



The Pentaho Big Data Guide



This document is copyright © 2012 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 have questions that are not covered in this guide, or if you would like to report errors in the documentation, 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 on the topics covered in this guide, visit <http://www.pentaho.com/training>.

Limits of Liability and Disclaimer of Warranty

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.

Company Information

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

E-mail: communityconnection@pentaho.com

Sales Inquiries: sales@pentaho.com

Documentation Suggestions: documentation@pentaho.com

Sign-up for our newsletter: <http://community.pentaho.com/newsletter/>

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
Creating a New Hadoop Configuration.....	6
Configuring MapR Configuration on Windows.....	6
PDI Hadoop Configurations.....	7
Including/Excluding Classes or Packages for a Hadoop Configuration.....	7
Working with Big Data and Hadoop in PDI.....	9
Hadoop Job Process Flow.....	9
Hadoop Transformation Process Flow.....	11
Loading Data Into a Hadoop Cluster.....	12
Using a PDI Job Entry to Load Data into a Hadoop Distributed File System (HDFS).....	12
Using a PDI Transformation Step to Load Data into HBase.....	13
Transforming Data Within a Hadoop Cluster.....	13
Transforming Data Within Hive.....	13
Transforming Data With Pig.....	14
Extracting Data from a Hadoop Cluster.....	14
Extracting Data from HDFS to Load in a Relational Database.....	14
Hadoop to PDI Data Type Conversion.....	14
Hadoop Hive-Specific SQL Limitations.....	15
PDI Big Data Transformation Steps.....	16
Avro Input.....	16
Cassandra Input.....	17
Cassandra Output.....	18
CouchDB Input.....	20
Hadoop File Input.....	20
Hadoop File Output.....	25
HBase Input.....	28
HBase Output.....	30
HBase Row Decoder.....	32
MapReduce Input.....	33
MapReduce Output.....	33
MongoDb Input.....	34
MongoDb Output.....	34
SSTable Output.....	36
PDI Big Data Job Entries.....	38
Amazon EMR Job Executor.....	38
Amazon Hive Job Executor.....	38
Hadoop Copy Files.....	39
Hadoop Job Executor.....	40
Oozie Job Executor.....	42
Pentaho MapReduce.....	43
Pig Script Executor.....	45
Sqoop Export.....	46
Sqoop Import.....	47

Getting Started with PDI and Hadoop

Pentaho's Big Picture of Big Data

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

What Exactly Is Big Data?

The term big data applies to very large, complex, or dynamic datasets which 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, data retrieval, and data querying/process--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 modern 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

What is Hadoop?

[Apache Hadoop](#) is a Java-based, open-source software framework which enables applications to work with potentially thousands of independently distributed hardware systems and petabytes of data.

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.

What is a Hadoop Distributed File System, or HDFS?

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](#), which includes these modules:

- **Hadoop Distributed File System (HDFS)**: A distributed file system that provides high-throughput access to application data.
- [Hadoop MapReduce](#): the key algorithm to distribute work around a cluster.

Other Hadoop-related projects at Apache include:

- [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 ad hoc querying
- [Pig](#): A high-level, data-flow language and execution framework for parallel computation
- [ZooKeeper](#): A high-performance coordination service for distributed applications

Configuring Your Big Data Environment

What This Section Covers

This section covers configuring PDI to communicate with Hadoop distributions other than the default configuration, referred to as Hadoop 0.20.2.

Supported Big Data Technologies

A list of supported big data technology can be found in the *Supported Components* section within the *PDI Installation Guide*.

The Pentaho Data Integration Big Data plugin comes pre-configured for use with Hadoop 0.20.2. There are transformation steps and job entries that enable you to load, transform, and extract data to and from a Hadoop Distributed File System (HDFS), as well as other Hadoop configurations.

Setting the Active Hadoop Configuration

About Hadoop Configurations within PDI

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.

Because the Kettle client comes pre-configured for Apache Hadoop no further configuration is required in order to use PDI with the default version and configuration of Hadoop.

Locating Which Version of Hadoop is in Use

The Hadoop distribution configuration can be found at the following location: `plugins/pentaho-big-data-plugin/hadoop-configurations/configuration/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-XXXXXXXXXX
```

The property `active.hadoop.configuration=hadoop-XXXXXXXXXX`, is representative of the version currently in use, where `XXXXXXXXXX` is the active version of Hadoop. This property configures which 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 provided default Hadoop version.

Creating 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.

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.
2. Copy the `jar` files for your specified Hadoop version.
3. Paste the `jar` files in the `lib/` directory.
4. Change the `active.hadoop.configuration=` property in the `plugins/pentaho-big-data-plugin/hadoop-configurations/configuration/plugin.properties` file to match your specific Hadoop configuration. This property configures which distribution of Hadoop to use when communicating with a Hadoop cluster. Update this property if you are using a version other than provided default Hadoop version.

Configuring MapR Configuration on Windows

To communicate with a MapR cluster the MapR client tools must be installed on the machine.

The MapR configuration properties are defined in the `config.properties` file, found within `C:\opt\mapr\mapr-client-{version}.{architecture}`. These steps explain how to configure your MapR configuration with the locally installed MapR client tools on Windows.

1. Back up your MapR cluster configuration.
2. Open the MapR `config.properties` file for editing.
3. Remove the existing `classpath` and `library.path` properties.
4. From the "Windows-specific..." section, update the properties from `classpath` and `library.path` to match your locally-installed MapR client tools installation directory.

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/hadoop-configurations/` 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> , <code>hadoop-client-*</code>)
pmr/	Jars 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 this shim that are not client-only or special <code>pmr</code> jars 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 define configuration name, additional classpath and native libraries that 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/hadoop-`

`configurations/configuration/plugin.properties`. For additional information, see the developer notes within the `plugin.properties` file.

Including Additional Class Paths or Libraries

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 that expect a single class shared by all class loaders, as with Apache Commons Logging for example.

Working with Big Data and Hadoop in PDI

This section contains guidance and instructions for configuring and using the big data functionality in Pentaho Data Integration.

This encompasses the following functions from within the PDI environment:

- Loading data into a Hadoop cluster
- Transforming data within a Hadoop cluster
- Extracting data from a Hadoop cluster
- Reporting on data within a Hadoop cluster
- Accessing other big data-related technology and databases, such as MongoDB, Cassandra, Hive, HBase, Sqoop, and Oozie using PDI transformations or jobs.

PDI's Big Data Plugin # No Additional Installation Required!

By default PDI is pre-configured to work with Apache Hadoop. But PDI can be configured to communicate with most popular Hadoop configurations by updating the `plugins/pentaho-big-data-plugin/plugin.properties` file to match your active Hadoop configuration. Specific instructions for changing Hadoop configurations is covered in the *PDI Administrators Guide*.

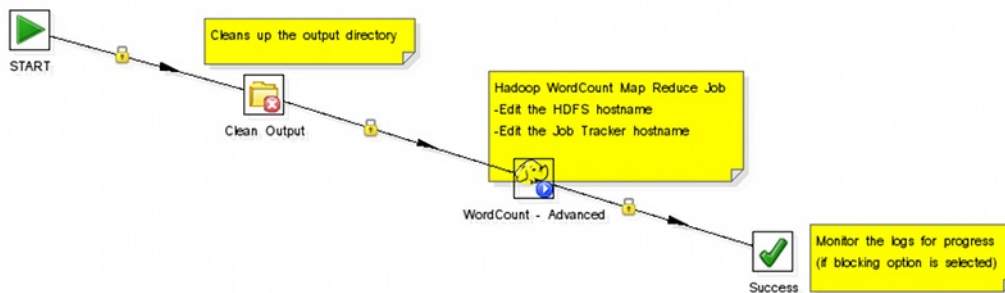
For a list of supported big data technology, including which configurations of Hadoop are currently supported, see the *Compatibility Matrix* in the *PDI Installation Guide*.

Hadoop Job Process Flow

There are two paradigms for jobs in PDI:

- native PDI jobs, which are processes that run transformations or other jobs
- Hadoop jobs, which are executed on the Hadoop node containing the data you are working with

PDI enables you to design and execute Hadoop jobs in a similar manner to native PDI jobs. The relevant step is called **Hadoop Job Executor**.



This step requires a custom mapper/reducer Java class.

The screenshot shows the 'Hadoop Job Executor' dialog box. It has a title bar with a close button. The fields are as follows:

- Name:** WordCount - Simple
- Hadoop Job Name:** PDI Hadoop - WordCount - Simple
- Jar:** ./samples/jobs/hadoop/pentaho-mapreduce-sample.jar (with a 'Browse...' button)
- Configuration:** Simple (selected with a radio button), Advanced (unselected)
- Command line arguments:** /junit/wordcount/input /junit/wordcount/pdioutput

At the bottom right are 'OK' and 'Cancel' buttons.

If you are using the Amazon Elastic MapReduce (EMR) service, you can use a similar Hadoop job step called **Amazon EMR Job Executor**. This differs from the standard Hadoop Job Executor in that it contains connection information for Amazon S3 and configuration options for EMR.

The screenshot shows the 'Amazon EMR Job Executor' dialog box. It has a title bar with a close button. The fields are as follows:

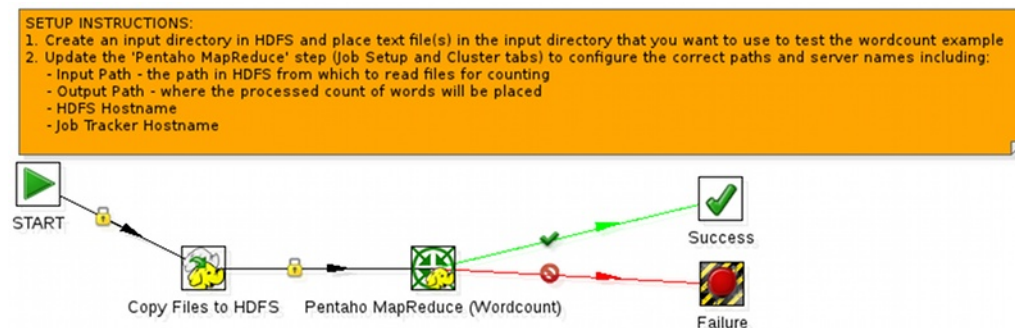
- Name:** Amazon EMR Job Executor
- EMR Job Flow Name:** (empty field)
- Existing JobFlow Id (optional):** (empty field)
- AWS Access Key:** (empty field)
- AWS Secret Key:** (empty field)
- S3 Staging Directory:** (empty field) with a 'Browse...' button
- MapReduce Jar:** (empty field) with a 'Browse...' button
- Command Line Arguments:** (empty field)
- Number of Instances:** 2
- Master Instance Type:** Small [m1.small] (dropdown menu)
- Slave Instance Type:** Small [m1.small] (dropdown menu)
- Enable Blocking:** (unchecked checkbox)
- Logging Interval:** 60

At the bottom right are 'OK' and 'Cancel' buttons.

You can execute a PDI job that includes Hadoop-oriented transformations through the **Pentaho MapReduce**. In addition to ordinary transformation work, you can use this job entry to design mapper/reducer functions within PDI, removing the need to provide a Java class. To do this, create transformations that act as a mapper and a reducer, then reference them properly in the step configuration.

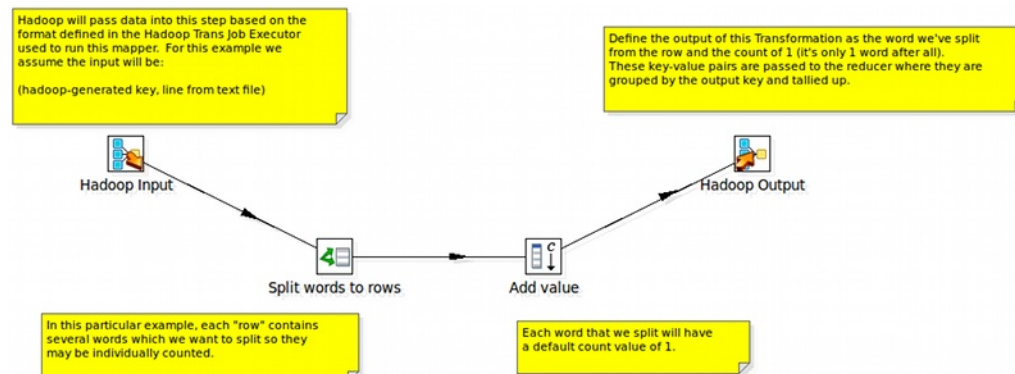
The screenshot shows the 'Pentaho MapReduce' configuration window. It has a title bar with a close button. Inside, there are fields for 'Name' (set to 'Pentaho MapReduce') and 'Hadoop Job Name'. Below these are tabs for 'Mapper', 'Combiner', 'Reducer', 'Job Setup', 'Cluster', and 'User Defined'. The 'Mapper' tab is selected. It contains a 'Look in:' dropdown set to 'Local', a 'Mapper Transformation' field with a 'Browse...' button, and two empty fields for 'Mapper Input Step Name' and 'Mapper Output Step Name'. At the bottom right are 'OK' and 'Cancel' buttons.

The workflow for the transformation job executor looks something like this.

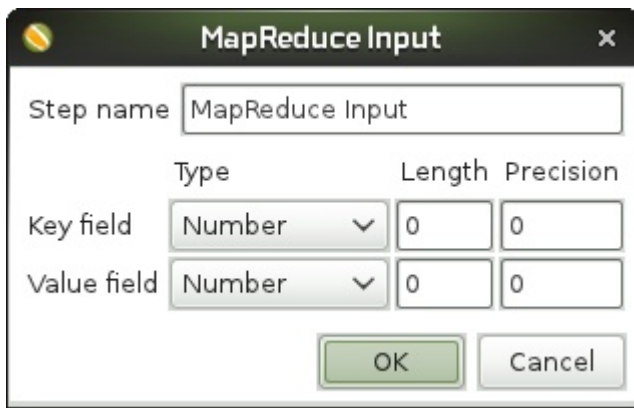


Hadoop Transformation Process Flow

Pentaho Data Integration enables you to pull data from a Hadoop cluster, transform it, and pass it back to the cluster. You can use specially-designed transformations as Hadoop mappers and reducers, which completely removes the need to create a Java class for these purposes. However, you must follow a specific workflow in order to properly communicate with Hadoop, as shown in this sample transformation.



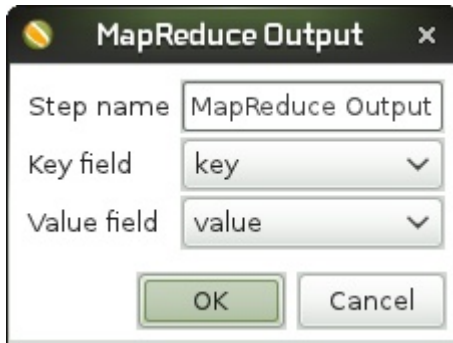
Hadoop communicates in key/value pairs. First, PDI uses a **MapReduce Input** step that defines the data type and name of the key and value.



The **MapReduce Input** dialog box has a title bar with a yellow icon and a close button. It contains a text field for "Step name" with the value "MapReduce Input". Below this is a table with columns "Type", "Length", and "Precision". The "Key field" row has "Number" in the Type column, "0" in the Length column, and "0" in the Precision column. The "Value field" row also has "Number" in the Type column, "0" in the Length column, and "0" in the Precision column. At the bottom are "OK" and "Cancel" buttons.

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

Then, PDI uses a **MapReduce Output** step that passes the output back to Hadoop.



The **MapReduce Output** dialog box has a title bar with a yellow icon and a close button. It contains a text field for "Step name" with the value "MapReduce Output". Below this are two dropdown menus: "Key field" with the value "key" and "Value field" with the value "value". At the bottom are "OK" and "Cancel" buttons.

What happens in the middle is entirely up to you.

Loading Data Into a Hadoop Cluster

These sections contain guidance and instructions on:

- Using a PDI job to load data into a Hadoop Distributed File System (HDFS)
- Using a PDI job to load data into a Hive database
- Using a PDI transformation to source data from a flat file and write the data to an HBase database

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

This section describes how to use a PDI job to move a file into a Hadoop Distributed File System (HDFS); you can use PDI jobs to put files into the HDFS from many different sources. Use these instructions to load data into other Hadoop distributions and Hadoop-related tools, such as Hive, HBase, and MapR, by setting the destination field in the **Hadoop Copy Files** job entry.

1. If it is not already running, start Hadoop and PDI.
2. Select **File > New > Job**.
3. From the Design palette on the left, under the **General** folder, drag the **Start** entry onto the canvas.
4. Under the **Big Data** folder, drag a **Hadoop Copy Files** entry onto the canvas.
5. Connect the two entries by hovering over the **Start** entry and selecting the output connector (green arrow pointing to the right), then drag the connector arrow to the **Hadoop Copy Files** entry.
6. Edit the properties of the **Hadoop Copy Files** entry by double-clicking it. Enter the source and destination information, then click **Add**.

Click **OK** to close the window, then save the job by selecting **File > Save As**.

Run the job by clicking the green **Run** button on the job toolbar, or by choosing **Action > Run** from the menu. Ensure the job finished successfully within the **Execution results** window, as well as checking your destination database to ensure data was loaded.

Using a PDI Transformation Step to Load Data into HBase

This section describes how to use a PDI transformation that sources data from a flat file and writes to an HBase table. These instructions can be used to load data into other Hadoop distributions and Hadoop-related tools, such as Hive, HBase, and MapR, by setting the destination field in the input transformation step. These instructions assume you have already created an existing HBase database table.

1. If not already running, start Hadoop, PDI, and HBase.
2. Select **File > New > Transformation**.
3. From the **Design** palette on the left, under the **Input** folder, drag a step onto the canvas.
4. Edit the input step by double clicking it. Then set the source directory or file from which to extract data. Be sure to select the **Add** button once you have set your source destination. Most input steps have an **Add** button.
5. Switch to the **Content** tab and be sure to configure how your content will be formatted.
6. From the **Fields** tab, select **Get Fields** to list the available fields. From this list you can format each particular field to appear as needed.
7. On the Design palette, under **Big Data**, drag the **HBase Output** to the canvas.
8. Connect your input and HBase Output step by hovering over the input step and selecting the **output connector** (green arrow pointing to the right), then drag the connector arrow to the **HBase Output** step.
9. Edit the HBase Output step by double-clicking it. You must now set your Zookeeper host(s) and port number.
10. Create a HBase mapping by switching to the **Create/Edit mappings** tab and changing the options available as needed.
Be sure to click **Save mapping** when finished.
11. Go to the **Configure connection** tab and designate use of the mapping created in the previous step by selecting **Get mappings for the specified table** and chose the mapping previously created and saved.
Select **OK** to close the dialog, then save the transformation.

Run the transformation by clicking the green **Run** button on the transformation toolbar, or by choosing **Action > Run** from the menu. Ensure the transformation finished successfully within the **Execution results** window.

Transforming Data Within a Hadoop Cluster

These sections contain guidance and instructions on:

- Reading data from a Hive table, transforming it, and writing it to a Hive table within the workflow of a PDI job
- Invoking a Pig script from a PDI job

Transforming Data Within Hive

This section describes how to read data from a Hive database, transform it, and write it to a Hive database table within the workflow of a PDI job. These instructions can be used to read, transform, and write data into other Hadoop distributions and Hadoop-related tools by using the appropriate job entries within PDI. These instructions assume you have already created an existing Hive database table.

1. If not already running, start Hadoop, PDI, and your Hive server.
2. Select **File > New > Job**.
3. From the **View** palette on the left, right-click on **Database connections** and select **New**.
4. In the **Database Connection** dialog, name your connection and set the **Connection Type** as **Hadoop Hive**. Set the appropriate **Host Name** and other connection settings require to access your Hive database.
Click **Test** to ensure your configuration is correct. Once the test is successful, click **OK** to close the **Database Connection** dialog.
5. From the **Design** palette on the left, under the **General** folder, drag the **Start** entry onto the canvas.
6. Under the **Scripting** folder, drag a **SQL** entry onto the canvas. This will allow you to run a HiveQL script.
7. Connect the two entries by hovering over the **Start** entry and selecting the **output connector** (green arrow pointing to the right), then drag the connector arrow to the **SQL** entry.
8. Edit the properties of the **SQL** entry by double-clicking it. For **Connection**, select **Hive**, and enter the SQL script you want to execute.
Click **OK** to close the **Execute SQL Script** dialog.

Run the job by clicking the green **Run** button on the job toolbar, or by choosing **Action > Run** from the menu. Ensure the job finished successfully within the **Execution results** window.

Transforming Data With Pig

This section describes how to invoke a Pig script from a PDI job.

1. If not already running, start Hadoop, Pig, and PDI.
2. Select **File > New > Job**.
3. From the **Design** palette on the left, under the **General** folder, drag the **Start** entry onto the canvas.
4. Under the **Big Data** folder, drag a **Pig Script Executor** entry onto the canvas.
5. Connect the two entries by hovering over the **Start** entry and selecting the **output connector** (green arrow pointing to the right), then drag the connector arrow to the **Pig Script Executor** entry.
6. Edit the properties of the **Pig Script Executor** entry by double-clicking it. Enter the appropriate source and destination information, then click **Add**.
Click **OK** to close the window, then save the job by selecting **File > Save As**.

Run the job by clicking the green **Run** button on the job toolbar, or by choosing **Action > Run** from the menu. Ensure the job finished successfully within the **Execution results** window.

Extracting Data from a Hadoop Cluster

This section contains guidance and instructions on extracting data from Hadoop using HDFS, Hive, and HBase.

Extracting Data from HDFS to Load in a Relational Database

This section describes how to use a PDI transformation to extract data from HDFS and load it into a relational database table, such as MySQL. You can use this tool to extract files from a HDFS or many other sources. These instructions can be contextually used to extract data from other Hadoop distributions and Hadoop-related tools by appropriately setting the source fields in the transformation step configuration dialog. These instructions assume you already have a relational database with information to extract.

1. If not already running, start Hadoop and PDI.
2. Select **File > New > Database Connection**. Set the appropriate connection information as it relates to your specific relational database.
Click **Test** to test your connection.
3. Once your database connection details have been tested and verified, select **File > New > Transformation**.
4. Under the **Big Data** folder, drag a **Hadoop File Input** entry onto the canvas.
Double-click the step to bring up the **Hadoop File Input** dialog.
5. Edit the properties of the **Hadoop File Input** to match your directory configuration.
6. Configure how you would like to format your data within the **Content** and **Fields** tabs.
7. Next, drag a **Table Output** onto the canvas.
8. Connect the two steps by hovering over the **Hadoop File Input** step and selecting the **output connector** (green arrow pointing to the right), then drag the connector arrow to the **Table Output** step.
9. Configure the **Table Output** step as necessary.

Run the transformation by clicking the green **Run** button on the transformation toolbar, or by choosing **Action > Run** from the menu. Ensure the transformation finished successfully within the **Execution results** window, as well as checking your destination database to ensure data was loaded.

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 Apache 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
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 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.

PDI Big Data Transformation Steps

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

Please see the *PDI Users Guide* 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 <code>null</code> 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

Options

Option	Definition
Step name	The name of this step as it appears in the transformation workspace.
Host name or IP address	The location of the MongoDB server. Check mongodb.log to see which IP address was used to run.
Port	The TCP/IP port to connect to, the default is 27017 (also check mongodb.log for the actual value).
Database	The name of the database to retrieve data from. Use the "show dbs" command when connected using the "mongodb" utility to list all databases on the server.
Collection	The name of the collection to retrieve data from. Use the "show collections" command when connected using the "mongodb" utility to list all collections in the database.
Name of the JSON output field	The name of the field that will contain the JSON output from the server. You can parse this JSON then using the "JSON Input" step, <code>eval("{\""+jsonString+"\"")</code> , in JavaScript or using a User Defined Java Class step .
Query expression (JSON)	The query expression in JSON that will limit the output (see examples below).
Authentication user	The user to use for the connection with MongoDB.
Authentication password	The password to use.

Query Examples

The [Advanced Queries](#) page in the MongoDB wiki space details how to use queries. What is not mentioned is that in order for us to pass these queries to MongoDB using the Java API (on which PDI is built) we need to add appropriate quoting. Below are some translated examples:

Query expression	Description
<code>{ 'name' : "MongoDB" }</code>	Query all values where the name field equals to "MongoDB".
<code>{ 'name' : { '\$regex' : 'm.*', '\$options' : "i" } }</code>	Uses a regular expression to find names 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.

MongoDb Output

MongoDb Output writes to a MongoDB collection.

Configure connection

Option	Definition
Step name	The name of this step as it appears in the transformation workspace.
Host name or IP address	The network name or address of the MongoDB instance
Port	Port number of the MongoDB instance
Username	If the database requires authentication, this is the user credential
Password	The password for the given username
Database	The name of the database to use. If you entered the correct connection and authentication information, you can click Get DBs to show a list of databases.
Collection	The collection you want to write to in the specified database. If you entered the correct connection and authentication information, you can click Get collections to show a list of collections. By default, data is inserted into the target collection. If the specified collection doesn't exist, it will be created before data is inserted.
Batch insert size	Mongo DB allows for fast bulk insert operations. This option sets the batch size. If left blank, the default size will be 100 rows.
Truncate collection	Deletes any existing data in the target collection before inserting begins. MongoDB will allow duplicate records to be inserted unless you are using unique indexes.
Upsert	Changes the write mode from insert to upsert, which updates if a match is found, and inserts a new record if there is no match.
Multi-update	Updates all matching documents, rather than just the first.
Modifier update	Enables modifier (\$) operators to be used to mutate individual fields within matching documents. This type of update is fast and involves minimal network traffic.

Mongo document fields

Option	Definition
#	The order of this field in the list
Name	The name of this field, descriptive of its content
Mongo document path	The hierarchical path to each field
Use field name	Specifies whether the incoming field name will be used as the final entry in the path. When this is set to Y for a field, a preceding . (dot) is assumed.
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 "Y" in this column is replaced with the new document constructed with incoming values for all of the defined field paths. If no matching document is found, then a new document is inserted into the collection.

Option	Definition
Modifier operation	In-place modifications of existing document fields. These are much faster than replacing a document with a new one. It is also possible to update more than one matching document in this way. Selecting the Modifier update checkbox in conjunction with Upsert enables this mode of updating. Selecting the Multi-update checkbox as well enables each update to apply to all matching documents (rather than just the first). Valid modifier operations are: \$set , \$inc , and \$push . It also supports the positional operator (\$) for matching inside of arrays.
Get fields	Populates the left-hand column of the table with the names of the incoming fields
Preview document structure	Shows the structure that will be written to MongoDB in JSON format

Create/drop indexes

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. Compound indexes are specified by a comma-separated list of paths.
Index opp	Specifies whether this index will be created or dropped
Unique	Makes the index unique
Sparse	Makes the index sparse
Show indexes	Shows the index information available

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.

Option	Definition
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 the *PDI Users Guide* 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>*\.</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://incubator.apache.org/oozie/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 enterprise 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 enterprise 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 enterprise 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 from your Hadoop cluster that will be used when using a file-based input format derived from FileInputFormat .
Output Path	The directory output from the MapReduce should be written to when using a file-based output format derived from FileOutputFormat .
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
Hadoop Distribution	The Hadoop Distribution to connect to. Only Cloudera has a different option; all others are generic .
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.

Option	Definition
Job tracker port	The port number of the machine that operates a Hadoop job tracker.
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:

Option	Definition
	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. **Advance 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.

Option	Definition
Command line view	Field which accepts command line arguments, typically used to allow you to paste an existing Sqoop command line argument.