Oracle® Fusion Middleware

Integrating Big Data with Oracle Data Integrator

12 *c* (12.2.1.2.6)

E79168-01

December 2016

ORACLE

Oracle Fusion Middleware Integrating Big Data with Oracle Data Integrator, 12 *c* (12.2.1.2.6)

E79168-01

Primary Author: Aslam Khan

Contributing Authors: Alex Kotopoulis

Contents

B Pig Knowledge Modules

C Spark Knowledge Modules

D Component Knowledge Modules

E Considerations, Limitations, and Issues

List of Tables

Preface

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

This preface contains the following topics:.

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*.

- *Oracle Fusion Middleware Release Notes for Oracle Data Integrator*
- *Oracle Fusion Middleware Understanding Oracle Data Integrator*
- *Oracle Fusion Middleware Developing Integration Projects with Oracle Data Integrator*

- *Oracle Fusion Middleware Administering Oracle Data Integrator*
- *Oracle Fusion Middleware Installing and Configuring Oracle Data Integrator*
- *Oracle Fusion Middleware Upgrading Oracle Data Integrator*
- *Oracle Fusion Middleware Application Adapters Guide for Oracle Data Integrator*
- *Oracle Fusion Middleware Developing Knowledge Modules with Oracle Data Integrator*
- *Oracle Data Integrator Connectivity and Knowledge Modules Guide for Oracle Data Integrator Developer's Guide*
-
- *Oracle Fusion Middleware Oracle Data Integrator Tools Reference*
- *Oracle Fusion Middleware Data Services Java API Reference for Oracle Data Integrator*
- *Oracle Fusion Middleware Open Tools Java API Reference for Oracle Data Integrator*
- *Oracle Fusion Middleware Getting Started with SAP ABAP BW Adapter for Oracle Data Integrator*
- *Oracle Fusion Middleware Java API Reference for Oracle Data Integrator*
- *Oracle Fusion Middleware 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:

1.1 Overview of Hadoop Data Integration

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.

Oracle Data Integrator can be used 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 pre-configured knowledge modules (KMs), Oracle Data Integrator lets you to do the following:

- 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

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 first in HDFS using `LKM SQL to File Direct`, and then use `LKM File to Spark` 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 KMs that Oracle Data Integrator provides to integrate data between different source and target technologies.

Table 1-1 Big Data 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
Generic SQL	**HDFS File**	LKM SQL to File SQOOP Direct
	Hive	LKM SQL to Hive SQOOP
	HBase	LKM SQL to HBase SQOOP Direct
	Pig	NA
	Spark	NA
Hadoop	**HBase**	RKM HBase
	Hive	RKM Hive
HDFS	**Kafka**	LKM Kafka to Spark
HDFS	**Spark**	LKM HDFS to Spark
HDFS File	**OS File**	NA
	Generic SQL	LKM File to SQL SQOOP
	Oracle SQL	LKM File to Oracle OLH-OSCH Direct
	HDFS File	NA

Table 1-1 (Cont.) Big Data Knowledge Modules

Source	Target	Knowledge Module
	Hive	LKM File to Hive LOAD DATA Direct LKM HDFS File to Hive LKM HDFS File to Hive (Direct)
	HBase	NA
	Pig	LKM File to Pig
	Spark	LKM File to Spark
Hive	**OS File**	LKM Hive to File Direct
	Generic SQL	LKM Hive to SQL SQOOP
	Oracle SQL	LKM Hive to Oracle OLH-OSCH Direct
	HDFS File	LKM Hive to File Direct
	Hive	IKM Hive Append
	HBase	LKM Hive to HBase Incremental Update HBASE-SERDE Direct
	Pig	LKM Hive to Pig
	Spark	LKM Hive to Spark
HBase	**OS File**	NA
	Generic SQL	LKM HBase to SQL SQOOP
	Oracle SQL	NA
	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
	Generic SQL	LKM SQL to Pig SQOOP
	Oracle SQL	NA
	HDFS File	LKM Pig to File
	Hive	LKM Pig to Hive

Table 1-1 (Cont.) Big Data Knowledge Modules

Source	Target	Knowledge Module
	HBase	LKM Pig to HBase
	Pig	NA
	Spark	NA
Spark	**OS File**	LKM Spark to File
	Generic SQL	LKM Spark to SQL
	Oracle SQL	NA
	HDFS File	LKM Spark to File LKM Spark to HDFS
	Hive	LKM Spark to Hive
	HBase	NA
	Pig	NA
	Spark	LKM SQL to Spark
	Kafka	LKM Spark to Kafka

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:

2.1 Hadoop Data Integration with Oracle Data Integrator

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

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

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

When the data has been aggregated, condensed, or processed into a smaller data set, you can load it into an Oracle database, 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.

For more information, see Integrating Hadoop Data .

2.2 Generate Code in Different Languages with Oracle Data Integrator

By default, Oracle Data Integrator (ODI) uses HiveQL to implement the mappings. However, Oracle Data Integrator also lets you to implement the mappings using Pig Latin and Spark Python. Once your mapping is designed, you can either implement it using the default HiveQL, or choose to implement it using Pig Latin or Spark Python.

Support for Pig Latin and Spark Python in ODI is achieved through a set of component KMs that are specific to these languages. These component KMs are used only when a Pig data server or a Spark data server is used as the staging location for your mapping.

For example, if you use a Pig data server as the staging location, the Pig related KMs are used to implement the mapping and Pig Latin code is generated. Similarly, to generate Spark Python code, you must use a Spark data server as the staging location for your mapping.

Recommendation is to run Spark applications on yarn. Following this recommendation ODI only supports yarn-client and yarn-cluster mode execution and has introduced a runtime 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 allows you to automatically generate 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

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.

    Note that 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.

In Lambda architecture, the data structure model is used with different technologies. For example, the data source for the batch implementation can be HDFS, whereas the streaming implementation can read data from Kafka. In ODI this is represented by using Generic Technologies like **Generic File** and **Generic SQL**.

2.6 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.

2.7 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 a number of previous RDDs (window duration).

The Spark KMs support batch as well as streaming transformations. While the Python code for non-streaming operates on RDD 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**: Time in seconds RDDs are combined to produce the RDDs of the windowed DStream
- **Sliding Interval**: Interval at which the window operation is performed.

Windowing is supported by:

- XKM Spark Aggregation
- XKM Spark Join

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

2.8 Spark Repartitioning and Caching

Caching

In ODI, we leverage on the Spark caching mechanism 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

The number of partitions initially determined by the data block and if the source is HDFS file, the platform runs the Spark application has more available slots for running tasks than number of partitions is loaded, then the platform resource is not fully used.

The repartition can be done in any step of the whole process, it can be done immediately after data is loaded from source or after processing of filter component. In ODI there are Spark base KM options for you to decide whether and where to do repartition.

- **Repartition**

  : If this option set to true, repartition is applied after the transformation of component.

- **Level of Parallelism**

  : Number of partitions and the default is 0.

- **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 Key Function**: User defined key of partitions and the key definition must be a comma separated column list.
- **Partition Function**: User defined partition Lambda function. Default value is a pyspark defined hash function `portable_hash`, which simple compute a hash base on entire row of RDD.

2.9 Kafka Integration with Oracle Data Integrator

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 good practice to use only a single message format per topic. Kafka is integrated into ODI as a new technology.

3

Setting Up the Environment for Integrating Hadoop Data

This chapter provides information steps you need to perform to set up the environment to integrate Hadoop data.

This chapter includes the following sections:

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, 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 runtime 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 a logical agent and a context from the appropriate drop-down lists.
19. Click **Next**.
20. In the **Validate all the settings** panel, click **Test All Settings** to 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 **CDH <version>**.
Base Directory	Specify the base directory. This base directory is automatically populated in all other panels of the wizard. **Note:** This option appears only if the distribution is other than **Manual**.
Distribution Type	Select a distribution type, either **Normal** or **Kerberized**.
Technologies	Select the technologies that you want to configure. **Note:** Data server creation panels only for the selected technologies are displayed.

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	User name with its password.
Hadoop Data Server	Hadoop data server that you want to associate with the HDFS data server.
Additional Classpath	Specify additional classpaths.

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	Quorum of the HBase installation. For example, `localhost:2181`.
User/Password	User name with its password.
Hadoop Data Server	Hadoop data server that you want to associate with the HBase data server.
Additional Classpath	By default, the following classpaths are added: • `/usr/lib/hbase/*` • `usr/lib/hbase/lib/*` Specify the additional classpaths, if required.

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. This name appears in Oracle Data Integrator.
User/Password	User name with its password.
Hadoop Data Server	Hadoop data server that you want to associate with the Kafka data server.
Additional Classpath	The following additional classpaths are added by default: • `/opt/cloudera/parcels/CDH/lib/kafka/libs/*` • `/opt/cloudera/parcels/CDH/lib/base dir basedir/lib/kafka/libs/*` If required, you can add more additional classpaths. **Note:** 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	There are two values, **PLAINTTEXT** or **SASL_PLAINTTEXT**. **SASL_PLAINTTEXT** is used for Kerberized Kafka server. Default value is **PLAINTTEXT**.
oracle.odi.prefer.dataserver.packages	Retrieves the topic and message from Kafka server. The address is **oracle.odi**.

3.2 Creating and Initializing the 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.

Table 3-6 (Cont.) Hadoop Data Server Definition

Field	Description
User/Password	Hadoop user with its password. 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: • Simple Username Authentication • Kerberos Principal Username/Password • Kerberos Credential Cache
HDFS Node Name URI	URI of the HDFS node name. `hdfs://localhost:8020`
Resource Manager/Job Tracker URI	URI of the resource manager or the job tracker. `localhost:8032`
ODI HDFS Root	Path of the ODI HDFS root directory. `/user/<login_username>/odi_home`.
Additional Class Path	Specify additional classpaths. Add the following additional classpaths: • `/usr/lib/hadoop/*` • `/usr/lib/hadoop/lib/*` • `/usr/lib/hadoop-hdfs/*` • `/usr/lib/hadoop-mapreduce/*` • `/usr/lib/hadoop-yarn/*` • `/usr/lib/oozie/lib/*` • `/etc/hadoop/conf/`

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: 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	Description/Value
HADOOP_HOME	Location of Hadoop dir. For example, `/usr/lib/hadoop`
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`

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

Property	Description/Value
HIVE_HOME	Location of Hive dir. For example, `/usr/lib/hive`
HIVE_CONF	Location of Hive configuration files such as hive-site.xml. For example, `/home/shared/hive-conf`
HADOOP_CLASSPATH	`$HIVE_HOME/lib/hive-metastore-*.jar:$HIVE_HOME/lib/libthrift-*.jar:$HIVE_HOME/lib/libfb*.jar:$HIVE_HOME/lib/hive-exec-*.jar:$HIVE_CONF`
HADOOP_CLIENT_OPTS	`-Dlog4j.debug -Dhadoop.root.logger=INFO,console -Dlog4j.configuration=file:/etc/hadoop/conf.cloudera.yarn/log4j.properties`
ODI_ADDITIONAL_CLASSPATH	`$HIVE_HOME/lib/'*':$HADOOP_HOME/client/*:$HADOOP_CONF`
HIVE_SESSION_JARS	`$HIVE_HOME/lib/hive-contrib-*.jar:<ODI library directory>/wlhive.jar` • Actual path of `wlhive.jar` can be determined under ODI installation home. • 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. • List of JARs is separated by ":", wildcards in file names must not evaluate to more than one file. • Follow the steps for Hadoop Security models, such as Apache Sentry, to allow the Hive ADD JAR call used inside ODI Hive KMs: – Define the environment variable HIVE_SESSION_JARS as empty. – 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	Decription/Value
HBASE_HOME	Location of HBase dir. For example, `/usr/lib/hbase`

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

Property	Decription/Value
HADOOP_CLASSPATH	`$HBASE_HOME/lib/hbase-*.jar:$HIVE_HOME/lib/hive-hbase-handler*.jar:$HBASE_HOME/hbase.jar`
ODI_ADDITIONAL_CLASSPATH	`$HBASE_HOME/hbase.jar`
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: • Define the environment variable HIVE_SESSION_JARS as empty. • Add all required jars for Hive in the global Hive configuration hive-site.xml.

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

Property	Description/Value
OLH_HOME	Location of OLH installation. For example, `/u01/connectors/olh`
OLH_FILES	`usr/lib/hive/lib/hive-contrib-1.1.0-cdh5.5.1.jar`
ODCH_HOME	Location of OSCH installation. For example, `/u01/connectors/osch`
HADOOP_CLASSPATH	`$OLH_HOME/jlib/*:$OSCH_HOME/jlib/*` In order to work with OLH, the Hadoop jars in the `HADOOP_CLASSPATH` have to be manually resolved without wildcards.

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

Property	Description/Value
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. For example: `$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`
OLH_SHAREDLIBS	`$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`
ODI_ADDITIONAL_CLASSPATH	`$OSCH_HOME/jlib/'*'`

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

Property	Description/Value
SQOOP_HOME	Location of Sqoop dir. For example, `/usr/lib/sqoop`
SQOOP_LIBJARS	Location of the SQOOP library jars. For example, `usr/lib/hive/lib/hive-contrib-1.1.0-cdh5.5.1.jar`

Creating and Initializing the Hadoop Data Server

3.3 Creating a Hadoop Physical Schema

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

Create for this physical schema a logical schema using the standard procedure, as described in *Creating a Logical Schema* 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.

To configure the Oracle Data Integrator agent:

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

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

2. Install Hive on your Oracle Data Integrator agent computer.
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. Note that you need to 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.

To install and configure Oracle Loader for Hadoop:

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

   See *Installing Oracle Loader for Hadoop* 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 Oracle SQL Connector for Hadoop Distributed File System Setup in *Oracle Big Data Connectors User's Guide*.

3. Set the properties for Oracle Loader for Hadoop on your ODI agent computer. Note that 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 node04 of the Oracle Big Data Appliance, where the Oracle Data Integrator agent runs.
2. Set the environment variables by using the following commands. 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 -D
javax.xml.parsers.DocumentBuilderFactory=com.sun.org.apache.x
erces.internal. jaxp.DocumentBuilderFactoryImpl -D
java.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 , we need to 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=trueAddVMOption -
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`	**Example of configuration file**
		`JDBC_DRIVER_01 { com.sun.security.auth.module.Krb5LoginModule required debug=true useTicketCache=true ticketCache="/tmp/krb5cc_500" doNotPrompt=true ; };`
		Example of Hive URL
		`jdbc:weblogic:hive://slc05jvn.us.oracle.com:10000;DatabaseName=default;AuthenticationMethod=kerberos;ServicePrincipalName=hive/slc05jvn.us.oracle.com@US.ORACLE.COM`

HBase	`export HBASE_HOME=/scratch/shixu/etc/hbase/conf`
	`export HBASE_CONF_DIR = $HBASE_HOME/conf`
	`export HBASE_OPTS ="-Djava.security.auth.login.config=$HBASE_CONF_DIR/hbase-client.jaas"export HBASE_MASTER_OPTS ="-Djava.security.auth.login.config=$HBASE_CONF_DIR/hbase-server.jaas"`
	ODI Studio Configuration:
	`AddVMOption -Djava.security.auth.login.config=$HBASE_CONF_DIR/hbase-client.jaas"`
		Example of Hbase configuration file:
		`hbase-client.jaas Client { com.sun.security.auth.module.Krb5LoginModule required useKeyTab=false useTicketCache=true; };`

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

Technology	Configuration	Example

Spark	Spark Kerberos configuration is done through spark submit parameters
	
	`--principal // define principle name`
	`--keytab // location of keytab file`
		Example of spark-submit command:
		
		`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 shixu@US.ORACLE.com --keytab /tmp/shixu.tab --queue default /tmp/New_Mapping_Physical.py`
		`./bin/spark-submit --class org.apache.spark.examples.SparkPi --master yarn-cluster --num-executors 1 --driver-memory 512m --executor-memory 512m --executor-cores 1 lib/spark-examples*.jar 10`

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:**
		`KafkaClient {`
		`com.sun.security.auth.module.Krb5LoginModule required useKeyTab=false useTicketCache=true ticketCache="/tmp/krb5cc_1500"`
		`serviceName="kafka"; };`
		The location of Kafka configuration file is set in ODI Studio VM option
		`AddVMOption -Djava.security.auth.login.config=/scratch/shixu/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.	

See also, "HiveServer2 Security Configuration" in the CDH5 Security Guide at the following URL:

`http://www.cloudera.com/content/cloudera-content/cloudera-docs/CDH5/latest/CDH5-Security-Guide/cdh5sg_hiveserver2_security.html`

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

   See *Oracle Big Data Appliance Software User's Guide* for instructions about managing Kerberos on Oracle Big Data Appliance.

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

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

4.1 Integrating 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. See Setting Up File Data Sources See Setting Up Hive Data Sources See Setting Up HBase Data Sources See Setting Up Kafka Data Sources See Setting Up Cassandra Data Sources See Setting Up HDFS Data Sources
Import Hadoop Knowledge Modules	Import the Hadoop KMs into Global Objects or a project. See Importing Hadoop Knowledge Modules
Create Oracle Data Integrator Models	Reverse-engineer the Hive and HBase models to create Oracle Data Integrator models. See Creating a Oracle Data Integrator Model from a Reverse-Engineered Hive, HBase, and HDFS Models
Integrate Hadoop Data	Design mappings to load, validate, and transform Hadoop data. See Loading Data from Files into Hive See Loading Data from HBase into Hive See Loading Data from Hive into Hbase See Loading Data from an SQL Database into Hive, HBase, and File using SQOOP See Validating and Transforming Data Within Hive See Loading Data into an Oracle Database from Hive and File See Loading Data into an SQL Database from Hbase, Hive and File using SQOOP See Loading Data from Kafka to Spark See Loading Data from HDFS File to Hive See Loading Data from HDFS File to Spark See Loading Data from Hive to Files

4.2 Setting Up File Data Sources

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. For HDFS files, 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.

   Note:

   No dedicated technology is defined for HDFS files.

Integrating Hadoop Data

4.3 Setting Up HDFS Data Sources

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 allows 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

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

Prerequisites

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

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

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

Prerequisites

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

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.mydomain.com,zkhost2.mydomain.com,zkhost3.mydomain.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

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

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

1. Create a Data Server object under Kafka technology.
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 type of file.

   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 Kafka Integration with Oracle Data Integrator.

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

4.7 Setting Up Cassandra Data Sources

This topic provides steps in Oracle Data Integrator that are required for connecting to a Cassandra system. Oracle Data Integrator connects to Cassandra by using JDBC.

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

You must add all Cassandra-specific flex fields.

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.mycompany.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

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.

- CKM Hive
- RKM Hive

- RKM HBase
- RKM Cassandra

Integrating Hadoop Data

4.9 Creating a Oracle Data Integrator Model from a Reverse-Engineered Hive, HBase, and HDFS Models

You must create a ODI Model from a reverse-engineered Hive, HBase, and HDFS Models. The reverse engineering process creates Hive and HBase data stores for the corresponding Hive and HBase 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 Tables

4.9.1 Creating a Model

To create a model that is based on the technology hosting Hive, HBase, or HDFS and on the logical schema created when you configured the Hive, HBase, HDFS or File connection, follow the standard procedure described in *Oracle Fusion Middleware 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 *Oracle Fusion Middleware 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.

A storage tab is added to the Hive data store and there is flexibility of how data is stored and formatted within Hive. If the Hive table already exists, you can use the **Reverse Engineer** process on the Hive model, using the custom Hive RKM to populate the fields.

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 *Oracle Fusion Middleware 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 Tables

HDFS files are used in reverse engineering. You can reverse engineer HDFS using file technology or HDFS technology.

Reverse Engineering HDFS with File Technology

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.

Reverse Engineering HDFS with HDFS Technology

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

- Create a HDFS data store.
- From the Storage Tab, choose from the **Storage Format** field and corresponding schema file must be specified in the **Schema File** field.
- 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 files are used in KMs such as File to Hive, File to Spark and this uses the ODI file technology as a source. You can also use the HDFS LKMs (LKM HDFS File to Hive) and these KMs use the ODI HDFS technology.

Depending on which KMs you want to use, you can select a different technology for the files. Reverse Engineering HDFS will support the Avro, Json, Parquet and delimited formats.

Refer to *Reverse-engineer a File Model* in *Oracle Data Integrator Connectivity and Knowledge Modules Guide for Oracle Data Integrator Developer's Guide* for more information.

Creating a Oracle Data Integrator Model from a Reverse-Engineered Hive, HBase, and HDFS Models

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 *Oracle Fusion Middleware Developing Integration Projects with Oracle Data Integrator*. This topic details information specific to Cassandra tables.

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

4.10 Loading Data from Files into Hive

The KMs support Loading Data from HDFS, however, 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.

   Refer to *Oracle Data Integrator Connectivity and Knowledge Modules Guide for Oracle Data Integrator Developer's Guide* for information about reverse engineering and configuring local file data sources.

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

4.11 Loading Data from Hive to 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.

   Refer to "Setting Up Hive Data Sources" for information about reverse engineering and configuring 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.

4.12 Loading Data from HBase 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.

   Refer to "Setting Up HBase Data Sources" for information about reverse engineering and configuring 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.

4.13 Loading Data from Hive 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.

   Refer to "Setting Up Hive Data Sources" for information about reverse engineering and configuring 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 as well as 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.

4.14 Loading Data from an SQL Database into Hive, HBase, and File 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.

   Refer to *Oracle Data Integrator Connectivity and Knowledge Modules Guide for Oracle Data Integrator Developer's Guide* for information about reverse engineering and configuring SQL data sources.

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.

   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].

4.15 Loading Data from an SQL Database 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.

   Refer to *Oracle Data Integrator Connectivity and Knowledge Modules Guide for Oracle Data Integrator Developer's Guide* for information about reverse engineering and configuring SQL data sources.

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.

4.16 Loading Data from an SQL Database into File using SQOOP

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

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

   Refer to *Oracle Data Integrator Connectivity and Knowledge Modules Guide for Oracle Data Integrator Developer's Guide* for information about reverse engineering and configuring SQL data sources.

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 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].

4.17 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.

   Refer to *Oracle Data Integrator Connectivity and Knowledge Modules Guide for Oracle Data Integrator Developer's Guide* for information about reverse engineering and configuring SQL data sources.

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.

4.18 Validating and Transforming Data Within Hive

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

- IKM Hive Control Append

  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 (Deprecated).

- IKM Hive Transform

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

4.19 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 4-2 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. For more information, see IKM File-Hive to Oracle (OLH-OSCH) [Deprecated].

Table 4-2 (Cont.) Knowledge Modules to load data into Oracle Database

Knowledge Module	Use To...
LKM File to Oracle OLH-OSCH	Load data from an HDFS file into an Oracle staging table using Oracle Loader for Hadoop. 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. 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. 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. For more information, see LKM Hive to Oracle OLH-OSCH Direct.

4.20 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 4-3 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. For more information, see IKM File-Hive to SQL (SQOOP) [Deprecated].
LKM HBase to SQL SQOOP	Load data from an HBase source into an SQL database target using SQOOP. 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. 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. For more information, see LKM Hive to SQL SQOOP.

4.21 Loading Data from Kafka to Spark

Loading data from Kafka to Spark.

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

   Refer to Setting Up Kafka Data Sources for configuring Kafka data sources.

2. Create a mapping using the Kafka data store as the source and the corresponding Spark table as the target.

3. Use the LKM Kafka to Spark `zookeeper.connect` in case of receiver-based connection or `metadata.broker.list` in case of direct connection knowledge module, specified in the physical diagram of the mapping.

   This integration knowledge module loads data from Kafka into Spark and supports inserting new rows as well as updating existing data.

   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 tells you how to audit Hadoop logs.

This chapter includes the following sections:

5.1 Executing Oozie Workflows with Oracle Data Integrator

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. 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. 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. 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 needs to be associated to this Hadoop data server.

To set up the Oozie runtime engine:

1. In the Topology Navigator, right-click the **Oozie Runtime Engine** node in the Physical Architecture navigation tree and click **New**.
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.

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

Field	Values
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. The poll frequency can be specified in seconds (s), minutes (m), hours (h), days (d), and years (d). For example, 5m or 4h.
Lifespan	Time period for which the Hadoop audit logs retrieval coordinator stays enabled to schedule audit logs retrieval workflows. 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. 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. 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 **Oozie Runtime Engine** node in the Logical Architecture navigation tree.
2. Select **New Logical Agent.**
3. Fill in the **Agent 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 agent 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 object or scenario using the Oozie runtime engine to execute an Oozie Workflow on the Oozie engine. When running the ODI design object or scenario, you can choose to only deploy the Oozie workflow without executing it.

To deploy or 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 **Run Using** drop-down list, select the Oozie runtime engine.
3. Select **Deploy Only** check box to only deploy the Oozie workflow without executing it.
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

Spark Streaming Support

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

This chapter includes the following sections:

- Enabling Streaming Support for Oracle Data Integrator
- Enabling Streaming Support
- Spark Streaming DataServer Properties
- Extra Spark Streaming Data Properties
- Execute Mapping in Streaming Mode

6.1 Enabling Streaming Support for Oracle Data Integrator

Summarizes the steps you need to perform to enable streaming support for Oracle Data Integrator.

Table 6-1 Enabling Streaming Support

Step	Description
Enable streaming support	When steaming mode is enabled, ODI generates the Spark code which performs the transformation to run the streaming function. See Enabling Streaming Support.
Spark execution and Checkpointing	Checkpointing ensures that the Spark application will restart from where it left, if a checkpoint is found. See Execute Mapping in Streaming Mode.

For additional information, refer to Spark Documentation and Cassandra.

6.2 Enabling Streaming Support

This topic provides the steps to enable streaming support in ODI.

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.

6. Enable the **Streaming** flag in the Physical design of a mapping.

   ODI generates the Spark code which performs the transformation to run the streaming function.

6.2.1 Spark Streaming DataServer Properties

Provides the streaming properties that are specific to Spark Technology that are added to the Spark Execution Unit.

Table 6-2 Spark Streaming DataServer Properties

Key	Value
spark.checkpointingBaseDir	This property defines the base directory and under this base directory every Spark EU will create a sub-directory with the name EU. 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. • If set to false, the Spark Streaming application will ignore any existing checkpoints. • If there is no checkpoint, it will start normally.
spark.batchDuration	Displays the time in seconds and Spark batches up Stream input data.
spark.rememberDuration	Displays the time in seconds and the sets the Spark Streaming context to remember RDDs.
spark.checkpointing	Enables Spark checkpointing.
spark.streaming.timeout	Displays the time in seconds and the Spark waits before stopping a Streaming applications. Default is 60.
odi-execution-mode	• **SYNCHRONOUS**: Spark application is submitted and monitored through `OdiOSCommand`. • **ASYNCHRONOUS**: Spark application is submitted asynchronously through `OdiOSCommand` and then monitored through Spark REST APIs.
spark.ui.enabled	Enables the Spark Live REST API. **Note:** Set to true for asynchronous execution.

Table 6-2 (Cont.) Spark Streaming DataServer Properties

Key	Value
spark.eventLog.enabled	Enables Spark event logs. This allows the logs to be accessible by the Spark History Server. **Note:** Set to true for asynchronous execution.
principal	Kerberized User name.
keytab	Kerberized Password.
odi.spark.enableUnsupportedSparkModes	This check is introduced, as only yarn-client and yarn-cluster are supported.

6.2.2 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 6-3 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	
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.

Table 6-3 (Cont.) Extra Spark Streaming Properties

Key	Value
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	

6.3 Execute Mapping in Streaming Mode

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

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. Create a mapping using the data store.
6. Enable the **Streaming** flag in the Physical design of a mapping.
7. Enable the `spark.checkpointing` to set the Checkpointing directory.

   Every mapping will have its unique checkpointing directory.
8. 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

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:

7.1 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

7.2 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.

7.2.1 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 7-1 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

7.2.2 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 7-2 Hive Data Server Connection Details

Field	Description
JDBC Driver	`DataDirect Apache Hive JDBC Driver` Use this JDBC driver to connect to the Hive Data Server. The driver documentation is available at the following URL: http://media.datadirect.com/download/docs/jdbc/alljdbc/help.html#page/userguide/rfi1369069225784.html#
JDBC URL	`jdbc:weblogic:hive://<host>:<port>[;property=value[;...]]` For example, `jdbc:weblogic:hive://localhost:10000;DatabaseName=default;User=default;Password=default` **Kerberized**: `jdbc:weblogic:hive://<host>:<port>;DatabaseName=<value>;AuthenticationMethod=kerberos;ServicePrincipalName=<value>` For example, `jdbc:weblogic:hive://localhost:10000;DatabaseName=default;AuthenticationMethod=kerberos;ServicePrincipalName=hive`

Setting Up Hive Data Server

7.3 Creating a Hive Physical Schema

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

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

Setting Up Hive Data Server

7.4 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.

7.4.1 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 7-3 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: • **Local Mode** Select to run the job in local mode. In this mode, pig scripts located in the local file system are executed. MapReduce jobs are not created. • **MapReduce Mode** Select to run the job in MapReduce mode. In this mode, pig scripts located in the HDFS are executed. MapReduce jobs are created. **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. **Note:** This field is displayed only when the **MapReduce Mode** option is set to Process Type.
Additional Classpath	Specify additional classpaths. Add the following additional classpaths: • `/usr/lib/pig/lib` • `/usr/lib/pig/pig-0.12.0-cdh<version>.jar` Replace <version> with the Cloudera version you have. For example, `/usr/lib/pig/pig-0.12.0-cdh5.3.0.jar`. • `/usr/lib/hive/lib` • `/usr/lib/hive/conf` For pig-hcatalog-hive, add the following classpath in addition to the ones mentioned above: `/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

7.4.2 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 7-4 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

7.5 Creating a Pig Physical Schema

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

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

Setting Up Pig Data Server

7.6 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.

7.6.1 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 7-5 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. **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: • `/usr/lib/spark/*` • `/usr/lib/spark/lib/*` If required, you can add more additional classpaths. **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

7.6.2 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 7-6 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. Note that jars 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.

Table 7-6 (Cont.) Spark Data Server Properties

Property	Description
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**.
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

7.7 Creating a Spark Physical Schema

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

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

Setting Up Spark Data Server

7.8 Generating Code in Different Languages

By default, Oracle Data Integrator generates HiveQL code. To generate Pig Latin or Spark code, you must use the Pig data server or the Spark data server as the staging location for your 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

8

Working with Unstructured Data

This chapters provides an overview of the Jagged component and the Flatten component. These components help you to process unstructured data.

This chapter includes the following sections:

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 *Oracle Fusion Middleware Developing Integration Projects with Oracle Data Integrator*.
- XKM Jagged.

9

Working with Complex files

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

This chapter includes the following sections:

9.1 HDFS Formats

HDFS file formats supported are Json, Avro and Parquet. The format is specified by setting the storage format value which can be found on the storage tab of the Data Store. For all files of HDFS, the storage type (Json, Avro, Parquet) are defined in the data store. JSON, Avro and Parquet formats contain complex data types, like array or Object. During the Reverse Engineer phase, the schema definition for these types are converted to Avro and stored in the data format column of the attribute with the complex data type. This information is used when flattening this data in the mappings.

For JSON, Avro and Parquet that each type requires the location of a schema file to be entered. For Delimited, you will need to specify the Record and field separator information, number of heading lines. 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 HDFS files.

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	Yes
Delimited	No	No	Yes	Yes	Yes
JSON	Yes (Schema required)	Yes	Yes	Yes	Yes

Table 9-1 (Cont.) HDFS File Formats

File Format	Reverse Engineer	Complex Type Support	Load into Hive	Load into Spark	Write from Spark
Parquet	Yes (Schema required)	Yes	Yes	Yes	Yes

Separate KMs for each file format are not required. You can create just one or two KMs for each target (a standard LKM and where appropriate a Direct Load LKM). The file can either be delimited or fixed format. The new LKM HDFS File to Hive supports loading only HDFS file into Hive, the file can be in the format of JSON, Avro, Parquet, Delimited etc, with complex data.

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 Files

Provides information on working with user defined metadata that drives the flatten component.

Oracle Data Integrator provides a Flatten component that can process input data with complex structure and produce flatten representation of the same data using standard data types. The input data may be in a database, in an XML, or any other source.

There are three check-box properties which are Include Nulls, Collection, and Structure. The collection check-box indicates whether the complex type attribute is a collection such as an array. It is automatically assigned if the complex type attribute has a data format defined. The structure check-box indicates whether the complex type is an object, record, or structure, not just a collection of scalar types. The Include Nulls check-box indicates whether null complex data should be processed. Some technologies, particularly Spark, can drop records containing null complex attributes.

Working with Complex Files with Improved Flattening for HDFS

The complex type attribute is an upstream attribute, of a collection or object type, to be flattened. Each flatten component can flatten only one complex type attribute, since just a single property is used to specify it.

Complex type member attributes that comprise a collection or object are automatically added to the flatten component when the complex type attribute is selected. The complex type member attributes can be added, changed, or deleted manually as well.

The complex type (CT) attribute can be accessed as a data store column, and the collection or object attributes are determined from the data_format field.

Note: The flatten component is only supported with Spark 1.3 and above.

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

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

9.3 Identifying, Adding and Removing Flattened Attributes

Without complex type metadata, user input is required to add the flattened attributes. With the addition of complex type metadata, the attributes can be added automatically on definition of the complex type attribute.

The addition of new flattened attributes occurs when the **Complex Type Attribute** property is set.

The new flattened attributes have null expressions, since they do not directly reference upstream attributes, but are derived from an upstream attribute of a complex type. The name of the flattened attribute can match the name of the complex type member. However, if the flattened attribute receives a different name due to a name conflict with another attribute, the attribute tag property is used to identify the complex type member. If the flattened attribute name is changed manually, ensure that the tag property is set to the name of the complex type member.

For example, consider a complex type attribute `full_name` which contains member attributes `first_name` and `last_name`. When `full_name` is selected as the complex type attribute, two new flattened attributes, `first_name` and `last_name`, are created. If attribute `last_name` is changed to `lname`, its tag is set to `last_name` to identify the complex type member name to be used in code generation.

A user may not wish to flatten all the attributes of the collection of a complex type. Extra attributes can be deleted.

9.4 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**.

4. Create a mapping with HDFS file as source and Hive file as target.
5. Use the LKM file HDFS 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.

9.5 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 chapter includes the following sections:

- CKM Hive (Deprecated)
- 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. Set this option to NO, if you wish to retain temporary objects (tables, files and scripts) after integration. Useful for debugging. Default: true.
SQOOP_PARALLELISM	Number of SQOOP parallel mappers Specifies the degree of parallelism. More precisely the number of mappers. Number of mapper processes used for extraction. When SQOOP_PARALLELISM > 1, SPLIT_BY must be defined.
SPLIT_BY	Target column name for splitting the source data. 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. 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(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-1 (Cont.) LKM SQL to Hive SQOOP

Option	Description
BOUNDARY_QUERY	Query to retrieve min/max value for calculating data chunks using SPLIT_BY column. 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. Example for hard-coded ranges for an Oracle source: SELECT 1000, 2000 FROM DUAL For preserving context independence regular table names should be inserted through odiRef.getObjectName calls. For example: SELECT MIN(EMPNO), MAX(EMPNO) FROM < %=odiRef.getObjectName(EMP")%>"
TEMP_DIR	Local directory for temporary files. Directory used for storing temporary files like squoop script, stdout and stderr redirects. Leave blank to use system's default temp dir (<? =System.getProperty(java.io.tmp")?>)".
MAPRED_OUTPUT_BASE_DIR	MapReduce Output Directory. 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? 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. Extra optional properties for SQOOP file: section Hadoop properties.
EXTRA_SQOOP_CONF_PROPERTIES	Optional SQOOP properties. Extra optional properties for SQOOP file: section SQOOP properties.
EXTRA_SQOOP_CONNECTOR_CONF_PROPERTIES	Optional SQOOP connector properties. 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 prior to 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. Set this option to NO, if you wish to retain temporary objects (tables, files and scripts) after integration. Useful for debugging. Default: true.
SQOOP_PARALLELISM	Number of SQOOP parallel mappers Specifies the degree of parallelism. More precisely the number of mappers. Number of mapper processes used for extraction. When SQOOP_PARALLELISM > 1, SPLIT_BY must be defined.
SPLIT_BY	Target column name for splitting the source data. 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. 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(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-2 (Cont.) LKM SQL to File SQOOP Direct

Option	Description
BOUNDARY_QUERY	Query to retrieve min/max value for calculating data chunks using SPLIT_BY column. 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. Example for hard-coded ranges for an Oracle source: SELECT 1000, 2000 FROM DUAL For preserving context independence regular table names should be inserted through odiRef.getObjectName calls. For example: SELECT MIN(EMPNO), MAX(EMPNO) FROM < %=odiRef.getObjectName(EMP")%>"
TEMP_DIR	Local directory for temporary files. Directory used for storing temporary files like squoop script, stdout and stderr redirects. Leave blank to use system's default temp dir (<? =System.getProperty(java.io.tmp")?>)".
MAPRED_OUTPUT_BASE_DIR	MapReduce Output Directory. 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? 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. Extra optional properties for SQOOP file: section Hadoop properties.
EXTRA_SQOOP_CONF_PROPERTIES	Optional SQOOP properties. Extra optional properties for SQOOP file: section SQOOP properties.
EXTRA_SQOOP_CONNECTOR_CONF_PROPERTIES	Optional SQOOP connector properties. 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? Check this option, if you wish to create the target table.
TRUNCATE	Replace existing target data? Set this option to true, if you wish to replace any existing target table content with the new data.
DELETE_TEMPORARY_OBJECTS	Delete temporary objects at end of mapping. Set this option to NO, if you wish to retain temporary objects (tables, files and scripts) after integration. Useful for debugging. Default: true.
SQOOP_PARALLELISM	Number of SQOOP parallel mappers Specifies the degree of parallelism. More precisely the number of mappers. Number of mapper processes used for extraction. When SQOOP_PARALLELISM > 1, SPLIT_BY must be defined.
SPLIT_BY	Target column name for splitting the source data. 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. 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(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. 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. Example for hard-coded ranges for an Oracle source: SELECT 1000, 2000 FROM DUAL For preserving context independence regular table names should be inserted through odiRef.getObjectName calls. For example: SELECT MIN(EMPNO), MAX(EMPNO) FROM < %=odiRef.getObjectName(EMP")%>"
TEMP_DIR	Local directory for temporary files. Directory used for storing temporary files like squoop script, stdout and stderr redirects. Leave blank to use system's default temp dir (<? =System.getProperty(java.io.tmp")?>)".
MAPRED_OUTPUT_BASE_DIR	MapReduce Output Directory. 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? 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. Extra optional properties for SQOOP file: section Hadoop properties.
EXTRA_SQOOP_CONF_PROPERTIES	Optional SQOOP properties. Extra optional properties for SQOOP file: section SQOOP properties.
EXTRA_SQOOP_CONNECTOR_CONF_PROPERTIES	Optional SQOOP connector properties. 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. Specifies the degree of parallelism. More precisely the number of mappers. Number of mapper processes used for extraction. When SQOOP_PARALLELISM > 1, SPLIT_BY must be defined.
WORK_TABLE_OPTIONS	Work table options. Use this option if you wish to override standard technology specific work table options. When left blank, these options values are used. Oracle: NOLOGGING DB2 UDB: NOT LOGGED INITIALLY Teradata: no fallback, no before journal, no after journal
TERADATA_WORK_TABLE_TYPE	Teradata work table type. Use SET or MULTISET table for work table.
TERADATA_OUTPUT_METHOD	Teradata Load Method. Specifies the way the Teradata Connector will load the data. Valid values are: • batch.insert: multiple JDBC connections using batched prepared statements (simplest to start with) • multiple.fastload: multiple FastLoad connections • internal.fastload: single coordinated FastLoad connections (most performant) Please see Cloudera's Teradata Connectors User Guide for more details.
TEMP_DIR	Local directory for temporary files. Directory used for storing temporary files like squoop script, stdout and stderr redirects. Leave blank to use system's default temp dir (<? =System.getProperty(java.io.tmp")?>)".

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

Option	Description
MAPRED_OUTPUT_BASE_DIR	MapReduce Output Directory. 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? 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. Extra optional properties for SQOOP file: section Hadoop properties.
EXTRA_SQOOP_CONF_PROPERTIES	Optional SQOOP properties. Extra optional properties for SQOOP file: section SQOOP properties.
EXTRA_SQOOP_CONNECTOR_CONF_PROPERTIES	Optional SQOOP connector properties. 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. Set this option to NO, if you wish to retain temporary objects (tables, files and scripts) after integration. Useful for debugging.

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

Option	Description
SQOOP_PARALLELISM	Number of SQOOP parallel mappers. Specifies the degree of parallelism. More precisely the number of mappers. Number of mapper processes used for extraction. When SQOOP_PARALLELISM > 1, SPLIT_BY must be defined.
WORK_TABLE_OPTIONS	Work table options. Use this option if you wish to override standard technology specific work table options. When left blank, these options values are used. Oracle: NOLOGGING DB2 UDB: NOT LOGGED INITIALLY Teradata: no fallback, no before journal, no after journal
TERADATA_WORK_TABLE_TYPE	Teradata work table type. Use SET or MULTISET table for work table.
TERADATA_OUTPUT_METHOD	Teradata Load Method. Specifies the way the Teradata Connector will load the data. Valid values are: • batch.insert: multiple JDBC connections using batched prepared statements (simplest to start with) • multiple.fastload: multiple FastLoad connections • internal.fastload: single coordinated FastLoad connections (most performant) Please see Cloudera's Teradata Connectors User Guide for more details.
TEMP_DIR	Local directory for temporary files. Directory used for storing temporary files like squoop script, stdout and stderr redirects. Leave blank to use system's default temp dir (<? =System.getProperty(java.io.tmp")?>)".
MAPRED_OUTPUT_BASE_DIR	MapReduce Output Directory. 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? 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. Extra optional properties for SQOOP file: section Hadoop properties.

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

Option	Description
EXTRA_SQOOP_CONF_PROPERTIES	Optional SQOOP properties. Extra optional properties for SQOOP file: section SQOOP properties.
EXTRA_SQOOP_CONNECTOR_CONF_PROPERTIES	Optional SQOOP connector properties. 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. Set this option to NO, if you wish 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. The unloading from HBase data is done via Hive. This KM option defines the Hive database, which will be used for creating the Hive HBase-SerDe table for unloading the HBase data.
SQOOP_PARALLELISM	Number of SQOOP parallel mappers. Specifies the degree of parallelism. More precisely the number of mappers. Number of mapper processes used for extraction. When SQOOP_PARALLELISM > 1, SPLIT_BY must be defined.
WORK_TABLE_OPTIONS	Work table options. Use this option if you wish to override standard technology specific work table options. When left blank, these options values are used. Oracle: NOLOGGING DB2 UDB: NOT LOGGED INITIALLY Teradata: no fallback, no before journal, no after journal

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

Option	Description
TERADATA_WORK_TABLE_TYPE	Teradata work table type. Use SET or MULTISET table for work table.
TERADATA_OUTPUT_METHOD	Teradata Load Method. Specifies the way the Teradata Connector will load the data. Valid values are: • batch.insert: multiple JDBC connections using batched prepared statements (simplest to start with) • multiple.fastload: multiple FastLoad connections • internal.fastload: single coordinated FastLoad connections (most performant) Please see Cloudera's Teradata Connectors User Guide for more details.
TEMP_DIR	Local directory for temporary files. Directory used for storing temporary files like squoop script, stdout and stderr redirects. Leave blank to use system's default temp dir (<? =System.getProperty(java.io.tmp")?>)".
MAPRED_OUTPUT_BASE_DIR	MapReduce Output Directory. 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? 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. Extra optional properties for SQOOP file: section Hadoop properties.
EXTRA_SQOOP_CONF_PROPERTIES	Optional SQOOP properties. Extra optional properties for SQOOP file: section SQOOP properties.
EXTRA_SQOOP_CONNECTOR_CONF_PROPERTIES	Optional SQOOP connector properties. Extra optional properties for SQOOP file: section SQOOP connector properties.

A.7 LKM HDFS File to Hive

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

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, if you want to retain the temporary objects (tables, files and scripts) post integration.

A.8 LKM HDFS File to Hive (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 (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, if you want to retain the temporary objects (tables, files and scripts) post integration.
CREATE_TARG_TABLE	Create target table? Check this option, if you wish to create the target table.
TRUNCATE	Replace existing target data? Set this option to true, if you wish 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. Check this option if you wish to create the target table.
TRUNCATE	Replace all target table data. 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 in order 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. Check this option if you wish to create the target table.
TRUNCATE	Replace all target table data. Set this option to true, if you wish 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. 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

Option	Description
EXTERNAL_TABLE	Preserve file in original location? Defines whether to declare the target/staging table as externally managed. Default: false 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. If EXTERNAL_TABLE is set to true: All files in the directory given by the physical data schema will be loaded. So any filename or wildcard information from the source DataStore's resource name will be ignored. The directory structure and file names must comply with Hives directory organization for tables, e.g. for partitioning and clustering. The directory and its files must reside in HDFS. 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? Defines whether the source file is to be considered local (= outside of the current Hadoop cluster). Default: true If FILE_IS_LOCAL is set to true, the data file(s) are copied into the Hadoop cluster first. 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? This checkbox option defines whether the KM should stop, if no input file has been found.
OVERRIDE_ROW_FORMAT	Custom row format clause. 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. Example for reading Apache Combined WebLog files: ROW FORMAT SERDE 'org.apache.hadoop.hive.contrib.serde2.RegexSerDe' <EOL>WITH SERDEPROPERTIES (<EOL> input.regex" = "([^]*) ([^]*) ([^]*) (-|\\[[^\\]]*\\]) ([^ \"]*|\"[^\"]*\") (-|[0-9]*) (-|[0-9]*) (\".*?\") (\".*?\") (\".*?\")"

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. Check this option if you wish to create the target table.
TRUNCATE	Replace all target table data. 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. 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? Defines whether to declare the target/staging table as externally managed. Default: false 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. If EXTERNAL_TABLE is set to true: All files in the directory given by the physical data schema will be loaded. So any filename or wildcard information from the source DataStore's resource name will be ignored. The directory structure and file names must comply with Hives directory organization for tables, e.g. for partitioning and clustering. The directory and its files must reside in HDFS. No Hive LOAD-DATA-statements are submitted and thus loading of files to a specific partition (using a target-side expression) is not possible.

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

Option	Description
FILE_IS_LOCAL	Is this a local file? Defines whether the source file is to be considered local (= outside of the current Hadoop cluster). Default: true If FILE_IS_LOCAL is set to true, the data file(s) are copied into the Hadoop cluster first. 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? This checkbox option defines whether the KM should stop, if no input file has been found.
OVERRIDE_ROW_FORMAT	Custom row format clause. 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. Example for reading Apache Combined WebLog files: ROW FORMAT SERDE 'org.apache.hadoop.hive.contrib.serde2.RegexSerDe' <EOL>WITH SERDEPROPERTIES (<EOL> input.regex" = "([^]*) ([^]*) ([^]*) (-|\\[[^\\]]*\\]) ([^ \"]*|\"[^\"]*\") (-|[0-9]*) (-|[0-9]*) (\".*?\") (\".*?\") (\".*?\")"

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 as well as 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. Check this option if you wish to create the target table.
TRUNCATE	Replace all target table data. Set this option to true, if you wish to replace the target table content with the new data.
HBASE_WAL	Disable Write-Ahead-Log. HBase uses a Write-Ahead-Log to protect against data loss. For better performance, WAL can be disabled. Please note that this setting applies to all Hive commands executed later in this session.
DELETE_TEMPORARY_OBJECTS	Delete temporary objects at end of mapping. Set this option to NO, if you wish 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? Defines whether the target file is to be considered local (outside of the current Hadoop cluster).
STORED_AS	File format. 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 Hive to Oracle OLH-OSCH

Option	Description
DELETE_TEMPORARY_OBJECTS	Delete temporary objects at end of mapping. Set this option to NO, if you wish to retain temporary objects (tables, files and scripts) after integration. Useful for debugging.
OLH_OUTPUT_MODE	How to transfer data into Oracle? This option specifies how to load the Hadoop data into Oracle. Permitted values are JDBC, OCI, DP_COPY | DP_OSCH, and OSCH. • JDBC output mode: The data is inserted using a number of 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. • OCI output mode: The data is inserted using a number of direct insert OCI connections in direct path mode. For direct loading (no C$ table), the target table must be partitioned. For standard loading, FLOW_TABLE_OPTIONS must explicitely specify partitioning: e.g. 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. • DP_COPY output mode: OLH creates a number of 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. Enter the maximum number of errors allowed in the file. Examples: UNLIMITED to except all errors. Integer value (10 to allow 10 rejections). This value is used in OLH job definitions as well as in external table definitions.

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

Option	Description
EXT_TAB_DIR_LOCATION	Directory for ext tab data files. File system path of the external table. **Note:** • Only applicable, if OLH_OUTPUT_MODE = DP_* or OSCH • For OLH_OUTPUT_MODE = DP_*: this path must be accessible both from the ODI agent and from the target database engine. • For OLH_OUTPUT_MODE = DP_*: the name of the external directory object is the I$ table name. • For OLH_OUTPUT_MODE = DP_COPY: ODI agent will use hadoop-fs command to copy dp files into this directory. • For OLH_OUTPUT_MODE = DP_*|OSCH: this path will contain any external table log/bad/dsc files. • ODI agent will remove any files from this directory during clean up before launching OLH/OSCH.
WORK_TABLE_OPTIONS	Option for Flow table creation. Use this option to specify 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.
OVERRIDE_INPUTFORMAT	Class name of InputFormat. By default the InputFormat class is derived from the source DataStore/Technology (DelimitedTextInputFormat or HiveToAvroInputFormat). This option allows the user to specify the class name of a custom InputFormat. Default: <empty>. Cannot be used with OLH_OUTPUT_MODE=OSCH. 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 See KM option EXTRA_OLH_CONF_PROPERTIES for details on how to specify the regular expression.

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

Option	Description
EXTRA_OLH_CONF_PROPERTIES	Optional extra OLH properties. Allows adding extra parameters to OLH. E.g. for changing the default OLH date format: <property> <name>oracle.hadoop.loader.defaultDateFormat</name> <value>yyyy-MM-dd HH:mm:ss</value> </property> 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> Cannot be used with OLH_OUTPUT_MODE=OSCH Example (loading apache weblog file format): When OLH RegexInputFormat is used for reading custom file formats, this KM option specified the regular expression and other parsing details: <property> <name>oracle.hadoop.loader.input.regexPattern</name> <value>([^]*) ([^]*) ([^]*) (-|\[[^\]]*\]) ([^ \]*|\"[^\"]*\") (-|[0-9]*) (-|[0-9]*) (\".*?\") (\".*?\") (\".*?\")</value> <description>RegEx for Apache WebLog format</description> </property>"
MAPRED_OUTPUT_BASE_DIR	MapReduce Output Directory. 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. Directory used for storing temporary files like squoop script, stdout, and stderr redirects. 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. Check this option if you wish to create the target table.
TRUNCATE	Replace all target table data. Set this option to true, if you wish to replace the target table content with the new data.
DELETE_ALL	Delete all rows. 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. Set this option to NO, if you wish to retain temporary objects (tables, files and scripts) after integration. Useful for debugging.
OLH_OUTPUT_MODE	How to transfer data into Oracle? This option specifies how to load the Hadoop data into Oracle. Permitted values are JDBC, OCI, DP_COPY | DP_OSCH, and OSCH. • JDBC output mode: The data is inserted using a number of 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. • OCI output mode: The data is inserted using a number of direct insert OCI connections in direct path mode. For direct loading (no C$ table), the target table must be partitioned. For standard loading, FLOW_TABLE_OPTIONS must explicitely specify partitioning: e.g. 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. • DP_COPY output mode: OLH creates a number of 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. Enter the maximum number of errors allowed in the file. Examples: UNLIMITED to except all errors. Integer value (10 to allow 10 rejections). This value is used in OLH job definitions as well as 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. File system path of the external table. **Note:** • Only applicable, if OLH_OUTPUT_MODE = DP_* or OSCH • For OLH_OUTPUT_MODE = DP_*: this path must be accessible both from the ODI agent and from the target database engine. • For OLH_OUTPUT_MODE = DP_*: the name of the external directory object is the I$ table name. • For OLH_OUTPUT_MODE = DP_COPY: ODI agent will use hadoop-fs command to copy dp files into this directory. • For OLH_OUTPUT_MODE = DP_*|OSCH: this path will contain any external table log/bad/dsc files. • ODI agent will remove any files from this directory during clean up before launching OLH/OSCH.
WORK_TABLE_OPTIONS	Option for Flow table creation. Use this option to specify 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.
OVERRIDE_INPUTFORMAT	Class name of InputFormat. By default the InputFormat class is derived from the source DataStore/Technology (DelimitedTextInputFormat or HiveToAvroInputFormat). This option allows the user to specify the class name of a custom InputFormat. Default: <empty>. Cannot be used with OLH_OUTPUT_MODE=OSCH. 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 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. Allows adding extra parameters to OLH. E.g. for changing the default OLH date format: <property> <name>oracle.hadoop.loader.defaultDateFormat</name> <value>yyyy-MM-dd HH:mm:ss</value> </property> 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> Cannot be used with OLH_OUTPUT_MODE=OSCH Example (loading apache weblog file format): When OLH RegexInputFormat is used for reading custom file formats, this KM option specified the regular expression and other parsing details: <property> <name>oracle.hadoop.loader.input.regexPattern</name> <value>([^]*) ([^]*) ([^]*) (-|\[[^\]]*\]) ([^ \]*|\"[^\"]*\") (-|[0-9]*) (-|[0-9]*) (\".*?\") (\".*?\") (\".*?\")</value> <description>RegEx for Apache WebLog format</description> </property>"
MAPRED_OUTPUT_BASE_DIR	MapReduce Output Directory. 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. Directory used for storing temporary files like squoop script, stdout, and stderr redirects. 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? By default the Hive source data is getting materialized in a Hive staging table prior to extraction by OLH. If USE_HIVE_STAGING_TABLE is set to false, OLH directly accesses the Hive source data. USE_HIVE_STAGING_TABLE=0 is only possible, if all these conditions are true. • Only a single source table • No transformations, filters, joins. • No datasets • 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. Set this option to NO, if you wish to retain temporary objects (tables, files and scripts) after integration. Useful for debugging.
OLH_OUTPUT_MODE	How to transfer data into Oracle? This option specifies how to load the Hadoop data into Oracle. Permitted values are JDBC, OCI, DP_COPY | DP_OSCH, and OSCH. • JDBC output mode: The data is inserted using a number of 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. • OCI output mode: The data is inserted using a number of direct insert OCI connections in direct path mode. For direct loading (no C$ table), the target table must be partitioned. For standard loading, FLOW_TABLE_OPTIONS must explicitely specify partitioning: e.g. 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. • DP_COPY output mode: OLH creates a number of 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. Enter the maximum number of errors allowed in the file. Examples: UNLIMITED to except all errors. Integer value (10 to allow 10 rejections). This value is used in OLH job definitions as well as 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. File system path of the external table. **Note:** • Only applicable, if OLH_OUTPUT_MODE = DP_* or OSCH • For OLH_OUTPUT_MODE = DP_*: this path must be accessible both from the ODI agent and from the target database engine. • For OLH_OUTPUT_MODE = DP_*: the name of the external directory object is the I$ table name. • For OLH_OUTPUT_MODE = DP_COPY: ODI agent will use hadoop-fs command to copy dp files into this directory. • For OLH_OUTPUT_MODE = DP_*|OSCH: this path will contain any external table log/bad/dsc files. • ODI agent will remove any files from this directory during clean up before launching OLH/OSCH.
WORK_TABLE_OPTIONS	Option for Flow table creation. Use this option to specify 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.
OVERRIDE_INPUTFORMAT	Class name of InputFormat. By default the InputFormat class is derived from the source DataStore/Technology (DelimitedTextInputFormat or HiveToAvroInputFormat). This option allows the user to specify the class name of a custom InputFormat. Default: <empty>. Cannot be used with OLH_OUTPUT_MODE=OSCH. 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 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. Allows adding extra parameters to OLH. E.g. for changing the default OLH date format: <property> <name>oracle.hadoop.loader.defaultDateFormat</name> <value>yyyy-MM-dd HH:mm:ss</value> </property> 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> Cannot be used with OLH_OUTPUT_MODE=OSCH Example (loading apache weblog file format): When OLH RegexInputFormat is used for reading custom file formats, this KM option specified the regular expression and other parsing details: <property> <name>oracle.hadoop.loader.input.regexPattern</name> <value>([^]*) ([^]*) ([^]*) (-|\[[^\]]*\]) ([^ \]*|\"[^\"]*\") (-|[0-9]*) (-|[0-9]*) (\".*?\") (\".*?\") (\".*?\")</value> <description>RegEx for Apache WebLog format</description> </property>"
MAPRED_OUTPUT_BASE_DIR	MapReduce Output Directory. 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. Directory used for storing temporary files like squoop script, stdout, and stderr redirects. 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

Option	Description
CREATE_TARG_TABLE	Create target table. Check this option if you wish to create the target table.
TRUNCATE	Replace all target table data. Set this option to true, if you wish to replace the target table content with the new data.
DELETE_ALL	Delete all rows. Set this option to true, if you wish to replace the target table content with the new data.
USE_HIVE_STAGING_TABLE	Use intermediate Hive staging table? By default the Hive source data is getting materialized in a Hive staging table prior to extraction by OLH. If USE_HIVE_STAGING_TABLE is set to false, OLH directly accesses the Hive source data. USE_HIVE_STAGING_TABLE=0 is only possible, if all these conditions are true. • Only a single source table • No transformations, filters, joins. • No datasets • 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. Set this option to NO, if you wish to retain temporary objects (tables, files and scripts) after integration. Useful for debugging.
OLH_OUTPUT_MODE	How to transfer data into Oracle? This option specifies how to load the Hadoop data into Oracle. Permitted values are JDBC, OCI, DP_COPY | DP_OSCH, and OSCH. • JDBC output mode: The data is inserted using a number of 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. • OCI output mode: The data is inserted using a number of direct insert OCI connections in direct path mode. For direct loading (no C$ table), the target table must be partitioned. For standard loading, FLOW_TABLE_OPTIONS must explicitely specify partitioning: e.g. 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. • DP_COPY output mode: OLH creates a number of 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.

Table A-19 (Cont.) LKM Hive to Oracle OLH-OSCH

Option	Description
REJECT_LIMIT	Max number of errors for OLH/EXTTAB. Enter the maximum number of errors allowed in the file. Examples: UNLIMITED to except all errors. Integer value (10 to allow 10 rejections). This value is used in OLH job definitions as well as in external table definitions.
EXT_TAB_DIR_LOCATION	Directory for ext tab data files. File system path of the external table. **Note:** • Only applicable, if OLH_OUTPUT_MODE = DP_* or OSCH • For OLH_OUTPUT_MODE = DP_*: this path must be accessible both from the ODI agent and from the target database engine. • For OLH_OUTPUT_MODE = DP_*: the name of the external directory object is the I$ table name. • For OLH_OUTPUT_MODE = DP_COPY: ODI agent will use hadoop-fs command to copy dp files into this directory. • For OLH_OUTPUT_MODE = DP_*|OSCH: this path will contain any external table log/bad/dsc files. • ODI agent will remove any files from this directory during clean up before launching OLH/OSCH.
WORK_TABLE_OPTIONS	Option for Flow table creation. Use this option to specify 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.
OVERRIDE_INPUTFORMAT	Class name of InputFormat. By default the InputFormat class is derived from the source DataStore/Technology (DelimitedTextInputFormat or HiveToAvroInputFormat). This option allows the user to specify the class name of a custom InputFormat. Default: <empty>. Cannot be used with OLH_OUTPUT_MODE=OSCH. 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 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

Option	Description
EXTRA_OLH_CONF_PROPERTIES	Optional extra OLH properties. Allows adding extra parameters to OLH. E.g. for changing the default OLH date format: <property> <name>oracle.hadoop.loader.defaultDateFormat</name> <value>yyyy-MM-dd HH:mm:ss</value> </property> 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> Cannot be used with OLH_OUTPUT_MODE=OSCH Example (loading apache weblog file format): When OLH RegexInputFormat is used for reading custom file formats, this KM option specified the regular expression and other parsing details: <property> <name>oracle.hadoop.loader.input.regexPattern</name> <value>([^]*) ([^]*) ([^]*) (-|\[[^\]]*\]) ([^ \]*|\"[^\"]*\") (-|[0-9]*) (-|[0-9]*) (\".*?\") (\".*?\") (\".*?\")</value> <description>RegEx for Apache WebLog format</description> </property>"
MAPRED_OUTPUT_BASE_DIR	MapReduce Output Directory. 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. Directory used for storing temporary files like squoop script, stdout, and stderr redirects. 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, please see *Runtime Logging for ODI components* 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?)`. 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 DataStore (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`, please note that 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. 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. If this option is set to true: • All files in the directory given by the physical data schema will be loaded. So any filename or wildcard information from the source DataStore's resource name will be ignored. • The directory structure and file names must comply with Hives directory organization for tables, for example, for partitioning and clustering. • The directory and its files must reside in HDFS. • 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. A Hive staging table is required if: • Target table is partitioned, but data spreads across partitions • Target table is clustered • Target table (partition) is sorted, but input file is not • Target table is already defined and target table definition does not match the definition required by the KM • Target column order does not match source file column order • There are any unmapped source columns • There are any unmapped non-partition target columns • The source is a fixed length file and the target has non-string columns 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.

Table A-21 (Cont.) IKM File to Hive Options

Option	Description
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>. This option only applies if USE_STAGE_TABLE is set to true. 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 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.
OVERRIDE_ROW_FORMAT	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: `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` The list of columns in the source DataStore 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 DataStore must have as many columns as the SerDe returns columns. Only data of mapped columns will be transferred.
STOP_ON_FILE_NOT_FOUND	Defines whether the KM should stop, if input file is not found.

Table A-21 (Cont.) IKM File to Hive Options

Option	Description
HIVE_COMPATIBILE	Specifies the Hive version compatibility. The values permitted for this option are 0.7 and 0.8. • 0.7: Simulates the append behavior. Must be used for Hive 0.7 (CDH3). • 0.8: Uses Hive's append feature, which provides better performance. Requires Hive 0.8 (CDH4) or later.

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 if you want 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 if you want to retain the temporary objects, which might be useful for debugging.

Table A-23 (Cont.) IKM Hive to HBase Incremental Update (HBase-SerDe) Options

Option	Description
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.
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. 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.

Table A-24 (Cont.) IKM SQL to Hive-HBase-File (SQOOP) Options

Option	Description

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:
	`SELECT 1000, 2000 FROM DUAL`
	For preserving context independence, regular table names should be inserted through `odiRef.getObjectName` calls. For example:
	`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: `<?=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 if you want 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. Setting this option to `false` is only possible, if all these conditions are true: • All target columns are mapped • Existing Hive table uses standard hive row separators (\n) and column delimiter (\01) Setting this option to `false` provides better performance by avoiding an extra data transfer step. This option is applicable only if the target technology is Hive.
USE_GENERIC_JDBC_CONNECTOR	Specifies whether to use the generic JDBC connector if a connector for the target technology is not available. 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.

Table A-24 (Cont.) IKM SQL to Hive-HBase-File (SQOOP) Options

Option	Description
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` if you want 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. • 0.7: Simulates the append behavior. Must be used for Hive 0.7 (CDH3). • 0.8: Uses Hive's append feature, which provides better performance. Requires Hive 0.8 (CDH4) or later.

A.28 CKM Hive (Deprecated)

Note: This KM is deprecated and only used for backward compatibility.

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. • 0.7: Simulates the append behavior. Must be used for Hive 0.7 (CDH3). • 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 if you want 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:
	Local script location: `file:///tmp/odi/script1.pl`
	HDFS script location: `hdfs://namenode:nnPort/tmp/odi/script1.pl`
	Ensure that the following requirements are met:
	• 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.
	• 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.
	When the KM option `TRANSFORM_SCRIPT` is set, the following paragraphs provide some configuration help:
	• **For HDFS script locations:**
	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) as well as 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:
	`logon as hdfs user hdfs dfs -chown oracle:hadoop /tmp/odi/myscriptdir`
	• **For local script locations:**
	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:
	`chown oracle:hadoop /tmp/odi/myscriptdir`
	`chmod g+s /tmp/odi/myscriptdir`

Table A-27 (Cont.) IKM Hive Transform Options

Option	Description
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. Script example (1-to-1 transformation): `#! /usr/bin/csh -f cat` All mapped source columns are spooled as tab separated data into this script via 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.
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. **Warning:** Using wider permissions like 0777 poses a security risk. 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	Provide an optional, comma-separated list of source column names, which enables the knowledge module to sort the data before the transformation script is applied.
POST_TRANSFORM_DISTRIBUTE	Provides an optional, comma-separated list of target column names, which enables the knowledge module to distribute the data after the transformation script is applied.
POST_TRANSFORM_SORT	Provides an optional, comma-separated list of target column names, which enables the knowledge module to sort the data after the transformation script is applied.

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 . 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 requirements.

See Also:

- "*Oracle Loader for Hadoop Setup*" in 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. • **JDBC output mode:** The data is inserted using a number of 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. • **OCI output mode:** The data is inserted using a number of direct insert OCI connections in direct path mode. If USE_ORACLE_STAGINGis 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. • **DP_COPY output mode:** OLH creates a number of 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, an external table is defined in the target database, which accesses the files from EXT_TAB_DIR_LOCATION. • **DP_OSCH output mode:** OLH creates a number of DataPump export files. After the export phase an external table is created on the target database, which accesses these output files directly via OSCH. Please note that 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. • **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 and external table. Examples: UNLIMITED to except all errors. Integer value (10 to allow 10 rejections) This value is used in job definitions as well as in external table definitions.

Table A-28 (Cont.) IKM File - Hive to Oracle (OLH-OSCH)

Option	Description
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 . If this option is set to `false`, directly accesses the Hive source data. Setting this option to `false` is only possible, if all these conditions are true: • Only a single source table • No transformations, filters, joins • No datasets Setting this option to `false` provides better performance by avoiding an extra data transfer step. This option is applicable only if the source technology is Hive.
USE_ORACLE_STAGING_TABLE	Uses an intermediate Oracle database staging table. 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: • `OLH_OUTPUT_MODE` is set to `JDBC` or `OCI` • All source columns are mapped • All target columns are mapped • No target-side mapping expressions 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 loads data in multiple transactions.
EXT_TAB_DIR_LOCATION	Specifies the file system path of the external table. Please note the following: • Only applicable, if `OLH_OUTPUT_MODE` = `DP_*|OSCH` • For `OLH_OUTPUT_MODE` = `DP_*`: this path must be accessible both from the ODI agent and from the target database engine. • For `OLH_OUTPUT_MODE` = `DP_*`: the name of the external directory object is the I$ table name. • For `OLH_OUTPUT_MODE` = `DP_COPY`: ODI agent will use `hadoop-fs` command to copy dp files into this directory. • For `OLH_OUTPUT_MODE` = `DP_*|OSCH`: this path will contain any external table log/bad/dsc files. • 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: `<?=System.getProperty("java.io.tmp")?>`

Table A-28 (Cont.) IKM File - Hive to Oracle (OLH-OSCH)

Option	Description
MAPRED_OUTPUT_BASE_DIR	Specifies an HDFS directory, where the 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 if you want to retain the temporary objects, which might be useful for debugging.
OVERRIDE_INPUTFORMAT	By default the InputFormat class is derived from the source DataStore/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. 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 See KM option EXTRA_OLH_CONF_PROPERTIES for details on how to specify the regular expression.
EXTRA_OLH_CONF_PROPERTIES	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. 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: <property> <name>oracle.hadoop.loader.input.regexPattern</name> <value>([^]*) ([^]*) ([^]*) (-|\[[^\]]*\]) ([^ \"]*|\"[^\"]*\") (-|[0-9]*) (-|[0-9]*) (\".*?\") (\".*?\") (\".*?\")</value> <description>RegEx for Apache WebLog format</description> </property>

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 datastore content with new data. If this option is set to `false`, the new data is appended to the target datastore.
DELETE_ALL	Deletes all the rows in the target datastore.
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.
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. For File sources setting this option to `false` is only possible, if all these conditions are met: • All source columns must be mapped • Source and target columns have same order • First file column must map to first target column • no mapping gaps • only 1-to-1 mappings (no expressions) Please note the following: • 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. • 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. 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.

Table A-29 (Cont.) IKM File-Hive to SQL (SQOOP)

Option	Description
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: • For Oracle: NOLOGGING • For DB2: NOT LOGGED INITIALLY • For Teradata: no fallback, no before journal, no after journal 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: <?=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 if you want to retain the temporary objects, which might be useful for debugging.
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. This option is applicable only to Teradata targets.
TERADATA_FLOW_TABLE_TYPE	Type of the Teradata flow table, either SET or MULTISET. This option is applicable only to Teradata targets.
TERADATA_OUTPUT_METHOD	Specifies the way the Teradata Connector will load the data. Valid values are: • batch.insert: multiple JDBC connections using batched prepared statements (simplest to start with) • multiple.fastload: multiple FastLoad connections • internal.fastload: single coordinated FastLoad connections (most performant) 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 chapter includes the following sections:

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. Select the storage function to be used to load data.
Schema for Complex Fields	The pig schema for simple/complex fields separated by comma (,). 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 datastore.
Function Class	Fully qualified name of the class to be used as storage function to load data. 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. Specify the parameters that the loader function expects. For example, the XMLLoader function may look like XMLLoader('MusicStore', 'movie', 'id:double, name:chararray, director:chararry', options) Here the first three arguments are parameters, which can be specified as -rootElement MovieStore -tableName movie -schema where, MusicStore - the root element of the xml movie - The element that wraps the child elements such as id, name, etc. Third Argument is the representation of data in pig schema. The names of the parameters are arbitrary and there can be any number of parameters.

Table B-1 (Cont.) LKM File to Pig

Option	Description
Options	Additional options required for the storage function Specify additional options required for the storage function. For example, the XMLLoader function may look like XMLLoader('MusicStore', 'movie', 'id:double, name:chararray, director:chararry', options) The last argument options can be specified as -namespace com.imdb -encoding utf8
Jars	The jar containing the storage function class and dependant libraries separated by colon (:). Specify the jar containing the storage function class and dependant libraries separated by colon (:).
Storage Convertor	The converter that provides functions to cast from bytearray to each of Pig's internal types. Specify the converter that provides functions to cast from bytearray to each of Pig's internal types. 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. 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.

Table B-2 (Cont.) LKM Pig to File

Option	Description
Record Name	The Avro record name to be assigned to the bag of tuples being stored. 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. Specify a namespace for the bag of tuples being stored.
Delete Target File	Delete target file before Pig writes to the file. 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. 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. Specify the parameters that the loader function expects. For example, the XMLLoader function may look like XMLLoader('MusicStore', 'movie', 'id:double, name:chararray, director:chararry', options) Here the first three arguments are parameters, which can be specified as -rootElement MovieStore -tableName movie -schema where, MusicStore - the root element of the xml movie - The element that wraps the child elements such as id, name, etc. Third Argument is the representation of data in pig schema. The names of the parameters are arbitrary and there can be any number of parameters.
Options	Additional options required for the storage function Specify additional options required for the storage function. For example, the XMLLoader function may look like XMLLoader('MusicStore', 'movie', 'id:double, name:chararray, director:chararry', options) The last argument options can be specified as -namespace com.imdb -encoding utf8
Jars	The jar containing the storage function class and dependant libraries separated by colon (:). Specify the jar containing the storage function class and dependant libraries separated by colon (:).
Storage Convertor	The converter that provides functions to cast from bytearray to each of Pig's internal types. Specify the converter that provides functions to cast from bytearray to each of Pig's internal types. 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. 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. 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.
Greater Than Min Key	Loads rows with key greater than the key specified for this option. 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. 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. 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. 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 Specify the maximum number of rows to retrieve per region.
Cached Rows	Number of rows to cache. Specify the number of rows to cache.
Storage Convertor	The name of Caster to use to convert values. 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. Specify the delimiter to be used to separate columns in the columns list of HBaseStorage function. If unspecified, the default is whitespace.

Table B-3 (Cont.) LKM HBase to Pig

Option	Description
Timestamp	Return cell values that have a creation timestamp equal to this value. 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. 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. 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 can not be changed. HBaseStore function is used to load data into hbase table.
Storage Convertor	The name of Caster to use to convert values. 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. 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. 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 can not be changed. 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 can not be changed. HCatStorer is used to store data into a hive table.
Partition	The new partition to be created. 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 File to Pig

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. Select the storage function to be used to load data.

Table B-7 (Cont.) LKM File to Pig

Option	Description
Schema for Complex Fields	The pig schema for simple/complex fields separated by comma (,). 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 datastore.
Function Class	Fully qualified name of the class to be used as storage function to load data. 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. Specify the parameters that the loader function expects. For example, the XMLLoader function may look like XMLLoader('MusicStore', 'movie', 'id:double, name:chararray, director:chararry', options) Here the first three arguments are parameters, which can be specified as -rootElement MovieStore -tableName movie -schema where, MusicStore - the root element of the xml movie - The element that wraps the child elements such as id, name, etc. Third Argument is the representation of data in pig schema. The names of the parameters are arbitrary and there can be any number of parameters.
Options	Additional options required for the storage function. Specify additional options required for the storage function. For example, the XMLLoader function may look like XMLLoader('MusicStore', 'movie', 'id:double, name:chararray, director:chararry', options) The last argument options can be specified as -namespace com.imdb -encoding utf8
Jars	The jar containing the storage function class and dependant libraries separated by colon (:). Specify the jar containing the storage function class and dependant libraries separated by colon (:).
Storage Convertor	The converter that provides functions to cast from bytearray to each of Pig's internal types. Specify the converter that provides functions to cast from bytearray to each of Pig's internal types. 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 needs to 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, e.g. rating_table(obj_rating('-1', 'Unknown')). 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 chapter includes the following sections:

- 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.

The following tables describes 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.
CACHE_DATA	Persist the data with the default storage level.
InputFormatClass	Classname of Hadoop InputFormat. For example, org.apache.hadoop.mapreduce.lib.input.TextInputFormat.
KeyClass	Fully qualified classname of key Writable class. For example, org.apache.hadoop.io.Text.
ValueClass	Fully qualified classname of value Writable class. For example, org.apache.hadoop.io.LongWritable.
KeyConverter	Fully qualified classname of key converter class.
ValueConverter	Fully qualified classname of value converter class.
Job Configuration	Hadoop configuration. For example, {'hbase.zookeeper.quorum': 'HOST', 'hbase.mapreduce.inputtable': 'TAB'}

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
STREAMING_MODE	This option indicates whether the mapping should be executed in streaming mode. Default is FALSE.
Storage Function	If STREAMING_MODE is set to true, the load function is changed to textFileStream automatically. 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.

The following tables describes the options for LKM Spark to File.

Table C-3 LKM Spark to File

Option	Description
Storage Function	The storage function to be used to load/store data.
InputFormatClass	Classname of Hadoop InputFormat. For example, org.apache.hadoop.mapreduce.lib.input.TextInputFormat.
KeyClass	Fully qualified classname of key Writable class. For example, org.apache.hadoop.io.Text.
ValueClass	Fully qualified classname of value Writable class. For example, org.apache.hadoop.io.LongWritable.
KeyConverter	Fully qualified classname of key converter class.
ValueConverter	Fully qualified classname of value converter class.
Job Configuration	Hadoop configuration. For example, {'hbase.zookeeper.quorum': 'HOST', 'hbase.mapreduce.inputtable': 'TAB'}

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. 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.

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 tables describes the options for LKM Spark to Hive.

Table C-5 LKM Spark to Hive

Option	Description
CREATE_TARGET_TABLE	Create the target table.
OVERWRITE_TARGET_TABLE	Overwrite the target table.

C.5 LKM HDFS to Spark

This KM will load data from HDFS file to Spark.

Table C-6 LKM HDFS to Spark

Option	Description
Storage Function	The storage function is used to load or store data.
streamingContext	Name of the Streaming context.
InputFormatClass	Classname of Hadoop InputFormat. For example, org.apache.hadoop.mapreduce.lib.input.TextInputFormat.
KeyClass	Fully qualified classname of key Writable class. For example, org.apache.hadoop.io.Text.
ValueClass	Fully qualified classname of value Writable class. For example, org.apache.hadoop.io.LongWritable.
KeyConverter	Fully qualified classname of key converter class.
ValueConverter	Fully qualified classname of value converter class.
Job Configuration	Hadoop configuration. For example, {'hbase.zookeeper.quorum': 'HOST', 'hbase.mapreduce.inputtable': 'TAB'}
Delete Spark Mapping Files	Delete temporary objects at the end of mapping.
Cache	Cache RDD across operations after computation.
Storage Level	The storage level is used to cache data.
Repartition	Repartiton the RDD after transformation of this component.
Level of Parallelism	Number of partitions.
Sort Partitions	Sort partitions by a key function when you repartition RDD.
Partition Sort Order	Sort partition order
Partition Key Function	Define keys of partition.

Table C-6 (Cont.) LKM HDFS to Spark

Option	Description
Partition Function	Customized partitioning function.

C.6 LKM Spark to HDFS

This KM will load data from Spark to HDFS file.

Table C-7 LKM Spark to HDFS

Option	Description
Storage Function	The storage function is used to load or store data.
OutputFormatClass	Class name of Hadoop Input Format.
KeyClass	Fully qualified classname of key Writable class. For example, org.apache.hadoop.io.Text.
ValueClass	Fully qualified classname of value Writable class. For example, org.apache.hadoop.io.LongWritable.
KeyConverter	Fully qualified classname of key converter class.
ValueConverter	Fully qualified classname of value converter class.
Job Configuration	Hadoop configuration. For example, {'hbase.zookeeper.quorum': 'HOST', 'hbase.mapreduce.inputtable': 'TAB'}
DELETE_TEMPORARY_OBJECTS	Delete temporary objects at the end of mapping.
Delete Spark Mapping Files	Delete temporary objects at the end of mapping.
Cache	Cache RDD across operations after computation.
Storage Level	The storage level is used to cache data.
Repartition	Repartiton the RDD after transformation of this component.
Level of Parallelism	Number of partitions.
Sort Partitions	Sort partitions by a key function when you repartition RDD.
Partition Sort Order	Sort partition order
Partition Key Function	Define keys of partition.
Partition Function	Customized partitioning function.

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-8 LKM Kafka to Spark for streaming

Option	Description
Storage Function	If STREAMING_MODE is set to true, the load function is changed to textFileStream automatically. Default is createStream.
Key Decoder	Decodes the message key. Default is empty.
Value Decoder	Decodes the message value. Default is empty.
Group Id	Receiver group id parameter used to call the KafkaUtils.createStream. **Note:** In a group of receivers (all receivers having the same Group Id) every message will be received by a single receiver only.
Kakfa Params	Parameter used to call KafkaUitls.createStream.
storageLevel	Storage level is used for storing the received objects. Default is StorageLevel.MEMORY_AND_DISK_2.
Number of Partitions	Number of partition each thread gets data from Kafka
From Offsets	Parameter used in conjunction with createDirectStream 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-9 LKM Spark to Kafka

Option	Description
value.serializer	org.apache.kafka.common.serialization.StringSerializer

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.

Table C-10 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.

C.10 LKM Spark to SQL

This KM will load data from Spark into a Cassandra table and can be defined on the AP node that have Spark source and SQL target. It can work with other JDBC targets.

Table C-11 LKM Spark to SQL

Option	Description
CREATE_TARG_TABLE	Create target table.

C.11 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.12 XKM Spark Aggregate

Summarize rows, for example, using SUM and GROUP BY.

The following tables describes the options for XKM Spark Aggregate.

Table C-12 XKM Spark Aggregate

Option	Description
CACHE_DATA	Persist the data with the default storage level.

Table C-12 (Cont.) XKM Spark Aggregate

Option	Description
NUMBER_OF_TASKS	Task number.

Table C-13 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.13 XKM Spark Distinct

Eliminates duplicates in data and functionality is identical to the existing batch processing.

C.14 XKM Spark Expression

Define expressions to be reused across a single mapping.

C.15 XKM Spark Filter

Produce a subset of data by a filter condition.

The following tables describes the options for XKM Spark Filter.

Table C-14 XKM Spark Filter

Option	Description
CACHE_DATA	Persist the data with the default storage level.

C.16 XKM Spark Input Signature and Output Signature

Supports code generation for reusable mapping.

C.17 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-15 XKM Spark Join

Option	Description
CACHE_DATA	Persist the data with the default storage level.
NUMBER_OF_TASKS	Task number.

C.18 XKM Spark Lookup

Lookup data for a driving data source.

The following tables describes the options for XKM Spark Lookup.

Table C-16 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-17 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. • If the lookup record is present and not expired, the lookup data is served from the cache. • If the lookup record is missing or expired, the data is re-loaded from the SQL source.

Table C-17 (Cont.) XKM Spark Lookup for streaming

Option	Description
CACHE_RELOAD	This option defines when the lookup source data is loaded and refreshed and here are the corresponding values: • NO_RELOAD: The lookup source data is loaded once on Spark application startup. • RELOAD_EVERY_BATCH: The lookup source data is reloaded for every new Spark batch. • 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.19 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-18 XKM Spark Pivot

Option	Description
CACHE_DATA	Persist the data with the default storage level.

Note: XKM Spark Pivot does not support streaming.

C.20 XKM Spark Set

Perform UNION, MINUS or other set operations.

C.21 XKM Spark Sort

Sort data using an expression.

The following tables describes the options for XKM Spark Sort.

Table C-19 XKM Spark Sort

Option	Description
CACHE_DATA	Persist the data with the default storage level.
NUMBER_OF_TASKS	Task number.

C.22 XKM Spark Split

Split data into multiple paths with multiple conditions.

The following tables describes the options for XKM Spark Split.

Table C-20 XKM Spark Split

Option	Description
CACHE_DATA	Persist the data with the default storage level.

C.23 XKM Spark Table Function

Spark table function access.

The following tables describes the options for XKM Spark Table Function.

Table C-21 XKM 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.24 IKM Spark Table Function

Spark table function as target.

The following tables describes the options for IKM Spark Table Function.

Table C-22 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.25 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-23 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 chapter includes the following sections:

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. 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. Default is XMLT.
DEFAULT_EXPRESSION	Default expression for null XMLTYPE objects. For example, <row> < attribute1/><row/> 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')). 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. 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, i.e. 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 labelled 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.
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 sections:

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 as long as 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, as well as 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 via 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 a 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.

Index