Pentaho Data Integration Fundamentals (DI-1000)

Table of Contents

[PDI (Pentaho Data Integration) BASICS 1](#_Toc50028854)

[Platform Components 1](#_Toc50028855)

[Two main types of objects 1](#_Toc50028856)

[Navigation in Spoon 1](#_Toc50028857)

[TRANSFORMATIONS 2](#_Toc50028858)

[Preview Steps 2](#_Toc50028859)

[Run a Transformation 2](#_Toc50028860)

[Transformation Best Practices 3](#_Toc50028861)

[Transformation Canvas 3](#_Toc50028862)

[Data Movement 4](#_Toc50028863)

[Step Copies (multi-threading and parallel processing) 4](#_Toc50028864)

[DATABASE CONNECTIONS 4](#_Toc50028865)

[Database Drivers 4](#_Toc50028866)

[Database Connections (basic info) 5](#_Toc50028867)

[Creating Database Connections 5](#_Toc50028868)

[Database Explorer 5](#_Toc50028869)

[PDI STEPS 5](#_Toc50028870)

[Table Input Step 5](#_Toc50028871)

[CSV File Input Step 6](#_Toc50028872)

[Fixed File Input Step 7](#_Toc50028873)

[Text File Input 7](#_Toc50028874)

[Microsoft Excel Input 7](#_Toc50028875)

[Get Data from XML Step 8](#_Toc50028876)

[JSON Input Step 8](#_Toc50028877)

[Generate Rows Step 8](#_Toc50028878)

[Data Grid Step 9](#_Toc50028879)

[Sort Rows Step 9](#_Toc50028880)

[Get System Info Step 9](#_Toc50028881)

[String Operations Step 9](#_Toc50028882)

[Calculator Step 10](#_Toc50028883)

[Group By Step 10](#_Toc50028884)

[Number Range Step 10](#_Toc50028885)

[Select Values Step 11](#_Toc50028886)

[Filter Rows Step 11](#_Toc50028887)

[Switch / Case Step 11](#_Toc50028888)

[RegEx Evaluation Step 12](#_Toc50028889)

[Add Sequence Step 12](#_Toc50028890)

[Merge Join Step 12](#_Toc50028891)

[Merge Rows (Diff) Step 12](#_Toc50028892)

[Synchronize After Merge Step 13](#_Toc50028893)

[Dummy (do nothing) 13](#_Toc50028894)

[Database Lookup Step 13](#_Toc50028895)

[Stream Lookup Step 14](#_Toc50028896)

[Formula Step 14](#_Toc50028897)

[Modified Java Script Value Step 15](#_Toc50028898)

[User Defined Java Expression Step 15](#_Toc50028899)

[User Defined Java Class Step 15](#_Toc50028900)

[Table Output Step 15](#_Toc50028901)

[Insert / Update Step 16](#_Toc50028902)

[Dimension Lookup / Update Step 16](#_Toc50028903)

[Text File Output Step 17](#_Toc50028904)

[Microsoft Excel Output Step 17](#_Toc50028905)

[Microsoft Excel Writer Step 17](#_Toc50028906)

[XML Output Step 18](#_Toc50028907)

[MongoDB Output Step 19](#_Toc50028908)

[METADATA INJECTION 19](#_Toc50028909)

[Basics Details 19](#_Toc50028910)

[ETL Metadata Injection Step 19](#_Toc50028911)

[Inspect Data from a Step 20](#_Toc50028912)

[Using Data Collected from an Input Step to Set Configuration of Another Step 20](#_Toc50028913)

[Copying/Pasting Steps 20](#_Toc50028914)

[JOBS 20](#_Toc50028915)

[Job Use 20](#_Toc50028916)

[Jobs Best Practices 20](#_Toc50028917)

[Job Orchestration 21](#_Toc50028918)

[Job Hops 21](#_Toc50028919)

[JOB ENTRIES 21](#_Toc50028920)

[Start Job Entry 21](#_Toc50028921)

[Abort Job Entry 22](#_Toc50028922)

[Job Job Entry 22](#_Toc50028923)

[Mail Job Entry 22](#_Toc50028924)

[Transformation Job Entry 22](#_Toc50028925)

[Convert file between Windows and Unix Job Entry 22](#_Toc50028926)

[Set Variables Job Entry 22](#_Toc50028927)

[File Exists Job Entry 22](#_Toc50028928)

[SQL Job Entry 22](#_Toc50028929)

[Success Job Entry 22](#_Toc50028930)

[HTTP Job Entry 22](#_Toc50028931)

[Write to Lob Job Entry 22](#_Toc50028932)

[Wait for File Job Entry 23](#_Toc50028933)

[PENTAHO REPOSITORY 23](#_Toc50028934)

[Basics 23](#_Toc50028935)

[Repository Features 23](#_Toc50028936)

[Connecting to a Repository 23](#_Toc50028937)

[Repository Explorer 24](#_Toc50028938)

[Importing/Exporting Repository Objects 25](#_Toc50028939)

[WORKFLOWS 27](#_Toc50028940)

[Transformations 27](#_Toc50028941)

[LOGGING 27](#_Toc50028942)

[Log File Location 27](#_Toc50028943)

[Logging Levels 28](#_Toc50028944)

[Logging Tab (Execution Results Panel) 28](#_Toc50028945)

[Logging Execution Metrics 28](#_Toc50028946)

[DEBUGGING 29](#_Toc50028947)

[Pentaho Local 29](#_Toc50028948)

[Pentaho Server 29](#_Toc50028949)

[DATA TYPES 29](#_Toc50028950)

[Important Notes 29](#_Toc50028951)

[Data Types 30](#_Toc50028952)

[Convert Data Types 30](#_Toc50028953)

[Format (when configuring fields) 30](#_Toc50028954)

[CONFIGURING FIELDS 31](#_Toc50028955)

[Format 31](#_Toc50028956)

[Length 31](#_Toc50028957)

[Precision 31](#_Toc50028958)

[VARIABLES AND PARAMETERS 31](#_Toc50028959)

[Syntax for Referencing a Variable/Parameter 31](#_Toc50028960)

[Inspect Parameters/Variables Loaded into Memory 31](#_Toc50028961)

[Variables 32](#_Toc50028962)

[Parameters 32](#_Toc50028963)

[PDI’s HOME DIRECTORY 32](#_Toc50028964)

[Configuration Files 32](#_Toc50028965)

[SCHEDULING 33](#_Toc50028966)

[Scheduling 33](#_Toc50028967)

[Scheduling via Spoon (built-in scheduler) 33](#_Toc50028968)

[Scheduler Perspective 33](#_Toc50028969)

## PDI (Pentaho Data Integration) BASICS

### Platform Components

* Pentaho Server
  + Dedicated server for remote execution
  + Runs on Apache Tomcat by default
* Spoon
  + GUI for developing, testing, debugging, scheduling, and monitoring jobs/transformations
* Kitchen, Pan
  + CLI driven job/transformation runners for OS-level scripting and scheduling
* Carte
  + Light-weight HTTP server for remote execution and parallel execution of jobs on a scale-out cluster
* Pentaho Repository
  + Repo provides ability to store and manage jobs in an Apache Jackrabbit database
  + Has versioning capabilities

### Two main types of objects

* Transformations
  + Logical tasks (steps) connected using arrows (hops)
  + ETL
  + Rows of data flowing through are the stream
  + Steps run in parallel
    - Data is flowing and going through each step in a stream
    - Still ordered, but steps do not complete before data heads to the next step
  + Transformation files are XML files with .ktr extension
* Jobs
  + Workflow models for coordinating resources, execution, and dependencies of ETL activities
  + Job entries and hops
  + No data flowing through it, defines a process flow
  + Used to orchestrate transformations
  + Can run transformations or other jobs
  + Job files are saved as XML files with .kjb extension

### Navigation in Spoon

* Data Integration Perspective
  + Designing/testing transformations/jobs
  + Multi-tabbed UI
  + Each transformation/job has its own tab
* Schedule Perspective
  + Used to schedule transformations/jobs
* Welcome Screen
  + **Help** -> **Welcome Screen**
  + Links to:
    - Documentation
    - Tutorials
    - Marketplace
    - Forums
    - Blogs/Books
* Samples
  + <spoon\_install\_path>\data\_integration\samples
  + usually C:\.....\Pentaho\design-tools\data-integration\samples
* Transformations
  + Left Navigation Pane
    - View
    - Design
      * Steps search box (can type step names here)
      * Hover over a step to see a short description
      * Add a step by dragging to canvas or double clicking on it
  + Preview a step
    - Click on a step (hovering provides other options)
  + Execute a transformation
    - Play button on the file’s toolbar -> Run
  + View database tables
    - Left Navigation Pane
    - Database connections -> open the database -> navigate the tree to the table of interest -> right click
  + Execution Results (bottom of window)
    - Logging
    - Execution History
    - Step Metrics
      * Data for step execution separated by step
    - Performance Graph
    - Metrics
    - Preview Data

## TRANSFORMATIONS

### Preview Steps

* Previewing actually executes the entire transformation, AND it displays results
* Can execute from a step’s configuration diaglogue box
* Can **right-click a step** -> **Preview**
  + Executes the entire transformation
  + Allows you to select multiple steps to view outputs and toggle between them (Quick Launch to view)
* Click the eyeball icon in the Transformation’s toolbar
  + Same as right-click a step
  + Executes entire transformation, let’s you pick steps to view

### Run a Transformation

* Methods
  + Click **Run** (play) icon in toolbar
  + **Action** -> **Run**
  + **F9**
* Run Options
  + Run configuration
    - Pentaho Local (local machine)
      * use for dev
    - Pentaho Server
    - Spark Engine
  + Options
    - Log level
      * sets the amount of detail included in the log
    - Clear log before running
    - Enable safe mode
    - Gather performance metrics
  + Parameters/Values
    - allows editing of these before running for testing different values if needed

### Transformation Best Practices

* Transformation naming conventions
  + Transformation filename
    - verb\_short\_description
  + Transformation steps
    - CSV input
      * csvi: file\_name
    - Add constants
      * ac: attribute\_or\_subject\_name\_cons
    - Add sequence
      * aseq: attribute\_added
    - Database lookup
      * dblkp: attribute\_looked\_up
    - Filter rows
      * fr: attribute\_filtered
    - Generate rows
      * gr: name
    - Get system info
      * gsi: attribute\_gotten
    - Table output
      * to: table\_name
    - Text file output
      * tfo: filename.txt
* Build transformations inside out
  + i.e. pieces first, then assemble
  + Build transformations before creating jobs that call them
  + Create sub-jobs before creating jobs that call them
* Preview steps/transformations before adding/connecting them
* Add notes (**right-click canvas** -> **New Note**) to help with documentation
  + Easily viewed/displayed directly on the canvas
* Add descriptions (documentation) to steps
  + **Right-click a step** -> **Description**
  + No indication that a step has a description from the canvas
  + Use **ctrl + f** to do meta data search the descriptions of all steps in all open transformations/jobs
* Add Transformation Properties
  + Double click an empty area of the canvas
  + Add documentation

### Transformation Canvas

* Hop Symbols
  + Green check mark
    - anything that meets criteria (such as filter rows) follows this path
  + Red x
    - anything not meeting criteria (such as filter rows) follows this path
  + Blue arrows (active hops)
  + Gray arrows (inactive hops)
* **Right-click canvas** for options

### Data Movement

* Editing Data Movement Type
  + Set when creating a second hop from a source step
  + **Right-click a step** -> **Data Movement**, to set the movement out of this step
* Data Movement Types
  + Distribute (Round Robin)
    - Default data movement type
    - No symbol on the hop
    - Sends outgoing rows distributed evenly to destination steps (if multiple hops)
    - No destination steps receive the same rows
    - Impossible to predict which rows will go to which step
  + Copy (Copy Data to Next Steps)
    - Hops have two pieces of paper on them (copies)
    - Sends all rows to all destination steps
    - Every destination receives the same rows (all of them)
    - If 7000 rows are leaving the step with a copy hop, and there are 3 such hops, 21,000 rows will enter the stream
  + Custom Row Distribution (Load Balance)
    - Background info
      * Transformation properties (double click transformation canvas) Miscellaneous tab
        + Nr of rows in rowset

basically a buffer

the number of rows a step can hold onto as it processes the current row

* + - Hopes have a scale on them
    - Distribute outgoing rows to the destination step that is the least busy (Nr of rows in rowset property)
    - No destination steps receive the same rows

### Step Copies (multi-threading and parallel processing)

* Exactly duplicates a step and its configuration
  + Be cautious of unexpected consequences
  + Not good for reading data from csv, tables, etc. (duplicates data)
    - For csv files always check **Run in Parallel** if using step copies to avoid data duplication
    - **NOTE:** order of rows not guaranteed, if important, use a sort step after reading in the file
* **Right-click a step** -> **Change Number of Copies to Start…**
* Max performance by keeping the number of copies 3-4 times the number of processing CPU cores
  + test under realistic load conditions
* Shows a x3 by the step to indicate 3 copies of that step will run

## DATABASE CONNECTIONS

### Database Drivers

* Must be installed
  + JDBC Driver Reference at help.pentaho.com for JDBC driver download URL’s and other details
  + Pentaho Server
    - <pentaho\_install\_path>\server\pentaho-server\tomcat\lib
    - server must be restarted to recognize
  + Spoon dev machines
    - <pentaho\_install\_path>\design-tools\data-integration\lib
    - must restart Spoon to recognize
* JDBC preferred to ODBC (only use ODBC when that’s all there is)
* Do not use JNDI connections

### Database Connections (basic info)

* Pentaho repository (preferred methinks)
  + can be created here and made available for all jobs/transformations stored in the repository
* Local only
  + stored within the .kjb and .ktr files
* Sharing database connections (not using the repository)
  + Open transformation/job with the connection
  + View tab of transformation/job (left panel) -> **right-click specific Database Connection** -> **Share**
  + Save the transformation/job
* Parameterizing Database Connections (best practice)
  + Use global variables for all of the ‘Settings’ in the General tab of a database connection
  + refer to PDI documentation for parameterizing passwords
* Created connections can be found in the View Tab (left panel) under Database connections
  + If a connection does not appear, it is probably not Shared

### Creating Database Connections

* View Tab (left panel) -> right click Database connections -> New
* Add a descriptive **Connection Name** that includes the name of the database
* Set the **Connection Type**
* Choose the correct **Access**
* Enter all of the **Settings**
* Click **Test** to ensure the connection is working then **OK** if all is well
* Right click an existing connection and click **Share** to make this connection available to other transformations

### Database Explorer

* **Right click an existing Database connection** (View Tab) -> **Explore**
* Tables
  + Shows a list of all the tables available
  + **Can right-click a table** to perform actions
    - DDL only shows the SQL statement that would create the specific table
    - Actions dependent on user’s permission levels
    - View SQL starts with SELECT \* FROM table\_name
      * Can edit this and execute SQL statements here
* Can access by going to **Tools** (menu bar) -> **Database** -> **Explore**
  + choose the db to explore

## PDI STEPS

**Input Steps:**

### Table Input Step

* ti
* Choose an existing Connection or create a New… connection
  + Best practice to create database connections using the method in **Creating Database Connections** and share if going to reuse connections
* SQL
  + Can type in SQL, copy/paste SQL, or click **Get SQL select statement…** to let Spoon help (opens Database Explorer dialogue) -> click the table of interest, then click OK if using this method
  + Using variables in SQL statements
    - ${VAR\_NAME} syntax
      * must check the Replace variables in script? box for this to work
      * see VARIABLES AND PARAMETERS section for managing variables/parameters
    - ? method
      * do not need to check Replace variables in script?
      * cannot be used in place of a table name
      * used to accept values from previous steps
        + each row of input should specify the variables to use in the correct order
        + the field names for the data do not matter
      * put a ? anywhere you are reading a value from a previous step
      * the number of inputs to the Table Input step must exactly match the number of ? in the SQL query
      * Insert data from step selectbox
        + choose the step where you need to get the inputs
      * checking Execute for each row? will send the query to the database for every input row
        + if 1000 rows are input to the Table Input step, 1000 queries will be sent to the database
        + no need to check if only one row of values are being input (i.e. the input values for the variables do not change)
      * to preview the data once configured, you must close the dialogue box, right click the step, then click preview
* **Preview** the data to ensure you’re getting what you expect

### CSV File Input Step

* csvi
* Reads only one file per step
* Can read delimited files (doesn’t have to be commas)
* Delimiter can accept hex chars using $[value]
* Enclosure can be used to define char(s) that can enclose fields (delimiter chars appearing between enclosure chars will not be identified as a delimiter)
* Lazy conversion?
  + checking (usually best practice) will avoid doing any unnecessary data type conversions
* Running in parallel?
  + check this if using multiple steps and wanting to run in parallel
  + not supported if the file contains line breaks or carriage returns in the data
* Get Fields (button)
  + Spoon will try to find fields for you (check delimiter if not getting expected result)
  + Enter 0 for sample size if you want to sample the entire file (can cause issues with large files)
    - Use a sample size that is representative of the data
  + Scan results will give some helpful info
  + Spoon will automatically populate the fields now based on this, EXAMINE for ERRORS!
* Can set Trim type to go ahead and trim spaces from the field
* **Important Notes**
  + Must always read in all fields
  + Format should be the format found in the file, not the format you want later
* Preview (button)
  + Lets you preview the output of the step

### Fixed File Input Step

* fxdi
* Fields are specified by column widths, padding, and alignment
* Each record has the exact same width for a given fields
  + You need the record width to configure the step
* Enter the linewidth in bytes (chars)
* Line feeds present (is each ‘row’ of data on a separate line?)
* Get Fields (button)
  + More helpful than for the csvi step
  + Let’s you configure where column breaks are (click the data at the appropriate char locations)
    - Set these areas so there are trailing spaces rather than leading spaces (arrows at the beginning of each new column/field)
    - Click an arrow to remove it
    - Be aware of fields with no padding/trailing spaces
  + Clicking Guess All on the next wizard might help, or might not
* Probably want to double check the configuration and Preview

### Text File Input

* tfi
* Shares a UI with many other file input steps and can be configured to read multiple files
  + Get Data From XML
  + Get File Names
  + JSON Input
  + LDIF Input
  + Microsoft Access Input
  + Microsoft Excel Input
  + Property Input
  + YAML Input
* Configure the **Selected Files** section to ensure the files to read are all included
  + Options for adding files
    - File or Directory
      * Browse… -> choose file -> Add
      * can simply use a directory
    - Using Regex
      * examples
        + .\*\.txt is everything with .txt extension
        + text.\*\.txt is everything that starts with ‘text’ and ends ‘.txt’
      * Enter only the path in the File or Directory field
      * Use the Regular Expression box
      * Use the optional Exclude Regular Expression box
      * Then click Add (only adds one row to Selected Files grid that specifies this configuration)
    - Click Show filenames(s)…
      * Will display all paths/filenames that would be read if run under the current configuration

### Microsoft Excel Input

* msxlsi
* Works best for small data sets
* Files Tab
  + Configure using filename or directory or regex like with Text File Input above
* Sheets Tab
  + Get Sheetnames…
* Error Handling Tab
  + Can configure to enforce strict types, ignore errors, and skip error lines
  + Can store failing rows in a new file
    - set path in Failing line numbers files directory
    - this file contains only a list of row numbers for rows that failed, not the actual data
* Fields Tab
  + Get fields from header row…
  + If using multiple files, must remove redundant field names
* Additional output fields Tab
  + Can add a limited selection of fields to the output data (meta data type stuff)

### Get Data from XML Step

* gdxml
* File Tab
  + Can read multiple files per step if they are all the same format
* Content Tab
  + Loop XPath
    - The item that you want to loop through to bring in data as rows
    - Get XPath Nodes is helpful here
* Fields Tab
  + Get fields is helpful

### JSON Input Step

* jsoni
* File Tab
  + Can read multiple files per step if they are all the same structure
  + Can read in json produced in a previous step
* Fields Tab
  + No Get fields button!
    - must manually configure
  + Name
    - can be whatever
  + Path
    - $..field\_name (I think that the two dots represent the level from which to get the data????)

**Data Generating Steps (usually for testing):**

### Generate Rows Step

* gr
* This step primarily used for testing purposes
  + Generate input to test transformations
* Outputs any specified number of identical (or empty) rows
* Default is empty rows (no fields or values)
* Can contain arbitrary number of static fields with values
* Configuration
  + Specify field names
  + Data types
  + Format info for data validation
  + Other details
  + Limit – number of rows to generate
  + Never stop generating rows (until the transformation is stopped)
  + Interval – time between generation of rows
  + field names for current row time and previous row time

### Data Grid Step

* dg
* Basically generates a table of data
  + Good for testing
* Meta Tab
  + you define field names and their properties
* Data Tab
  + you enter data in as many rows as you want

**Transformation Steps:**

### Sort Rows Step

* sr
* Sorting is memory intensive, so best practice to sort during input steps (i.e. Table Input using SQL) if possible
* Configuration
  + Sort directory
    - ensure a value here for temporary file storage in the case of out of memory during sorting
    - default location is standard temporary directory for the system
  + Sort size (rows in memory)
    - number of rows to store in memory before creating temporary files
    - more rows = fast sort time
    - more rows = higher chance of out of memory errors
  + Free memory threshold (in %)
    - percent of JVM free memory to trigger temporary file storage during sorting
    - will trigger temp file creation even if Sort size is not met
    - decreases chance of OutOfMemoryException but does not guarantee it won’t happen
  + Only pass unique rows? (verifies keys only)
    - If checked, only unique rows for the first sort field defined in the Fields grid are returned
  + Fields
    - creates the sort order
    - can specify ascending/descending for each field specified

### Get System Info Step

* gsi
* Create a row of system info
  + No data coming into this step
  + Configuring
    - Type will open a dialog that shows available system info
* Add system info fields to every row of data
  + Must have data coming into this step
  + Will add any fields configured to all rows of data

### String Operations Step

* so
* Before using
  + Useful to preview data in import steps to check formats
  + Click in a cell to view potential hidden trailing/leading spaces (entire contents highlighted)
* Step Configuration
  + In stream field
    - Will only accept input fields that are strings
  + Out stream field
    - Leave blank to replace the contents of In stream field or provide a name of a new field to create with the output of the transformation
  + Digits
    - get only digits or no digits

### Calculator Step

* cal
* New field
  + name of the new field to hold the calculated value
  + new calculated fields can be used in subsequent calculator rows
* Calculation
  + clicking here will open a dialogue where you can choose from a large yet limited selection of available calculations that can involve up to 3 fields
  + specify the field names in Field A, Field B, Field C (leave blank any not used)
* Value type, Length, Precision
  + specify the output parameters
  + if getting unexpected results, check input datatypes and see Note on datatypes below
* Note on datatypes
  + the calculator step can output a different datatype than the inputs, but calculations are performed on input data first
    - Example: getting percents as a decimal
    - If inputs are integers, and the calculation should produce a percent as a decimal (floating point numbers less than 1.0) any result will = 0 even if the output is Number
    - This occurs because PDI does the calculation on integers first, returning an integer (which is 0)
    - This 0 value is then converted to a floating point number (0.000 or to the specified Length/Precision)
    - **Bottom line is if calculations are providing unexpected results, check the input datatypes**

### Group By Step

* gb
* **NOTE:**
  + Incoming rows must be sorted by the field(s) by which the rows are grouped
    - either do this during an input step or add a Sort Rows Step prior to any Group By Steps
  + No formatting of aggregate values occurs during this step
    - if wanting to format the data, add a Select Values Step after Group By Steps to format
* Configuration
  + Inlude all rows?
    - check to include all rows that go into the step to be included in the output
      * can configure temp file storage if this is checked
  + The fields that make up the group
    - enter the field(s) that you want to group by here
  + Aggregates
    - any aggregate calculations to perform (unlimited number of these)
    - Name = name of the new field to create
    - Subject = field upon which to calculate
    - Type = aggregate operation to perform
    - Value = certain aggregations use a value in the calculation, you specify the value here

### Number Range Step

* nr
* Takes an input field, examines set ranges, and outputs to a new field based on ranges
  + like a switch case that acts on one field, but cases can only be specified ranges for the input field
* Choose the Input field and supply a name for the new Output field
* Can set a default value (best practice to do this)
* Ranges (min <= x < max)
  + The lower bound is inclusive (value <= x)
  + The upper bound is exclusive (x < value)
  + A blank value = infinity (lower bound would be negative infinity)
  + Value is the output value for rows that meet the case

### Select Values Step

* sv
* Uses
  + Filter fields
  + Rename fields
  + Edit metadata and/or change datatype and format
  + Duplicate fields
    - choose a field twice, and in at least one of those set a ‘Rename to’ value
* Reorder fields
  + Highlight one or more fields in Select & Alter tab
  + Hold ctrl and use up/down arrow keys
* Meta-data tab
  + rename fields
  + set/change data types
  + length and precision
  + Binary to Normal
  + data format

### Filter Rows Step

* fr
  + usually name with the data that is selected to flow out of this step
    - tho you can set where non-selected data flows as well in configuration
* Provides simple filters for routing rows of data
  + Use JavaScript, Python, R, or other scripting steps if needing complex logic
* Configuration
  + Can add multiple conditions by clicking the Add condition button on the right of the condition pane
  + Use either the <field> or <value> to set the filter condition
    - Value
      * Value entry must be in the same format as that found in the data, and same with Conversion format
* Adding hops after configuration will open context menu to choose Result is TRUE or Result is FALSE
* Hops with green checkmark are the TRUE case
* Hops with red x are the FALSE case
  + usually good practice to at least catch FALSE rows with a dummy step

### Switch / Case Step

* swcs
* Use to filter data going to subsequent steps based on matches to field contents as a string
* Configuration
  + Can only pick one field to examine per step
  + Use string contains comparison (unchecked only finds exact matches, checked finds it somewhere)
  + Case value data type
    - Should match the input field data type
  + Case values
    - Value
      * the value to match exactly or to find contained in the input field
    - Target step
      * must already be on the canvas to populate
      * if not yet present, creating a hop after Values have been entered will bring up a popup menu that will let you choose the case applying to this hop
    - Order matters here!
  + Default target step
    - catches everything else

### RegEx Evaluation Step

* re
* Provides ability to create new fields using regular expressions
* Configuration
  + Field to evaluate = field in which to run the regex against
  + Result field name = output field name (new field created by this step)
  + Create fields for capture groups
    - enable this to create new fields for parts of the regex enclosed between ( and )
    - if checked, you must define the corresponding number of Capture Group Fields in the bottom

### Add Sequence Step

* aseq
* Start at value should be the row number where the data actually starts
  + if there is a header row, set to 2 if wanting to represent the row number (say for an Excel formula variable)

### Merge Join Step

* mj
* **NOTE**
  + input data MUST be sorted by the specified keys
* Configuration
  + First Step (equivalent to Left Table)
  + Second Step (equivalent to Right Table)
  + Join Type (SQL join styles)
* Keys
  + match key(s) from first step to key(s) from second step
* Considerations
  + the Stream Lookup Step can do similar type actions, but that step requires all data to be read into memory (thus Merge Join is superior here)

### Merge Rows (Diff) Step

* mrd
* Use: determine which rows are new, deleted, changed, or identical
  + can then update a database by inserting, updating, and deleting records in a database table using the Synchronize After Merge Step
* **NOTE**
  + Input data MUST be sorted by the Keys to match
  + It is important for the input fields to have the same names from both input sources
* Configuration
  + Reference rows origin
    - this is the reference (original) data
    - this is the data used as a baseline for comparisons
  + Compare rows origin
    - this is the new data with changes
  + Flag fieldname
    - new field that contains the flag value
  + Keys to match
    - the Key field(s) specified must have the same name in each input step
    - these are the row/record identifiers
  + Values to compare
    - fields specified here must have the same name in each input step
    - these are the values being checked for changes
    - if looking for any changes at all, include all fields here

### Synchronize After Merge Step

* sam
* Similar to Insert / Update Step
* Configuration
  + General Tab
    - Define the (database) Connection at the top
    - The key(s) to look up the value(s)
      * fields to compare from the input data and the target table
      * Table field is the field name in the target table
      * Stream field1 is the stream field to compare
      * Stream field2 is only used when Comparator is BETWEEN
    - Update fields
      * Insert and Update fields go in this grid
      * These are the fields that will be inserted if no match is found
      * Set Update = N if you do not want a field updated if a match is found
  + Advanced Tab
    - Operation fieldname
      * the stream field name that contains the flag set in a Merge Rows (Diff) Step
    - The remaining defined fields are the flag values to perform Inserts/Updates/Deletes
      * if set with a Merge Rows (Diff) Step, probably won’t change these
    - Perform lookup
      * if checked, lookups will be performed based on the General Tab’s configuration
      * if unchecked, only the flag values will be used to determine the operation to perform

### Dummy (do nothing)

* dmmy
* Used for testing
  + allows you to see what the inputs going to this step would be (outputs from the previous step)
  + run the transformation, click Preview data in the Execution Results pane at the bottom, click on the different dummy steps to preview what each one receives
* Used to make logic clear
  + can preview the data at this step easily or catch false cases
* Provide execution metrics
  + again for catching things like false cases

**Lookup Steps:**

### Database Lookup Step

* dblkp
* Details
  + can lookup values in a database using a primary key, foreign key, or ID
  + looked up values are added as new fields to the stream
* Configuration
  + Needs a database connection specified
  + Enable cache? will cache the results of the lookup for use later
    - set cache size if checked
  + Load all data from table
    - preloads all of the data from the table in the cache
    - could result in OutOfMemoryExceptions if looking up a large table
  + The key(s) to look up the value(s)
    - Table field = database field name for the lookup
    - Field1 = stream field for which to lookup
    - Comparator = comparison method
    - Field2 only used if comparator = BETWEEN
  + Values to return from the lookup table
    - Field = the database field name
    - New name allows you to rename the field in the stream if desired
    - Default = a default value you want returned (instead of <null>) if a match isn’t found in the database
    - Type = datatype
  + Do not pass the row if the lookup fails
    - essentially creates an INNER JOIN for this step
    - if no database match is found, the row is omitted from output
  + Order by
    - helps specify the lookup value to take if there are multiple matches

### Stream Lookup Step

* slkp
* Details
  + allows you to lookup data using information coming in from multiple steps
* **NOTE**
  + all input data must fit into memory
* Configuration
  + Lookup step = the step where you are looking up values from (not the primary data stream)
  + The key(s) to look up the value(s)
    - Field (from the main stream)
    - LookupField (from the lookup step)
  + Specify the fields to retrieve
    - Field = field name from the lookup step
    - New name (can rename if desired) of field added to the main stream
    - Default (can specify a default value if no match is found other than <null>)
    - Type (can set datatype)
  + Can attempt to optimize this step using the remaining features
    - read the docs

**Scripting Steps:**

### Formula Step

* frmla
* Can formulate expressions using OpenFormula syntax
* Clicking in Formula opens the formula dialogue box

### Modified Java Script Value Step

* scrpt
* Details
  + write custom JavaScript to do a calculation as a variable
  + add to Fields to include the calculation as a field in output rows
  + can create multiple scripts or expressions for each step
* Configuration
  + ScriptValue
    - scripting area
    - right-click the dark-gray area to add, copy, set, and remove scripts
  + Java script functions (left panel)
    - view of available Scripts, Constants, Functions, Input fields, and Output fields
    - Transform Scripts
      * a list of scripts created in this step
    - Transform Constants
      * list of predefined static constants
    - Transform Functions
      * some prewritten JavaScript functions that can make scripting easier
      * double click or drag the function to the ScriptValue window
    - Input and Output fields
      * list of fields available for use in the scripts (double click or drag to use)
  + Fields
    - Fieldname is the name of any variables you are adding to rows (can Rename to if desired)
    - can specify metadata here too

### User Defined Java Expression Step

* udje
* Define a one line Java expression to create new values which can be written to a new field and/or replace the value in the input field
* Expressions compile into pure Java when the step runs
* Use of standard Java object methods is allowed
* if / then / else expressions must use conditional expression operators (conditional ? true\_value : else\_value)

### User Defined Java Class Step

* udjc
* Allows you to create custom steps in Spoon
* Much better performance than the Modified JavaScript Value step when executing similar logic
* To use this step successfully, you must have extensive Java knowledge and read all of the step’s documentation

**Output Steps:**

### Table Output Step

* to
* Used to write rows to a table
  + see Insert/Update Step about updating existing rows or upserting
* Configuration
  + Target table
    - must already exist, will not be created at runtime
  + Commit size
    - uses transactions to insert (every N rows, this is faster but not always supported)
  + Truncate table
    - will delete all existing data from the table before inserting (use with extreme caution!)
    - if multiple steps write to the same table or using step copies, truncate the table before the transformation starts (i.e. in a job)
  + Specify database fields
    - if enabled, you specify the fields to insert and their mapping to database columns in the Database fields tab (preferred)
    - if disabled, all fields going to this step are inserted in the order they are input to this step
  + Main options tab
    - Partition data over tables
      * split the data over multiple tables
    - Use batch update for inserts
      * enable to use batch inserts which groups insert statements to limit round trips to the database (fastest, enabled by default)
    - Is the name of the table
      * use to split the data over one or more tables, the name of the target table must be in a field you specify
    - Return auto-generated key
      * enable if you want to get back the key that was generated by inserting a row into the table
  + Database fields tab
    - specify the fields to insert and their db mappings
  + SQL
    - will display DDL that would result in creating/modifying the target table match what’s being loaded based on current configuration (mostly used for testing purposes and modifying new tables for use with PDI)
    - you can edit this DDL statement
    - **CAUTION!** if you click Execute, this DDL script is run against the database, and it will create or modify the table based on your SQL script if your user has the appropriate permissions!

### Insert / Update Step

* iu
* Important:
  + If multiple rows with the same keys match, only the first one is examined for a match
  + If an update is triggered, all rows with the matching key(s) are updated
* The key(s) to look up the value(s)
  + This is where you specify the items to match
  + All entries here must be met for an existing row to be updated
  + Table field (the field in the database table to examine)
  + Comparator (a weird word that just means how to compare the fields)
  + Stream field1 (the stream field to compare)
  + Stream field2 (only used when the comparator is set to Between)
* Update fields
  + all of the fields you want to either insert or update including keys in the section above
  + you can set update statements to ignore certain fields (Update N) for instance a key that you’re checking to be = in the database and stream that doesn’t need updating
  + fields not listed here would not be inserted if a match isn’t found
    - to include fields on an insert but not an update, just set Update = N for that field

### Dimension Lookup / Update Step

* dimlu
* Slow Changing Dimensions
  + data that rarely/never changes
* Configuration
  + Keys Tab
    - Dimension field in the target table to match
    - Field in stream that matches the Dimension field in the target table
    - These are the keys that identify a row’s identity (like a customer number)
      * there can be multiple rows with the same key value
      * these are essentially updates for that row or key value
      * VERSION and valid dates are automatically updated
  + Fields Tab
    - Dimension field
      * field name in the target table that the stream field is mapped to
    - Type of dimension update
      * Insert: insert new versions for rows with new data
      * Update: update existing rows
    - Technical key field
      * field to use as the primary key of the dimension
    - Version field
      * column that holds the dimension’s revision number
      * will be automatically incremented when a new row is added with the same Key field as specified on the Keys Tab
    - Date range start and Table date range end
      * start and end dates that the dimension is valid
      * the DATE\_FROM and DATE\_TO (example names) fields are updated if new rows with existing keys are added
    - Stream Datefield
      * date the dimension entry was last changed (if you have this)
      * incoming data MUST be sorted by this field if you specify this
* Notes
  + If the target table is empty, a row with all <null> values is created with TECH\_KEY = 0
    - only the VERSION field is populated, everything else is <null>
  + For small tables with less than a million rows, the **Steam Lookup and Update Step** may be preferable (does fewer lookups)

### Text File Output Step

* tfo
* Generates a delimited text file on the local system
* File Tab
  + Do NOT add .txt or .csv on the end of the filename
  + Use the Extension field to set the extension (txt, csv, …)
* Content Tab
  + Separator is the delimiter
* Fields Tab
  + Choose fields to write to the file
  + If you leave it blank, all fields are used in the same order they go in

### Microsoft Excel Output Step

* msxlso
* Use for basic Excel outputs, and use Excel Write Step for more advanced output configurations

### Microsoft Excel Writer Step

* msxlsx
* Features
  + Supports xls and xlsx extensions
  + Insert formulas, hyperlinks, comments, and comment authors
  + Protect sheets with usernames/passwords
  + Auto size column widths
  + xlsx version supports MS Office styles
* Large Excel Files
  + it’s better to use csv for reading large files
* Charts or other objects
  + use xlsx format when working with template files to preserve these objects in the output
* Autosize columns considerations
  + may get incorrect width if appropriate fonts are not installed on the server
* Basic Configuration
  + File & Sheet Tab
    - Set path and filename (without extension)
    - Choose extension
    - Provide sheet name
  + Content Tab
    - Write Header
    - Auto size columns
    - Field grid
* Advanced Configuration
  + Templates
    - Creating a template
      * Create the layout and include fonts and cell formatting
      * Use placeholder text that describes what is there
        + header
        + rows
        + images/logos
      * Can add formulas using the transformation
    - Set in the Template file property in configuration
  + Content Tab
    - Enable Force formula recalculation if adding calucations from formulas using a scripting step
    - Can remove a field from the fields grid and add to the Cell comment column (scroll right)
      * add it to the field for which you want the comment to be added
    - Can remove fields that are only used in formulas of scripting steps (like a line\_nr field)
    - Field title lets you rename the headers
    - Set field contains formula to Y if using a scripting step to calculate a value
    - Can set hyperlinks for a field by clicking Hyperlink and selecting the input field to be the hyperlink
      * my guess is that you could set a field to have a string as the value (displayed) then choose a different field in the Hyperlink dropdown, which would be the url

### XML Output Step

* xmlo
* Content Tab
  + Parent XML element
    - the element that houses a list of records (Customers)
  + Row XML element
    - the element that houses a records (Customer)
* Fields Tab
  + Element name
    - Output name for the field
  + Content Type
    - Attribute
      * assigned within the tag
      * <Customer Attribute1=’value’><Element1>Value</Element1></Customer>
    - Element
      * See above example for attributes vs elements

### MongoDB Output Step

* mngo
* Configure connection
  + Host
    - mcmongodb01p
  + Authentication
    - SCRAM-SHA-1
  + Check ‘Use all replica set members/mongos’ to write if the write operation fails
    - May not be needed once we finish setting up all replicas
* Output options
  + Get DBs (will populate the selectbox, will also tell you if your connection is working)
  + Get collections (will populate the selectbox after choosing a db)
  + Choices for update or upsert records
    - Set ‘Y’ on Match field for updates in Mongo document fields tab
* Mongo document fields
  + Click ‘Preview document structure’ to get a view of what each documents looks like
  + Will produce a document with this structure for every row in the input data to this step
  + For JSON structure
    - Name
      * provides the fields to use
    - Mongo document path
      * you can nest the fields inside a key, this specifies which fields go in which key

## METADATA INJECTION

### Basics Details

* Helps with repetitive design
* Process
  + Create a transformation that serves as the mapping for the injection
    - Two Steps
      * one that gets the data to inject from somewhere (table input, file input, etc.)
        + possible that this data would be processed in additional steps prior to injecting, but usually this is already done
        + this input must contain all of the field names for variables that will be used in the metadata injection
        + rows of data are the different values for those variables

example: using two fields (fieldname, new\_fieldname) with values that specify fields to rename

* + - * ETL Metadata Injection Step (details below)
  + Create a transformation for the logic to repeat over and over (template transformation)
    - Steps that need metadata injection are created, but configuration is left blank
  + Run the transformation that has the Metadata Injection Step (emi) NOT the template transformation
    - This will automatically call the template transformation

### ETL Metadata Injection Step

* emi
* Configuration
  + Transformation
    - directory to the template transformation
  + Inject Metadata
    - Details
      * this area will read all steps from the template transformation and provide stuff to edit here
      * scroll to any steps where you want to inject metadata
      * the properties available for a step correspond to its normal configuration but may be named differently
    - Injection
      * click on the item to configure
      * select the source step and field to use as that value
      * repeat for any things to inject

**Other Transformation Helpful Notes:**

### Inspect Data from a Step

* Click a step -> click the inspect data button (right button of the graph) on the fly out inspection bar (blue) at the top of the canvas
* Sream View
  + select/deselect columns to view
  + sort by clicking on headers
  + drag and rearrange columns
  + can select different ‘view’ options using the visualization inspector dropdown (probably says table when loaded by default)
    - drag and drop to build the plot
  + can publish as a data source (cloud button in the top right) if we have the Business Analytics Suite (don’t think we have this)

### Using Data Collected from an Input Step to Set Configuration of Another Step

* Scenario: using data stored in an Excel file to set filepath and filenames for a Text File Input Step
* The step to read in settings must occur first and have a hop to the step it configures
* Configure the first step to read in the data you need into rows
* In the second step’s configuration, there is an Accept filenames from previous step checkbox (File Tab)
  + This disables the top input area with File or directory and regex info

### Copying/Pasting Steps

* Can help with creation of many similar steps
* Should rename and double check configuration if doing this

## JOBS

### Job Use

* Use jobs to ensure proper execution of ETL process
* Can control and apply logic to orchestrate the ETL flow
* Example of common job processes
  + resource checks, logging

### Jobs Best Practices

* Job naming conventions
  + Job filename
    - j\_short\_description
  + Job entries
    - Transformations
      * use transformation naming conventions
      * verb\_short\_description
    - Simple Evaluation
      * se-purpose\_or\_attribute
    - SQL
      * sql-action\_purpose\_or\_filename
    - Write to log
      * wtl-purpose\_or\_jobname
    - Job
      * j-\_job\_name
    - Set variables
      * svar-purpose
* Build transformations first, then arrange into jobs
* Always preview steps/transformations before including in Job stream
* Loops should generally be avoided in jobs, but if used, you should set a limit to the number of loops
  + JavaScript Job Entry can do this

### Job Orchestration

* Tracking Job Entry Execution
  + When a Job Entry is executed, a green checkmark appears in the upper-right corner of the entry
  + Job Entries that are not executed will not have a green checkmark
    - An exeption is the Abort Job Entry, which ends with a red x when executed
* Common Job Tasks
  + get FTP files
  + checking conditions
  + running transformation
  + running another job
  + email error log
* Job Entries
  + primary building blocks (homologous to steps in transformations)
  + every job starts with the **Start** job entry
* Job Entry Excecution Order (Backtracking Algorithm)
  + Job Entries are not (by default) executed in parallel
  + Order depends on the order they were created in the job
  + A path is followed to the end before the next possible path is considered

### Job Hops

* Hop types (click on hop icon to change the hop type)
  + Unconditional (lock icon on hop)
    - next entry is executed regardless of the result of the previous job entry
  + Success (True) (green checkmark icon on hop)
    - next entry is executed only when previous job entry is successful/evaluates to True
  + Fail (False) (red x icon on hop)
    - next entry is executed only when previous job entry is a failure/evalutates to False

## JOB ENTRIES

### Start Job Entry

* START
* Required first step for all jobs
* Only unconditional hops can be used from Start

### Abort Job Entry

* abrt
* Halts execution of the job and can write a message to the log file

### Job Job Entry

* Executes a previously defined job
* Helps break jobs into manageable units

### Mail Job Entry

* mail
* Sends and email
* Attachments (like log files) supported

### Transformation Job Entry

* Runs a previously defined transformation

### Convert file between Windows and Unix Job Entry

* Converts CRLF/LF line endings
* Can set to guess, and it will guess what it finds and convert to the other

### Set Variables Job Entry

* Set variables manually or read in
* Can set scope
* Best practice to set in a parent job before using

### File Exists Job Entry

* fe
* Check if a specified file exists or not

### SQL Job Entry

* sql
* Execute one or more SQL scripts separated by ;
* Flexible, and allows procedure calls, DDL scripts …
  + truncating tables, dropping indexes, partition loading, refresh materialized views, disable constraints, etc.
* Can read SQL script from file rather than pasting into the SQL Script box

### Success Job Entry

* Success
* Clears any error states and forces it to a success state
* Good idea to use this to identify where the job execution is considered successful

### HTTP Job Entry

* Gets or uploads a file using HTTP

### Write to Lob Job Entry

* wtl
* Writes a specific string to the PDI logging system

### Wait for File Job Entry

* Sleeps the current running job and periodically checks for a specified file
* Continues the current job once the specified file is found
* Can wait indefinitely or timeout after a period of time

## PENTAHO REPOSITORY

### Basics

* A collaborative database for storing/managing jobs/transformations
* Provides revision history and ability to revert to previous versions

### Repository Features

* Content Management
  + Browse/explore contents
  + Revision history/version control to restore previous version of jobs/transformations
* Enterprise Security
  + LDAP (and MSAD) authentication configurable
  + Permission levels for users
* Scheduling
  + Available using Pentaho Enterprise’s built-in scheduler
* Technology
  + Repo based on Apache Jackrabbit (JCR – Java Content Repository)
* Unsupported Repositories
  + PDI can also create:
    - Database repository
      * uses central rdbms to store ETL metadata
    - File repository
      * uses local file system to store ETL metadata
    - These are not supported or recommended for production

### Connecting to a Repository

* First time setup
  + With a job/transformation open -> **Connect** (upper right of the window)
  + Get Started
  + Display Name (enter something for you, this is your display name)
  + URL for the Pentaho Server
  + Description
  + Can check to launch it on startup
* Logging in
  + Use Pentaho Server un/pw
  + Upper right of window now has your username and the Display Name of the connection
  + Can click here to Disconnect
* Saving
  + Spoon saves to the repository by default when you click save
  + Your user folder is private to you and any admin users
  + The Public folder is for shared files and is visible to everyone
  + To save to the file system
    - **File** -> **Export** -> to **XML**
    - Transformations/Jobs are saved as XML files anyway
* Opening
  + Use the open button just like always, it simply connects to the repo
  + Importing from the file system
    - **File** -> **Import from an XML file**
    - Does not save the file to the repo until you click **Save**

### Repository Explorer

* NOTE: Version History/Control is disabled by default, must be enabled
  + See PDI Documentation
  + If enabled, a new Version is created every time an object is saved
  + Not a replacement for git, which is more robust
* Launching Methods
  + Click third button (hovering reveals **‘Explore Repository’**) on Spoon toolbar
  + **Ctrl + e**
* Browse Tab
  + Manging jobs/transformations, access control to jobs/transformations, and version histories
  + Folders Pane
    - Right click a folder to create a new folder, open, rename, delete, or export
    - Can drag and drop jobs/transformations to folders
    - All deleted objects are placed in the Trash
      * these take up space in the repo, so empty regularly
  + Version History Tab (bottom)
    - Click on a job/transformation to view version history at the bottom
    - New versions are saved every time you click save if enabled
    - Right click a version to open it or restore to it (overwrites the current file and creates a new version that is a copy of the one used to restore)
    - You cannot delete a version from the Repository Explorer, and this must be done using a command line utility to purge the repository (see PDI documentation)
  + Access Control Tab (bottom)
    - Can specify users/roles for specific jobs/transformations if you have permissions to do so
    - Read will alow the user to execute the job/transformation
  + Right click a job/transformation to open, rename, delete, lock, or create a new folder in the current folder
    - Locked jobs/transformations
      * can still be executed
      * save as also works
      * often done before moving to production
      * can’t move a locked object
      * only an admin or the user who locked an object can unlock it (**right-click** -> **lock**)
      * **right-click** **-> lock notes** to view the user, message, and datetime when an object was locked
  + Note:
    - ‘Folders’ pane looks like a filesystem tree, but this is actually stored in a database
    - administrator roles can see and access all users home folders here
* Connections Tab
  + Lists all database connections stored in the repository
  + Icons in upper right of window
    - Edit selected connection
    - Add new database connection
    - Delete selected connection
  + User/Role pane
    - Specify the users or roles that can access the selected connection and the permissions those users have
    - Manage Access Permission
      * Checking allows the specified user/role to modify the User/Role and Permissions in this window for other users/roles
* Security Tab
  + Users Radio Button
    - Manage users, permissions, and roles
    - Click on an existing user to see roles assigned to that user
    - Assign or unassign roles to a user using the + and x buttons on the far right of this window (Member of pane)
    - Edit selected user (pencil), add new user (+), or delete (x) selected user with three buttons to the left (Available pane)
  + Roles Radio Button
    - Edit, create, or delete roles (Available pane icons)
    - Add/remove users from a selected role (Members pane icons)
    - Grant/remove permissions for a selected role in the Permissions pane (checkboxes, click Apply)
  + System Roles Radio Button
    - Assigned by PDI implicitly
    - Authenticated assigned to any user that successfully logs in
    - Can set permissions for system roles just like regular roles

### Importing/Exporting Repository Objects

* General Info
  + Only current versions of .ktr and .kjb objects are processed
  + Exported content is a concatenated XML file
  + Database connections may be included depending on Kettle Options dialog settings in Spoon
  + Do not use to backup/restore the Repository, instead use Backup and Restore utility
  + Can configure Import/Export Rules
    - Requires a description -> can set number of chars needed
    - Has to log to a log table -> schema, table, connection
    - Must have a note
    - Has no disabled hops
    - Validate the configuration of database connection -> db connection criteria (server, un, pw, etc.)
* Configuring Kettle Options for Import/Export of Database Connections
  + **Tools** -> **Options**
  + Only save used connections to XML
    - if checked, only connections used are saved/exported (best practice)
    - if unchecked, all database connections are saved/exported even if not used in current job/transformation
* Using Spoon
  + Connect to the repository if not already connected
  + Export
    - Single Job/Transformation
      * open the job/transformation to export
      * **File** -> **Export** -> **To XML**
      * navigate to folder where the export should be saved
      * provide filename and save (saves as .kjb or .ktr file, which recall are XML files)
    - Repository Folder and Subfolders
      * open Repository Explorer (third toolbar button)
      * Browse Tab and Folders Pane -> **right-click folder to export** -> **Export**
      * navigate to location on filesystem to save the file
      * provide ‘File name’ even though exporting a folder -> click **Save**
      * results in a single XML file with all of the exported content
      * dialogue opens asking if you want to Apply import rules?
        + can ‘Add rule’ or ‘Remove selected rule(s)’ here
        + a list of available rules to choose from is displayed

you cannot create custom rules

* + - * + Import/Export buttons in this window can Import/Export the set of rules that are currently defined in this window (can reuse sets of rules)
        + click OK to run the export
      * details on export process can be seen in Spoon’s log file (/path/to/Pentaho/design-tools/data-integration/logs/spoon.log)
        + shows details of jobs/transformations that were not exported because they failed rules (no error dialogues are shown if this happens during export)
    - All Repository Folders (not a substitute for backups)
      * **Tools** -> **Repository** -> **Export Repository**
      * remaining process is identical to exporting a Repository Folder
        + define location, filename, and rules
  + Import
    - Single Job/Transformation
      * **File** -> **Import from an XML file**
      * navigate to and select the .kjb or .ktr file to import and click open
      * must save the object to save it in the Repository
    - Folders or Entire Repository
      * **Tools** -> **Repository** -> **Import Repository**
      * locate the file to import then click open
      * option to Apply import rules?
        + can apply different rules for import than were applied for export
        + process is the same as during the export process
      * navigate to destination location (can create new folders there)
      * Enter a comment to be applied to all successfully imported objects
      * there is a dialogue that shows import progress, successfully imported objects, objects that don’t pass rules, and any errors
* Using the Command Line
  + Allows you to script and automate this process
  + Import
    - use import.bat (Windows) or import.sh (Linux) utility
      * these files are available in the /data-integration/ folder
    - supports importing individual .ktr/.kjb files or a repository export file
    - requires a ‘rules’ file defining the rules to use
    - sample import-rules.xml file also available there, and I think you can export one from Spoon if you follow the steps under Using Spoon -> Export -> Repository Folder and Subfolders
    - sample syntax:
      * import.bat -rep=PRODUCTION -user=admin -pass=p@55word -dir=/ -file=myJob.kjb -rules=import-rules.xml -coe=false -replace=true -comment=”Ready for production”
      * a full list of arguments and descriptions is available at help.pentaho.com
  + Export
    - use either kitchen.bat/.sh or pan.bat/.sh
      * these files are available in the /data-integration/ folder
    - exported content is in XML format
    - rules are NOT supported for command line export (everything is exported)
    - sample kitchen.bat syntax:
      * call kitchen.bat /file:C:\Pentaho\_samples\repository\_export.kjb “/param:rep\_name=PDI2000” “/param:rep\_user=admin” “/param:rep\_password=p@55word” “/param:rep\_folder=/public/dev” “/param:target\_filename=C:\Pentaho\_samples\repository\export\dev.xml”

## WORKFLOWS

### Transformations

* **Spoon** -> **New File** toolbar icon -> **Transformation** (**crtl + n**)
* Navigate Design tab in left nav bar to locate steps or search for them
* Drag/drop steps to the canvas from the Design tab or double click the step name
* Double click a step to open its configuration diaglogue
  + Help button in lower left corner of these dialogues for documentation
* Best practice to preview a step to run a step or transformation and instantly view the stream at a particular break point
  + Previewing actually executes the logic, it is NOT a test/sandbox
  + Runs this and show me the results (stream)
* Automatically add hops
  + Click on an existing step
  + Add a new step from the Design tab by double clicking
  + NOTE: some steps produce automatic dialogues that are helpful when you add them, and this does NOT occur when you add steps in this way
* Manually add hops
  + Method 1
    - Hover over the source step
    - Click, hold, and drag the box with an arrow to the destination step
  + Method 2
    - Hold shift
    - Click on the source step, hold, and drag to the destination step
* Delete hops
  + **Right-click a hop** -> **Delete**
* Execute a transformation
  + Click the **Run** (play) button on the Transformation’s toolbar
  + Confirm/edit settings
  + Click **Run**
  + Look for green checks on each step to signify successful completion
* Viewing Errors
  + They show up red in the Logging Tab of the Execution Results pane at the bottom
  + Click the Show Errors button on the Logging Tab
  + The step(s) in error state have a red border and the do not enter (red circle) symbol by them
  + Hover over the red circle for errors associated with that step
  + View the log for more details
* View/edit/configure fields going into a step
  + **Right-click a step** -> **Input Fields**
  + Highlight a field and click Edit Origin Step to modify
* View fields coming out of a step
  + **Right-click a step** -> **Output Fields**

## LOGGING

### Log File Location

* Pentaho Local execution
  + <spoon\_install\_dir>\logs\spoon.log
* Pentaho Server execution
  + <pentaho\_install\_path>/pentaho-server/logs/pentaho.log
  + if not there, check for a tomcat dir in /pentaho-server/

### Logging Levels

* Nothing
* Error
  + errors only
* Minimal
  + extremely basic details + errors
* Basic
  + default
  + most common for production environments
  + I/O, R/W, Update and Error
* Detailed
  + Detailed info for each step
    - general SQL executed
    - file execution path output
    - code execution in the Modified Java Script Value step
* Debug
  + most common level for dev/debugging
  + detail + debug lines
* Row Level
  + be careful with this one
  + every row that’s in the PDI stream is included in the log
  + dev only
  + can create a ginormous amount of data and impact performance

### Logging Tab (Execution Results Panel)

* Log Entry Format
  + timestamp – step\_name.copy\_number – message

### Logging Execution Metrics

* Can log execution metrics to a database
* It is best practice to use this to help optimize ETL processes
* Data in Execution Metrics
  + number of records inserted/modified
  + total time elapsed for execution
  + exceptions
  + errors
  + debugging details
  + step performance monitoring
* Configuring
  + details below apply to jobs/transformations though the name of fields might be slightly different
  + open the job/transformation to configure
  + **ctrl + t** to open the Log dialog for a job/transformation or Navigate to the Log tab after **double clicking the canvas**
  + Left Pane
    - lists the level to configure
    - Job log table must be configured to do the Job entry log table
    - Job log table -> logs job level metrics
    - Job entry log table -> logs metrics for each job entry
    - other options are more advanced configurations (see documentation for details)
  + Connection Configuration
    - Specify the database connection, schema (if supported by the target db), and destination table
    - Logging interval (seconds)
      * sets the log interval; if left blank, one row will be insterted for every run
    - Log line timeout (days)
      * sets the amount of time rows are left in the database for this job/transformation before being deleted the next time the job/transformation runs
      * best practice to set this (30 days is common)
    - Log size limit in lines
      * sets the max limit for log lines stored in the LOG\_FIELD (long text/CLOB field)
      * not needed if not including LOG\_FIELD in the Log table fields
    - Best practice to use variables for setting up the connections
      * kettle.properties already has variables named for this purpose, you just need to set them
  + Log table fields
    - Configure the fields to log
    - Always check ID\_JOB, JOBNAME
    - The LOG\_FIELD will contain the entire log that was output and can bloat the database where the logging metrics are being output
      * best practice to only use this when troubleshooting
    - Best practice to check CLIENT so you know which process ran the job (Spoon, pan, kitchen, carte)
    - Best practice to include JOBNAME/TRANSNAME and STEPNAME when configuring Job entry log table
  + Creating tables to hold the logging metrics
    - If the tables do not yet exist, you can click the SQL button after configuring the connection
    - This generates the DDL SQL commands needed to create the table in the connected database to exactly match what you have configured
    - Click Execute to run the DDL statement against the connected database
* Quickly view logs
  + **Spoon main view** -> **View Tab** (left side) -> **Database connection** -> the connection where you’re storing logging metrics

## DEBUGGING

### Pentaho Local

* Hover over the red circle on a step to see error messages
  + they don’t last long, take a screenshot if needed
* Examine the log for messages (can be time consuming)
* Click the **Show Error Lines** button in the Logging Tab (first button)

### Pentaho Server

* A new tab opens when running transformations on a server
* Expand the Tranformation and click on the step
* View the log in the bottom pane that opens for the log entries associated with that particular step

## DATA TYPES

### Important Notes

* Each data type maps to a specific Java Class
* Creating custom data types involves creating Java Classes and implementing them into PDI

### Data Types

|  |  |  |
| --- | --- | --- |
| Name | Details | Java Class |
| Big Number | Number with unlimited precision | java.math.BigDecimal |
| Binary | Array of bytes (any type of binary data) | java.lang.byte[] |
| Boolean | true / false | java.lang.Boolean |
| Date | Datetime with millisecond precision | java.util.date |
| Integer | Signed long 64-bit integer | java.lang.Long |
| Internet Address | IP address | java.net.InetAddress |
| Number | (Double) precision float | java.lang.Double |
| String | Character data, <= 2,147,483,647 chars | java.lang.String |
| Timestamp | Datetime with nanosecond precision | java.sql.Timestamp |

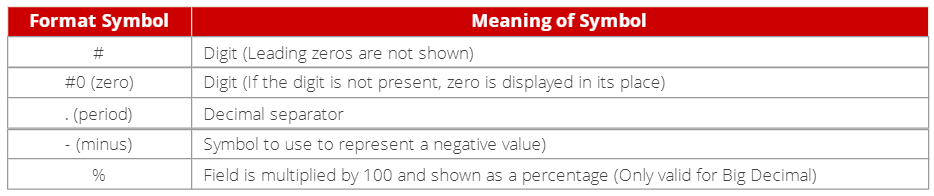
### Convert Data Types

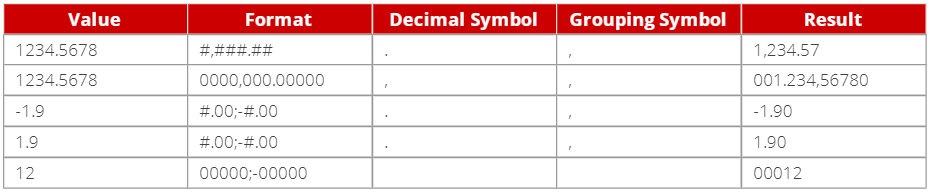
* Usually performed in a Select Values Step
* The Calculator step uses input datatype to calculate values
  + More details in the Calculator Step section

### Format (when configuring fields)

* Available in the documentation
* Different chars for different datatypes
* Can specify multiple formats using ; as a separator
  + often used to format positive vs negative numbers

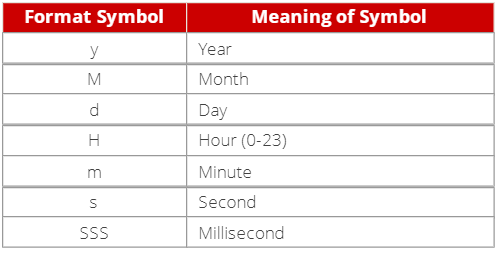
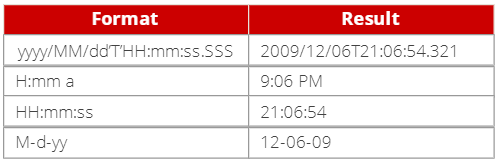
#### Numeric Format Chars





#### Date Format Chars

* When importing dates, you must use the same format that is in the input source
* To change date formats, you must read as string matching the source format first in one step (like Select Values), then do another Select Values step (or similar) that lets you change the field’s Format configuration

Converting to Booleans

* Booleans are either Y or N when set by Pentaho
* 0 -> N, 1 -> Y, true -> Y, any other string -> N

## CONFIGURING FIELDS

### Format

* See data types section

### Length

* Total number of significant digits

### Precision

* The number of floating point digits

## VARIABLES AND PARAMETERS

### Syntax for Referencing a Variable/Parameter

* Can use in jobs, transformations, and system configuration menus anywhere the symbol is displayed
  + usually to the right of the input box appearing as 
  + Hover over the symbol for tips on entering variables (**ctrl + space**)
  + Once (**ctrl + space**) is keyed, a select box opens
  + Click a variable/parameter once, and a tool tip displays its current in memory value
  + Double click to add the variable/parameter at the cursor location
  + You can hard code additional characters outside of the variable syntax
  + You can add multiple variables to a box
  + Hovering over the input box that contains variable(s) and any other chars will display what it currently evaluates to
* Use either
  + ${variable\_name}
  + %%variable\_name%%
* See Table Input Step for more info on using variables/parameters in SQL statements

### Inspect Parameters/Variables Loaded into Memory

* Click **Run** button
* Inspect the Parameters and Variables tabs
* Note: Variables that have been overridden by parameters will not appear
* Can edit parameters/variables directly in this dialogue prior to running
  + these changes are not permanent and do not modify existing configurations (good for testing)

### Variables

* Variables are global in scope
  + within the environment (VM) or within a parent/root job
* Project related variables and values can be defined in the kettle.properties file
* **kettle.properties**
  + Use for environment variables to be available to every job/project
  + Edit kettle.properties from Spoon
    - **Edit** -> **Edit the kettle.properties file**
    - Edit the existing variables and values
  + Edit kettle.properties directory (text editor)
    - open the file in the .kettle directory
    - edit the variables
      * VARIABLE\_NAME=value
    - must restart Spoon to refresh variables
* **Create custom .properties files**
  + Use for variables that are related to few jobs/projects
  + Use a Set Variables job entry to load into Spoon
* Internal Variables
  + Start with Internal.\_\_\_\_

### Parameters

* Local variables whose scope is only within the job/transformation in which they are defined
* Parameters override Variables with the same name
* **Create or Edit Parameters**
  + Open the job/transformation properties dialogue by **double clicking the canvas** (**ctrl + t** or **ctrl + j**)
  + Click on **Parameters tab**
  + **Edit/add parameters**

## PDI’s HOME DIRECTORY

### Configuration Files

* Stored in C:\Users\<user\_name>\.kettle
  + ~/.kettle
* To unify settings across users
  + declare a KETTLE\_HOME environment variable
  + leave out .kettle, this is added by PDI
* .kettle contents
  + kettle.properties
    - global system variables/values
    - used for parameterization
    - only one that you should edit manually
  + shared.xml
    - objects that have been shared
    - available for all users that point to the same PDI Home Directory
    - examples of shared objects
      * database connections
      * steps
      * slave servers
      * partition schemas
      * cluster schemas
    - sharing an object
      * right-click on the object in the View tab (left panel) of Data Integration Perspective -> Share
  + db.cache
    - database cache for metadata
  + repositories.xml
    - connection details for PDI database and solution repositories
  + .spoonrc
    - UI settings including last opened transformation/job and dialog box last position/size
  + .languageChoice
    - default language

## SCHEDULING

### Scheduling

* Enterprise PDI’s Built-in Scheduler
  + Uses Quartz; a Java enterprise scheduler
  + Can schedule jobs/transformations
  + Items must be in the Repository to schedule
  + Current version is always executed
    - if you update an object, no need to modify the scheduler
* Third-Party Scheduler
  + No official integration is supported
  + Typically used to run Pan/Kitchen scripts
  + Details on Customer Portal
* OS Level Scheduler
  + Schedule pan.bat/Kitchen.bat via Window Task scheduler
  + Schedule pan.sh/Kitchen.sh via cron utility job scheduler (Unix)

### Scheduling via Spoon (built-in scheduler)

* Connect to Repository
* Open the job/transformation to schedule
* **Action** -> **Schedule**
  + if Schedule is grayed out, be sure the object is saved in the Repository
* Start
  + Set the start datetime
* End
  + Set the end datetime or No end to run indefinitely
* Repeat
  + Set frequency and specify exactly what day the object should run
* Details
  + Enable safe mode
    - when activated, PDI checks that all rows have the same layout as the first row
    - if the layouts differ, an error is generated and reported
    - best practice is to only use when troubleshooting, because performance is affected in safe mode
  + Choose the logging level when the scheduled object runs
* Parameters, Variables, and Arguments
  + Can set these for when the job runs on the scheduler

### Scheduler Perspective

* Allows schedule management
  + edit a scheduled run
  + stop a scheduled run
  + enable or disable a schedule
  + delete a schedule
* **View** -> **Perspectives** -> **Scheduler**
* Provides the following info about a schedule
  + Name
  + Type (job/transformation)
  + State
  + Next Run
  + Last Run
  + Scheduled By (username)
* IMPORTANT
  + The fourth icon (calendar with square or triangle)
    - will enable/disable the entire scheduler
    - only indication is the square vs. triangle
    - schedules will still say ‘Normal’ even when the entire scheduler is disabled