HITACHI
Inspire the Next

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

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

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 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:

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.

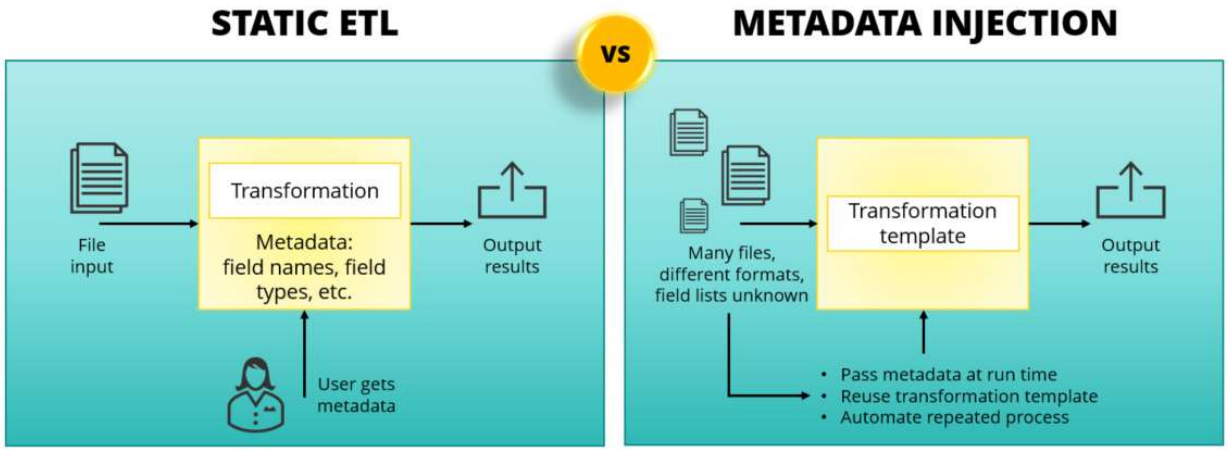


Figure 1: Comparing Static ETL with Metadata Injection for Data Onboarding

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.

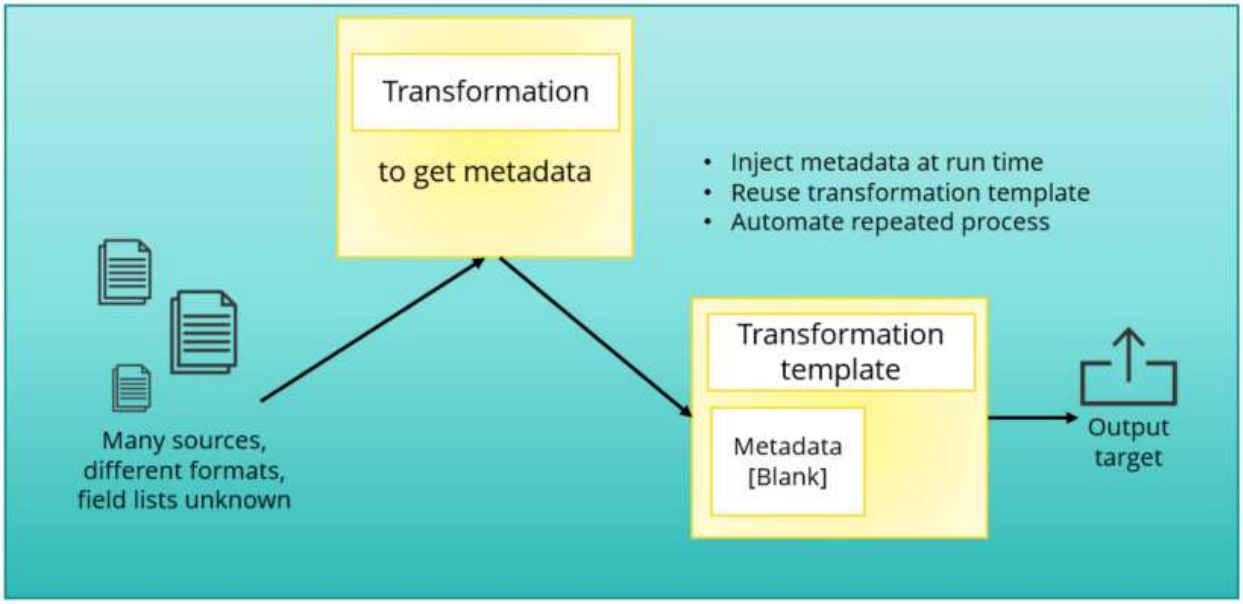


Figure 2: Metadata Injection Solution Architecture

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:

- 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
Add XML	Transform
Annotate stream	Flow
Append streams	Flow
Avro input	Big Data

Step Name	Category
JSON input	Input
MapReduce input	Big Data
MapReduce output	Big Data
Memory group by	Statistics

Step Name	Category
Combination lookup/update	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

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 → ETL metadata injection**.*

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

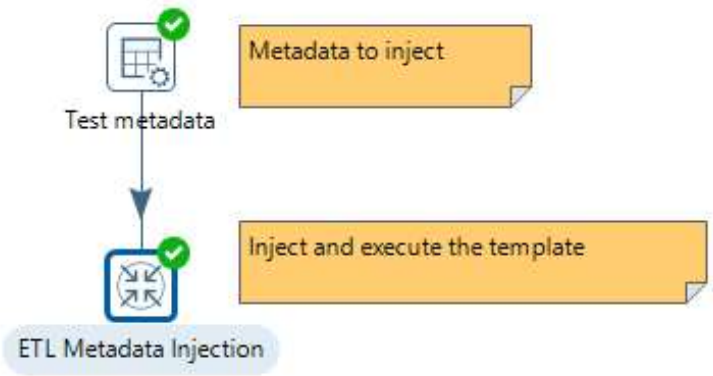
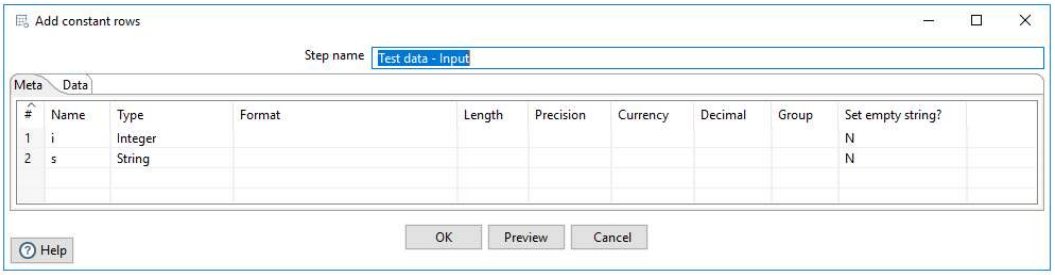


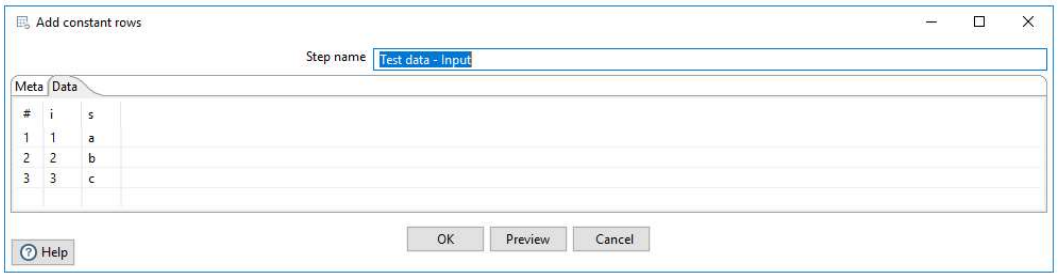
Figure 3: Standard Metadata Injection

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`.
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**:



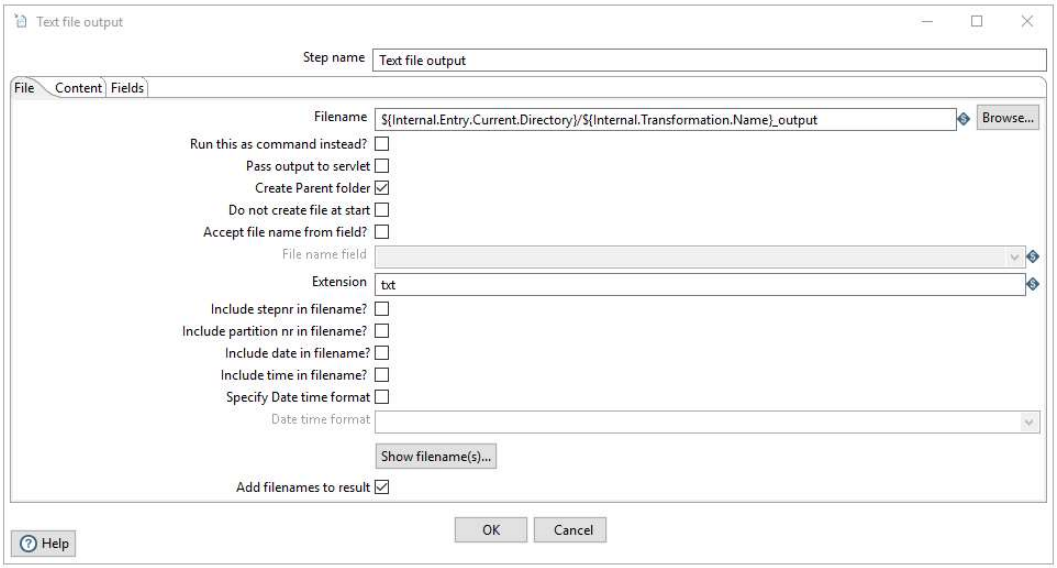


Tab	#	Columns and Parameters
Meta tab	**1**	**Name:** i - **Type:** Integer - **Set empty string:** N
	2	**Name:** s - **Type:** String - **Set empty string:** N
Data tab	**1**	**i:** 1 - **s:** a
	2	**i:** 2 - **s:** b
	3	**i:** 3 - **s:** c

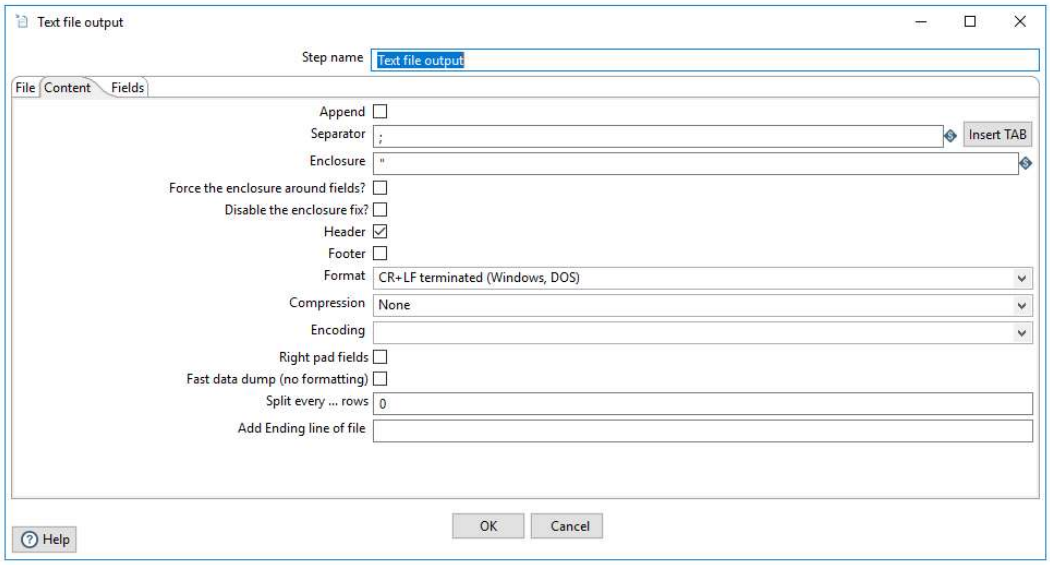
4. Drag the **Transform → Select values** step to the design surface and connect the two steps with a hop, but do not configure it.

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 **Select values** step, and configure the **File** tab as shown:



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

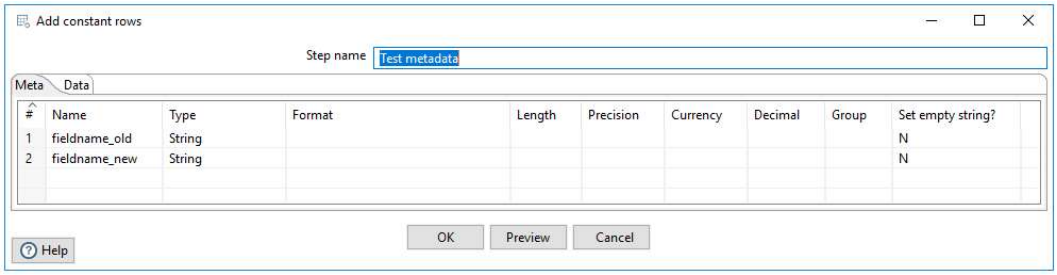


7. Do not configure the **Fields** tab, and save the template transformation.

Use the Template to Create a Transformation

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

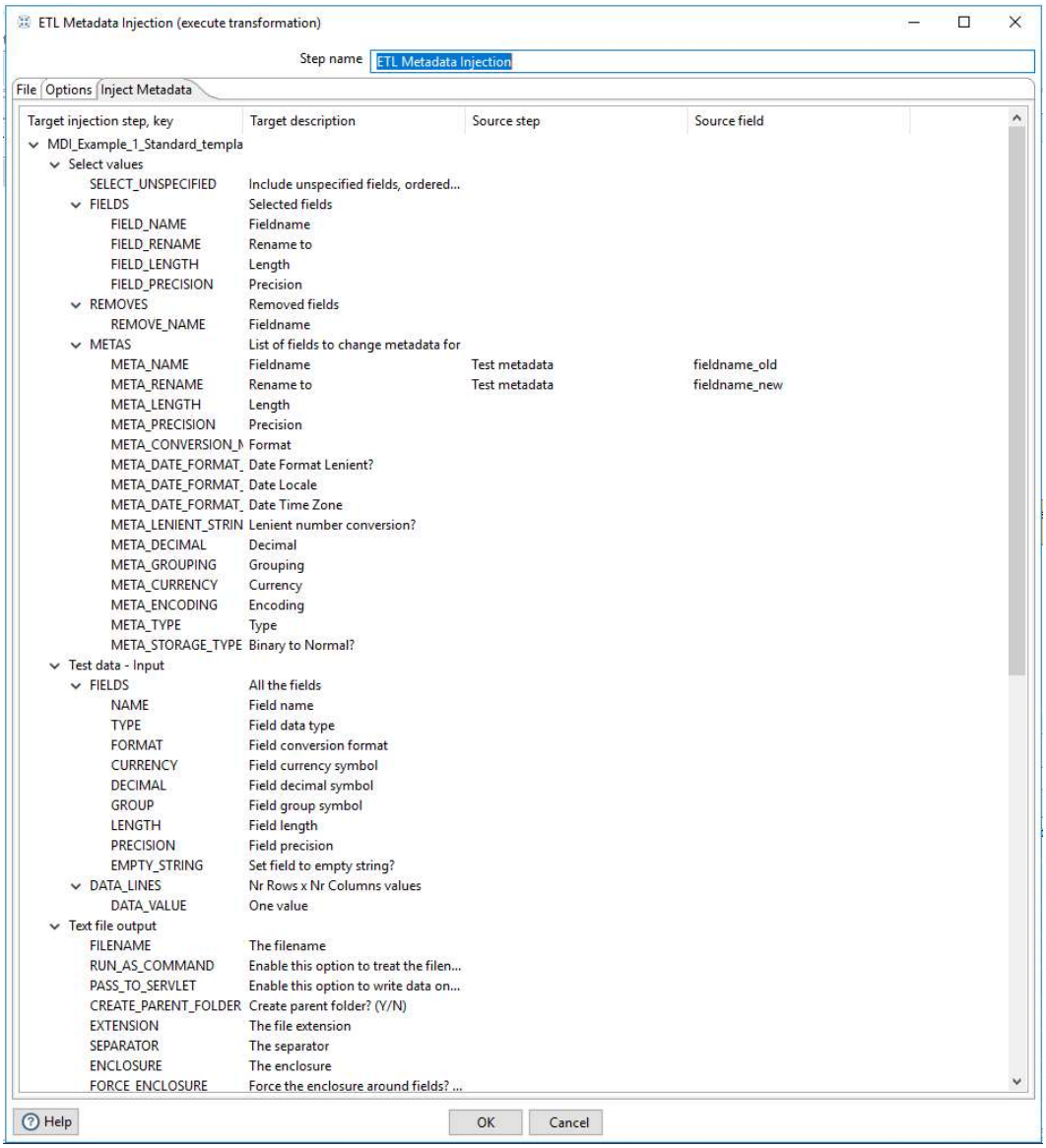
1. Create a new transformation called `MDI_Example_1_Standard`.
2. Drag **Input-→ Data grid** and **Flow → ETL metadata injection** steps to the design surface and connect them.
3. Rename the **Add Constant Row** to **Test Metadata**.
4. Next, open the **Test Metadata** step and configure the **Meta** tab:



5. Configure the **Data** tab:

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:



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)**:

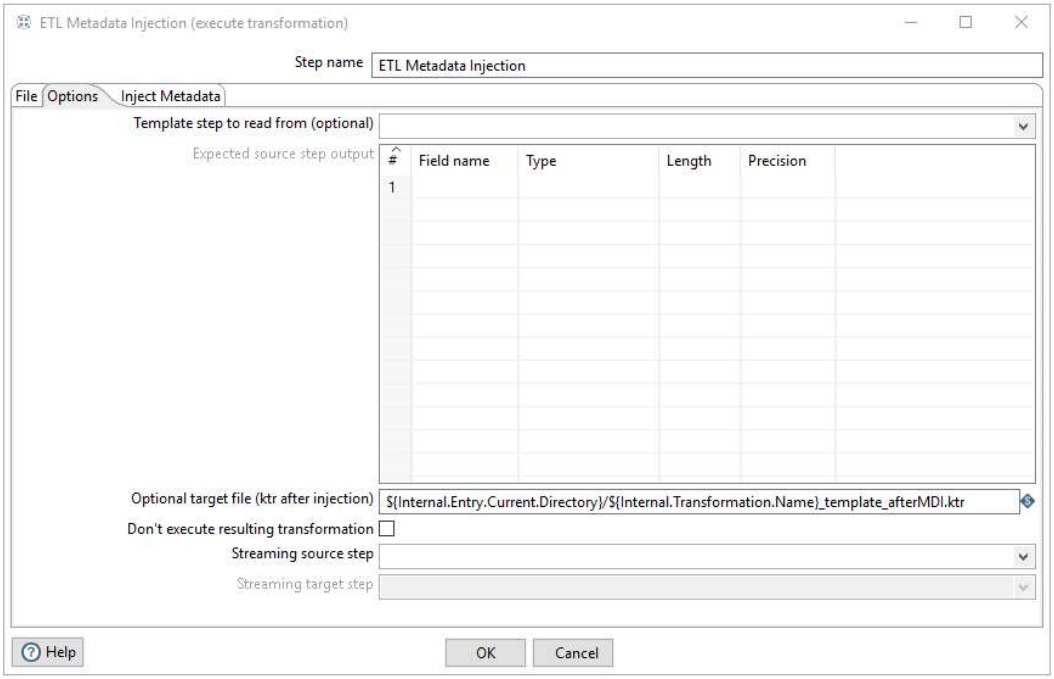


Figure 4: Options Tab in Metadata Injection Step

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:

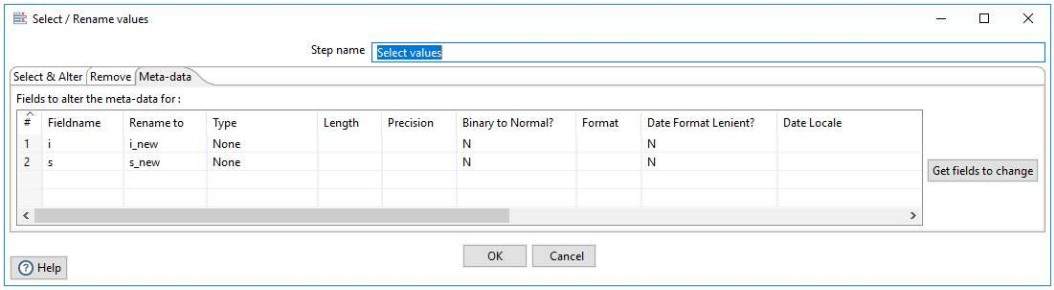


Figure 5: Meta-data Tab

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:

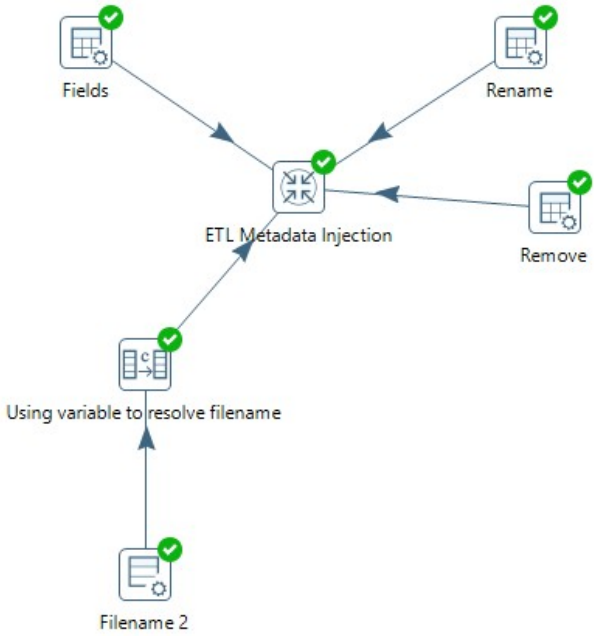


Figure 6: Full Metadata Injection

This pattern can be examined or built upon from in our `samples` → `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:

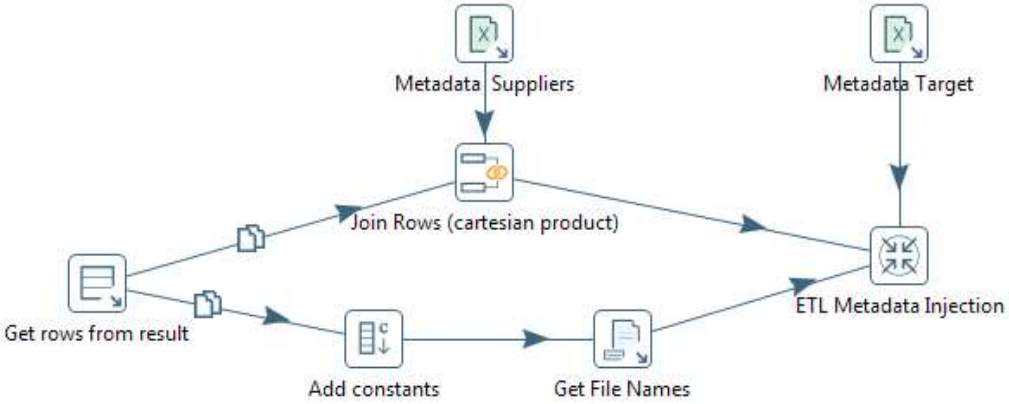


Figure 7: Complex Metadata Injection

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 Data Lake Operations.

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 input sources in the ETL Metadata Injection 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:

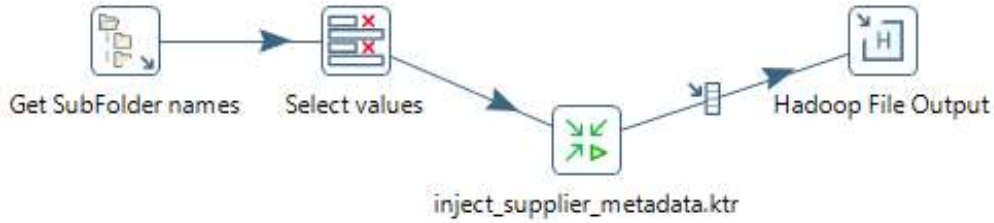


Figure 8: Example of a Process All Suppliers Transformation

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, 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.*

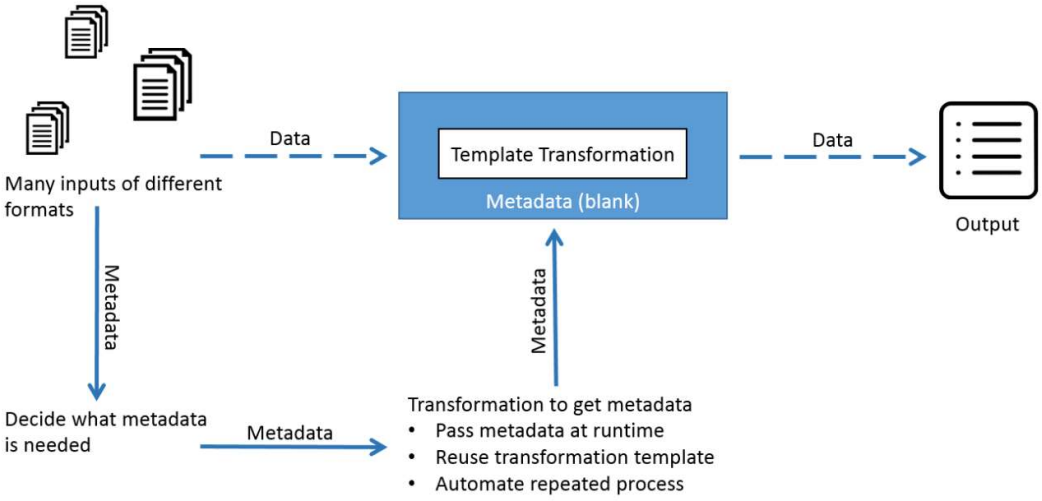


Figure 9: Using the ETL Metadata Injection Step

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:

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):

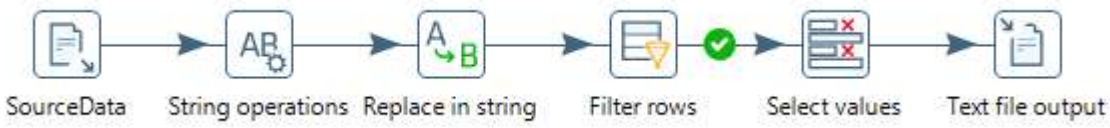
#	Field Name	Field Type	Description
1	`Key`	String	Group name, used to pull different groupings of rules for different runs (for example: all rules for `windows`, `linux`, `ruleset01`, `ruleset02`, `emailspamrules`)
2	`Field`	String	Field in transformation to perform search on, replace with weighted 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:

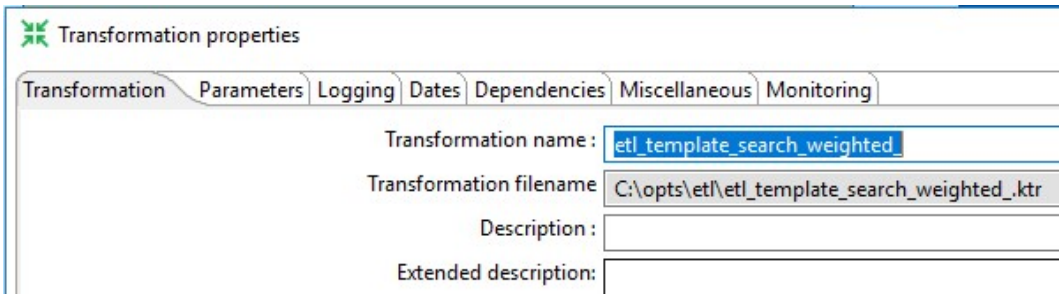
```
Key|Field|UseRegExYN|SearchRegEx|ReplaceWithValue|SetEmptyYN|WholeWordYN|Ca
seSensitiveYN
windows|threat|Y|.*powershell\.exe.*File.*|10|N|Y|N
windows|threat|Y|.*powershell\.exe.*IEX.*|10|N|Y|N
windows|threat|Y|.*net user /add.*|20|N|Y|N
linux|threat|Y|.*sudo su -.*|05|N|Y|N
linux|threat|Y|.*rm -f -R.*|25|N|Y|N
linux|threat|Y|.*rm -R -f.*|25|N|Y|N
```

Step 2: Building the ETL Template Transformation

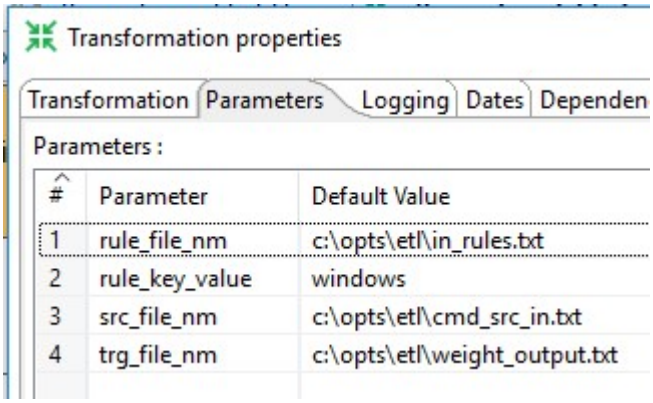
After you create the sample source file, you'll need to build a template for ETL transformations. This transformation will lay out steps for data processing rules:



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



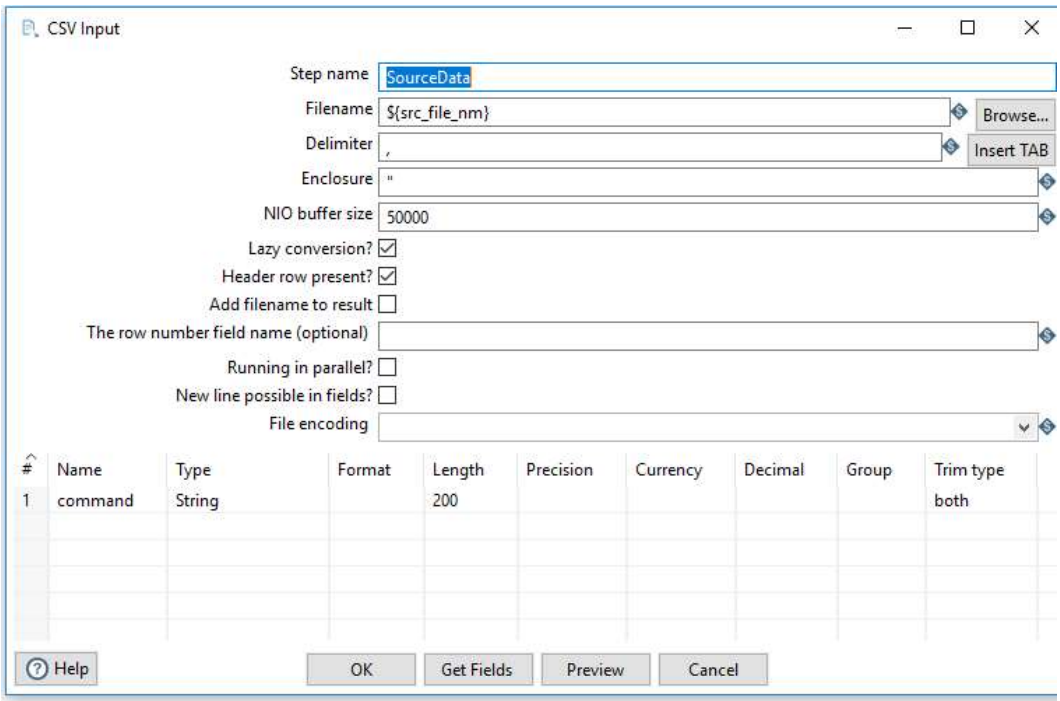
2. Set up the parameters for the input/output text files (this could easily be changed to be RDBMS Source/Target locations):



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

Adding the CSV Input Step

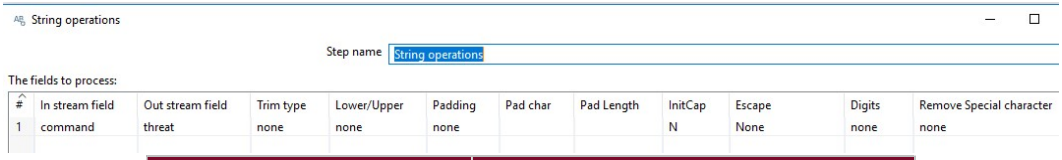
Use these parameters to fill out the **CSV Input** step:



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

Adding the String Operations Step

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



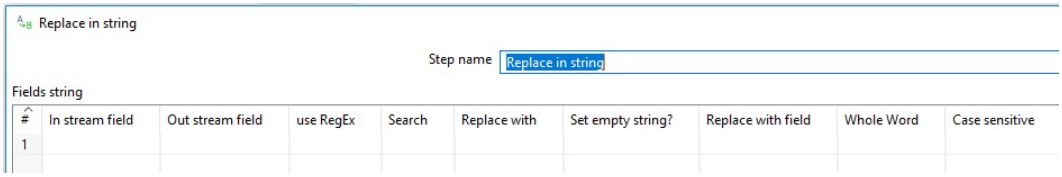
Field name	Parameters
In stream field	`command`
Out stream field	`threat`
Trim type	`none`
Lower/Upper	`none`
Padding	`none`
InitCap	`N`

Field name	Parameters
Escape	None
Digits	none
Remove Special character	none

Adding the Replace in String Step

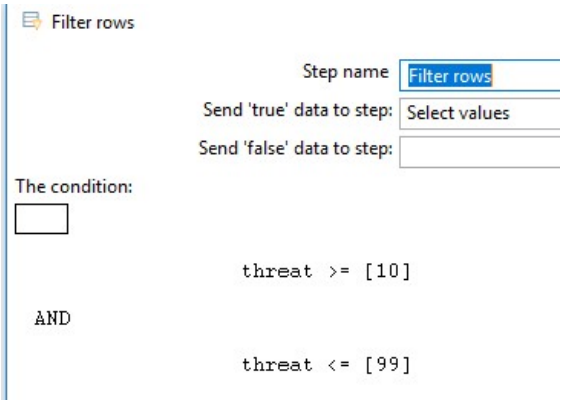
While setting up a **Replace in string** step, you will be leaving the parameters blank.

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



Adding the Filter Rows Step

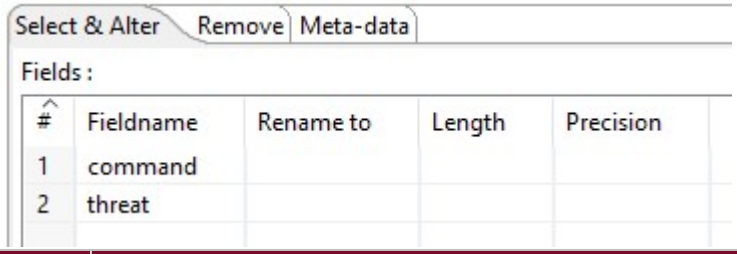
After you add the **Filter rows** step, you will need to add two conditions:



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

Adding the Select Values Step

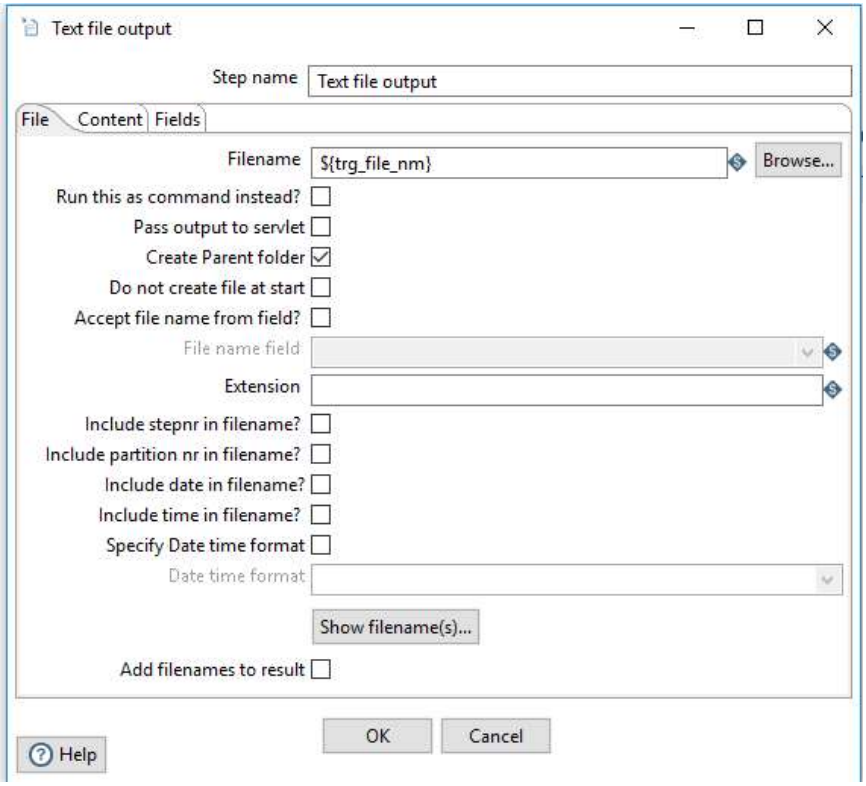
Use the following parameters, which will filter fields down to the two we want to keep:



#	Fieldname
1	`command`
2	`threat`

Adding the Text File Output Step

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

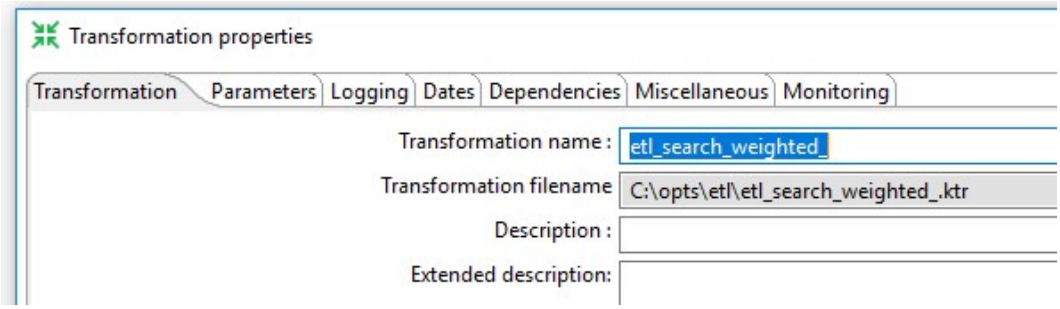


Field name	Parameters
Filename	`${trg_file_nm}`
Create Parent Folder	Checked

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`:



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

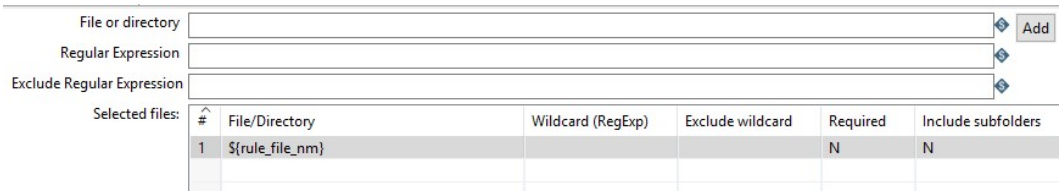
#	Parameter	Default Value
1	**rule_file_nm**	`c:\opts\etl\in_rules.txt`
2	**rule_key_value**	`windows`
3	**src_file_nm**	`c:\opts\etl\cmd_src_in.txt`
4	**trg_file_nm**	`c:\opts\etl\weight_output.txt`

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.

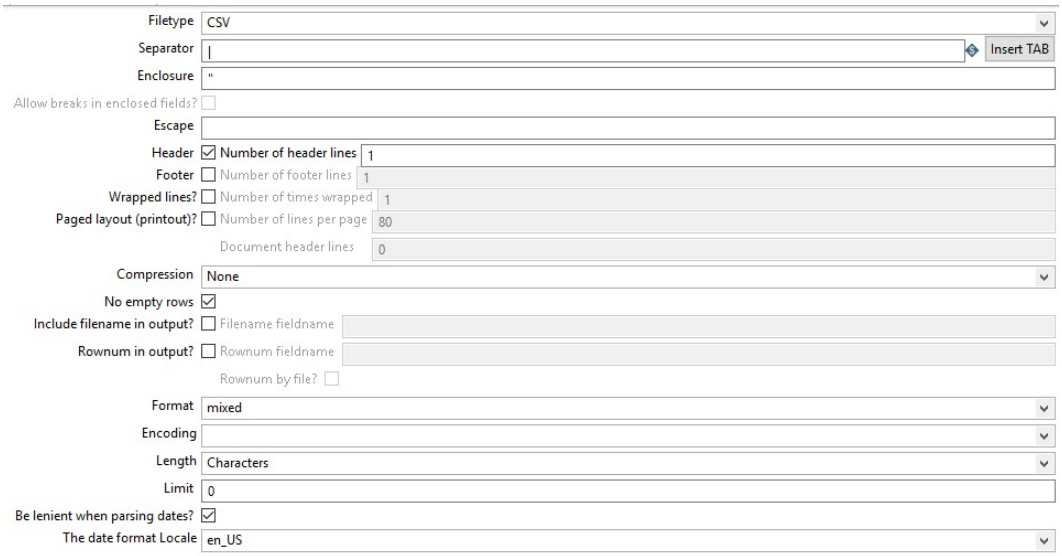
1. Start with the **File** tab to enter these parameters:



Field name	Parameters
File/Directory	`${rule_file_nm}`

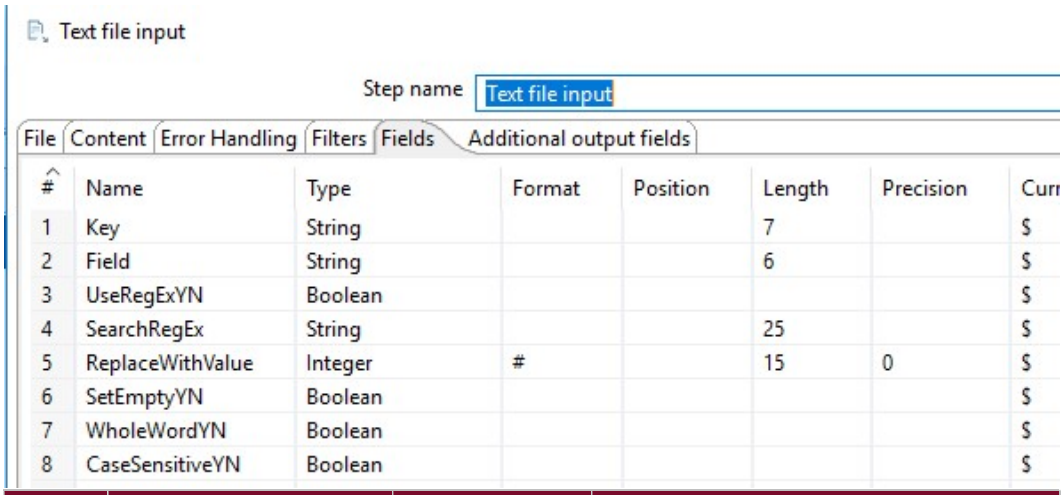
Field name	Parameters
Required	N
Include subfolders	N

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



Field name	Parameters
Filetype	CSV
Separator	|
Enclosure	"
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

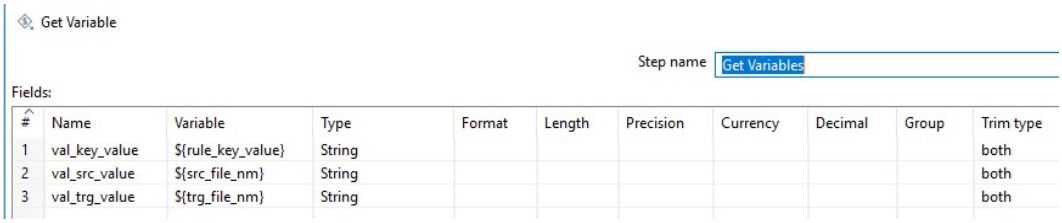
3. Add the following eight fields on the **Fields** tab:



#	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*

Adding the Get Variables Step

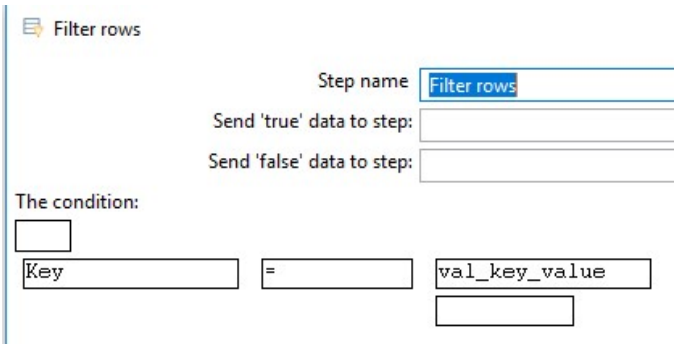
Add the **Get variables** step with the following parameters:



#	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`

Adding the Filter Rows Step

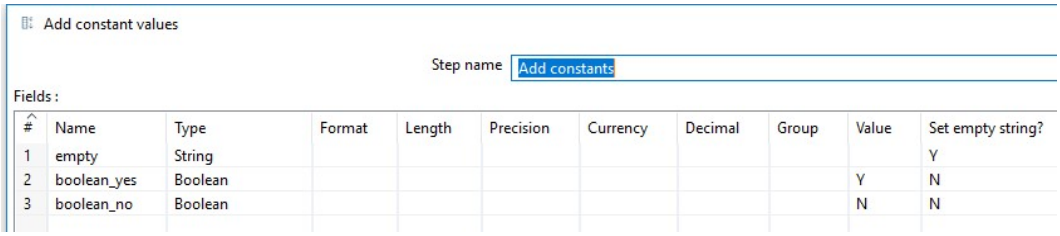
Create a **Filter rows** step and use these parameters:



Field name	Parameter
The condition:	`Key = val_key_value`

Adding the Add Constants Step

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



#	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`

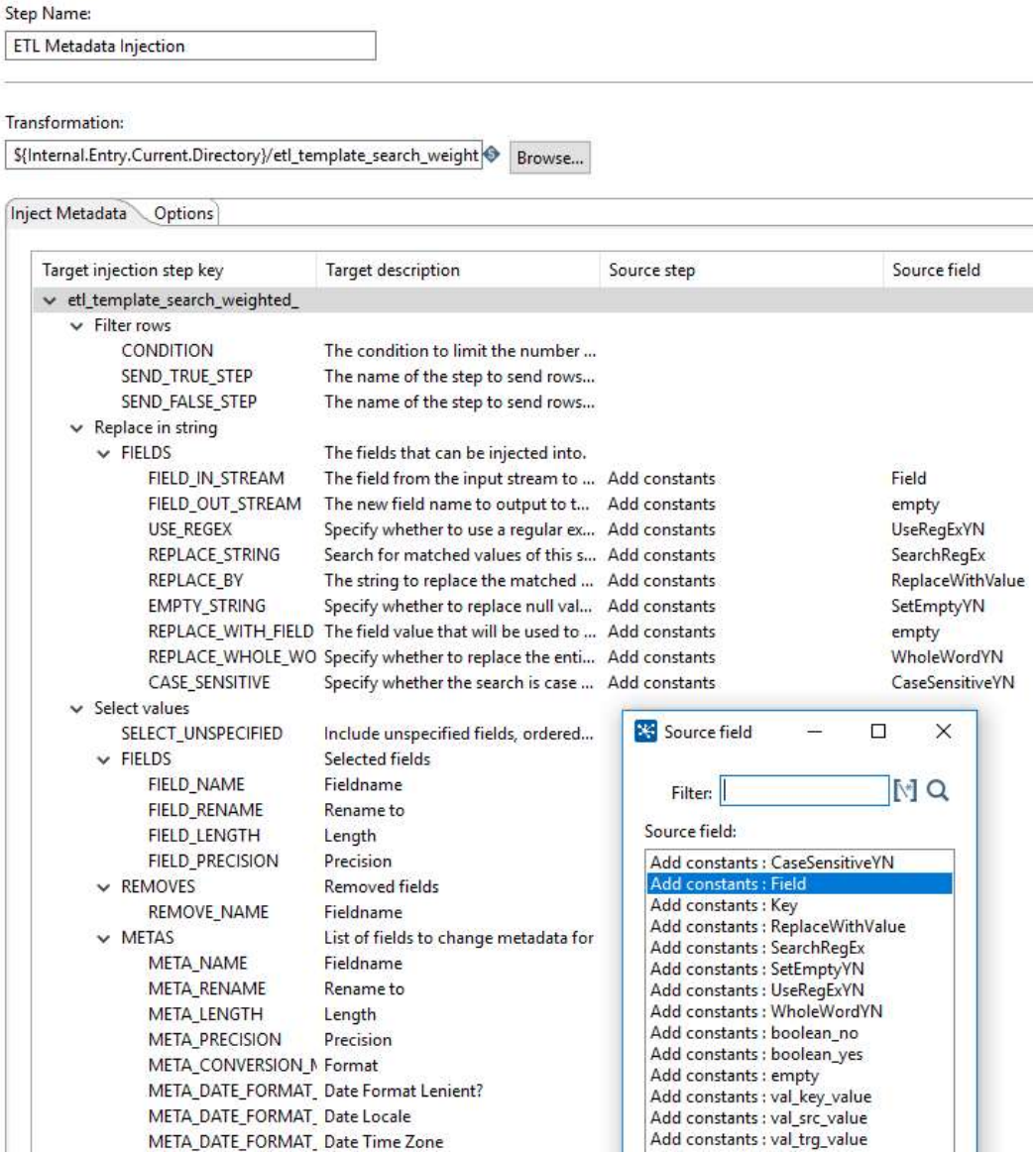
Adding the ETL Metadata Injection Step

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.

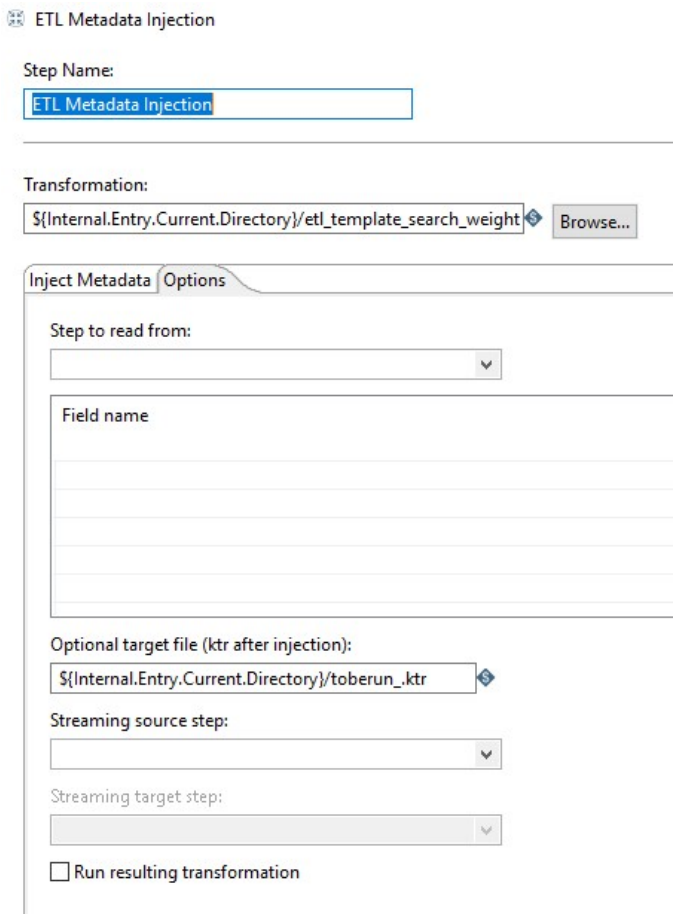
1. Map these fields to the following:

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



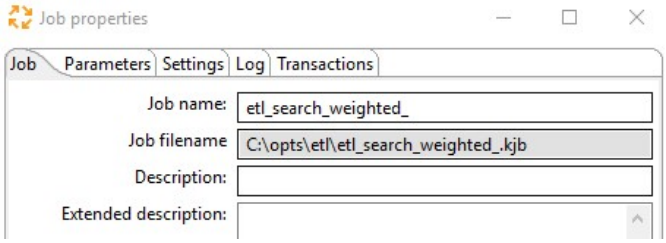
2. On the **Options** tab, set the following parameters:
    a. Transformation: `${Internal.Entry.Current.Directory}/etl_template_search_weighted_.ktr`
    b. Fill in the **Optional target file (ktr after injection)** field with the output target file for the runnable ETL transformation.
    c. Uncheck **Run resulting transformation**, to prevent the ETL transformation from being executed after you build it.



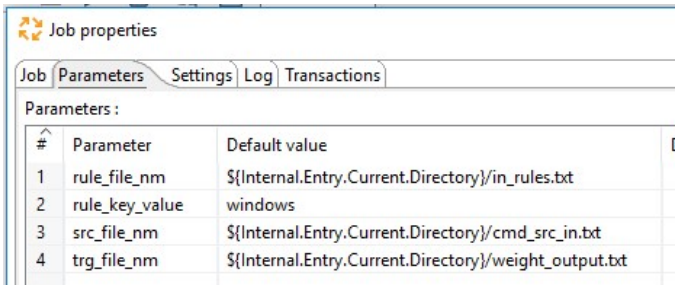
Step 4: Building the Job

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**:



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:

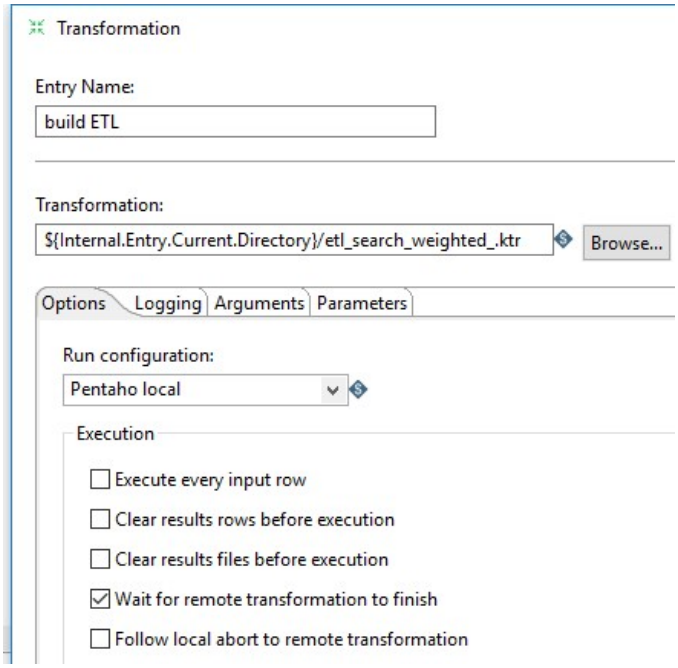


3. Select and connect these items for the job. We will rename the transformations in a moment:

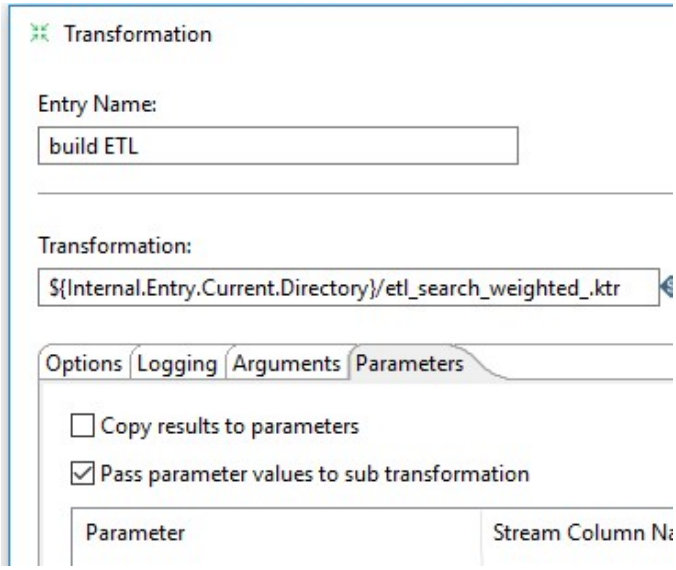
Step 5: Renaming the First Transformation

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**.



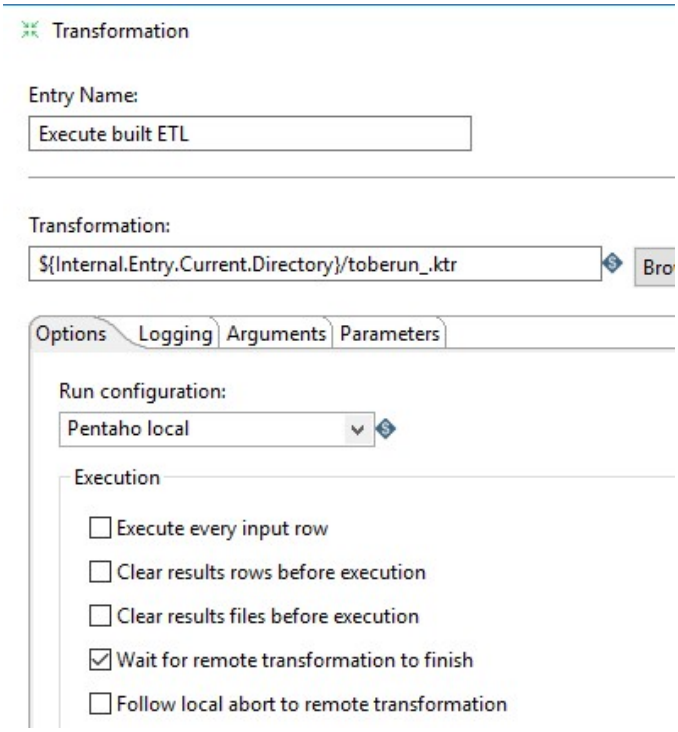
4. On the **Parameters** tab, check **Pass parameter values to sub transformation** to make sure the parameters flow through properly during execution:



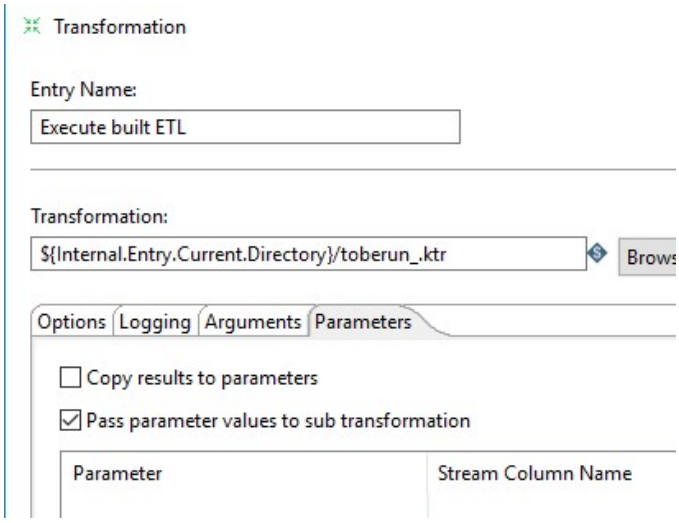
Step 6: Renaming the Second Transformation

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**.



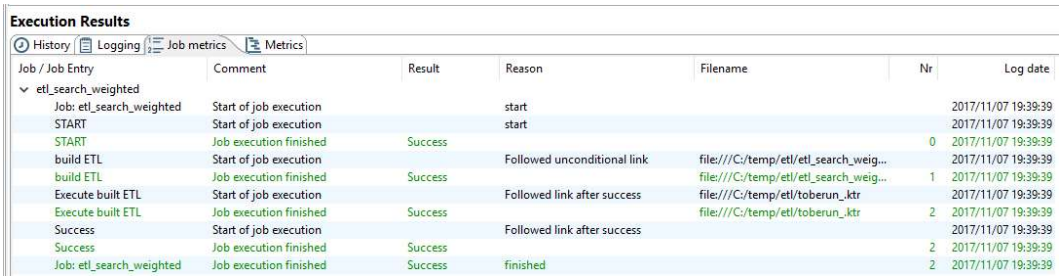
4. On the **Parameters** tab, check **Pass parameter values to sub transformation**, so the parameters will flow through properly during execution.



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:



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

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:

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