Pentaho Data Integration version 2.5.0

Frequently Asked Questions

FAQ

Index

Pentaho Data Integration FAQ

1. Preface

This document contains the “Frequently Asked Questions” on Pentaho Data Integration, formerly known as KETTLE. The questions and answers in this document are mainly a summary of questions answered on the discussion forums. If a question is contained herein it probably doesn't make sense any more to ask it again on the discussion forum. However if an answer herein is not obvious ask for a further explanation on the forums referring to the specific FAQ question/answer.

2. Beginning user questions

2.1. Problems starting spoon

Q: When I start spoon I get the following error: “Could not find the main class. Program will exit”. How do I start spoon?

A: You can get the above message because of several reasons, the root cause is always that kettle.jar is not on the classpath.

- Assuming you downloaded the binary version of Pentaho Data Integration: check whether you extracted the zip file maintaining the directory structure: under the main directory there should be a directory called “lib” that contains a file called kettle.jar. If this is not the case re-extract the zip file in the proper way.
- When you fetched the sources of Pentaho Data Integration and compiled yourself you are probably executing the spoon script from the wrong directory. The source distribution has a directory called “bin” that contains the scripts, but if you compile the proper way the “distribution”-ready Pentaho Data Integration will be in a directory called “distrib”. You should start the spoon script from that directory.

2.2. What's the difference between transformations and jobs?

Q: In Spoon I can make jobs and transformations, what's the difference between the two?

A: Transformations are about moving and transforming rows from source to target. Jobs are more about high level flow control: executing transformations, sending mails on failure, ftp'ing files, ...

2.3. Rule on mixing row 'types' on a hop in a transformation

Q: In the manuals I read that row types may not be mixed, what does that mean?

A: Not mixing of rows means that every row which is sent over a single hop needs to be of the same structure: same fieldnames, types, order of fields. So if you want to do stuff like "add an extra field if condition is true for a row but not otherwise" it will not work (because you will get different type of rows depending on conditions). You can switch on "Enable safe mode" to explicitly check for this at runtime.

As of v2.5.0 checking of mixing of rows is also done automatically at design/verify time, but “Enable safe mode” still needs to be switched on to check it at runtime (as this causes a slight processing overhead).

A way to look at this is that a hop is very similar to a database table in some aspects, you also cannot store different type of rows in a database table. Theoretically the reason is that PDI wants to be able to do uniform/consistent transformations on your data and having variable rows makes this much more complex.

Technically most of the steps use optimization techniques which transformation column names into indexes into a row (e.g. The column with name “sid” is column 4), having variable row would make this not possible (and break most of the current steps).

2.4. On duplicate fieldnames in a transformation

Q: Why can't I duplicate fieldnames in a single row?

A: You can't. PDI will complain in most of the cases if you have duplicate fieldnames. Before PDI v2.5.0 you were able to force duplicate fields, but also only the first value of the duplicate fields could ever be used.

2.5. On empty strings and NULL

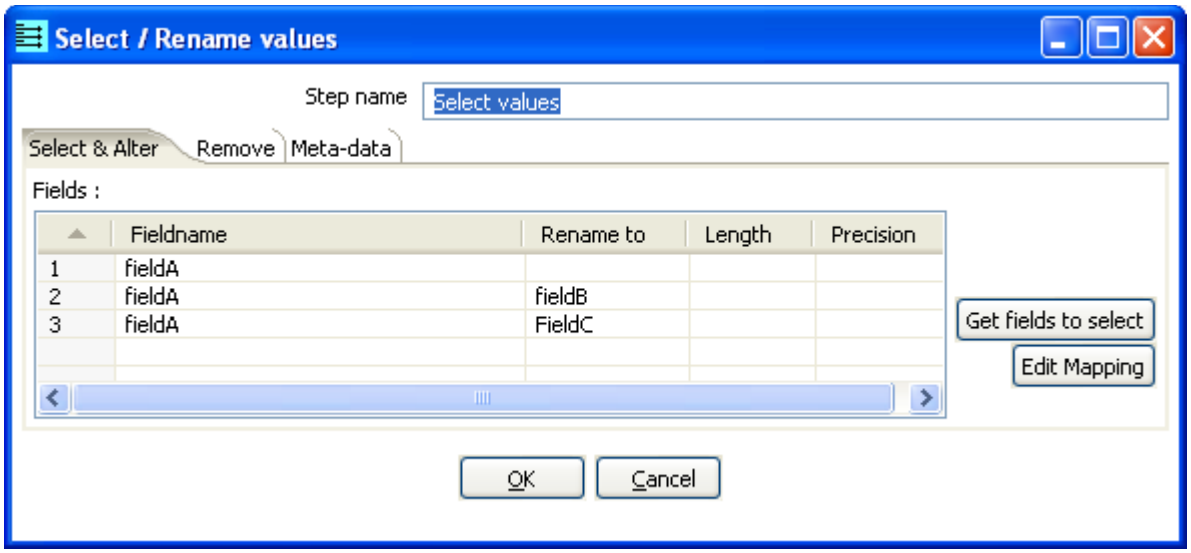
PDI follows Oracle in its use of empty string and NULLs: they are considered to the same. If you would find a step that doesn't follow this convention, let us knows since it's probably a bug.

2.6. How to copy/duplicate a field in a transformation?

Q: How do you duplicate a field in a row in a transformation?

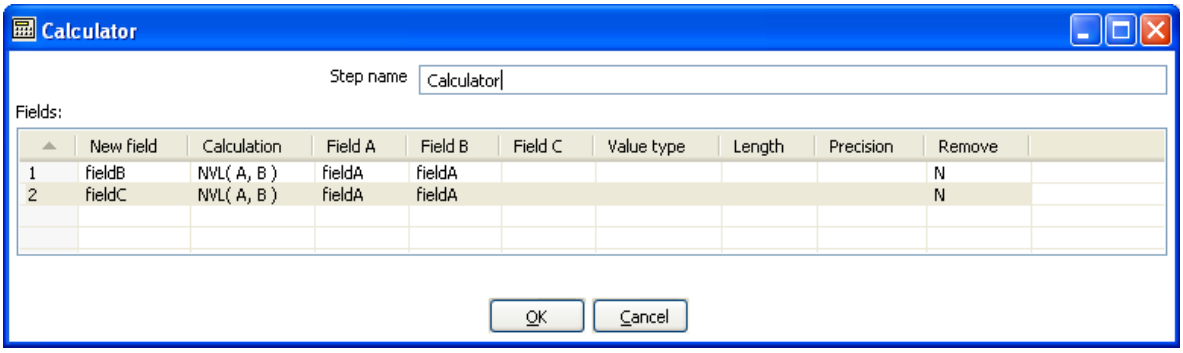
A: Several solutions exist:

1) Use a “Select Values” step renaming a field while selecting also the original one. The result will be that the original field will be duplicated to another name. It will look as follows:



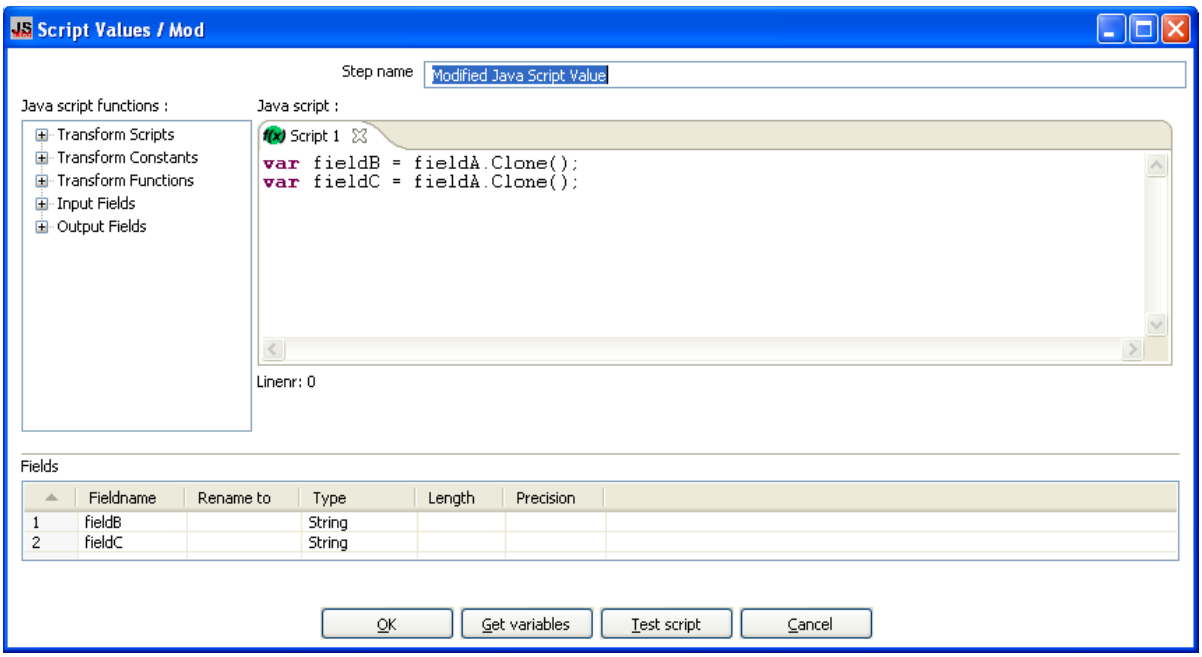
This will duplicate fieldA to fieldB and fieldC.

2) Use a calculator step and use e.g. The NLV(A,B) operation as follows:



This will have the same effect as the first solution: 3 fields in the output which are copies of each other: fieldA, fieldB, and fieldC.

3) Use a JavaScript step to copy the field:



This will have the same effect as the previous solutions: 3 fields in the output which are copies of each other: fieldA, fieldB, and fieldC.

2.7. How to do a database join with PDI?

Q: How do you do a database join with PDI?

A: If you want to join 2 tables from the same database, you use a “Table Input” step and do the join in SQL itself, in 1 database this the fastest solution.

If you want to join 2 tables that are not in the same database. You can use the the “Database Join” step, but you have to realize that PDI will launch a query for each input row from the main stream. You can find more information on this in 1 of the weekly tips.

2.8. How to sequentialize transformations?

Q: By default all steps in a transformation run in parallel, how can I make it so that 1 row gets processed completely until the end before the next row is processed?

A: This is not possible, one of the basic things in PDI transformations is that all of the steps run in parallel. So you can't sequentialize them. This would require architectural changes to PDI and sequential processing would also result in very slow processing.

3. Bug reports and feature requests

3.1. Preface

Lately we've been getting a lot of bug reports and feature requests.

While this is great and in general we like bug reports a lot there is always a high risk of some report or detail being lost in the noise when posted on a forum or sent through e-mail. People sometimes forget...

For this reason, we would like to ask you to post bug reports if you have even the slightest doubt that something is not working as it should.

The same goes for feature requests: if you have a good idea, why not share it with others AND put it on the developers TODO list.

If you think that somehow we will probably not implement it anyway, think again, last release we implemented over 50 registered feature requests.

The next major release of Pentaho Data Integration will probably feature an even more impressive change list.

3.2. The links to click...

To file a bug report go here: http://www.javaforge.com/proj/tracker/submitNew.do?tracker_id=1273

To file a change request, go here: http://www.javaforge.com/proj/tracker/submitNew.do?tracker_id=1274

To see a list of the open bugs in the latest development version, go here:

http://www.javaforge.com/proj/tracker/browseTracker.do?tracker_id=1273&reset=open&view_id=-1&pagesize=0

To see a list of the open change requests in the latest development version, go here:

http://www.javaforge.com/proj/tracker/browseTracker.do?tracker_id=1274&reset=open&view_id=-1&pagesize=0

3.3. What to put in a bug report?

In general, the rule is that the easier it is for us to find or reproduce the bug, the faster it gets fixed. Use your common sense, but in general it's always nice to have as many of the following as possible:

- a short but complete description of the problem
- a stack trace if any is available. From version 2.4.0 on, you need to click on the "Details" button in the error dialogs.
- a version number of Kettle, the build number & data are also nice. Use “pan -version” to obtain this information.
- sometimes a small transformation to show the problem can be really helpful, consider attaching it to the bug report.
- consider attaching a sample of the input file you used

3.4. What to put in a change request?

We always favor those change requests that have a clear and urgent use-case. If there are certain things that you need Pentaho Data Integration to do, explain this to us as clearly as possible in the change request. Something may be a lot of work, but we still might favor that particular feature request over any others because you convinced us it was more important.

3.5. A word of thanks

Pentaho Data Integration is advancing very fast at the moment, this is mainly possible because of the large group of users, testers and bug reporters.

Thank you for your continued support and for your help in making "Kettle" a world class data integration tool. Reporting bugs and filing feature requests really pushes PDI forward.

4. Source code access

4.1. Windows

As this is a question that comes back regularly, here are some instructions on how to get access to the latest PDI source code:

To get to the latest sources, you need to install a Subversion client, like Tortoise SVN.

The checkout can be done from the Windows explorer afther installation at the following URL:

```
svn://source.pentaho.org/svnkettleroot/Kettle/trunk
```

Create a directory, for Example "Kettle" and right click on that directory, pick "SVN Checkout..."

To update to the latest changes, pick “SVN Update...”

4.2. Linux

If you are running Linux, you need to install the "svn" Subversion client. The checkout command for the latest stuff is then:

```
svn checkout svn://source.pentaho.org/svnkettleroot/Kettle/trunk/
```

4.3. Web SVN access

Recently we have added subversion web access on the following URL:

```
http://kettle.pentaho.org/svn/Kettle/trunk/
```

4.4. Eclipse

4.4.1. Subclipse

If you are using Eclipse as an IDE, you can install the Subclipse Subversion plugin.

4.4.2. Creating patches

If you have code, fixes or changes or anything else to contribute to the project, you can make patches against the latest trunk version. Click right on the project or a folder --> Team -->Create patch... You can then simply send the patch to one of the developers for review / inclusion in the code base.

5. Further User questions

5.1. Strings bigger than defined String length

Q: I'm trying to use the Select values step to set the max length for a field. But I'm still seeing longer strings getting through, and this leads to SQL errors further on.

Isn't the Select Values step supposed to truncate strings to the correct size? Or must I truncate the string in some other way?

If so, what's the 'best' other way?

A: A transformation is all about metadata really. Early in the game the original Kettle developer decided not to annoy users with data type "errors" like these. "Select Values" too is all about metadata, not the data itself.

However, if you want to "truncate" a field, you are going to have to do implement some Javascript logic to force it like that.

```
var string = originalfield.getString();
if ( string.length() > 50 )
{
    string = string.substring(0,50);
}
```

5.2. Decimal point doesn't show in .csv output

Q: In a database table I have a single decimal column containing the following contents:

```
100.23
100.20
100.00
100.00
```

I output this table via a File output step (with extension .csv) and open it with Excel, I get the following:

```
100.23
100.20
100
100
```

How do I fix this and add the ".00" to the last 2 rows?

A: First of all in the File output step you have to use a format of "#.00" for the column. This will get you the a precision after the decimal point of 2 decimals on all of the values in the output file, if you open it with Notepad.

If you open the .csv file with Excel the ".00" will still not be there, the problem is that .csv output when opened in Excel uses a default formatting which hides the ".00". As of PDI version 2.4.0 there's an Excel output step that can do some basic Excel formatting.

5.3. Function call returning boolean fails in Oracle

Q: The return type Boolean for a db function gives an error "Invalid column type" in Pentaho Data Integration. My is as follows:

```
FUNCTION test(id number) RETURN BOOLEAN IS

BEGIN
         return true;
END
```

A: It's an Oracle thing. That sort of return value is only valid in a PL/SQL block. This is because an Oracle table can't contain a boolean data type (boolean is not part of the SQL standard). It's suggested that you return a varchar with 'true' / 'false' in it (or 'Y' / 'N'). If you then set the convert the data type to boolean you might find that you will get a boolean.

For a reference from the Oracle manuals on this behaviour:

> "It is not feasible for Oracle JDBC drivers to support calling arguments or return values of the PL/SQL `RECORD`, `BOOLEAN`, or table with non-scalar element types. However, Oracle JDBC drivers support PL/SQL index-by table of scalar element types. For a complete description of this, see Chapter 11, "Accessing PL/SQL Index-by Tables"
>
> As a workaround to PL/SQL `RECORD`, `BOOLEAN`, or non-scalar table types, create wrapper procedures that handle the data as types supported by JDBC. For example, to wrap a stored procedure that uses PL/SQL booleans, create a stored procedure that takes a character or number from JDBC and passes it to the original procedure as `BOOLEAN` or, for an output parameter, accepts a `BOOLEAN` argument from the original procedure and passes it as a `CHAR` or `NUMBER` to JDBC. Similarly, to wrap a stored procedure that uses PL/SQL records, create a stored procedure that handles a record in its individual components (such as `CHAR` and `NUMBER`) or in a structured object type. To wrap a stored procedure that uses PL/SQL tables, break the data into components or perhaps use Oracle collection types."

5.4. Difference between variables/arguments in launcher

Q: When running a Transformation, the dialog has two tables, one for Arguments and one for Variables. What is the difference between the two?

A: Arguments are command line arguments that you would normally specify during batch processing (via Pan). Variables are environment or PDI variables that you would normally set in a previous transformation in a job or in the Operating System.

5.5. How to use database connections from repository

Q: If you create a new database connection directly in the repository explorer, how can use it in a new or existing transformation?

A: Create a new transformation (or job) or close and re-open the ones you have loaded in Spoon.

5.6. How to use JNDI?

Q: We will be using PDI integrated in a web application deployed on an application server. We've created a JNDI datasource in our application server. Of course Spoon doesn't run in the context of the application server, so how can we use the JNDI data source in PDI?

A: If you look in the PDI main directory you will see a sub-directory “simple-jndi", which contains a file called “jdbc.properties”. You should change this file so that the JNDI information matches the one you use in your application server.

After that you set in the connection tab of Spoon the “Method of access” to JNDI, the “Connection type” to the type of database you're using. And “Connection name” to the name of the JDNI datasource (as used in “jdbc.properties”).

5.7. On inserting booleans into a MySQL database

Q: How do you insert booleans into a MySql database, PDI encodes a boolean as 'Y' or 'N' and this can't be insert into a BIT(1) column in MySql.

A: BIT is not a standard SQL data type. It's not even standard on MySQL as the meaning (core definition) changed from MySQL version 4 to 5.

Also a BIT uses 2 bytes on MySQL. That's why in PDI we made the safe choice and went for a char(1) to store a boolean.

There is a simple workaround available: change the data type with a Select Values step to "Integer" in the metadata tab. This converts it to 1 for "true" and 0 for "false", just like MySQL expects.

5.8. Calculator ignores result type on division

Q: I made a transformation using A/B in a calculator step and it rounded wrong: the 2 input fields are integer but my result type was Number(6, 4) so I would expect the integers to be cast to Number before executing the division.

If I wanted to execute e.g. 28/222, I got 0.0 instead of 0.1261 which I expected. So it seems the result type is ignored. If I change the input types both to Number(6, 4) I get as result 0.12612612612612611 which still ignores the result type (4 places after the comma).

Why is this?

A: Length & Precision are just metadata pieces.

If you want to round to the specified precision, you should do this in another step. However: please keep in mind that rounding double point precision values is futile anyway. A floating point number is stored as an approximation (it floats) so 0.1261 (your desired output) could (would probably) end up being stored as 0.126099999999 or 0.1261000000001 (Note: this is not the case for BigNumbers)

So in the end we round using BigDecimals once we store the numbers in the output table, but NOT during the transformation. The same is true for the Text File Output step. If you would have specified Integer as result type, the internal number format would have been retained, you would press "Get Fields" and it the required Integer type would be filled in. The required conversion would take place there and then.

In short: we convert to the required metadata type when we land the data somewhere, NOT BEFORE.

5.9. HTTP Client Step questions

5.9.1. The HTTP client step doesn't do anything

Q: The HTTP client step doesn't do anything, how do I make it work?

A: The HTTP client step needs to be triggered. Use a Row generator step generating e.g. 1 empty row and link that with a hop to the HTTP client step.

5.9.2. The HTTP client step and SOAP

Q: Does the HTTP client support SOAP?

A: No, it just calls an URL with arguments. Future steps may provide SOAP functionality, Work is underway on a WebService step supporting WSDL. The first experimental version appeared in PDI v2.5.0

5.10. Javascript questions

5.10.1. How to check for the existence of fields in rows

Q: How do I check for the existence of a certain value in a row?

A: The following snippet will let you check this. But keep in mind that you can not mix rows in PDI, all rows flowing over a single hop have to have the same number of fields, which have to be of the same name and type.

The snippet:

```
var idx = row.searchValueIndex("lookup");
if ( idx < 0 )
{
   // doesn't exist
}
else
{
   var lookupValue = row.getValue(idx);
}
```

5.10.2. How to add a new field in a row

Q: How do I add a new field to a row in a Javascript step?

A: Note upfront that the order in which add fields to a row is important. You always have to add the fields in the same order to the row to always get the same structure of row. Now to add a field:

1) Define it as “var” in the source and add it as a field in the lower level table:

2) Clone an existing field and add it to the row:

```
var newValue = row.getValue(0).Clone();
row.addValue(newValue);
```

3) Use following steps

```
var value = Packages.be.ibridge.kettle.core.value.Value.getInstance();
value.setName("name_of_field");
value.setValue("value_of_field");   // possibly using types other than String
row.addValue(value);
```

5.10.3. How to replace a field in a row

Q: How do I replace a field in a row in a Javascript step?

A: Use setValue on the input field as follows (assuming field1 is a field in the input row):

```
field1.setValue(100);
```

setValue() takes all possible types that can be used in PDI (also String, Dates, ...).

5.10.4. How to create a new row

Q: How do I create a new row in the javascript step?

A: In the javascript step you have 2 special objects: “row” and “_step_”:

```
var newRow = row.Clone(); // make a copy

// modify newRow

_step_.putRow(newRow);    // sends an extra row on the output of the step
```

Note that you should make sure that the row you're putting out to the next steps is of the same layout as the row that normally gets sent out. And also not that newRow is being put out **before** the regular row.

5.10.5. How to use something as NVL in javascript?

Q: How do I use something as the Oracle NVL in javascript?

A: You can use the following construction (to get something as nvl(a, '0'))

```
var a;
if ( fieldname.isNull() )
{
    a = '0';
}
else
{
    a = fieldName.getString();
}
```

and you can also use:

```
fieldName.nvl('1');
```

which would replace the value of fieldName with the value of '1' if fieldName is null.

5.10.6. Example of how to split fields

Q: In a field I have merchant code containing numbers and characters, ex. "12345McDonalds", i want to split it, the field doesn't have a consistent layout and I want to split the first part which is numeric from the second part. How to do this?

A: Use following piece of javascript, Merchant_Code is the name of the input field

```
java;

var str = Merchant_Code.getString();

var code = "";
var name = "";

for (i = 0; i < str.length(); i++ )
{
    c = str.charAt(i);
    if ( ! java.lang.Character.isDigit(c) )
    {
        code = str.substring(0, i);
        name = str.substring(i);
        Alert("code="+code+", name="+name);
        break;
    }
}
```

The Alert() is just to show the fields of course. After the outer for loop you could add code and name in new separate fields e.g.

5.11. Shell job entry questions

5.11.1. How to check for the return code of a shell script/batch file

The Shell script considers a return code of 0 to mean success, anything else is failure. You can use hops to control the resulting flow.

5.12. Call DB Procedure Step questions

5.12.1. The Call DB Procedure step doesn't do anything

Q: The Call DB Procedure step doesn't do anything, my transformation finishes without doing anything and without issuing errors. How do I make it work?

A: The Call DB Procedure needs to be triggered. Use a Row generator step generating e.g. 1 empty row and link that with a hop to the Call DB Procedure step.

6. Twilight user-development questions

6.1. Things that were once proposed but were rejected

6.1.1. Implement connection type as a variable parameter

"I can parametrize host, user-id, password in database steps in transformations using variables, but I can't set the connection type like that yet. I have jobs that need to execute on multiple types of databases (Oracle, MySql, ...) and I would like the connection type set in the same way so that I can run the same transformation on multiple databases."

This has been requested a few times. Reasons for not implementing it in PDI:

- For anything but simple SQL statements the SQL you write will be database type dependent. E.g. If you use Oracle analytics you're SQL won't run anymore on DB2 or MySql. Currently you know the type of the database and you could use the full functionality of the database;
- Starting with PDI version 2.3.1 and continuing in later versions more database specific settings were introduced. So just specifying "this is Oracle, MySql, ..." is not sufficient anymore, and a way to parametrize these specific options would need to be found as well (which would make it pretty complex);
- How many data warehouses run on multiple types of databases. Most data warehouses are created based on specific operational systems and targeting only specific database types since resulting reports and cubes would also need to run on those databases. So the use of connection type parameterizing would probably also not be that huge.

Possible workaround: maintain duplicate jobs for multiple databases. Alternatively you can use the generic ODBC which supports variable substitution for the driver as of PDI version 2.5.0GA. The disadvantage of the latter solution being that the special database processing for some types of database will not be done of course.

6.1.2. Implement "on the fly DDL" creation of tables, ...

"I want tables to be created on the fly when they don't exist yet".

The reason for not implementing "on the fly DDL" in PDI is that it would work for small examples, but it would make your DBA's life harder for real projects. Most companies have setup pretty complex database privileges which would be hard to maintain using "on the fly DDL".

In the database steps there usually is an "SQL" button thats shows the DDL that could be executed. This should mostly be used in quick mock-ups, not for real projects. Do a proper design for real projects.

We are not supporters of any "on the fly DDL" in any step.

6.1.3. Implement a step that shows a dialog and asks parameters

The reason for not implementing this is that PDI is an ETL tool, not a reporting tool. You cannot assume there will be an interactive user answers questions when a job/transformation is running.

You can have a look at the Pentaho framework where you can indeed build webpages that ask for parameters, drop-downs, selectors, ... to parametrize transformations this way. Alternatively in the examples directory there's an example called dialog.ktr that shows a way to make a dialog box using javascript in PDI.

Besides this, as of PDI v2.4.0 the launching dialog also supports the entering of parameters via a GUI style which satisfies most of the people asking to enter parameters.

As of 2.5GA there is a job entry called “Display Msgbox Info” that will display a message in a dialog box, however even this functionality should only be used for debugging purposes. There's absolutely no guarantee that it will work when scheduled (e.g. On UNIX if you do not have a controlling terminal the job will probably even abort when you display a dialog box).

6.1.4. Implement serialization of transformations using reflection

It has been advised a few times to use XML “bean” serialization to automatically serialize and deserialize jobs/transformation as that would make it “easier” to develop them.

It is possible to end up with decent XML code using say XStream, but only by setting all the element names manually etc, you end up with a situation that is far worse than the current one.

The situation with the XML serialization that we have right now is:

1) it always works, regardless if the element/parent element/etc exists or not, is empty or not, etc;

2) it's extremely simple, everyone understands it, even those that don't know the XStream/whatever API;

3) the generated XML is understandable and readable. (more so than other tools I might add) This makes PDI more open.

4) we rarely every have any issues with it, no surprises, no NPEs flying around etc. It's very resilient.

5) It's easy to test and debug.

That should be plenty of reason to keep it stable for the time being, ***not*** using reflection.

6.1.5. Implement retry on connection issues

Usually what is asked is as follows: “My jobs occasionally receive an "I/O Error: Connection reset" while performing DB Lookup steps. This causes the entire transformation to fail. Is there any way to configure the transformation to retry if there are connection issues, or to open another connection?”

An I/O error usually means that the underlying hardware is failing, let's assume it's the network. The connection with the database is dropped so you don't know which SQL statement was executed correctly and which one wasn't. You can't ask the database because the connection is dead and you can't assume that the database rolls back the statements until the last commit.
Well, the problem is that it can occur at the time you do a commit. If you commit 10.000 rows at a time: did they go into the table or not?

6.1.6. Implement GUI components in a transformation or job

Some people request implementation of GUI elements in a transformation or a job. In ETL this is normally not a good thing, most of the times there are not going to be people around to press buttons when the transformation or job is running. The transformation/job may also be running in a an environment that will not allow it to display dialogs: e.g. When running on a carte server on a remote system.

Some GUI components slipped in, but these are mostly for debugging purposes.

6.1.7. Hardcoding Locale

”No-one is going to use dates in another format as the default English Locale, so why don't we hardcode the English Locale.” or whatever other Locale feature that can be hardcoded (numbers, ...)

Well, not all people use English as default Locale (or your own preference for that matter), so hardcoding a locale is not a good idea, people should have a choice of default Locale.

See also “on Locales” in the developer guidelines.

6.1.8. Removing repository support

For a new step I need very a very expressive way of saving the meta-data for my step. I can do this with XML, but I will never be able to get it saved to the repository. Why don't we remove support for the repository as no-one really uses it anyway?

Some people do use the repository, and if you want to make a step that is going to be included in the general release you need to support the repository.

7. Development questions

7.1. Development guidelines

7.1.1. Priority on development

Correctness/Consistency

If a tool is not correct it's not going to be trusted however fast it may be. It can't be that the same input will produce output A in one case, and output B in another case.

Backwards compatibility

Everyone like upgrades to go smoothly. Install the new binaries and be able to run without testing is the ideal. Of course, in some cases compatibility has to be broken for the greater good in the long term, but then it should be clearly documented (for upgrades).

Speed

There is a need for a speed. No-one wants to wait 30 minutes to insert 100.000 rows.

User friendliness

It should not be a torment to use a tool. It should allow both novice and expert users to get their job done. As example: any XML or configuration file should have a GUI element to manage this and should never be edited manually.

7.1.2. Use English in the source code

Since PDI is developed by an international group of people use English in the source code for everything: identifiers, comments, ...

7.1.3. Division of functionality in steps and job entries

One of the ideas of Pentaho Data Integration is to make simple steps and job entries which have a single purpose, and be able to make complex transformations by linking them together, much like UNIX utilities.

Putting too much (diverse) functionality in 1 step/job entry will make them less intuitive for people to use, and since most people only start reading manuals when they get into problems we need all the intuitivity we can get.

7.1.4. Rows on a single hop have to be of the same structure

As in user section of this document: all rows that flow over a single hop have to be of the same structure. You shouldn't try to build things that try to circumvent this, which will be harder as of v2.5.0 because of the design time check on the structure of the rows.

7.1.5. Null and “” are the same in PDI

As in Oracle the empty string “” and NULL should be considered the same by all steps. This is to be in line with the rest with PDI.

7.1.6. On converting data to fit the corresponding Metadata

Length & Precision are just metadata pieces.

If you want to round to the specified precision, you should do this in another step. However: please keep in mind that rounding double point precision values is futile anyway. A floating point number is stored as an approximation (it floats) so 0.1261 (your desired output) could (would probably) end up being stored as 0.126099999999 or 0.1261000000001 (Note: this is not the case for BigNumbers)

So in the end we round using BigDecimals once we store the numbers in the output table, but NOT during the transformation. The same is true for the Text File Output step. If you would have specified Integer as result type, the internal number format would have been retained, you would press "Get Fields" and it the required Integer type would be filled in. The required conversion would take place there and then.

In short: we convert to the required metadata type when we land the data somewhere, NOT BEFORE.

The strategy of keeping the same datatype as long as possible has saved us from many a conversion error like the one described above.

7.1.7. On logging in steps in Pentaho Data Integration

Q: How to use logging in PDI steps? This applies to the main PDI steps as to any PDI steps you develop yourself.

A: All detailed, debug, and rowlevel statements needs to be preceded by the following:

```
if (log.isDetailed())
  ...

if (log.isDebug())
  ...

if (log.isRowlevel())
  ...
```

That is because otherwise, the string calculation of what you send to the log is always calculated. For Basic and Minimal logging levels this doesn't matter as normally they would always be “on”, but it does for Debug and Rowlevel.

7.1.8. On using XML in Pentaho Data Integration

Q: What's the idea of using XML in PDI?

A: XML is for machines, not for humans. So unless the functionality is in fact on the processing of XML itself (XML input step/XML output step) the XML should be kept hidden. Behind the screens XML will be used but users of PDI should not to be required to know this or manipulate XML in any way. Every XML configuration/setup file should be managed through a GUI element in PDI.

7.1.9. On dropdown boxes and storing values

Don't use the index of the dropdown box as a value in the XML export file or database repository.

Suppose you have currently 3 possible values in a dropdown box. And if someone chooses the first value you put “1” in the XML export file to indicate value 1. This would work fine, except that:

- if someone wants to add extra values in the future he must use the order you defined first;
- it makes the XML output very much unreadable.

It's better to convert from a Locale string in the GUI to some English equivalent which is then stored. As example:

- Suppose on the GUI you have a dropdown box with values “Date mask” and “Date time mask”;
- Instead of using a 1 in the output for “Date mask” and 2 for “Date time mask”, it would be better to put in the output “DATE_MASK” for “Date mask” and “DATE_TIME_MASK” for “Date time mask”;
- Also note that DATE_MASK/DATE_TIME_MASK would then not be allowed to be subject to I18N translation (which is ok for transformation/job files).

7.1.10. On using I18N in PDI

Q: Some more details on using I18N

A:

- Only translate what a normal user will see, it doesn't make sense to translate all debug message in PDI. Some performance improvements were achieved in PDI just by removing some of translations for debug messages;
- Make sure you don't translate strings used in the control logic of PDI:
    - If you would e.g. make the default name of a new step “language dependent” this would still make jobs/transformations usable across different locales;
    - If you would e.g. make tags used in the XML generated for the step language dependent there would be a problem when a user would switch his locale;
    - If you would translate non-tag strings used in the control logic you will also have a problem. E.g. in the repository manager “Administrator” is used to indicate which user is administrator (and this is used in the PDI control logic). So if you would translate administrator to a certain language, this would work as long as you wouldn't switch locales.

7.1.11. On using Locale's in PDI

PDI should always use the default Locale, so the Locale should not be hardcoded somewhere to English or so. However some steps may choose to be able to override the default Locale but this is then step specific and it should always be possible to select the Locale via the GUI of the step.

7.1.12. On reformatting source code

Try to keep reformatting code to a minimum, especially on things like {'s at the end of the line or at the start of the next line, not all people like the same and why should your specific preference be used.

When changing code try to use the same formatting as the surrounding code, even if it's not your preference.

If you really feel a need for some reformatting do it in a separate SVN check-in, **DON'T** mix reformatting source code with real changes. It's **VERY** annoying not being able to easily see what changed because someone decided to reformat source code at the same time. “My tool does automatic formatting” is a very lame excuse as all known IDEs allow to switch it off.

7.1.13. On checking persistence correctness

All steps and job entries need to implement persistence to XML format **AND** the repository. Since saving to XML format and saving to the repository is done using separate methods it sometimes happened that the meta-data would be saved properly in one format, but not in the other.

The following procedure does a basic check whether data is properly saved. It's not a 100% check, but it will detect most obvious mistakes.

Procedure part 1:

- Make a job or a transformation in Spoon with only the step/job entry which is new or was changed. Fill in all values that you can;
- Save it to XML format, close it in Spoon;
- Reload it in Spoon;
- Save it XML format using a new name;
- Take your favorite text compare tool and compare both XML files, all changes should be explainable.

If the latter test succeeds it shows that saving/loading of XML has no obvious mistakes. If you can have multiple exclusive values in the meta-data this diminishes the value of the test a bit and you maybe should try with some extra testcases.

Procedure part 2:

- Load the original XML format of part 1 in Spoon;
- Connect to a repository, and save the job/transformation in there;
- Close it in Spoon;
- Reload the the job/transformation from the repository;
- Save it to XML format;
- Take your favorite text compare tool and compare both XML files, all changes should be explainable.

Part 1 shows that the loading/saving to XML works. If part 1 succeeds, but part 2 chances are very high something is wrong in the methods loading/saving to the repository.

7.1.14. On using non-temporary storage

Don't implement functionality that stores data in non-temporary storage in a PDI specific/internal format. With non-temporary we mean surviving a job or a transformation execution. The reason for this is to avoid having to deal with conversions.

Reason: suppose e.g. a step would serialize rows into a file to be read out in a next transformation. If you would upgrade PDI in between runs the row may not de-serialize correctly anymore. To solve this conversion applications would be required (or some old/new format logic if even possible). As long as you don't save data into an internal format that survives a job/transformation you're always ok.

7.2. How do you start developing your own plug-in step

Q: I need some complex calculation in my ETL. I have created own logic in java classes. How can I make my own step and integrate this in PDI?

A: You can start using the DummyPlugin as example. You can also have a look at how some other steps are implemented. In essence you need 4 classes to make a new step implementing the corresponding interfaces:

- StepMetaInterface: contains the meta-data;
- StepInterface: performs the actual work, implements logic;
- StepDialogInterface: pops up a dialog in Spoon;
- StepData: contains temporary data like ResultSets, file handels, input streams, etc.

You can implement these the easiest way by looking at the existing steps and by inheriting from the base classes: BaseStepMeta, BaseStep. Package up these 4 classes and any others you might need in a jar file, for example “foo.jar”.

After that all you need to do is design an icon, save it in PNG format, for example “foo.png”. 32x32 is the default although size but other dimensions should work equally well. Then create a bootstrap in the form of plugin.xml. Look at the example plugin.xml to see all the options in action.

Finally, put all 3 files: “foo.jar”, “foo.png” and “plugin.xml” in a directory with a name of your choice (for example “foo”:

```
plugins/steps/foo/
```

or

```
$HOME/.kettle/plugins/steps/foo/
```

The PDI step loader will search for the file “plugin.xml” during startup and load the specified class as well as the extra jars/classes you might need.

7.3. Checklist for end of step/job entry development

Following is a list of things you need to think of before considering a change to a step/job entry or a complete new step/job entry to be complete.

- Does the change break anything compared to the previous release(s). If possible nothing should break, but sometimes there's no other way. If something breaks and there's no way around it, at least inform the PDI tech lead;
- Is the source code completely in English. Especially check the key fields that will be saved in the XML file/repository whether they are in English. It's hard to change these later on;
- Does loading/saving work correctly with both XML format and a repository. This is mostly for new steps/job entries or for changes in existing attributes. A check for this included in the FAQ “On checking persistence correctness”;
- Is the documentation up to date with the changes?

7.4. On using Subversion

1. Always make sure that whatever you put in SVN keeps PDI buildable and startable. Nothing is more annoying as not being able to even start PDI as someone checked in half-working code. If the change is too big to do at once, work by making small steps towards the full change (but at all times keeping PDI buildable/runnable).

7.5. Can you change the XML input step to process my file?

Q: I have to process an XML file which currently can't be processed by KETTLE, e.g. there's one optional field which depends on the value of an element and that should also be included as a field in a row, ... Can you build this functionality in in the XML input step?

A: First of all it would depend what functionality you need. If the functionality is generally useful it can be built in. If it would only be useful for you it wouldn't make sense to build it in.

As alternative solutions: consider processing the XML file via a Javascript step, or if what is required is very complex consider writing your own PDI step which you maintain yourself (outside of the PDI distribution).

7.6. On Serializable and Binary

Q: If I need to select a type I can choose between Serializable and Binary. Are they the same, or what's the difference?

A: Binary is used for images, sounds, GIS data, BLOBs, ... You can read e.g. a database BLOB from one database and insert it in another database. Binary uses getByte() and setByte() to get and set its value.

Serializable is used for some proprietary plugins build by a company using it to pass Java objects from one step to another. Unless you're developing your own steps/plugins Serializable is not something to be used. The way to read/write data depends on the objects being stored in a Serializable.

7.7. Success factors of PDI

7.7.1. Modular design

Pentaho Data Integration as a runtime engine consists of a “row engine” taking care of transporting data from one step to the next. The steps are separate and can even use a plug-in architecture.

7.7.2. No generation of code

The biggest advantage of not generating code is that the job is always in the correct state. The object code and the “source” code of the jobs can never be out of sync as explicit object code is not generated. An extra advantage is that jobs always use the latest version of the components after upgrading the core, it does not require re-compiling/regenerating their own jobs.

The biggest disadvantage would be speed, but comparing the speed of PDI to other ETL tools the disadvantage doesn't seem that big (in a lot of cases PDI is even faster than similar jobs in other ETL tools).