HITACHI
Inspire the Next

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

Contents

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, 8.0
Java JDK	1.8.x

The Components Reference 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 ETL Metadata Injection.

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.

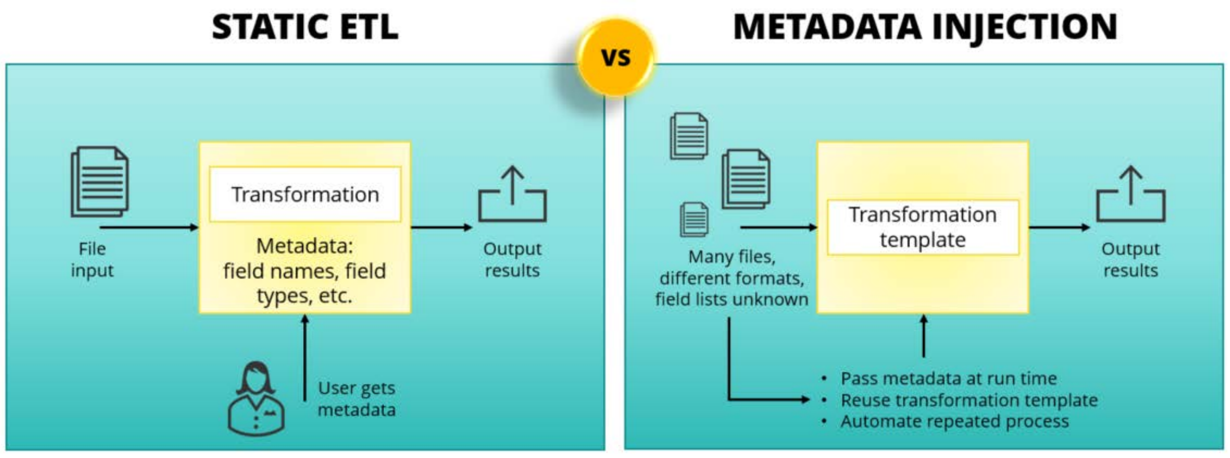

STATIC ETL
VS
METADATA INJECTION
File input
Transformation
Metadata: field names, field types, etc.
Output results
User gets metadata
Many files, different formats, field lists unknown
Transformation template
Output results
• Pass metadata at run time
• Reuse transformation template
• Automate repeated process

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.

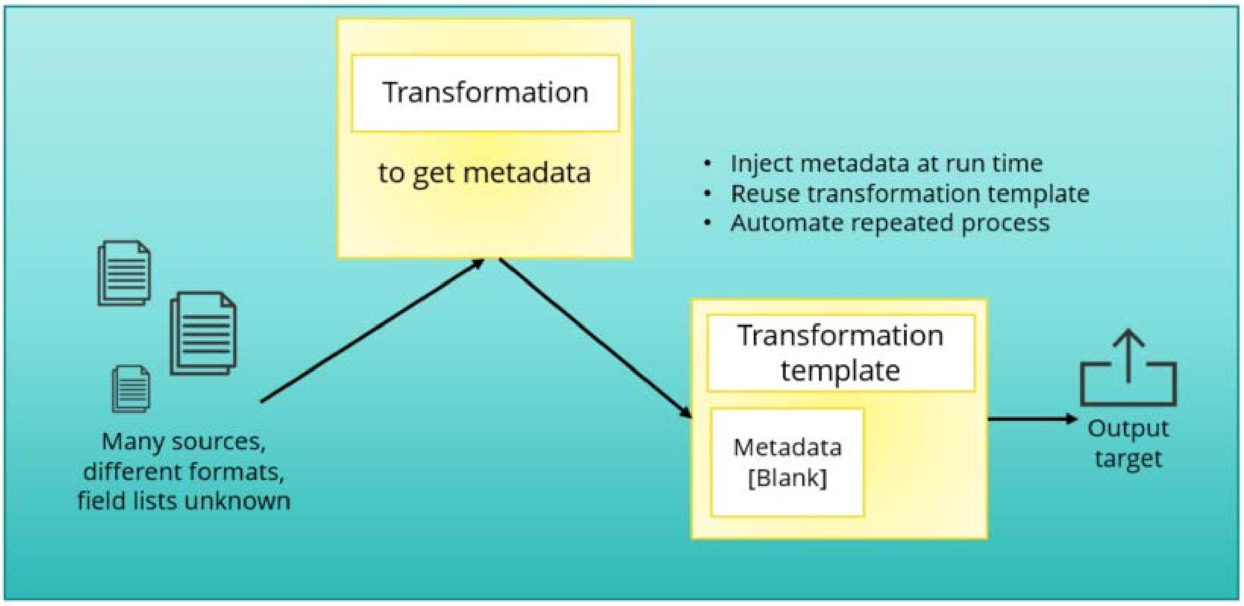

Transformation
to get metadata
Inject metadata at run time
Reuse transformation template
Automate repeated process
Many sources, different formats, field lists unknown
Transformation template
Metadata [Blank]
Output target

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:

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

Table 1: Example Metadata Injection Steps

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

Step Name	Category
Combination lookup/input	Data Warehouse
CouchDb input	Big Data
Data Validator	Validation
ElasticSearch Bulk Insert	Bulk Loading
ETL Metadata Injection	Flow
Get table names	Input
Get Variables	Job
Greenplum Load	Bulk Loading
Hadoop File Input	Big Data
Hadoop File Output	Big Data
HBase Input	Big Data
HBase Output	Big Data
HBase Row Decoder	Big Data
If field value is null	Utility

Step Name	Category
Merge Join	Joins
Merge Rows (diff)	Joins
Multiway Merge Join	Joins
MySQL Bulk Loader	Bulk Loading
Null if...	Utility
Oracle Bulk Loader	Bulk Loading
Replace in string	Transform
Shared Dimension	Flow
Sorted Merge	Joins
Switch/Case	Flow
Synchronize after merge	Output
Vertica Bulk Loader	Bulk Loading
XML Join	Joins

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
- Complex Metadata Injection

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 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->ETL Metadata Injection**.*

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

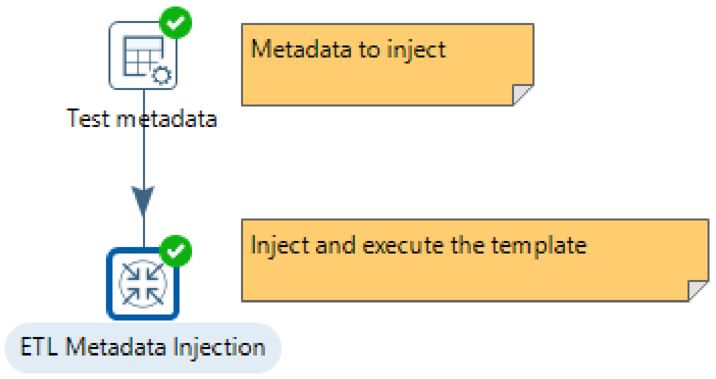
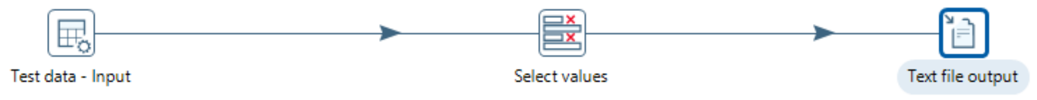


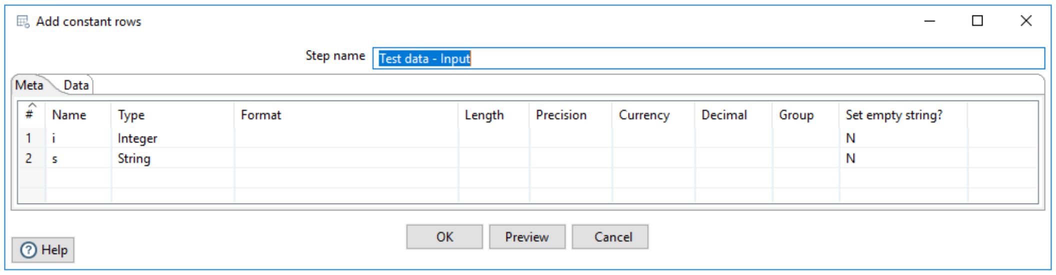
Figure 3: Standard Metadata Injection

Developing the Application

An effective 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:

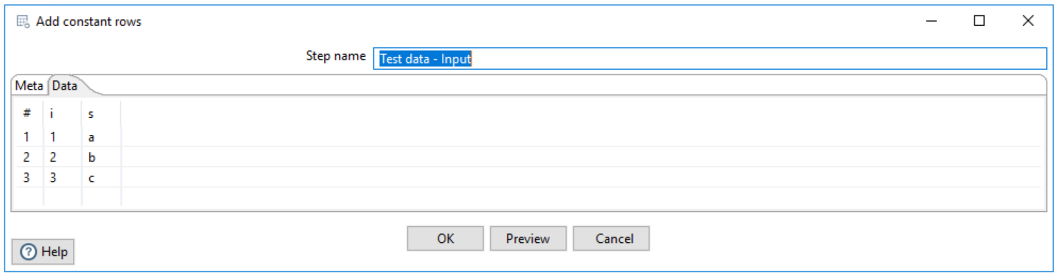


1. Create a new transformation and name it `MDI_Example_1_Standard_template`.
2. From the **Design** tab, drag the **Input >Data Grid** step to the workspace and name it **Test data – Input**.
3. Configure the **Meta** and **Data** tabs for the step as follows, then click **OK**:



Step name: Test data - Input

#	Name	Type	Format	Length	Precision	Currency	Decimal	Group	Set empty string?
1	i	Integer							N
2	s	String							N



Step name: Test data - Input

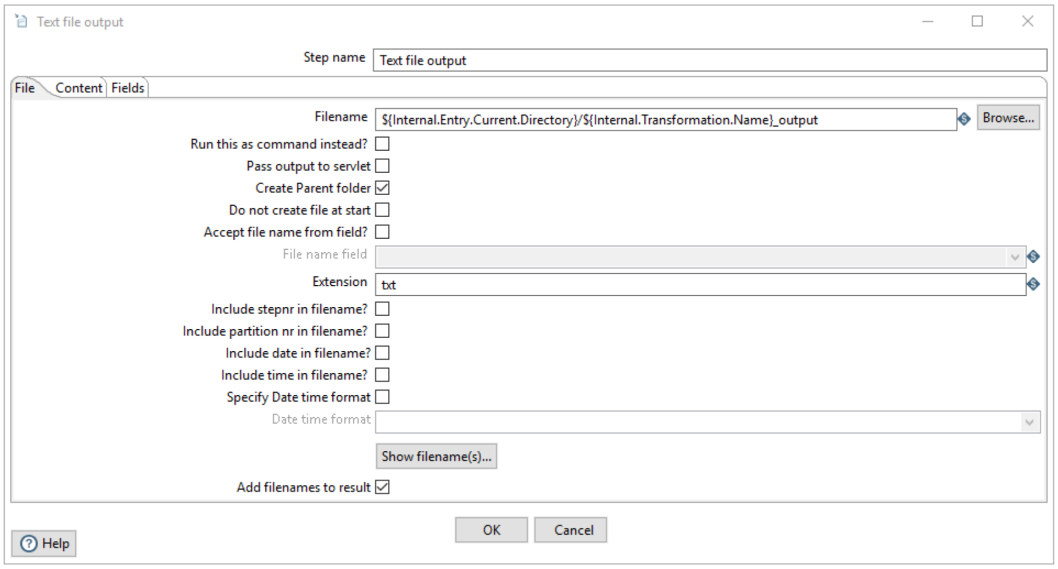
#	i	s
1	1	a
2	2	b
3	3	c

Tab	#	Columns and Parameters
Meta tab	**1**	**Name:** i - **Type:** Integer - **Set empty string:** N
	2	**Name:** s - **Type:** String - **Set empty string:** N
Data tab	**1**	**i:** 1 - **s:** a
	2	**i:** 2 - **s:** b
	3	**i:** 3 - **s:** 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 **Add constant row** step, and configure the **File** tab as shown:



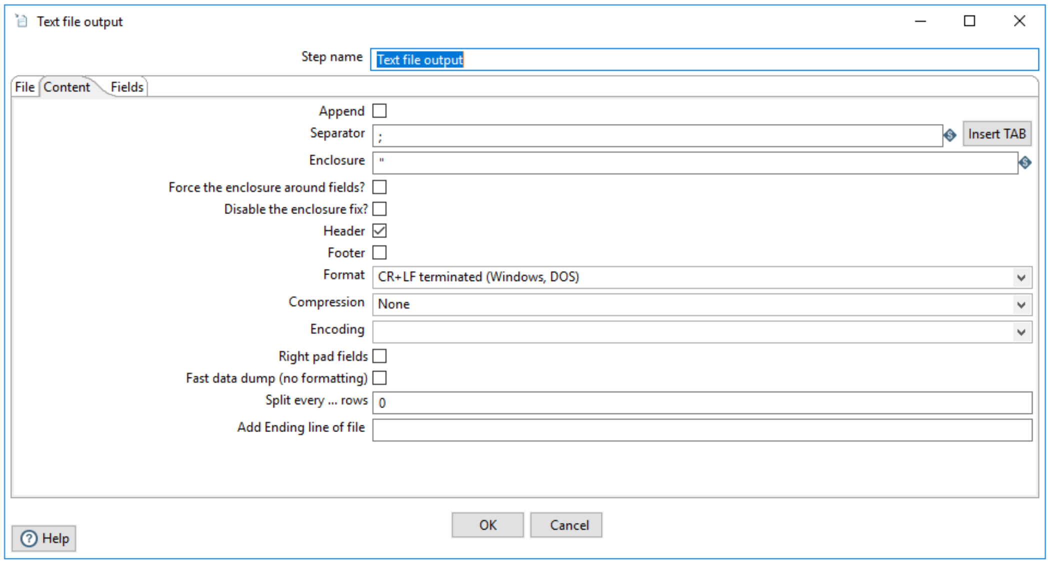
Text file output

Step name: Text file output

File | Content | Fields

Filename: ${Internal.Entry.Current.Directory}/${Internal.Transformation.Name}_output
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:
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:
Add filenames to result ☑

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



Text file output

Step name: Text file output

File | Content | Fields

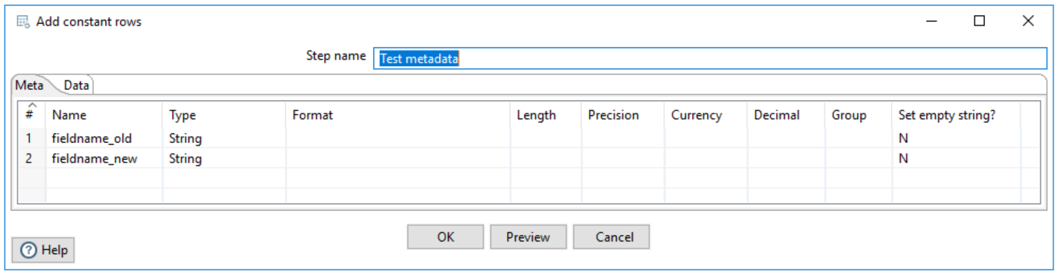
Append ☐
Separator: ;
Enclosure: "
Force the enclosure around fields? ☐
Disable the enclosure fix? ☐
Header ☑
Footer ☐
Format: CR+LF terminated (Windows, DOS)
Compression: None
Encoding:
Right pad fields ☐
Fast data dump (no formatting) ☐
Split every ... rows: 0
Add Ending line of file:

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:

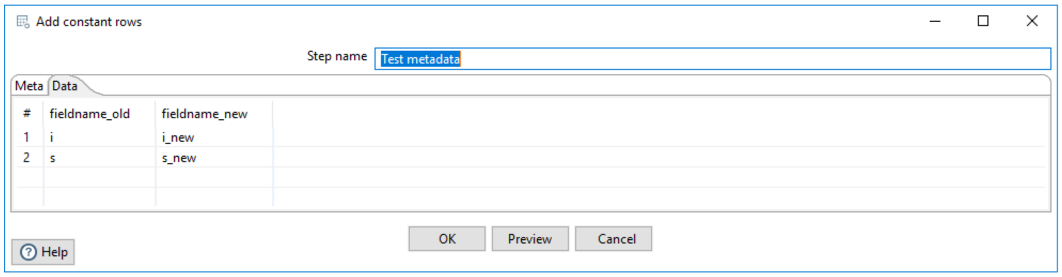
1. Create a new transformation called `MDI_Example_1_Standard`.
2. Drag **Input->Data Grid** and **Flow->ETL Metadata Injection** steps to the design surface and connect them.
3. Rename the **Add Constant Row** to test metadata, as shown in the original snapshot.
4. Next, open the **Test Metadata** step and configure the **Meta** tab:



Step name: Test metadata

#	Name	Type	Format	Length	Precision	Currency	Decimal	Group	Set empty string?
1	fieldname_old	String							N
2	fieldname_new	String							N

5. Configure the **Data** tab:

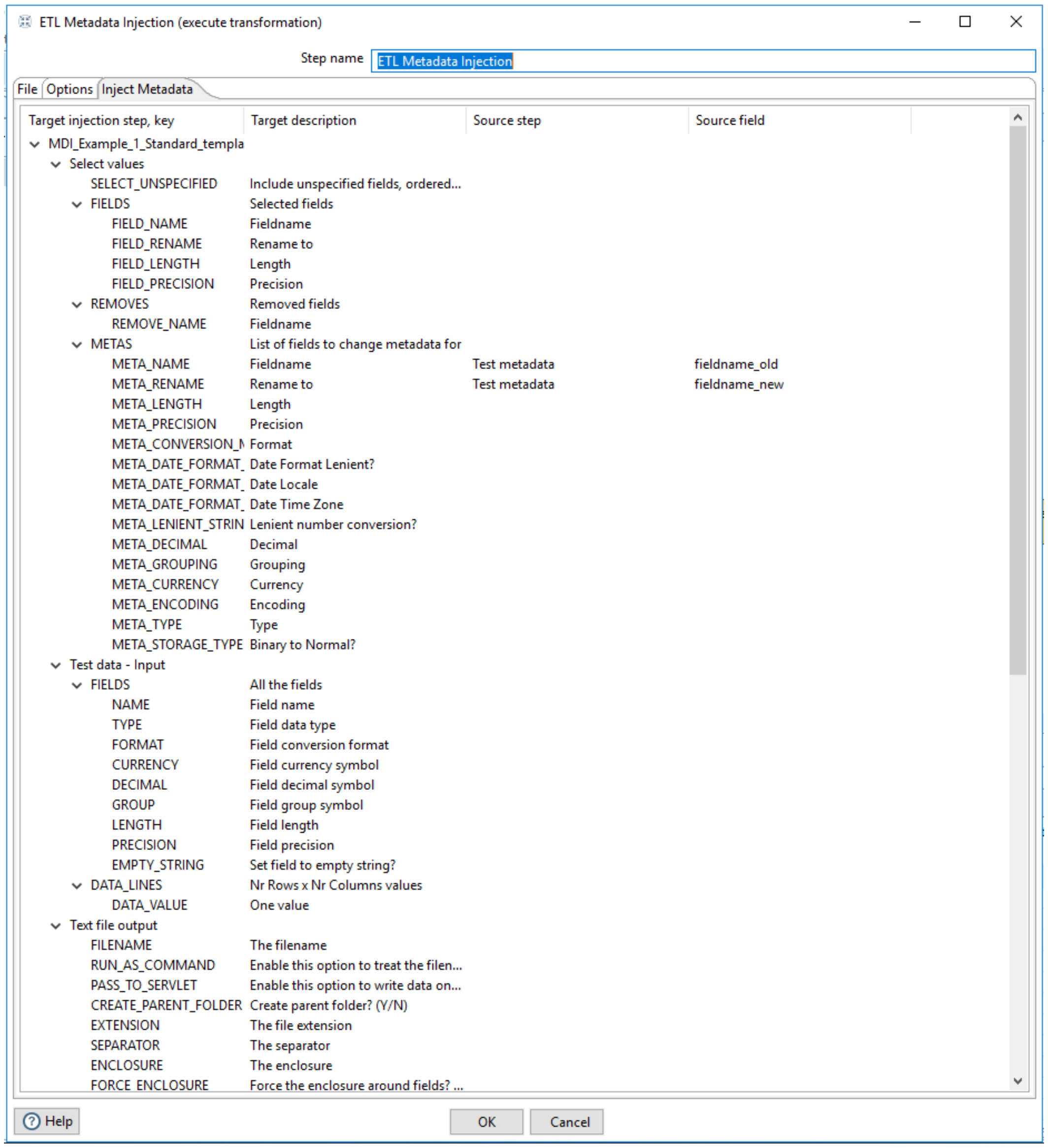


Step name: Test metadata

#	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.

The ETL Metadata Injection step can be edited using the tabs for **File**, **Option**, and **Inject Metadata**. Selecting the **Inject Metadata** tab reveals the **Target injection step key**, **Target description**, **Source step**, and **Source field** for the target transformation:



Target injection step, key	Target description	Source step	Source field
MDI_Example_1_Standard_templa			
Select values			
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		
METAS	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_I	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_TYPE	Type		
META_STORAGE_TYPE	Binary to Normal?		
Test data - Input			
FIELDS	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		
Text file output			
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? ...		

Figure 4: ETL Metadata Injection

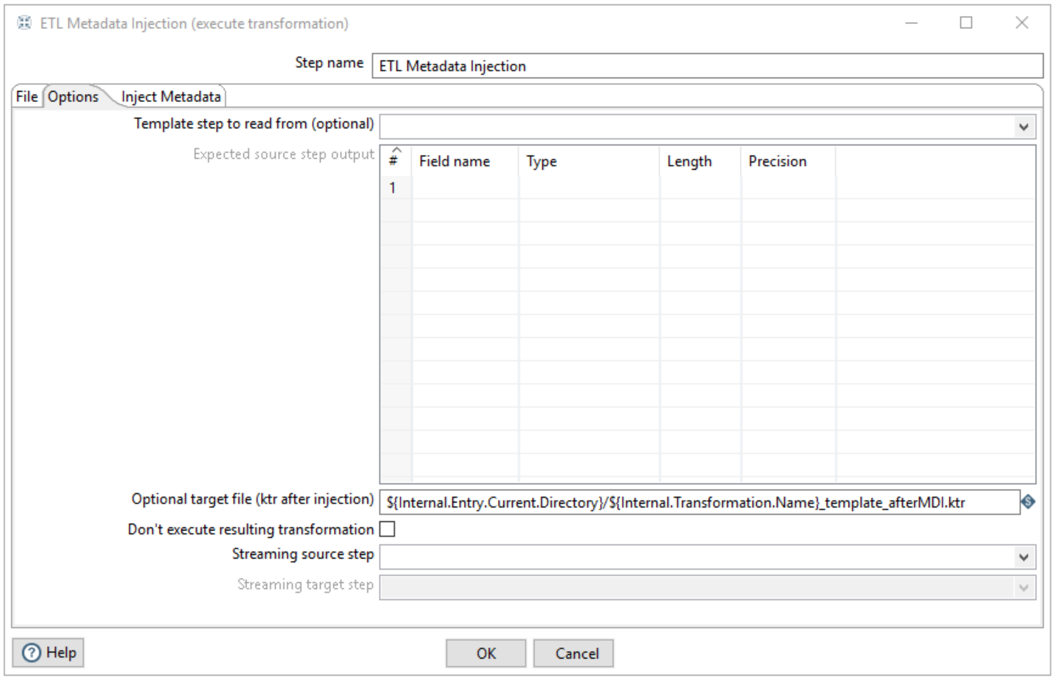
Your solution should resemble the original snapshot under **Standard Metadata Injection** and render a text file called `MDI_Example_1_Standard_template_output.txt` containing columns with **new** in the name.

You can practice this compute pattern by exploring the Pentaho webinar Self-Service Data Onboarding

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 **Metadata Injection** step, and on the **Options** tab, specifying an **Optional target file (ktr after injection)**:



Step name: ETL Metadata Injection

Template step to read from (optional):

Expected source step output

#	Field name	Type	Length	Precision
1				

Optional target file (ktr after injection): ${Internal.Entry.Current.Directory}/${Internal.Transformation.Name}_template_afterMDI.ktr

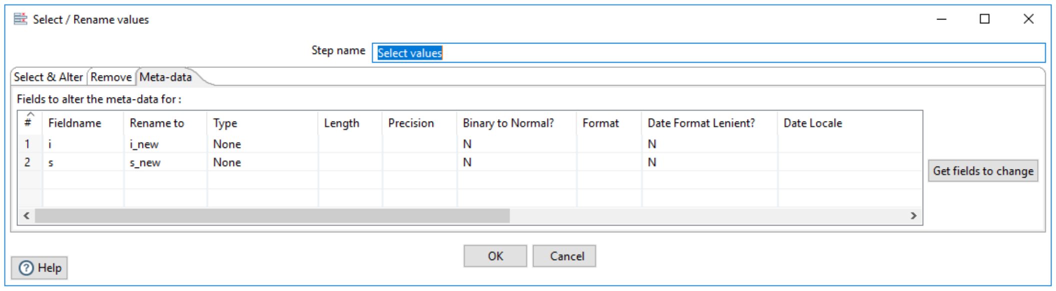
Don't execute resulting transformation: ☐

Streaming source step:

Streaming target step:

Figure 5: 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

Fields to alter the meta-data for :

#	Fieldname	Rename to	Type	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	

Figure 6: Inspect on 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:

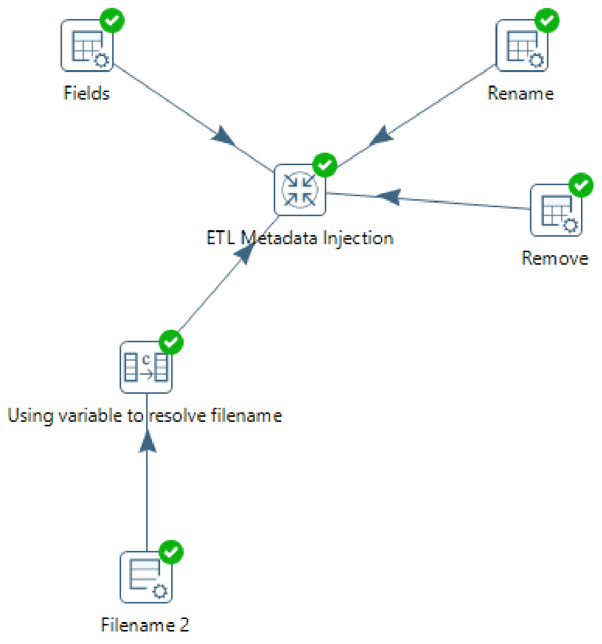


Figure 7: Full Metadata Injection

This pattern can be examined or built upon from in our `samples->transformations->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: [ETL Metadata Injection]().

The ETL metadata injection transformation may look like the following:

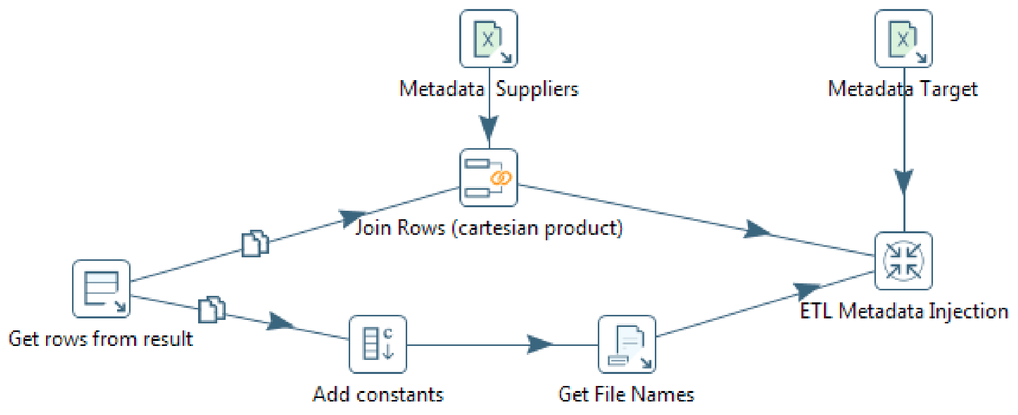


Figure 8: Big Data Metadata Injection

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.

Filling the 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 Filling the Data Lake.

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 Transformation for All Suppliers in the ETL Metadata Injection documentation with the target Hadoop by replacing the **Text File Output** with the **Hadoop File Output**. The `process_all_suppliers.ktr` might look like the following:

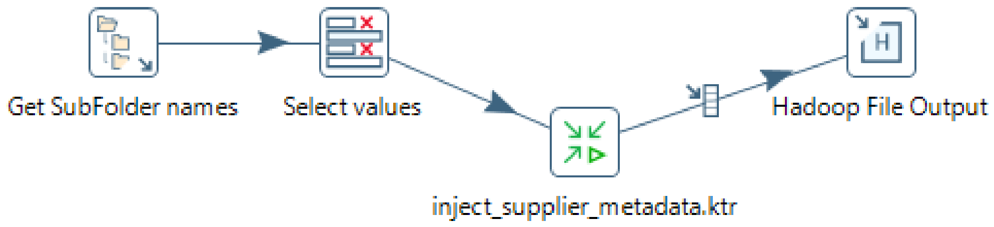


Figure 9: Example of a Process All Suppliers Transformation

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, filter-specific 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.

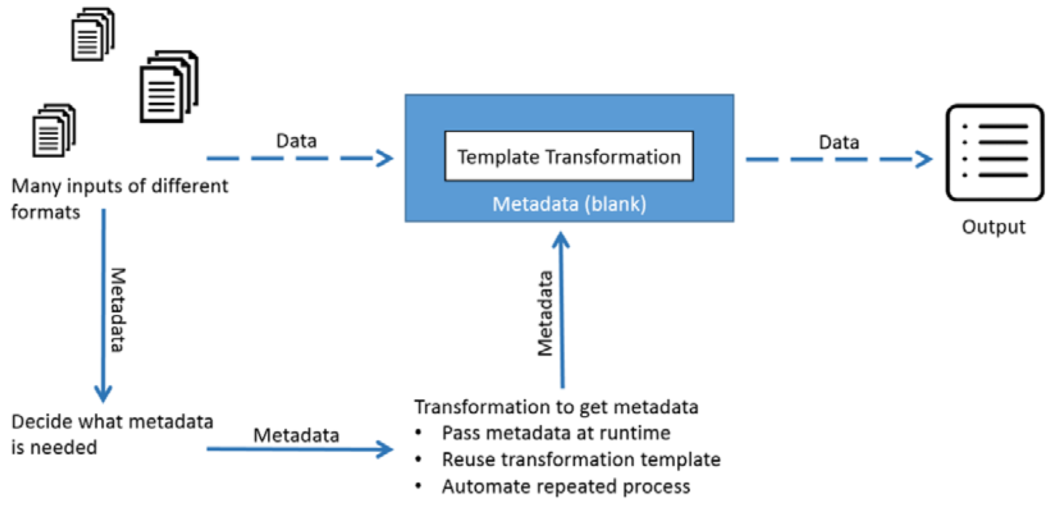

Many inputs of different formats
Data
Template Transformation
Metadata (blank)
Data
Output
Metadata
Metadata
Decide what metadata is needed
Metadata
Transformation to get metadata
• Pass metadata at runtime
• Reuse transformation template
• Automate repeated process

Figure 10: Using 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: Build 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:

Make sure you try this in a test environment, not in a production environment.

```
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	`Key`	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

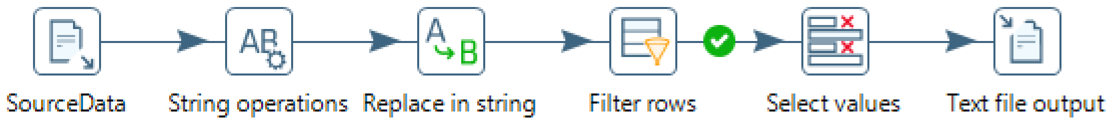
4. Use this input for the dynamic rules to be pulled in and built into the output:

```
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
```

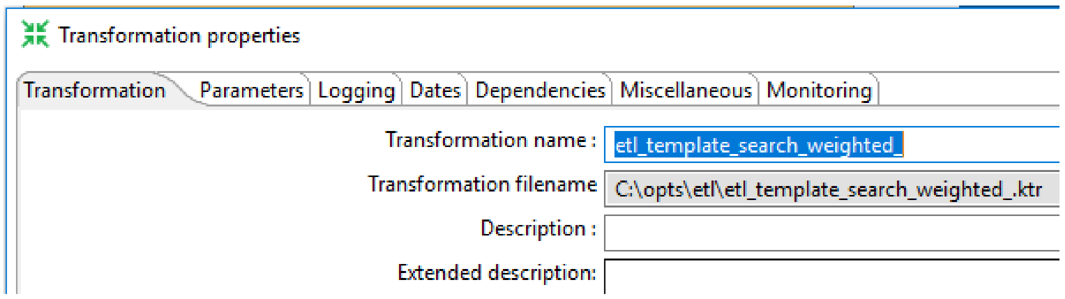
Step 2: Build the ETL Template Transformation

After you create the sample source file, you'll need to build a template for ETL transformations:

1. This transformation will lay out steps for data processing rules:

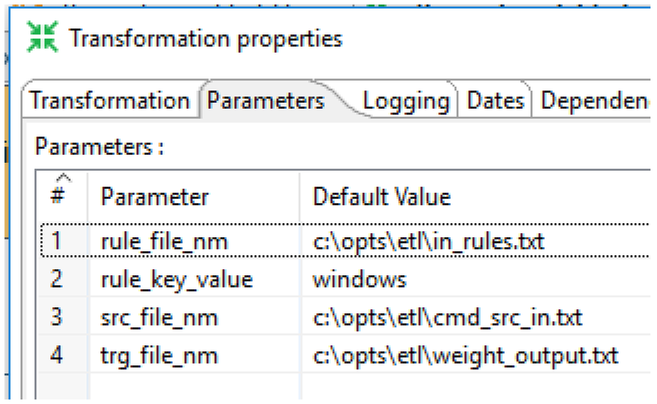


2. Create the transformation called `etl_template_search_weighted_.ktr`:



Transformation name : etl_template_search_weighted_
Transformation filename C:\opts\etl\etl_template_search_weighted_.ktr
Description :
Extended description:

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



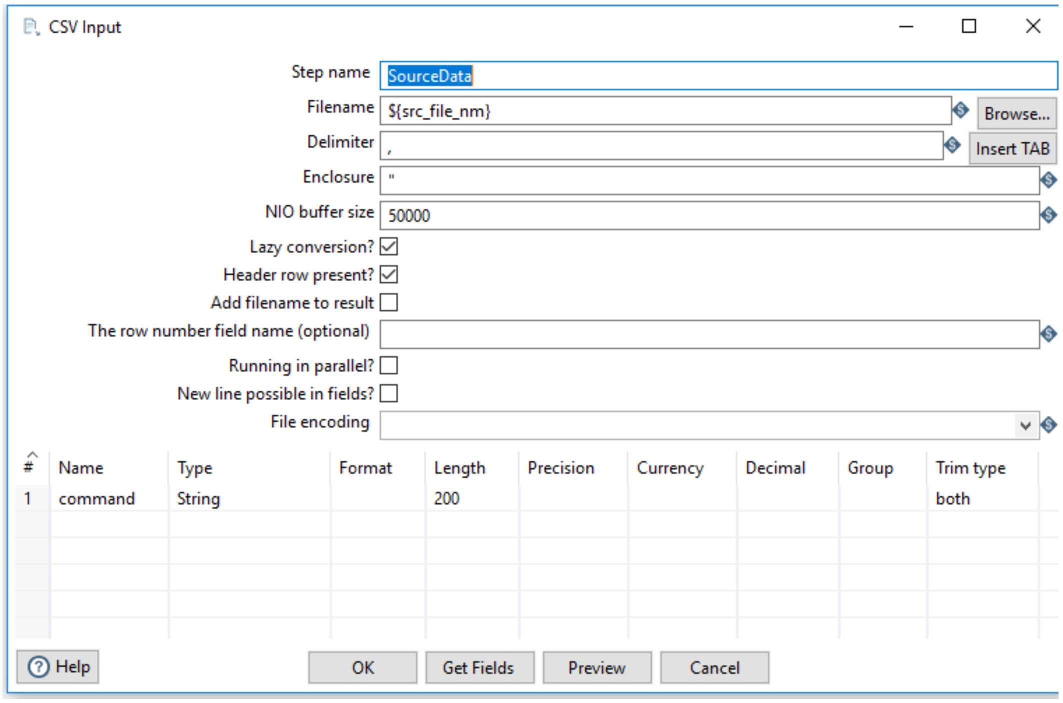
Parameters :

#	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

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

*Pt. 1: Add the **CSV Input** step*

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



CSV Input

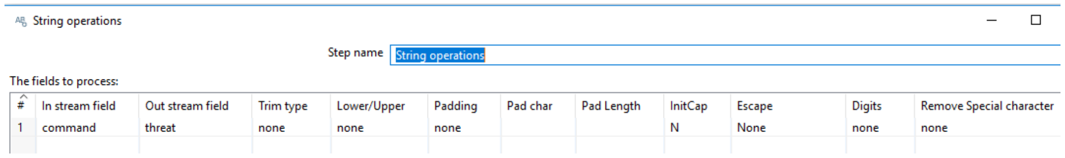
Step name: SourceData
Filename: ${src_file_nm}
Delimiter: ,
Enclosure: "
NIO buffer size: 50000
Lazy conversion? ✓
Header row present? ✓
Add filename to result: ☐
The row number field name (optional):
Running in parallel? ☐
New line possible in fields? ☐
File encoding:

#	Name	Type	Format	Length	Precision	Currency	Decimal	Group	Trim type
1	command	String		200					both

Step name	Filename	Parameters
SourceData	$(src_file_nm) ← ktr parameter	**Name:** command **Type:** String **Length:** 200 **Type:** both

Pt. 2: Add the **String operations** step

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


String operations

Step name: String operations

The fields to process:

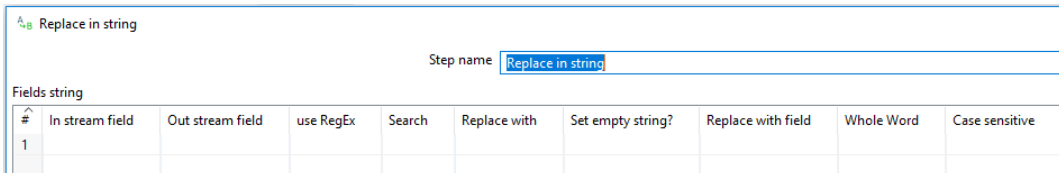
#	In stream field	Out stream field	Trim type	Lower/Upper	Padding	Pad char	Pad Length	InitCap	Escape	Digits	Remove Special character
1	command	threat	none	none	none			N	None	none	none

Field name	Parameters
In stream field	`command`
Out stream field	`threat`
Trim type	`none`
Lower/Upper	`none`
Padding	`none`
InitCap	`N`
Escape	`None`
Digits	`none`
Remove Special character	`none`

Pt. 3: Add 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.


Replace in string

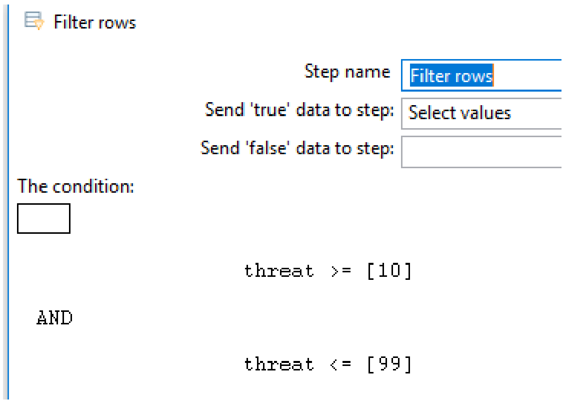
Step name: Replace in string

Fields string

#	In stream field	Out stream field	use RegEx	Search	Replace with	Set empty string?	Replace with field	Whole Word	Case sensitive
1									

*Pt. 4: Add the **Filter Rows** step*

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



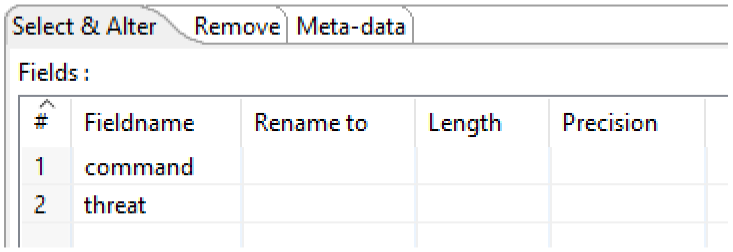
Step name: Filter rows
Send 'true' data to step: Select values
Send 'false' data to step:
The condition:

```
                threat >= [10]
AND
                threat <= [99]
```

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

*Pt. 5: Add the **Select Values** step*

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

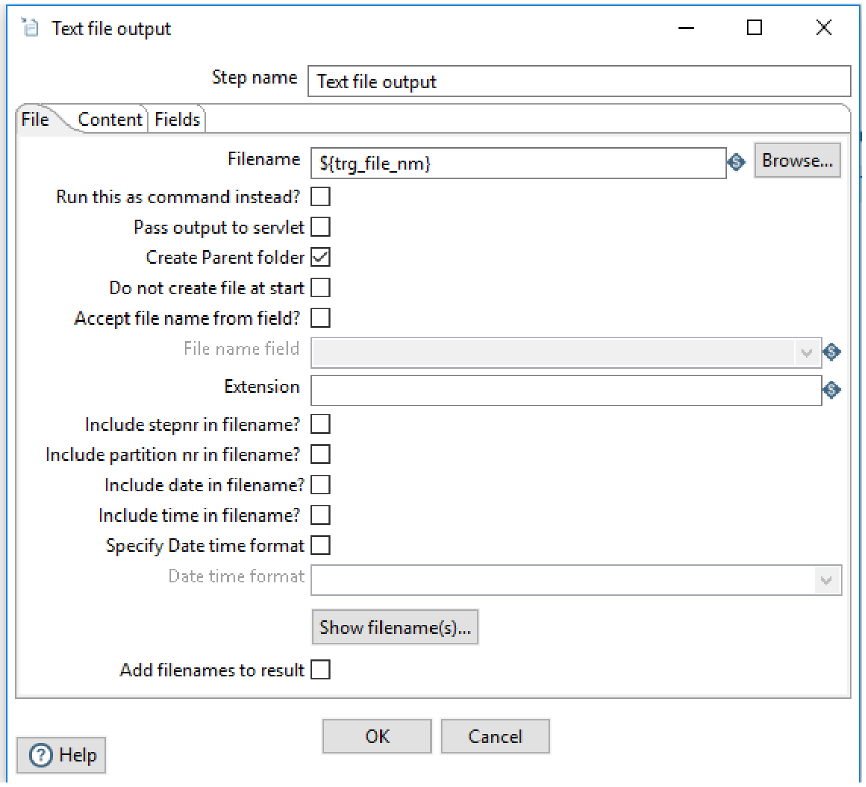


#	Fieldname	Rename to	Length	Precision
1	command			
2	threat			

#	Fieldname
1	`command`
2	`threat`

*Pt. 6: Add the **Text File Output** step*

Use these parameters to set up the **Text File Output** step:



Step name: Text file output

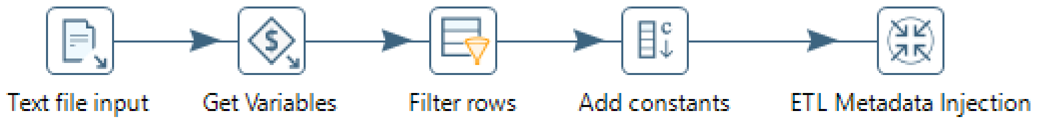
Filename: ${trg_file_nm}

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:
Extension:
Include stepnr in filename? ☐
Include partition nr in filename? ☐
Include date in filename? ☐
Include time in filename? ☐
Specify Date time format ☐
Date time format:
Add filenames to result ☐

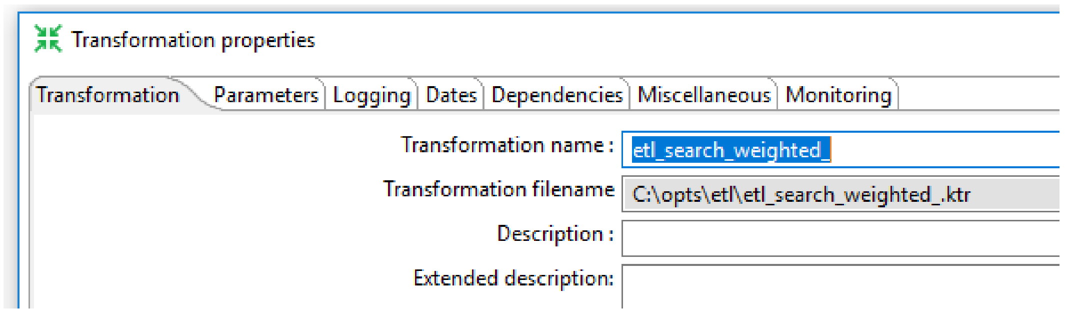
Field name	Parameters
Filename	`${trg_file_nm}`
Create Parent Folder	Checked

Step 3: Build 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 name : etl_search_weighted_
Transformation filename C:\opts\etl\etl_search_weighted_.ktr
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:

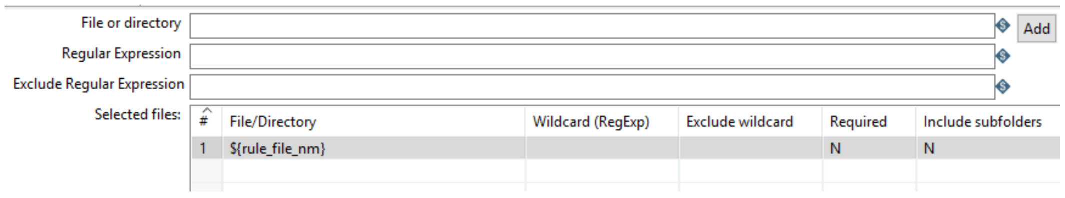
#	Parameter	Default Value
1	**rule_file_nm**	`c:\opts\etl\in_rules.txt`
2	**rule_key_value**	`windows`
3	**src_file_nm**	`c:\opts\etl\cmd_src_in.txt`
4	**trg_file_nm**	`c:\opts\etl\weight_output.txt`

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

*Pt. 1: Add the **Text File Input** step*

Use with the following parameters to create a **Text File Input** step.

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

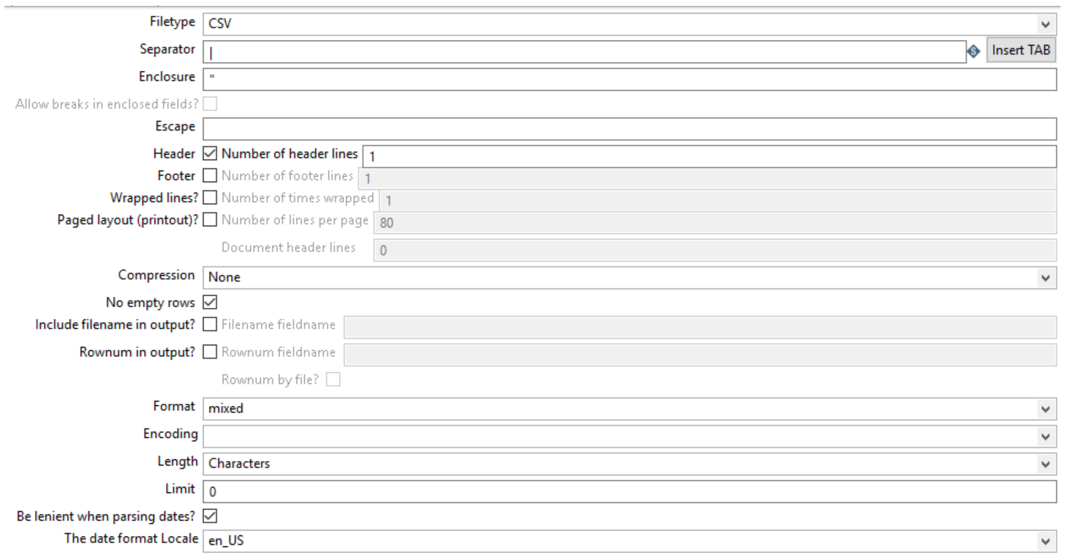


File or directory
Regular Expression
Exclude Regular Expression
Selected files:

#	File/Directory	Wildcard (RegExp)	Exclude wildcard	Required	Include subfolders
1	${rule_file_nm}			N	N

Field name	Parameters
File/Directory	`${rule_file_nm}`
Required	N
Include subfolders	N

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

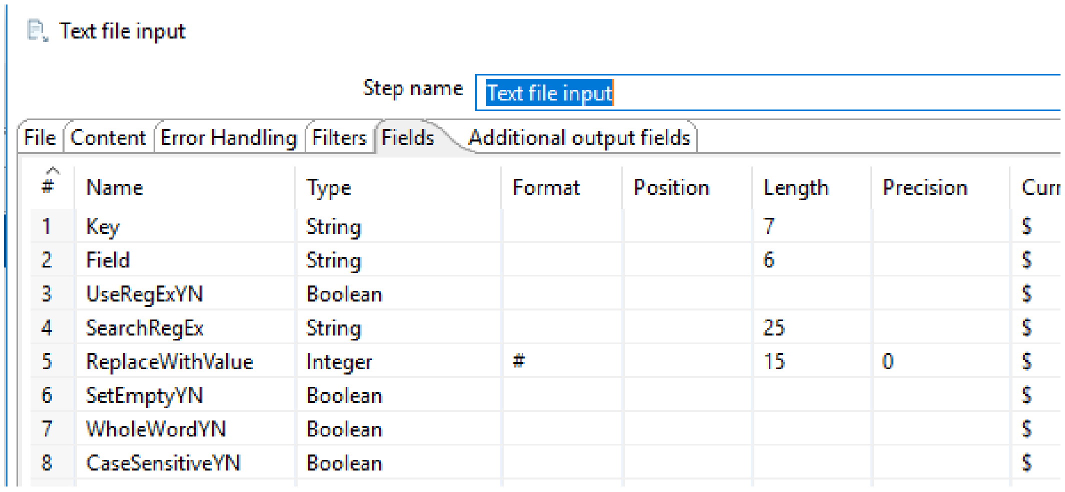


Filetype	CSV
Separator	|
Enclosure	"
Allow breaks in enclosed fields?	☐
Escape	
Header	☑ Number of header lines 1
Footer	☐ Number of footer lines 1
Wrapped lines?	☐ Number of times wrapped 1
Paged layout (printout)?	☐ Number of lines per page 80
	Document header lines 0
Compression	None
No empty rows	☑
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	"
Header	check it
Number of header lines	1
No empty rows	check it
Format	mixed

Field name	Parameters
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:



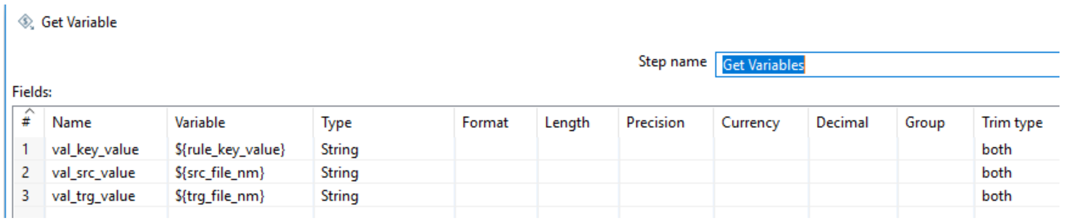
Step name: Text file input

#	Name	Type	Format	Position	Length	Precision	Curr
1	Key	String			7		$
2	Field	String			6		$
3	UseRegExYN	Boolean					$
4	SearchRegEx	String			25		$
5	ReplaceWithValue	Integer	#		15	0	$
6	SetEmptyYN	Boolean					$
7	WholeWordYN	Boolean					$
8	CaseSensitiveYN	Boolean					$

#	Name	Type	Parameters
1	`Key`	`String`	set `Length` to `7`
2	`Field`	`String`	set `Length` to `6`
3	`UseRegExYN`	`Boolean`	*no parameters*
4	`SearchRegEx`	`String`	set `Length` to `25`
5	`ReplaceWithValue`	`Integer`	set `Format` to `#`, `Length` to `15`, and set `Precision` to `0`
6	`SetEmptyYN`	`Boolean`	*no parameters*
7	`WholeWordYN`	`Boolean`	*no parameters*
8	`CaseSensitiveYN`	`Boolean`	*no parameters*

*Pt. 2: Add the **Get Variables** step*

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



Get Variable

Step name: Get Variables

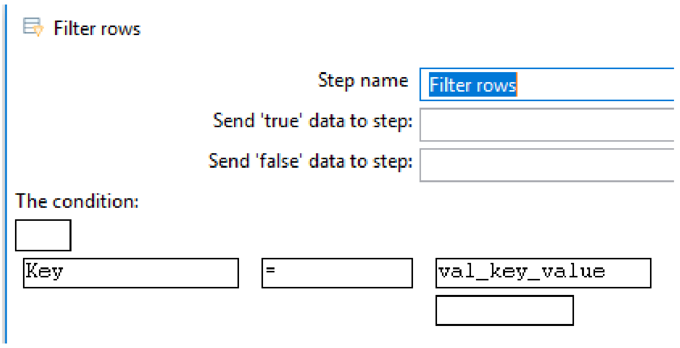
Fields:

#	Name	Variable	Type	Format	Length	Precision	Currency	Decimal	Group	Trim type
1	val_key_value	${rule_key_value}	String							both
2	val_src_value	${src_file_nm}	String							both
3	val_trg_value	${trg_file_nm}	String							both

#	Name	Variable	Type	Trim type
1	`val_key_value`	`${rule_key_value}`	`String`	`both`
2	`val_sec_val`	`${src_file_nm}`	`String`	`both`
3	`val_trg_value`	`${trg_file_nm}`	`String`	`both`

*Pt. 3: Add the **Filter Rows** step*

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



Filter rows

Step name: Filter rows

Send 'true' data to step:

Send 'false' data to step:

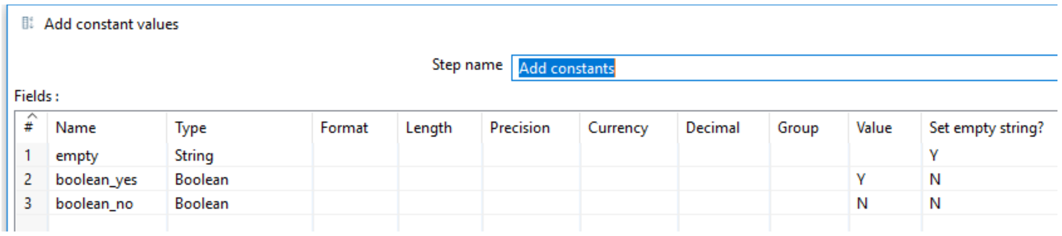
The condition:

Key = val_key_value

Field name	Parameter
The condition:	`Key = val_key_value`

*Pt. 4: Add the **Add Constants** step*

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


Add constant values

Step name: Add constants

Fields :

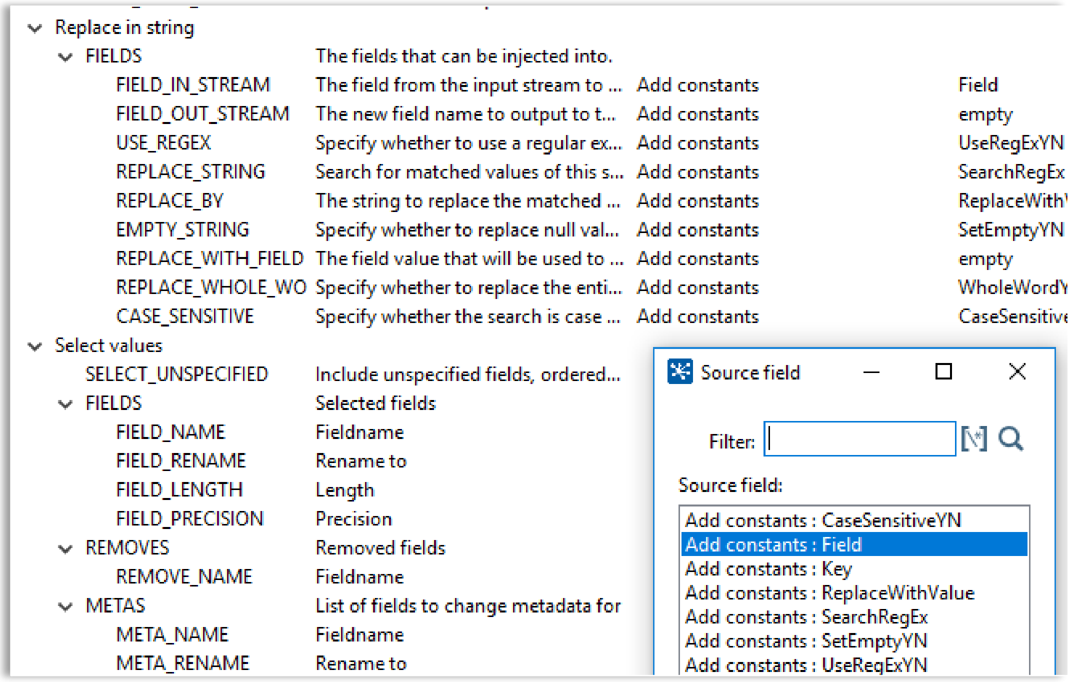
#	Name	Type	Format	Length	Precision	Currency	Decimal	Group	Value	Set empty string?
1	empty	String								Y
2	boolean_yes	Boolean							Y	N
3	boolean_no	Boolean							N	N

#	Name	Type	Parameters
1	`empty`	`String`	`Set empty string to Y`
2	`Boolean_yes`	`Boolean`	`Set Value to Y and Set empty string to N`
3	`Boolean_no`	`Boolean`	`Set Value to N and Set empty string to N`

*Pt. 5: Add the **ETL Metadata Injection** step:*

Create an ETL Metadata Injection step, then follow these guidelines to configure it:

1. Go to the **Inject Metadata** tab and locate the Replace in string section.
2. Expand that section to see the fields that can be connected to files/properties you want to populate for the fully built ETL transformation.



Replace in string			
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_REGEX	Specify whether to use a regular ex...	Add constants	UseRegExYN
REPLACE_STRING	Search for matched values of this s...	Add constants	SearchRegEx
REPLACE_BY	The string to replace the matched ...	Add constants	ReplaceWith
EMPTY_STRING	Specify whether to replace null val...	Add constants	SetEmptyYN
REPLACE_WITH_FIELD	The field value that will be used to ...	Add constants	empty
REPLACE_WHOLE_WO	Specify whether to replace the enti...	Add constants	WholeWord
CASE_SENSITIVE	Specify whether the search is case ...	Add constants	CaseSensitive
Select values			
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		
METAS	List of fields to change metadata for		
META_NAME	Fieldname		
META_RENAME	Rename to		

Source field

Filter:

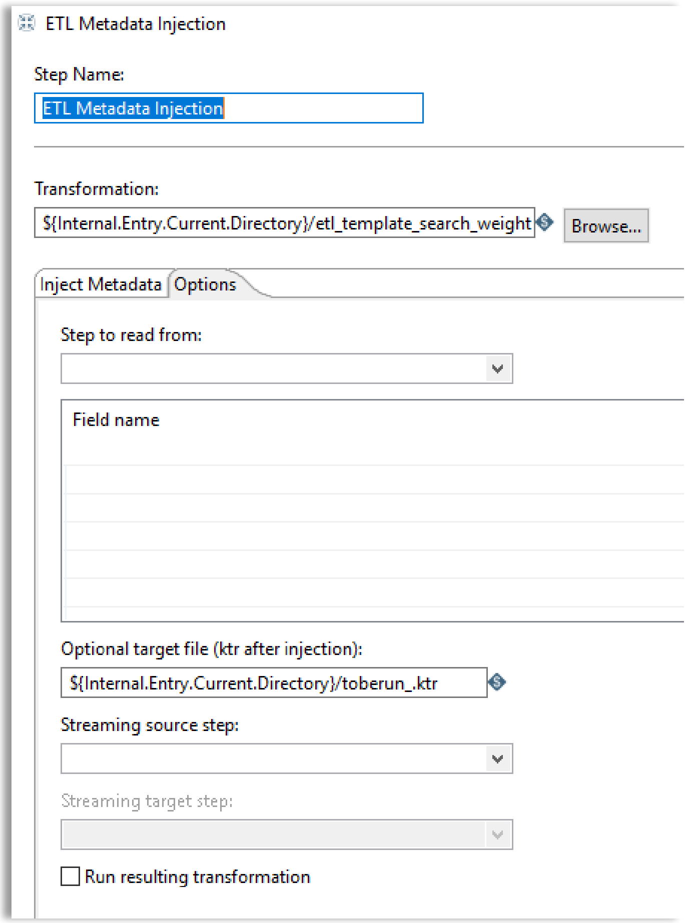
Source field:

- Add constants : CaseSensitiveYN
- Add constants : Field
- Add constants : Key
- Add constants : ReplaceWithValue
- Add constants : SearchRegEx
- Add constants : SetEmptyYN
- Add constants : UseRegExYN

3. 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

4. On the **Options** tab, set the following parameters:


ETL Metadata Injection

Step Name: ETL Metadata Injection

Transformation: ${Internal.Entry.Current.Directory}/etl_template_search_weight

Step to read from:

Field name

Optional target file (ktr after injection): ${Internal.Entry.Current.Directory}/toberun_.ktr

Streaming source step:

Streaming target step:

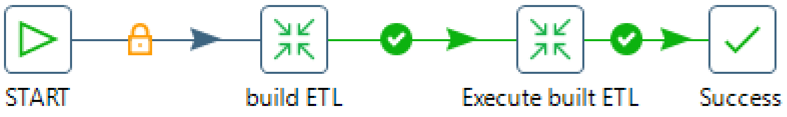
Run resulting transformation

Field name	Parameter
Transformation	`${Internal.Entry.Current.Directory}/etl_template_search_weighted_.ktr`
Optional target file (ktr after injection)	Fill in the output target file for the runnable ETL transformation.

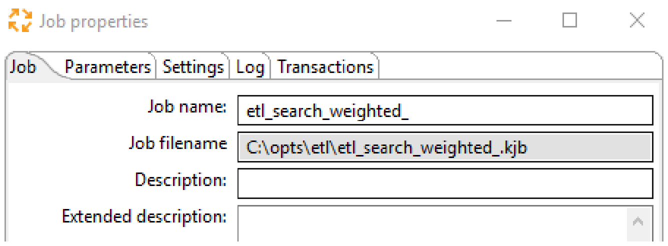
Field name	Parameter
Run resulting transformation	Uncheck

Step 4: Build 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 filename:

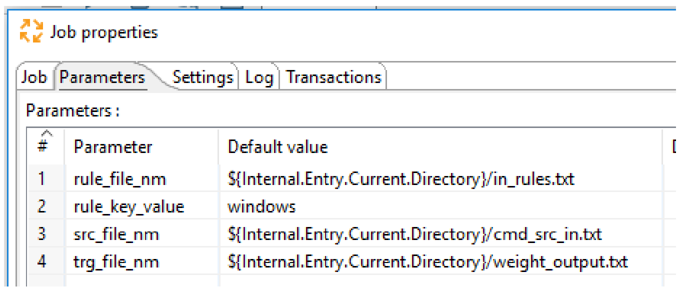


Job properties

Job | Parameters | Settings | Log | Transactions

Job name: etl_search_weighted_
Job filename C:\opts\etl\etl_search_weighted_.kjb
Description:
Extended description:

2. Add the following parameters in the **Parameters** tab:



Job properties

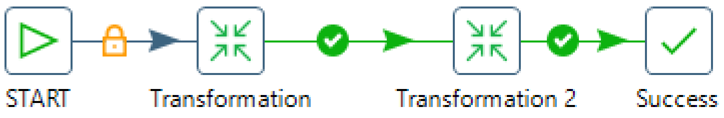
Job | Parameters | Settings | Log | Transactions

Parameters :

#	Parameter	Default value
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

They will be picked up correctly when they are shared, if you use the same names as those in the two ETL transformations.

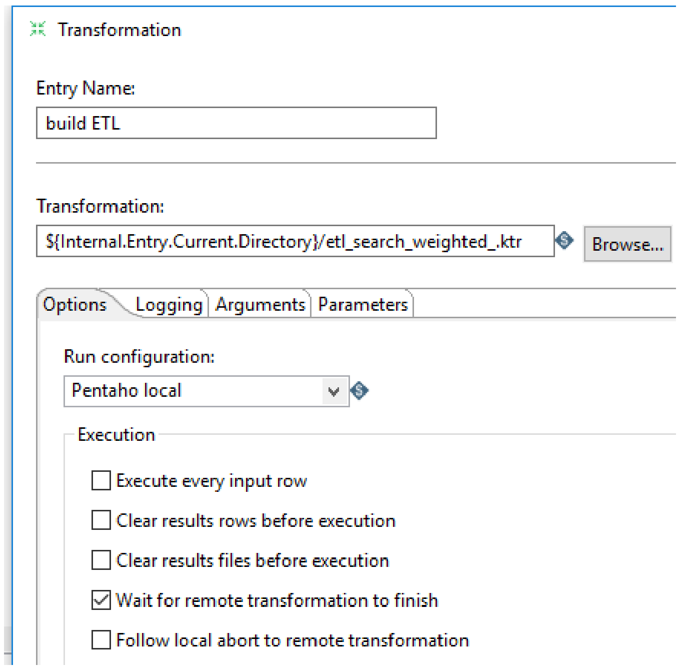
3. Select and connect these items for the job. We will rename the transformations in a moment:



Step 5: Rename the First Transformation

Now it is time to rename the first transformation. These steps show you how:

1. Open the first transformation and change the **Entry Name** to `build ETL`.
    a. On the **Options** tab in the Transformation field, enter: `${Internal.Entry.Current.Directory}/etl_search_weighted_.ktr`
    b. Then check **Wait for remote transformation to finish**.


Transformation

Entry Name:
build ETL

Transformation:
${Internal.Entry.Current.Directory}/etl_search_weighted_.ktr

Options | Logging | Arguments | Parameters

Run configuration:
Pentaho local

Execution

- [ ] Execute every input row
- [ ] Clear results rows before execution
- [ ] Clear results files before execution
- [x] Wait for remote transformation to finish
- [ ] Follow local abort to remote transformation

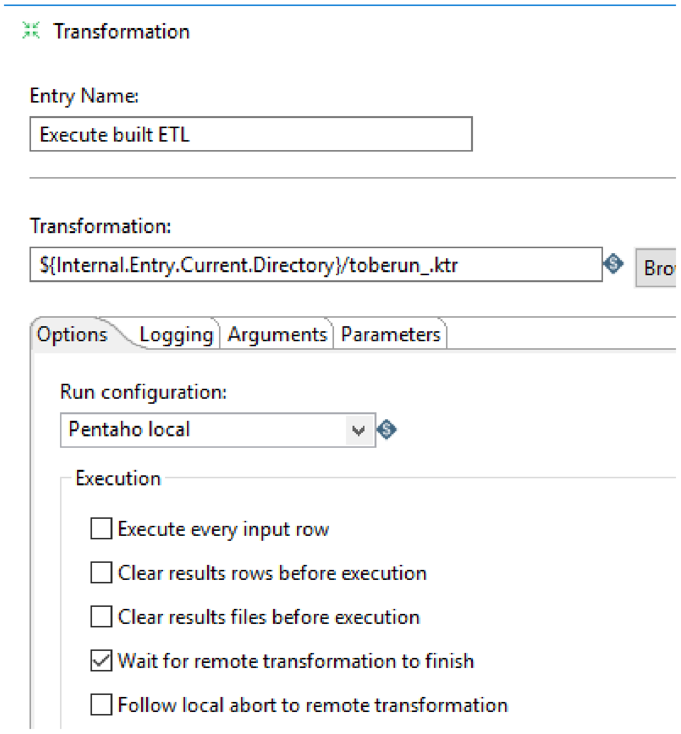
2. On the **Parameters** tab, check **Pass parameter values to sub transformation** to make sure the parameters flow through properly during execution.

Step 6: Rename the Second Transformation

Rename the second transformation and enter these parameters:

1. Change the **Entry Name** to `Execute Built ETL`.
2. On the **Options** tab tab in the Transformation field, enter: `${Internal.Entry.Current.Directory}/toberun_.ktr`

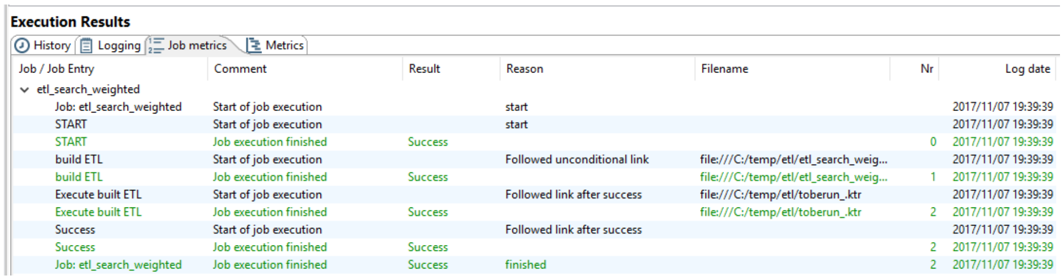
3. Then check **Wait for remote transformation to finish**.


Transformation
Entry Name:
Execute built ETL
Transformation:
${Internal.Entry.Current.Directory}/toberun_.ktr
Bro
Options
Logging
Arguments
Parameters
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
Follow local abort to remote transformation

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

Step 7: Run Job and Validate Output

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


Execution Results

History | Logging | Job metrics | Metrics

Job / Job Entry	Comment	Result	Reason	Filename	Nr	Log date
etl_search_weighted						
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/toberun_.ktr		2017/11/07 19:39:39
Execute built ETL	Job execution finished	Success		file:///C:/temp/etl/toberun_.ktr	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
Job: etl_search_weighted	Job execution finished	Success	finished		2	2017/11/07 19:39:39

For the windows rule group use case, validate one row by sending 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:

- Hitachi Vantara: Report Pre-Processors
- Matt Casters' Blog: Dynamic and Scalable ETL
- Pentaho Customer Use Case: Kingland Systems
- Pentaho Data Sheet: Data Integration
- Pentaho ETL Metadata Injection
- Pentaho Metadata Injection: Accelerating Complex Data Onboarding Processes
- Pentaho Streamlined Data Refinery (SDR)
- (VIDEO) Pentaho Metadata Injection: Dynamic and Scalable Data Onboarding
- (WEBINAR) Pentaho Self-Service Data Onboarding