



Work with Big Data



This document supports Pentaho Business Analytics Suite 5.0 GA and Pentaho Data Integration 5.0 GA, documentation revision August 28, 2013, copyright © 2013 Pentaho Corporation. No part may be reprinted without written permission from Pentaho Corporation. All trademarks are the property of their respective owners.

Help and Support Resources

If you do not find answers to your questions here, please contact your Pentaho technical support representative.

Support-related questions should be submitted through the Pentaho Customer Support Portal at <http://support.pentaho.com>.

For information about how to purchase support or enable an additional named support contact, please contact your sales representative, or send an email to sales@pentaho.com.

For information about instructor-led training, visit <http://www.pentaho.com/training>.

Liability Limits and Warranty Disclaimer

The author(s) of this document have used their best efforts in preparing the content and the programs contained in it. These efforts include the development, research, and testing of the theories and programs to determine their effectiveness. The author and publisher make no warranty of any kind, express or implied, with regard to these programs or the documentation contained in this book.

The author(s) and Pentaho shall not be liable in the event of incidental or consequential damages in connection with, or arising out of, the furnishing, performance, or use of the programs, associated instructions, and/or claims.

Trademarks

Pentaho (TM) and the Pentaho logo are registered trademarks of Pentaho Corporation. All other trademarks are the property of their respective owners. Trademarked names may appear throughout this document. Rather than list the names and entities that own the trademarks or insert a trademark symbol with each mention of the trademarked name, Pentaho states that it is using the names for editorial purposes only and to the benefit of the trademark owner, with no intention of infringing upon that trademark.

Third-Party Open Source Software

For a listing of open source software used by each Pentaho component, navigate to the folder that contains the Pentaho component. Within that folder, locate a folder named licenses. The licenses folder contains HTML files that list the names of open source software, their licenses, and required attributions.

Contact Us

Global Headquarters Pentaho Corporation
Citadel International, Suite 340
5950 Hazelhine National Drive
Orlando, FL 32822
Phone: +1 407 812-OPEN (6736)
Fax: +1 407 517-4575
<http://www.pentaho.com>
Sales Inquiries: sales@pentaho.com

Contents

| | |
|---|----|
| Getting Started with PDI and Hadoop..... | 4 |
| Pentaho, Big Data, and Hadoop..... | 4 |
| About Hadoop..... | 4 |
| Big Data Resources..... | 4 |
| Configuring Your Big Data Environment..... | 6 |
| Setting the Active Hadoop Configuration..... | 6 |
| Define Hadoop Connections..... | 6 |
| Configuring for MapR..... | 8 |
| Creating a New Hadoop Configuration..... | 8 |
| Working with Big Data and Hadoop in PDI..... | 9 |
| Pentaho MapReduce Workflow..... | 9 |
| PDI Hadoop Job Workflow..... | 11 |
| Hadoop to PDI Data Type Conversion..... | 12 |
| Hadoop Hive-Specific SQL Limitations..... | 13 |
| Big Data Tutorials..... | 13 |
| Hadoop Tutorials..... | 13 |
| MapR Tutorials..... | 21 |
| Cassandra Tutorials..... | 22 |
| MongoDB Tutorials..... | 22 |
| PDI Hadoop Configurations..... | 24 |
| Including/Excluding Classes or Packages for a Hadoop Configuration..... | 24 |
| PDI Big Data Transformation Steps..... | 26 |
| Avro Input..... | 26 |
| Cassandra Input..... | 27 |
| Cassandra Output..... | 28 |
| CouchDB Input..... | 30 |
| Hadoop File Input..... | 30 |
| Hadoop File Output..... | 35 |
| HBase Input..... | 38 |
| HBase Output..... | 40 |
| HBase Row Decoder..... | 42 |
| MapReduce Input..... | 43 |
| MapReduce Output..... | 43 |
| MongoDB Input..... | 44 |
| MongoDB Output..... | 48 |
| Splunk Input..... | 51 |
| Splunk Output..... | 53 |
| SSTable Output..... | 54 |
| PDI Big Data Job Entries..... | 55 |
| Amazon EMR Job Executor..... | 55 |
| Amazon Hive Job Executor..... | 55 |
| Hadoop Copy Files..... | 56 |
| Hadoop Job Executor..... | 57 |
| Oozie Job Executor..... | 59 |
| Pentaho MapReduce..... | 60 |
| Pig Script Executor..... | 62 |
| Sqoop Export..... | 63 |
| Sqoop Import..... | 64 |

Getting Started with PDI and Hadoop

Pentaho provides a complete big data analytics solution that supports the entire big data analytics process. From big data aggregation, preparation, and integration, to interactive visualization, analysis, and prediction, Pentaho allows you to harvest the meaningful patterns buried in big data stores. Analyzing your big data sets gives you the ability to identify new revenue sources, develop loyal and profitable customer relationships, and run your organization more efficiently and cost effectively.

Pentaho, Big Data, and Hadoop

The term big data applies to very large, complex, or dynamic datasets that need to be stored and managed over a long time. To derive benefits from big data, you need the ability to access, process, and analyze data as it is being created. However, the size and structure of big data makes it very inefficient to maintain and process it using traditional relational databases.

Big data solutions re-engineer the components of traditional databases—data storage, retrieval, query, processing—and massively scales them.

Pentaho Big Data Overview

Pentaho increases speed-of-thought analysis against even the largest of big data stores by focusing on the features that deliver performance.

- **Instant access**—Pentaho provides visual tools to make it easy to define the sets of data that are important to you for interactive analysis. These data sets and associated analytics can be easily shared with others, and as new business questions arise, new views of data can be defined for interactive analysis.
- **High performance platform**—Pentaho is built on a modern, lightweight, high performance platform. This platform fully leverages 64-bit, multi-core processors and large memory spaces to efficiently leverage the power of contemporary hardware.
- **Extreme-scale, in-memory caching**—Pentaho is unique in leveraging external data grid technologies, such as Infinispan and Memcached to load vast amounts of data into memory so that it is instantly available for speed-of-thought analysis.
- **Federated data integration**—Data can be extracted from multiple sources, including big data and traditional data stores, integrated together and then flowed directly into reports, without needing an enterprise data warehouse or data mart.

About Hadoop

The Apache Hadoop software library is a framework that allows for the distributed processing of large data sets across clusters of computers using simple programming models. It is designed to scale up from single servers to thousands of machines, each offering local computation and storage. Rather than rely on hardware to deliver high-availability, the library itself is designed to detect and handle failures at the application layer, so delivering a highly-available service on top of a cluster of computers, each of which may be prone to failures.

A Hadoop platform consists of a Hadoop kernel, a [MapReduce](#) model, a distributed file system, and often a number of related projects—such as [Apache Hive](#), [Apache HBase](#), and others.

A Hadoop Distributed File System, commonly referred to as HDFS, is a Java-based, distributed, scalable, and portable file system for the Hadoop framework.

Big Data Resources

- [Pentaho Big Data Analytics Center](#)
- [Pentaho Big Data Wiki](#)
- [Apache Hadoop project](#) -- A project that contains libraries that allows for the distributed processing of large data sets across clusters of computers using simple programming models. There are several modules, including the [Hadoop Distributed File System \(HDFS\)](#), which is a distributed file system that provides high-throughput access to application data and [Hadoop MapReduce](#), which is a key algorithm to distribute work around a cluster.

- **Avro**—A data serialization system
- **Cassandra**—A scalable multi-master database with no single points of failure
- **HBase**—A scalable, distributed database that supports structured data storage for large tables
- **Hive**—A data warehouse infrastructure that provides data summarization and on-demand querying
- **Pig**—A high-level, data-flow language and execution framework for parallel computation
- **ZooKeeper**—A high-performance coordination service for distributed applications
- **MongoDB**— A NoSQL open source document-oriented database system developed and supported by 10gen
- **Splunk** - A data collection, visualization and indexing engine for operational intelligence that is developed by Splunk, Inc.
- **CouchDB**—A NoSQL open source document-oriented database system developed and supported by Apache
- **Sqoop**—Software for transferring data between relational databases and Hadoop
- **Oozie**—A workflow scheduler system to manage Hadoop jobs

Configuring Your Big Data Environment

This section covers configuring PDI to communicate with Hadoop distributions other than the default configuration, Apache Hadoop 0.20.X. A list of supported big data technologies can be found in the *PDI Installation Guide*, [Compatibility Matrix](#).

Setting the Active Hadoop Configuration

Hadoop configurations within PDI are collections of the Hadoop libraries required to communicate with a specific version of Hadoop and Hadoop-related tools, such as Hive, HBase, Sqoop, or Pig.

The Kettle client comes pre-configured for Apache Hadoop 0.20.X, no further configuration is required in order to use PDI with this version of Hadoop.

Locating Which Version of Hadoop Is In Use

The Hadoop distribution configuration can be found at this location: `plugins/pentaho-big-data-plugin/plugin.properties`. In that file, locate this statement.

```
# The Hadoop Configuration to use when communicating with a Hadoop cluster.
# This is used for all Hadoop client tools including HDFS, Hive, HBase, and Sqoop.
active.hadoop.configuration=hadoop-20
```

The property `active.hadoop.configuration` sets the distribution of Hadoop to use when communicating with a Hadoop cluster. This is the property to update if you are using a version other than Apache Hadoop 0.20.X. For future releases, see the [Compatibility Matrix](#) for information as new configurations are certified.



Note: The `active.hadoop.configuration` property value matches the Hadoop distribution in use. For instance, if the value is set to `hadoop-20`, Pentaho will use the libraries located in the `hadoop-20` folder.

Define Hadoop Connections

You can connect PDI to several Hadoop distributions. We have pre-configured PDI for the default distribution, Apache Hadoop 0.20.X. If you are using that distribution, no further configuration is required.

If you are using another distribution of Hadoop, you must perform additional configuration tasks. We have collections of Hadoop libraries within PDI that create connections with specific versions of Hadoop and Hadoop-related tools, such as Hive, HBase, Sqoop, or Pig.

For all distributions, you must review the [Supported Technologies](#) to ensure your work environment is compliant with the system components and versions that are officially supported by Pentaho.

Locate the `plugin.properties` File

To update your Hadoop version, you have to edit the `plugin.properties` file.

1. Find `plugins/pentaho-big-data-plugin/plugin.properties`.
2. Locate this statement.

```
# The Hadoop Configuration to use when communicating with a Hadoop cluster.
# This is used for all Hadoop client tools including HDFS, Hive, HBase, and Sqoop.
active.hadoop.configuration=hadoop-20
```

3. The `active.hadoop.configuration` property matches the Hadoop distribution that is in use. For example, if the value is set to `hadoop-20`, PDI will use the libraries located in the `hadoop-20` folder.

Once you have located this file, update the `plugin.properties` file so that PDI is locating your Hadoop distribution.

- [Connect to Cloudera](#)
- [Connect to MapR](#)
- [Create a New Hadoop Connection](#)

Connect to Cloudera

1. After [locating the `plugin.properties` file](#), update it to match your distribution of Cloudera.
2. Find the value to which you must change your `plugin.properties` file in [Configuring Pentaho Hadoop for your Hadoop Distro and Version](#).
3. Edit the `active.hadoop.configuration` property with the appropriate value. For example, if your current value is `active.hadoop.configuration=hadoop-20`, and the value you find is `ABC123`, edit the property to look like this.
`active.hadoop.configuration=hadoop-ABC123`

Once you have modified the properties, restart Spoon.

Connect to MapR

After [locating the `plugin.properties` file](#), update it to match your distribution of MapR. The MapR client tools must be installed on the server and each workstation.

For All Operating Systems

1. After locating the properties file, update the `active.hadoop.configuration` property to **mapr**.
`active.hadoop.configuration=mapr`
2. Open `pentaho\design-tools\data-integration\plugins\pentaho-big-data-plugin\hadoop-configurations\mapr\config.properties` with a text editor. Remove the existing `classpath` and `library.path` properties by deleting these lines.

```
classpath=/opt/mapr/conf,/opt/mapr/hadoop/hadoop-0.20.2/lib
```

```
library.path=/opt/mapr/lib
```

For Windows

3. Uncomment the Windows-specific `classpath` and `library.path` properties and edit them to match your local MapR client tools installation directory. These are the Windows-specific lines that you must uncomment and edit.

```
classpath=file:///C:/opt/mapr/conf,file:///C:/opt/mapr/mapr-client-1.2.9.14720GA-0.amd64/hadoop/hadoop-0.20.2/lib
```

```
library.path=C:\\opt\\mapr\\mapr-client-1.2.9.14720GA-0.amd64\\lib
```

Once you have modified the properties, restart Spoon.

Create a New Hadoop Connection

If you have a Hadoop distribution not supported by Pentaho, or have modified your Hadoop distribution in such a way that it is no longer compatible with Pentaho, you may need to create a new Hadoop connection. You can verify if your Hadoop distribution is supported by reviewing [Configuring Pentaho for your Hadoop Distro and Version](#).

If you are changing the version of Hadoop, you must swap the appropriate `.jar` files within the plugin directory and then update the `plugin.properties` file.



Caution: Creating a new Hadoop configuration is not supported by Pentaho. Please contact [Pentaho support](#) to for assistance with meeting your requirements.

1. Identify which Hadoop configuration most closely matches the version of Hadoop to which you want to connect.
2. Copy the folder you selected, then paste and rename it. The name of this folder will be the name of your new configuration.
3. Copy the `.jar` files for your specified Hadoop version.
4. Delete the original `.jar` files and paste the recently copied `.jar` files into the `lib/` directory.
5. [Locate the `plugin.properties` file](#) and edit it to match your specific Hadoop configuration. This property configures which distribution of Hadoop to use when communicating with a Hadoop cluster and must match the name of the folder you created.

Once you have modified the properties, restart Spoon.

Configuring for MapR

To communicate with MapR, the MapR client tools must be installed on the local machine.

For All Operating Systems

1. Within the file `plugins/pentaho-big-data-plugin/plugin.properties`, the `active.hadoop.configuration` property identifies which folder to use with the active Hadoop configuration. Change this property to **mapr**.
2. Within the file `pentaho\design-tools\data-integration\plugins\pentaho-big-data-plugin\hadoop-configurations\mapr\config.properties`. Remove the existing `classpath` and `library.path` properties by deleting

```
classpath=/opt/mapr/conf,/opt/mapr/hadoop/hadoop-0.20.2/lib
```

```
library.path=/opt/mapr/lib
```

For Windows

3. Uncomment the Windows-specific `classpath` and `library.path` properties and edit them to match your local MapR client tools installation directory. Once completed, the properties should look like this.

```
classpath=file:///C:/opt/mapr/conf,file:///C:/opt/mapr/mapr-client-1.2.9.14720GA-0.amd64/hadoop/hadoop-0.20.2/lib
```

```
library.path=C:\\opt\\mapr\\mapr-client-1.2.9.14720GA-0.amd64\\lib
```

Creating a New Hadoop Configuration

If you have a Hadoop distribution not supported by Pentaho, or you have modified your Hadoop Installation in such a way that it is no longer compatible with Pentaho, you may need to create a new Hadoop configuration.

Changing which version of Hadoop PDI can communicate with requires you to swap the appropriate `jar` files within the plugin directory and then update the `plugin.properties` file.



Caution: Creating a new Hadoop configuration is not officially supported by Pentaho. Please inform Pentaho support regarding your requirements.

1. Identify which Hadoop configuration most closely matches the version of Hadoop you want to communicate with. If you compare the default configurations included the differences are apparent. Copy this folder, then paste and rename it. The name of this folder will be the name of your new configuration.
2. Copy the `jar` files for your specified Hadoop version.
3. Paste the `jar` files into the `lib/` directory.
4. Change the `active.hadoop.configuration=` property in the `plugins/pentaho-big-dataplugin/plugin.properties` file to match your specific Hadoop configuration. This property configures which distribution of Hadoop to use when communicating with a Hadoop cluster and must match the name of the folder you created in Step 1. Update this property if you are using a version other than the default Hadoop version.

Working with Big Data and Hadoop in PDI

Pentaho Data Integration (PDI) can operate in two distinct modes, job orchestration and data transformation. Within PDI they are referred to as jobs and transformations.

PDI jobs sequence a set of entries that encapsulate actions. An example of a PDI big data job would be to check for existence of new log files, copy the new files to HDFS, execute a MapReduce task to aggregate the weblog into a click stream and stage that clickstream data in an analytic database.

PDI transformations consist of a set of steps that execute in parallel and operate on a stream of data columns. The columns usually flow from one system, through the PDI engine, where new columns can be calculated or values can be looked up and added to the stream. The data stream is then sent to a receiving system like a Hadoop cluster, a database, or even the Pentaho Reporting Engine.

The tutorials within this section illustrate how to use PDI jobs and transforms in typical big data scenarios. PDI job entries and transformation steps are described in the [Transformation Step Reference](#) and [Job Entry Reference](#) sections of Administer the DI Server.

PDI's Big Data Plugin

The Pentaho Big Data plugin contains all of the job entries and transformation steps required for working with Hadoop, Cassandra, and MongoDB.

By default, PDI is pre-configured to work with Apache Hadoop 0.20.X. But PDI can be configured to communicate with most popular Hadoop distributions. Instructions for changing Hadoop configurations are covered in the [Configure Your Big Data Environment](#) section.

For a list of supported big data technology, including which configurations of Hadoop are currently supported, see the section on [Supported Components](#).

Using PDI Outside and Inside the Hadoop Cluster

PDI is unique in that it can execute both outside of a Hadoop cluster and within the nodes of a hadoop cluster. From outside a Hadoop cluster, PDI can extract data from or load data into Hadoop HDFS, Hive and HBase. When executed within the Hadoop cluster, PDI transformations can be used as Mapper and/or Reducer tasks, allowing PDI with Pentaho MapReduce to be used as visual programming tool for MapReduce.

These videos demonstrate using PDI to work with Hadoop from both inside and outside a Hadoop cluster.

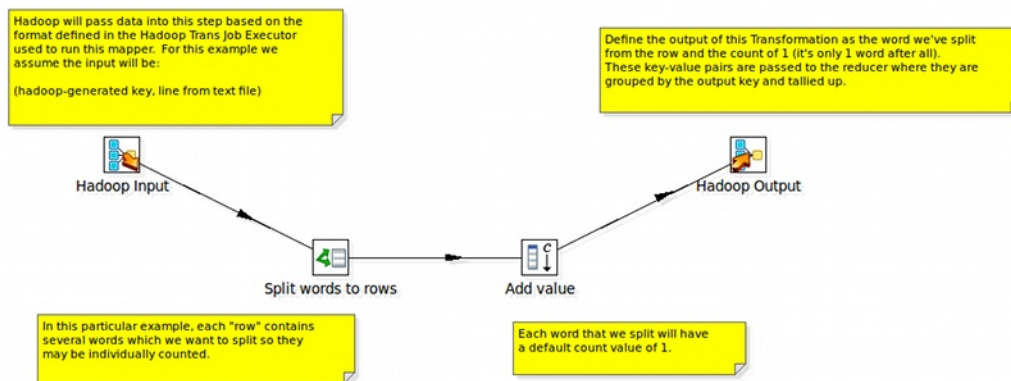
- Loading Data into Hadoop from outside the Hadoop cluster is a 5-minute video that demonstrates moving data using a PDI job and transformation: <http://www.youtube.com/watch?v=Ylekzmd6TAc>
- Use [Pentaho MapReduce](#) to interactively design a data flow for a MapReduce job without writing scripts or code. Here is a 12 minute video that provides an overview of the process: <http://www.youtube.com/watch?v=KZe1UugxXcs>.

Pentaho MapReduce Workflow

PDI and Pentaho MapReduce enables you to pull data from a Hadoop cluster, transform it, and pass it back to the cluster. Here is how you would approach doing this.

PDI Transformation

Start by deciding what you want to do with your data, open a PDI transformation, and drag the appropriate steps onto the canvas, configuring the steps to meet your data requirements. Drag the specifically-designed Hadoop **MapReduce Input** and Hadoop **MapReduce Output** steps onto the canvas. PDI provides these steps to completely avoid the need to write Java classes for this functionality. Configure both of these steps as needed. Once you have configured all the the steps, add hops to sequence the steps as a transformation. Follow the workflow as shown in this sample transformation in order to properly communicate with Hadoop. Name this transformation Mapper.



Hadoop communicates in key/value pairs. PDI uses the **MapReduce Input** step to define how key/value pairs from Hadoop are interpreted by PDI. The **MapReduce Input** dialog box enables you to configure the **MapReduce Input** step.

MapReduce Input

Step name: MapReduce Input

| | Type | Length | Precision |
|-------------|--------|--------|-----------|
| Key field | Number | 0 | 0 |
| Value field | Number | 0 | 0 |

OK Cancel

PDI uses a **MapReduce Output** step to pass the output back to Hadoop. The **MapReduce Output** dialog box enables you to configure the **MapReduce Output** step.

MapReduce Output

Step name: MapReduce Output

Key field: key

Value field: value

OK Cancel

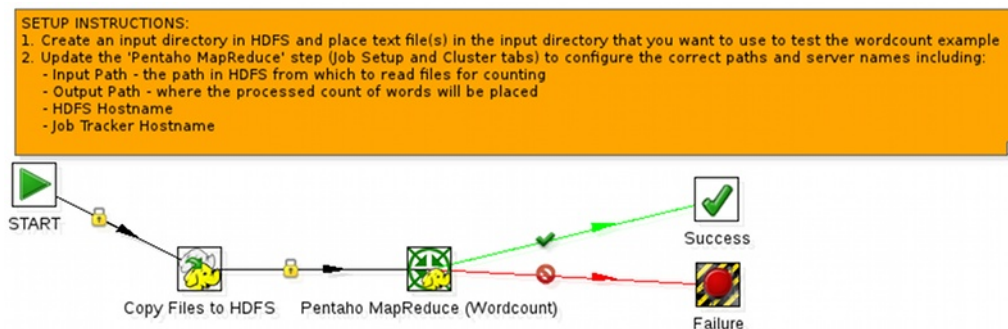
What happens in the middle is entirely up to you. Pentaho provides many sample steps you can alter to create the functionality you need.

PDI Job

Once you have created the Mapper transformation, you are ready to include it in a **Pentaho MapReduce** job entry and build a MapReduce job. Open a PDI job and drag the specifically-designed **Pentaho MapReduce** job entry onto the canvas. In addition to ordinary transformation work, this entry is designed to execute mapper/reducer functions within PDI. Again, no need to provide a Java class to achieve this.

Configure the **Pentaho MapReduce** entry to use the transformation as a mapper. Drag and drop a Start job entry, other job entries as needed, and result jobentries to handle the output onto the canvas. Add hops to sequence the entries into a job that you execute in PDI.

The workflow for the job should look something like this.



The Pentaho **MapReduce** dialog box enables you to configure the Pentaho MapReduce entry.

Pentaho MapReduce

Name: Pentaho MapReduce

Hadoop Job Name:

Mapper | Combiner | Reducer | **Job Setup** | Cluster | User Defined

Look in: Local

Mapper Transformation: Browse...

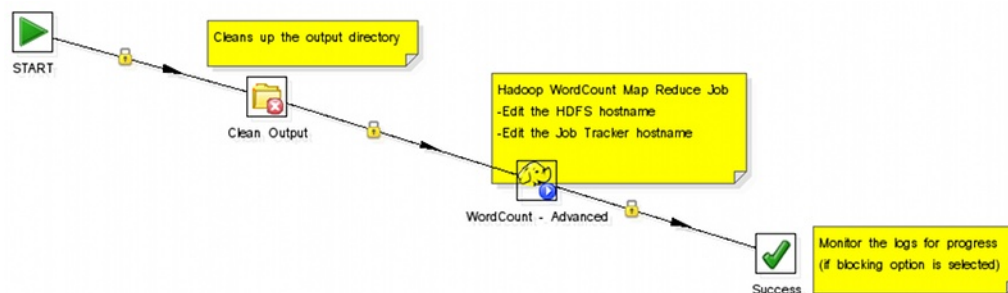
Mapper Input Step Name:

Mapper Output Step Name:

OK Cancel

PDI Hadoop Job Workflow

PDI enables you to execute a Java class from within a PDI/Spoon job to perform operations on Hadoop data. The way you approach doing this is similar to the way would for any other PDI job. The specifically-designed job entry that handles the Java class is **Hadoop Job Executor**. In this illustration it is used in the **WordCount - Advanced** entry.



The **Hadoop Job Executor** dialog box enables you to configure the entry with a `jar` file that contains the Java class.

Hadoop Job Executor

Name: WordCount - Simple

Hadoop Job Name: PDI Hadoop - WordCount - Simple

Jar: /samples/jobs/hadoop/pentaho-mapreduce-sample.jar Browse...

Configuration
☒ Simple ☐ Advanced

Command line arguments: /junit/wordcount/input /junit/wordcount/pdioutput

OK Cancel

If you are using the Amazon Elastic MapReduce (EMR) service, you can **Amazon EMR Job Executor**. job entry to execute the Java class This differs from the standard Hadoop Job Executor in that it contains connection information for Amazon S3 and configuration options for EMR.

Amazon EMR Job Executor

Name: Amazon EMR Job Executor

EMR Job Flow Name: ◆

Existing JobFlow Id (optional): ◆

AWS Access Key: ◆

AWS Secret Key: ◆

S3 Staging Directory: ◆ Browse...

MapReduce Jar: ◆ Browse...

Command Line Arguments: ◆

Number of Instances: 2 ◆

Master Instance Type: Small [m1.small] ▼

Slave Instance Type: Small [m1.small] ▼

Enable Blocking ☐

Logging Interval: 60 ◆

OK Cancel

Hadoop to PDI Data Type Conversion

The **Hadoop Job Executor** and **Pentaho MapReduce** steps have an advanced configuration mode that enables you to specify data types for the job's input and output. PDI is unable to detect foreign data types on its own; therefore you must specify the input and output data types in the **Job Setup** tab. This table explains the relationship between Hadoop data types and their PDI equivalents.

| PDI (Kettle) Data Type | Apache Hadoop Data Type |
|------------------------|----------------------------------|
| java.lang.Integer | org.apache.hadoop.io.IntWritable |
| java.lang.Long | org.apache.hadoop.io.IntWritable |

| PDI (Kettle) Data Type | Apache Hadoop Data Type |
|-----------------------------------|-----------------------------------|
| java.lang.Long | org.apache.hadoop.io.LongWritable |
| org.apache.hadoop.io.IntWritable | java.lang.Long |
| java.lang.String | org.apache.hadoop.io.Text |
| java.lang.String | org.apache.hadoop.io.IntWritable |
| org.apache.hadoop.io.LongWritable | org.apache.hadoop.io.Text |
| org.apache.hadoop.io.LongWritable | java.lang.Long |

For more information on configuring **Pentaho MapReduce** to convert to additional data types, see <http://wiki.pentaho.com/display/BAD/Pentaho+MapReduce>.

Hadoop Hive-Specific SQL Limitations

There are a few key limitations in Hive that prevent some regular Metadata Editor features from working as intended, and limit the structure of your SQL queries in Report Designer:

- **Outer joins are not supported.**
- **Each column can only be used once in a SELECT clause.** Duplicate columns in SELECT statements cause errors.
- **Conditional joins can only use the = conditional unless you use a WHERE clause.** Any non-equal conditional in a FROM statement forces the Metadata Editor to use a cartesian join and a WHERE clause conditional to limit it. This is not much of a limitation, but it may seem unusual to experienced Metadata Editor users who are accustomed to working with SQL databases.

Big Data Tutorials

These sections contain guidance and instructions about using Pentaho technology as part of your overall big data strategy. Each section is a series of scenario-based tutorials that demonstrate the integration between Pentaho and Hadoop using a sample data set.

Hadoop Tutorials

These tutorials are organized by topic and each set explains various techniques for loading, transforming, extracting and reporting on data within a Hadoop cluster. You are encouraged to perform the tutorials in order as the output of one is sometimes used as the input of another. However, if you would like to jump to a tutorial in the middle of the flow, instructions for preparing input data are provided.

Loading Data into a Hadoop Cluster

These scenario-based tutorials contain guidance and instructions on loading data into HDFS (Hadoop's Distributed File System), Hive and HBase using Pentaho Data Integration (PDI)

Prerequisites

To perform the tutorials in this section you must have these components installed.

PDI—The primary development environment for the tutorials. See the [Data Integration Installation Options](#) if you have not already installed PDI.

Apache Hadoop 0.20.X—A single-node local cluster is sufficient for these exercises, but a larger and/or remote configuration also works. If you are using a different distribution of Hadoop see [Configure Your Big Data Environment](#). You need to know the addresses and ports for your Hadoop installation.

***Hive**—A supported version of Hive. Hive is a Map/Reduce abstraction layer that provides SQL-like access to Hadoop data. For instructions on installing or using Hive, see the [Hive Getting Started Guide](#).

***HBase**—A supported version of HBase. HBase is an open source, non-relational, distributed database that runs on top of HDFS. For instructions on installing or using HBase, see the [Getting Started section of the Apache HBase Reference Guide](#).

*Component only required for corresponding tutorial.

Sample Data

The tutorials in this section were created with this sample weblog data.

| Tutorial | File Name | Content |
|---|---|-----------------------------------|
| Using a Job Entry to Load Data into Hadoop's Distributed File System (HDFS) | weblogs_rebuild.txt.zip | Unparsed, raw weblog data |
| Using a Job Entry to Load Data into Hive | weblogs_parse.txt.zip | Tab-delimited, parsed weblog data |
| Using a Transformation Step to Load Data into HBase | weblogs_hbase.txt.zip | Prepared data for HBase load |

Using a Job Entry to Load Data into Hadoop's Distributed File System (HDFS)

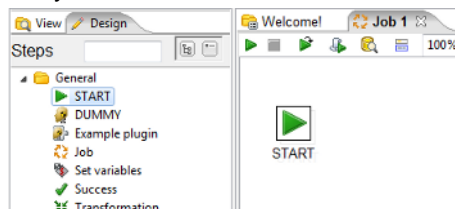
In order to follow along with this tutorial, you will need

- Hadoop
- Pentaho Data Integration

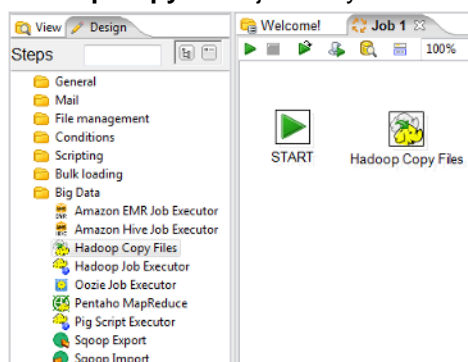
You can use PDI jobs to put files into HDFS from many different sources. This tutorial describes how to create a PDI job to move a sample file into HDFS.

If not already running, start Hadoop and PDI. Unzip the sample data files and put them in a convenient location:
[weblogs_rebuild.txt.zip](#).


1. Create a new Job by selecting **File > New > Job**.
2. Add a Start job entry to the canvas. From the **Design** palette on the left, under the **General** folder, drag a **Start** job entry onto the canvas.

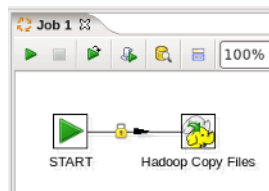


3. Add a Hadoop Copy Files job entry to the canvas. From the **Design** palette, under the **Big Data** folder, drag a **Hadoop Copy Files** job entry onto the canvas.



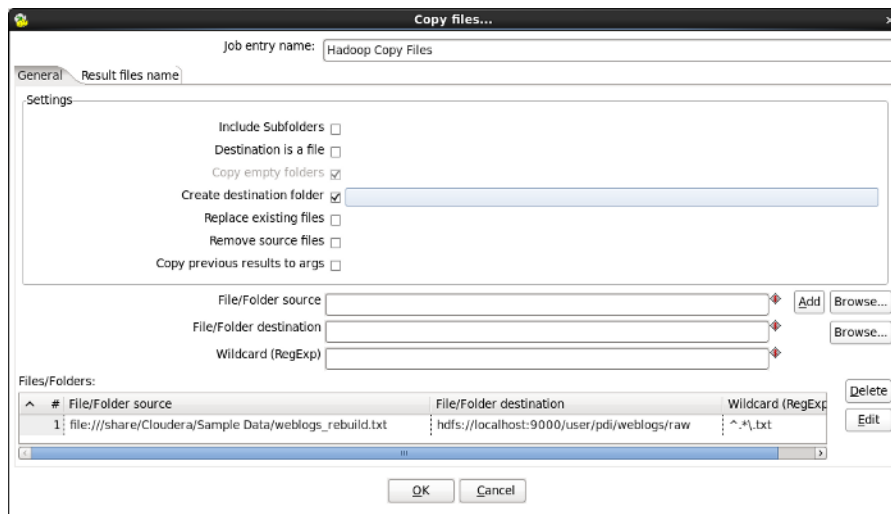
4.

Connect the two job entries by hovering over the **Start** entry and selecting the output connector , then drag the connector arrow to the **Hadoop Copy Files** entry.




5. Enter the source and destination information within the properties of the **Hadoop Copy Files** entry by double-clicking it.
 - a) For **File/Folder source(s)**, click **Browse** and navigate to the folder containing the downloaded sample file `weblogs_rebuild.txt`.
 - b) For **File/Folder destination(s)**, enter `hdfs://<NAMENODE>:<PORT>/user/pdi/weblogs/raw`, where `NAMENODE` and `PORT` reflect your Hadoop destination.
 - c) For **Wildcard (RegExp)**, enter `^.*\.txt`.
 - d) Click **Add** to include the entries to the list of files to copy.
 - e) Check the **Create destination folder** option to ensure that the `weblogs` folder is created in HDFS the first time this job is executed.

When you are done your window should look like this (your file paths may be different).



Click **OK** to close the window.

6. Save the job by selecting **Save as** from the **File** menu. Enter `load_hdfs.kjb` as the file name within a folder of your choice.
7. Run the job by clicking the green Run button on the job toolbar , or by selecting **Action > Run** from the menu. The **Execute a job** window opens. Click **Launch**.

An **Execution Results** panel opens at the bottom of the Spoon interface and displays the progress of the job as it runs. After a few seconds the job finishes successfully.

The 'Execution results' panel shows a table of job execution details. The 'load_hdfs' job is highlighted, showing its progress from start to finish.

| Job / Job Entry | Comment | Result | Reason | Filename | Nr | Log date |
|-------------------|------------------------|---------|-----------------------------|----------|----|---------------------|
| Job: load_hdfs | Start of job execution | | start | | | 2012/01/27 08:25:45 |
| START | Start of job execution | | start | | | 2012/01/27 08:25:45 |
| START | Job execution finished | Success | | | 0 | 2012/01/27 08:25:45 |
| Hadoop Copy Files | Start of job execution | | Followed unconditional link | | | 2012/01/27 08:25:45 |
| Hadoop Copy Files | Job execution finished | Success | | | 0 | 2012/01/27 08:25:46 |
| Job: load_hdfs | Job execution finished | Success | finished | | 0 | 2012/01/27 08:25:46 |

If any errors occurred the job entry that failed will be highlighted in red and you can use the **Logging** tab to view error messages.

8. Verify the data was loaded by querying Hadoop.
 - a) From the command line, query Hadoop by entering this command.

```
hadoop fs -ls /user/pdi/weblogs/raw
```

This statement is returned


```
-rwxrwxrwx 3 demo demo 77908174 2011-12-28 07:16 /user/pdi/weblogs/raw/weblog_raw.txt
```

Using a Job Entry to Load Data into Hive

In order to follow along with this tutorial, you will need

- Hadoop
- Pentaho Data Integration
- Hive

PDI jobs can be used to put files into Hive from many different sources. This tutorial instructs you how to use a PDI job to load a sample data file into a Hive table.

 **Note:** Hive could be defined with external data. Using the external option, you could define a Hive table that uses the HDFS directory that contains the parsed file. For this tutorial, we chose not to use the external option to demonstrate the ease with which files can be added to non-external Hive tables.

If not already running, start Hadoop, PDI, and the Hive server. Unzip the sample data files and put them in a convenient location: [weblogs_parse.txt.zip](#).

This file should be placed in the `/user/pdi/weblogs/parse` directory of HDFS using these three commands.

```
hadoop fs -mkdir /user/pdi/weblogs
hadoop fs -mkdir /user/pdi/weblogs/parse
hadoop fs -put weblogs_parse.txt /user/pdi/weblogs/parse/part-00000
```

If you previously completed the [Using Pentaho MapReduce to Parse Weblog Data](#) tutorial, the necessary files will already be in the proper directory.

1. Create a Hive Table.

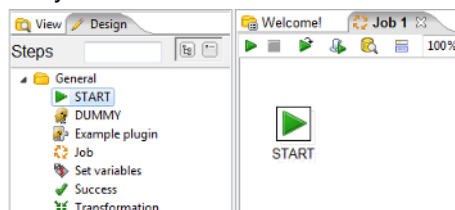
- Open the Hive shell by entering 'hive' at the command line.
- Create a table in Hive for the sample data by entering

```
create table weblogs (
  client_ip      string,
  full_request_date string,
  day            string,
  month          string,
  month_num      int,
  year           string,
  hour           string,
  minute         string,
  second         string,
  timezone       string,
  http_verb      string,
  uri            string,
  http_status_code string,
  bytes_returned string,
  referrer       string,
  user_agent     string)
row format delimited
fields terminated by '\t';
```

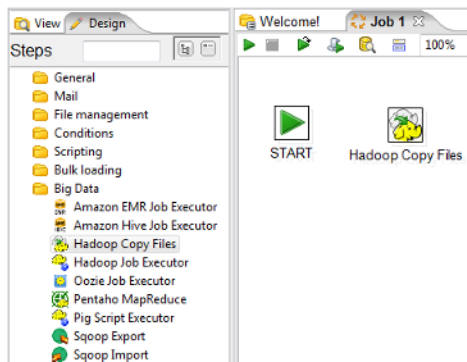
- Close the Hive shell by entering 'quit'.

2. Create a new Job to load the sample data into a Hive table by selecting **File > New > Job**.


3. Add a Start job entry to the canvas. From the **Design** palette on the left, under the **General** folder, drag a **Start** job entry onto the canvas.

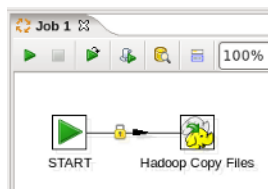


4. Add a Hadoop Copy Files job entry to the canvas. From the **Design** palette, under the **Big Data** folder, drag a **Hadoop Copy Files** job entry onto the canvas.



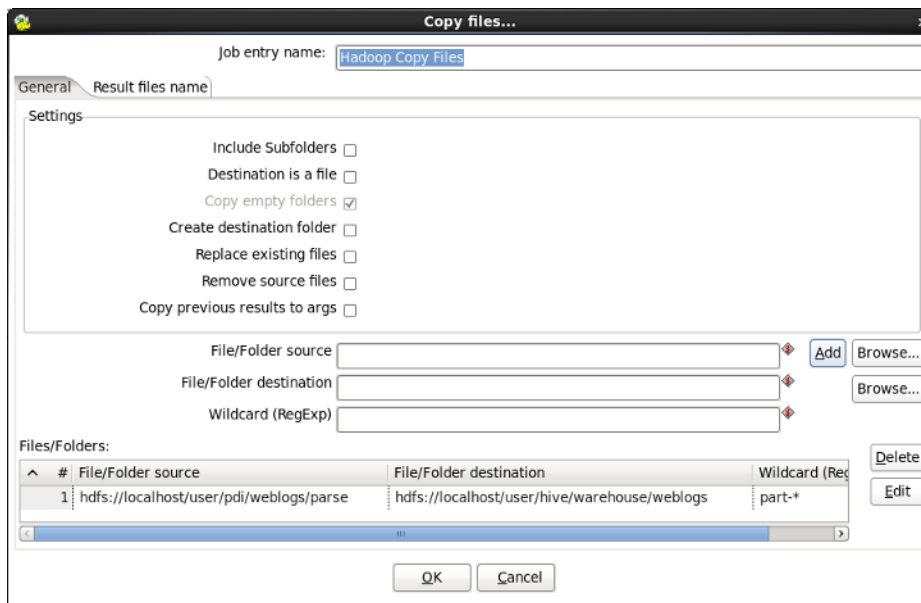
5.

Connect the two job entries by hovering over the **Start** entry and selecting the output connector , then drag the connector arrow to the **Hadoop Copy Files** entry.



6. Enter the source and destination information within the properties of the **Hadoop Copy Files** entry by double-clicking it.
- For **File/Folder source(s)**, enter `hdfs://<NAMENODE>:<PORT>/user/pdi/weblogs/parse`, where **NAMENODE** and **PORT** reflect your Hadoop destination.
 - For **File/Folder destination(s)**, enter `hdfs://<NAMENODE>:<PORT>/user/hive/warehouse/weblogs`.
 - For **Wildcard (RegExp)**, enter `part-.*`.
 - Click the **Add** button to add the entries to the list of files to copy.

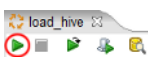
When you are done your window should look like this (your file paths may be different)



Click **OK** to close the window.

7. Save the job by selecting **Save as** from the **File** menu. Enter `load_hive.kjb` as the file name within a folder of your choice.

8.

Run the job by clicking the green Run button on the job toolbar , or by selecting **Action > Run** from the menu. The **Execute a job** window opens. Click **Launch**.

An **Execution Results** panel opens at the bottom of the Spoon interface and displays the progress of the job as it runs. After a few seconds the job finishes successfully.

| Job / Job Entry | Comment | Result | Reason | Filename |
|-----------------|------------------------|---------|-----------------------------|----------|
| load_hive | | | | |
| Job: load_hive | Start of job execution | | start | |
| START | Start of job execution | | start | |
| START | Job execution finished | Success | | |
| Copy Files | Start of job execution | | Followed unconditional link | |
| Copy Files | Job execution finished | Success | | |
| Job: load_hive | Job execution finished | Success | finished | |

If any errors occurred the job entry that failed will be highlighted in red and you can use the **Logging** tab to view error messages.

9. Verify the data was loaded by querying Hive.
 - a) Open the Hive shell from the command line by entering `hive`.
 - b) Enter this query to verify the data was loaded correctly into Hive.

```
select * from weblogs limit 10;
```

Ten rows of data are returned.

Using a Transformation Step to Load Data into HBase

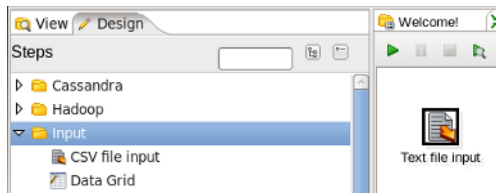
In order to follow along with this tutorial, you will need

- Hadoop
- Pentaho Data Integration
- HBase

This tutorial describes how to use data from a sample flat file to create a HBase table using a PDI transformation. For the sake of brevity, you will use a prepared sample dataset and a simple transformation to prepare and transform your data for HBase loads.

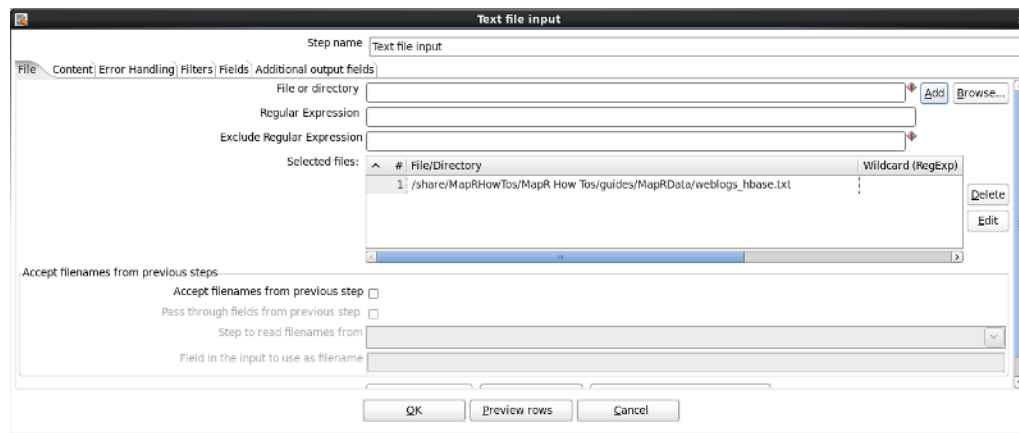
If not already running, start Hadoop, PDI, and HBase. Unzip the sample data files and put them in a convenient location: [weblogs_hbase.txt.zip](#)

1. Create a HBase Table.
 - a) Open the HBase shell by entering `hbase shell` at the command line.
 - b) Create the table in HBase by entering `create 'weblogs', 'pageviews'` in the HBase shell. This creates a table named `weblogs` with a single column family named `pageviews`.
 - c) Close the HBase shell by entering `quit`.
2. From within the Spoon, create a new transformation by selecting **File > New > Transformation**.
3. Identify the source where the transformation will get data from. For this tutorial your source is a text file (.txt). From the **Input** folder of the **Design** palette on the left, add a **Text File Input** step to the transformation by dragging it onto the canvas.



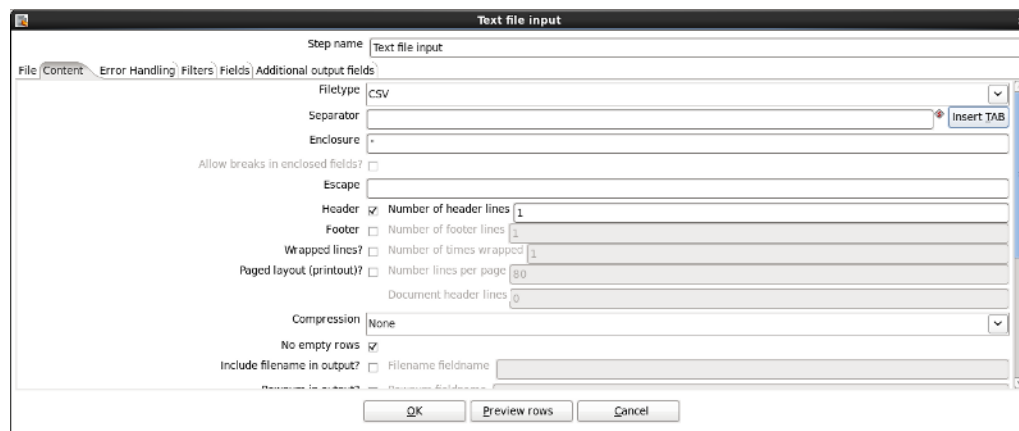
4. Edit the properties of the **Text file input** step by double-clicking the icon. The **Text file input** dialog box appears.
5. From the **File** tab, in the **File or Directory** field, click **Browse** and navigate to the `weblog_hbase.txt` file. Click **Add**.

The file appears in the **Selected files** pane.



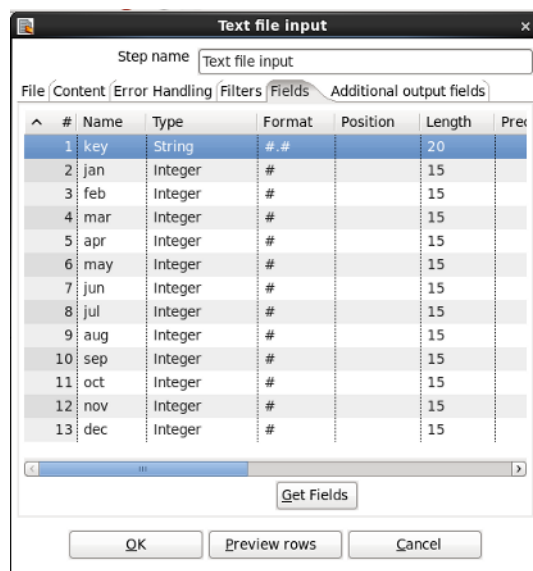
6. Configure the contents of the file by switching to the **Content** tab.

- For **Separator**, clear the contents and click **Insert TAB**.
- Check the **Header** checkbox.
- For **Format**, Select **Unix** from the drop-down menu.




7. Configure the input fields.

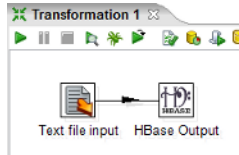
- From the **Fields** tab, select **Get Fields** to populate the list the available fields.
- A dialog box appears asking for **Number of sample lines**. Enter **100** and click **OK**.
- Change the **Type** of the field named **key** to **String** and set the **Length** to **20**.



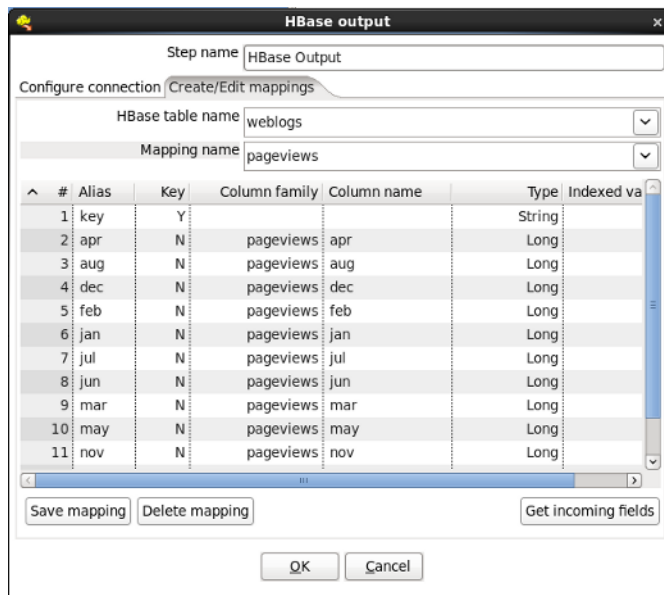
Click **OK** to close the window.

8. On the **Design** palette, under **Big Data**, drag the **HBase Output** to the canvas. Create a hop to connect your input

and **HBase Output** step by hovering over the input step and clicking the output connector , then drag the connector arrow to the **HBase Output** step.




9. Edit the **HBase Output** step by double-clicking it. You must now enter your Zookeeper host(s) and port number.
- For the **Zookeeper hosts(s)** field, enter a comma separated list of your HBase Zookeeper Hosts. For local single node clusters use `localhost`.
 - For **Zookeeper port**, enter the port for your Zookeeper hosts. By default this is 2181.
10. Create a HBase mapping to tell Pentaho how to store the data in HBase by switching to the **Create/Edit mappings** tab and changing these options.
- For **HBase table name**, select **weblogs**.
 - For **Mapping name**, enter `pageviews`.
 - Click **Get incoming fields**.
 - For the alias **key** change the **Key** column to **Y**, clear the **Column family** and **Column name** fields, and set the **Type** field to **String**. Click **Save mapping**.



| # | Alias | Key | Column family | Column name | Type | Indexed value |
|----|-------|-----|---------------|-------------|--------|---------------|
| 1 | key | Y | | | String | |
| 2 | apr | N | pageviews | apr | Long | |
| 3 | aug | N | pageviews | aug | Long | |
| 4 | dec | N | pageviews | dec | Long | |
| 5 | feb | N | pageviews | feb | Long | |
| 6 | jan | N | pageviews | jan | Long | |
| 7 | jul | N | pageviews | jul | Long | |
| 8 | jun | N | pageviews | jun | Long | |
| 9 | mar | N | pageviews | mar | Long | |
| 10 | may | N | pageviews | may | Long | |
| 11 | nov | N | pageviews | nov | Long | |

11. Configure the HBase out to use the mapping you just created.
- Go back to the **Configure connection** tab and click **Get table names**.
 - For **HBase table name**, enter `weblogs`.
 - Click **Get mappings for the specified table**.
 - For **Mapping name**, select `pageviews`. Click **OK** to close the window.

Save the transformation by selecting **Save as** from the **File** menu. Enter `load_hbase.ktr` as the file name within a folder of your choice.

12. Run the transformation by clicking the green **Run** button on the transformation toolbar , or by choosing **Action > Run** from the menu. The **Execute a transformation** window opens. Click **Launch**.

An **Execution Results** panel opens at the bottom of the Spoon interface and displays the progress of the transformation as it runs. After a few seconds the transformation finishes successfully.

| Execution Results | | | | | | | | | | |
|--|-----------------|--------|-------|---------|-------|--------|---------|----------|--------|----------|
| Execution History Logging Step Metrics Performance Graph | | | | | | | | | | |
| # | Stepname | Copynr | Read | Written | Input | Output | Updated | Rejected | Errors | Active |
| 1 | Text file input | 0 | 0 | 27300 | 27301 | 0 | 1 | 0 | 0 | Finished |
| 2 | HBase Output | 0 | 27300 | 27300 | 0 | 0 | 0 | 0 | 0 | Finished |

If any errors occurred the transformation step that failed will be highlighted in red and you can use the **Logging** tab to view error messages.

13. Verify the data was loaded by querying HBase.

a) From the command line, open the HBase shell by entering this command.

```
hbase shell
```

b) Query HBase by entering this command.

```
scan 'weblogs', {LIMIT => 10}
```

Ten rows of data are returned.

Transforming Data within a Hadoop Cluster

These tutorials contain guidance and instructions on transforming data within the Hadoop cluster using Pentaho MapReduce, Hive, and Pig.

- [Using Pentaho MapReduce to Parse Weblog Data](#)—How to use Pentaho MapReduce to convert raw weblog data into parsed, delimited records.
- [Using Pentaho MapReduce to Generate an Aggregate Dataset](#)—How to use Pentaho MapReduce to transform and summarize detailed data into an aggregate dataset.
- [Transforming Data within Hive](#)—How to read data from a Hive table, transform it, and write it to a Hive table within the workflow of a PDI job.
- [Transforming Data with Pig](#)—How to invoke a Pig script from a PDI job.

Extracting Data from a Hadoop Cluster

These tutorials contain guidance and instructions on extracting data from Hadoop using HDFS, Hive, and HBase.

- [Extracting Data from HDFS to Load an RDBMS](#)—How to use a PDI transformation to extract data from HDFS and load it into a RDBMS table.
- [Extracting Data from Hive to Load an RDBMS](#)—How to use a PDI transformation to extract data from Hive and load it into a RDBMS table.
- [Extracting Data from HBase to Load an RDBMS](#)—How to use a PDI transformation to extract data from HBase and load it into a RDBMS table.
- [Extracting Data from Snappy Compressed Files](#)—How to configure client-side PDI so that files compressed using the Snappy codec can be decompressed using the Hadoop file input or Text file input step.

Reporting on Data within a Hadoop Cluster

These tutorials contain guidance and instructions about reporting on data within a Hadoop cluster.

- [Reporting on HDFS File Data](#)—How to create a report that sources data from a HDFS file.
- [Reporting on HBase Data](#)—How to create a report that sources data from HBase.
- [Reporting on Hive Data](#)—How to create a report that sources data from Hive.

MapR Tutorials

These tutorials are organized by topic and each set explains various techniques for loading, transforming, extracting and reporting on data within a MapR cluster. You are encouraged to perform the tutorials in order as the output of one is sometimes used as the input of another. However, if you would like to jump to a tutorial in the middle of the flow, instructions for preparing input data are provided.

Loading Data into a MapR Cluster

These tutorials contain guidance and instructions on loading data into CLDB (MapR's distributed file system), Hive, and HBase.

- [Loading Data into CLDB](#)—How to use a PDI job to move a file into CLDB.
- [Loading Data into MapR Hive](#)—How to use a PDI job to load a data file into a Hive table.
- [Loading Data into MapR HBase](#)—How to use a PDI transformation that sources data from a flat file and writes to an HBase table.

Transforming Data within a MapR Cluster

These tutorials contain guidance and instructions on leveraging the massively parallel, fault tolerant MapR processing engine to transform resident cluster data.

- [Using Pentaho MapReduce to Parse Weblog Data in MapR](#)—How to use Pentaho MapReduce to convert raw weblog data into parsed, delimited records.
- [Using Pentaho MapReduce to Generate an Aggregate Dataset in MapR](#)—How to use Pentaho MapReduce to transform and summarize detailed data into an aggregate dataset.
- [Transforming Data within Hive in MapR](#)—How to read data from a Hive table, transform it, and write it to a Hive table within the workflow of a PDI job.
- [Transforming Data with Pig in MapR](#)—How to invoke a Pig script from a PDI job.

Extracting Data from a MapR Cluster

These tutorials contain guidance and instructions on extracting data from a MapR cluster and loading it into an RDBMS table.

- [Extracting Data from CLDB to Load an RDBMS](#)—How to use a PDI transformation to extract data from MapR CLDB and load it into a RDBMS table.
- [Extracting Data from Hive to Load an RDBMS in MapR](#)—How to use a PDI transformation to extract data from Hive and load it into a RDBMS table.
- [Extracting Data from HBase to Load an RDBMS in MapR](#)—How to use a PDI transformation to extract data from HBase and load it into a RDBMS table.

Reporting on Data within a MapR Cluster

These tutorials contain guidance and instructions about reporting on data within a MapR cluster.

- [Reporting on CLDB File Data](#)—How to create a report that sources data from a MapR CLDB file.
- [Reporting on HBase Data in MapR](#)—How to create a report that sources data from HBase.
- [Reporting on Hive Data in MapR](#)—How to create a report that sources data from Hive.

Cassandra Tutorials

These tutorials demonstrate the integration between Pentaho and the Cassandra NoSQL Database, specifically techniques about writing data to and reading data from Cassandra using graphical tools. These tutorials also include instructions on how to sort and group data, create reports, and combine data from Cassandra with data from other sources.

- [Write Data To Cassandra](#)—How to read data from a data source (flat file) and write it to a column family in Cassandra using a graphic tool.
- [How To Read Data From Cassandra](#)—How to read data from a column family in Cassandra using a graphic tool.
- [How To Create a Report with Cassandra](#)—How to create a report that uses data from a column family in Cassandra using graphic tools.

MongoDB Tutorials

These tutorials demonstrate the integration between Pentaho and the MongoDB NoSQL Database, specifically how to write data to, read data from, MongoDB using graphical tools. These tutorials also include instructions on sorting and grouping data, creating reports, and combining data from Mongo with data from other sources.

- [Write Data To MongoDB](#)—How to read data from a data source (flat file) and write it to a collection in MongoDB
- [Read Data From MongoDB](#)—How to read data from a collection in MongoDB.

- [*Create a Report with MongoDB*](#)—How to create a report that uses data from a collection in MongoDB.
- [*Create a Parameterized Report with MongoDB*](#)—How to create a parameterize report that uses data from a collection in MongoDB.

PDI Hadoop Configurations

Within PDI, a Hadoop configuration is the collection of Hadoop libraries required to communicate with a specific version of Hadoop and related tools, such as Hive HBase, Sqoop, or Pig.

Hadoop configurations are defined in the `plugin.properties` file and are designed to be easily configured within PDI by changing the `active.hadoop.configuration` property. The `plugin.properties` file resides in the `pentaho-big-data-plugin/` folder.

All Hadoop configurations share a basic structure. Elements of the structure are defined in the table following this code block.

```
configuration/
|-- lib/
|--   |-- client/
|--   |-- pmr/
|--   '-- *.jar
|-- config.properties
|-- core-site.xml
'-- configuration-implementation.jar
```

| Configuration Element | Definition |
|----------------------------------|--|
| lib/ | Libraries specific to the version of Hadoop this configuration was created to communicate with. |
| client/ | Libraries that are only required on a Hadoop client, for instance <code>hadoop-core-*</code> or <code>hadoop-client-*</code> |
| pmr/ | Jar files that contain libraries required for parsing data in input/output formats or otherwise outside of any PDI-based execution. |
| *.jar | All other libraries required for Hadoop configuration that are not client-only or special <code>pmr jar</code> files that need to be available to the entire JVM of Hadoop job tasks. |
| config.properties | Contains metadata and configuration options for this Hadoop configuration. Provides a way to define a configuration name, additional classpath, and native libraries the configuration requires. See the comments in this file for more details. |
| core-site.xml | Configuration file that can be replaced to set a site-specific configuration, for example <code>hdfs-site.xml</code> would be used to configure HDFS. |
| configuration-implementation.jar | File that must be replaced in order to communicate with this configuration. |

Including/Excluding Classes or Packages for a Hadoop Configuration

You have the option to include or exclude classes or packages from loading with a Hadoop configuration. Configure these options within the `plugin.properties` file located at `plugins/pentaho-big-data-plugin`. For additional information, see the comments within the `plugin.properties` file.

Including Additional Class Paths or Libraries

To to include additional class paths, native libraries, or a user-friendly configuration name, include the directory within `classpath` property within the `big data plugin.properties` file.

Exclude Classes or Packages

To exclude classes or packages from being loaded twice by a Hadoop configuration class loader, include them in the `ignored.classes` property within the `plugin.properties` file. This is necessary when logging libraries expect a single class shared by all class loaders, as with Apache Commons Logging for example.

PDI Big Data Transformation Steps

This section contains reference documentation for transformation steps which enable PDI to work with big data technologies.

Please see [Create DI Solutions](#) for additional transformation step references.

Avro Input

The Avro Input step decodes binary or JSON Avro data and extracts fields from the structure it defines, either from flat files or incoming fields.

Source tab

| Option | Definition |
|--|---|
| Avro source is in file | Indicates the source data comes from a file. |
| Avro source is defined in a field | Indicates the source data comes from a field, and you can select an incoming field to decode from the Avro field to decode from drop-down box. In this mode of operation, a schema file must be specified in the Schema file field. |
| Avro file | Specifies the file to decode. |
| Avro field to decode from | Specifies the incoming field containing Avro data to decode. |
| JSON encoded | Indicates the Avro data has been encoded in JSON. |

Schema tab

| Option | Definition |
|-------------------------------------|--|
| Schema file | Indicates an Avro schema file. |
| Schema is defined in a field | Indicates the schema specified to use for decoding an incoming Avro object is found within a field. When checked, this option enables the Schema in field is a path and Cache schemas options. This also changes the Schema file label to Default schema file , which the user can specify if an incoming schema is missing. |
| Schema in field is a path | Indicates that the incoming schema specifies a path to a schema file. If left unchecked, the step assumes the incoming schema is the actual schema definition in JSON format. |
| Cache schemas in memory | Enables the step to retain all schemas seen in memory and uses this before loading or parsing an incoming schema. |
| Field containing schema | Indicates which field contains the Avro schema. |

Avro fields tab

| Option | Definition |
|---|---|
| Do not complain about fields not present in the schema | Disables issuing an exception when specified paths or fields are not present in the active Avro schema. Instead a <code>null</code> value is returned. OR Instead the system returns a <code>null</code> value. |

| Option | Definition |
|-------------------|---|
| Preview | Displays a review of the fields or data from the designated source file. |
| Get fields | Populates the fields available from the designated source file or schema and gives each extracted field a name that reflects the path used to extract it. |

Lookup fields tab

| Option | Definition |
|----------------------------|--|
| Get incoming fields | Populates the Name column of the table with the names of incoming Kettle fields. The Variable column of the table allows you to assign the values of these incoming fields to variable. A default value (to use in case the incoming field value is <code>null</code>) can be supplied in the Default value column. These variables can then be used anywhere in the Avro paths defined in the Avro fields tab. |

Cassandra Input

Configure Cassandra Input

Cassandra Input is an input step that enables data to be read from a Cassandra column family (table) as part of an ETL transformation.

| Option | Definition |
|------------------------------|--|
| Step name | The name of this step as it appears in the transformation workspace. |
| Cassandra host | Connection host name input field. |
| Cassandra port | Connection host port number input field. |
| Username | Input field for target keyspace and/or family (table) authentication details. |
| Password | Input field for target keyspace and/or family (table) authentication details. |
| Keyspace | Input field for the keyspace (database) name. |
| Use query compression | If checked, tells the step whether or not to compress the text of the CQL query before sending it to the server. |
| Show schema | Opens a dialog that shows metadata for the column family named in the CQL SELECT query. |

CQL SELECT Query

The large text box at the bottom of the dialog enables you to enter a CQL SELECT statement to be executed. Only a single SELECT query is accepted by the step.

```
SELECT [FIRST N] [REVERSED] <SELECT_EXPR>
FROM <COLUMN_FAMILY> [USING <CONSISTENCY>] [WHERE <CLAUSE>] [LIMIT N];
```



Important: Cassandra Input does not support the CQL range notation, for instance `name1..nameN`, for specifying columns in a SELECT query.

Select queries may name columns explicitly (in a comma separated list) or use the * wildcard. If the wildcard is used then only those columns defined in the metadata for the column family in question are returned. If columns are selected explicitly, then the name of each column must be enclosed in single quotation marks. Because Cassandra is a sparse column oriented database, as is the case with HBase, it is possible for rows to contain varying numbers of columns which might or might not be defined in the metadata for the column family. The Cassandra Input step can emit columns that are not defined in the metadata for the column family in question if they are explicitly named in the SELECT clause. Cassandra Input uses type information present in the metadata for a column family. This, at a minimum, includes a default type (column validator) for the column family. If there is explicit metadata for individual columns available, then this is used for type information, otherwise the default validator is used.

| Option | Definition |
|---------------------|--|
| LIMIT | If omitted, Cassandra assumes a default limit of 10,000 rows to be returned by the query. If the query is expected to return more than 10,000 rows an explicit LIMIT clause must be added to the query. |
| FIRST N | Returns the first N [where N is determined by the column sorting strategy used for the column family in question] column values from each row, if the column family in question is sparse then this may result in a different N (or less) column values appearing from one row to the next. Because PDI deals with a constant number of fields between steps in a transformation, Cassandra rows that do not contain particular columns are output as rows with null field values for non-existent columns. Cassandra's default for FIRST (if omitted from the query) is 10,000 columns. If a query is expected to return more than 10,000 columns, then an explicit FIRST must be added to the query. |
| REVERSED | Option causes the sort order of the columns returned by Cassandra for each row to be reversed. This may affect which values result from a FIRST N option, but does not affect the order of the columns output by Cassandra Input. |
| WHERE clause | Clause provides for filtering the rows that appear in results. The clause can filter on a key name, or range of keys, and in the case of indexed columns, on column values. Key filters are specified using the KEY keyword, a relational operator (one of =, >, >=, <, and <=) and a term value. |

Cassandra Output

Configure Cassandra Output


Cassandra Output is an output step that enables data to be written to a Cassandra column family (table) as part of an ETL transformation.


| Option | Definition |
|-----------------------|---|
| Step name | The name of this step as it appears in the transformation workspace. |
| Cassandra host | Connection host name input field. |
| Cassandra port | Connection host port number input field. |
| Username | Target keyspace and/or family (table) authentication details input field. |

| Option | Definition |
|--------------------|---|
| Password | Target keyspace and/or family (table) authentication details input field. |
| Keyspace | Input field for the keyspace (database) name. |
| Show schema | Opens a dialog box that shows metadata for the specified column family. |

Configure Column Family and Consistency Level

This tab contains connection details and basic query information, in particular, how to connect to Cassandra and execute a CQL (Cassandra query language) query to retrieve rows from a column family (table).


 **Important:** Note that Cassandra Output does not check the types of incoming columns against matching columns in the Cassandra metadata. Incoming values are formatted into appropriate string values for use in a textual CQL INSERT statement according to PDI's field metadata. If resulting values cannot be parsed by the Cassandra column validator for a particular column then an error results.

 **Note:** Cassandra Output converts PDI's dense row format into sparse data by ignoring incoming field values that are `null`.

| Option | Definition |
|---|--|
| Column family (table) | Input field to specify the column family, to which the incoming rows should be written. |
| Get column family names button | Populates the drop-down box with names of all the column families that exist in the specified keyspace. |
| Consistency level | Input field enables an explicit write consistency to be specified. Valid values are: ZERO, ONE, ANY, QUORUM and ALL. The Cassandra default is ONE. |
| Create column family | If checked, enables the step to create the named column family if it does not already exist. |
| Truncate column family | If checked, specifies whether any existing data should be deleted from the named column family before inserting incoming rows. |
| Update column family metadata | If checked, updates the column family metadata with information on incoming fields not already present, when option is selected. If this option is not selected, then any unknown incoming fields are ignored unless the Insert fields not in column metadata option is enabled. |
| Insert fields not in column metadata | If checked, inserts the column family metadata in any incoming fields not present, with respect to the default column family validator. This option has no effect if Update column family metadata is selected. |
| Commit batch size | Allows you to specify how many rows to buffer before executing a BATCH INSERT CQL statement. |
| Use compression | Option compresses (gzip) the text of each BATCH INSERT statement before transmitting it to the node. |

Pre-insert CQL

Cassandra Output gives you the option of executing an arbitrary set of CQL statements prior to inserting the first incoming PDI row. This is useful for creating or dropping secondary indexes on columns.

 **Note:** Pre-insert CQL statements are executed *after* any column family metadata updates for new incoming fields, and before the first row is inserted. This enables indexes to be created for columns corresponding new to incoming fields.

| Option | Definition |
|--|---|
| CQL to execute before inserting first row | Opens the CQL editor, where you can enter one or more semicolon-separated CQL statements to execute before data is inserted into the first row. |

CouchDB Input

The CouchDB Input step retrieves all documents from a given view in a given design document from a given database. The resulting output is a single String field named **JSON**, one row for each received document. For information about CouchDB, design documents, or views, see <http://guide.couchdb.org>.

| Option | Definition |
|--------------------------------|--|
| Step Name | The name of this step as it appears in the transformation workspace. |
| Host name or IP | Connection host name input field. |
| Port | Connection host port number input field. |
| Database | Name of the incoming database. |
| Design document | Identify the source design document. Design documents are a special type of CouchDB document that contains application code. See http://guide.couchdb.org for more information about design documents in CouchDB. |
| View name | Identify the source CouchDB view. For more on views in CouchDB, see http://guide.couchdb.org/editions/1/en/views.html#views . |
| Authentication user | The username required to access the database. |
| Authentication password | The password required to access the database. |

Hadoop File Input

The Hadoop File Input step is used to read data from a variety of different text-file types stored on a Hadoop cluster. The most commonly used formats include comma separated values (CSV files) generated by spreadsheets and fixed width flat files.

This step enables you to specify a list of files to read, or a list of directories with wild cards in the form of regular expressions. In addition, you can accept file names from a previous step.

These tables describe all available Hadoop File Input options.

File Tab Options

| Option | Description |
|---------------------------|--|
| Step Name | Optionally, you can change the name of this step to fit your needs. Every step in a transformation must have a unique name. |
| File or Directory | Specifies the location and/or name of the text file to read. Click Browse to navigate to the file, select Hadoop in the file dialog to enter in your Hadoop credentials, and click Add to add the file/directory/wildcard combination to the list of selected files (grid). |
| Regular expression | Specify the regular expression you want to use to select the files in the directory specified in the previous option. |

| Option | Description |
|--|--|
| | For example, you want to process all files that have a .txt output. |
| Selected Files | Contains a list of selected files (or wild card selections) along with a property specifying if a file is required or not. If a file is required and it isn't found, an error is generated. Otherwise, the file name is skipped. |
| Show filenames(s)... | Displays a list of all files that are loaded based on the current selected file definitions. |
| Show file content | Displays the raw content of the selected file. |
| Show content from first data line | Displays the content from the first data line for the selected file. |

Selecting file using Regular Expressions... The Text File Input step can search for files by wildcard in the form of a regular expression. Regular expressions are more sophisticated than using '*' and '?' wildcards. This table describes a few examples of regular expressions.

| File Name | Regular Expression | Files selected |
|-----------|----------------------|---|
| /dirA/ | .userdata\.txt | Find all files in /dirA/ with names containing user data and ending with .txt |
| /dirB/ | AAA.* | Find all files in /dirB/ with names that start with AAA |
| /dirC/ | [ENG:A-Z][ENG:0-9].* | Find all files in /dirC/ with names that start with a capital and followed by a digit (A0-Z9) |

Accepting file names from a previous step... This option allows even more flexibility in combination with other steps, such as Get File Names. You can specify your file name and pass it to this step. This way the file name can come from any source; a text file, database table, and so on.

| Option | Description |
|---|--|
| Accept file names from previous steps | Enables the option to get file names from previous steps |
| Step to read file names from | Step from which to read the file names |
| Field in the input to use as file name | Text File Input looks in this step to determine which filenames to use |

Content Tab

Options under the **Content** tab allow you to specify the format of the text files that are being read. This table is a list of the options associated with this tab.

| Option | Description |
|------------------|---|
| File type | Can be either CSV or Fixed length. Based on this selection, Spoon launches a different helper GUI when you click Get Fields in the Fields tab. |
| Separator | One or more characters that separate the fields in a single line of text. Typically this is a semicolon (;) or a tab. |
| Enclosure | Some fields can be enclosed by a pair of strings to allow separator characters in fields. The enclosure string is optional. If you use repeat an enclosures allow text line 'Not the nine o'clock news.'. With ' the enclosure string, this gets parsed as Not the nine o'clock news. |

| Option | Description |
|---|---|
| Allow breaks in enclosed fields? | Not implemented |
| Escape | Specify an escape character (or characters) if you have these types of characters in your data. If you have a backslash (/) as an escape character, the text 'Not the nine o'clock news' (with a single quote ['] as the enclosure) gets parsed as Not the nine o'clock news. |
| Header & number of header lines | Enable if your text file has a header row (first lines in the file). You can specify the number of times the header lines appears. |
| Footer & number of footer lines | Enable if your text file has a footer row (last lines in the file). You can specify the number of times the footer row appears. |
| Wrapped lines and number of wraps | Use if you deal with data lines that have wrapped beyond a specific page limit. Headers and footers are never considered wrapped. |
| Paged layout and page size and doc header | Use these options as a last resort when dealing with texts meant for printing on a line printer. Use the number of document header lines to skip introductory texts and the number of lines per page to position the data lines |
| Compression | Enable if your text file is in a Zip or GZip archive. Only the first file in the archive is read. |
| No empty rows | Do not send empty rows to the next steps. |
| Include file name in output | Enable if you want the file name to be part of the output |
| File name field name | Name of the field that contains the file name |
| Rownum in output? | Enable if you want the row number to be part of the output |
| Row number field name | Name of the field that contains the row number |
| Format | Can be either DOS, UNIX, or mixed. UNIX files have lines that are terminated by line feeds. DOS files have lines separated by carriage returns and line feeds. If you specify mixed, no verification is done. |
| Encoding | Specify the text file encoding to use. Leave blank to use the default encoding on your system. To use Unicode, specify UTF-8 or UTF-16. On first use, Spoon searches your system for available encodings. |
| Be lenient when parsing dates? | Disable if you want strict parsing of data fields. If case-lenient parsing is enabled dates like Jan 32nd become Feb 1st. |
| The date format Locale | This locale is used to parse dates that have been written in full such as "February 2nd, 2006." Parsing this date on a system running in the French (fr_FR) locale would not work because February is called Février in that locale. |
| Add filenames to result | Adds filenames to result filenames list. |

Error Handling Tab

Options under the **Error Handling** tab allow you to specify how the step reacts when errors occur, such as, malformed records, bad enclosure strings, wrong number of fields, premature line ends. This describes the options available for Error handling.

| Option | Description |
|---|---|
| Ignore errors? | Enable if you want to ignore errors during parsing |
| Skip error lines | Enable if you want to skip those lines that contain errors. You can generate an extra file that contains the line numbers on which the errors occur. Lines with errors are not skipped. The fields that have parsing errors are empty (null). |
| Error count field name | Add a field to the output stream rows. This field contains the number of errors on the line. |
| Error fields field name | Add a field to the output stream rows; this field contains the field names on which an error occurred. |
| Error text field name | Add a field to the output stream rows; this field contains the descriptions of the parsing errors that have occurred. |
| Warnings file directory | When warnings are generated, they are placed in this directory. The name of that file is <warning_dir>/filename.<date_time>.<warning extension> |
| Error files directory | When errors occur, they are placed in this directory. The name of the file is <errorfile_dir>/filename.<date_time>.<errorfile_extension> |
| Failing line numbers files directory | When a parsing error occurs on a line, the line number is placed in this directory. The name of that file is <errorline_dir>/filename.<date_time>.<errorline extension> |

Filters Tab

Options under the **Filters** tab enables you to specify the lines you want to skip in the text file. This table describes the available options for defining filters.

| Option | Description |
|------------------------|---|
| Filter string | The string for which to search. |
| Filter position | The position where the filter string must be placed in the line. Zero (0) is the first position in the line. If you specify a value below zero (0), the filter string is searched for in the entire string. |
| Stop on filter | Specify Y here if you want to stop processing the current text file when the filter string is encountered. |
| Positive match | Turns filters into positive mode when turned on. Only lines that match this filter will be passed. Negative filters take precedence and are immediately discarded. |

Fields Tab

The options under the **Fields** tab allow you to specify the information about the name and format of the fields being read from the text file. Available options include:

| Option | Description |
|------------------|--|
| Name | Name of the field. |
| Type | Type of the field can be either String, Date or Number. |
| Format | See Number Formats for a complete description of format symbols. |
| Length | For Number: Total number of significant figures in a number. For String: total length of string. For Date: length of printed output of the string, for instance, 4 only gives back the year. |
| Precision | For Number: Number of floating point digits. For String, Date, Boolean: unused. |

| Option | Description |
|-----------------|--|
| Currency | Used to interpret numbers like \$10,000.00 or E5.000,00. |
| Decimal | A decimal point can be a "." (10;000.00) or "," (5.000,00). |
| Grouping | A grouping can be a dot "," (10;000.00) or "." (5.000,00). |
| Null if | Treat this value as <code>null</code> . |
| Default | Default value in case the field in the text file was not specified (empty). |
| Trim | Type trim this field, left, right, both, before processing. |
| Repeat | If the corresponding value in this row is empty, repeat the one from the last time it was not empty (Y/N). |

Number formats... The information about number formats was taken from the Sun Java API documentation, [Decimal Formats](#).

| Symbol | Location | Localized | Meaning |
|----------|----------------------|-----------|--|
| 0 | Number | Yes | Digit |
| # | Number | Yes | Digit, zero shows as absent |
| . | Number | Yes | Decimal separator or monetary decimal separator |
| - | Number | Yes | Minus sign |
| , | Number | Yes | Grouping separator |
| E | Number | Yes | Separates mantissa and exponent in scientific notation. Need not be quoted in prefix or suffix. |
| ; | Sub pattern boundary | Yes | Separates positive and negative sub patterns |
| % | Prefix or suffix | Yes | Multiply by 100 and show as percentage |
| \u2030 | Prefix or suffix | Yes | Multiply by 1000 and show as per mille |
| (\u00A4) | Prefix or suffix | No | Currency sign, replaced by currency symbol. If doubled, replaced by international currency symbol. If present in a pattern, the monetary decimal separator is used instead of the decimal separator. |
| ' | Prefix or suffix | No | Used to quote special characters in a prefix or suffix, for example, "'###" formats 123 to "#123". To create a single quote itself, use two in a row: "' o'clock". |

Scientific Notation... In a pattern, the exponent character immediately followed by one or more digit characters indicates scientific notation, for example "0.###E0" formats the number 1234 as "1.234E3".

Date formats... The information about Date formats was taken from the Sun Java API documentation, [Date Formats](#).

| Letter | Date or Time Component | Presentation | Examples |
|--------|------------------------|-------------------|--|
| G | Era designator | Text | AD |
| y | Year | Year | 1996 or 96 |
| M | Month in year | Month | July, Jul, or 07 |
| w | Week in year | Number | 27 |
| W | Week in month | Number | 2 |
| D | Day in year | Number | 189 |
| d | Day in month | Number | 10 |
| F | Day of week in month | Number | 2 |
| E | Day in week | Text | Tuesday or Tue |
| a | Am/pm marker | Text | PM |
| H | Hour in day (0-23) | Number 0 | n/a |
| k | Hour in day (1-24) | Number 24 | n/a |
| K | Hour in am/pm (0-11) | Number 0 | n/a |
| h | Hour in am/pm (1-12) | Number 12 | n/a |
| m | Minute in hour | Number 30 | n/a |
| s | Second in minute | Number 55 | n/a |
| S | Millisecond | Number 978 | n/a |
| z | Time zone | General time zone | Pacific Standard Time, PST, or GMT-08:00 |
| Z | Time zone | RFC 822 time zone | -0800 |

Hadoop File Output

The Hadoop File Output step is used to export data to text files stored on a Hadoop cluster. This is commonly used to generate comma separated values (CSV files) that can be read by spreadsheet applications. It is also possible to generate fixed width files by setting lengths on the fields in the fields tab.

These tables describe all available Hadoop File Output options.

File Tab


The options under the **File** tab is where you define basic properties about the file being created.

| Option | Description |
|------------------|--|
| Step name | Optionally, you can change the name of this step to fit your needs. Every step in a transformation must have a unique name. |
| Filename | Specifies the location and/or name of the text file to which to write. Click Browse to navigate to the file. Select Hadoop in the file dialogue to enter in your Hadoop credentials. |
| Extension | Adds a point and the extension to the end of the file name (.txt). |

| Option | Description |
|------------------------------------|---|
| Accept file name from field? | Enables you to specify the file name(s) in a field in the input stream. |
| File name field | When the previous option is enabled, you can specify the field that contains the filename(s) at runtime. |
| Include stepnr in filename | If you run the step in multiple copies (Launching several copies of a step), the copy number is included in the file name before the extension. (_0). |
| Include partition nr in file name? | Includes the data partition number in the file name. |
| Include date in file name | Includes the system date in the filename (_20101231) |
| Include time in file name | Includes the system time in the filename (_235959) |
| Specify Date time format | Allows you to specify the date time format from the list within the Date time format dropdown list.. |
| Date time format | Dropdown list of date format options. |
| Show file name(s) | Displays a list of the files that are generated. This is a simulation and depends on the number of rows that go into each file. |

Content tab


The **Content** tab contains these options for describing the content being read.

| Option | Description |
|------------------------------------|---|
| Append | Enables to append lines to the end of the specified file. |
| Separator | Specifies the character that separates the fields in a single line of text. Typically this is semicolon (;) or a tab. |
| Enclosure | A pair of strings can enclose some fields. This allows separator characters in fields. The enclosure string is optional. Enable if you want the text file to have a header row (first line in the file). |
| Force the enclosure around fields? | Forces all field names to be enclosed with the character specified in the Enclosure property above |
| Header | Enable this option if you want the text file to have a header row (first line in the file) |
| Footer | Enable this option if you want the text file to have a footer row (last line in the file) |
| Format | Can be either DOS or UNIX; UNIX files have lines are separated by line feeds, DOS files have lines separated by carriage returns and line feeds |
| Encoding | Specify the text file encoding to use. Leave blank to use the default encoding on your system. To use Unicode, specify UTF-8 or UTF-16. On first use, Spoon searches your system for available encodings. |
| Compression | Specify the type of compression, .zip or .gzip to use when compressing the output.  Note: Only one file is placed in a single archive. |

| Option | Description |
|---------------------------------------|---|
| Fast data dump (no formatting) | Improves the performance when dumping large amounts of data to a text file by not including any formatting information. |
| Split every ... rows | If the number N is larger than zero, split the resulting text-file into multiple parts of N rows. |
| Add Ending line of file | Allows you to specify an alternate ending row to the output file. |

Fields tab

The fields tab is where you define properties for the fields being exported. The table below describes each of the options for configuring the field properties:

| Option | Description |
|----------------------|---|
| Name | The name of the field |
| Type | Type of the field can be either String, Date or Number. |
| Format | The format mask to convert with. See Number Formats for a complete description of format symbols. |
| Length | The length option depends on the field type follows: <ul style="list-style-type: none"> • Number - Total number of significant figures in a number • String - total length of string • Date - length of printed output of the string (for example, 4 returns year) |
| Precision | The precision option depends on the field type as follows: <ul style="list-style-type: none"> • Number - Number of floating point digits • String - unused • Date - unused |
| Currency | Symbol used to represent currencies like \$10,000.00 or E5.000,00 |
| Decimal | A decimal point can be a "." (10,000.00) or "," (5.000,00) |
| Group | A grouping can be a "," (10,000.00) or "." (5.000,00) |
| Trim type | The trimming method to apply on the string  Note: Trimming works when there is no field length given only. |
| Null | If the value of the field is null, insert this string into the text file |
| Get | Click to retrieve the list of fields from the input fields stream(s) |
| Minimal width | Change the options in the Fields tab in such a way that the resulting width of lines in the text file is minimal. So instead of save 0000001, you write 1, and so on. String fields will no longer be padded to their specified length. |

HBase Input

This step reads data from an HBase table according to user-defined column metadata.

Configure Query

This tab contains connection details and basic query information. You can configure a connection in one of two ways: either via a comma-separated list of hostnames where the zookeeper quorum reside, or via an **hbase-site.xml** (and, optionally, **hbase-default.xml**) configuration file. If both zookeeper and HBase XML configuration options are supplied, then the zookeeper takes precedence.

| Option | Definition |
|---|---|
| Step name | The name of this step as it appears in the transformation workspace. |
| Zookeeper host(s) | Comma-separated list of hostnames for the zookeeper quorum. |
| URL to hbase-site.xml | Address of the hbase-site.xml file. |
| URL to hbase-default.xml | Address of the hbase-default.xml file. |
| HBase table name | The source HBase table to read from. Click Get Mapped Table Names to populate the drop-down list of possible table names. |
| Mapping name | A mapping to decode and interpret column values. Click Get Mappings For the Specified Table to populate the drop-down list of available mappings. |
| Start key value (inclusive) for table scan | A starting key value to retrieve rows from. This is inclusive of the value entered. |
| Stop key value (exclusive) for table scan | A stopping key value for the scan. This is exclusive of the value entered. Both fields or the stop key field may be left blank. If the stop key field is left blank, then all rows from (and including) the start key will be returned. |
| Scanner row cache size | The number of rows that should be cached each time a fetch request is made to HBase. Leaving this blank uses the default, which is to perform no caching; one row would be returned per fetch request. Setting a value in this field will increase performance (faster scans) at the expense of memory consumption. |
| # | The order of query limitation fields. |
| Alias | The name that the field will be given in the output stream. |
| Key | Indicates whether the field is the table's key field or not. |
| Column family | The column family in the HBase source table that the field belongs to. |
| Column name | The name of the column in the HBase table (family + column name uniquely identifies a column in the HBase table). |
| Type | The PDI data type for the field. |
| Format | A formatting mask to apply to the field. |
| Indexed values | Indicates whether the field has a predefined set of values that it can assume. |

| Option | Definition |
|----------------------------|--|
| Get Key/Fields Info | Assuming the connection information is complete and valid, this button will populate the field list and display the name of the key. |

Create/Edit Mappings

This tab creates or edits a mapping for a given HBase table. A mapping simply defines metadata about the values that are stored in the table. Since most information is stored as raw bytes in HBase, this enables PDI to decode values and execute meaningful comparisons for column-based result set filtering.

| Option | Definition |
|-------------------------|--|
| HBase table name | Displays a list of table names. Connection information in the previous tab must be valid and complete in order for this drop-down list to populate. |
| Mapping name | Names of any mappings that exist for the table. This box will be empty if there are no mappings defined for the selected table, in which case you can enter the name of a new mapping. |
| # | The order of the mapping operation. |
| Alias | The name you want to assign to the HBase table key. This is required for the table key column, but optional for non-key columns. |
| Key | Indicates whether or not the field is the table's key. |
| Column family | The column family in the HBase source table that the field belongs to. Non-key columns must specify a column family and column name. |
| Column name | The name of the column in the HBase table. |
| Type | Data type of the column. Key columns can be of type: String Integer Unsigned integer (positive only) Long Unsigned long (positive only) Date Unsigned date. Non-key columns can be of type: String, Integer, Long, Float, Double, Boolean, Date, BigInteger, Serializable, Binary. |
| Indexed values | String columns may optionally have a set of legal values defined for them by entering comma-separated data into this field. |

Filter Result Set

This tab provides two fields that limit the range of key values returned by a table scan. Leaving both fields blank will result in all rows being retrieved from the source table.

| Option | Definition |
|------------------------------|--|
| Match all / Match any | When multiple column filters have been defined, you have the option returning only those rows that match all filters, or any single filter. Bounded ranges on a single numeric column can be defined by defining two filters (upper and lower bounds) and selecting Match all ; similarly, open-ended ranges can be defined by selecting Match any . |
| # | The order of the filter operation. |
| Alias | A drop-down box of column alias names from the mapping. |

| Option | Definition |
|--------------------------|---|
| Type | Data type of the column. This is automatically populated when you select a field after choosing the alias. |
| Operator | A drop-down box that contains either equality/inequality operators for numeric, date, and boolean fields; or substring and regular expression operators for string fields. |
| Comparison value | A comparison constant to use in conjunction with the operator. |
| Format | A formatting mask to apply to the field. |
| Signed comparison | Specifies whether or not the comparison constant and/or field values involve negative numbers (for non-string fields only). If field values and comparison constants are only positive for a given filter, then HBase's native lexicographical byte-based comparisons are sufficient. If this is not the case, then it is necessary for column values to be deserialized from bytes to actual numbers before performing the comparison. |

Performance Considerations

Specifying fields in the Configure query tab will result in scans that return just those columns. Since HBase is a sparse column-oriented database, this requires that HBase check to see whether each row contains a specific column. More lookups equate to reduced speed, although the use of Bloom filters (if enabled on the table in question) mitigates this to a certain extent. If, on the other hand, the fields table in the Configure query tab is left blank, it results in a scan that returns rows that contain all columns that exist in each row (not only those that have been defined in the mapping). However, the HBase Input step will only emit those columns that are defined in the mapping being used. Because all columns are returned, HBase does not have to do any lookups. However, if the table in question contains many columns and is dense, then this will result in more data being transferred over the network.

HBase Output

This step writes data to an HBase table according to user-defined column metadata.

Configure Connection

This tab contains HBase connection information. You can configure a connection in one of two ways: either via a comma-separated list of hostnames where the zookeeper quorum reside, or via an **hbase-site.xml** (and, optionally, **hbase-default.xml**) configuration file. If both zookeeper and HBase XML configuration options are supplied, then the zookeeper takes precedence.

| Option | Definition |
|---------------------------------|--|
| Step name | The name of this step as it appears in the transformation workspace. |
| Zookeeper host(s) | Comma-separated list of hostnames for the zookeeper quorum. |
| URL to hbase-site.xml | Address of the hbase-site.xml file. |
| URL to hbase-default.xml | Address of the hbase-default.xml file. |
| HBase table name | The HBase table to write to. Click Get Mapped Table Names to populate the drop-down list of possible table names. |

| Option | Definition |
|-------------------------------------|---|
| Mapping name | A mapping to decode and interpret column values. Click Get Mappings For the Specified Table to populate the drop-down list of available mappings. |
| Disable write to WAL | Disables writing to the Write Ahead Log (WAL). The WAL is used as a lifeline to restore the status quo if the server goes down while data is being inserted. Disabling WAL will increase performance. |
| Size of write buffer (bytes) | The size of the write buffer used to transfer data to HBase. A larger buffer consumes more memory (on both the client and server), but results in fewer remote procedure calls. The default (in the hbase-default.xml) is 2MB (2097152 bytes), which is the value that will be used if the field is left blank. |

Create/Edit Mappings

This tab creates or edits a mapping for a given HBase table. A mapping simply defines metadata about the values that are stored in the table. Since just about all information is stored as raw bytes in HBase, this allows PDI to decode values and execute meaningful comparisons for column-based result set filtering.



Note: The names of fields entering the step are expected to match the aliases of fields defined in the mapping. All incoming fields must have a matching counterpart in the mapping. There may be fewer incoming fields than defined in the mapping, but if there are more incoming fields then an error will occur. Furthermore, one of the incoming fields must match the key defined in the mapping.

| Option | Definition |
|----------------------------|--|
| HBase table name | Displays a list of table names. Connection information in the previous tab must be valid and complete in order for this drop-down list to populate. |
| Mapping name | Names of any mappings that exist for the table. This box will be empty if there are no mappings defined for the selected table, in which case you can enter the name of a new mapping. |
| # | The order of the mapping operation. |
| Alias | The name you want to assign to the HBase table key. This is required for the table key column, but optional for non-key columns. |
| Key | Indicates whether or not the field is the table's key. |
| Column family | The column family in the HBase source table that the field belongs to. Non-key columns must specify a column family and column name. |
| Column name | The name of the column in the HBase table. |
| Type | Data type of the column. Key columns can be of type: String Integer Unsigned integer (positive only) Long Unsigned long (positive only) Date Unsigned date. Non-key columns can be of type: String, Integer, Long, Float, Double, Boolean, Date, BigInteger, Serializable, Binary. |
| Indexed values | String columns may optionally have a set of legal values defined for them by entering comma-separated data into this field. |
| Get incoming fields | Retrieves a field list using the given HBase table and mapping names. |


Performance Considerations

The **Configure connection** tab provides a field for setting the size of the write buffer used to transfer data to HBase. A larger buffer consumes more memory (on both the client and server), but results in fewer remote procedure calls. The default (defined in the hbase-default.xml file) is 2MB. When left blank, the buffer is 2MB, **auto flush** is enabled, and **Put** operations are executed immediately. This means that each row will be transmitted to HBase as soon as it arrives at the step. Entering a number (even if it is the same as the default) for the size of the write buffer will disable auto flush and will result in incoming rows only being transferred once the buffer is full.

There is also a checkbox for disabling writing to the **Write Ahead Log (WAL)**. The WAL is used as a lifeline to restore the status quo if the server goes down while data is being inserted. However, the tradeoff for error-recovery is speed.

The **Create/edit mappings** tab has options for creating new tables. In the **HBase table name** field, you can suffix the name of the new table with parameters for specifying what kind of compression to use, and whether or not to use Bloom filters to speed up lookups. The options for compression are: NONE, GZ and LZO; the options for Bloom filters are: NONE, ROW, ROWCOL. If nothing is selected (or only the name of the new table is defined), then the default of NONE is used for both compression and Bloom filters. For example, the following string entered in the HBase table name field specifies that a new table called "NewTable" should be created with GZ compression and ROWCOL Bloom filters:

```
NewTable@GZ@ROWCOL
```

 **Note:** Due to licensing constraints, HBase does not ship with LZO compression libraries. These must be manually installed on each node if you want to use LZO compression.

HBase Row Decoder

The HBase Row Decoder step decodes an incoming key and HBase result object according to a mapping.

| Option | Definition |
|------------------|--|
| Step Name | The name the step as it appears in the transformation workspace. |

Configure fields tab

| Option | Definition |
|---------------------------|---|
| Key field | Input key field. |
| HBase result field | Field containing the serialized HBase result. |

Create/Edit mappings tab

| Option | Definition |
|-------------------------|--|
| Zookeeper host | Hostname for the zookeeper quorum. |
| Zookeeper port | Database entry port for the zookeeper quorum. |
| HBase table name | Displays a list of table names which have mappings defined for them. |
| Mapping name | Names of any mappings that exist for the table. This box will be empty if there are no mappings defined for the selected table. You can define a mapping from scratch or use the connection fields to access any mappings already saved into HBase. |
| Save mapping | Saves the mapping in HBase as long as valid connection details were provided and the mapping was named. If the mapping was only needed locally then connection details and mapping name are not needed, the mapping will be serialized into the transformation metadata automatically. |

| Option | Definition |
|-------------------------|---|
| Delete mapping | Deletes the mapping. |
| Create a tuple template | Partially populates the table with special fields that define a tuple mapping for use in the tuple output mode. Tuple output mode allows the step to output all the data in wide HBase rows where the number of columns may vary from row to row. It assumes that all column values are of the same type. A tuple mapping consists of the following output fields: KEY, Family, Column, Value and Timestamp. The type for "Family" and "Timestamp" is preconfigured to "String" and "Long" respectively. You must provide the types for "KEY", "Column" (column name) and "Value" (column value). The default behavior is to output all column values in all column families. |

MapReduce Input

This step defines the key/value pairs for Hadoop input. The output of this step is appropriate for whatever data integration transformation tasks you need to perform.

| Option | Definition |
|-------------|---|
| Step name | The name of this step as it appears in the transformation workspace. |
| Key field | The Hadoop input field and data type that represents the key in MapReduce terms. |
| Value field | The Hadoop input field and data type that represents the value in MapReduce terms. |


MapReduce Output

This step defines the key/value pairs for Hadoop output. The output of this step will become the output to Hadoop, which changes depending on what the transformation is used for.

If this step is included in a transformation used as a **mapper** and there is a combiner and/or reducer configured, the output will become the input pairs for the combiner and/or reducer. If there are no combiner or reducers configured the output is passed to the format configured for the job it was executed with.

If this step is included in a transformation used as a **combiner** and there is a reducer configured, the output will become the input pairs for the reducer. If no reducer configured, the output is passed to the format configured for the job it was executed with.

If this step is included in a transformation used as a **reducer**, then the output is passed to the format configured for the job it was executed with.

 **Note:** You are not able to define the data type for the key or value here; it is defined earlier in your transformation. However, a reducer or combiner that takes this output as its input will have to know what the key and value data types are, so you may need to make note of them somehow.

| Option | Definition |
|-------------|--|
| Step name | The name of this step as it appears in the transformation workspace. |
| Key field | The Hadoop output field that represents the key in MapReduce terms. |
| Value field | The Hadoop output field that represents the value in MapReduce terms. |

MongoDB Input

The **MongoDB Input** transformation step enables you to retrieve [documents](#) or records from a [collection](#) within MongoDB. For additional information about MongoDB, see the [MongoDB documentation](#).

Configure connection tab

The **Configure connection** tab enables you to specify the database and collection to query.

| Option | Definition |
|---|--|
| Step name | Name of the step as it appears in the transformation workspace. |
| Host name(s) or IP address(es) | Indicates the network name or address of the MongoDB instance or instances. You can input multiple host names or IP addresses, separated by a comma. You can also specify a different port number for each host name by separating the host name and port number with a colon, and separating each combination of host name and port number with a comma. For example, to include the host name and port number for two different MongoDB instances, you would input <code>localhost1:27017,localhost2:27018</code> and leave the Port field empty. |
| Use all replica set members/mongos | Differentiates between a replica set containing one node and a stand-alone single Mongo host. If there is a replica set, and it contains more than one host, then the Java driver discovers all hosts automatically. It is good practice to list more than one replica set host in the hosts field so that the driver has a better chance of connecting successfully if one is down. |
| Port | Indicates the port number of the MongoDB instance or instances. Specify a default port to use if no port numbers are specified in the Host name(s) or IP address(es) field. |
| Username | Indicates the username required to access the database. |
| Password | Indicates the password associated with the provided Username . |
| Authenticate using Kerberos | Indicates whether to use the Kerberos service to manage the authentication process. |
| Connection timeout | Designates how long to wait for a connection to a database (in milliseconds) before terminating the connection attempt. Leave blank to never terminate the connection. |
| Socket timeout | Designates how long to wait for a write operation (in milliseconds) before terminating the operation. Leave blank to never terminate the operation. |
| Preview | Displays a first look of the data. Clicking Preview causes the Enter preview size window to appear. Enter the maximum number of records that you want to preview, then click OK . The preview data appears in the Examine preview data window. |

Input Options tab

The **Input Options** tab enables you to specify which database and collection you want to retrieve information from. You can also indicate the read preferences and tag sets in this tab. See [Tag Sets](#) for more information.

| Option | Definition |
|--|--|
| Database | Name of the database to retrieve data from. Click Get DBs to populate the drop-down menu with a list of databases on the server. |
| Collection | Name of the collection to retrieve data from. Click Get collections to populate the drop-down menu with a list of collections within the database. |
| Read preference | Indicates which node to read first— Primary , Primary preferred , Secondary , Secondary preferred , or Nearest . |
| Tag set specification/#/Tag Set | Tags allow you to customize write concerns and read preferences for a replica set. The Tag set specification section of the window allows you to specify criteria for selecting replica set members. When you click Get tags , the Tag set specification populates with the tag sets that are available on the database, in order of execution. You can join, delete, copy, or paste tag sets, then click Test tag set to see which replica set members match the Tag set specification criteria you specified. The # field indicates the number of the tag set. The Tag set field displays the tag set criteria. |
| Get Tags | Retrieves a list of the tag sets that are in the database indicated in the Database field. |
| Join tags | Appends selected tag sets so that nodes that match the criteria are queried or written to simultaneously. If you select individual tag sets, then click Join tags , the tag sets are combined to create one tag set. Note that this change only occurs in the MongoDB Input window, not on the database. |
| Test tag set | Displays the set members that match the tags indicated in the tag set specification. Clicking Test tag set displays the id, host name, priority, and tags for each replica set member that matches the tag set specification criteria. |

Query tab

The **Query** tab enables you to refine your read request. This tab operates in two different modes. You can create a query using JSON Query expression or using the Aggregation Framework. By default, the **Query** tab is in JSON Query expression mode. You can enter a JSON Query expression when the **Query is aggregation pipeline** checkbox is deselected. MongoDB queries use a *JSON-like* query language that includes a variety of [query operators](#). To place the Aggregation Framework mode **Query is aggregation pipeline** checkbox. You can then enter a query, using the Aggregation Framework, in the **Aggregation pipeline specification** field that appears. See [MongoDB's Aggregation Framework](#) for additional information, including code examples.

| Option | Definition |
|--|--|
| Query expression (JSON) <i>(Field is visible if Query is aggregation pipeline checkbox is not selected.)</i> | JSON expression to limit the output. See the sub-section Query Examples (JSON Query Expressions) for additional details. |
| Aggregation pipeline specification (JSON) <i>(Field is visible if Query is aggregation pipeline checkbox is selected.)</i> | Use this field if you want to use the MongoDB Aggregation Framework to perform a simple or complex aggregations or selections such as totalling or averaging field values. |

| Option | Definition |
|--|---|
| | Note that the method name (which includes the collection name of the database you selected in the Input Options tab), appears after the Aggregation pipeline specification (JSON) label for this field. See the sub-section Query Examples (JSON Aggregate Pipeline) for additional details. |
| Query is aggregation pipeline | Pipes multiple JSON expressions together to execute at once. An aggregation pipeline strings several JSON expressions together, with the output of the previous expression becoming the input for the next. When selected, the Aggregation pipeline specification (JSON) field appears. When deselected, the Query expression (JSON) field appears. |
| Execute for each row | Perform the query on each row of data. |
| Fields expression (JSON) <i>(Field is visible if Query is aggregation pipeline check box is not selected.)</i> | This field becomes active only if Query is aggregation pipeline is <i>not</i> selected. Controls the fields to return, or in MongoDB terms, the projection. If empty, all fields are returned. Enter <code>true</code> or <code>false</code> after the fields to indicate selected or not, respectively. See the MongoDB documentation [http://docs.mongodb.org/manual/reference/method/db.collection.find/] for more information about projections. |

Fields Tab

The **Fields** tab enables you to define properties for the exported fields. The **Fields** tab operates in two different modes that impact how query results are formatted. You can indicate that you want the query result to be stored in a single JSON field. To do this, click the **Output single JSON field** check box. If you decide to do this and you want to parse the results of the field, you can apply a transformation step later in the process. Or, you can uncheck the **Output single JSON field** check box and instead, click the **Get Fields** button to apply Pentaho's Schema on Read functionality. This functionality parses fields, paths, and data types and displays them. You can then review and adjust this information, as needed.

| Option | Definition |
|---|---|
| Output single JSON field | Indicates whether the JSON result of the query should be outputted to a single field that has the String data type. You can parse this JSON using the JSON Input transformation step, <code>eval("{" + jsonString + "}")</code> in JavaScript, or using a User Defined Java Class step . |
| Name of JSON output field <i>(Field is active if Output single JSON field check box is selected.)</i> | Designates the name of the field that contains the JSON output from the server. |
| Get fields <i>(Field is active if Output single JSON field check box is not selected.)</i> | Creates a sample set of documents, then displays the name and field for each record. Pentaho's Schema on Read functionality determines the field names, paths, and the data type for each field in the sample. |
| # <i>(Field is active if Output single JSON field check box is not selected.)</i> | The order of this entry in the list. |
| Name <i>(Field is active if Output single JSON field check box is not selected.)</i> | Displays a user-friendly name of the field that is based on the value in the Path field. The name that appears here maps the name of the field as it appears in the PDI transformation with the field that appears in the MongoDB database. You can edit the name as desired. |
| Path <i>(Field is active if Output single JSON field check box is not selected.)</i> | Indicates the JSON path of the field in MongoDB. If the path shown is an array, you can specify a specific |

| Option | Definition |
|---|--|
| | element in the array by passing it the key value, which is contained in the bracketed part of the array. For example <code>\$.emails[0]</code> indicates that you want the result to display the first value in the array. <code>\$.emails[1]</code> indicates that you want the result to display the second value in the array and so forth. If you want to display all array values, use the asterisk as the key, like this <code>\$.email[*]</code> . If the array contains records, and not just strings, you can specify that you want to display the record like this: <code>\$.emails[*].sender</code> . |
| Type (Field is active if Output single JSON field check box is not selected.) | Indicates the data type. |
| Indexed Values (Field is active if Output single JSON field check box is not selected.) | Allows you to enter a comma-separated list of legal values for String fields. If you specify values in this field, the Kettle indexed data type is applied to the data. If not, the String data type is applied. Usually, you will only need to modify this field if you are using Weka metadata for nominal fields. |
| Sample: array min: max index (Field is active if Output single JSON field check box is not selected.) | Indicates minimum and maximum values for the index seen in the sampled documents. |
| Sample: #occur/#docs (Field is active if Output single JSON field check box is not selected.) | Indicates how often the field occurs as well as the number of documents processed. |
| Sample: disparate types (Field is active if Output single JSON field check box is not selected.) | If several documents are sampled, but the same field contain different data types, the Sample: disparate types field is populated with a "Y." The Type field displays the String data type. In this instance, the Kettle type for the field in question is set to the String data type, so it is able to output values of differing types. |

Query Examples (JSON Query Expressions)

MongoDB has a rich query system that allows you to select and filter documents in a collection along specific fields and values. The [MongoDB Extended JASON](#) page in the MongoDB wiki space details how to use queries. Pentaho supports only the features discussed on this page. This table displays some examples of the syntax and structure of the queries you can use to request data from MongoDB.

| Query expression | Description |
|--|---|
| <code>{ name : "MongoDB" }</code> | Queries all values where the <code>name</code> field has a value equal to MongoDB |
| <code>{ name : { '\$regex' : "m.*", '\$options' : "i" } }</code> | Uses a regular expression to find <code>name</code> fields starting with m, case insensitive |
| <code>{ name : { '\$gt' : "M" } }</code> | Searches all strings greater than M |
| <code>{ name : { '\$lte' : "T" } }</code> | Searches all strings less than or equal to T |
| <code>{ name : { '\$in' : ["MongoDB", "MySQL"] } }</code> | Finds all names that are either MongoDB or MySQL |
| <code>{ name : { '\$nin' : ["MongoDB", "MySQL"] } }</code> | Finds all names that are either MongoDB or MySQL |
| <code>{ \$where : "this.count == 1" }</code> | Uses JavaScript to evaluate a condition |
| <code>{ \$query: {}, \$orderby: { age : -1 } }</code> | Returns all documents in the collection named collection sorted by the age field in descending order. |

Query Examples (JSON Aggregate Pipeline)

MongoDB has a rich query system that allows you to select and filter documents using the aggregation pipeline framework. The [Aggregation Framework Examples](#) page in the MongoDB wiki provides additional examples of function calls. This table displays some examples of the syntax and structure of the queries you can use to request data from MongoDB.

| Query expression | Description |
|---|--|
| <pre>{ \$match : {state : "FL", city : "ORLANDO" } }, { \$sort : {pop : -1 } }</pre> | Returns all fields from all documents where the <code>state</code> field has a value of <code>FL</code> and the <code>city</code> field has a value of <code>ORLANDO</code> . The documents will be returned sorted by the <code>pop</code> field in descending order. |
| <pre>{ \$group : { _id: "\$state" } }, { \$sort : { _id : 1 } }</pre> | Returns one field named <code>_id</code> containing the distinct values for <code>state</code> in ascending order. Similar to the SQL: <code>SELECT DISTINCT state AS _id FROM collection ORDER BY state ASC.</code> |
| <pre>{ \$match : {state : "FL" } }, { \$group: { _id: "\$city" , pop: { \$sum: "\$pop" } } }, { \$sort: { pop: -1 } }, { \$project: { _id : 0, city : "\$_id" } }</pre> | Gets all documents where the <code>state</code> field has a value of <code>FL</code> , aggregates all values of <code>pop</code> for each city, sorts by population descending and returns one field named <code>city</code> . |
| <pre>{ \$unwind : "\$result" }</pre> | Peels off the elements of an array individually, and returns one document for each element of the array. |

MongoDB Output

The MongoDB Output step enables you to insert data to a MongoDB collection and specify a number of options that control what and how data is written. These tables describe the available options within the **MongoDB Output** step.

Configure connection tab

The **Configure connection tab** is where you enter basic connection details. Click **Get DBs** and **Get collections** to retrieve the names of existing databases and collections within the connected database.

| Option | Definition |
|---|--|
| Step name | Name of this step as it appears in the transformation workspace |
| Host name(s) or IP address(es) | Indicates the network name or address of the MongoDB instance or instances. You can input multiple host names or IP addresses, separated by a comma. You can also specify a different port number for each host name by separating the host name and port number with a colon, and separating each combination of host name and port number with a comma. For example, to include the host name and port number for two different MongoDB instances, you would input <code>localhost1:27017,localhost2:27018</code> and leave the Port field empty. |
| Port | Indicates the port number of the MongoDB instance or instances. Use this to specify a default port if no ports are given as part of the Host name(s) or IP address(es) field. |
| Use all replica set members/mongos | Differentiates between a replica set containing one node and a stand-alone single Mongo host. If there is a replica set, and it contains more than one host, then the Java driver discovers all hosts automatically. It is good practice to list more than one replica set host in the hosts field |

| Option | Definition |
|------------------------------------|--|
| | so that the driver has a better chance of connecting successfully if one is down. |
| Username | Indicates the user name required to access the database |
| Password | Indicates the password associated with the provided Username . |
| Authenticate using Kerberos | Indicates whether to use the Kerberos service to manage the authentication process. |
| Connection timeout | Designates how long to wait for a connection to a database (in milliseconds) before terminating the connection attempt. Leave blank to never terminate the connection. |
| Socket timeout | Designates how long to wait for a write operation (in milliseconds) before terminating the operation. Leave blank to never terminate the operation. |

Output options tab

The **Output options** tab provides additional controls for inserting data into a MongoDB collection. If the specified collection does not exist, it is created before a document is inserted.

| Option | Definition |
|---------------------------------|---|
| Database | Name of the database to write data to. Click Get DBs to populate the drop-down menu with a list of databases on the server. |
| Collection | Name of the collection to write data to. Click Get collections to populate the drop-down menu with a list of collections within the database. |
| Batch insert size | Sets the batch size for fast bulk insert operations. If left blank, the default size is 100 rows. |
| Truncate collection | Deletes any existing data in the target collection before inserting begins. |
| Upsert | Changes the write mode from insert to upsert, which either updates the first document matched in the target collection or, if no document matches, inserts a new document into the target collection according to the incoming fields specified in the Mongo document fields tab . |
| Multi-update | Updates all matching documents, rather than just the first. |
| Modifier update | Enables modifier operators to be used to modify individual fields within matching documents. To set the Modifier operation see the Mongo document fields tab . |
| Write concern (w option) | http://docs.mongodb.org/manual/reference/glossary/#term-write-concern specifies the minimum number of servers that must succeed for a write operation. A value of -1 disables all acknowledgement of write operation errors. Zero (0) disables basic acknowledgment of write operations, but returns information about socket exceptions and networking errors. 1 provides acknowledgment of write operations on the primary node. >1 waits for successful write operations to the specified number of slaves, including the primary. |

| Option | Definition |
|--|---|
| w Time out | Designates how long to wait for a response to write operations (in milliseconds) before terminating the operation. Leave blank to never terminate. |
| Journalized writes | Writes the operation to the journal first, and after to the core data files. This confirms the write operation can survive a shutdown and ensures the write operation is durable. |
| Read preference | Indicates which node to read first— Primary , Primary preferred , Secondary , Secondary preferred , or Nearest |
| Number of retries for write operations | Indicates the number of times that a write operation is attempted. |
| Delay, in seconds, between retry attempts | Indicates the number of seconds between write operation retry attempts. |

Mongo document fields tab

The **Mongo document fields** tab enables you to define how field values which are coming into the step get written to a Mongo document. Configure the **Modifier policy** column in the **Mongo document fields** tab for control over when execution of a modifier operation affects a particular field. This can be particularly useful when the data for one Mongo document is split over several incoming PDI rows and in situations where it is not possible to execute different modifier operations that affect the same field simultaneously. The **Modifier policy** can be set to these values: `Insert&Update`, `Insert`, and `Update`. Only these modifier operations are supported: `$set`, `$inc`, and `$push`. You can set the **Modifier policy** to these values.

| Option | Definition |
|-------------------------------|---|
| # | The order of this entry in the list. |
| Name | The name of this field, descriptive of its content. |
| Mongo document path | Defines the hierarchical path to each field |
| Use field name | Specifies whether the incoming field name is used as the final entry in the path. When this is set to <code>Y</code> for a field, a preceding <code>.</code> (dot) is assumed. |
| JSON | Indicates if a field is in JSON format |
| Match field for upsert | Specifies which of the fields should be used for matching when performing an upsert operation. The first document in the collection that matches all fields tagged as <code>Y</code> in this column is replaced with the new document constructed with incoming values for all of the defined field paths. If a matching document does not exist, then a new document is inserted into the collection. <code>Insert&Update</code> : The operation gets executed whether or not a match exists in the collection according to the match conditions. <code>Insert</code> : The operation is executed on an insert only, for instance if a matching document does not exist. <code>Update</code> : Update only, for instance if the record exists. |
| Modifier operation | In-place modifications of existing document fields. Update more than one matching document by selecting the Modifier update option in conjunction with the Upsert option. Selecting the Multi-update option also enables each update to apply to all matching documents, rather than just the first. <code>\$set</code> —Sets the value of a field. Used to create the bulk of initial document structure for a new document. <code>\$inc</code> —If the field does not exist, sets the value |

| Option | Definition |
|-----------------------------------|--|
| | of a field. If the field exists, increases (or decreases, with a negative value) the value of a field. <code>\$push</code> —If the field does not exist, sets the value of a field. If the field exists, appends the value of a field. Used for appending to existing arrays in documents. |
| Modifier policy | Controls when execution of a modifier operation affects a particular field |
| Get fields | Populates the left-hand column of the table with the names of the incoming fields |
| Preview document structure | Displays the structure to be written to MongoDB in JSON format |

Create/drop indexes tab

The **Create/drop indexes tab** enables you to specify which indexes to create or remove. An index is a data structure that allows you to quickly locate documents based on the values stored in the specified fields. Fundamentally, indexes in MongoDB are similar to indexes in other database systems. MongoDB supports indexes on any field or sub-field contained in documents within a MongoDB collection.

Each row in the table can be used to create a single index (using one field) or a compound index (using multiple fields). The dot (.) notation is used to specify a path to a field to use in the index. This path can be optionally postfixed by a direction indicator. Compound indexes are specified by a comma-separated list of paths.

| Option | Definition |
|---------------------|---|
| # | The order of this field in the list. |
| Index fields | Specifies a single index (using one field) or a compound index (using multiple fields). The . (dot) notation is used to specify a path to a field to use in the index. This path can be optionally postfixed by a direction indicator, : 1 for ascending or : -1 for descending. Compound indexes are specified by a comma-separated list of paths. |
| Index opp | Specifies whether the index is created or dropped. |
| Unique | Indicates whether to display entries for documents that have a duplicate value for the indexed field. |
| Sparse | Indicates whether the index should contain only entries from those documents that have a value in the indexed field. |
| Show indexes | Displays the index information available. |

Further reading

See the [Big Data MongoDB Tutorials](#), or [MongoDB Output](#) section of the Pentaho Wiki for scenario-based examples of working with MongoDB and Pentaho.

Splunk Input

The **Splunk Input** transformation step enables you to connect to a Splunk server, enter a Splunk query, and get results back for use within a PDI Transformation. Once you have completed those steps, you can stream data from Splunk into your transformation. Make sure that you have read access to a Splunk server before you use the **Splunk Input** step. To learn more about Splunk see their [online documentation](#).

Configure connection tab

The **Configure connection** tab enables you to specify the database and collection to query.

| Option | Definition |
|---------------------------------------|--|
| Step name | Name of the step as it appears in the transformation workspace. |
| Host name(s) or IP address(es) | Indicates the network name or address of the Splunk instance or instances. |
| Port | Indicates the port number of the Splunk (splunkd) server. The default value is 8089. |
| Username | Indicates the username required to access the Splunk server. |
| Password | Indicates the password associated with the provided Username . |
| Execute for each row | If checked, a new query is issued for each row of data coming into the step. You can reference incoming fields of data using the <code>?{<Field>}</code> syntax. For instance, if you want to use the incoming field <code>Size</code> to drive the limit of results coming in, type this: <code>search *head ?{Size}</code> . |
| Splunk Query Expression | This is the definition of the splunk query. Note that unlike the queries defined in the Splunk user interface, you must start the query with the term <code>search</code> . Here is an example: <code>search * head 100</code> . One capability of Splunk search is field selection. This allows you to get access to Splunk-parsed fields within the <code>_raw</code> column. To select specific fields, use this syntax at the end of your defined search query: <code>... field index source OpCode</code> . |
| Preview | Provides a first look at the data. Clicking Preview causes the Enter preview size window to appear. Enter the maximum number of records that you want to preview, then click OK . The preview data appears in the Examine preview data window. |

Fields Tab

The **Fields** tab enables you to define properties for the exported fields.

| Option | Definition |
|--------------------|--|
| # | Number of the record returned. |
| Name | Name of the field. |
| Splunk name | Indicates the Splunk name for the field. |
| Type | Specifies the data type of the field. |
| Length | Indicates the length of the field. |
| Format | Specifies the format of the field. |
| Get Fields | Displays the field metadata and displays it in the Fields tab. After you have detected the field metadata using the Get Fields button on the Fields tab, you may choose to delete metadata fields that are not relevant to your specific query. Since each field must be translated to its mapped data type, removing unused fields should increase performance. |

Raw Field Parsing

The input step automatically attempts to parse the raw field into a number of child fields denoted by `_raw.<Field Name>`. It parses the raw field assuming that the field is formatted with name value pairs separated by a newline character, like this: `<Name1>=<Value1>\n <Name2>=<Value2>\n`. If raw field data is not formatted like this, you must post-process those fields with other steps in the transformation flow. Note that your secondary steps may include String variables.

Date Handling

Kettle does not support the parsing of ISO-8601 date formats, which is Splunk's format for passing date objects through web services. However, you can edit the date string returned from Splunk using the Modified Java Script Value step. Use this script to parse the date.

```
var dateobj = str2date((substr(_time, 0, 23) + "GMT" + substr(_time, 23)).trim(),
  "YYYY-MM-dd'T'HH:mm:ss.SSSz");
```

Splunk Output

The **Splunk Output** transformation step enables you to connect to a Splunk server and write events to a Splunk index. By default, the step writes events as name value pairs separated by newline characters, but can also write arbitrary formats by customizing event data. You must have write access to a Splunk server before you use the **Splunk Output** step. To learn more about Splunk see their [online documentation](#).

| Option | Definition |
|---------------------------------------|--|
| Step name | Name of the step as it appears in the transformation workspace. |
| Host name(s) or IP address(es) | Specifies the network name or address of the Splunk instance or instances. |
| Port | Indicates the port number of the Splunk (splunkd) server. The default value is 8089, but your administrator may have changed the port number. |
| Username | Specifies the username required to access the Splunk server. |
| Password | Indicates the password associated with the Username . |
| Index to write to | Specifies the Splunk index where the events are stored. Usually, this is the main index. Check your Splunk server for a list of available indices. This field can be parameterized with incoming fields (<code>?{<Field>}</code>) or transformation parameters (<code>\${Parameter}</code>). |
| Event host | Indicates the hostname of the original event host. If you want to gather data from a router and write it to Splunk, use the router's host name. This field can be parameterized with incoming fields (<code>?{<Field>}</code>) or transformation parameters (<code>\${Parameter}</code>). |
| Event source type | Indicates the format type of the event data. The list of known source types appears here . To define a new format, follow these instructions . |
| Event source | Indicates the source of the event data. See Splunk documentation for more details. |
| Customize Splunk event | If checked, enables the Splunk Event Data option and allows you to customize the data coming into Splunk. This is useful if you want to write a different format than the |

| Option | Definition |
|--------------------------|---|
| | default, which is name value pairs separated by newline characters. |
| Splunk event data | Allows you to specify customized event text. This field can be parameterized with incoming fields (<code>{<Field>}</code>) or transformation parameters (<code>{Parameter}</code>). |
| | |

SSTable Output

The SSTable Output step writes to a filesystem directory as a Cassandra SSTable.

| Option | Definition |
|---|--|
| Step name | The name of this step as it appears in the transformation workspace. |
| Cassandra yaml file | Location of yaml file. A <code>cassandra.yaml</code> file is the main configuration file for Cassandra and defines node and cluster configuration details. |
| Directory | Location to write the output to. This directory points to the target table to load to and must match the Keyspace field. |
| Keyspace | Name of the keyspace of the target table to load to. This name must match the Directory field. |
| Column family (table) | Name of the table to upload to. This assumes the metadata for this table was previously defined in Cassandra. |
| Incoming field to use as the row key | Allows you to select which incoming row to use as the row key. This drop-down box will be populated with the names of incoming transformation fields. |
| Buffer (MB) | The buffer size to use. A new table file is written every time the buffer is full. |

PDI Big Data Job Entries

This section contains reference documentation for job entries which enable PDI to work with big data technologies.

Please see [Using Pentaho Data Integration](#) for additional transformation step and job entry references.

Amazon EMR Job Executor

This job entry executes Hadoop jobs on an Amazon Elastic MapReduce (EMR) account. In order to use this step, you must have an Amazon Web Services (AWS) account configured for EMR, and a premade Java JAR to control the remote job.

| Option | Definition |
|-------------------------------|---|
| Name | The name of this Amazon EMR Job Executor step instance. |
| EMR Job Flow Name | The name of the Amazon EMR job flow (series of steps) you are executing. |
| AWS Access Key | Your Amazon Web Services access key. |
| AWS Secret Key | Your Amazon Web Services secret key. |
| S3 Staging Directory | The Amazon Simple Storage Service (S3) address of the working directory for this Hadoop job. This directory will contain the MapReduce JAR, and log files will be placed here as they are created. |
| MapReduce JAR | The Java JAR that contains your Hadoop mapper and reducer classes. The job must be configured and submitted using a static main method in any class in the JAR. |
| Command line arguments | Any command line arguments that must be passed to the static main method in the specified JAR. |
| Number of Instances | The number of Amazon Elastic Compute Cloud (EC2) instances you want to assign to this job. |
| Master Instance Type | The Amazon EC2 instance type that will act as the Hadoop "master" in the cluster, which handles MapReduce task distribution. |
| Slave Instance Type | The Amazon EC2 instance type that will act as one or more Hadoop "slaves" in the cluster. Slaves are assigned tasks from the master. This is only valid if the number of instances is greater than 1. |
| Enable Blocking | Forces the job to wait until each step completes before continuing to the next step. This is the only way for PDI to be aware of a Hadoop job's status. If left unchecked, the Hadoop job is blindly executed, and PDI moves on to the next step. Error handling/routing will not work unless this option is checked. |
| Logging Interval | Number of seconds between log messages. |

Amazon Hive Job Executor

This job executes Hive jobs on an Amazon Elastic MapReduce (EMR) account. In order to use this step, you must have an Amazon Web Services (AWS) account configured for EMR, and a premade Java JAR to control the remote job.

| Option | Definition |
|---------------------------------------|---|
| Name | The name of this job as it appears in the transformation workspace. |
| Hive Job Flow Name | The name of the Hive job flow to execute. |
| Existing JobFlow Id (optional) | The name of a Hive Script on an existing EMR job flow. |
| AWS Access Key | Your Amazon Web Services access key. |
| AWS Secret Key | Your Amazon Web Services secret key. |
| Bootstrap Actions | References to scripts to invoke before the node begins processing data. See http://docs.amazonwebservices.com/ElasticMapReduce/latest/DeveloperGuide/Bootstrap.html for more information. |
| S3 Log Directory | The URL of the Amazon S3 bucket in which your job flow logs will be stored. Artifacts required for execution (e.g. Hive Script) will also be stored here before execution. (Optional) |
| Hive Script | The URL of the Hive script to execute within Amazon S3. |
| Command Line Arguments | A list of arguments (space-separated strings) to pass to Hive. |
| Number of Instances | The number of Amazon EC2 instances used to execute the job flow. |
| Master Instance Type | The Amazon EC2 instance type that will act as the Hadoop "master" in the cluster, which handles MapReduce task distribution. |
| Slave Instance Type | The Amazon EC2 instance type that will act as one or more Hadoop "slaves" in the cluster. Slaves are assigned tasks from the master. This is only valid if the number of instances is greater than 1. |
| Keep Job Flow Alive | Specifies whether the job flow should terminate after completing all steps. |

Hadoop Copy Files

This job entry copies files in a Hadoop cluster from one location to another.

General

| Option | Definition |
|----------------------------------|--|
| Include Subfolders | If selected, all subdirectories within the chosen directory will be copied as well |
| Destination is a file | Determines whether the destination is a file or a directory |
| Copy empty folders | If selected, will copy all directories, even if they are empty the Include Subfolders option must be selected for this option to be valid |
| Create destination folder | If selected, will create the specified destination directory if it does not currently exist |
| Replace existing files | If selected, duplicate files in the destination directory will be overwritten |

| Option | Definition |
|--------------------------------------|---|
| Remove source files | If selected, removes the source files after copy (a move procedure) |
| Copy previous results to args | If selected, will use previous step results as your sources and destinations |
| File/folder source | The file or directory to copy from; click Browse and select Hadoop to enter your Hadoop cluster connection details |
| File/folder destination | The file or directory to copy to; click Browse and select Hadoop to enter your Hadoop cluster connection details |
| Wildcard (RegExp) | Defines the files that are copied in regular expression terms (instead of static file names), for instance: <code>.*\.txt</code> would be any file with a <code>.txt</code> extension |
| Files/folders | A list of selected sources and destinations |

Result files name

| Option | Definition |
|---------------------------------------|---|
| Add files to result files name | Any files that are copied will appear as a result from this step; shows a list of files that were copied in this step |

Hadoop Job Executor

This job entry executes Hadoop jobs on a Hadoop node. There are two option modes: **Simple** (the default condition), in which you only pass a premade Java JAR to control the job; and **Advanced**, in which you are able to specify static main method parameters. Most of the options explained below are only available in Advanced mode. The **User Defined** tab in Advanced mode is for Hadoop option name/value pairs that are not defined in the **Job Setup** and **Cluster** tabs.

General

| Option | Definition |
|-------------------------------|---|
| Name | The name of this Hadoop Job Executor step instance. |
| Hadoop Job Name | The name of the Hadoop job you are executing. |
| Jar | The Java JAR that contains your Hadoop mapper and reducer job instructions in a static main method. |
| Command line arguments | Any command line arguments that must be passed to the static main method in the specified JAR. |

Job Setup

| Option | Definition |
|---------------------------|--|
| Output Key Class | The Apache Hadoop class name that represents the output key's data type. |
| Output Value Class | The Apache Hadoop class name that represents the output value's data type. |
| Mapper Class | The Java class that will perform the map operation. Pentaho's default mapper class should be sufficient for most needs. Only change this value if you are supplying your own Java class to handle mapping. |

| Option | Definition |
|-----------------------|--|
| Combiner Class | The Java class that will perform the combine operation. Pentaho's default combiner class should be sufficient for most needs. Only change this value if you are supplying your own Java class to handle combining. |
| Reducer Class | The Java class that will perform the reduce operation. Pentaho's default reducer class should be sufficient for most needs. Only change this value if you are supplying your own Java class to handle reducing. If you do not define a reducer class , then no reduce operation will be performed and the mapper or combiner output will be returned. |
| Input Path | The path to your input file on the Hadoop cluster. |
| Output Path | The path to your output file on the Hadoop cluster. |
| Input Format | The Apache Hadoop class name that represents the input file's data type. |
| Output Format | The Apache Hadoop class name that represents the output file's data type. |

Cluster

| Option | Definition |
|--------------------------------|--|
| HDFS Hostname | Hostname for your Hadoop cluster. |
| HDFS Port | Port number for your Hadoop cluster. |
| Job Tracker Hostname | If you have a separate job tracker node, type in the hostname here. Otherwise use the HDFS hostname. |
| Job Tracker Port | Job tracker port number; this cannot be the same as the HDFS port number. |
| Number of Mapper Tasks | The number of mapper tasks you want to assign to this job. The size of the inputs should determine the number of mapper tasks. Typically there should be between 10-100 maps per node, though you can specify a higher number for mapper tasks that are not CPU-intensive. |
| Number of Reducer Tasks | The number of reducer tasks you want to assign to this job. Lower numbers mean that the reduce operations can launch immediately and start transferring map outputs as the maps finish. The higher the number, the quicker the nodes will finish their first round of reduces and launch a second round. Increasing the number of reduce operations increases the Hadoop framework overhead, but improves load balancing. If this is set to 0 , then no reduce operation is performed, and the output of the mapper will be returned; also, combiner operations will also not be performed. |
| Enable Blocking | Forces the job to wait until each step completes before continuing to the next step. This is the only way for PDI to be aware of a Hadoop job's status. If left unchecked, the Hadoop job is blindly executed, and PDI moves on to the next step. Error handling/routing will not work unless this option is checked. |
| Logging Interval | Number of seconds between log messages. |

Oozie Job Executor

This job entry executes Oozie Workflows. It is a front end on top of the OozieClient Java API that submits jobs to an Oozie server using web service calls.

Oozie is a workflow/coordination system to manage Hadoop jobs. Oozie Workflow jobs are Directed Acyclical Graphs (DAGs) of actions. Oozie Coordinator jobs are recurrent Oozie Workflow jobs and can be configured so a job is triggered by time (frequency) and data availability.

Oozie is integrated with the rest of the Hadoop stack supporting several types of Hadoop jobs out of the box (Java map-reduce, Streaming map-reduce, Pig, Distcp, etc.). To learn more about Oozie and Oozie Workflows, visit Oozie's website: <http://oozie.apache.org/index.html>.

Oozie Job Executor (Quick Setup Mode)

| Option | Definition |
|------------------------------|--|
| Name | The name of this job instance. |
| Oozie URL | Field to enter an Oozie URL. <i>This must be a valid Oozie location.</i> |
| Enable Blocking | Option blocks the rest of a transformation from executing until the Oozie job finishes when checked. |
| Polling Interval (ms) | Field allows you to set the interval rate to check for Oozie workflows. |
| Workflow Properties | Field to enter the Workfile Properties file. This path is required and must be a valid job properties file. In the properties file, the <code>oozie.wf.application.path</code> path must be set. |

Oozie Job Executor (Advanced Setup Mode)

If you have not set the Oozie path within your workflow properties file, you can add the needed path with Advanced Setup Mode within the Oozie Job Executor. To access Advanced Setup Mode, from within the Oozie Job Executor dialog, click **Advanced Options**.

Advanced Setup Mode allows you to add the needed Oozie path to your workflow properties file. It does not add the path directly to the properties file, instead the path is added by the Oozie Job Executor, not directly changing your workflow properties file.

| Option | Definition |
|----------------------------------|---|
| Workflow Properties | Displays the arguments, and their values, that are set within the workflow properties file found at the Oozie URL specified within the <code>Oozie URL</code> field. |
| Add Argument (green plus button) | Allows you to add a workflow property argument. Use this button to add the required Oozie path if it is not already set. This does not add the path to the properties file, instead it adds it to the PDI job, which adds it to the workflow configuration upon execution of the job. |
| Delete Argument (red "x" button) | Allows you to delete an argument. To delete an argument from the Oozie Executor job, select the desired argument from Workflow Properties, then click the Delete Argument button. |

Pentaho MapReduce



Note: This entry was formerly known as **Hadoop Transformation Job Executor**.

This job entry executes transformations as part of a Hadoop MapReduce job. This is frequently used to execute transformations that act as mappers and reducers in lieu of a traditional Hadoop Java class. The **User Defined** tab is for Hadoop option name/value pairs that are not defined in the **Job Setup** and **Cluster** tabs. Any properties defined here will be set in the MapReduce job configuration.

General

| Option | Definition |
|------------------------|--|
| Name | The name of this Hadoop Job Executor step instance |
| Hadoop Job Name | The name of the Hadoop job you are executing |

Mapper

| Option | Definition |
|--------------------------------|--|
| Look in | Sets the context for the Browse button. Options are: Local (the local filesystem), Repository by Name (a PDI database or solution repository), or Repository by Reference (a link to a transformation no matter which repository it is in). |
| Mapper Transformation | The KTR that will perform the mapping functions for this job. |
| Mapper Input Step Name | The name of the step that receives mapping data from Hadoop. This must be a MapReduce Input step. |
| Mapper Output Step Name | The name of the step that passes mapping output back to Hadoop. This must be a MapReduce Output step. |

Combiner

| Option | Definition |
|----------------------------------|---|
| Look in | Sets the context for the Browse button. Options are: Local (the local filesystem), Repository by Name (a PDI database or solution repository), or Repository by Reference (a link to a transformation no matter which repository it is in). |
| Combiner Transformation | The KTR that will perform the combiner functions for this job. |
| Combiner Input Step Name | The name of the step that receives combiner data from Hadoop. This must be a MapReduce Input step. |
| Combiner Output Step Name | The name of the step that passes combiner output back to Hadoop. This must be a MapReduce Output step. |
| Combine single threaded | Indicates if the Single Threaded transformation execution engine should be used to execute the combiner transformation. If false, the normal multi-threaded transformation engine will be used. The Single Threaded transformation execution engine reduces overhead when processing many small groups of output. |

Reducer

| Option | Definition |
|---------------------------------|--|
| Look in | Sets the context for the Browse button. Options are: Local (the local filesystem), Repository by Name (a PDI database or solution repository), or Repository by Reference (a link to a transformation no matter which repository it is in). |
| Reducer Transformation | The KTR that will perform the reducer functions for this job. |
| Reducer Input Step Name | The name of the step that receives reducing data from Hadoop. This must be a MapReduce Input step. |
| Reducer Output Step Name | The name of the step that passes reducing output back to Hadoop. This must be a MapReduce Output step. |
| Reduce single threaded | Indicates if the Single Threaded transformation execution engine should be used to execute the reducer transformation. If false, the normal multi-threaded transformation engine will be used. The Single Threaded transformation execution engine reduces overhead when processing many small groups of output. |

Job Setup

| Option | Definition |
|---|--|
| Suppress Output of Map Key | If selected the key output from the Mapper transformation will be ignored and replaced with NullWritable. |
| Suppress Output of Map Value | If selected the value output from the Mapper transformation will be ignored and replaced with NullWritable. |
| Suppress Output of Reduce Key | If selected the key output from the Combiner and/or Reducer transformations will be ignored and replaced with NullWritable. |
| Suppress Output of Reduce Value | If selected the key output from the Combiner and/or Reducer transformations will be ignored and replaced with NullWritable. |
| Input Path | A comma-separated list of input directories , such as <code>/wordcount/input</code> , from your Hadoop cluster where the source data for the MapReduce job is stored. |
| Output Path | The directory on your Hadoop cluster where you want the output from the MapReduce job to be stored., such as <code>//wordcount/output</code> . The output directory cannot exist prior to running the MapReduce job. |
| Input Format | The Apache Hadoop class name that describes the input specification for the MapReduce job. See InputFormat for more information. |
| Output Format | The Apache Hadoop class name that describes the output specification for the MapReduce job. See OutputFormat for more information. |
| Clean output path before execution | If enabled the output path specified will be removed before the MapReduce job is scheduled. |

Cluster

| Option | Definition |
|--------------------------------|--|
| HDFS Hostname | Hostname for your Hadoop cluster. |
| HDFS Port | Port number for your Hadoop cluster. |
| Job Tracker Hostname | If you have a separate job tracker node, type in the hostname here. Otherwise use the HDFS hostname. |
| Job Tracker Port | Job tracker port number; this cannot be the same as the HDFS port number. |
| Number of Mapper Tasks | The number of mapper tasks you want to assign to this job. The size of the inputs should determine the number of mapper tasks. Typically there should be between 10-100 maps per node, though you can specify a higher number for mapper tasks that are not CPU-intensive. |
| Number of Reducer Tasks | The number of reducer tasks you want to assign to this job. Lower numbers mean that the reduce operations can launch immediately and start transferring map outputs as the maps finish. The higher the number, the quicker the nodes will finish their first round of reduces and launch a second round. Increasing the number of reduce operations increases the Hadoop framework overhead, but improves load balancing. If this is set to 0 , then no reduce operation is performed, and the output of the mapper becomes the output of the entire job; also, combiner operations will also not be performed. |
| Enable Blocking | Forces the job to wait until each step completes before continuing to the next step. This is the only way for PDI to be aware of a Hadoop job's status. If left unchecked, the Hadoop job is blindly executed, and PDI moves on to the next job entry. Error handling/routing will not work unless this option is checked. |
| Logging Interval | Number of seconds between log messages. |

Pig Script Executor

Executes a script written in Apache Pig's "Pig Latin" language on a Hadoop cluster. All log entries pertaining to this script execution that are generated by Apache Pig will show in the PDI log.

General

| Option | Definition |
|-----------------------------|---|
| Job Entry Name | The name of this Pig Script Executor instance. |
| HDFS hostname | The hostname of the machine that operates a Hadoop distributed filesystem. |
| HDFS port | The port number of the machine that operates a Hadoop distributed filesystem. |
| Job tracker hostname | The hostname of the machine that operates a Hadoop job tracker. |
| Job tracker port | The port number of the machine that operates a Hadoop job tracker. |

| Option | Definition |
|------------------------|--|
| Pig script | The path (remote or local) to the Pig Latin script you want to execute. |
| Enable blocking | If checked, the Pig Script Executor job entry will prevent downstream entries from executing until the script has finished processing. |
| Local execution | Executes the script within the same Java virtual machine that PDI is running in. This option is useful for testing and debugging because it does not require access to a Hadoop cluster. When this option is selected, the HDFS and job tracker connection details are not required and their corresponding fields will be disabled. |

Script Parameters

| Option | Definition |
|-----------------------|--|
| # | The order of execution of the script parameters. |
| Parameter name | The name of the parameter you want to use. |
| Value | The value you're substituting whenever the previously defined parameter is used. |

Sqoop Export

The Sqoop Export job allows you to export data from Hadoop into an RDBMS using Apache Sqoop. This job has two setup modes:

- **Quick Mode** provides the minimum options necessary to perform a successful Sqoop export.
- **Advanced Mode**'s default view provides options for to better control your Sqoop export. **Advance Mode** also has a command line view which allows you to reuse an existing Sqoop command from the command line.

For additional information about Apache Sqoop, visit <http://sqoop.apache.org/>.

Quick Setup

| Option | Definition |
|----------------------------|---|
| Name | The name of this job as it appears in the transformation workspace. |
| Namenode Host | Host name or IP address of the Hadoop NameNode. |
| Namenode Port | Port number of the Hadoop NameNode. |
| Jobtracker Host | Host name of the Hadoop JobTracker. |
| Job Tracker Port | Port number of the Hadoop JobTracker |
| Export Directory | Path of the directory within HDFS to export from. |
| Database Connection | Select the database connection to export to. Clicking Edit... allows you to edit an existing connection or you can create a new connection from this dialog by clicking New.... |
| Table | Destination table to export into. If the source database requires it a schema may be supplied in the format: SCHEMA.TABLE_NAME. This table must exist and its structure must match the input data's format. |

Advanced Setup

| Option | Definition |
|-------------------|--|
| Default/List view | List of property and value pair settings which can be modified to suit your needs including options to configure an export from Hive or HBase. |
| Command line view | Field which accepts command line arguments, typically used to allow you to paste an existing Sqoop command line argument. |

Sqoop Import

The Sqoop Import job allows you to import data from a relational database into the Hadoop Distributed File System (HDFS) using Apache Sqoop. This job has two setup modes:

- **Quick Mode** provides the minimum options necessary to perform a successful Sqoop import.
- **Advanced Mode**'s default view provides options for to better control your Sqoop import. **Advanced Mode** also has a command line view which allows you to paste an existing Sqoop command line argument into.

For additional information about Apache Sqoop, visit <http://sqoop.apache.org/>.

Quick Setup

| Option | Definition |
|----------------------------|---|
| Name | The name of this job as it appears in the transformation workspace. |
| Database Connection | Select the database connection to import from. Clicking Edit... allows you to edit an existing connection or you can create a new connection from this dialog by clicking New.... |
| Table | Source table to import from. If the source database requires it a schema may be supplied in the format: SCHEMA.YOUR_TABLE_NAME. |
| Namenode Host | Host name of the target Hadoop NameNode. |
| Namenode Port | Port number of the target Hadoop NameNode. |
| Jobtracker Host | Host name of the target Hadoop JobTracker. |
| Job Tracker Port | Port number of the target Hadoop JobTracker. |
| Target Directory | Path of the directory to import into. |

Advanced Setup

| Option | Definition |
|-------------------|--|
| Default/List view | List of property and value pair settings which can be modified to suit your needs including options to configure an import to Hive or HBase. |
| Command line view | Field which accepts command line arguments, typically used to allow you to paste an existing Sqoop command line argument. |