

Meeting

**Pentaho Data
Integration**

License

This work is licensed under the Creative Commons Attribution-Noncommercial-Share Alike 3.0 Unported License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-nc-sa/3.0/> or send a letter to Creative Commons, 171 Second Street, Suite 300, San Francisco, California, 94105, USA.

All trademarks and logos included or mentioned in this document belong to their legitimate owners.

Author

María Carina Roldán
Bachelor degree in Information Technology (UNLP – Argentina)
email: maria.carina.roldan@gmail.com

Preface

This document is an introduction to PDI, the integrating tool of the Pentaho Suite. The text is built on the classic "Hello, World", explaining the fundamental concepts of the tool in a step-by-step fashion.

The intended audience of this work are those who have never used PDI, but those who did, can also get benefit of it, because the text clarify some concepts seen again and again in the PDI forum (i.e. user-defined variables)

The text is organized as follows:

- ✓ **Introduction** – In the first place the tool is introduced, and instructions are given for the installation.
- ✓ **Hello, world** – This is the first example, with step-by-step instructions.
- ✓ **Hello, world again** – It's a complete and enhanced version of the example. In this example more important concepts are explained.

At the end of the paper, tips and links are given for those who want to work with PDI from now on.

Contents

<u>License.....</u>	<u>ii</u>
<u>Author.....</u>	<u>ii</u>
<u>Preface.....</u>	<u>iii</u>
<u>Contents.....</u>	<u>1</u>
<u>Introduction.....</u>	<u>2</u>
<u>Installing PDI.....</u>	<u>2</u>
<u>Meeting Spoon.....</u>	<u>3</u>
Repository and Files.....	3
Starting Spoon.....	3
<u>Hello, World.....</u>	<u>4</u>
Task.....	5
Preparing the Environment.....	6
Step by Step.....	6
How does it work?.....	15
Verify, Preview and Execute!.....	15
Pan.....	18
<u>Hello World again!.....</u>	<u>20</u>
Task.....	20
Preparing the Environment.....	22
A. Creating the transformation which takes the parameter.....	23
B. Modificating the transformation Hello.ktr.....	26
C. Building the main job.....	28
How does it work?.....	32
Kitchen.....	33
<u>Conclusion.....</u>	<u>35</u>

Introduction

PDI (aka Kettle) is the component of Pentaho responsible for the ETL processes. Yet every ETL tool including PDI is most frequently seen in data warehouses environments, PDI can also be used for other purposes:

- Migrating data between applications or databases
- Exporting data from databases to flat files
- Loading data massively into databases
- Data cleansing
- Integrating applications
- etc.

PDI is easy to use. Every process is created with a graphical tool where you specify what to do without writing code to indicate how to do it. Therefore it is said that the solution is meta-data oriented.

PDI can be used as a stand-alone application, or integrated with the other components of the Pentaho Suite. As an ETL tool, is the most popular among the open source tools.

PDI supports a vast kind of input and output formats, including flat files, data sheets, and connection with commercial and free database engines. Moreover, the transformation capabilities of PDI allow you to manipulate data with very few limitations.

In this tutorial you'll see you how easy is to work with PDI. Through the build of a simple "Hello, World" you will meet the most used characteristics of this tool. After reading the whole paper, you will have the willing to build your own project.

Installing PDI

The first step for working with PDI, is to install the tool.

PDI can be downloaded from <http://community.pentaho.com/sourceforge/>

At this time, the newest released version is 3.0.3. The file you have to download is `Kettle-3.0.3.GA-nnnn.zip`

PDI does not require installation (except if you download the .exe file)

The only prerequisite for working with PDI is to have JRE 5.0 or higher installed.

It can be downloaded from <http://www.javasoft.com/>

After you check this prerequisite, you simply have to unzip the zip file in a folder of your choice. Under Unix-like environments, you will need to make the shell scripts executable. If this is your situation, execute the following commands: (assuming that you chose `Kettle` as the installation folder)

```
cd Kettle
chmod +x *.sh
```

Now you have all you need to start with PDI.

Meeting Spoon

When you see PDI screenshots, what you are really seeing are Spoon screenshots. Spoon is the graphical tool in which you design and test every PDI process. The other PDI components execute the process designed with Spoon, and are executed from a terminal window as you'll see in a while.

Repository and Files

In Spoon you build **Jobs** and **Transformations**. In order to save the Jobs and Transformations, PDI offers two methods:

- Database Repository
- Files

If you choose the repository method, the repository has to be created the first time you execute Spoon.

If you choose the files method, the Jobs are saved in files with extension `kjb`, and the Transformations, in files with extension `ktr`.

In this tutorial you'll work with the second method.

Starting Spoon

Let's start working. Start Spoon executing:

`Spoon.bat` in Windows

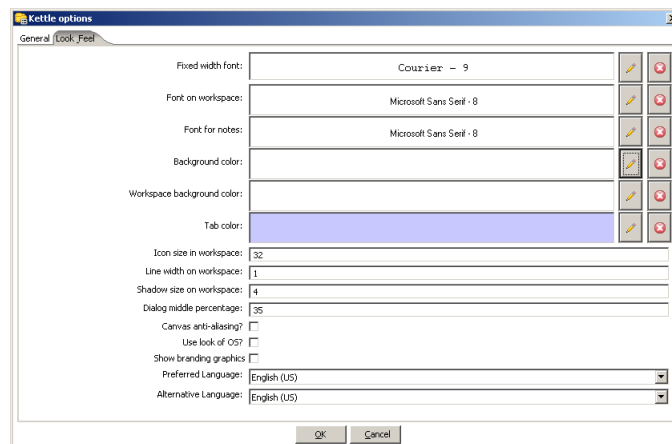
or

`Spoon.sh` in other platforms like Unix, Linux, ...

As soon as Spoon starts, a dialog window appears asking for the repository connection data. As said, you'll work with files. So, click the **No Repository** button.



The first you'll see will be a welcome window. This first time, you probably want to change some of the look and feel. Click **Options...** from the menu **Edit**. A window appears where you can change various general and visual characteristics. If you change something, it will be necessary to restart Spoon in order to see the changes applied.



Now you're ready to face the "Hello, World".

Hello, World

"Hello, World" will be your first experience with PDI. Although this will be a simple example, it will allow you to meet some of the fundamentals of PDI:

- Working with the Spoon tool
- Transformations

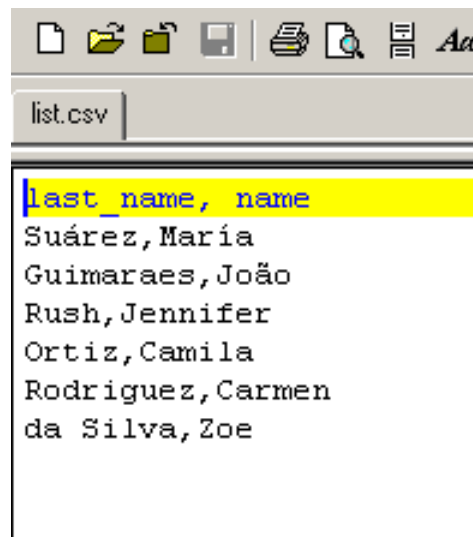
- Steps and Hops
- Predefined variables
- Previewing and Executing from Spoon
- Executing Transformations from a terminal window with the Pan tool.

Pay attention! Each time you see a box like this, have in mind that it contains important definitions and tips that will be useful to you beyond the tutorial.

Task

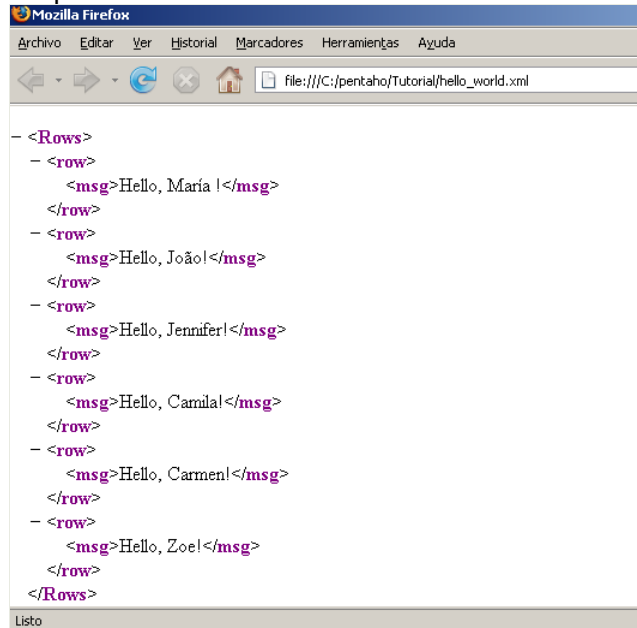
Let's suppose that you have a `csv` file containing a list of people, and want to create an `xml` file containing greetings for each of them.

If this were the content of your file:



```
list.csv
last_name, name
Suárez, María
Guimaraes, João
Rush, Jennifer
Ortiz, Camila
Rodriguez, Carmen
da Silva, Zoe
```

This would be the output:



```
- <Rows>
- <row>
  <msg>Hello, María !</msg>
</row>
- <row>
  <msg>Hello, João!</msg>
</row>
- <row>
  <msg>Hello, Jennifer!</msg>
</row>
- <row>
  <msg>Hello, Camila!</msg>
</row>
- <row>
  <msg>Hello, Carmen!</msg>
</row>
- <row>
  <msg>Hello, Zoe!</msg>
</row>
</Rows>
```

The creation of the file with greetings from the flat file will be the task of your first Transformation.

A **Transformation** is an entity made of **Steps** linked by **Hops**. This Steps and Hops conform paths through which flows data: the data enter, is transformed and leave. Therefore it's said that a Transformation is **data-flow** oriented.

Preparing the Environment

Before starting with the Transformation, create the folder `Tutorial` (inside or outside the installation folder; you choose). There you'll save all the files of the tutorial.

Then create a file like the one seen in the introduction of this example, and save it with the name `list.csv` in the created folder.

The existence of this file is not mandatory, but if the file exists, it will be easier to configure the first Step of the Transformation.

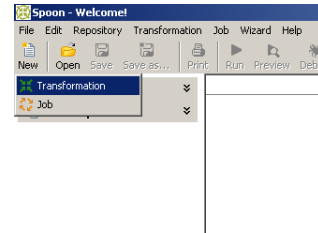
Step by Step

The proposed task will be accomplished in three subtasks:

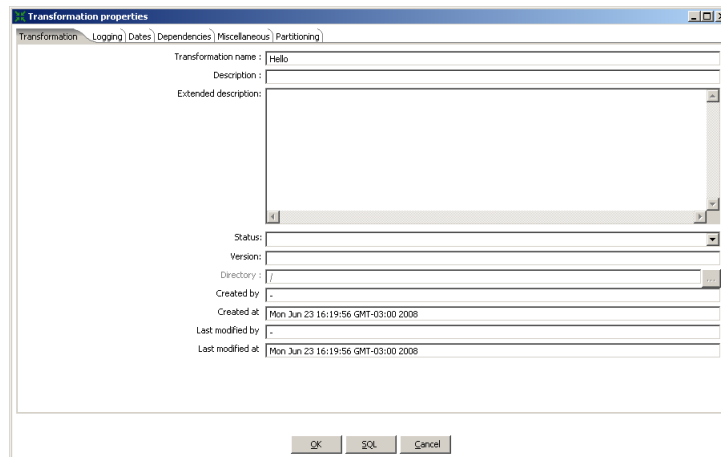
- 1)Creating the Transformation
- 2)Constructing the skeleton of the Transformation using Steps and Hops
- 3)Configuring the Steps in order to specify their behavior

Subtask 1. Creating the Transformation

–Select **New** → **Transformation**. Another option for doing the same is to create a Transformation from the menu **File** → **New** → **Transformation** or to press **<Ctrl-N>**. Almost the entire screen is now occupied by this new Transformation. This is the workspace where you will be dropping Steps and Hops.



–Select the option **Transformation** → **Configuration**. A window appears where you can specify Transformation properties. In this case, just write a name and a description, like here:



–Press the **Save** button. Save the Transformation in the folder **Tutorial** with the name **hello**. The file **hello.ktr** will have been created.

Subtask 2. Constructing the skeleton of the Transformation using Steps and Hops

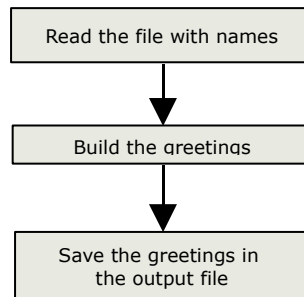
A **Step** is the minimal unit inside a Transformation. A big set of Steps is available. These Steps are grouped in categories like Input or Output among others. Each Step is conceived to accomplish a specific function, going from reading a parameter to normalizing a dataset.

A **Hop** is a graphical representation of data flowing between two Steps: an origin and a destination. The data that flows through that Hop constitute the **Output Data** of the origin Step and the **Input Data** of the destination Step.

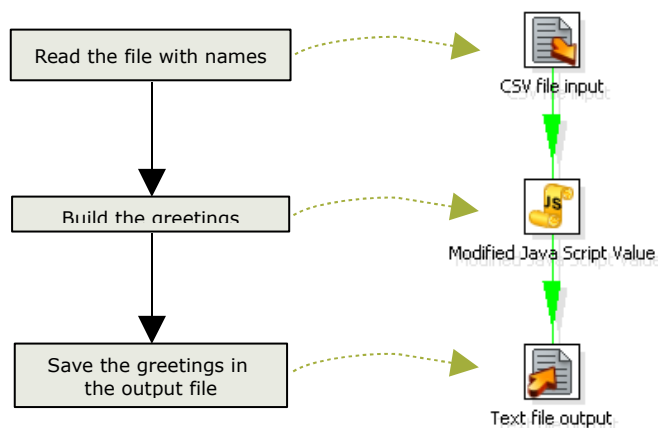
A Hop has only one origin and one destination. However, more than one Hop could leave a Step. In that case, the Output Data can be copied to every destination, or can be distributed to them.

Also more than one Hop can reach a Step. In that case, the Step has to have the ability to merge the Input from the different Steps in order to create the Output.

Let's see what the Transformation has to do:



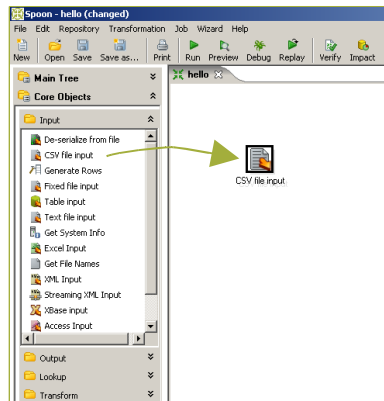
For each of the items you'll use a different Step, according to the next diagram:



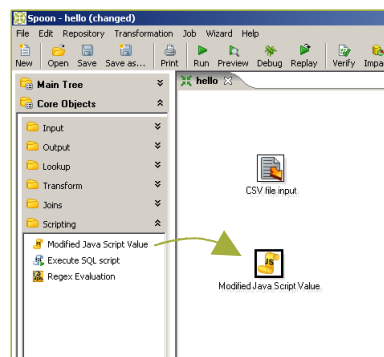
Have in mind that here you have a one to one correspondence because the Transformation is very simple. It isn't always so!

These are the steps to follow:

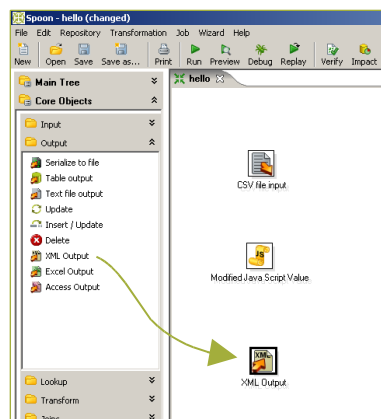
-To the left of the workspace is the Steps Palette. Identify in the palette the **Input** category. Click the **CSV file input** step and drag it to the workspace.



–Identify in the palette the **Scripting** category. Click the **Modified Java Script Value** step and drag it in the same way.

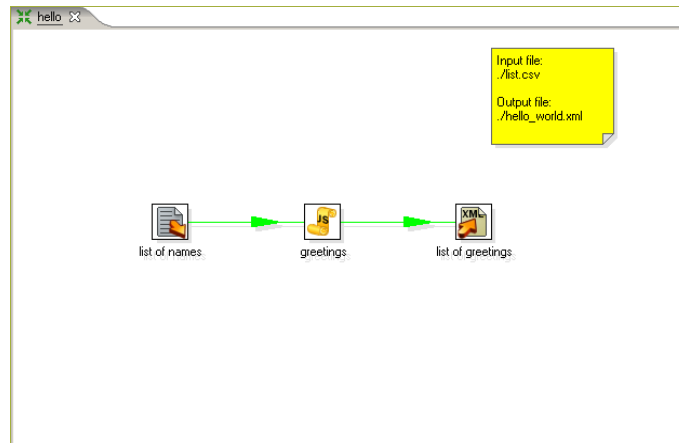


–Identify the palette the **Output** category. Click the **XML Output** step and drag it too.



–Link **CSV file input** with **Modified Java Script Value** by creating a Hop: Click the first Step and holding the **<Shift>** key pressed, drag the cursor towards the second Step. Release the button. (The Spoon User Manual explains other ways to do the same)
 –Link **Modified Java Script Value** with **XML Output** following the same indications.

You will have the following screen:



Subtask 3. Configuring the Steps in order to specify their behavior

Every Step has a **configuration window**. These windows vary according to the functionality of the Steps and the category to which they belong. However, all of them have this in common:

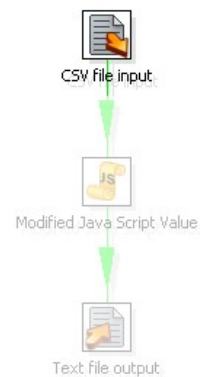
Step Name is a representative name inside the Transformation.
Step Description allows you to clarify the purpose of the Step.


a) Configuring the CSV file input Step

–Double click on the **CSV file input** Step. The configuration window belonging to this kind of Step will appear. Here you'll indicate the location, format and content of the input file.

–**Step name**: The first thing you should do *in any Step you configure* is to erase the default name and put a more representative one, according to the function that the Step is going to accomplish within the Transformation. In this case, you will put "name list".

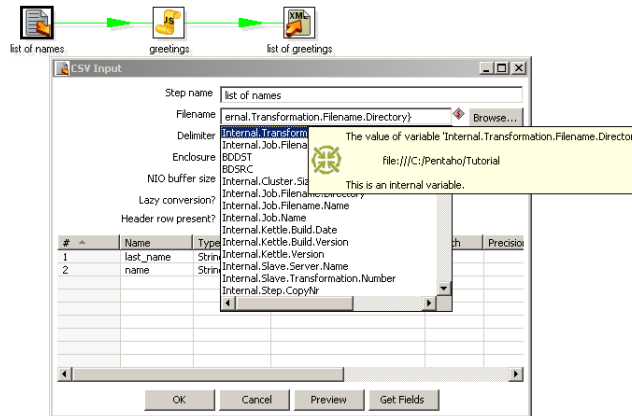
–**Filename**: Here you have to put the name and location of the input file. Just to the right of the text box, you'll see a symbol with the dollar sign.



This symbol  to the right of any text box implies that in that box, you could use **variables** as well as plain text.

A variable can be written manually as **`${name_of_the_variable}`** or selected from the variable window, to which you access pressing `<Ctrl-Space>`. The window shows predefined variables as well as user-defined variables.

You haven't created any variables yet, so in this case, you'll only see predefined variables:



Among this, select:

```
${Internal.Transformation.Filename.Directory}
```

Beside the name of the variable, write the name of the file you created. The text will become:

```
${Internal.Transformation.Filename.Directory}/list.csv
```

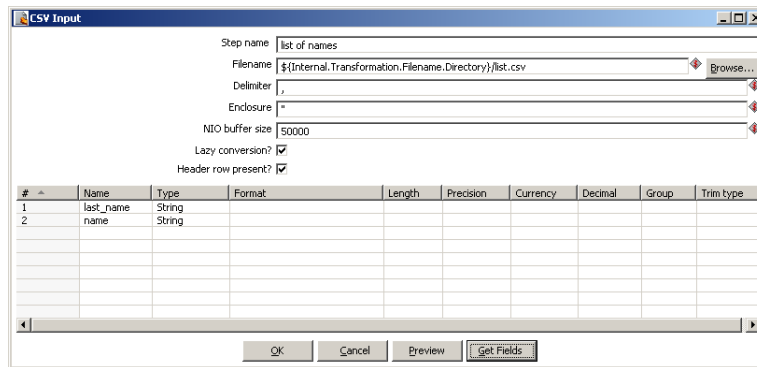
In runtime, the variable will be replaced by its value, which will be the path where the Transformation was saved. The Transformation will search the file `list.csv` in that location.

–**Get Fields**: Press this button to ask the application to bring to the grid the list of column names of the input file. Observe that, by default, the Step consider that the file has headers (**Header row present** is checked).

The button **Get Fields** is present in the configuration window of many Steps. Its purpose is to load a grid with data coming from external sources (i.e. files) or previous Steps. Even when the fields can be written manually, this button gives you a shortcut when there are many available fields and you want to use all or almost all of them.

The grid has now the names of the columns of your file: `last_name` and `name`.

The screenshot should look like this:



–**Preview Rows**: Just to be sure that the file will be read as expected, press the Preview button. A window showing data from the file will appear: so many rows as you ask (except that the number of rows in the file were less).

–**OK**: Press this button, and you are done with the Step **CSV file input**.

b) Configuring the Modified Java Script Value Step

–Double click on the **Modified Java Script Value** Step. The configuration window for this kind of Step will appear. The window is, indeed, very different from that which you saw in the previous Step. In this case, the Step allow you to write JavaScript code, and you will use it to build the message “Hello, ” concatenated with each of the names.

–**Step name**: Name this Step “Greetings”.

–**Java script**: The main area of the configuration window is for coding. To the left, there is a tree with a set of available functions that you can use in the code. In particular, the last two branches have the input and the output fields, ready to use in the code. In this example there are two fields: `last_name` and `name`. Write the following code:

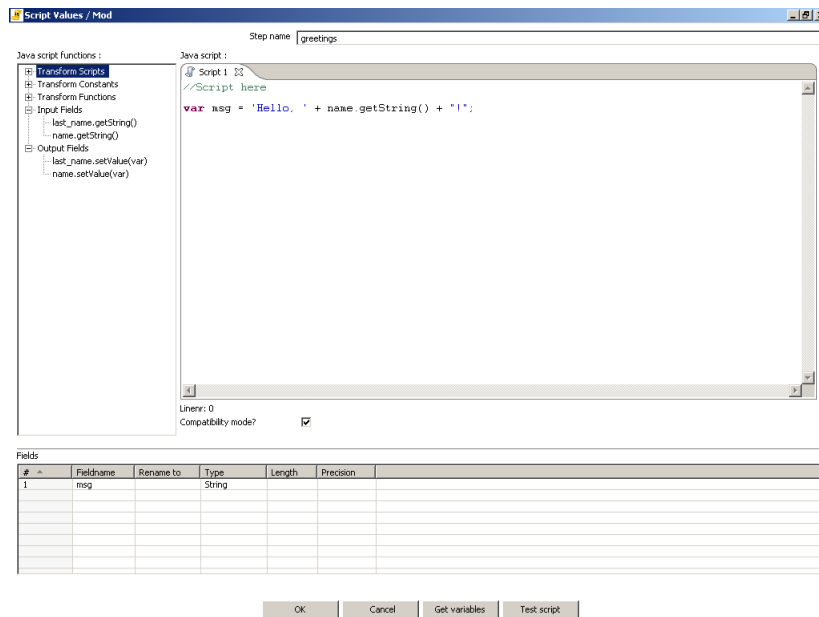
```
var msg = 'Hello, ' + name.getString() + "!";
```

The text `name.getString()` can be written manually, or by double clicking on the text in the function's tree.

–**Fields**: At the bottom you can put any variable created in the code that you wish to add to the output fields. In this case, you have created the variable `msg`. (**Don't mix this variables with PDI variables! They are not the same.**) As you need to send this message to the output file, you have to write the variable name in the grid.

You have configured the Step, and have this result:





Just to know! "modified" is not an adjective for "JavaScript", but for the Step. you are not dealing with a variant of JavaScript. Is the Step itself that is modified: It is an enhanced version of the original Step, which you found in previous versions of PDI.

–OK: Press this button, and the Step **Modified Script Value** is configured.

–Select the Step you just configured. In order to check that the new field will leave this step, you will see now the Input and Output Fields.

Input Fields: The data columns that reach a Step are called Input Fields.

Output Fields: The data columns that leave a Step are called Output Fields.

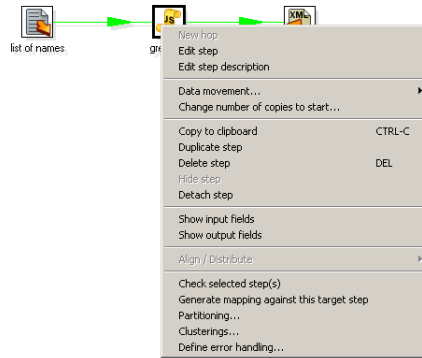
There are Steps that simply transform the input data. In this cases, the input and output fields are the same.

However that is not always the case:

There are Steps that add fields to the Output, for example: **Calculator**

There are other Steps that filter or combine data causing that the Output has less fields that the Input, for example: **Group by**

Pressing the right button just over the Step, a contextual menu appears:



- Select **Show Input Fields**. You'll see that the Input Fields are those coming from the **CSV file input** Step: last_name and name.
- Select **Show Output Fields**. You'll see that not only you have the existing fields, but also the new field: msg.

c) Configuring the XML Output Step

-Double click the **XML Output** Step. The configuration window for this kind of Step will appear. Here you're going to set the name and location of the output file, and to establish which of the fields you want to include. You may include all or some of the fields that reach the Step.

- Step name**: Name the Step "File with Greetings".
- File**: In this box write:

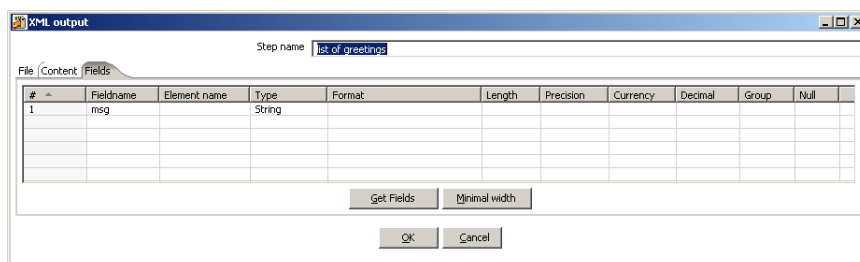
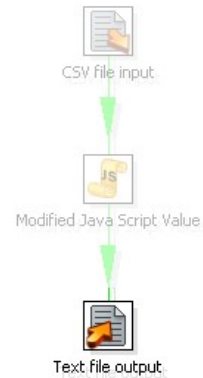
```

${Internal.Transformation.Filename.Directory}/Hello.xml

```

Remember that you can select the name of the variable from the variable's window pressing **<Ctrl-Space>**.

- Contents**: This tab allows you to configure the format of the output file. Leave the default.
- Fields**: Press the button **Get Fields**. The grid is filled with the three input fields. In the output file you only want to include the message, so delete name and last_name.



Don't forget to save the Transformation again.

How does it work?

When you execute a Transformation, almost all Steps are executed simultaneously. The Transformation execute **asynchronously**: The rows of data flow through the Steps at their own pace. Each processed row flow to the next Step without waiting for the others.

In real Transformations, forgetting this characteristic can be the main cause of unexpected results.

It's almost all configured. A Transformation read the input file, for each row (i.e. for each name) the JavaScript code create the message and finally the message is sent to the output file.

This example is too small and you have very few rows of names. Therefore is very difficult to notice the asynchronism in the execution. However, it always works in that way: It's possible that at the same time that a name is being written in the output file, another is leaving the first Step of the Transformation.

Verify, Preview and Execute!

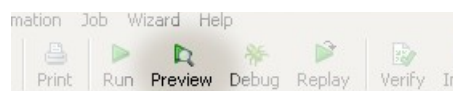
Before executing the Transformation, let's check that everything is ok. Press the **Verify** button.



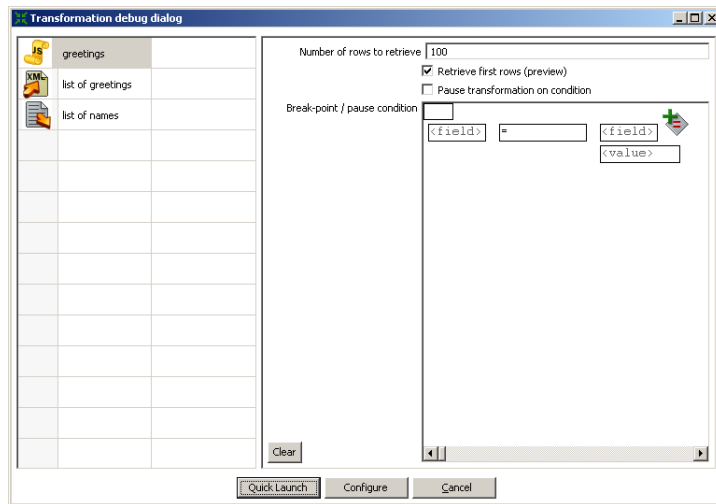
Spoon will verify if the Transformation is syntactically correct, looking for unreachable Steps, inexistent connections, etc.

If everything is ok (it should if you followed the indications) you are ready to preview the output.

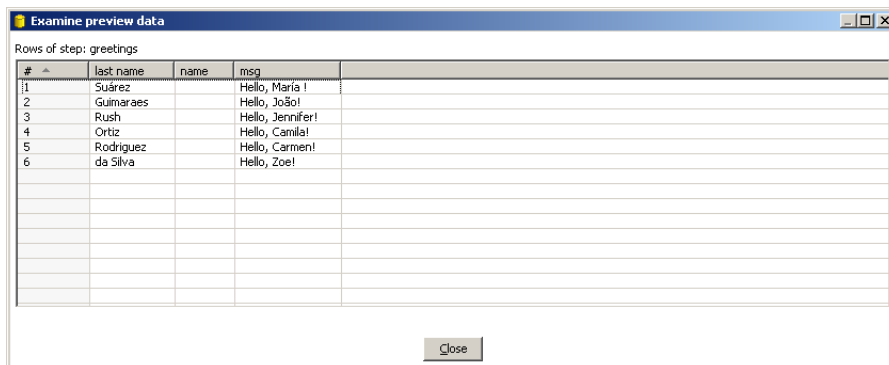
Select the JavaScript Step and then click the **Preview** button.



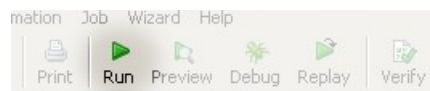
The following window will appear:



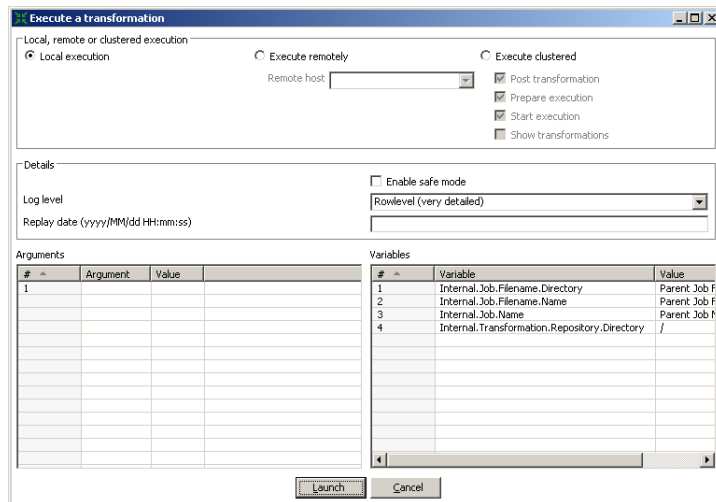
As you can see, Spoon suggests you to preview the selected Step. Press **Quick Launch**. After that, you will see a window with a sample of the output of the JavaScript Step: the messages created for each of the names coming from the input file.



If the output is what you expected, you're ready to execute the Transformation. Press the **Run** button.

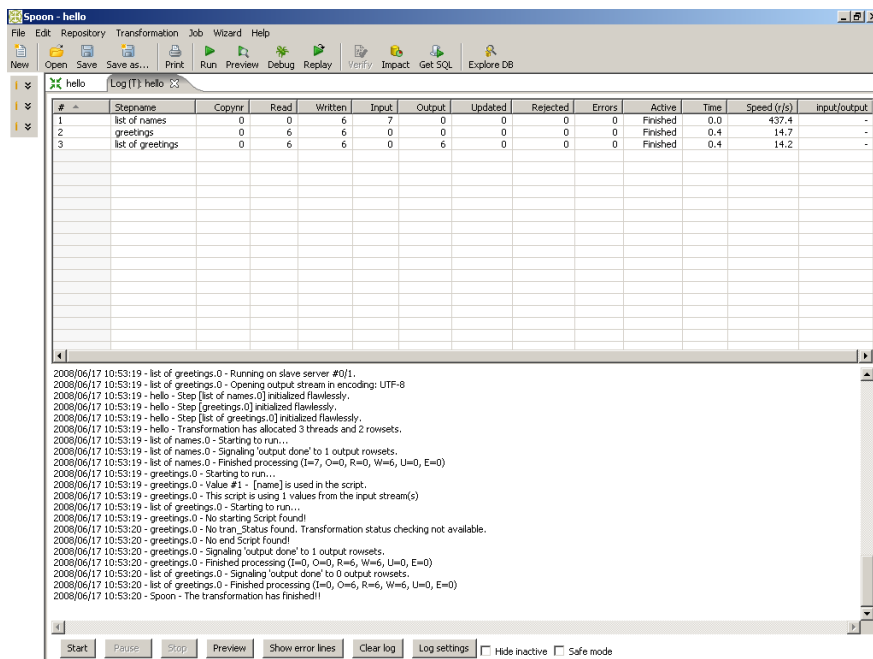


Spoon shows a window where you can set, among other information, the parameters for the execution and the logging level.



In this opportunity just press **Launch** at the bottom of the window.

Immediately you'll see a new tab window besides the Job window: This is the **log tab** containing the log of the current execution. (You will have noted that this tabbed window was also present when you did the preview).



The log tab has two sections:

In the upper side you can see, for each Step of the Transformation, the executed operations. In particular, pay attention to these:

- Read: contains the number of rows coming from previous Steps.
- Written: contains the number of rows leaving from this Step toward the next.
- Input: number of rows read from a file or table.
- Output: number of rows written to a file or table.
- Errors: errors in the execution. If there are errors, the whole row will become red.

In the lower side of the window, you will see the execution step by step. The detail will depend of the log level established. If you pay attention to this detail, you will see the asynchronism of the execution.

The last line of the text will be:

```
Spoon - The transformation has finished!!
```

If there weren't error messages in the text, you will be able to look for the generated file `Hello.xml` and check its content.

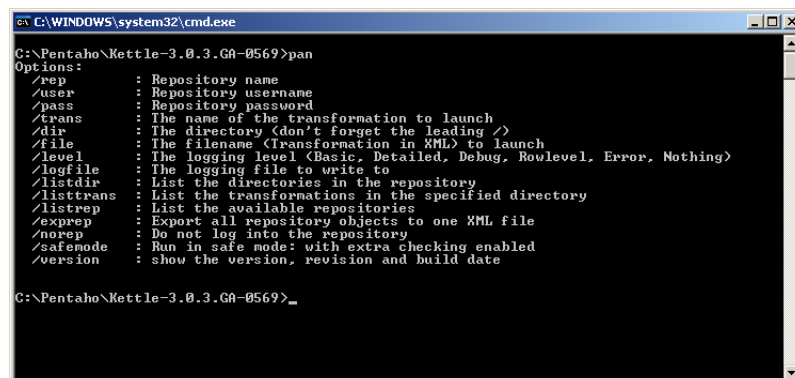
Pan

Pan is the tool that allows you to execute Transformations from a terminal window. The script is `pan.bat` (Windows) or `pan.sh` (other platforms), and you'll find it in the installation folder.

If you execute:

```
Pan
```

You'll see a description of the command with a list of the available options.



```
C:\WINDOWS\system32\cmd.exe
C:\Pentaho\Kettle-3.0.3.GA-0569>pan
Options:
/rep      : Repository name
/user     : Repository username
/pass    : Repository password
/trans   : The name of the transformation to launch
/dir     : The directory (don't forget the leading /)
/file    : The filename (Transformation in XML) to launch
/level   : The logging level (Basic, Detailed, Debug, Rowlevel, Error, Nothing)
/logfile  : The logging file to write to
/listdir  : List the directories in the repository
/listtrans : List the transformations in the specified directory
/listrep  : List the available repositories
/exprep  : Export all repository objects to one XML file
/norep   : Do not log into the repository
/safemode : Run in safe mode: with extra checking enabled
/version  : show the version, revision and build date

C:\Pentaho\Kettle-3.0.3.GA-0569>_
```

To execute your Transformation, try the simplest command:

```
Pan /file <Jobs_path>/Hello.ktr /norep
```

`/norep` is a command to ask Spoon not to connect to the repository.

`/file` precedes the name of the file corresponding to the Transformation to be executed.

`<Jobs_path>` is the full path of the folder Tutorial, for example:

```
c:/Pentaho/Tutorial (Windows)
```

or

```
/home/PentahoUser/Tutorial
```

The other options (i.e. log level) take default values.

After you enter this command, the Transformation will be executed quite in the same way it did inside Spoon. In this case, the log will be written in the same terminal, unless you redirect the log to a file. The format of the log text will vary a little, but the information will be basically the same that you saw in the graphical environment.

```
C:\WINDOWS\system32\cmd.exe
C:\Pentaho\Kettle-3.0.3.GA-0569>Pan /file c:/Pentaho/Tutorial/Hello.ktr /norep
INFO 23-06 16:09:06.910 <LogWriter.java:println:406> -Pan - Start of run.
2008/06/23 16:09:09:137 GMT-03:00 [INFO] DefaultFileReplicator - Using "C:\DOCUMENTS\1\AR2135\1\CONFIG\1\Temp\ufs_cache" as
temporary files store.
INFO 23-06 16:09:09.731 <LogWriter.java:println:406> -hello - Dispatching started for transformation [hello]
INFO 23-06 16:09:09.731 <LogWriter.java:println:406> -hello - No of arguments detected:0
INFO 23-06 16:09:09.731 <LogWriter.java:println:406> -hello - This is not a replay transformation
INFO 23-06 16:09:09.746 <LogWriter.java:println:406> -hello - This transformation can be replayed with replay date: 20
08/06/23 16:09:09
INFO 23-06 16:09:09.746 <LogWriter.java:println:406> -hello - Initialising 3 steps...
INFO 23-06 16:09:09.762 <LogWriter.java:println:406> -list of greetings.0 - Opening output stream in encoding: UTF-8
INFO 23-06 16:09:09.777 <LogWriter.java:println:406> -greetings.0 - Starting to run...
INFO 23-06 16:09:09.777 <LogWriter.java:println:406> -list of greetings.0 - Starting to run...
INFO 23-06 16:09:09.777 <LogWriter.java:println:406> -list of names.0 - Starting to run...
INFO 23-06 16:09:09.809 <LogWriter.java:println:406> -list of names.0 - Finished processing (I=7, O=0, R=0, W=6, U=0,
E=0)
INFO 23-06 16:09:10.356 <LogWriter.java:println:406> -greetings.0 - Finished processing (I=0, O=0, R=6, W=6, U=0, E=0)
INFO 23-06 16:09:10.356 <LogWriter.java:println:406> -list of greetings.0 - Finished processing (I=0, O=6, R=6, W=6, U
=0, E=0)
INFO 23-06 16:09:10.418 <LogWriter.java:println:406> -hello - Transformation ended.
INFO 23-06 16:09:10.418 <LogWriter.java:println:406> -Pan - Finished!
INFO 23-06 16:09:10.418 <LogWriter.java:println:406> -Pan - Start=2008/06/23 16:09:09.356, Stop=2008/06/23 16:09:10.41
8
INFO 23-06 16:09:10.418 <LogWriter.java:println:406> -Pan - Processing ended after 1 seconds.
INFO 23-06 16:09:10.418 <LogWriter.java:println:406> -hello -
INFO 23-06 16:09:10.418 <LogWriter.java:println:406> -hello - Process greetings'.0 ended successfully, processed 6 lin
es. < 6 lines/s>
INFO 23-06 16:09:10.418 <LogWriter.java:println:406> -hello - Process list of greetings'.0 ended successfully, process
ed 6 lines. < 6 lines/s>
INFO 23-06 16:09:10.418 <LogWriter.java:println:406> -hello - Process list of names'.0 ended successfully, processed 0
lines. < 0 lines/s>
C:\Pentaho\Kettle-3.0.3.GA-0569>
```

That's it. A simple Transformation that allowed you to meet the basic elements needed to work with PDI

Hello World again!

Now that the Transformation has been created and executed, the next task will be enhancing it. In this part of the tutorial you'll meet some new concepts, as important as those seen in the first part of this tutorial:

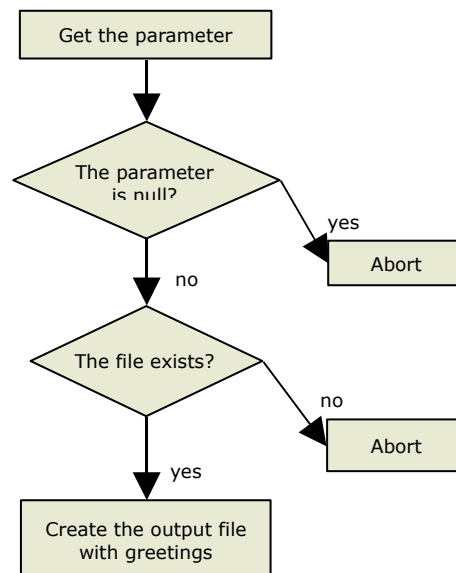
- Jobs
- Job Entries
- Job Hops
- Using Parameters and System Information
- User-defined Variables
- Configuration file kettle.properties
- Executing Jobs from a terminal window with the Kitchen tool.

Task

These are the improvements that you'll make to your previous task:

- You won't look for the input file in the same folder, but in a new one, a folder independent to that where the Transformations are saved. The name of the input file won't be fixed; the Transformation will receive it as a parameter.
- You will validate the existence of the input file (exercise: execute the Transformation you created, setting as the name of the file, a file that doesn't exist. See what happens!)
- The name the output file will be dependent of the name of the input file.

Let's design a simple diagram with the task to be accomplish:



You're going to implement this diagram with a Job.

Before starting, let's see some new definitions:

A **Job** is a component made by **Job Entries** linked by **Hops**. These Entries and Hops are arranged according to the expected order of execution. Therefore it is said that a Job is **flow-control** oriented.

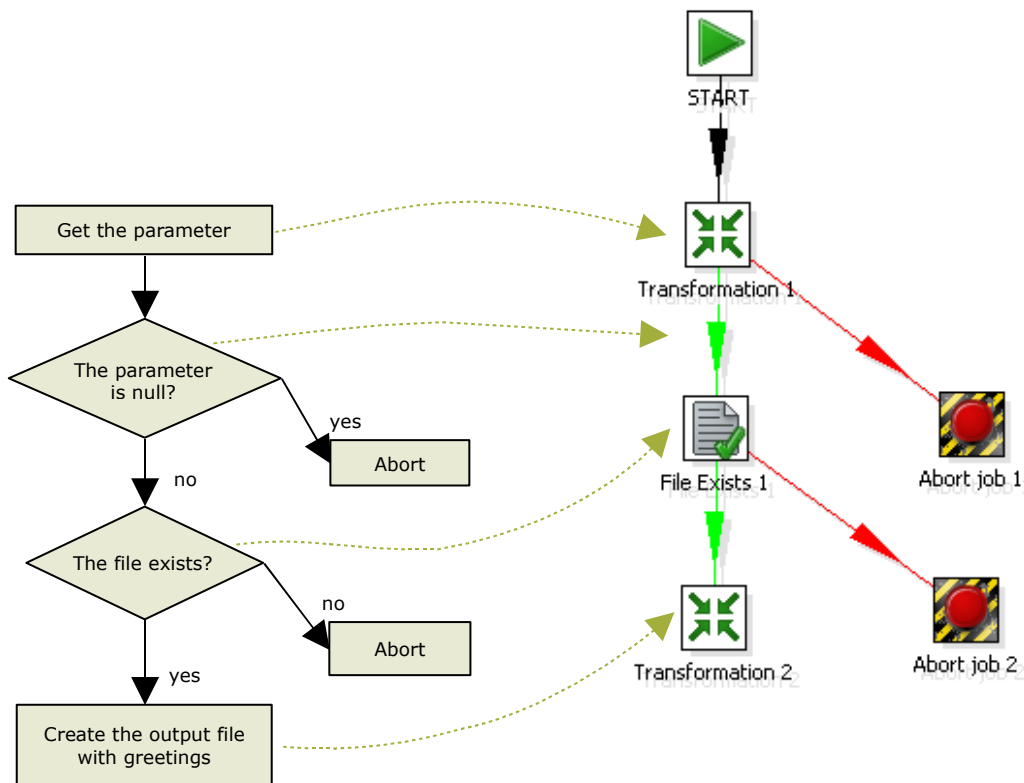
A **Job Entry** is a unit of execution inside a Job. Each Job Entry is conceived to accomplish a specific function ranging from verifying the existence of a table, to sending an email.

From a Job it is possible to execute a Transformation or another Job, that is, **Job** and **Transformation** are also Job Entries.

A **Hop** is a graphical representation that identifies the sequence of execution between two Job Entries.

Even when a Hop has only one origin and one destination, a particular Job Entry can be reached by more than a Hop, and more than a Hop can leave any particular Job Entry.

Just as you did with the first Transformation, let's make a correspondence between the elements in the diagram and the Job Entries and Hops that you will create.



- Getting the parameter will be resolved by a new Transformation
- The verification of the existence of the parameter will be made by using the result of the new Transformation. Acting according to that result will be accomplished by using the conditional execution of the next Steps.
- The verification of the existence of the file will be made by a Job Entry specific for that function.
- Executing the main task of the Job will be made by a variation of the Transformation you made in the first part of this tutorial.

Preparing the Environment

For this part of the tutorial, the input and output files will be in a different folder. Create then the folder `Files` in the path of your choice. Copy to this folder the file `list.csv` or create another, taking into account its format. Rename it if you wish.

In order to avoid writing the full path each time you need to reference the folder or the files, you will create a variable containing this information. The variable will be always available. For doing that, edit the configuration file **kettle.properties**. This file is located in the following folder

`C:\Documents and Settings\\.kettle\ (Windows)`

or

`$HOME/.kettle (other platforms)`

It is created at the same time that the folder `.kettle` is: the first time you execute Spoon.

At the end of the file `kettle.properties`, write this line:

```
FILES=<path_of_the_created_folder>
```

For example:

```
FILES=c:/Pentaho/Files
```

or

```
FILES=/home/PentahoUser/Files
```

Save the file.

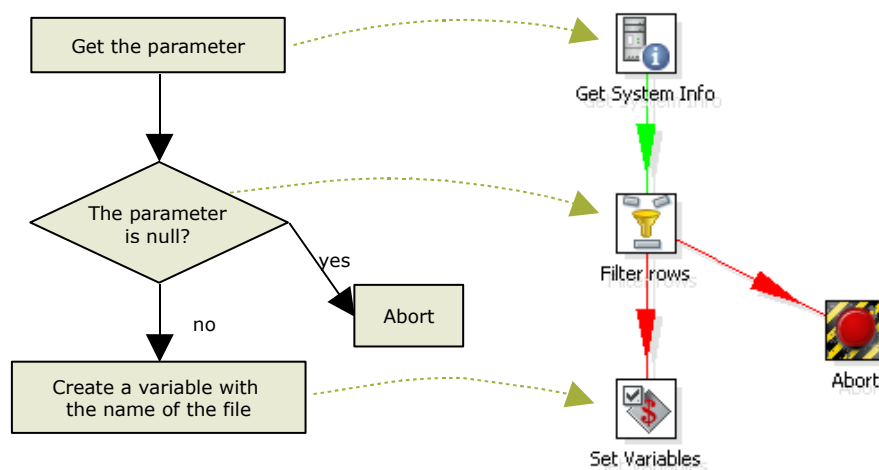
Spoon read the file when it starts, so, to get the new variable `FILES` available, restart Spoon.

Now you are ready to start. The full task will be developed in three stages:

- A. Creating the Transformation which takes the parameter
- B. Modifying the Transformation `Hello.ktr`.
- C. Building the Job according to the diagram.


A. Creating the transformation which takes the parameter


This new Transformation is quite simple. This diagram shows the task involved and their correspondence with PDI Steps:





Step by Step

1. Creating the Transformation: Create a new Transformation in the same way you did before. Name this Transformation `"get_file_name"`.
2. Drag to the workspace the following Steps, name them and link them according to the diagram.

 Get System Info (Input category)

 Filter Rows (Transform category)

 Abort (Transform category)

 Set Variable (Job category)

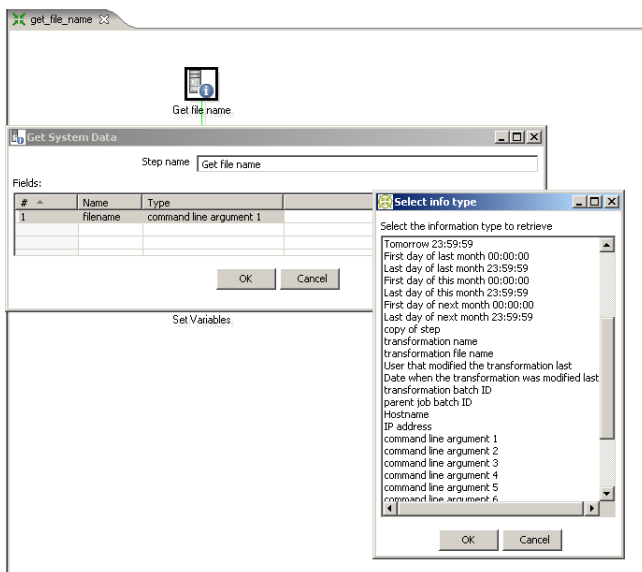
3. Configure the Steps as explained below.

a) Configuring the Get System Info Step (Input category)

This Step allow to capture information coming from sources outside the Transformation, like system date or parameters entered as part of the command line. In this case, you will use the Step to get the first and only parameter.

The configuration window of this Step has a grid. In this grid, each row you fill, will become a new column containing system data.

- Double click the Step.
- In the first cell, below the **Name** column, write "my_file".
- When you click the cell of the same row below **Type**, a window will show up with the available options. Select "command line argument 1".



-Click **OK**.

b) Configuring the Filter Rows Step (Transform category)

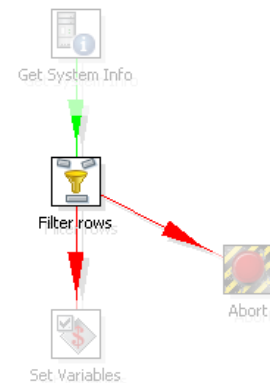
This Step divides the output in two, based upon a condition. Those rows for which the condition evaluates to true follow one path in the diagram, the others follow another.

-Double click the Step

-Write the condition: In **Field** select "my_file" and replace the sign "=" by "IS NULL".

-In the drop down list beside "Send 'true' data to Step" select **Abort**.

-In the drop down list beside "Send 'false' data to Step" select **Set Variable**



Filter rows

Step name: Filter rows

Send 'true' data to step: Abort

Send 'false' data to step: Set Variables

The condition:

filename	IS NULL	-	-
----------	---------	---	---

OK Cancel

-Click **OK**.

Now, a NULL parameter will reach the "Abort" Step. A NOT NULL parameter will reach the "Set Variable" Step.

c) Configuring the Abort Step (Transform category)

In fact, you don't have anything to configure in this Step. If a row of data reaches this Step, the Transformation aborts. Then the Transformation fails. you will use that result in the main Job.

d) Configuring the "Set Variable" Step ("Job" category)

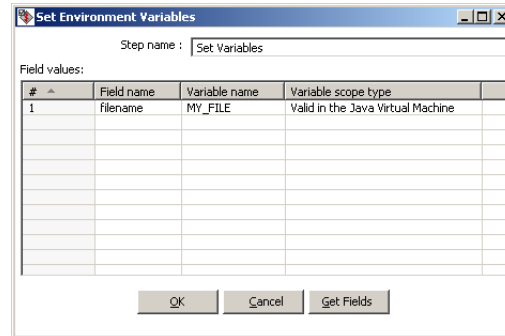
This Step allows you to create variables and assign it the content of some of the input fields.

The configuration window of the Step has a grid. Each row in this grid is meant to hold a new variable.

-Double click the Step



-Press the **Get Fields** button. The only existing field will appear: `my_file`. As the name for the new variable, Spoon set a default value: the name of the selected field, in upper case: `MY_FILE`. Leave the default. You will have this window:



-Click **OK**.

So, you created the variable `MY_FILE`, which will be used later.

Executing

Let's test the Transformation. Press the **Run** button.

In the first window observe the grid **Parameters**. You can use this grid to supply the parameter you would write in the command line. In the first row, in the column **Value**, write `"list"`.

Press the **Launch** button.

Observing the log, you'll see a text like this:

```
Set Variables.0 - Set variable MY_FILE to value [list]
```

Press **Execute** again, but don't write any parameters. This time, you'll see this:

```
Abort.0 - Row nr 1 causing abort : []
Abort.0 - Aborting after having seen 1 rows.
```

In the upper side of the window, you will see that the Step **Abort** indicates that an error occurred. This tells you that the Transformation failed, as expected.

B. Modifying the transformation Hello.ktr

Let's modify the Transformation `Hello` in order to make the names of the files depending of a parameter. If the parameter were `xxx`, the Transformation would read the file `xxx.csv` and create the file `xxx_with_greetings.xml`. And, as a plus, let's add a filter to discard the empty rows in the input file.

Step by Step

- Open the Transformation `Hello.ktr`
- Open the configuration window of the **CSV File Input** Step.
- Filename**: Delete the content of this text box, and press **<Ctrl-Space>** to see the list of existing variables. You'll see, in the list, the variable you added to

kettle.properties: FILES. Select it and add the name of the variable created in the previous Transformation. The text become:

```
${FILES}/${MY_FILE}.csv
```

-Press **OK**.

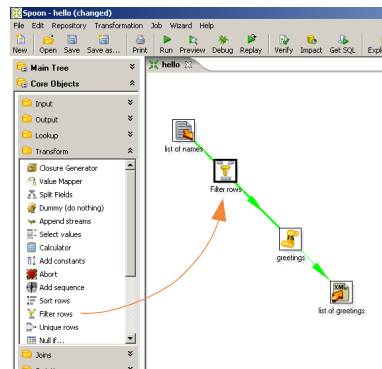
-Open the configuration window of the **XML Output** Step.

-**Filename**: Replace the content of the text box by this:

```
${FILES}/${MY_FILE}_with_greetings.xml
```

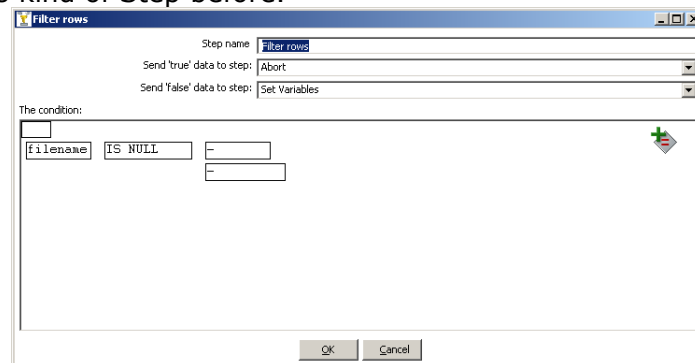
-Press **OK**.

-Drag to the workspace a **Filter Rows** Step without releasing it until it is over the Hop that leaves **CSV Input**. When you see that the Hop become wider, release the button.



- You will have linked the new Step to the sequence of existent steps. Write the condition: As **Field** select "name" and as comparator select "IS NOT NULL".

Leave "Send 'true' data to Step" and "Send 'true' data to Step" blank. This will cause that only the rows that fulfill the condition (i.e. rows with not null names) follow to the next Step. This is a variant to the way you used this kind of Step before.

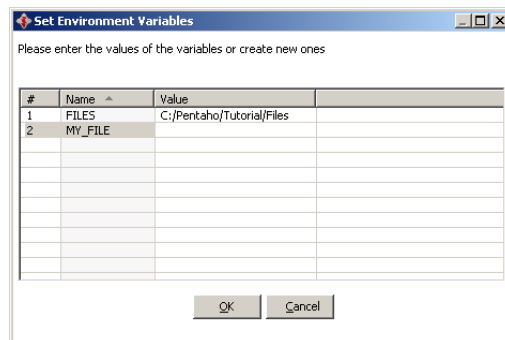


-Press **OK**.

-As you want to keep the original version of the Transformation, press **Save As** and name the Transformation "Hello_with_parameters". The file Hello_with_parameters.ktr will have been created.

Executing the Transformation

To test the changes you made, you need that the variable `MY_FILE` exists and has a value. As this Transformation is independent of the Transformation that creates the variable, in order to execute it, you'll have to create the variable manually. Select **Set Environment Variables** from the menu **Edit**. You'll see the existent variables. At the bottom of the list, write `MY_FILE` as the name of the variable, and, as the content, the name of your file without the extension! (i.e. `list`)



Press **OK**.

The, press **Run**. In the list of variables, you'll see yours. Press the **Launch** button, and let the Transformation do its task. After that, verify that the file:

```
<your_file>_with_greetings.xml
```

(being `<your_file>` the name of the input file) had been created in the folder designated to the tutorial files. Also verify the content of the file.

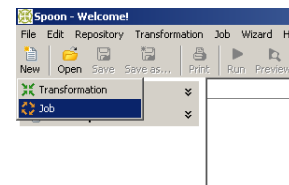
C. Building the main job

The last task of this part of the tutorial is the construction of the main job.

Step by Step

1)Creating the Job:

a)Select **New** → **Job**. You may also create a Job from the menu **Files** → **New** → **Job** or by pressing **<Ctrl-Alt-N>**. The screen is now occupied by the Job workspace, where you will drop Job Entries and Hops.



b)Select the option **Job** → **Configuration**. A window appears where you can specify some Job properties. Write a name and a description.

c)Press the **Save** button. Save the Job in the `Tutorial` folder, under the name `Hello`. The file `Hello.jkb` will have been created.

2) Building the skeleton of the Job with Job Entries and Hops: To the left of the workspace there is a Palette of Job Entries. Unlike the palette of Transformation Steps, this palette doesn't group the Entries into categories¹.

Let's build the Job:

a) Drag to the screen this Job Entries, name them and link them as in the diagram:



Start



Transformation x 2



File Exists

b) Drag to the screen the following entries



Abort Job x 2

name them and link them to the other entries, as in the diagram. You'll see that the Hops became red. In a while you'll know why.

c) Configure the Steps as explained below.

a) Configuring the first of the two Transformation Job Entries

The purpose of the **Transformation** entry – as you may have guessed – is to execute a Transformation. Let's configure this entry to execute the Transformation that gets the parameter.

-Double click the entry. The configuration window appears.

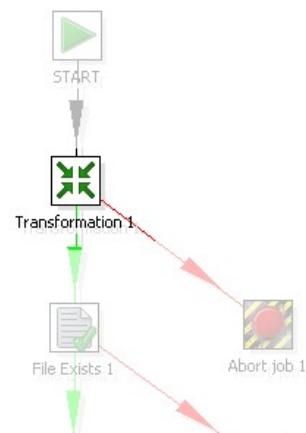
-**Transformation filename:** Here you have to write the name of the Transformation file:

"get_file_name.ktr". As the Transformations and the Jobs are in the same folder, in order to locate the Transformation file, you can use the Job path as a reference. Write this string:

```
${Internal.Job.Filename.Directory}/get_file_name.ktr
```

The variable can be written manually, or selected from the variable window. The file name can also be written manually, or you can search it with the **Browse** button.

-Press **OK**.



¹In the 3.1.0 version– not released yet - the Job Entries are grouped in Categories as General, Mail, File management, etc.

b) Configuring the second of the two Transformation Job Entries

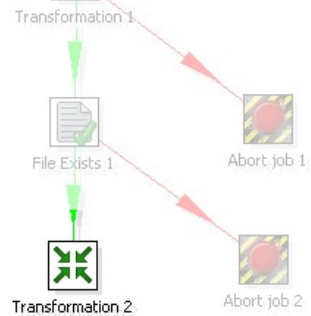
This entry will execute the main function of the Job.

-Double click the entry. The configuration window appears.

-**Transformation filename**: In this case, enter the name of the other Transformation:

```
${Internal.Job.Filename.Directory}/Hello_with_parameter.ktr
```

-Press **OK**.



c) Configuring the File Exists Job Entry

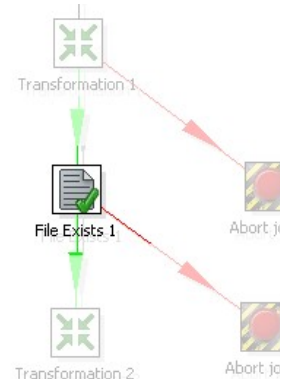
The **File Exists** entry verifies the existence of a file. The entry has a successful result only if the file exists.

-Double click the entry. The configuration window appears.

-**Filename**: Here you have to put the complete path of the file whose existence you want to verify. The name is the same that you wrote in the modified Transformation Hello:

```
${FILES}/${MY_FILE}.csv
```

Remember that the variable `${FILES}` was defined in the `kettle.properties` file and the variable `${MY_FILE}` was created in the Job Entry that is going to be executed before this one.

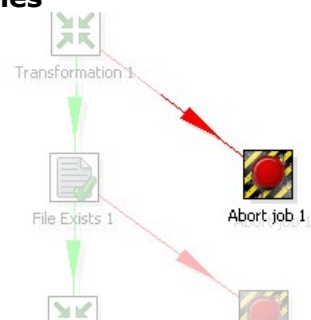


d) Configuring the first of the two Abort Job Job Entries

The first **Abort Job** entry will be executed when there wasn't a parameter in the command line. What you will configure here is the message containing the cause of the abortion.

-In the **Message** textbox write:

```
The file name is missing
```



e) Configuring the second of the two **Abort Job** Job Entries

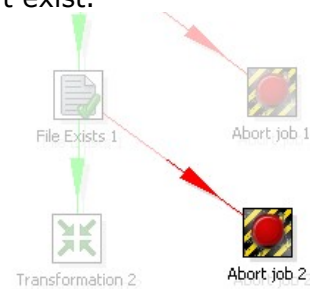
This **Abort Job** entry will be executed when the file doesn't exist.

-In the **Message** textbox write this text:

```
The file ${FILES}/${MY_FILE}.csv  
does not exist
```

In runtime, the tool will replace the variable names by its values, showing for example:

```
The file c:/Pentaho/Files/list.csv  
does not exist
```



f) Configuring the Hops

A Job Entry can be executed:

- unconditionally: it's executed always
- when the previous Job Entry was successful
- when the previous Job Entry failed

This execution is represented by different colors in the Hops:

- a **black** Hop indicates that the following Job Entry is always executed
- a **green** Hop indicates that the following Job Entry is executed only if the previous Job Entry was successful
- a **red** Hop indicates that the following Job Entry is executed only if the previous Job Entry failed

As a consequence of the order in which the Job Entries of your Job were created and linked, all of the Hops took the right color, that is, the Steps will execute as you need:

- The first **Transformation** entry will be always executed (The Hop that goes from **Start** toward this entry, is **black**)
- If the Transformation that gets the parameter doesn't find a parameter, that is, the Transformation failed, the control goes through the **red** Hop towards the **Abort Job** entry.
- If the Transformation is successful, the control goes through the **green** Hop towards the **File Exists** entry.
- If the file doesn't exist, that is, the verification of the existence fails, the control goes through the **red** Hop, towards the second **Abort Job** entry.
- If the verification is successful, the control goes through the **green** Hop towards the main **Transformation** entry.

If you wished to change the condition for the execution of a Job Entry, the steps to follow would be:

- Select the Hop that reach this Job Entry
- Right click the mouse, bringing a contextual menu
- In the menu, select **Evaluation** followed by one of the three available conditions.

How does it work?

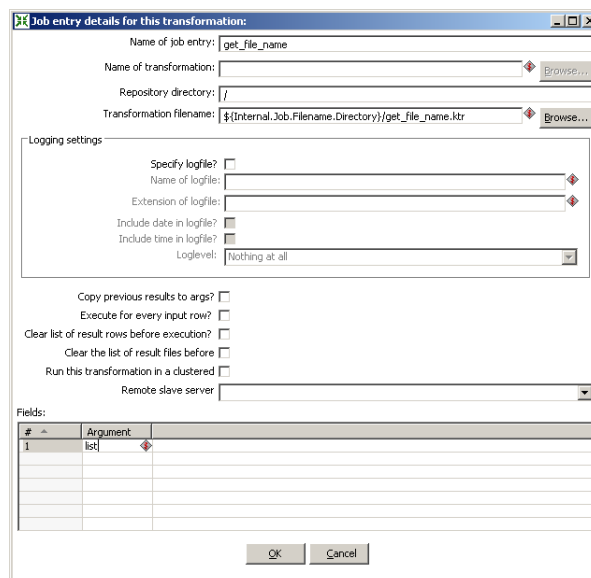
When you execute a Job, the execution is tied to the order of the Job Entries, the direction of the Hops, and the condition under which an entry is or not executed. The execution follows a **sequence**. The execution of a Job Entry cannot begin until the execution of the Job Entries that precede it has finished.

In real situations, a Job can be a solution to solve problems related with sequence of tasks in the Transformations: If you need that a part of a Transformation finishes before another part begins, a solution could be to divide the Transformation in two independent Transformations, and execute them from a Job, one after the other.

Executing the Job

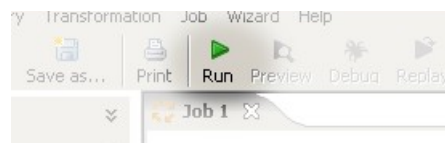
To execute a Job, in the first place you need to supply the parameter. As the only place where the parameter is used is in the Transformation "get_file_name" (after that you only use the variable where the parameter is saved) let's write the parameter as follows:

- Double click the "get_file_name" Transformation.
- The appearing window has a grid named **Fields**. In the first row write the name of the file created in the Tutorial folder (without extension).



-Press **OK**.

Press the **Run** button.



A window appears with general information related with the execution of the Job. Press **Execute**.

The **Job log tabbed window** appears beside the Job window.

The tabbed windows corresponding to Jobs, as those corresponding to Transformations, are divided in two:

The upper half shows the Job Entries of your Job. For each executed Job Entry, you'll see, among other data, the result of the execution. As said, the execution of the entries follows a sequence. In consequence, if an entry fails, you won't see the entries that follow because they never start.

In the second half of the window, you can see the log detail, including the starting and ending time of the Job Entries. In particular, when an Entry is a Transformation, the log corresponding to the transformation is also included.

At the time you see at the end of the log the text:

```
Spoon - Job has ended.
```

the new file has been created. If the input file was: `list.csv`, the output file should be: `list_with_greetings.xml` and should be in the same folder. Find it and check its content.

Change the name of the parameter. Replace it by a name of an inexistent file. Execute the Job. You'll see that the Job aborts, and the log shows the following message:

```
Abort - The file <parameter> does not exist
```

Where `<parameter>` is the parameter you supplied.

Now try deleting the parameter, and executing the Job one more time. In this case the Job aborts as well, and in the log you can see this message:

```
Abort - The file name is missing
```

just as you expected.

Kitchen

Kitchen is the tool used to execute Jobs from a terminal window. The script is `Kitchen.bat` (Windows) or `Kitchen.sh` (other platforms), and you'll find it in the installation folder.

If you execute:

```
Kitchen
```

you'll see a description of the command with a list of the available options.

```
C:\WINDOWS\system32\cmd.exe
C:\Pentaho\Kettle-3.0.3.GA-0569>kitchen
Options:
/rep      : Repository name
/user     : Repository username
/pass    : Repository password
/job      : The name of the transformation to launch
/dir      : The directory (don't forget the leading /)
/file     : The filename (Job XML) to launch
/level   : The logging level (Basic, Detailed, Debug, Rowlevel, Error, Nothing)
/logfile  : The logging file to write to
/listdir  : List the directories in the repository
/listjobs : List the jobs in the specified directory
/listrep  : List the available repositories
/norep    : Do not log into the repository
/version  : show the version, revision and build date

C:\Pentaho\Kettle-3.0.3.GA-0569>_
```

To execute the Job, try the simplest command:

```
Kitchen /file <Jobs_path>/Hello.kjb <par> /norep
```

`/norep` is a command to ask Spoon not to connect to the repository.

`/file` precedes the name of the file corresponding to the Job to be executed.

`<Jobs_path>` is the full path of the folder Tutorial, for example:

```
c:/Pentaho/Tutorial (Windows)
```

or

```
/home/PentahoUser/Tutorial
```

`<par>` is the parameter that the Job is waiting. Remember that the expected parameter is the name of the input file, without the `csv`.

The other options (i.e. log level) take default values.

After you enter this command, the Job will be executed quite in the same way it did inside Spoon. In this case, the log will be written in the same terminal, unless you redirect the log to a file. The format of the log text will vary a little, but the information will be basically the same that you saw in the graphical environment.

Try to execute the Job:

–without parameters

–with an invalid parameter (an inexistent file)

–with a valid parameter

and verify that everything works as expected.

Also experiment the `Kitchen` command changing some of the options (i.e. log level).

And this is the end of the tutorial. You build a Job that enhanced the Transformation of the first part, and allowed you to know more elements of PDI, essential for those who work with this tool day by day.

Conclusion

Now that you saw an overview of PDI, you are ready to get benefit of the tool making your own projects.

For going beyond, I suggest you to turn to this sources:

■User Manuals

The user manuals are in the folder `\docs\English` inside the installation folder.

■Examples

The folder `Samples` inside the installation folder, contains several examples. This examples can be useful in order to understand how a particular Step works, o to take them as a starting point for your own Transformations and Jobs.

■Forum

The Pentaho forum is an interesting knowledge source. There you can see the problems that other people have faced and the way in which they can be resolved.

You can also expose doubts or problems you have, or help other people after you have acquired some experience with the tool.

You can reach the forum from this url:

<http://forums.pentaho.org/forumdisplay.php?f=69>

Besides this links, in the Welcome Screen you will find some extra sources.

I hope this tutorial had been useful for you to start working with PDI.

Comments, critics and suggestions are welcome,
Good luck!