cont.) IKM Hive Transform Options**

| Option | Description |
|---|---|
| TRANSFORM_SCRIPT_MODE | Unix/HDFS file permissions for script file in octal notation with leading zero. For example, full permissions for owner and group: 0770.<br>**Warning:** Using wider permissions like 0777 poses a security risk.<br>See also KM option description for TRANSFORM_SCRIPT_NAME for details on directory permissions. |
| PRE_TRANSFORM_DISTRIBUTE | Provides an optional, comma-separated list of source column names, which enables the knowledge module to distribute the data before the transformation script is applied. |
| PRE_TRANSFORM_SORT | Provides an optional, comma-separated list of source column names, which enables the knowledge module to sort the data before the transformation script is applied. |
| POST_TRANSFORM_DISTRIBUTE | Provides an optional, comma-separated list of target column names, which enables the knowledge module to distribute the data after the transformation script is applied. |
| POST_TRANSFORM_SORT | Provides an optional, comma-separated list of target column names, which enables the knowledge module to sort the data after the transformation script is applied. |

## A.30 IKM File-Hive to Oracle (OLH-OSCH) [Deprecated]

**Note:** This KM is deprecated and only used for backward compatibility.

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

**See Also:**

- Oracle Loader for Hadoop Setup in *Oracle Big Data Connectors User's Guide* for the required versions of Hadoop and Hive.
- Configuring the Oracle Data Integrator Agent to Execute Hadoop Jobs for required environment variable settings.

The following table describes the options for this integration knowledge module.

**Table A-28 IKM File - Hive to Oracle (OLH-OSCH)**

| Option | Description |
|---|---|
| `OLH_OUTPUT_MODE` | Specifies how to load the Hadoop data into Oracle. Permitted values are JDBC, OCI, DP_COPY, DP_OSCH, and OSCH.<br>• **JDBC output mode:** The data is inserted using several direct insert JDBC connections. In very rare cases JDBC mode may result in duplicate records in target table due to Hadoop trying to restart tasks.<br>• **OCI output mode:** The data is inserted using several direct insert OCI connections in direct path mode. If `USE_ORACLE_STAGING`is set to `false`, target table must be partitioned. If `USE_ORACLE_STAGING` is set to `true`, `FLOW_TABLE_OPTIONS` must explicitly specify partitioning, for example, `"PARTITION BY HASH(COL1) PARTITIONS 4"`. In very rare cases OCI mode may result in duplicate records in target table due to Hadoop trying to restart tasks.<br>• **DP_COPY output mode:** OLH creates several DataPump export files. These files are transferred by a "`Hadoop fs -copyToLocal`" command to the local path specified by `EXT_TAB_DIR_LOCATION`. The path must be accessible by the Oracle Database engine. Once the copy job is complete, an external table is defined in the target database, which accesses the files from `EXT_TAB_DIR_LOCATION`.<br>• **DP_OSCH output mode:** OLH creates several DataPump export files. After the export phase an external table is created on the target database, which accesses these output files directly through OSCH. The path must be accessible by the Oracle Database engine. Once the copy job is complete, an external table is defined in the target database, which accesses the files from `EXT_TAB_DIR_LOCATION`.<br>• **OSCH output mode:** In OSCH mode loading, OLH is bypassed. ODI creates an external table on the target database, which accesses the input files through OSCH. Please note that only delimited and fixed length files can be read. No support for loading from Hive or custom Input Formats such as RegexInputFormat, as there is no OLH pre-processing. |
| `REJECT_LIMIT` | Specifies the maximum number of errors for Oracle Loader for Hadoop and external table. Examples: `UNLIMITED` to except all errors. Integer value (10 to allow 10 rejections) This value is used in Oracle Loader for Hadoop job definitions and, also in external table definitions. |
| `CREATE_TARG_TABLE` | Creates the target table. |
| `TRUNCATE` | Replaces the target table content with the new data. |
| `DELETE_ALL` | Deletes all the data in target table. |
| `USE_HIVE_STAGING_TABLE` | Materializes Hive source data before extraction by Oracle Loader for Hadoop. If this option is set to `false`, Oracle Loader for Hadoop directly accesses the Hive source data. Setting this option to `false` is only possible, if all these conditions are true:<br>• Only a single source table<br>• No transformations, filters, joins<br>• No datasets<br>Setting this option to `false` provides better performance by avoiding an extra data transfer step.<br>This option is applicable only if the source technology is Hive. |

**Table A-28 (Cont.) IKM File - Hive to Oracle (OLH-OSCH)**

| Option | Description |
|---|---|
| USE_ORACLE_STAGING_TABLE | Uses an intermediate Oracle database staging table.<br>The extracted data is made available to Oracle by an external table. If USE_ORACLE_STAGING_TABLE is set to true (default), the external table is created as a temporary (I$) table. This I$ table data is then inserted into the target table. Setting this option to false is only possible, if all these conditions are true:<br>• OLH_OUTPUT_MODE is set to JDBC or OCI<br>• All source columns are mapped<br>• All target columns are mapped<br>• No target-side mapping expressions<br>Setting this option to false provides better performance by avoiding an extra data transfer step, but may lead to partial data being loaded into the target table, as Oracle Loader for Hadoop loads data in multiple transactions. |
| EXT_TAB_DIR_LOCATION | Specifies the file system path of the external table. Please note the following:<br>• Only applicable, if OLH_OUTPUT_MODE = DP_*\|OSCH<br>• For OLH_OUTPUT_MODE = DP_*: this path must be accessible both from the ODI agent and from the target database engine.<br>• For OLH_OUTPUT_MODE = DP_*: the name of the external directory object is the I$ table name.<br>• For OLH_OUTPUT_MODE = DP_COPY: ODI agent will use hadoop-fs command to copy dp files into this directory.<br>• For OLH_OUTPUT_MODE = DP_*\|OSCH: this path will contain any external table log/bad/dsc files.<br>• ODI agent will remove any files from this directory during clean up before launching OLH/OSCH. |
| TEMP_DIR | Specifies the directory used for storing temporary files, such as sqoop script, stdout and stderr redirects. Leave this option blank to use system's default temp directory:<br>`<?=System.getProperty("java.io.tmp")?>` |
| MAPRED_OUTPUT_BASE_DIR | Specifies an HDFS directory, where the Oracle Loader for Hadoop job will create subdirectories for temporary files/datapump output files. |
| FLOW_TABLE_OPTIONS | Specifies the attributes for the integration table at create time and used for increasing performance. This option is set by default to NOLOGGING. This option may be left empty. |
| DELETE_TEMPORARY_OBJECTS | Removes temporary objects, such as tables, files, and scripts post data integration. Set this option to NO, to retain the temporary objects, which might be useful for debugging. |
| OVERRIDE_INPUTFORMAT | By default the InputFormat class is derived from the source Data Store/Technology (DelimitedTextInputFormat or HiveToAvroInputFormat). This option allows the user to specify the class name of a custom InputFormat. Cannot be used with OLH_OUTPUT_MODE=OSCH.<br>Example, for reading custom file formats like web log files the OLH RegexInputFormat can be used by assigning the value: oracle.hadoop.loader.lib.input.RegexInputFormat<br>See KM option EXTRA_OLH_CONF_PROPERTIES for details on how to specify the regular expression. |

**Table A-28 (Cont.) IKM File - Hive to Oracle (OLH-OSCH)**

<table>
<tr><th>Option</th><th>Description</th></tr>
<tr><td>EXTRA_OLH_CONF_PROPERTIES</td><td>Particularly when using custom InputFormats (see KM option OVERRIDE_INPUTFORMAT for details) the InputFormat may require additional configuration parameters. These are provided in the OLH configuration file. This KM option allows adding extra properties to the OLH configuration file. Cannot be used with OLH_OUTPUT_MODE=OSCH.<br>Example, (loading apache weblog file format): When OLH RegexInputFormat is used for reading custom file formats, this KM option specifies the regular expression and other parsing details:<br>&lt;property&gt; &lt;name&gt;oracle.hadoop.loader.input.regexPattern&lt;/name&gt; &lt;value&gt;([^ ]*) ([^ ]*) ([^ ]*) (-|\[[^\]]*\]) ([^ \"]*|\"[^\"]*\") (-|[0-9]*) (-|[0-9]*) (\".*?\") (\".*?\") (\".*?\")&lt;/value&gt; &lt;description&gt;RegEx for Apache WebLog format&lt;/description&gt; &lt;/property&gt;</td></tr>
</table>

## A.31 IKM File-Hive to SQL (SQOOP) [Deprecated]

**Note:** This KM is deprecated and only used for backward compatibility.

IKM File-Hive to SQL (SQOOP) supports:

- Filters, Joins, Datasets, Transformations and Aggregations in Hive
- Inline views generated by IKM Hive Control Append
- Inline views generated by IKM Hive Transform
- Hive-HBase source tables using LKM HBase to Hive (HBase SerDe)
- File source data (delimited file format only)

The following table describes the options for this integration knowledge module.

**Table A-29 IKM File-Hive to SQL (SQOOP)**

| Option | Description |
|---|---|
| CREATE_TARG_TABLE | Creates the target table. |
| TRUNCATE | Replaces the target data store content with new data. If this option is set to false, the new data is appended to the target data store. |
| DELETE_ALL | Deletes all the rows in the target data store. |
| SQOOP_PARALLELISM | Specifies the degree of parallelism. More precisely the number of mappers used during SQOOP export and therefore the number of parallel JDBC connections. |

**Table A-29 (Cont.) IKM File-Hive to SQL (SQOOP)**

| Option | Description |
|---|---|
| `USE_TARGET_STAGING_TABLE` | By default the source data is staged into a target-side staging table, before it is moved into the target table. If this option is set to `false`, SQOOP loads the source data directly into the target table, which provides better performance and less need for tablespace in target RDBMS by avoiding an extra data transfer step.<br>For File sources setting this option to `false` is only possible, if all these conditions are met:<br>• All source columns must be mapped<br>• Source and target columns have same order<br>• First file column must map to first target column<br>• no mapping gaps<br>• only 1-to-1 mappings (no expressions)<br>Please note the following:<br>• SQOOP uses multiple writers, each having their own JDBC connection to the target. Every writer uses multiple transactions for inserting the data. This means that in case `USE_TARGET_STAGING_TABLE` is set to `false`, changes to the target table are no longer atomic and writer failures can lead to partially updated target tables.<br>• The Teradata Connector for SQOOP always creates an extra staging table during load. This connector staging table is independent of the KM option. |
| `USE_GENERIC_JDBC_CONNECTOR` | Specifies whether to use the generic JDBC connector if a connector for the target technology is not available.<br>For certain technologies SQOOP provides specific connectors. These connectors take care of SQL-dialects and optimize performance. When there is a connector for the respective target technology, this connector should be used. If not, the generic JDBC connector can be used. |
| `FLOW_TABLE_OPTIONS` | When creating the target-side work table, RDBMS-specific table options can improve performance. By default this option is empty and the knowledge module will use the following table options:<br>• For Oracle: `NOLOGGING`<br>• For DB2: `NOT LOGGED INITIALLY`<br>• For Teradata: `no fallback, no before journal, no after journal`<br>Any explicit value overrides these defaults. |
| `TEMP_DIR` | Specifies the directory used for storing temporary files, such as sqoop script, stdout and stderr redirects. Leave this option blank to use system's default temp directory:<br>`<?=System.getProperty("java.io.tmp")?>` |
| `MAPRED_OUTPUT_BASE_DIR` | Specifies an HDFS directory, where SQOOP creates subdirectories for temporary files. A subdirectory called like the work table will be created here to hold the temporary data. |
| `DELETE_TEMPORARY_OBJECTS` | Deletes temporary objects such as tables, files, and scripts after data integration. Set this option to `NO`, to retain the temporary objects, which might be useful for debugging. |

**Table A-29 (Cont.) IKM File-Hive to SQL (SQOOP)**

| Option | Description |
|---|---|
| `TERADATA_PRIMARY_INDEX` | Primary index for the target table. Teradata uses the primary index to spread data across AMPs. It is important that the chosen primary index has a high cardinality (many distinct values) to ensure evenly spread data to allow maximum processing performance. Please follow Teradata's recommendation on choosing a primary index.<br>This option is applicable only to Teradata targets. |
| `TERADATA_FLOW_TABLE_TYPE` | Type of the Teradata flow table, either SET or MULTISET.<br>This option is applicable only to Teradata targets. |
| `TERADATA_OUTPUT_METHOD` | Specifies the way the Teradata Connector will load the data. Valid values are:<br>• `batch.insert`: multiple JDBC connections using batched prepared statements (simplest to start with)<br>• `multiple.fastload`: multiple FastLoad connections<br>• `internal.fastload`: single coordinated FastLoad connections (most performant)<br>This option is applicable only to Teradata targets. |
| `EXTRA_HADOOP_CONF_PROPERTIES` | Optional generic Hadoop properties. |
| `EXTRA_SQOOP_CONF_PROPERTIES` | Optional SQOOP properties. |
| `EXTRA_SQOOP_CONNECTOR_CONF_PROPERTIES` | Optional SQOOP connector properties. |

B

# Pig Knowledge Modules

This appendix provides information about the Pig knowledge modules.

This appendix includes the following sections:

- LKM File to Pig
- LKM Pig to File
- LKM HBase to Pig
- LKM Pig to HBase
- LKM Hive to Pig
- LKM Pig to Hive
- LKM SQL to Pig SQOOP
- XKM Pig Aggregate
- XKM Pig Distinct
- XKM Pig Expression
- XKM Pig Filter
- XKM Pig Flatten
- XKM Pig Join
- XKM Pig Lookup
- XKM Pig Pivot
- XKM Pig Set
- XKM Pig Sort
- XKM Pig Split
- XKM Pig Subquery Filter
- XKM Pig Table Function
- XKM Pig Unpivot

## B.1 LKM File to Pig

This KM loads data from a file into Pig.

The supported data formats are:

- Delimited
- JSON
- Pig Binary
- Text
- Avro

- Trevni
- Custom

Data can be loaded and written to local file system or HDFS.

The following table describes the options for LKM File to Pig.

**Table B-1 LKM File to Pig**

| Option | Description |
|---|---|
| Storage Function | The storage function to be used to load data.<br>Select the storage function to be used to load data. |
| Schema for Complex Fields | The pig schema for simple/complex fields separated by comma (,).<br>Redefine the datatypes of the fields in pig schema format. This option primarily allows to overwrite the default datatypes conversion for data store attributes, for example: PO_NO:int,PO_TOTAL:long MOVIE_RATING:{(RATING:double,INFO:chararray)}, where the names of the fields defined here should match with the attributes names of the data store. |
| Function Class | Fully qualified name of the class to be used as storage function to load data.<br>Specify the fully qualified name of the class to be used as storage function to load data. |
| Function Parameters | The parameters required for the custom function.<br>Specify the parameters that the loader function expects.<br>For example, the XMLLoader function may look like XMLLoader('MusicStore', 'movie', 'id:double, name:chararray, director:chararry', options)<br>Here the first three arguments are parameters, which can be specified as -rootElement MovieStore -tableName movie -schema<br>where,<br>MusicStore - the root element of the xml<br>movie - The element that wraps the child elements such as id, name, etc.<br>Third Argument is the representation of data in pig schema.<br>The names of the parameters are arbitrary and there can be any number of parameters. |
| Options | Additional options required for the storage function<br>Specify additional options required for the storage function.<br>For example, the XMLLoader function may look like XMLLoader('MusicStore', 'movie', 'id:double, name:chararray, director:chararry', options)<br>The last argument options can be specified as -namespace com.imdb -encoding utf8 |
| Jars | The jar containing the storage function class and dependent libraries separated by colon (:).<br>Specify the jar containing the storage function class and dependent libraries separated by colon (:). |
| Storage Convertor | The converter that provides functions to cast from bytearray to each of Pig's internal types.<br>Specify the converter that provides functions to cast from bytearray to each of Pig's internal types.<br>The supported converter is Utf8StorageConverter. |

## B.2 LKM Pig to File

This KM unloads data to file from pig.

The supported data formats are:

- Delimited
- JSON
- Pig Binary
- Text
- Avro
- Trevni
- Custom

Data can be stored in local file system or in HDFS.

The following table describes the options for LKM Pig to File.

**Table B-2 LKM Pig to File**

| Option | Description |
|---|---|
| Storage Function | The storage function to be used to load data.<br>Select the storage function to be used to load data. |
| Store Schema | If selected, stores the schema of the relation using a hidden JSON file. |
| Record Name | The Avro record name to be assigned to the bag of tuples being stored.<br>Specify a name to be assigned to the bag of tuples being stored. |
| Namespace | The namespace to be assigned to Avro/Trevni records, while storing data.<br>Specify a namespace for the bag of tuples being stored. |
| Delete Target File | Delete target file before Pig writes to the file.<br>If selected, the target file is deleted before storing data. This option effectively enables the target file to be overwritten. |
| Function Class | Fully qualified name of the class to be used as storage function to load data.<br>Specify the fully qualified name of the class to be used as storage function to load data. |

**Table B-2 (Cont.) LKM Pig to File**

| Option | Description |
|---|---|
| Function Parameters | The parameters required for the custom function.<br>Specify the parameters that the loader function expects.<br>For example, the XMLLoader function may look like XMLLoader('MusicStore', 'movie', 'id:double, name:chararray, director:chararry', options)<br>Here the first three arguments are parameters, which can be specified as -rootElement MovieStore -tableName movie -schema<br>where,<br>MusicStore - the root element of the xml<br>movie - The element that wraps the child elements such as id, name, etc.<br>Third Argument is the representation of data in pig schema.<br>The names of the parameters are arbitrary and there can be any number of parameters. |
| Options | Additional options required for the storage function<br>Specify additional options required for the storage function.<br>For example, the XMLLoader function may look like XMLLoader('MusicStore', 'movie', 'id:double, name:chararray, director:chararry', options)<br>The last argument options can be specified as -namespace com.imdb -encoding utf8 |
| Jars | The jar containing the storage function class and dependent libraries separated by colon (:).<br>Specify the jar containing the storage function class and dependent libraries separated by colon (:). |
| Storage Convertor | The converter that provides functions to cast from bytearray to each of Pig's internal types.<br>Specify the converter that provides functions to cast from bytearray to each of Pig's internal types.<br>The supported converter is Utf8StorageConverter. |

## B.3 LKM HBase to Pig

This KM loads data from a HBase table into Pig using HBaseStorage function.

The following table describes the options for LKM HBase to Pig.

**Table B-3 LKM HBase to Pig**

| Option | Description |
|---|---|
| Storage Function | The storage function to be used to load data.<br>HBaseStorage is used to load from a HBase table into pig. |
| Load Row Key | Load the row key as the first value in every tuple returned from HBase.<br>If selected, Loads the row key as the first value in every tuple returned from HBase. The row key is mapped to the 'key' column of the HBase data store in ODI. |

**Table B-3 (Cont.) LKM HBase to Pig**

| Option | Description |
|---|---|
| Greater Than Min Key | Loads rows with key greater than the key specified for this option.<br>Specify the key value to load rows with key greater than the specified key value. |
| Less Than Min Key | Loads rows with row key less than the value specified for this option.<br>Specify the key value to load rows with key less than the specified key value. |
| Greater Than Or Equal Min Key | Loads rows with key greater than or equal to the key specified for this option.<br>Specify the key value to load rows with key greater than or equal to the specified key value. |
| Less Than Or Equal Min Key | Loads rows with row key less than or equal to the value specified for this option.<br>Specify the key value to load rows with key less than or equal to the specified key value. |
| Limit Rows | Maximum number of row to retrieve per region<br>Specify the maximum number of rows to retrieve per region. |
| Cached Rows | Number of rows to cache.<br>Specify the number of rows to cache. |
| Storage Convertor | The name of Caster to use to convert values.<br>Specify the class name of Caster to use to convert values. The supported values are HBaseBinaryConverter and Utf8StorageConverter. If unspecified, the default value is Utf8StorageConverter. |
| Column Delimiter | The delimiter to be used to separate columns in the columns list of HBaseStorage function.<br>Specify the delimiter to be used to separate columns in the columns list of HBaseStorage function. If unspecified, the default is whitespace. |
| Timestamp | Return cell values that have a creation timestamp equal to this value.<br>Specify a timestamp to return cell values that have a creation timestamp equal to the specified value. |
| Min Timestamp | Return cell values that have a creation timestamp less than to this value.<br>Specify a timestamp to return cell values that have a creation timestamp less than to the specified value. |
| Max Timestamp | Return cell values that have a creation timestamp less than this value.<br>Specify a timestamp to return cell values that have a creation timestamp greater than or equal to the specified value. |

## B.4 LKM Pig to HBase

This KM stores data into a HBase table using HBaseStorage function.

The following table describes the options for LKM Pig to HBase.

**Table B-4 LKM Pig to HBase**

| Option | Description |
|---|---|
| Storage Function | The storage function to be used to store data. This is a read-only option, which cannot be changed.<br>HBaseStore function is used to load data into HBase table. |
| Storage Convertor | The name of Caster to use to convert values.<br>Specify the class name of Caster to use to convert values. The supported values are HBaseBinaryConverter and Utf8StorageConverter. If unspecified, the default value is Utf8StorageConverter. |
| Column Delimiter | The delimiter to be used to separate columns in the columns list of HBaseStorage function.<br>Specify the delimiter to be used to separate columns in the columns list of HBaseStorage function. If unspecified, the default is whitespace. |
| Disable Write Ahead Log | If it is true, write ahead log is set to false for faster loading into HBase.<br>If selected, write ahead log is set to false for faster loading into HBase. This must be used in extreme caution, since this could result in data loss. Default value is false. |

## B.5 LKM Hive to Pig

This KM loads data from a Hive table into Pig using HCatalog.

The following table describes the options for LKM Hive to Pig.

**Table B-5 LKM Hive to Pig**

| Option | Description |
|---|---|
| Storage Function | The storage function to be used to load data. This is a read-only option, which cannot be changed.<br>HCatLoader is used to load data from a hive table. |

## B.6 LKM Pig to Hive

This KM stores data into a hive table using HCatalog.

The following table describes the options for LKM Pig to Hive.

**Table B-6 LKM Pig to Hive**

| Option | Description |
|---|---|
| Storage Function | The storage function to be used to load data. This is a read-only option, which cannot be changed.<br>HCatStorer is used to store data into a hive table. |

**Table B-6 (Cont.) LKM Pig to Hive**

| Option | Description |
|---|---|
| Partition | The new partition to be created.<br>Represents key/value pairs for partition. This is a mandatory argument when you are writing to a partitioned table and the partition column is not in the output column. The values for partition keys should NOT be quoted. |

## B.7 LKM SQL to Pig SQOOP

This KM integrates data from a JDBC data source into Pig.

It executes the following steps:

1. Create a SQOOP configuration file, which contains the upstream query.
2. Execute SQOOP to extract the source data and import into Staging file in csv format.
3. Runs LKM File To Pig KM to load the Staging file into PIG.
4. Drop the Staging file.

The following table describes the options for LKM SQL to Pig SQOOP.

**Table B-7 LKM SQL to Pig SQOOP**

| Option | Description |
|---|---|
| STAGING_FILE_DELIMITER | Sqoop uses this delimiter to create the temporary file. If not specified, \ \t will be used. |
| Storage Function | The storage function to be used to load data.<br>Select the storage function to be used to load data. |
| Schema for Complex Fields | The pig schema for simple/complex fields separated by comma (,).<br>Redefine the datatypes of the fields in pig schema format. This option primarily allows to overwrite the default datatypes conversion for data store attributes, for example: PO_NO:int,PO_TOTAL:long MOVIE_RATING:{(RATING:double,INFO:chararray)}, where the names of the fields defined here should match with the attributes names of the data store. |
| Function Class | Fully qualified name of the class to be used as storage function to load data.<br>Specify the fully qualified name of the class to be used as storage function to load data. |

**Table B-7 (Cont.) LKM SQL to Pig SQOOP**

| Option | Description |
|---|---|
| Function Parameters | The parameters required for the custom function.<br>Specify the parameters that the loader function expects.<br>For example, the XMLLoader function may look like XMLLoader('MusicStore', 'movie', 'id:double, name:chararray, director:chararry', options)<br>Here the first three arguments are parameters, which can be specified as -rootElement MovieStore -tableName movie -schema<br>where,<br>MusicStore - the root element of the xml<br>movie - The element that wraps the child elements such as id, name, etc.<br>Third Argument is the representation of data in pig schema.<br>The names of the parameters are arbitrary and there can be any number of parameters. |
| Options | Additional options required for the storage function.<br>Specify additional options required for the storage function.<br>For example, the XMLLoader function may look like XMLLoader('MusicStore', 'movie', 'id:double, name:chararray, director:chararry', options)<br>The last argument options can be specified as -namespace com.imdb -encoding utf8 |
| Jars | The jar containing the storage function class and dependent libraries separated by colon (:).<br>Specify the jar containing the storage function class and dependent libraries separated by colon (:). |
| Storage Convertor | The converter that provides functions to cast from bytearray to each of Pig's internal types.<br>Specify the converter that provides functions to cast from bytearray to each of Pig's internal types.<br>The supported converter is Utf8StorageConverter. |

## B.8 XKM Pig Aggregate

Summarize rows, for example using SUM and GROUP BY.

The following table describes the options for XKM Pig Aggregate.

**Table B-8 XKM Pig Aggregate**

| Option | Description |
|---|---|
| USING_ALGORITHM | Aggregation type; collected or merge. |
| PARTITION_BY | Specify the Hadoop partitioner. |
| PARTITIONER_JAR | Increase the parallelism of this job. |
| PARALLEL_NUMBER | Increase the parallelism of this job. |

**Note:**

When mapping has Pig staging, i.e when processing is done with Pig, and there is aggregator component in the Pig staging area, the clause must be set differently than in regular mappings for SQL based technologies.

## B.9 XKM Pig Distinct

Eliminates duplicates in data.

## B.10 XKM Pig Expression

Define expressions to be reused across a single mapping.

## B.11 XKM Pig Filter

Produce a subset of data by a filter condition.

## B.12 XKM Pig Flatten

Un-nest the complex data according to the given options.

The following table describes the options for XKM Pig Flatten.

**Table B-9 XKM Pig Flatten**

| Option | Description |
|---|---|
| Default Expression | Default expression for null nested table objects, for example, rating_table(obj_rating('-1', 'Unknown')).<br>This is used to return a row with default values for each null nested table object. |

## B.13 XKM Pig Join

Joins more than one input sources based on the join condition.

The following table describes the options for XKM Pig Join.

**Table B-10 XKM Pig Join**

| Option | Description |
|---|---|
| USING_ALGORITHM | Join type; replicated or skewed or merge. |
| PARTITION_BY | Specify the Hadoop partitioner. |
| PARTITIONER_JAR | Increase the parallelism of this job. |
| PARALLEL_NUMBER | Increase the parallelism of this job. |

## B.14 XKM Pig Lookup

Lookup data for a driving data source.

The following table describes the options for XKM Pig Lookup.

**Table B-11    XKM Pig Lookup**

| Option | Description |
|---|---|
| Jars | The jar containing the Used Defined Function classes and dependant libraries separated by colon (:). |

## B.15 XKM Pig Pivot

Takes data in separate rows, aggregates it, and converts it into columns.

## B.16 XKM Pig Set

Perform UNION, MINUS or other set operations.

## B.17 XKM Pig Sort

Sort data using an expression.

## B.18 XKM Pig Split

Split data into multiple paths with multiple conditions.

## B.19 XKM Pig Subquery Filter

Filter rows based on the results of a subquery.

## B.20 XKM Pig Table Function

Pig table function access.

The following table descriptions the options for XKM Pig Table Function.

**Table B-12    XKM Pig Table Function**

| Option | Description |
|---|---|
| PIG_SCRIPT_CONTENT | User specified pig script content. |

## B.21 XKM Pig Unpivot

Transform a single row of attributes into multiple rows in an efficient manner.

# C
# Spark Knowledge Modules

This appendix provides information about the Spark knowledge modules.

This appendix includes the following sections:

- LKM File to Spark
- LKM Spark to File
- LKM Hive to Spark
- LKM Spark to Hive
- LKM HDFS to Spark
- LKM Spark to HDFS
- LKM Kafka to Spark
- LKM Spark to Kafka
- LKM SQL to Spark
- LKM Spark to SQL
- LKM Spark to Cassandra
- RKM Cassandra
- XKM Spark Aggregate
- XKM Spark Distinct
- XKM Spark Expression
- XKM Spark Filter
- XKM Spark Flatten
- XKM Spark Input Signature and Output Signature
- XKM Spark Join
- XKM Spark Lookup
- XKM Spark Pivot
- XKM Spark Set
- XKM Spark Sort
- XKM Spark Split
- XKM Spark Table Function
- IKM Spark Table Function
- XKM Spark Unpivot

## C.1 LKM File to Spark

This KM will load data from a file into a Spark Python variable and can be defined on the AP between the execution units, source technology File, target technology Spark Python.

> **Note:**
>
> This KM also supports loading HDFS files, although it's preferable to use LKM HDFS to Spark for that purpose.

The following tables describe the options for LKM File to Spark.

**Table C-1 LKM File to Spark**

| Option | Description |
|---|---|
| Storage Function | The storage function to be used to load/store data.<br>Choose one of the following storage functions to load data:<br>• `textFile` is used to load data from HDFS, a local file system or any Hadoop-supported file system URI.<br>• `jsonFile` is used to load data from a JSON file where each line of the files is a JSON object.<br>• `newAPIHadoopFile` is used to load data from a Hadoop file with an arbitrary new API InputFormat.<br>• `newAPIHadoopRDD` is used to load data from Hadoop-readable dataset with an arbitrary new API InputFormat.<br>• `hadoopFile` is used to load data from a Hadoop file with an arbitrary InputFormat.<br>• `hadoopRDD` is used to load data from a Hadoop-readable dataset.<br>• `sequenceFile` is used to load data from an RDD of key-value pairs to Spark. |
| streamingContext | Name of Streaming Context. |
| InputFormatClass | Class for reading the format of input data.<br>For example,<br>• org.apache.hadoop.mapreduce.lib.input.TextInputFormat (default)<br>• org.apache.hadoop.mapred.TextInputFormat (hadoopFile and hadoopRDD) |
| KeyClass | Fully qualified classname for keys.<br>For example,<br>• org.apache.hadoop.io.LongWritable (default)<br>• org.apache.hadoop.io.Text |
| ValueClass | Fully qualified classname for values.<br>For example,<br>• org.apache.hadoop.io.Text (default)<br>• org.apache.hadoop.io.LongWritable |
| KeyConverter | Fully qualified classname of key converter class. |
| ValueConverter | Fully qualified classname of value converter class. |

**Table C-1 (Cont.) LKM File to Spark**

| Option | Description |
|---|---|
| Job Configuration | Hadoop configuration.<br>For example, {'hbase.zookeeper.quorum': 'HOST', 'hbase.mapreduce.inputtable': 'TAB'} |
| inferSchema | Infer DataFrame schema from data.<br>If set to True (default), the column names and types will be inferred from source data and DataFrame will be created with default options.<br>If set to False, the DataFrame schema will be specified based on the source data store definition. |
| dateFormat | Format for Date or Timestamp input fields. |
| Delete Spark Mapping Files | Delete temporary objects at end of mapping. |
| Cache | Cache RDD/DataFrame across operations after computation. |
| Storage Level | The storage level to be used to cache data. |
| Repartition | Repartition the RDD/DataFrame after transformation of this component. |
| Level of Parallelism | Number of partitions. |
| Sort Partitions | Sort partitions by a key function when repartitioning. |
| Partition Sort Order | Sort partitions order. |
| Partition Keys | Define keys for partitions. |
| Partition Function | Customized partitioning function. |

This LKM uses StreamingContext.textFileStream() method to transfer file context as data stream. The directory is monitored while the Spark application is running. Any files copied from other locations into this directory is detected.

**Table C-2 LKM File to Spark for Streaming**

| Option | Description |
|---|---|
| Storage Function | If STREAMING_MODE is set to true, the load function is changed to textFileStream automatically.<br>Default is textFile. |
| Source Data store | Source data store is a directory and field separator should be defined. |

# C.2 LKM Spark to File

This KM will store data into a file from a Spark Python variable and can be defined on the AP between the execution units, source technology Spark Python, target technology File.

> **Note:**
>
> This KM also supports writing to an HDFS File, although the LKM Spark to HDFS is preferable.

The following tables describes the options for LKM Spark to File.

**Table C-3    LKM Spark to File**

| Option | Description |
|---|---|
| Storage Function | Storage function to be used to load/store data.<br>Choose one of the following storage functions to store data:<br>• `saveAsTextFile` is used to store the data into HDFS, a local file system or any Hadoop-supported file system URI.<br>• `saveAsJsonFile` is used to store the data in JSON format into HDFS, a local file system or any Hadoop-supported file system URI.<br>• `saveAsNewAPIHadoopFile` is used to store the data to a Hadoop file with an arbitrary new API InputFormat.<br>• `saveAsHadoopFile` is used to store data to a Hadoop file with an arbitrary InputFormat.<br>• `saveAsSequenceFile` is used to store data into key-value pairs.<br>**Note:**<br>When spark.useDataFrames is set to True, the data will be saved as RDD of JSON strings for `saveAsNewAPIHadoopFile`, `saveAsHadoopFile`, and `saveAsSequenceFile`. |
| OutputFormatClass | Fully qualified classname for writing the data.<br>For example,<br>• org.apache.hadoop.mapreduce.lib.input.TextOutputFormat (default)<br>• org.apache.hadoop.mapred.TextOutputFormat (`saveAsHadoopFile`) |
| KeyClass | Fully qualified class for keys.<br>For example,<br>• org.apache.hadoop.io.NullWritable (default)<br>• org.apache.hadoop.io.Text |
| ValueClass | Fully qualified class for values.<br>For example,<br>• org.apache.hadoop.io.NullWritable<br>• org.apache.hadoop.io.Text (default) |
| KeyConverter | Fully qualified classname of key converter class. |
| ValueConverter | Fully qualified classname of value converter class. |
| Job Configuration | Allows adding or overriding Hadoop configuration properties.<br>For example, {'hbase.zookeeper.quorum': 'HOST', 'hbase.mapreduce.inputtable': 'TAB'} |
| SQL_EXPRESSIONS | Use SQL Expressions. |
| Delete Spark Mapping Files | Delete temporary objects at end of mapping. |
| Cache | Cache RDD/DataFrame across operations after computation. |

**Table C-3 (Cont.) LKM Spark to File**

| Option | Description |
|---|---|
| Storage Level | The storage level to be used to cache data. |
| Repartition | Repartition the RDD/DataFrame after transformation of this component. |
| Level of Parallelism | Number of partitions. |
| Sort Partitions | Sort partitions by a key function when repartitioning. |
| Partition Sort Order | Sort partitions order. |
| Partition Keys | Define keys for partitions. |
| Partition Function | Customized partitioning function. |

**Table C-4 LKM Spark to File for streaming**

| Option | Description |
|---|---|
| Storage Function | If STREAMING_MODE is set to true, the load function is changed to textFileStream automatically.<br>Default is textFile. |

## C.3 LKM Hive to Spark

This KM will load data from a Hive table into a Spark Python variable and can be defined on the AP between the execution units, source technology Hive, target technology Spark Python.

The following table describes the options for LKM Hive to Spark:

**Table C-5 LKM Hive to Spark**

| Option | Description |
|---|---|
| Delete Spark Mapping Files | Delete temporary objects at end of mapping. |
| Cache | Cache RDD/DataFrame across operations after computation. |
| Storage Level | The storage level to be used to cache data. |
| Repartition | Repartition the RDD/DataFrame after transformation of this component. |
| Level of Parallelism | Number of partitions. |
| Sort Partitions | Sort partitions by a key function when repartitioning. |
| Partition Sort Order | Sort partitions order. |
| Partition Keys | Define keys for partitions. |
| Partition Function | Customized partitioning function. |

## C.4 LKM Spark to Hive

This KM will store data into a Hive table from a Spark Python variable and can be defined on the AP between the execution units, source technology Spark, target technology Hive.

The following table describes the options for LKM Spark to Hive.

**Table C-6 LKM Spark to Hive**

| Option | Description |
|---|---|
| OVERWRITE_TARGET_TABLE | Overwrite the target table. |
| INFER_SCHEMA | Infer target DataFrame schema from RDD data.<br>**Note:**<br>This option is set to True by default. When set to True, the column names and types will be inferred from RDD data and DataFrame will be created with default options. If it is set to False, DataFrame schema will be specified based on target datastore definition. Set this option to False if, you are getting errors such as : `ValueError: Some types cannot be determined by the first X rows, please try again with sampling.` This usually happens if, one or more target columns receive NULL values. When using False there might be execution errors if source column datatype is different from target column datatype. In such a case it is recommended to add conversion function to attribute mapping. |
| SAMPLING_RATIO | The sample ratio of rows used for inferring. |
| SQL_EXPRESSIONS | Use SQL Expressions. |
| Delete Spark Mapping Files | Delete temporary objects at end of mapping. |
| Cache | Cache RDD/DataFrame across operations after computation. |
| Storage Level | The storage level to be used to cache data. |
| Repartition | Repartition the RDD/DataFrame after transformation of this component. |
| Level of Parallelism | Number of partitions. |
| Sort Partitions | Sort partitions by a key function when repartitioning. |
| Partition Sort Order | Sort partitions order. |
| Partition Keys | Define keys for partitions. |
| Partition Function | Customized partitioning function. |

## C.5 LKM HDFS to Spark

This KM will load data from HDFS file to Spark.

**Table C-7 LKM HDFS to Spark**

| Option | Description |
|---|---|
| streamingContext | Name of Streaming Context. |
| inferSchema | Infer DataFrame schema from data. |
| Delete Spark Mapping Files | Delete temporary objects at end of mapping. |
| Cache | Cache RDD/DataFrame across operations after computation. |
| Storage Level | The storage level to be used to cache data. |
| Repartition | Repartition the RDD/DataFrame after transformation of this component. |
| Level of Parallelism | Number of partitions. |
| Sort Partitions | Sort partitions by a key function when repartitioning. |
| Partition Sort Order | Sort partitions order. |
| Partition Keys | Define keys for partitions. |
| Partition Function | Customized partitioning function. |

> **Note:**
>
> Streaming is enabled when the streaming check box is selected in the physical schema. Streaming is only supported for the Delimited and JSON formats.

## C.6 LKM Spark to HDFS

This KM will load data from Spark to HDFS file.

**Table C-8 LKM Spark to HDFS**

| Option | Description |
|---|---|
| SQL_EXPRESSIONS | Use SQL Expressions. |
| Delete Spark Mapping Files | Delete temporary objects at end of mapping. |
| Cache | Cache RDD/DataFrame across operations after computation. |
| Storage Level | The storage level to be used to cache data. |
| Repartition | Repartition the RDD/DataFrame after transformation of this component. |
| Level of Parallelism | Number of partitions. |
| Sort Partitions | Sort partitions by a key function when repartitioning. |
| Partition Sort Order | Sort partitions order. |
| Partition Keys | Define keys for partitions. |
| Partition Function | Customized partitioning function. |

> **Note:**
>
> Streaming is enabled when the streaming check box is selected in the physical schema. Streaming is supported for all formats.

## C.7 LKM Kafka to Spark

This KM will load data with Kafka source and Spark target and can be defined on the AP node that exist in Spark execution unit and have Kafka upstream node.

**Table C-9 LKM Kafka to Spark for streaming**

| Option | Description |
|---|---|
| Storage Function | The storage function to be used to load data. |
| fromOffsets | Per-topic/partition Kafka offsets defining the (inclusive) starting point of the stream. |
| KeyDecoder | Converts message key. |
| ValueDecoder | Converts message value. |
| groupId | The group id for this consumer. |
| storageLevel | RDD Storage level. |
| numPartitions | Number of partitions for each consumer. |
| offsetRanges | List of offsetRange to specify topic:partition:[start, end) to consume. |
| leaders | Kafka brokers for each TopicAndPartition in offsetRanges. |
| messageHandler | A function used to convert KafkaMessageAndMetadata. |
| avroSchema | avroSchema have the content of .avsc file. This file is associated with .avro Data file. |
| Delete Spark Mapping Files | Delete temporary objects at end of mapping. |
| Cache | Cache RDD/DataFrame across operations after computation. |
| Storage Level | The storage level to be used to cache data. |
| Repartition | Repartition the RDD/DataFrame after transformation of this component. |
| Level of Parallelism | Number of partitions. |
| Sort Partitions | Sort partitions by a key function when repartitioning. |
| Partition Sort Order | Sort partitions order. |
| Partition Keys | Define keys for partitions. |
| Partition Function | Customized partitioning function. |

## C.8 LKM Spark to Kafka

LKM Spark to Kafka works in both streaming and batch mode and can be defined on the AP between the execution units and have Kafka downstream node.

**Table C-10 LKM Spark to Kafka**

| Option | Description |
|---|---|
| avroSchema | Has the content of .avsc file. This file is associated with .avro Data file. |
| Delete Spark Mapping Files | Delete temporary objects at end of mapping. |
| Cache | Cache RDD/DataFrame across operations after computation. |
| Storage Level | The storage level to be used to cache data. |
| Repartition | Repartition the RDD/DataFrame after transformation of this component. |
| Level of Parallelism | Number of partitions. |
| Sort Partitions | Sort partitions by a key function when repartitioning. |
| Partition Sort Order | Sort partitions order. |
| Partition Keys | Define keys for partitions. |
| Partition Function | Customized partitioning function. |

## C.9 LKM SQL to Spark

This KM is designed to load data from Cassandra into Spark, but it can work with other JDBC sources. It can be defined on the AP node that have SQL source and Spark target.

To use this KM, it is mandatory to configure the Hadoop Credential Provider and define the password. For more information, see Password Handling in Hadoop.

**Table C-11 LKM SQL to Spark**

| Option | Description |
|---|---|
| PARTITION_COLUMN | Column used for partitioning. |
| LOWER_BOUND | Lower bound of the partition column. |
| UPPER_BOUND | Upper bound of the partition column. |
| NUMBER_PARTITIONS | Number of partitions. |
| PREDICATES | List of predicates. |
| Delete Spark Mapping Files | Delete temporary objects at end of mapping. |

Table C-11 (Cont.) LKM SQL to Spark

| Option | Description |
|---|---|
| Cache | Cache RDD/DataFrame across operations after computation. |
| Storage Level | The storage level to be used to cache data. |
| Repartition | Repartition the RDD/DataFrame after transformation of this component. |
| Level of Parallelism | Number of partitions. |
| Sort Partitions | Sort partitions by a key function when repartitioning. |
| Partition Sort Order | Sort partitions order. |
| Partition Keys | Define keys for partitions. |
| Partition Function | Customized partitioning function. |

## C.10 LKM Spark to SQL

This KM will load data from Spark into JDBC targets and can be defined on the AP node that have Spark source and SQL target.

To use this KM, it is mandatory to the configure the Hadoop Credential Provider and define the password. For more information, see Password Handling in Hadoop.

Table C-12 LKM Spark to SQL

| Option | Description |
|---|---|
| CREATE_TARG_TABLE | Create target table. |
| TRUNCATE_TARG_TABLE | Truncate target table. |
| DELETE_TARG_TABLE | Delete target table. |

**Table C-12 (Cont.) LKM Spark to SQL**

| Option | Description |
|---|---|
| INFER_SCHEMA | Infer target DataFrame schema from RDD data.<br>**Note:**<br>This option is set to True by default. When set to True, the column names and types will be inferred from RDD data and DataFrame will be created with default options. If it is set to False, DataFrame schema will be specified based on target datastore definition. Set this option to False if you are getting errors such as : `ValueError: Some types cannot be determined by the first X rows, please try again with sampling.` This usually happens if, one or more target columns receive NULL values. When using False there might be execution errors if source column datatype is different from target column datatype. In such a case it is recommended to add conversion function to attribute mapping. |
| SAMPLING_RATIO | The sample ratio of rows used for inferring. |
| SQL_EXPRESSIONS | Use SQL Expressions. |
| Delete Spark Mapping Files | Delete temporary objects at end of mapping. |
| Cache | Cache RDD/DataFrame across operations after computation. |
| Storage Level | The storage level is used to cache data. |
| Repartition | Repartition the RDD/DataFrame after transformation of this component. |
| Level of Parallelism | Number of partitions. |
| Sort Partitions | Sort partitions by a key function when you repartition RDD/DataFrame. |
| Partition Sort Order | Sort partition order. |
| Partition Keys | Define keys of partition. |
| Partition Function | Customized partitioning function. |

## C.11 LKM Spark to Cassandra

To use this KM, it is mandatory to configure the Hadoop Credential Provider and define the password. For more information, see Password Handling in Hadoop.

**Table C-13 LKM Spark to Cassandra**

| Option | Description |
|---|---|
| CREATE_TARG_TABLE | Create target table. |

**Table C-13 (Cont.) LKM Spark to Cassandra**

| Option | Description |
|---|---|
| TRUNCATE_TARG_TABLE | Truncate target table. |
| DELETE_TARG_TABLE | Delete target table. |
| INFER_SCHEMA | Infer target DataFrame schema from RDD data. |
| SAMPLING_RATIO | The sample ratio of rows used for inferring. |
| SQL_EXPRESSIONS | Use SQL Expressions. |
| Delete Spark Mapping Files | Delete temporary objects at end of mapping. |
| Cache | Cache RDD/DataFrame across operations after computation. |
| Storage Level | The storage level to be used to cache data. |
| Repartition | Repartition the RDD/DataFrame after transformation of this component. |
| Level of Parallelism | Number of partitions. |
| Sort Partitions | Sort partitions by a key function when repartitioning. |
| Partition Sort Order | Sort partitions order. |
| Partition Keys | Define keys for partitions. |
| Partition Function | Customized partitioning function. |

# C.12 RKM Cassandra

RKM Cassandra reverses these metadata elements:

- Cassandra tables as data stores.

  The Mask field in the Reverse Engineer tab filters reverse-engineered objects based on their names. The Mask field cannot be empty and must contain at least the percent sign (%).
- Cassandra columns as attributes with their data types.

# C.13 XKM Spark Aggregate

Summarize rows, for example, using SUM and GROUP BY.

The following tables describes the options for XKM Spark Aggregate.

**Table C-14 XKM Spark Aggregate**

| Option | Description |
|---|---|
| CACHE_DATA | Persist the data with the default storage level. |
| NUMBER_OF_TASKS | Task number. |

**Table C-15 XKM Spark Aggregate for streaming**

| Option | Description |
|---|---|
| WINDOW_AGGREGATION | Enable window aggregation. |
| WINDOW_LENGTH | Number of batch intervals. |
| SLIDING_INTERVAL | The interval at which the window operation is performed. |
| STATEFUL_AGGREGATION | Enables stateful aggregation. |
| STATE_RETENTION_PERIOD | Time in seconds to retain a key or value aggregate in the Spark state object. |
| FORWARD_ONLY_UPDATED_ROWS | Modified aggregate values forwarded to downstream components. |

## C.14 XKM Spark Distinct

Eliminates duplicates in data and functionality is identical to the existing batch processing.

## C.15 XKM Spark Expression

Define expressions to be reused across a single mapping.

## C.16 XKM Spark Filter

Produce a subset of data by a filter condition.

The following tables describes the options for XKM Spark Filter.

**Table C-16 XKM Spark Filter**

| Option | Description |
|---|---|
| CACHE_DATA | Persist the data with the default storage level. |

## C.17 XKM Spark Input Signature and Output Signature

Supports code generation for reusable mapping.

## C.18 XKM Spark Join

Joins more than one input sources based on the join condition.

The following tables describes the options for XKM Spark Join.

**Table C-17 XKM Spark Join**

| Option | Description |
|---|---|
| CACHE_DATA | Persist the data with the default storage level. |
| NUMBER_OF_TASKS | Task number. |

# C.19 XKM Spark Lookup

Lookup data for a driving data source.

The following tables describes the options for XKM Spark Lookup.

**Table C-18 XKM Spark Lookup**

| Option | Description |
|---|---|
| CACHE_DATA | Persist the data with the default storage level. |
| NUMBER_OF_TASKS | Task number. |
| MAP_SIDE | Defines whether the KM will do a map-side lookup or a reduce-side lookup and significantly impacts lookup performance. |
| KEY_BASED_LOOKUP | Only data corresponding to the lookup keys are retrieved. |

**Table C-19 XKM Spark Lookup for streaming**

| Option | Description |
|---|---|
| MAP_SIDE | MAP_SIDE=true : Suitable for small lookup data sets fitting into memory. This setting provides better performance by broadcasting the lookup data to all Spark tasks. |
| KEY_BASED_LOOKUP | For any incoming lookup key a Spark cache is checked.<br>• If the lookup record is present and not expired, the lookup data is served from the cache.<br>• If the lookup record is missing or expired, the data is re-loaded from the SQL source. |
| CACHE_RELOAD | This option defines when the lookup source data is loaded and refreshed and here are the corresponding values:<br>• NO_RELOAD: The lookup source data is loaded once on Spark application startup.<br>• RELOAD_EVERY_BATCH: The lookup source data is reloaded for every new Spark batch.<br>• RELOAD_BASE_ON_TIME: The lookup source data is loaded on Spark application startup and refreshed after the time interval provided by KM option CacheReloadInterval. |
| CACHE_RELOAD_INTERVAL | Defines the time data to be retained in the Spark cache. After this time the expired data or records are removed from cache. |

# C.20 XKM Spark Pivot

Take data in separate rows, aggregates it and converts it into columns.

The following tables describes the options for XKM Spark Pivot.

**Table C-20 XKM Spark Pivot**

| Option | Description |
|---|---|
| CACHE_DATA | Persist the data with the default storage level. |

**Note:**

XKM Spark Pivot does not support streaming.

## C.21 XKM Spark Set

Perform UNION, MINUS or other set operations.

## C.22 XKM Spark Sort

Sort data using an expression.

The following tables describes the options for XKM Spark Sort.

**Table C-21 XKM Spark Sort**

| Option | Description |
|---|---|
| CACHE_DATA | Persist the data with the default storage level. |
| NUMBER_OF_TASKS | Task number. |

## C.23 XKM Spark Split

Split data into multiple paths with multiple conditions.

The following tables describes the options for XKM Spark Split.

**Table C-22 XKM Spark Split**

| Option | Description |
|---|---|
| CACHE_DATA | Persist the data with the default storage level. |

## C.24 XKM Spark Table Function

This KM allows applying custom transformation by executing arbitrary Spark/Python transformations as part of the overall Spark Python script.

The following table describes the options for XKM Spark Table Function.

**Table C-23 XKM Spark Table Function**

| Option | Description |
|---|---|
| SPARK_SCRIPT | User specifies the customized code content. |
| SPARK_SCRIPT_FILE | User specifies the path of spark script file. |
| CACHE_DATA | Persist the data with the default storage level. |

> **Note:**
>
> Only one of the options, either SPARK_SCRIPT or SPARK_SCRIPT_FILE must be set.
>
> - If SPARK_SCRIPT_FILE is set, then the specified file will be dynamically executed.
> - If SPARK_SCRIPT is set, its content will be inserted into the main Spark script.
> - If neither SPARK_SCRIPT nor SPARK_SCRIPT_FILE is set, a validation error is generated stating that at least one of the options must be specified.
> - If both SPARK_SCRIPT and SPARK_SCRIPT_FILE are set, a validation error is generated stating that only one of the options must be specified.

# C.25 IKM Spark Table Function

Spark table function as target.

The following tables describes the options for IKM Spark Table Function.

**Table C-24 IKM Spark Table Function**

| Option | Description |
|---|---|
| SPARK_SCRIPT_FILE | User specifies the path of spark script file. |
| CACHE_DATA | Persist the data with the default storage level. |

# C.26 XKM Spark Unpivot

Transform a single row of attributes into multiple rows in an efficient manner.

The following tables describes the options for XKM Spark Pivot.

**Table C-25 XKM Spark Unpivot**

| Option | Description |
|---|---|
| CACHE_DATA | Persist the data with the default storage level. |

**Note:**

XKM Spark Unpivot does not support streaming.

D

# Component Knowledge Modules

This appendix provides information about the knowledge modules for the Flatten and the Jagged component.

This appendix includes the following sections:

- XKM Oracle Flatten
- XKM Oracle Flatten XML
- XKM Spark Flatten
- XKM Jagged

## D.1 XKM Oracle Flatten

Un-nest the complex data according to the given options.

> **Note:**
>
> Flatten component is supported only with Spark 1.3.

The following tables describes the options for XKM Oracle Flatten.

**Table D-1 XKM Oracle Flatten**

| Option | Description |
|---|---|
| NESTED_TABLE_ALIAS | Alias used for nested table expression.<br>Default is NST. |
| DEFAULT_EXPRESSION | Default expression for null nested table objects. For example, rating_table(obj_rating('-1', 'Unknown')). |

## D.2 XKM Oracle Flatten XML

Un-nest the complex data in an XML file according to the given options.

The following tables describes the options for XKM Oracle Flatten XML.

**Table D-2 XKM Oracle Flatten XML**

| Option | Description |
|---|---|
| XML_XPATH | Specify XML path for XMLTABLE function. For example, '/ratings/rating'. |
| XML_IS_ATTRIBUTE | Set to True when data is stored as attribute values of record tag. For example, <row attribute1=..." /> " |

**Table D-2 (Cont.) XKM Oracle Flatten XML**

| Option | Description |
|---|---|
| XML_TABLE_ALIAS | Alias used for XMLTABLE expression.<br>Default is XMLT. |
| DEFAULT_EXPRESSION | Default expression for null XMLTYPE objects. For example, <row> < attribute1/><row/><br>This is used to return a row with default values for each null XMLTYPE object. |

# D.3 XKM Spark Flatten

Un-nest the complex data according to the given options.

The following tables describes the options for XKM Spark Flatten.

**Table D-3 XKM Spark Flatten**

| Option | Description |
|---|---|
| Default Expression | Default expression for null nested table objects. For example, rating_table(obj_rating('-1', 'Unknown')).<br>This is used to return a row with default values for each null nested table object. |
| CACHE_DATA | When set to TRUE, persist the results with Spark default storage level.<br>Default is FALSE. |

# D.4 XKM Jagged

Jagged component KMs process unstructured data using meta pivoting. Source data, represented as key-value free format, will be transformed into more structured entities in order to be loaded into database tables or file structures. Jagged component has one input group and one or multiple output groups based on the configuration of the component. Input group is connected to a source component, which has e key-value or id-key-value structure. Output groups are connected to the target components where data is stored in more structured way, that is, keys become column names and values are stored as table rows. Jagged KM is parsing the source data and is looking for key data matching the output group attributes. Once the relevant keys are identified the corresponding data is stored into a row. In case of key-value source each incoming record is delimited by a key marked as End of Data Indicator. In case of id-key-value source incoming records are delimited by a new value of the sequence defined as id. Target records can be consolidated by removing duplicates based on Unique Index attribute property. Some attributes can be labeled as required, meaning no new record is stored if any of the required keys is missing. Default values can be defined for some missing keys.

The following tables describes the options for XKM Jagged.

**Table D-4 XKM Jagged**

| Option | Description |
|---|---|
| TMP_DIR | Directory for temporary files. |

**Table D-4 (Cont.) XKM Jagged**

| Option | Description |
|---|---|
| FIELD_DELIMITER | Field delimiter for temporary files. |
| DELETE_TEMPORARY_OBJECTS | Delete temporary objects at end of mapping. |

E

# Considerations, Limitations, and Issues

This appendix lists the considerations, limitations, and issues that you must be aware of while working on Big Data integration projects in ODI.
This appendix includes the following section:

- Considerations, Limitations, and Issues

## E.1 Considerations, Limitations, and Issues

Please note the following when working on Big Data integration projects:

- Before ODI 12c (12.2.1.1) any Groovy, Jython, Beanshell code in ODI Procedures/Custom KMs were not able to access Hadoop/Pig classes, unless these JARs were added to ODI class path.

  Starting with ODI 12c (12.2.1.1), the ODI Procedures/Custom KMs can access Hadoop/Pig classes if they exist in the paths configured on Hadoop/Pig data servers.
- A new property `oracle.odi.prefer.dataserver.packages` is exposed on Hadoop and Pig data servers, and, also Hive data servers. This property lets you specify which packages are loaded child-first rather than parent-first.

  **Note:** Upgraded repositories will not show this property on upgraded Hadoop/Pig data servers. Only new data servers will show this property.
- In JEE environment, Agent application may be redeployed. However due to Pig's shutdown hook, Logging leak, and other undiscovered leaks, the execution classloader created will not get GC'd. Hence, in ODI 12c (12.2.1), if using Big Data features, the JEE Agent application must not be re-deployed, instead a server restart is required.
- Any package filter applied to a data server must be as specific as possible. Do not try to make things easier by specifying the widest possible filter. For example, if you specify `org.apache` as a filter element, you will get ClassCastException on Beanshell instantiation, XML parsers instantiation, and so on. This happens because according to Java Language Specification two class instances are castable only if they are same type declaration and are loaded by the same classloader. In this example, your interface will be under some sub-package of `org.apache`, for example, `org.apache.util.IMyInterface`. The interface class loaded by the Studio classloader/web application classloader is the casting target. When the implementation class is instantiated through reflection, the instance class's interface class is also loaded by the execution classloader. When JNIEnv code does the checking to see if the caster and castee share the same type declaration, it will turn out to be false since the LHS has Studio/web-application classloader and RHS has execution classloader.
- Execution classloader instances are cached. Changing the data server package filter or data server classpath results in the creation of a new classloader instance. The old classloader may not be GC'd immediately (or even ever). This can lead to running out of heap space. The only solution is a JVM restart.
- When using SDK to create Pig, Hadoop, or any other data server having package filtering property set on it, adding more data server properties requires attention to one detail. You must retrieve the current set of properties, add your properties to it and then set it on the data server. Otherwise, the filtering property will be lost.