

# Pentaho Data Integration (PDI) Techniques: Guidelines for Metadata Injection

# HITACHI Inspire the Next

Change log (if you want to use it):

Date	Version	Author	Changes
10/20/2017	1.0	Megan Brown	Combined two metadata injection documents
2/27/2020	1.1	Mark Monroe Megan Brown	Edited for 9.0

# Contents

Overview	1
Before You Begin	1
Terms You Should Know	1
Other Prerequisites	1
Use Cases	1
Metadata Injection	2
Pentaho Data Integration (PDI) Steps for Metadata Injection	3
Recommendations for Metadata Injection	4
Recommendations for Building Metadata Solutions	5
Standard Metadata Injection	5
Developing the Application	6
Use the Template to Create a Transformation	8
Troubleshooting Metadata Injection Solutions	10
Full Metadata Injection	11
Complex Metadata Injection	12
Data Lake	13
Use Case 1: Sourcing Files into Data Lake, Data Warehouse, Reporting ODS	14
Use Case 2: Search Field for Patterns, Evaluate, and Assign a Weight for Processing	15
Step 1: Building the Sample Source File	15
Step 2: Building the ETL Template Transformation	16
Adding the CSV Input Step	18
Adding the String Operations Step	18
Adding the Replace in String Step	19
Adding the Filter Rows Step	19
Adding the Select Values Step	20
Adding the Text File Output Step	20
Step 3: Building the ETL Building Transformation	21
Adding the Text File Input Step	21
Adding the Get Variables Step	23
Adding the Filter Rows Step	24
Adding the Add Constants Step	24

Adding the ETL Metadata Injection Step	24
Step 4: Building the Job	
Step 5: Renaming the First Transformation	27
Step 6: Renaming the Second Transformation	
Step 7: Running Job and Validating Output	
Related Information	

# Overview

This document covers some best practices for using template-driven designs, and navigating and operating levels of metadata injection. It contains an example of how to build the data-driven rule Extract/Transform/Load (ETL) transformation and make it flexible, so that it can be added to, changed, or removed without adding development cycles.

The intention of this document is to speak about topics generally; however, these are the specific versions covered here:

Software	Version(s)
Pentaho	6.1.x, 7.x, 8.x, 9.0

The <u>Components Reference</u> in Pentaho Documentation has a complete list of supported software and hardware.

# Before You Begin

Before beginning, use the following information to prepare for the procedures described in the main section of the document.

#### Terms You Should Know

Here are some terms you should be familiar with:

- **Metadata**: The collection of field names, datatypes, length, and precision, typically required for the data source and target within a transformation.
- **Static ETL**: ETL with parameters that do not change or that change infrequently, with minor alterations that can be handled manually
- **Dynamic ETL**: ETL dealing with data from many sources, or with dissimilar structures and frequent changes

#### **Other Prerequisites**

This document assumes that you have knowledge about Pentaho and Java JDK and that you have already installed software Pentaho server and configured your environment. More information about related topics outside of this document can be found at <u>ETL Metadata Injection</u>.

#### Use Cases

These use cases can be found later in the document:

- Use Case 1: Sourcing Files into Data Lake, Data Warehouse, Reporting ODS
- Use Case 2: Search Field for Patterns, Evaluate, and Assign a Weight for Processing

# **Metadata Injection**

Metadata is traditionally defined and configured at design time, in a process known as hard-coding, because it does not change at run time. This **static ETL** approach is a good one to take when you are onboarding just one or two data sources where you can easily enter metadata manually for your transformation.

However, this hard-coding approach presents some complications, including:

- Time consumption
- Repetitive manual tasks
- Error-prone solutions
- High labor costs of designing, developing, and supporting a fragile solution
- Added risk when predictable outcomes are jeopardized.

**Metadata injection** is the dynamic ETL alternative to scaling robust applications in an agile environment. One transformation can service many needs by building a framework that shifts time and resources to runtime decisions. This operation dramatically reduces upfront time-to-value and flattens the ongoing investment in maintenance.

When you are dealing with many data sources that have varying schemas, try metadata injection to drastically reduce your development time and accelerate your time to value.



Figure 1: Comparing Static ETL with Metadata Injection for Data Onboarding

Data integration is the main domain of metadata injection. As illustrated in Figure 1, metadata injection is useful in a case with one or more of the following challenges:

- Many data sources
- Different naming conventions
- Similar content
- Dissimilar structure
- Common destination

Metadata injection takes a detour at runtime to gather the metadata and inject it into another transformation.



Figure 2: Metadata Injection Solution Architecture

## Pentaho Data Integration (PDI) Steps for Metadata Injection

The **ETL metadata injection** step can be used in transformations to inject metadata into another transformation, normally with input and output steps for standardizing filenames, naming or renaming fields, removing fields, and adding fields.



Pentaho's metadata injection helps you accelerate productivity and reduce risk in complex data onboarding projects by dynamically scaling out from one template to many transformations.

PDI now has over 75 steps that can be templated to inject metadata or characteristics that can make small or large value changes, allowing each run to be different from the previous.

More information is available at:

- <u>Pentaho Metadata Injection: Accelerating Complex Data Onboarding Processes</u>
- (VIDEO) Pentaho Metadata Injection: Dynamic and Scalable Data Onboarding
- ETL Metadata Injection in Pentaho Documentation.

Step Name	Category	Step Name	Category
Add XML	Transform	JSON input	Input
Annotate stream	Flow	MapReduce input	Big Data
Append streams	Flow	MapReduce output	Big Data
Avro input	Big Data	Memory group by	Statistics

#### Table 1: Example Metadata Injection Steps

Step Name	Category	Step Name	Category
Combination lookup/update	Data Warehouse	Merge join	Joins
CouchDB input	Big Data	Merge rows (diff)	Joins
Data validator	Validation	Multiway merge join	Joins
Elasticsearch bulk insert	Bulk loading	MySQL bulk loader	Bulk loading
ETL metadata injection	Flow	Null if	Utility
Get table names	Input	Oracle bulk loader	Bulk loading
Get variables	Job	<b>Replace in string</b>	Transform
Greenplum load	Bulk loading	Shared dimension	Flow
Hadoop file input	Big Data	Sorted merge	Joins
Hadoop file output	Big Data	Switch/case	Flow
HBase input	Big Data	Synchronize after merge	Output
HBase output	Big Data	Vertica bulk loader	Bulk loading
HBase row decoder	Big Data	XML join	Joins
If field value is null	Utility		·

## **Recommendations for Metadata Injection**

ETL integration development takes time for gathering requirements, building, testing, documenting, deploying, and monitoring production. Rules, requirements, and data itself may change, over time. If that happens, the current rules may no longer apply or new rules may need to be added to the existing transformation to continue working.



We recommend using flexible, data-driven ETL patterns to make your data integration transformation powerful and adaptable to changing business rules without going through a development cycle.

Data integration can be made more flexible and reactive by building rules that can be injected into the transformation before running, and by using the appropriate parameters to pass into ETL jobs. For example:

- Passing in different filenames (paths and filenames can be different for each run)
- Passing different values into a custom database structured query language (SQL) statement to allow for different behaviors (from different tables' names, and where clause field name values)

# Recommendations for Building Metadata Solutions

The use of metadata injection is not limited to PDI, but also extends to business analytics (BA).

You can find details on these topics in the following sections:

- Standard Metadata Injection
- Full Metadata Injection
- <u>Complex Metadata Injection</u>

#### Standard Metadata Injection

We expect to have a transformation injecting metadata into another transformation. That second transformation normally accepts the metadata using an input and an output step.



We recommend you define a template transformation for reuse. The template normally has an input step and an output step. The descriptive grids such as field names and types are intentionally left blank.



We further recommend you define a transformation to inject the metadata into the template using **Flow**  $\rightarrow$  **ETL metadata injection**.

We will show the transformation using the Metadata Injection step, for demonstration purposes. Figure 3 shows the steps you might use:



Figure 3: Standard Metadata Injection

#### Developing the Application

A good way to learn how metadata injection works is to develop a simple application. The following steps will guide you through creating a simple application for metadata injection:



- 1. Create a new transformation and name it MDI Example 1 Standard template.
- From the **Design** tab, drag the **Input** → **Data Grid** step to the workspace and name it Test data – Input.
- 3. Configure the Meta and Data tabs for the step as follows, then click OK:

R, A	dd consta	nt rows		Step name						2 <b>7</b> 13	×
Meta	Data			Step name [lest data - ]	nput						
# 1 2	Name i s	Type Integer String	Format		Length	Precision	Currency	Decimal	Group	Set empty string? N N	
0	Help		1	C	)K Pr	eview (	Cancel				
鸣 A	dd constar	nt rows								(#	×
Meta	Data			Step name Test data - I	nput						
#	i s										
1	1 a										
2	2 b 3 c										
0	Help			o	K Pr	eview C	Cancel				
Та	b		#	Columns	and Pa	arame	ters				
М		h	1	Name: i - T	<b>ype:</b> li	nteger ·	Set er	npty s	tring:	Ν	
ा मस्म Tab Meta tab	2	Name: s - 1	Гуре: S	String -	Set em	npty st	ring:	N			
			1	<b>i:</b> 1 - <b>s:</b> a							
Da	eta Data   # Name   1 i   1 integer   2 s   String     Help     tal   a   2   b   a   2   b   a   2   b   a   c     Tab     Data tab	2	<b>i:</b> 2 - <b>s:</b> b								
			3	<b>i:</b> 3 - <b>s:</b> c							

4. Drag the **Transform** → **Select values** step to the design surface and connect the two steps with a hop, but do not configure it.



This is the foundation of the metadata injection approach to computing as defined by Pentaho.

5. Drag the **Output** → **Text file output** step to the design surface, connect it to the **Select values** step, and configure the **File** tab as shown:

step name	Text file output		
e Content Fields			
Filename	\${Internal.Entry.Current.Directory}/\${Internal.Transformation.Name}_output	•	Browse.
Run this as command instead?			
Pass output to servlet			
Create Parent folder	2		
Do not create file at start			
Accept file name from field?			
File name field			V
Extension	txt		
Include stepnr in filename?			
Include partition nr in filename?			
Include date in filename?			
Include time in filename?			
Specify Date time format			
Date time format			
	Show filename(s)		
Add filenames to result			

#### 6. Configure the **Content** tab as shown:

Step name	Text file output	
ile Content Fields		
Append		
Separator		Insert
Enclosure	#	
Force the enclosure around fields?		
Disable the enclosure fix?		
Header		
Footer		
Format	CR+LF terminated (Windows, DOS)	
Compression	None	
Encoding		
Right pad fields		
Fast data dump (no formatting)		
Split every rows	0	
Add Ending line of file		
-		

7. Do not configure the **Fields** tab, and save the template transformation.

#### Use the Template to Create a Transformation

Next, use your template to create a transformation with these steps:

- 1. Create a new transformation called MDI Example 1 Standard.
- Drag Input-→ Data grid and Flow → ETL metadata injection steps to the design surface and connect them.
- 3. Rename the Add Constant Row to Test Metadata.
- 4. Next, open the **Test Metadata** step and configure the **Meta** tab:

		Step name	t metadata						
leta Data									
# Name 1 fieldname old	Type String	Format		Length	Precision	Currency	Decimal	Group	Set empty string? N
2 fieldname_ne	v String								N

#### 5. Configure the **Data** tab:

		tant rows - C X Step name Test metadata						
Met	a Data							
#	fieldname_old	fieldname_new						
1	i	i_new						
2	s	s_new						

- 6. Open the **ETL metadata injection** step and configure the **File** tab.
- 7. Select Use a File for the transformation template and enter:
   \${Internal.Entry.Current.Directory}/\${Internal.Transformation.Name}\_
   template.ktr.
- 8. There is no need to configure the **Options** tab.
- 9. Save and reopen the transformation so that variable substitution can be performed correctly.

10. Select the **Inject Metadata** tab and configure it, including the **Target injection step key**, **Target description**, **Source step**, and **Source field** for the target transformation:

e re metadata injection (execute tra	maronnualony				
	Step name ETL Metadata	Injection			
Options Inject Metadata					
arget injection step, key	Target description	Source step	Source field		
<ul> <li>MDI_Example_1_Standard_templa</li> </ul>					
<ul> <li>Select values</li> </ul>					
SELECT_UNSPECIFIED	Include unspecified fields, ordered				
✓ FIELDS	Selected fields				
FIELD_NAME	Fieldname				
FIELD_RENAME	Rename to				
FIELD_LENGTH	Length				
FIELD_PRECISION	Precision				
✓ REMOVES	Removed fields				
REMOVE_NAME	Fieldname				
<ul> <li>METAS</li> </ul>	List of fields to change metadata for				
META_NAME	Fieldname	Test metadata	fieldname_old		
META_RENAME	Rename to	Test metadata	fieldname_new		
META_LENGTH	Length				
META_PRECISION	Precision				
META_CONVERSION_N	Format				
META_DATE_FORMAT	Date Format Lenient?				
META_DATE_FORMAT	Date Locale				
META_DATE_FORMAT_	Date Time Zone				
META_LENIENT_STRIN	Lenient number conversion?				
META_DECIMAL	Decimal				
META_GROUPING	Grouping				
META_CURRENCY	Currency				
META_ENCODING	Encoding				
META_TTPE	Pinanuta Narmal?				
V Test data - Input	binary to Norman				
V FIFLDS	All the fields				
NAME	Field name				
TYPE	Field data type				
FORMAT	Field conversion format				
CURRENCY	Field currency symbol				
DECIMAL	Field decimal symbol				
GROUP	Field group symbol				
LENGTH	Field length				
PRECISION	Field precision				
EMPTY_STRING	Set field to empty string?				
DATA_LINES	Nr Rows x Nr Columns values				
DATA_VALUE	One value				
<ul> <li>Text file output</li> </ul>					
FILENAME	The filename				
RUN_AS_COMMAND	Enable this option to treat the filen				
PASS_TO_SERVLET	Enable this option to write data on				
CREATE_PARENT_FOLDER	Create parent folder? (Y/N)				
EXTENSION	The file extension				
SEPARATOR	The separator				
ENCLOSURE	The enclosure				
FORCE ENCLOSURE	Force the enclosure around fields?				
	100	1000			

Your solution should render a text file called MDI\_Example\_1\_Standard\_template\_output.txt containing columns with **new** in the name.

#### Troubleshooting Metadata Injection Solutions

Developing metadata injection solutions is a powerful design pattern, but can make debugging more difficult due to the dynamic nature of this approach.

One way to solve this is by opening the **ETL metadata injection** step, and on the **Options** tab, specifying an **Optional target file (ktr after injection)**:

Step name	ETLI	Metadata Injecti	ion			
Options Inject Metadata						
Template step to read from (optional)						
Expected source step output	# 1	Field name	Туре	Length	Precision	
Optional target file (ktr after injection)	S{In	iternal.Entry.Cu	rrent.Directory}/9	Internal.Transform	mation.Name}_templa	te_afterMDI.ktr
Don't execute resulting transformation						
Streaming source step						
Streaming target step						

Figure 4: Options Tab in Metadata Injection Step

This will save the transformation at runtime, so we can inspect the target of the metadata injection on the **Meta-data** tab. Here we'll find the values of the injected metadata:

				Step name	Select values					
elec	t & Alter Rem	ove Meta-data								
ield	s to alter the m	neta-data for :								
#	Fieldname	Rename to	Туре	Length	Precision	Binary to Normal?	Format	Date Format Lenient?	Date Locale	
1	i	i_new	None			N		N		
2	s	s_new	None			N		N		Get fields to chanc
						1				
<									>	

Figure 5: Meta-data Tab

This debugging approach can aid in problem detection, isolation and correction.

## Full Metadata Injection

This compute pattern should be used when you need to inject metadata at runtime for the filename, rename field names, or remove a field name.



We recommend you use asynchronous steps to inject the metadata. We also recommend that you use variables to make the injection process dynamic at runtime.

The metadata injection transformation might look like this:



Figure 6: Full Metadata Injection

This pattern can be examined or built upon from in our samples  $\rightarrow$  transformations  $\rightarrow$  meta-inject folder supplied with our Client Tools.

## **Complex Metadata Injection**

This type of metadata injection offers a flexible, scalable, and repeatable process to onboard many data sources. Some of these sources present different formats or unknown field lists that need to be ingested regularly.

For example, you might have a requirement to load transaction data values from a supplier's spreadsheet, filter out specific values to examine, and output them to a text file. You can expand this repetitive transformation with a template using metadata injection to load data values from multiple suppliers' spreadsheets in various folders, filter out common, specific transaction values to examine, and output all of it to a single source text file. This compute pattern is documented in Pentaho Documentation: <u>ETL Metadata Injection.</u>



The ETL metadata injection transformation may look like the following:

Figure 7: Complex Metadata Injection



*We recommend you focus on a subset of data values common to all your input files. Develop three components to the solution:* 

- **Template Transformation:** The main repetitive transformation for processing the data per data source. This normally contains an input and output step.
- **Metadata Injection Transformation:** The transformation defining the structure of the metadata and how it is injected into the main transformation.
- **Transformation for All Data Sources:** The transformation going through all the data sources, calling the metadata injection transformation per data source and logging the entire process for possible troubleshooting, if needed.

#### Data Lake

The above example can be extended to provide a dynamic ETL data integration compute pattern for your big data projects. A blueprint for this can be found in <u>Data Lake Operations</u>.

We recommend you keep all Hadoop activities in the cluster as much as possible. This includes input, process, and output. We also recommend that you avoid RDBMS connections in Hadoop jobs and transformations.

Modify the transformation described in <u>Transformation for all input sources</u> in the <u>ETL Metadata</u> <u>Injection</u> documentation with the target Hadoop by replacing the **Text file output** step with the **Hadoop file output** step. The process\_all\_suppliers.ktr might then look like the following:



Figure 8: Example of a Process All Suppliers Transformation

# Use Case 1: Sourcing Files into Data Lake, Data Warehouse, Reporting ODS

This section provides a sample use case and example of how to build flexible ETL data integration jobs that source some of their rules and patterns from outside the job and inject them before each run.

Suppose you have a simple transformation to load transaction data values from a supplier, filterspecific values, and output them to a file. You would need to run this simple transformation for each supplier if you have more than one. Yet, with metadata injection, you can expand this simple repetitive transformation by inserting metadata from another transformation that contains the **ETL metadata injection** step. This step coordinates the data values from the various inputs through the metadata you define. This process reduces the need for you to adjust and run the repetitive transformation for each specific input.



Figure 9: Using the ETL Metadata Injection Step

# Use Case 2: Search Field for Patterns, Evaluate, and Assign a Weight for Processing

This section shows you what you will need for successful searching, evaluation, and processing weight assignment.

Suppose you have over 35 command security feeds/sources, and you want to search patterns within one field and give a weighted score if the pattern is found, without building 35+ different hard-coded ETL jobs. You also want to quickly add, change, and delete patterns or weighted scores as necessary. Your data is audit fields across companies' websites, lightweight directory access protocol (LDAP) command requests, and production server command line terminal sessions (both Linux and Windows).

It is best to set up only what is needed in the rules, leaving most things blank, and including only those things you know will not change.

In this example, we will store the rules in a local text file, with the first row containing column headings, delimited by the pipe symbol:

## Step 1: Building the Sample Source File

The first thing that you will need to do is create the sample source file using these steps:

- 1. This example will be a source text file. Name it C:\opts\etl\cmd src in.txt.
- 2. Use this template of sample data to create the source file:



```
powershellwibblefile
powershell.exe File
C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe -command
NewDriversAutoMap.ps1
cmd -filename
date
time
word.exe
notepad
java --classpath=:c:\bin Writer
```

3. Build a source for data rules to measure security threats (eight fields in total):

#	Field Name	Field Type	Description
1	Кеу	String	Group name, used to pull different groupings of rules for different runs (for example: all rules for windows, linux, ruleset01, ruleset02, emailspamrules)
2	Field	String	Field in transformation to perform search on, replace with weighted value
3	UseRegExYN	Boolean	Y/N value for step behavior
4	SearchRegEx	String	RegEx search pattern to be used
5	ReplaceWithValue	Integer	Value to be stored if SearchRegEx finds a match (this value will be filtered and used further downstream for appropriate action)
6	SetEmptyYN	Boolean	Y/N value for step behavior
7	WholeWordYN	Boolean	Y/N value for step behavior
8	CaseSensitiveYN	Boolean	Y/N value for step behavior

4. Use this input for the dynamic rules to be pulled in and built into the output:

Key|Field|UseRegExYN|SearchRegEx|ReplaceWithValue|SetEmptyYN|WholeWordYN|Ca seSensitiveYN

```
windows|threat|Y|.*powershell\.exe.*File.*|10|N|Y|N
windows|threat|Y|.*powershell\.exe.*IEX.*|10|N|Y|N
windows|threat|Y|.*net user /add.*|20|N|Y|N
linux|threat|Y|.*sudo su -.*|05|N|Y|N
linux|threat|Y|.*rm -f -R.*|25|N|Y|N
linux|threat|Y|.*rm -R -f.*|25|N|Y|N
```

## Step 2: Building the ETL Template Transformation

After you create the sample source file, you'll need to build a template for ETL transformations. This transformation will lay out steps for data processing rules:



1. Create the transformation called etl\_template\_search\_weighted\_.ktr:

💥 Transformation properties	
Transformation Parameters Logging Dates Dependencies	Miscellaneous Monitoring
Transformation name :	etl_template_search_weighted_
Transformation filename	C:\opts\etl\etl_template_search_weightedktr
Description :	
Extended description:	

2. Set up the parameters for the input/output text files (this could easily be changed to be RDBMS Source/Target locations):

Trans	formation Parame	eters Logging Dates Depender
Para	meters :	
*#	Parameter	Default Value
1	rule_file_nm	c:\opts\etl\in_rules.txt
2	rule_key_value	windows
3	src_file_nm	c:\opts\etl\cmd_src_in.txt
4	trg_file_nm	c:\opts\etl\weight_output.txt

3. Next, you will set up the steps that make up your transformation template.

### Adding the CSV Input Step

Use these parameters to fill out the **CSV Input** step:

Ð,	CSV Input									0.8		x נ
			Step na	ime Sou	rceData							
			Filen	ame \${sr	c_file_nm}						0	Browse.
			Delin	niter 🔶							•	Insert TAE
			Enclo	sure "							1999	•
			NIO buffer	size 500	00							
			l azy convers	ion? 🗹								
			Header row pres	ent? 🔽								
		A	dd filename to re	esult 🗌								
	The row i	number f	ield name (optio	nal)								•
			Running in para	illel? 🗌								
		New li	ine possible in fie	elds? 🗌								
			File encod	ling								~ <
* #	Name command	Type String	1	Format	Length 200	Precision	Curre	ency	Decimal	Group	Trim both	type
0	) Help			OK	Get Fields	Preview	N	Canc	el			
St	ep name	:	Filenam	e				Par	ameter	s		
Sc	ourceDat	a	\$(src_f	ile_	nm) 🗲 k	tr		Nar	ne: con	nmand		
			paramet	er				Тур	e: Stri	Ing		
								Len	<b>gth:</b> 20	0		
								Тур	- e: both	l		

#### Adding the String Operations Step

\_

Use these parameters to set up a **String operations** step:

AH O	String operations										_ L
-	6-14-1			Step name Strin	ng operation	15					
# 1	In stream field command	Out stream field threat	Trim type none	Lower/Upper none	Padding none	Pad char	Pad Length	InitCap N	Escape None	Digits none	Remove Special character none
		Field r	name			Param	neters				
		In stre	eam fi	eld		comma	nd				
		Out st	ream	field		threa	t				
		Trim t	уре			none					
		Lower	·/Uppe	er		none					
		Paddi	ng			none					
		InitCa	р			N					

Field name	Parameters
Escape	None
Digits	none
Remove Special character	none

#### Adding the Replace in String Step

While setting up a **Replace in string** step, you will be leaving the parameters blank.



The rows will be populated from the metadata injection build, and elements and data will come from the text source file.

B Replace in string								
			Ste	ep name Replace	in string			
elds string	Out stream field	use RegEx	Search	Replace with	Set empty string?	Replace with field	Whole Word	Case sensitiv
# In stream field								

#### Adding the Filter Rows Step

After you add the **Filter rows** step, you will need to add two conditions:

	Step name	Filter rows
	Send 'true' data to step:	Select values
	Send 'false' data to step:	5
The condition:		
	threat >= [10	]
AND	threat >= [10	1

#	Condition
1	threat >= [10]
2	threat <= [99]

#### Adding the Select Values Step

Use the following parameters, which will filter fields down to the two we want to keep:

(	Selec	t & Alter Ren	move Meta-dat	a		_
	Field	ls :				
	# 1 2	Fieldname command threat	Rename to	Length	Precision	
#		Fieldnam	e			
1		command				
2		threat				

#### Adding the Text File Output Step

Use these parameters to set up the **Text file output** step:

🗎 Text file output	- 🗆 X					
Step name Text	file output					
File Content Fields						
Filename S{tr	file_nm} 😵 Browse.					
Run this as command instead?						
Pass output to servlet						
Create Parent folder 🗹						
Do not create file at start 🗌						
Accept file name from field? 🗌						
File name field	v -					
Extension						
Include steppr in filename?						
Include partition nr in filename?						
Include date in filename?						
Include time in filename?						
Specify Date time format						
Date time format						
1000	200					
Sho	w filename(s)					
Add filenames to result 🗌						
	OK Cancel					
1 Help	OK Calicel					
Field name	Parameters					
Filename	<pre>\${trg_file_nm}</pre>					
Create Parent Folder Checked						

## Step 3: Building the ETL Building Transformation

After you build your template, it is time to create an ETL Building transformation. This transformation will be the driving one that pulls all business rules, populates all missing properties, and outputs a fully runnable ETL transformation:



1. Create a transformation called etl search weighted .ktr:

💥 Transformation properties	
Transformation Parameters Logging Dates Dependencies	Miscellaneous Monitoring
Transformation name :	etl_search_weighted_
Transformation filename	C:\opts\etl\etl_search_weightedktr
Description :	
Extended description:	

2. Set up the parameters for the input/output text files (this could easily be changed to be RDBMS Source/Target locations:

#	Parameter	Default Value
1	rule_file_nm	c:\opts\etl\in_rules.txt
2	rule_key_value	windows
3	src_file_nm	c:\opts\etl\cmd_src_in.txt
4	trg_file_nm	c:\opts\etl\weight_output.txt

3. Next, you will need to set up steps for the transformation.

#### Adding the Text File Input Step

Use the following parameters to create a **Text file input** step.

1. Start with the **File** tab to enter these parameters:

Fi	eld name	Parameters\${rule file	<pre>Parameters \${rule file nm}</pre>				
Selected files:	File/Directory <ol> <li>\${rule_file_nm}</li> </ol>	Wildcard (RegExp)	Exclude wildcard	Required N	Include subfolders N		
File or directory Regular Expression Exclude Regular Expression					Add		

#### PDI Techniques: Guidelines for Metadata Injection

Field name	Parameters
Required	Ν
Include subfolders	Ν

#### 2. Go to the **Content** tab and set the following parameters:

Filetype	CSV			~
Separator	1		٩	Insert TAB
Enclosure	н			
Allow breaks in enclosed fields?				
Escape				
Header	Number of header lines	1		
Footer [	🗌 Number of footer lines 📘			
Wrapped lines?	Number of times wrapped	d 1		
Paged layout (printout)? [	🗖 Number of lines per page	80		
	Document header lines	0		
Compression	None			~
No empty rows				
Include filename in output?	🗌 Filename fieldname			
Rownum in output? [	🗌 Rownum fieldname 📗			
	Rownum by file? 🗌			
Format	mixed			~
Encoding				~
Length	Characters			~
Limit	0			
Be lenient when parsing dates?				
The date format Locale	en_US			~

Field name	Parameters
Filetype	CSV
Separator	
Enclosure	Ш
Header	check it
Number of header lines	1
No empty rows	check it
Format	mixed
Length	Characters
Limit	0
Be lenient when parsing dates?	check it
The date format Locale	en_US

#### 3. Add the following eight fields on the **Fields** tab:

	_		510	p nume [	Text file inpl	u u				
File	Conte	ent Error Handlin	ng Filters F	ields A	dditional ou	tput fields				
#	Nar	ne	Туре		Format	Position	Length	Precision	Cu	
1	Key		String				7		s	
2	Fiel	ield String					6		S	
3	Use	seRegExYN Boolean							S	
4	Sea	rchRegEx	String				25		S	
5	Rep	laceWithValue	Integer		#		15	0	S	
6	Set	mptyYN	Boolean						S	
7	Wh	WholeWordYN E							S	
8	Cas	eSensitiveYN	Boolean						S	
#		Name		Туре						
1		Кеу		Strin	g	set Length to 7				
2		Field		String		set Length to 6				
3		UseRegExYN	1	Boolean		no parameters				
1		SearchRegE	Ex	Strin	g	set Length to 25				
5		ReplaceWit	chValue	Integer		<pre>set Format to # , Length 1 15, and set Precision to #</pre>			<b>to</b> 0	
6		SetEmptyYN	1	Boole	an	no parameters				
7		WholeWord	ζΝ	Boole	an	no parameters				
7		~ ~ ! !	Boolean		no parameters					

## Adding the Get Variables Step

Add the **Get variables** step with the following parameters:

Sield	Get Variable s:							Step r	name	Get Variable	s		
# 1 2 3	Name val_key_va val_src_va val_trg_va	alue lue lue	Variable S{rule_key_value} S{src_file_nm} S{trg_file_nm}	Type String String String		Format	Length	Precisio	'n	Currency	Decimal	Group	Trim type both both both
#		Name		Variable			Ту	ре	Trin	n type	e		
1		va	l_key_val	ue	\${ru	<pre>\${rule_key_value}</pre>		ue}	String		bot	both	
2		va	l_sec_val		\${sr	{src_file_nm}		String		bot	both		
3		va	l_trg_val	ue	\${tr	\${trg_file_nm}			String		bot	both	

#### Adding the Filter Rows Step

Create a **Filter rows** step and use these parameters:

The condition:		
Contraction and	Send 'false' data to step:	
	Send 'true' data to step:	
	Step name	Filter rows

#### Adding the Add Constants Step

Create an **Add constants** step with the following parameters:

0:	Add constant va	lues								
				Step	name Add co	nstants				
#	Name	Туре	Format	Length	Precision	Currency	Decimal	Group	Value	Set empty string?
1	empty	String								Υ
2	boolean_yes	Boolean							Y	N
3	boolean_no	Boolean							N	N

#	Name	Туре	Parameters
1	empty	String	Set empty string <b>to</b> Y
2	Boolean_yes	Boolean	Set Value <b>to</b> Y <b>and</b> Set empty string <b>to</b> N
3	Boolean_no	Boolean	Set Value <b>to</b> N <b>and</b> Set empty string <b>to</b> N

#### Adding the ETL Metadata Injection Step

On the **Inject Metadata** tab, connect fields to files/properties you want to populate for the fully built ETL transformation by locating the **Replace in string** section, and expand to see all the fields.

1. Map these fields to the following:

Field	Source Step	Source Field	
FIELD_IN_STREAM	Add constants	Field	
FIELD_OUT_STREAM	Add constants	empty	
USE_REGEX	Add constants	UseRegExYN	
REPLACE_STRING	Add constants	SearchRegEx	

REPLACE_BY	Add constants	ReplaceWithValue
EMPTY_STRING	Add constants	SetEmptyYN
REPLACE_WITH_FIELD	Add constants	empty
REPLACE_WHOLE_WORD	Add constants	WholeWordYN
CASE_SENSITIVE	Add constants	CaseSensitiveYN

Step Name:

ETL Metadata Injection

#### Transformation:

\${Internal.Entry.Current.Directory}/etl\_template\_search\_weight 📀 Browse...

arget injection step	key	Target description	Source step	Source field
<ul> <li>etl_template_sear</li> </ul>	ch_weighted_			
✓ Filter rows				
CONDITIC	N	The condition to limit the number		
SEND_TRU	E_STEP	The name of the step to send rows		
SEND_FAL	SE_STEP	The name of the step to send rows		
✓ Replace in stri	ng			
✓ FIELDS		The fields that can be injected into.		
FIELD_	IN_STREAM	The field from the input stream to	Add constants	Field
FIELD_	OUT_STREAM	The new field name to output to t	Add constants	empty
USE_R	EGEX	Specify whether to use a regular ex	Add constants	UseRegExYN
REPLA	CE_STRING	Search for matched values of this s	Add constants	SearchRegEx
REPLA	CE_BY	The string to replace the matched	Add constants	ReplaceWithValu
EMPTY	STRING	Specify whether to replace null val	Add constants	SetEmptyYN
REPLA	CE_WITH_FIELD	The field value that will be used to	Add constants	empty
REPLA	CE_WHOLE_WO	Specify whether to replace the enti	Add constants	WholeWordYN
CASE_	SENSITIVE	Specify whether the search is case	Add constants	CaseSensitiveYN
<ul> <li>Select values</li> </ul>		14. 1932		
SELECT_U	NSPECIFIED	Include unspecified fields, ordered	Source field —	
✓ FIELDS		Selected fields		Action Actions
FIELD_	NAME	Fieldname	Filter	OM
FIELD_	RENAME	Rename to		
FIELD_	LENGTH	Length	Source field:	
FIELD_	PRECISION	Precision	Add constants : CaseSensitive	eYN
✓ REMOVES		Removed fields	Add constants : Field	
REMO	VE_NAME	Fieldname	Add constants : Key	
✓ METAS		List of fields to change metadata for	Add constants : ReplaceWith	Value
META	NAME	Fieldname	Add constants : SetEmpty/N	
META	RENAME	Rename to	Add constants : UseRegExYN	
META	LENGTH	Length	Add constants : WholeWord	(N
META	PRECISION	Precision	Add constants : boolean_no	
META	CONVERSION_N	Format	Add constants : boolean_yes	
META	DATE_FORMAT_	Date Format Lenient?	Add constants : empty	
META	DATE_FORMAT	Date Locale	Add constants : val_key_value	22 I
META	DATE FORMAT	Date Time Zone	Add constants : val tro value	

#### 2. On the **Options** tab, set the following parameters:

#### a. Transformation:

\${Internal.Entry.Current.Directory}/etl\_template\_search\_weight
ed\_.ktr

- b. Fill in the **Optional target file (ktr after injection)** field with the output target file for the runnable ETL transformation.
- c. Uncheck **Run resulting transformation**, to prevent the ETL transformation from being executed after you build it.

#### PDI Techniques: Guidelines for Metadata Injection

TL Metadata Injection	
and a mation.	
{Internal.Entry.Current.Dire	ectory}/etl_template_search_weight Srowse
ject Metadata Options	
Step to read from:	
	<b>v</b>
Field name	
Suptornal target file (ktr at	hter injection): Directory/toberun, ktr
Streaming source step:	•
Streaming Source step.	~
Streaming target step:	

# Step 4: Building the Job

Build the job to run two transformations, **build ETL** and **Execute built ETL**:



1. In the Job properties, set the Job name and Job filename:

Nob properties		200	$\times$
Job Parameters Settings	.og Transactions		
Job name:	etl_search_weighted_		Ì
Job filename	C:\opts\etl\etl_search_weigh	tedkjb	
Description:			Ĩ.
Extended description:			^

2. Add the following parameters in the **Parameters** tab. Use the same names as those in the two ETL transformations so they will be picked up correctly when they are shared:

Ş J	ob properties		
ob	Parameters Setti	ings Log Transactions	
Para	meters :		
<b>*</b> #	Parameter	Default value	I
1	rule_file_nm	\${Internal.Entry.Current.Directory}/in_rules.txt	
2	rule_key_value	windows	
3	src_file_nm	\${Internal.Entry.Current.Directory}/cmd_src_in.txt	
4	trg_file_nm	\${Internal.Entry.Current.Directory}/weight_output.txt	

3. Select and connect these items for the job. We will rename the transformations in a moment:



## Step 5: Renaming the First Transformation

Next, rename the first transformation this way:

- 1. Open the first transformation and change the Entry Name to build ETL.
- 2. On the **Options** tab, in the **Transformation** field, enter: \${Internal.Entry.Current.Directory}/etl search weighted .ktr
- 3. Check **Wait for remote transformation to finish**.

K T	Transformation
Ent	ry Name:
bu	uild ETL
Tra	nsformation:
\${	Internal.Entry.Current.Directory}/etl_search_weightedktr 🛛 🔶 Browse
Or	ptions Logging) Arguments) Parameters)
	Run configuration
	Pentaho local v 🛇
	Execution
	Execute every input row
	Clear results rows before execution
	Clear results files before execution
	Wait for remote transformation to finish

4. On the **Parameters** tab, check **Pass parameter values to sub transformation** to make sure the parameters flow through properly during execution:

Entry Name:	
build ETL	
Transformation:	
\${Internal.Entry.Current.Directo	ory}/etl_search_weightedktr
Options Logging Arguments	Parameters
Options Logging Arguments	Parameters
Options Logging Arguments	Parameters ers sub transformation

#### Step 6: Renaming the Second Transformation

Next, rename the second transformation and enter these parameters:

- 1. Open the second transformation and change the Entry Name to Execute Built ETL.
- 2. On the **Options** tab, in the Transformation field, enter: \${Internal.Entry.Current.Directory}/toberun .ktr
- 3. Check Wait for remote transformation to finish.

Execute built ETL	
Transformation:	
{Internal.Entry.Current.Directory}/toberunktr	E
Options Logging Arguments Parameters	
Run configuration:	
Run configuration: Pentaho local 🗸 🗸 🗞	
Run configuration: Pentaho local v 🗣	
Run configuration: Pentaho local Execution Execute every input row	
Run configuration:  Pentaho local  Execution  Execute every input row  Clear results rows before execution	
Run configuration:  Pentaho local  Execution  Execute every input row  Clear results rows before execution  Clear results files before execution	
Run configuration:  Pentaho local  Execution  Execute every input row  Clear results rows before execution  Clear results files before execution  Wait for remote transformation to finish	

4. On the **Parameters** tab, check **Pass parameter values to sub transformation**, so the parameters will flow through properly during execution.

Entry Name:	
Execute built ETL	
Transformation:	
\${Internal.Entry.Current.Directory	y}/toberunktr 🛛 🕸 Brow
Options Logging Arguments F	Parameters
Options Logging Arguments F	Parameters
Options Logging Arguments F Copy results to parameters Pass parameter values to su	Parameters ub transformation

#### Step 7: Running Job and Validating Output

After you have finished, it is time to run the job and validate the output:

1. Run the job, and review and validate the output:

Execution Results						
🕽 History 📋 Logging 🕼 Job metrics 💦 Metrics						
Job / Job Entry	Comment	Result	Reason	Filename	Nr	Log date
<ul> <li>etl_search_weighted</li> </ul>						
Job: etl_search_weighted	Start of job execution		start			2017/11/07 19:39:39
START	Start of job execution		start			2017/11/07 19:39:39
START	Job execution finished	Success			0	2017/11/07 19:39:39
build ETL	Start of job execution		Followed unconditional link	file:///C:/temp/etl/etl_search_weig		2017/11/07 19:39:39
build ETL	Job execution finished	Success		file:///C:/temp/etl/etl_search_weig	1	2017/11/07 19:39:39
Execute built ETL	Start of job execution		Followed link after success	file:///C:/temp/etl/toberunktr		2017/11/07 19:39:39
Execute built ETL	Job execution finished	Success		file:///C:/temp/etl/toberunktr	2	2017/11/07 19:39:39
Success	Start of job execution		Followed link after success			2017/11/07 19:39:39
Success	Job execution finished	Success			2	2017/11/07 19:39:39
lob: etl search weighted	Job execution finished	Success	finished		2	2017/11/07 19:39:39

2. For the windows rule group use case, there is one row to send downstream for further processing:

#### C:\type weight\_output.txt

command;threat

powershell.exe File;10

# **Related Information**

Here are some links to information that you may find helpful while using this best practices document:

- <u>Hitachi Vantara: Data Lake Operations</u>
- Hitachi Vantara: Report Pre-Processors
- <u>Pentaho Components Reference</u>
- Pentaho Customer Use Case: Kingland Systems
- Pentaho Data Sheet: Data Integration
- Pentaho ETL Metadata Injection
- <u>Pentaho Metadata Injection: Accelerating Complex Data Onboarding Processes</u>
- (VIDEO) Pentaho Metadata Injection: Dynamic and Scalable Data Onboarding