



Connecting the
Data-Driven Enterprise >



Talend Open Studio for Data Integration

User Guide

6.5.0M1

Adapted for v6.5.0M1. Supersedes previous releases.

Publication date: July 27, 2017

Copyleft

This documentation is provided under the terms of the Creative Commons Public License (CCPL).

For more information about what you can and cannot do with this documentation in accordance with the CCPL, please read: <http://creativecommons.org/licenses/by-nc-sa/2.0/>

Notices

Talend is a trademark of Talend, Inc.

All brands, product names, company names, trademarks and service marks are the properties of their respective owners.

License Agreement

The software described in this documentation is licensed under the Apache License, Version 2.0 (the "License"); you may not use this software except in compliance with the License. You may obtain a copy of the License at <http://www.apache.org/licenses/LICENSE-2.0.html>. Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

This product includes software developed at AOP Alliance (Java/J2EE AOP standards), ASM, Amazon, Antlr, Apache ActiveMQ, Apache Ant, Apache Axiom, Apache Axis, Apache Axis 2, Apache Batik, Apache CXF, Apache Chemistry, Apache Common Http Client, Apache Common Http Core, Apache Commons, Apache Commons Bcel, Apache Commons JXPath, Apache Commons Lang, Apache Derby Database Engine and Embedded JDBC Driver, Apache Geronimo, Apache Hadoop, Apache Hive, Apache HttpClient, Apache HttpComponents Client, Apache JAMES, Apache Log4j, Apache Lucene Core, Apache Neethi, Apache POI, Apache ServiceMix, Apache Tomcat, Apache Velocity, Apache WSS4J, Apache WebServices Common Utilities, Apache Xml-RPC, Apache Zookeeper, Box Java SDK (V2), CSV Tools, DataStax Java Driver for Apache Cassandra, Ehcache, Ezmorph, Ganymed SSH-2 for Java, Google APIs Client Library for Java, Google Gson, Groovy, Guava: Google Core Libraries for Java, H2 Embedded Database and JDBC Driver, Hector: A high level Java client for Apache Cassandra, Hibernate Validator, HighScale Lib, HsqlDB, Ini4j, JClouds, JLine, JSON, JSR 305: Annotations for Software Defect Detection in Java, JUnit, Jackson Java JSON-processor, Java API for RESTful Services, Java Agent for Memory Measurements, Jaxb, Jaxen, Jettison, Jetty, Joda-Time, Json Simple, LightCouch, MetaStuff, Mondrian, OpenSAML, Paracel JDBC Driver, PostgreSQL JDBC Driver, Resty: A simple HTTP REST client for Java, Rocoto, SL4J: Simple Logging Facade for Java, SQLite JDBC Driver, Simple API for CSS, SshJ, StAX API, StAXON - JSON via StAX, The Castor Project, The Legion of the Bouncy Castle, W3C, Woden, Woodstox: High-performance XML processor, Xalan-J, Xerces2, XmlBeans, XmlSchema Core, Xmlsec - Apache Santuario, Zip4J, atinject, dropbox-sdk-java: Java library for the Dropbox Core API, google-guice. Licensed under their respective license.

Table of Contents

Preface	vii
1. General information	vii
1.1. Purpose	vii
1.2. Audience	vii
1.3. Typographical conventions	vii
2. Feedback and Support	vii
Chapter 1. Data Integration: Concepts and Principles	1
1.1. Data analytics	2
1.2. Operational integration	2
Chapter 2. Working with projects	5
2.1. Introduction to projects	6
2.2. How to create a project	6
2.3. How to import a demo project	8
2.4. How to import projects	10
2.5. How to open a project	12
2.6. How to delete a project	13
2.7. How to export a project	14
Chapter 3. Designing a Business Model	17
3.1. What is a Business Model	18
3.2. Opening or creating a Business Model	18
3.2.1. How to open a Business Model	19
3.2.2. How to create a Business Model	19
3.3. Modeling a Business Model	20
3.3.1. Shapes	21
3.3.2. Connecting shapes	22
3.3.3. How to comment and arrange a model	24
3.3.4. Business Models	25
3.4. Assigning repository elements to a Business Model	28
3.5. Editing a Business Model	29
3.6. Saving a Business Model	29
Chapter 4. Designing a Job	31
4.1. What is a Job design	32
4.2. Getting started with a basic Job	32
4.2.1. Creating a Job	32
4.2.2. Adding components to the Job	34
4.2.3. Connecting the components together	37
4.2.4. Configuring the components	39
4.2.5. Executing the Job	40
4.3. Working with components	41
4.3.1. How to add a component between two connected components	41
4.3.2. How to define component properties	45
4.3.3. How to define the start component	53
4.3.4. How to find Jobs containing a specific component	54
4.3.5. How to set default values in the schema of a component	56
4.4. Using connections	58
4.4.1. Connection types	58
4.4.2. How to define connection settings	63
4.5. Using contexts and variables	65
4.5.1. How to define context variables for a Job	65
4.5.2. How to centralize context variables in the Repository	71
4.5.3. How to apply Repository context variables to a Job	78
4.5.4. How to use variables in a Job	80
4.5.5. How to run a Job in a selected context	80
4.5.6. StoreSQLQuery	81
4.6. Using parallelization to optimize Job performance	81
4.6.1. How to execute multiple Subjobs in parallel	81
4.6.2. How to launch parallel iterations to read data	82
4.7. Handling Jobs: advanced subjects	83
4.7.1. How to map data flows	83
4.7.2. How to create queries using the SQLBuilder	83
4.7.3. How to download/upload Talend Community components	88
4.7.4. How to use the tPrejob and tPostjob components	94
4.7.5. How to use the Use Output Stream feature	95
4.8. Handling Jobs: miscellaneous subjects	95
4.8.1. How to use folders	95
4.8.2. How to share a database connection	96
4.8.3. How to handle error icons on components or Jobs	97
4.8.4. How to add notes to a Job design	99
4.8.5. How to display the code or the outline of your Job	100
4.8.6. How to manage the subjob display	101
4.8.7. How to define options on the Job view	103
Chapter 5. Managing Jobs	107
5.1. Activating/Deactivating a component or a subjob	108
5.1.1. Activate or deactivate a component	108
5.1.2. Activate or deactivate a subjob	108
5.1.3. Activate or deactivate all linked subjobs	109
5.2. Importing/exporting items and building Jobs	109
5.2.1. How to import items	109
5.2.2. How to build Jobs	113
5.2.3. How to export items	120
5.2.4. How to change context parameters in Jobs	122
5.3. Managing repository items	123
5.3.1. How to handle updates in repository items	123
5.4. Searching a Job in the repository	125
5.5. Managing Job versions	127
5.6. Documenting a Job	128
5.6.1. How to generate HTML documentation	129
5.6.2. How to update the documentation on the spot	130
5.7. Handling Job execution	130
5.7.1. How to run a Job in normal mode	130
5.7.2. How to run a Job in Java Debug mode	131
5.7.3. How to run a Job in Traces Debug mode	132
5.7.4. How to set advanced execution settings	133
5.7.5. How to show JVM resource usage during Job execution	135

5.7.6. How to deploy a Job on SpagoBI server	136
Chapter 6. Mapping data flows	141
6.1. Map editor interfaces	142
6.2. tMap operation	143
6.2.1. Setting the input flow in the Map Editor	144
6.2.2. Mapping variables	151
6.2.3. Working with expressions	152
6.2.4. Mapping the Output setting	160
6.2.5. Setting schemas in the Map Editor	165
6.2.6. Enabling automatic data type conversion	169
6.2.7. Solving memory limitation issues in tMap use	171
6.2.8. Handling Lookups	173
6.3. tXMLMap operation	176
6.3.1. Using the document type to create the XML tree	177
6.3.2. Defining the output mode	188
6.3.3. Editing the XML tree schema	193
Chapter 7. Managing Metadata	195
7.1. Objectives	196
7.2. Centralizing database metadata	197
7.2.1. Setting up a database connection	197
7.2.2. Retrieving table schemas	200
7.3. Centralizing JDBC metadata	205
7.3.1. Setting up a JDBC connection	205
7.3.2. Retrieving table schemas	207
7.4. Centralizing SAS metadata	208
7.4.1. Setting up a SAS connection	209
7.4.2. Retrieving table schemas	210
7.5. Centralizing File Delimited metadata	211
7.6. Centralizing File Positional metadata	216
7.7. Centralizing File Regex metadata	221
7.8. Centralizing XML file metadata	225
7.8.1. Setting up XML metadata for an input file	225
7.8.2. Setting up XML metadata for an output file	235
7.9. Centralizing File Excel metadata	245
7.10. Centralizing File LDIF metadata	250
7.11. Centralizing JSON file metadata	255
7.11.1. Setting up JSON metadata for an input file	255
7.11.2. Setting up JSON metadata for an output file	262
7.12. Centralizing LDAP connection metadata	270
7.13. Centralizing Azure Storage metadata	275
7.14. Centralizing Marketo metadata	279
7.15. Centralizing Salesforce metadata	283
7.16. Centralizing Snowflake metadata	287
7.17. Setting up a generic schema	291
7.17.1. Setting up a generic schema from scratch	292
7.17.2. Setting up a generic schema from an XML file	295
7.17.3. Saving a component schema as a generic schema	297
7.18. Centralizing MDM metadata	298
7.18.1. Setting up the connection	298
7.18.2. Defining MDM schema	301
7.19. Centralizing Web Service metadata	316
7.19.1. Setting up a simple schema	317
7.19.2. Setting up an advanced schema	322
7.20. Centralizing an FTP connection	330
7.21. Exporting metadata as context and reusing context parameters to set up a connection	333
7.21.1. How to export connection details as context variables	333
7.21.2. How to use variables of an existing context group to set up a connection	337
7.22. How to use centralized metadata in a Job	342
Chapter 8. Managing routines	345
8.1. What are routines	346
8.2. Accessing the System Routines	346
8.3. Customizing the system routines	347
8.4. Managing user routines	348
8.4.1. How to create user routines	348
8.4.2. How to edit user routines	350
8.4.3. How to edit user routine libraries	350
8.5. Calling a routine from a Job	352
8.6. Use case: Creating a file for the current date	352
Chapter 9. Using SQL templates	355
9.1. What is ELT	356
9.2. Introducing Talend SQL templates	356
9.3. Managing Talend SQL templates	356
9.3.1. Types of system SQL templates	357
9.3.2. How to access a system SQL template	357
9.3.3. How to create user-defined SQL templates	359
9.3.4. A use case of system SQL templates	361
Appendix A. GUI	367
A.1. Main window	368
A.2. Menu bar and Toolbar	368
A.2.1. Menu bar of <i>Talend Studio</i>	369
A.2.2. Toolbar of <i>Talend Studio</i>	370
A.3. Repository tree view	371
A.4. Design workspace	372
A.5. Palette	372
A.6. Configuration tabs	373
A.7. Outline and code summary panel	374
A.8. Shortcuts and aliases	375
Appendix B. Customizing Talend Studio and setting Studio preferences	377
B.1. Customizing project settings	378
B.1.1. Setting the compiler compliance level	378
B.1.2. Customizing Maven build script templates	379
B.1.3. Palette Settings	381
B.1.4. Type mapping	382
B.1.5. Version management	383
B.1.6. Status management	385
B.1.7. Job Settings	386
B.1.8. Stats & Logs	386
B.1.9. Context settings	387
B.1.10. Applying Project Settings	388
B.1.11. Status settings	389
B.1.12. Security settings	391
B.2. Customizing the workspace	391
B.2.1. How to change the Palette layout and settings	391
B.2.2. How to change panels positions	394
B.2.3. How to display Job configuration tabs/views	395
B.3. Filtering entries listed in the Repository tree view	396

B.3.1. How to filter by Job name	396	D.3.1. How to store a string in alphabetical order	449
B.3.2. How to filter by user	398	D.3.2. How to check whether a string is alphabetical	450
B.3.3. How to filter by job status	400	D.3.3. How to replace an element in a string	450
B.3.4. How to choose what repository nodes to display	401	D.3.4. How to check the position of a specific character or substring, within a string	450
B.4. Setting Talend Studio preferences	402	D.3.5. How to calculate the length of a string	450
B.4.1. Java Interpreter path (Talend)	403	D.3.6. How to delete blank characters	451
B.4.2. Designer preferences (Talend > Appearance)	403	D.4. TalendDataGenerator Routines	451
B.4.3. How to define the user component folder (Talend > Components)	404	D.4.1. How to generate fictitious data	451
B.4.4. How to change specific component settings (Talend > Components)	405	D.5. TalendDate Routines	452
B.4.5. Documentation preferences (Talend > Documentation)	406	D.5.1. How to format a Date	454
B.4.6. Exchange preferences (Talend > Exchange)	406	D.5.2. How to check a Date	454
B.4.7. Adding code by default (Talend > Import/Export)	407	D.5.3. How to compare Dates	454
B.4.8. Language preferences (Talend > Internationalization)	407	D.5.4. How to configure a Date	455
B.4.9. Palette preferences (Talend > Palette Settings)	408	D.5.5. How to parse a Date	455
B.4.10. Performance preferences (Talend > Performance)	408	D.5.6. How to retrieve part of a Date ...	455
B.4.11. Debug and Job execution preferences (Talend > Run/Debug)	409	D.5.7. How to format the Current Date	456
B.4.12. Displaying special characters for schema columns (Talend > Specific settings)	411	D.6. TalendString Routines	456
B.4.13. Schema preferences (Talend > Specific Settings)	411	D.6.1. How to format an XML string ...	456
B.4.14. SQL Builder preferences (Talend > Specific Settings)	413	D.6.2. How to trim a string	457
B.4.15. Usage Data Collector preferences (Talend > Usage Data Collector)	413	D.6.3. How to remove accents from a string	457
Appendix C. Theory into practice: Job examples	417	D.7. TalendStringUtil Routines	457
C.1. tMap Job example	418	Appendix E. SQL template writing rules	459
C.1.1. Input data	418	E.1. SQL statements	460
C.1.2. Output data	418	E.2. Comment lines	460
C.1.3. Reference data	418	E.3. The <%...%> syntax	460
C.1.4. Translating the scenario into a Job	419	E.4. The <%==...%> syntax	461
C.2. Using the output stream feature	426	E.5. The </.../> syntax	461
C.2.1. Input data	426	E.6. Code to access the component schema elements	462
C.2.2. Output data	427	E.7. Code to access the component matrix properties	462
C.2.3. Translating the scenario into a Job	427		
C.3. Using the Implicit Context Load feature	433		
C.3.1. Preparing context parameter sources	434		
C.3.2. Creating the Job and defining context variables	434		
C.3.3. Configuring the components	436		
C.3.4. Configuring the Implicit Context Load feature	437		
C.3.5. Executing the Job	438		
C.4. Using the Multi-thread Execution feature to run Jobs in parallel	440		
Appendix D. System routines	445		
D.1. Numeric Routines	446		
D.1.1. How to create a Sequence	446		
D.1.2. How to convert an Implied Decimal	446		
D.2. Relational Routines	447		
D.3. StringHandling Routines	447		

Preface

1. General information

1.1. Purpose

This User Guide explains how to manage *Talend Open Studio for Data Integration* functions in a normal operational context.

Information presented in this document applies to *Talend Open Studio for Data Integration 6.5.0M1*.

1.2. Audience

This guide is for users and administrators of *Talend Open Studio for Data Integration*.



The layout of GUI screens provided in this document may vary slightly from your actual GUI.

1.3. Typographical conventions

This guide uses the following typographical conventions:

- text in **bold**: window and dialog box buttons and fields, keyboard keys, menus, and menu options,
- text in **[bold]**: window, wizard, and dialog box titles,
- text in **courier**: system parameters typed in by the user,
- text in **italics**: file, schema, column, row, and variable names,
- The icon indicates an item that provides additional information about an important point. It is also used to add comments related to a table or a figure,
- The icon indicates a message that gives information about the execution requirements or recommendation type. It is also used to refer to situations or information the end-user needs to be aware of or pay special attention to.
- Any command is highlighted with a grey background or code typeface.

2. Feedback and Support

Your feedback is valuable. Do not hesitate to give your input, make suggestions or requests regarding this documentation or product and find support from the **Talend** team, on **Talend Community** at:

<https://community.talend.com/>



Chapter 1. Data Integration: Concepts and Principles

There is nothing new about the fact that organizations' information systems tend to grow in complexity. The reasons for this include the "layer stackup trend" (a new solution is deployed although old systems are still maintained) and the fact that information systems need to be more and more connected to those of vendors, partners and customers.

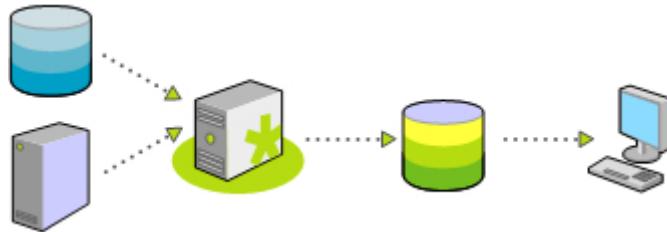
A third reason is the multiplication of data storage formats (XML files, positional flat files, delimited flat files, multi-valued files and so on), protocols (FTP, HTTP, SOAP, SCP and so on) and database technologies.

A question arises from these statements: How to manage a proper integration of this data scattered throughout the company's information systems? Various functions lie behind the data integration principle: business intelligence or analytics integration (data warehousing) and operational integration (data capture and migration, database synchronization, inter-application data exchange and so on).

Both ETL for analytics and ETL for operational integration needs are addressed by *Talend Studio*.

1.1. Data analytics

While mostly invisible to users of the BI platform, ETL processes retrieve the data from all operational systems and pre-process it for the analysis and reporting tools.



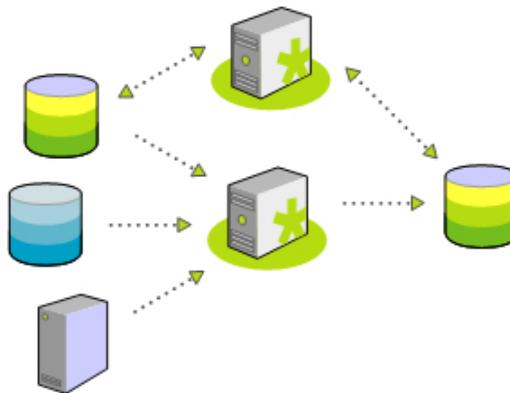
Talend Studio offers nearly comprehensive connectivity to:

- Packaged applications (ERP, CRM, etc.), databases, mainframes, files, Web Services, and so on to address the growing disparity of sources.
- Data warehouses, data marts, OLAP applications - for analysis, reporting, dashboarding, scorecarding, and so on.
- Built-in advanced components for ETL, including string manipulations, Slowly Changing Dimensions, automatic lookup handling, bulk loads support, and so on.

Most connectors addressing each of the above needs are detailed in **Talend**'s component documentation. For information about their orchestration in *Talend Studio*, see [Designing a Job](#). For high-level business-oriented modeling, see [Designing a Business Model](#).

1.2. Operational integration

Operational data integration is often addressed by implementing custom programs or routines, completed on-demand for a specific need.



Data migration/loading and data synchronization/replication are the most common applications of operational data integration, and often require:

- Complex mappings and transformations with aggregations, calculations, and so on due to variation in data structure,
- Conflicts of data to be managed and resolved taking into account record update precedence or "record owner",
- Data synchronization in nearly real time as systems involve low latency.

Most connectors addressing each of the above needs are detailed in **Talend**'s component documentation. For information about their orchestration in *Talend Studio*, see [Designing a Job](#). For high-level business-oriented modeling, see [Designing a Business Model](#). For information about designing a detailed data integration Job using the output stream feature, see [Using the output stream feature](#).



Chapter 2. Working with projects

Once you launch your *Talend Studio* and before you start a Business Model, a data integration Job, or any other tasks, you need first create or import a project.

This chapter deals with how to create, import, export, delete, and work in projects in *Talend Studio*. For how to launch and get started with *Talend Studio*, see the Getting Started Guide.

2.1. Introduction to projects

In *Talend Studio*, the highest physical structure for storing all different types of data integration Jobs, metadata, routines, etc. is the "project".

From the login window of the Studio, you can:

- create a local project.

When you launch the Studio for the first time, there are no default projects listed. You need to create a project that will hold all data integration Jobs and business models you design in the current instance of the Studio.

You can create as many projects as you need to store your data of different instances of your Studio.

When creating a new project, a tree folder is automatically created in the workspace directory on your repository server. This will correspond to the **Repository** tree view displayed on the main window of the Studio.

For more information, see [How to create a project](#).

- import the Demo project to discover the features of *Talend Studio* based on samples of different ready-to-use Jobs. When you import the Demo project, it is automatically installed in the workspace directory of the current session of the Studio.

For more information, see [How to import a demo project](#).

- import projects you have already created with previous releases of *Talend Studio* into your current *Talend Studio* workspace directory.

For more information, see [How to import projects](#).

- open a project you created or imported in the Studio.

For more information, see [How to open a project](#).

- delete local projects that you already created or imported and that you do not need any longer.

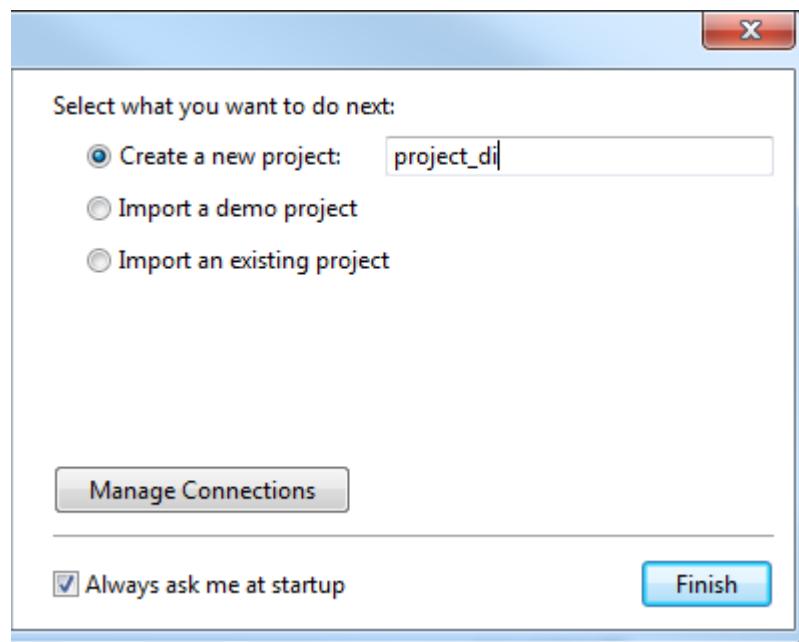
For more information, see [How to delete a project](#).

Once you launch *Talend Studio*, you can export the resources of one or more of the created projects in the current instance of the Studio. For more information, see [How to export a project](#).

2.2. How to create a project

To create a project at the initial startup of the Studio, do the following:

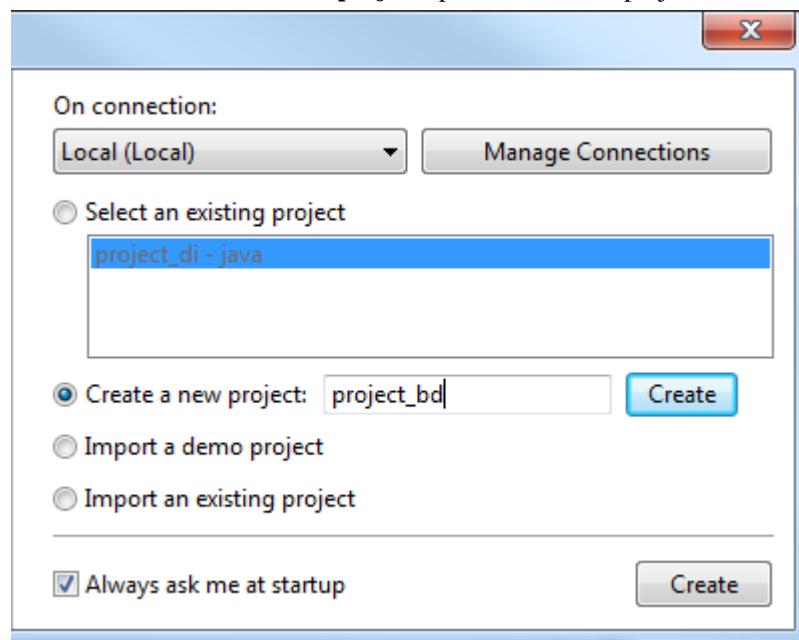
1. Launch *Talend Studio*.
2. On the login window, select the **Create a new project** option and enter a project name in the field.



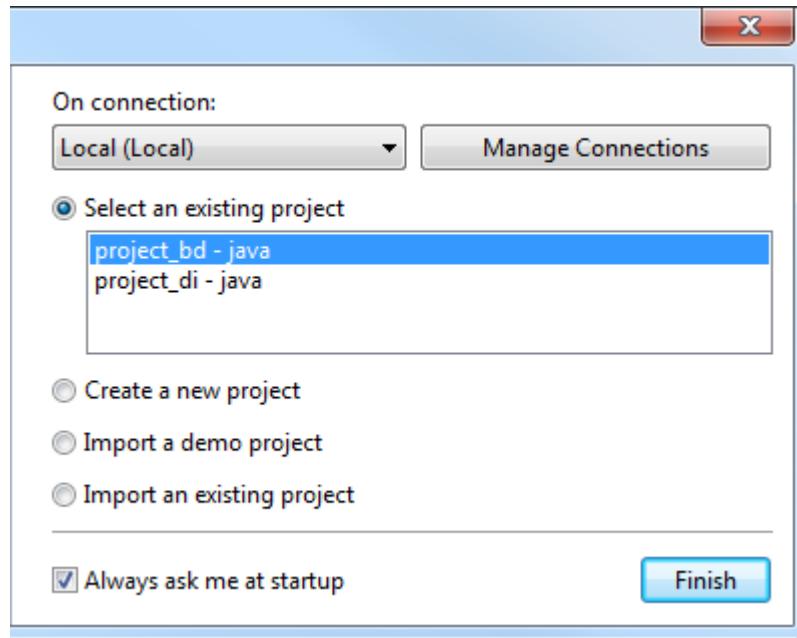
3. Click **Finish** to create the project and open it in the Studio.

To create a new project after the initial startup of the Studio, do the following:

1. On the login window, select the **Create a new project** option and enter a project name in the field.



2. Click **Create** to create the project. The newly created project is displayed on the list of existing projects.



3. Select the project on the list and click **Finish** to open the project in the Studio.

Later, if you want to switch between projects, on the Studio menu bar, use the combination **File > Switch Project or Workspace**.

2.3. How to import a demo project

You can import one or more demo projects that include numerous samples of ready to use Jobs into your *Talend Studio* to help you understand the functionalities of different **Talend** components.

To import a demo project, proceed as follows:

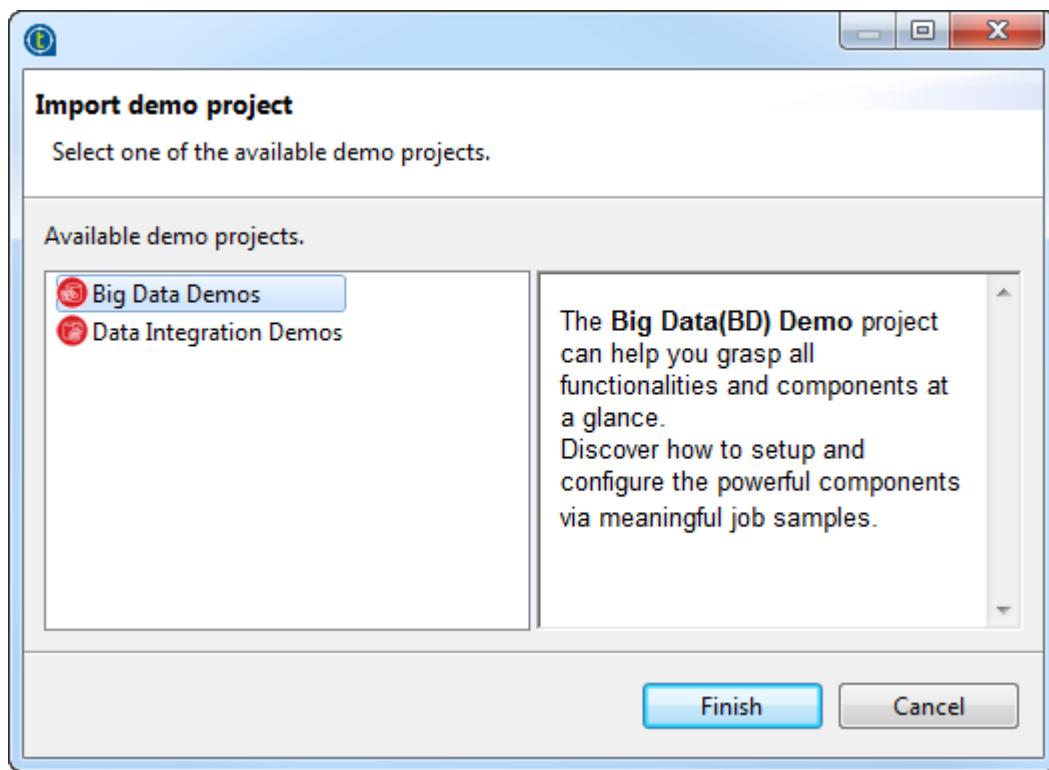
1. When launching your *Talend Studio*, select the **Import a demo project** option on the Studio login window and click **Select**, or click the **Demos** link on the welcome window, to open the [**Import demo project**] dialog box.

After launching the Studio, click  button on the toolbar, or select **Help > Welcome** from the Studio menu bar to open the welcome window and then click the **Demos** link, to open the [**Import demo project**] dialog box.

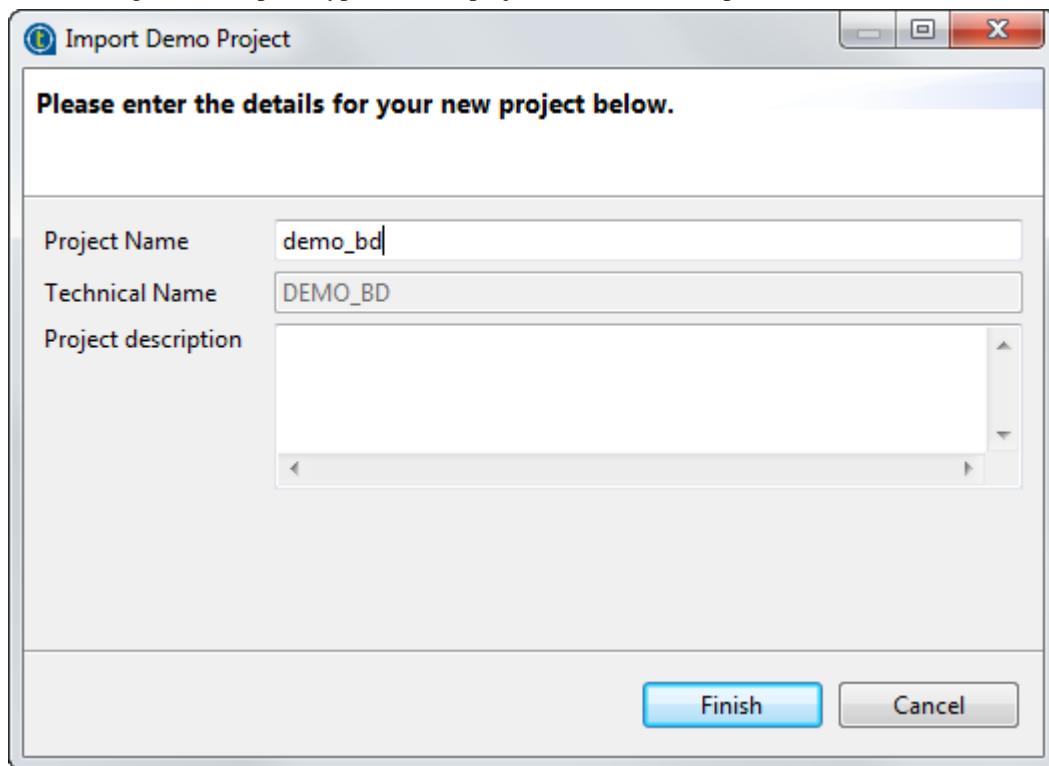
2. In the [**Import Demo Project**] dialog box, select the demo project you want to import and view the description on the right panel.



The demo projects available in the dialog box may vary depending on the product you are using.



3. Click **Finish** to close the dialog box.
4. In the new dialog box that opens, type in a new project name and description information if needed.



5. Click **Finish** to create the project.

All the samples of the demo project are imported into the newly created project, and the name of the new project is displayed in the **Project** list on the login screen.

6. To open the imported demo project in *Talend Studio*, back on the login window, select it from the **Project** list and then click **Finish**.

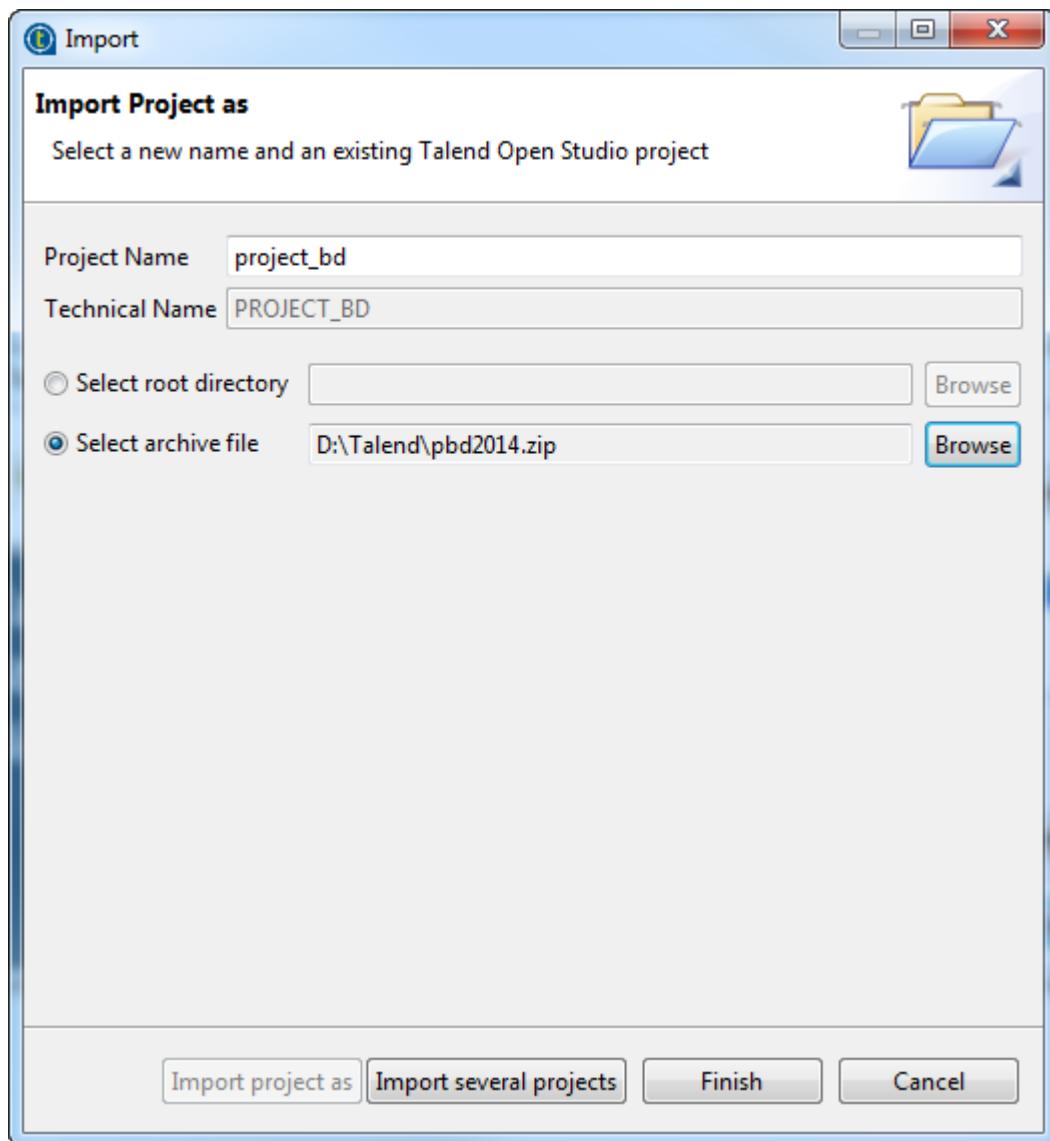
The Job samples in the open demo project are automatically imported into your workspace directory and made available in the **Repository** tree view under the **Job Designs** folder.

2.4. How to import projects

In *Talend Studio*, you can import one or more projects you already created with previous releases of the Studio.

To import a single project, do the following:

1. From the Studio login window, select **Import an existing project** then click **Select** to open the **[Import]** wizard.



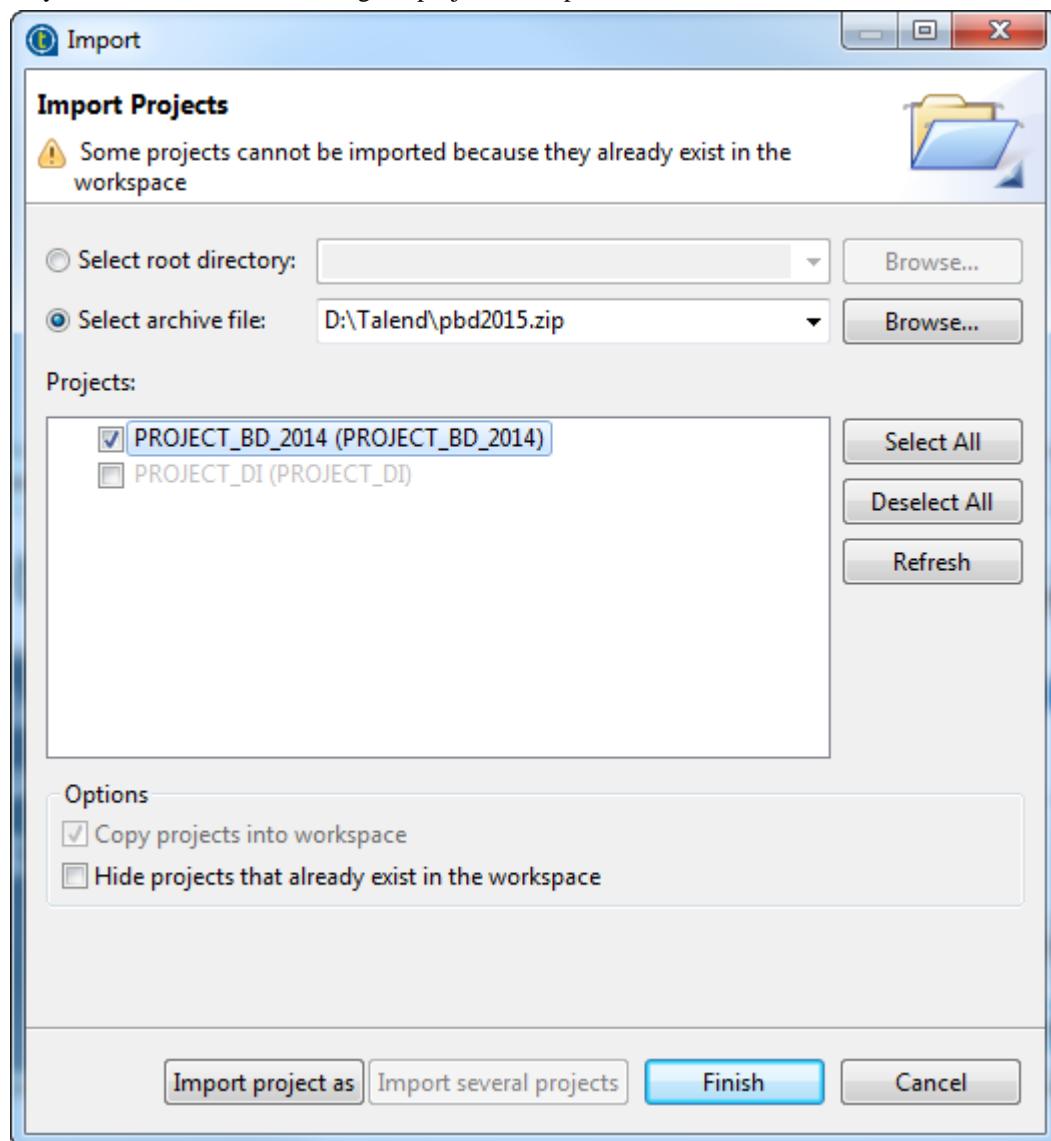
2. Click the **Import project as** button and enter a name for your new project in the **Project Name** field.
3. Click **Select root directory** or **Select archive file** depending on the source you want to import from.

4. Click **Browse...** to select the workspace directory/archive file of the specific project folder. By default, the workspace in selection is the current release's one. Browse up to reach the previous release workspace directory or the archive file containing the projects to import.

5. Click **Finish** to validate the operation and return to the login window.

To import several projects simultaneously, do the following:

1. From the Studio login window, select **Import an existing project** then click **Select** to open the **[Import]** wizard.
2. Click **Import several projects**.
3. Click **Select root directory** or **Select archive file** depending on the source you want to import from.
4. Click **Browse...** to select the workspace directory/archive file of the specific project folder. By default, the workspace in selection is the current release's one. Browse up to reach the previous release workspace directory or the archive file containing the projects to import.



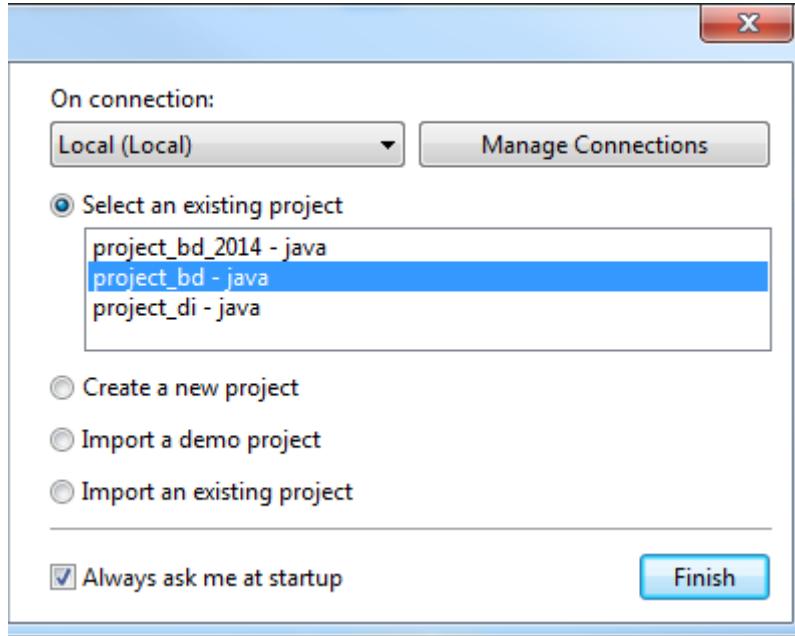
5. Select the **Copy projects into workspace** check box to make a copy of the imported project instead of moving it. This option is available only when you import several projects from a root directory.



If you want to remove the original project folders from the *Talend Studio* workspace directory you import from, clear this check box. But we strongly recommend you to keep it selected for backup purposes.

6. Select the **Hide projects that already exist in the workspace** check box to hide existing projects from the **Projects** list. This option is available only when you import several projects.
7. From the **Projects** list, select the projects to import and click **Finish** to validate the operation.

Upon successful project import, the names of the imported projects are displayed on the **Project** list of the login window.



You can now select the imported project you want to open in *Talend Studio* and click **Finish** to launch the Studio.



A generation initialization window might come up when launching the application. Wait until the initialization is complete.

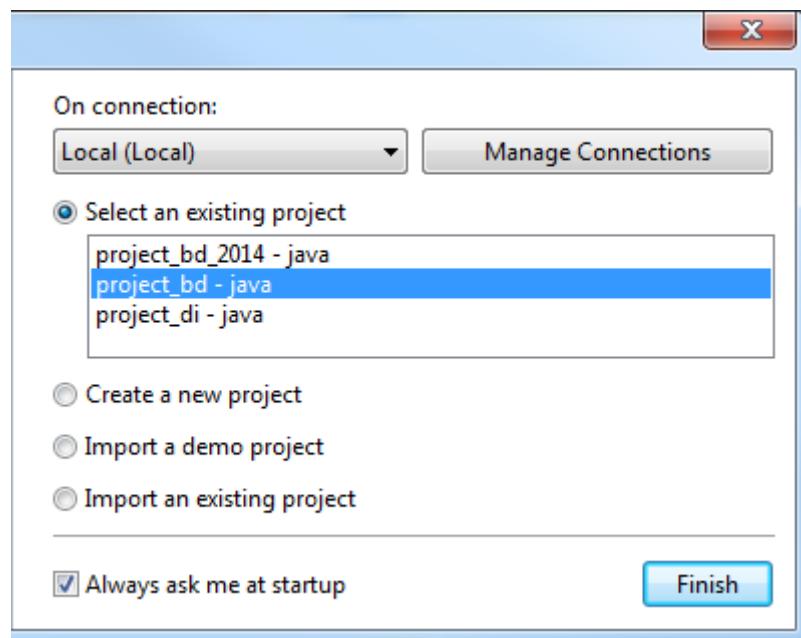
2.5. How to open a project



*When you launch Talend Studio for the first time, no project names are displayed on the **Project** list. First you need to create a project or import a Demo project in order to populate the **Project** list with the corresponding project names that you can then open in the Studio.*

To open a project in *Talend Studio*:

On the Studio login screen, select the project of interest from the project list and click **Finish**.

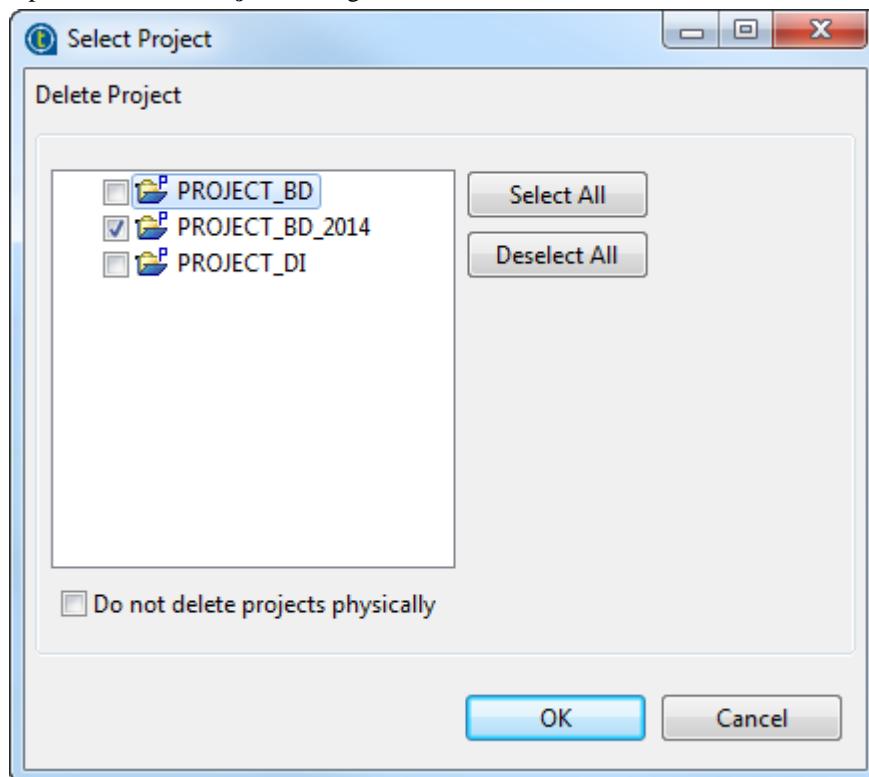


A progress bar appears. Wait until the task is complete and the *Talend Studio* main window opens.

-  When you open a project imported from a previous version of the Studio, an information window pops up to list a short description of the successful migration tasks.

2.6. How to delete a project

1. On the login screen, click **Manage Connections**, then on the dialog box that opens click **Delete Existing Project(s)** to open the [Select Project] dialog box.



2. Select the check box(es) of the project(s) you want to delete.
3. Click **OK** to validate the deletion.

The project list on the login window is refreshed accordingly.



*Be careful, this action is irreversible. When you click **OK**, there is no way to recuperate the deleted project(s).*

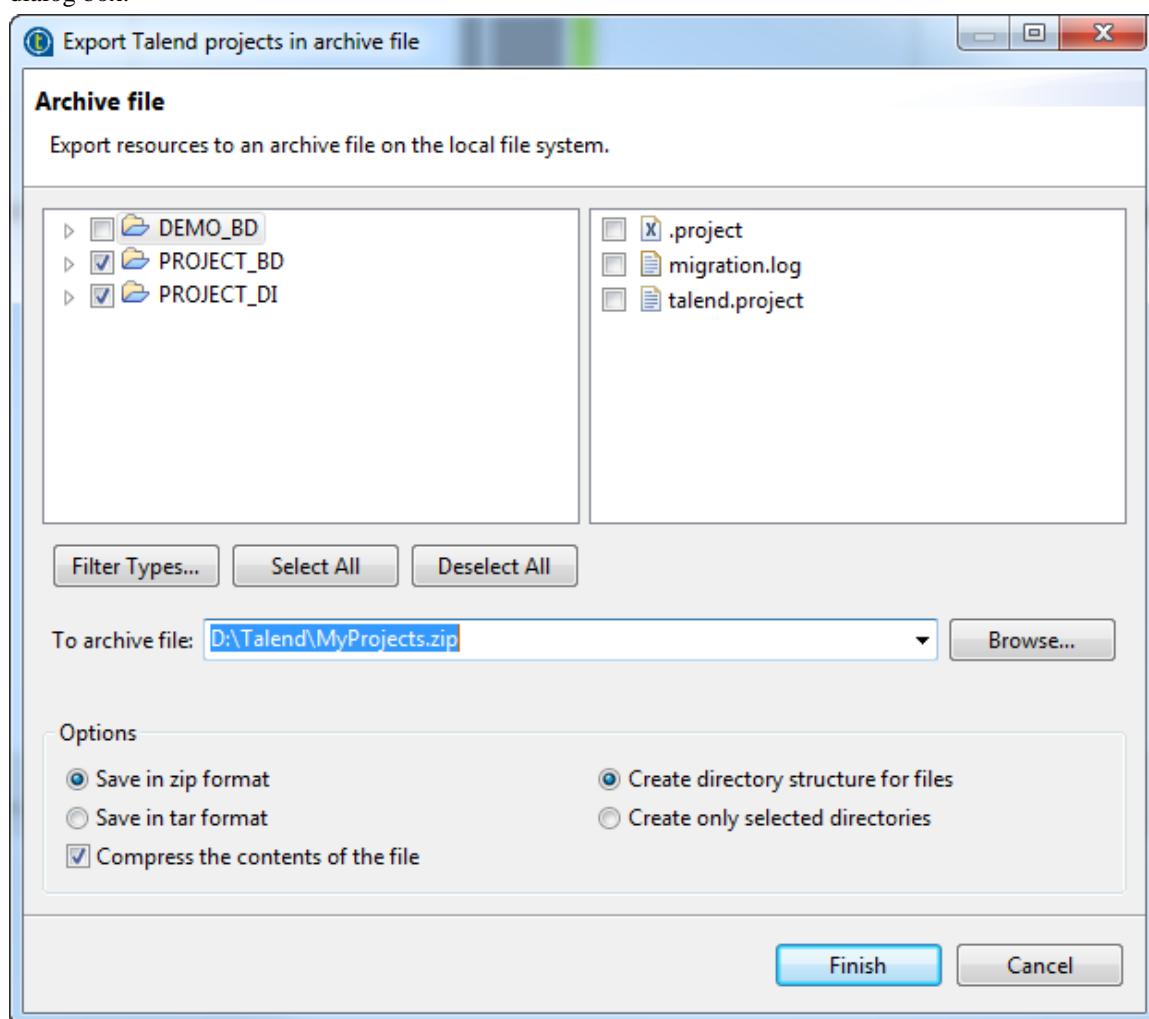


If you select the **Do not delete projects physically** check box, you can delete the selected project(s) only from the project list and still have it/them in the *workspace* directory of *Talend Studio*. Thus, you can recuperate the deleted project(s) any time using the **Import existing project(s) as local** option on the **Project** list from the login window.

2.7. How to export a project

Talend Studio allows you to export projects created or imported in the current instance of *Talend Studio*.

1. On the toolbar of the Studio main window, click to open the **[Export Talend projects in archive file]** dialog box.



2. Select the check boxes of the projects you want to export. You can select only parts of the project through the **Filter Types...** link, if need be (for advanced users).
3. In the **To archive file** field, type in the name of or browse to the archive file where you want to export the selected projects.

4. In the **Option** area, select the compression format and the structure type you prefer.
5. Click **Finish** to validate the changes.

The archived file that holds the exported projects is created in the defined place.



Chapter 3. Designing a Business Model

The **Integration** perspective of *Talend Studio* offers the best tool to formalize business descriptions into building blocks and their relationships. *Talend Studio* allows to design systems, connections, processes and requirements using standardized workflow notation through an intuitive graphical library of shapes and links.

This chapter aims at business managers, decision makers or developers who want to model their flow management needs at a macro level.

3.1. What is a Business Model

Talend's Business Models allow data integration project stakeholders to graphically represent their needs regardless of the technical implementation requirements. Business Models help the IT operation staff understand these expressed needs and translate them into technical processes (Jobs). They typically include both the systems and processes already operating in the enterprise, as well as the ones that will be needed in the future.

Designing Business Models is part of the enterprises' best practices that organizations should adopt at a very early stage of a data integration project in order to ensure its success. Because Business Models usually help detect and resolve quickly project bottlenecks and weak points, they help limit the budget overspendings and/or reduce the upfront investment. Then during and after the project implementation, Business Models can be reviewed and corrected to reflect any required change.

A Business Model is a non technical view of a business workflow need.

Generally, a typical Business Model will include the strategic systems or processes already up and running in your company as well as new needs. You can symbolize these systems, processes and needs using multiple shapes and create the connections among them. Likely, all of them can be easily described using repository attributes and formatting tools.

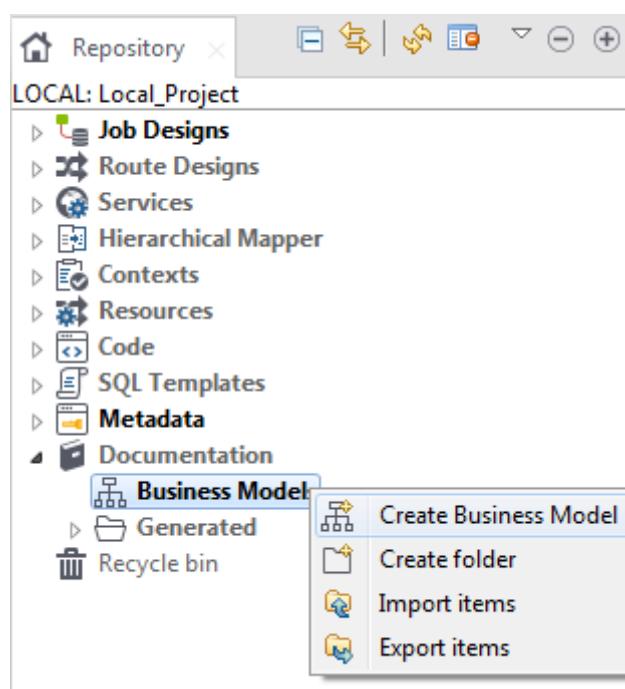
In the design workspace of the **Integration** perspective of *Talend Studio*, you can use multiple tools in order to:

- draw your business needs,
- create and assign numerous repository items to your model objects,
- define the business model properties of your model objects.

3.2. Opening or creating a Business Model

Open *Talend Studio* following the procedure as detailed in the Getting Started Guide.

In the **Repository** tree view of the **Integration** perspective, right-click the Documentation > **Business Models** nodes.



Select **Expand/Collapse** to display all existing Business Models (if any).

3.2.1. How to open a Business Model

Double-click the name of the Business Model to be opened.

The selected Business Model opens up on the design workspace.

3.2.2. How to create a Business Model

To create a business model, proceed as follows:

1. Right-click the **Business Models** node and select **Create Business Model**.

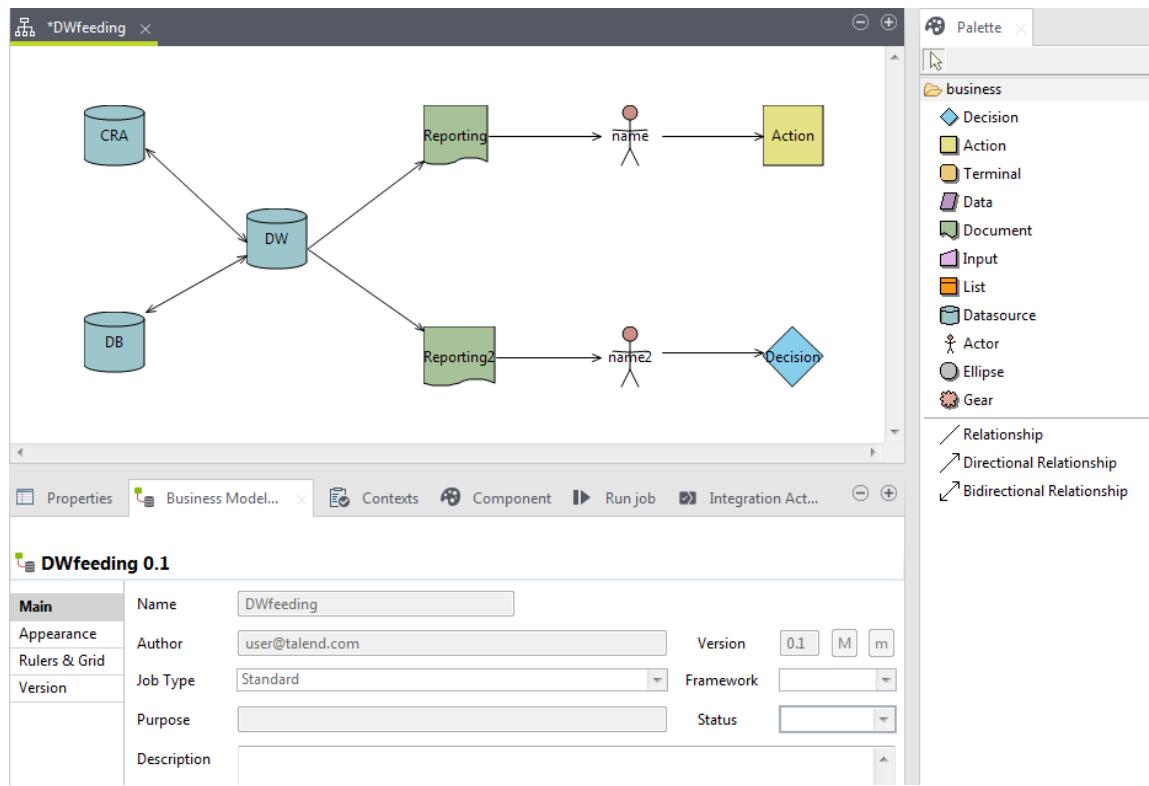
The creation wizard guides you through the steps to create a new Business Model.

2. Enter the Business Model properties according to the following table:

Field	Description
Name	the name of the new Business Model. A message comes up if you enter prohibited characters.
Purpose	Business Model purpose or any useful information regarding the Business Model use.
Description	Business Model description.
Author	a read-only field that shows by default the current user login.
Locker	a read-only field that shows by default the login of the user who owns the lock on the current Job. This field is empty when you are creating a Business Model and has data only when you are editing the properties of an existing Business Model.
Version	a read-only field. You can manually increment the version using the M and m buttons.
Status	a list to select from the status of the Business Model you are creating.
Path	a list to select from the folder in which the Business Model will be created.

3. The **Modeler** opens up on the empty design workspace.

You can create as many models as you want and open them all.



The **Modeler** is made of the following panels:

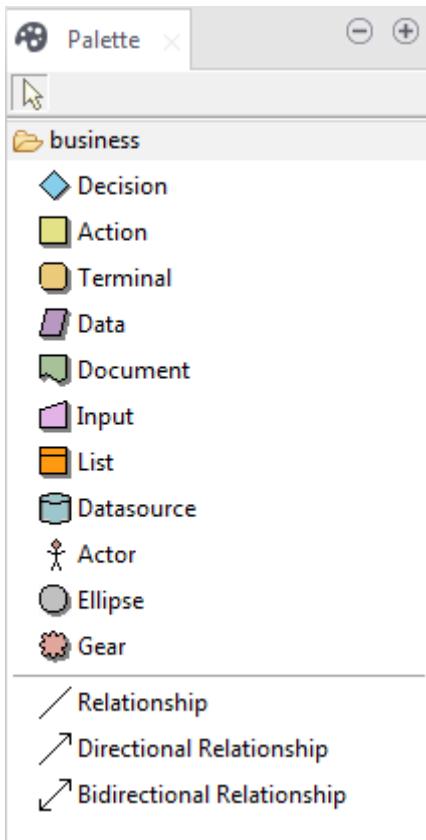
- The **Integration** perspective's design workspace
- a **Palette** of shapes and lines specific to the business modeling
- the **Business Model** panel showing specific information about all or part of the model.

3.3. Modeling a Business Model

If you have multiple tabs opened on your design workspace, click the relevant tab in order to show the appropriate model information.

In the **Business Model** view, you can see information relative to the active model.

Use the **Palette** to drop the relevant shapes on the design workspace and connect them together with branches and arrange or improve the model visual aspect by zooming in or out.



This **Palette** offers graphical representations for *objects* interacting within a Business Model.

The *objects* can be of different types, from strategic system to output document or decision step. Each one having a specific role in your Business Model according to the description, definition and assignment you give to it.

All objects are represented in the **Palette** as *shapes*, and can be included in the model.

Note that you must click the **business** folder to display the library of shapes on the **Palette**.

3.3.1. Shapes

Select the shape corresponding to the relevant *object* you want to include in your Business Model. Double-click it or click the shape in the **Palette** and drop it in the modeling area.

Alternatively, for a quick access to the shape library, keep your cursor still on the modeling area for a couple of seconds to display the quick access toolbar:



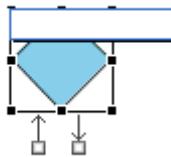
For instance, if your business process includes a decision step, select the diamond shape in the **Palette** to add this decision step to your model.



When you move the pointer over the quick access toolbar, a tooltip helps you to identify the shapes.

Then a simple click will do to make it show on the modeling area.

The shape is placed in a dotted black frame. Pull the corner dots to resize it as necessary.



Also, a blue-edged input box allows you to add a label to the shape. Give an expressive name in order to be able to identify at a glance the role of this shape in the model.

Two arrows below the added shape allow you to create connections with other shapes. You can hence quickly define sequence order or dependencies between shapes.

Related topic: [Connecting shapes](#).

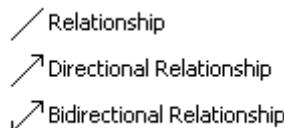
The available shapes include:

Callout	Details
Decision	The diamond shape generally represents an if condition in the model. Allows to take context-sensitive actions.
Action	The square shape can be used to symbolize actions of any nature, such as transformation, translation or formatting.
Terminal	The rounded corner square can illustrate any type of output terminal.
Data	A parallelogram shape symbolize data of any type.
Document	Inserts a Document object which can be any type of document and can be used as input or output for the data processed.
Input	Inserts an input object allowing the user to type in or manually provide data to be processed.
List	forms a list with the extracted data. The list can be defined to hold a certain nature of data.
Database	Inserts a database object which can hold the input or output data to be processed.
Actor	This schematic character symbolizes players in the decision-support as well technical processes.
Ellipse	Inserts an ellipse shape.
Gear	This gearing piece can be used to illustrate pieces of code programmed manually that should be replaced by a Talend Job for example.

3.3.2. Connecting shapes

When designing your Business Model, you want to implement relations between a source shape and a target shape.

There are two possible ways to connect shapes in your design workspace:

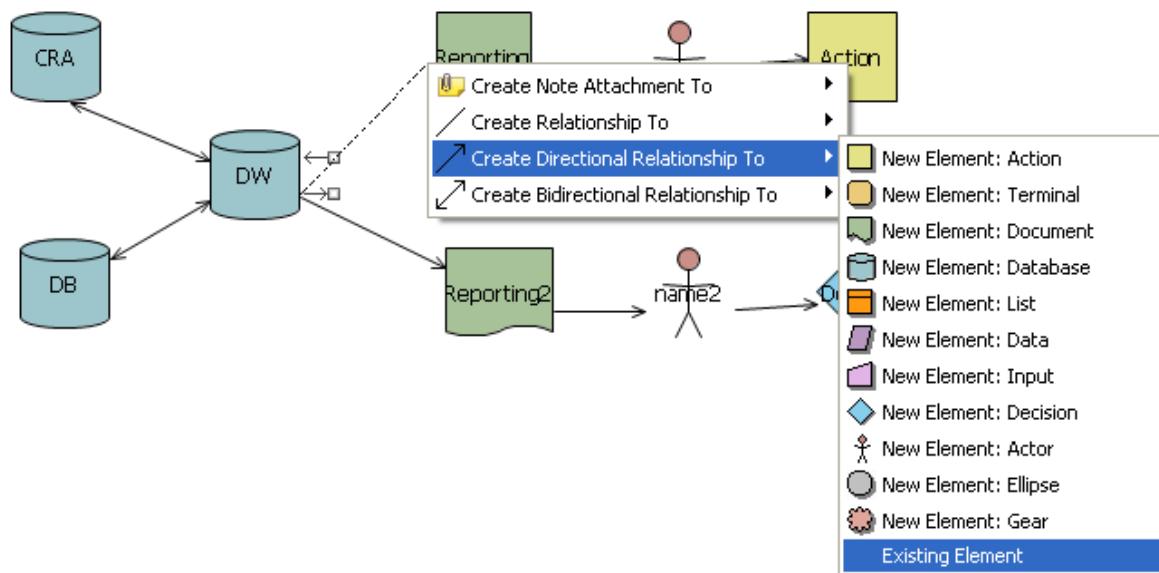


Either select the relevant **Relationship** tool in the **Palette**. Then, in the design workspace, pull a link from one shape to the other to draw a connection between them.

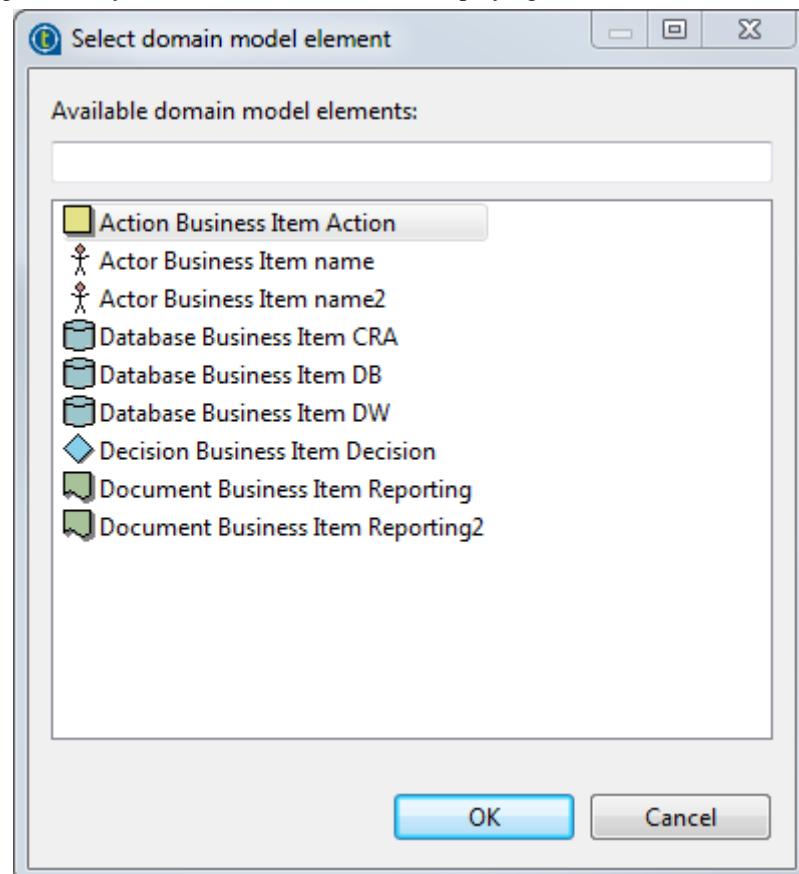
Or, you can implement both the relationship and the element to be related to or from, in a few clicks.

1. Simply move the mouse pointer over a shape that you already dropped on your design workspace, in order to display the double connection arrows.
2. Select the relevant arrow to implement the correct directional connection if need be.

3. Drag a link towards an empty area of the design workspace and release to display the connections popup menu.
4. Select the appropriate connection from the list. You can choose among **Create Relationship To**, **Create Directional Relationship To** or **Create Bidirectional Relationship To**.
5. Then, select the appropriate element to connect to, among the items listed.



You can create a connection to an existing element of the model. Select **Existing Element** in the popup menu and choose the existing element you want to connect to in the displaying list box.



The connection is automatically created with the selected shape.

The nature of this connection can be defined using **Repository** elements, and can be formatted and labelled in the **Properties** panel, see [Business Models](#).

When creating a connection, an input box allows you to add a label to the connection you have created. Choose a meaningful name to help you identify the type of relationship you created.



You can also add notes and comments to your model to help you identify elements or connections at a later date.

Related topic: [How to comment and arrange a model](#).

3.3.3. How to comment and arrange a model

The tools of the **Palette** allow you to customize your model:

Callout	Details
Select	Select and move the shapes and lines around in the design workspace's modeling area.
Zoom	Zoom in to a part of the model. To watch more accurately part of the model. To zoom out, press <i>Shift</i> and click the modeling area.
Note/Text/Note attachment	Allows comments and notes to be added in order to store any useful information regarding the model or part of it.

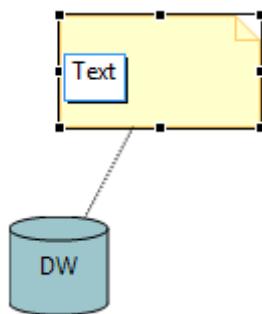
3.3.3.1. How to add a note or free text

To add a note, select the **Note** icon in the **Palette**, docked to the right of the design workspace.

Alternatively right-click the model or the shape you want to link the note to, and select *Add Note*. Or select the Note tool in the quick access toolbar.

A sticky note displays on the modeling area. If the note is linked to a particular shape, a line is automatically drawn to the shape.

Type in the text in the input box or, if the latter does not show, type in directly on the sticky note.

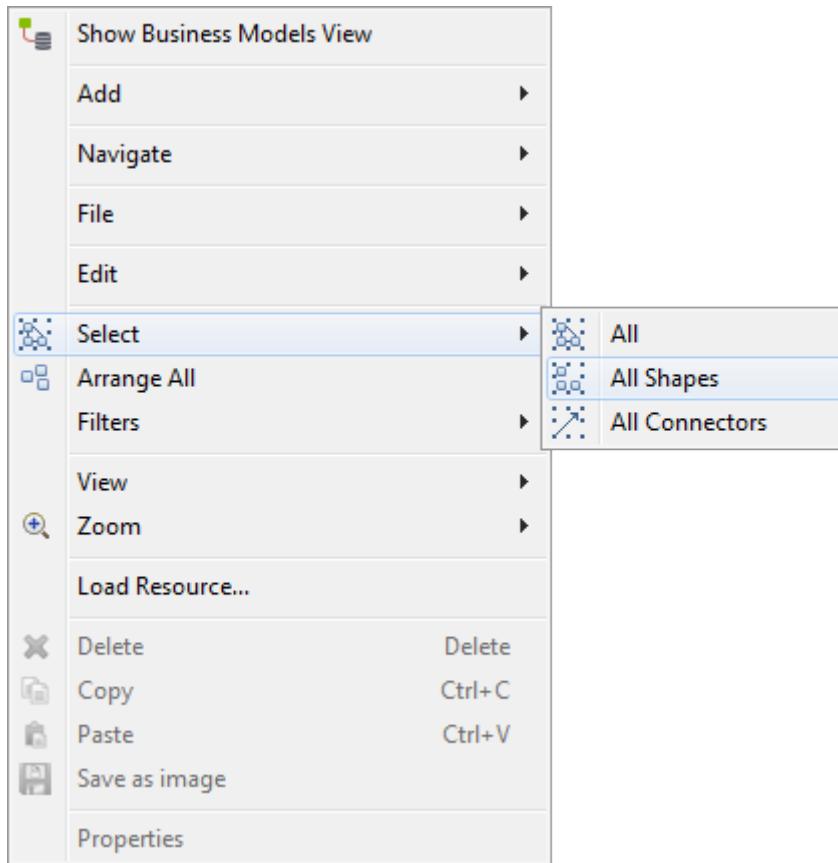


If you want to link your notes and specific shapes of your model, click the down arrow next to the **Note** tool on the **Palette** and select **Note attachment**. Pull the black arrow towards an empty area of the design workspace, and release. The popup menu offers you to attach a new Note to the selected shape.

You can also select the *Add Text* feature to type in free text directly in the modeling area. You can access this feature in the **Note** drop-down menu of the **Palette** or via a shortcut located next to the *Add Note* feature on the quick access toolbar.

3.3.3.2. How to arrange the model view

You can also rearrange the look and feel of your model via the right-click menu.



Place your cursor in the design area, right-click to display the menu and select *Arrange all*. The shapes automatically move around to give the best possible reading of the model.

Alternatively, you can select manually the whole model or part of it.

To do so, right-click any part of the modeling area, and click *Select*.

You can select:

- **All** shapes and connectors of the model,
- **All shapes** used in the design workspace,
- **All connectors** branching together the shapes.

From this menu you can also zoom in and out to part of the model and change the view of the model.

3.3.4. Business Models

The information in the **Business Models** view corresponds to the current selection, if any. This can be the whole model if you selected all shapes of it or more specifically one of the shapes it is made of. If nothing is selected, the **Business Models** tab gives general information about the model.

The **Business Models** view contains different types of information grouped in the **Main**, **Appearance**, **Rules & Grid**, and **Assignment** tabs.

The **Main** tab displays basic information about the selected item in the design workspace, being a Business Model or a Job. For more information about the **Main** tab, see [How to display Job configuration tabs/views](#).

3.3.4.1. Appearance tab

From the **Appearance** tab you can apply filling or border colors, change the appearance of shapes and lines in order to customize your Business Model or make it easier to read.

The **Business Model** view includes the following formats:

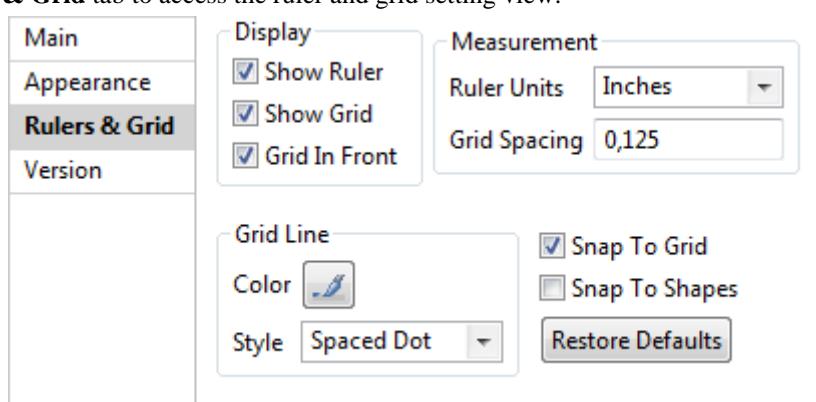
- fill the shape with selected color.
- color the shape border
- insert text above the shape
- insert gradient colors to the shape
- insert shadow to the shape

You can also move and manage shapes of your model using the edition tools. Right-click the relevant shape to access these editing tools.

3.3.4.2. Rulers and Grid tab

To display the **Rulers & Grid** tab, click  on the **Palette**, then click any empty area of the design workspace to deselect any current selection.

Click the **Rulers & Grid** tab to access the ruler and grid setting view.



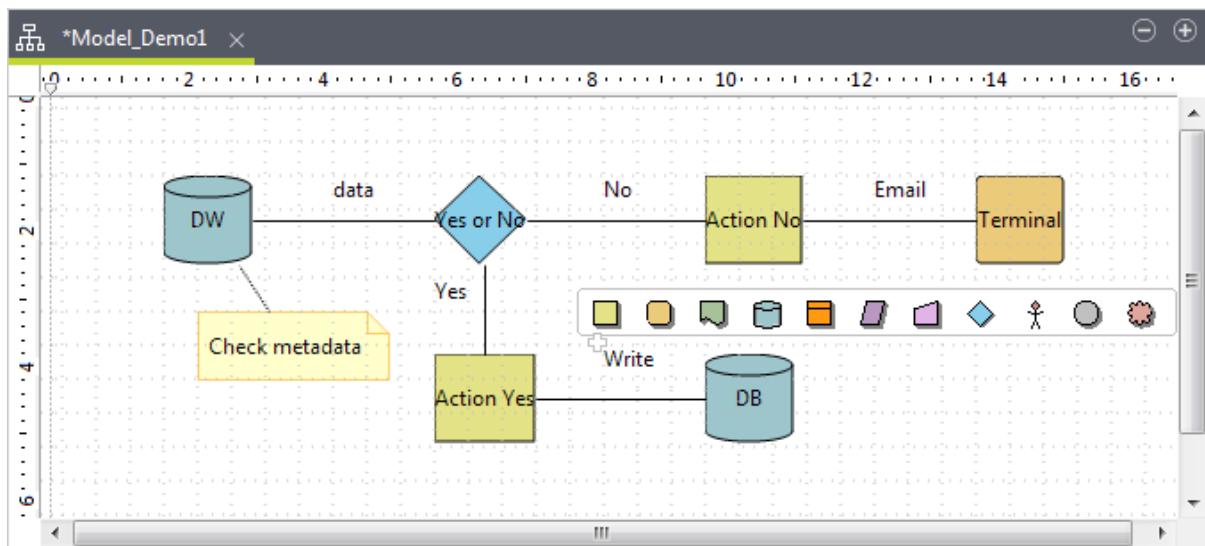
In the **Display** area, select the **Show Ruler** check box to show the **Ruler**, the **Show Grid** check box to show the **Grid**, or both heck boxes. **Grid in front** sends the grid to the front of the model.

In the **Measurement** area, select the ruling unit among **Centimeters**, **Inches** or **Pixels**.

In the **Grid Line** area, click the **Color** button to set the color of the grid lines and select their style from the **Style** list.

Select the **Snap To Grid** check box to bring the shapes into line with the grid or the **Snap To Shapes** check box to bring the shapes into line with the shapes already dropped in the Business Model.

You can also click the **Restore Defaults** button to restore the default settings.



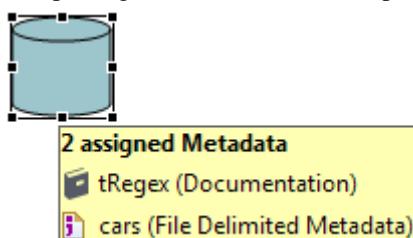
3.3.4.3. Assignment tab

The **Assignment** tab displays in a tabular form details of the **Repository** attributes you allocated to a shape or a connection.

To display any assignment information in the table, select a shape or a connection in the active model, then click the **Assignment** tab in the **Business Model** view.

Type	Name	Comment
Documentation	tRegex	
File Delimited Metadata	cars	

You can also display the assignment list placing the mouse over the shape you assigned information to.

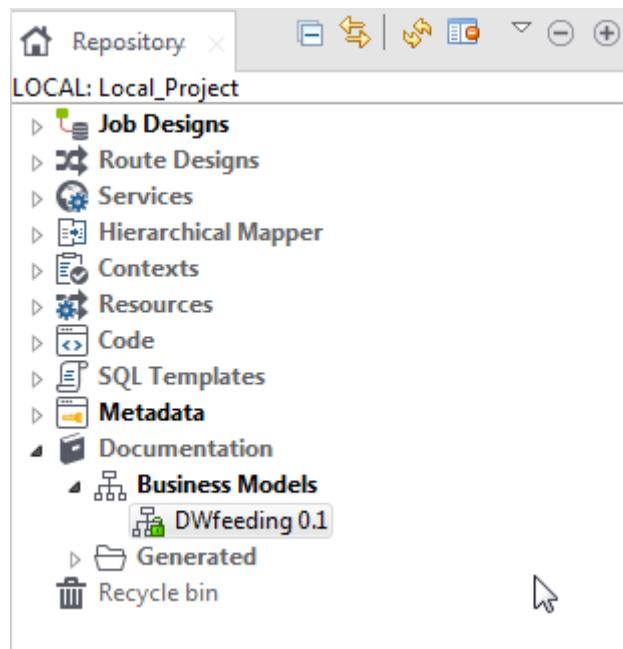


You can modify some information or attach a comment. Also, if you update data from the **Repository** tree view, assignment information gets automatically updated.

For further information about how to assign elements to a Business Model, see [Assigning repository elements to a Business Model](#).

3.4. Assigning repository elements to a Business Model

The **Assignment** tab in the **Business Models** view lists the elements from the **Repository** tree view which have been assigned to a shape in the Business Model.



You can define or describe a particular object in your Business Model by simply associating it with various types of information, for example by adding metadata items.

You can set the nature of the metadata to be assigned or processed, thus facilitating the Job design phase.

To assign a metadata item, simply drop it from the **Repository** tree view to the relevant shape in the design workspace.

The **Assignment** table, located underneath the design workspace, gets automatically updated accordingly with the assigned information of the selected object.

The types of items that you can assign are:

Element	Details
Job designs	If any Job Designs developed for other projects in the same repository are available, you can reuse them as metadata in the active Business Model.
Metadata	You can assign any descriptive data stored in the repository to any of the objects used in the model. It can be connection information to a database for example.
Business Models	You can use in the active model all other Business Models stored in the repository of the same project.
Documentation	You can assign any type of documentation in any format. It can be a technical documentation, some guidelines in text format or a simple description of your databases.
Routines (Code)	If you have developed some routines in a previous project, to automate tasks for example, you can assign them to your Business Model. Routines are stored in the Code folder of the Repository tree view.

For more information about the **Repository** elements, see [Designing a Job](#).

3.5. Editing a Business Model

Follow the relevant procedure according to your needs:

How to rename a Business Model

1. Right-click the relevant Business Model label on the **Repository** tree view then select **Edit properties** to display the corresponding **Main** properties information in the [**Edit properties**] dialog box.
2. Edit the model name in the **Name** field, then click **Finish** to close the dialog box. The model label changes automatically in the **Repository** tree view and will be reflected on the model tab of the design workspace, the next time you open the Business Model.



If the Business Model is open, the information in the [**Edit properties**] dialog box will be read-only so you will not be able to edit them.

How to copy and paste a Business Model

1. In **Repository > Business model**, right-click the Business Model name to be copied and select **Copy** in the popup menu, or press **Ctrl+C**.
2. Then right-click where you want to paste your Business Model, and select **Paste**.

How to move a Business Model

1. To move a Business Model from a location to another in your business models project folder, select a Business Model in the **Repository > Business Models** tree.
2. Then simply drop it to the new location.

How to delete a Business Model

1. Right-click the name of the model to be deleted and select **Delete** in the popup menu.
2. Alternatively, simply select the relevant Business Model, then drop it into the **Recycle bin** of the **Repository** tree view.

3.6. Saving a Business Model

To save a Business Model, click **File > Save** or press **Ctrl+S**. The model is saved under the name you gave during the creation process.

An asterisk displays in front of the Business Model name on the tab to indicate that changes have been made to the model but not yet saved.



To save a Business Model and increment its version at the same time:

1. Click **File>Save as....** The [**Save as**] dialog box displays.

2. Next to the **Version** field, click the **M** button to increment the major version and the **m** button to increment the minor version.
3. Click **Finish** to validate the modification

 By default, when you open a Business Model, you open its last version. Any previous version of the Business Model is read-only and thus cannot be modified.

You can access a list of the different versions of a Business Model and perform certain operations. To do that:

1. In the **Repository** tree view, select the Business Model you want to consult the versions of.
2. Click the **Business Models>Version** in succession to display the version list of the selected Job.
3. Right-click the Business Model version you want to consult.
4. Do one of the followings:

Select	To...
Edit properties	edit Business Model properties. Note: The Business Model should not be open on the design workspace, otherwise it will be in read-only mode.
Read Business Model	consult the Business Model in read-only mode.

 You can open and modify the last version of a Business Model from the **Version** view if you select **Edit Business Model** from the drop-down list.



Chapter 4. Designing a Job

Talend Studio is the tool with the capabilities that treat all of the different sources and targets required in data integration processes and all other associated operations.

Via *Talend Studio*, you are able to design data integration Jobs that allow you to put in place up and run dataflow management processes.

This chapter addresses the needs of programmers or IT managers who are ready to implement the technical aspects of a Business Model (regardless of whether it was designed in the Business Modeler of the **Integration** perspective of *Talend Studio*).

4.1. What is a Job design

A Job Design is the runnable layer of a business model. It is a graphical design, of one or more components connected together, that allows you to set up and run dataflow management processes. A Job Design translates business needs into code, routines and programs, in other words it technically implements your data flow.

The Jobs you design can address all of the different sources and targets that you need for data integration processes and any other related process.

When you design a Job in *Talend Studio*, you can:

- put in place data integration actions using a library of technical components.
- change the default setting of components or create new components or family of components to match your exact needs.
- set connections and relationships between components in order to define the sequence and the nature of actions.
- access code at any time to edit or document the components in the designed Job.
- create and add items to the repository for reuse and sharing purposes (in other projects or Jobs or with other users).



In order to be able to execute the Jobs you design in Talend Studio, you need to install an Oracle JVM 1.8 (IBM JVM is not supported). You can download it from <http://www.oracle.com/technetwork/java/javase/downloads/index.html>.

4.2. Getting started with a basic Job

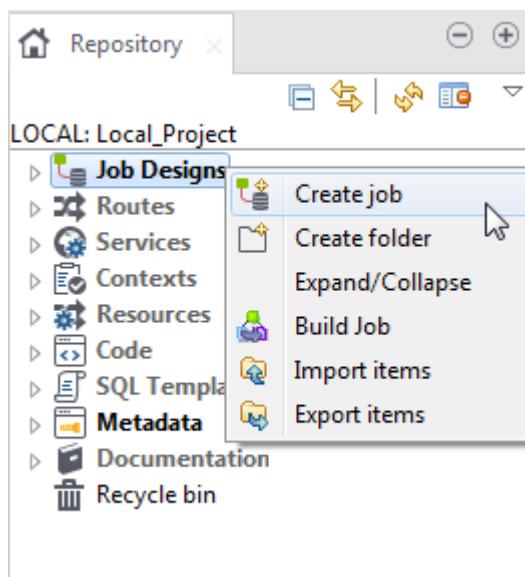
This section provides a continuous example that will help you create, add components to, configure, and execute a simple Job. This Job will be named *A_Basic_Job* and will read a text file, display its content on the **Run** console, and then write the data into another text file.

4.2.1. Creating a Job

Talend Studio enables you to create a Job by dropping different technical components from the **Palette** onto the design workspace and then connecting these components together.

To create the example Job described in this section, proceed as follows:

1. In the **Repository** tree view of the **Integration** perspective, right-click the **Job Designs** node and select **Create job** from the contextual menu.



The [New Job] wizard opens to help you define the main properties of the new Job.

New Job

Add a job in the repository

Name	A_Basic_Job
Purpose	This Job aims to create a Basic Job.
Description	This Job is a simple design to present the basic functionalities and configuration of a Talend Job.
Author	user@talend.com
Locker	
Version	0.1
Status	
Path	Select
<input type="button" value="Finish"/> <input type="button" value="Cancel"/>	

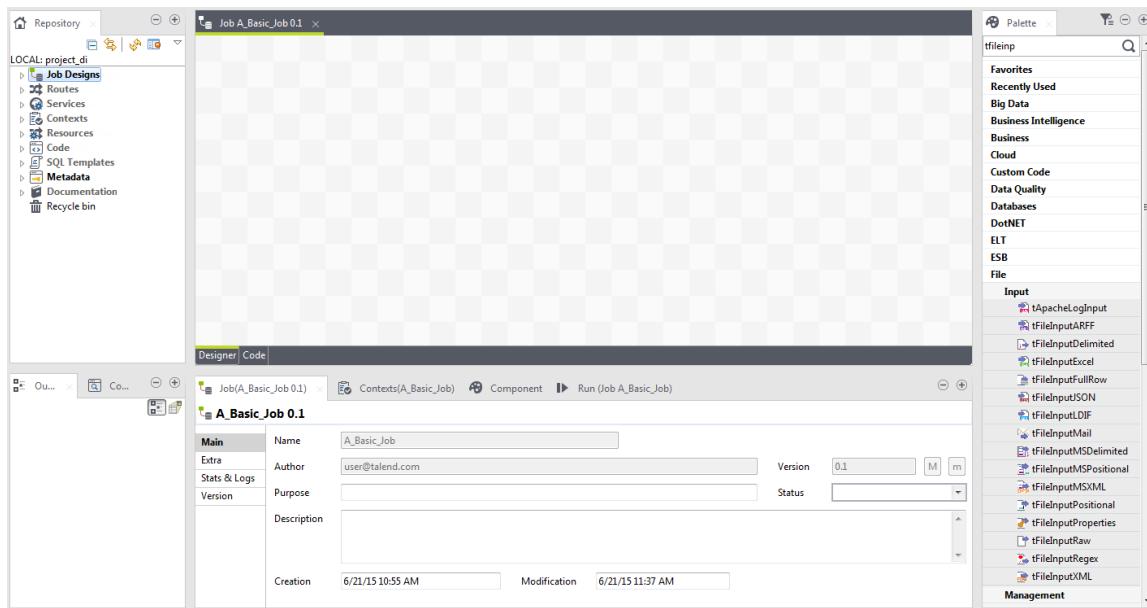
- Fill the Job properties as shown in the previous screenshot.

The fields correspond to the following properties:

Field	Description
Name	the name of the new Job. Note that a message comes up if you enter prohibited characters.
Purpose	Job purpose or any useful information regarding the Job use.

Field	Description
Description	Job description containing any information that helps you describe what the Job does and how it does it.
Author	a read-only field that shows by default the current user login.
Locker	a read-only field that shows by default the login of the user who owns the lock on the current Job. This field is empty when you are creating a Job and has data only when you are editing the properties of an existing Job.
Version	a read-only field. You can manually increment the version using the M and m buttons. For more information, see Managing Job versions .
Status	a list to select from the status of the Job you are creating.
Path	a list to select from the folder in which the Job will be created.

3. An empty design workspace opens up showing the name of the Job as a tab label.



The Job you created is now listed under the **Job Designs** node in the **Repository** tree view.

You can open one or more of the created Jobs by simply double-clicking the Job label in the **Repository** tree view.

4.2.2. Adding components to the Job

Now that the Job is created, components have to be added to the design workspace, a **tFileInputDelimited**, a **tLogRow**, and a **tFileOutputDelimited** in this example.

There are several ways to add a component onto the design workspace. You can:

- find your component on the **Palette** by typing the search keyword(s) in the search field of the **Palette** and drop it onto the design workspace.
- add a component by directly typing your search keyword(s) on the design workspace.
- add an output component by dragging from an input component already existing on the design workspace.
- drag and drop a centralized metadata item from the **Metadata** node onto the design workspace, and then select the component of interest from the **Components** dialog box.

This section describes the first three methods. For details about how to drop a component from the **Metadata** node, see [Managing Metadata](#).

4.2.2.1. Dropping the first component from the Palette

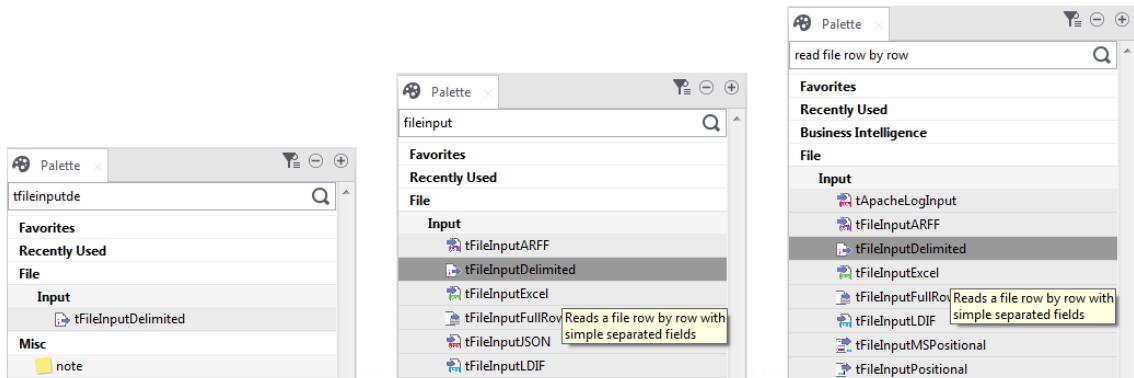
The first component of this example will be added from the **Palette**. This component defines the first task executed by the Job. In this example, as you first want to read a text file, you will use the **tFileInputDelimited** component.

To drop a component from the **Palette**, proceed as follows:

1. Enter the search keyword(s) in the search field of the **Palette** and press **Enter** to validate your search.

The keyword(s) can be the partial or full name of the component, or a phrase describing its functionality if you don't know its name, for example, *tfileinputde*, *fileinput*, or *read file row by row*. The **Palette** filters to the only families where the component can be found. If you cannot find the **Palette** view in the Studio, see [How to change the Palette layout and settings](#).

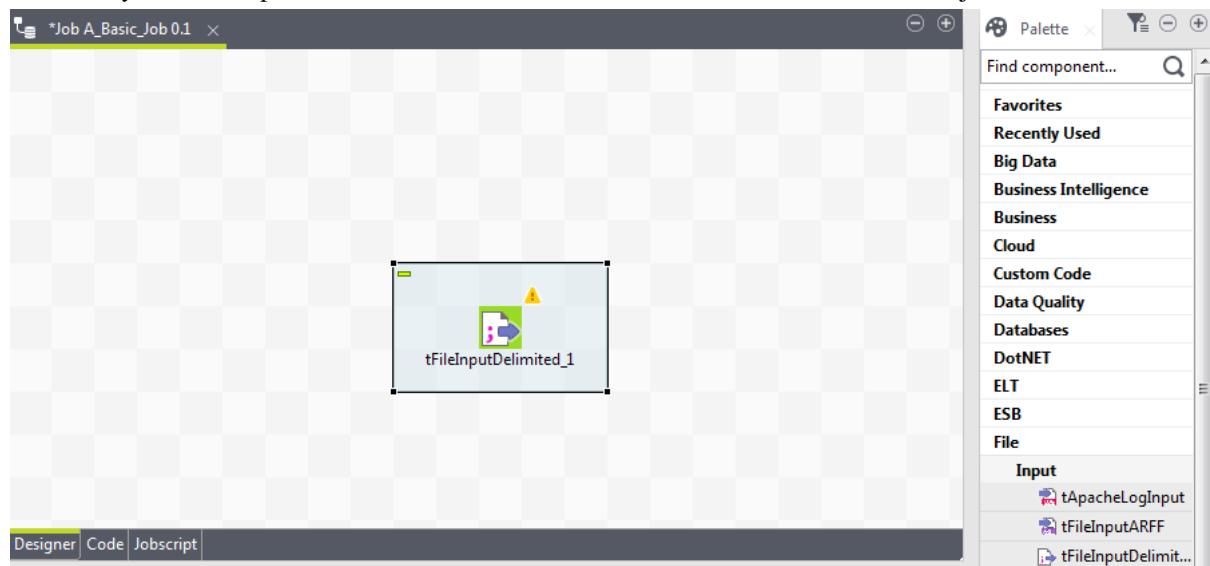
 To use a descriptive phrase as keywords for a fuzzy search, make sure the **Also search from Help when performing a component searching** check box is selected on the **Preferences > Palette Settings** view. For more information, see [Palette preferences \(Talend > Palette Settings\)](#).



2. Select the component you want to use and click on the design workspace where you want to drop the component.

Note that you can also drop a note to your Job the same way you drop components.

Each newly-added component is shown in a blue box to show that it as an individual Subjob.



4.2.2.2. Adding the second component by typing on the design workspace

The second component of our Job will be added by typing its name directly on the workspace, instead of dropping it from the **Palette** or from the **Metadata** node.

Prerequisite: Make sure you have selected the **Enable Component Creation Assistant** check box in the Studio preferences. For more information, see [How to use centralized metadata in a Job](#).

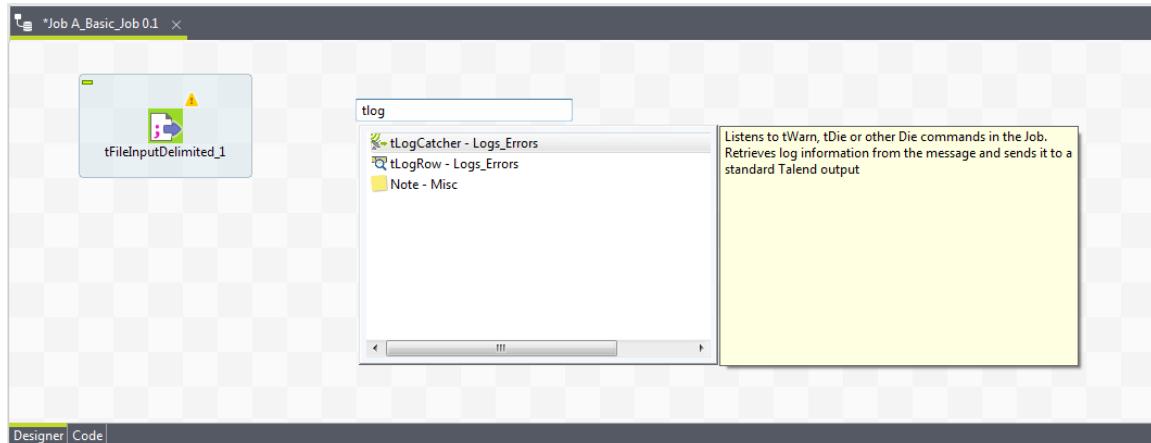
To add a component directly on the workspace, proceed as follows:

1. Click where you want to add the component on the design workspace, and type your keywords, which can be the full or partial name of the component, or a phrase describing its functionality if you don't know its name. In our example, start typing **tlog**.



To use a descriptive phrase as keywords for a fuzzy search, make sure the **Also search from Help when performing a component searching** check box is selected on the **Preferences > Palette Settings** view. For more information, see [Palette preferences \(Talend > Palette Settings\)](#).

A list box appears below the text field displaying all the matching components in alphabetical order.



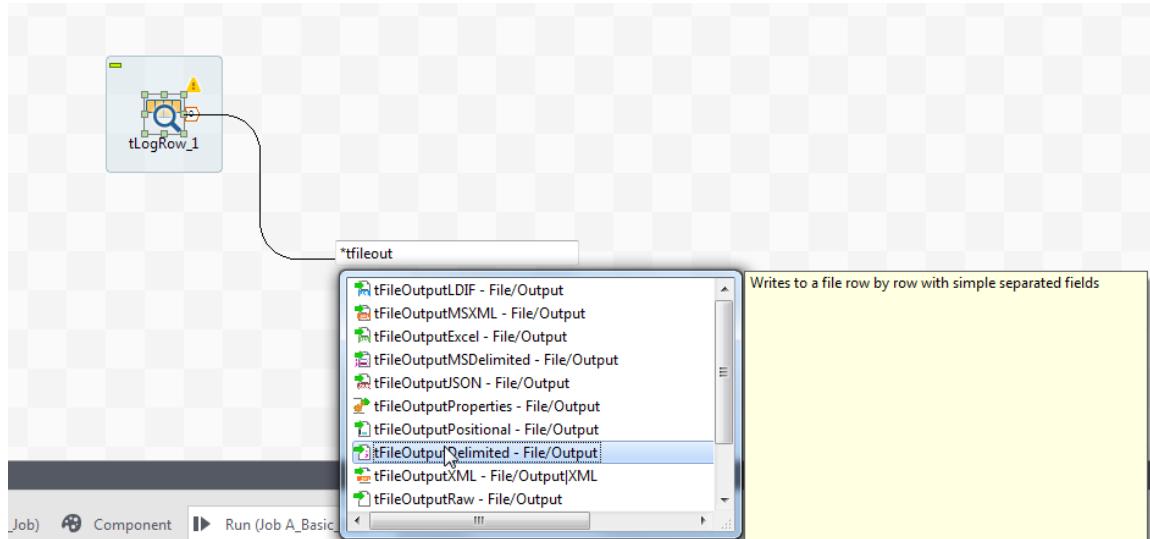
2. Double-click the desired component to add it on the workspace, **tLogRow** in our example.

4.2.2.3. Adding an output component by dragging from an input one

Now you will add the third component, a **tFileOutputDelimited**, to write the data read from the source file into another text file. We will add the component by dragging from the **tLogRow** component, which serves as an input component to the new one to be added.

1. Click the **tLogRow** component to show the **o** icon docked to it.
2. Drag and drop the **o** icon where you want to add a new component.

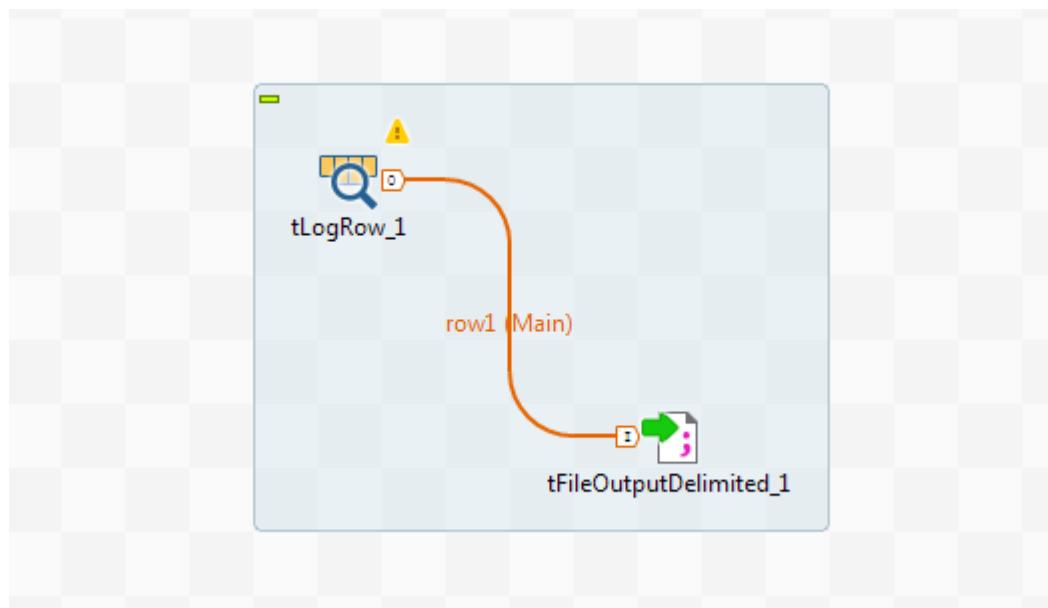
A text field and a component list appear. The component list shows all the components that can be connected with the input component.



- To narrow the search, type in the text field the name of the component you want to add or part of it, or a phrase describing the component's functionality if you don't know its name, and then double-click the component of interest, **tFileOutputDelimited** in this example, on the component list to add it onto the design workspace. The new component is automatically connected with the input component **tLogRow**, using a **Row > Main** connection.



To use a descriptive phrase as keywords for a fuzzy search, make sure the **Also search from Help when performing a component searching** check box is selected on the **Preferences > Palette Settings** view. For more information, see *Palette preferences (Talend > Palette Settings)*.



4.2.3. Connecting the components together

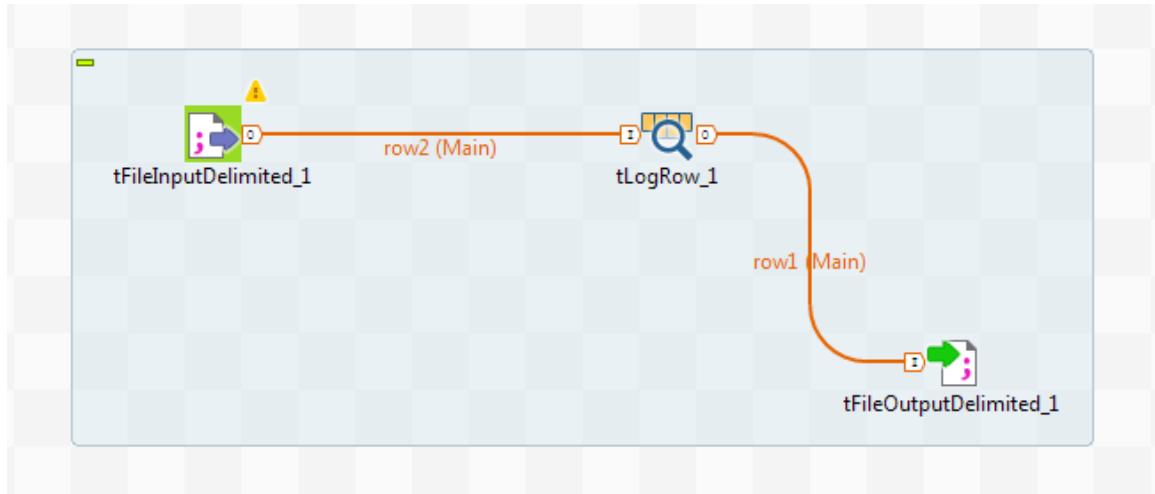
Now that the components have been added on the workspace, they have to be connected together. Components connected together form a subjob. Jobs are composed of one or several subjobs carrying out various processes.

In this example, as the **tLogRow** and **tFileOutputDelimited** components are already connected, you only need to connect the **tFileInputDelimited** to the **tLogRow** component.

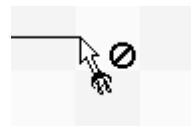
To connect the components together, use either of the following methods:

Right-click and click again

1. Right-click the source component, **tFileInputDelimited** in this example.
2. In the contextual menu that opens, select the type of connection you want to use to link the components, **Row > Main** in this example.
3. Click the target component to create the link, **tLogRow** in this example.



Note that a black crossed circle is displayed if the target component is not compatible with the link.

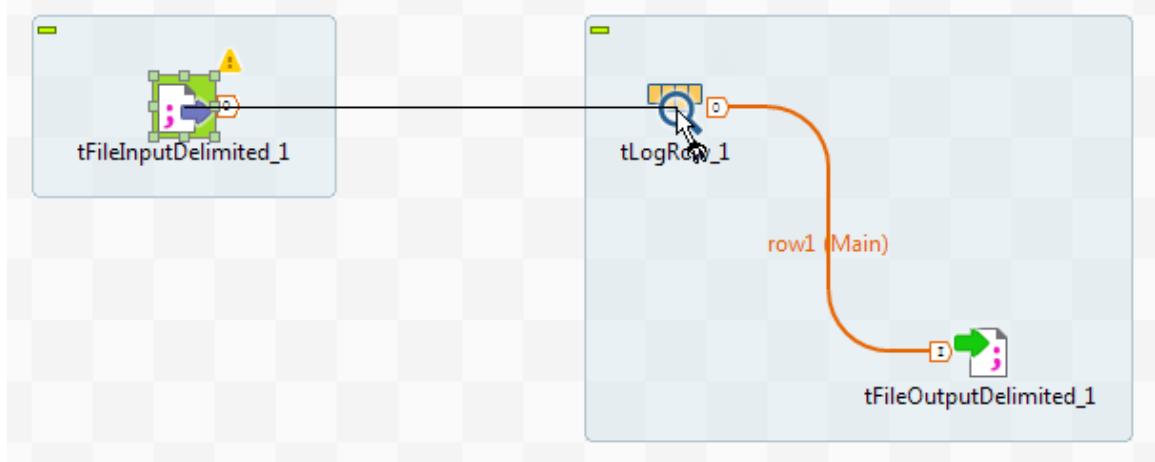


According to the nature and the role of the components you want to link together, several types of link are available. Only the authorized connections are listed in the contextual menu.

Drag and drop

1. Click the input component, **tFileInputDelimited** in this example.
2. When the **O** icon appears, click it and drag the cursor to the destination component, **tLogRow** in this example.

A **Row > Main** connection is automatically created between the two components.



While this method requires less operation steps, it works only with these types of **Row** connections: **Main**, **Lookup**, **Output**, **Filter**, and **Reject**, depending on the nature and role of the components you are connecting.

You can also drop components in the middle of a **Row** link. For more information, see [How to add a component between two connected components](#).

For more information on using various types of connections, see [Using connections](#).

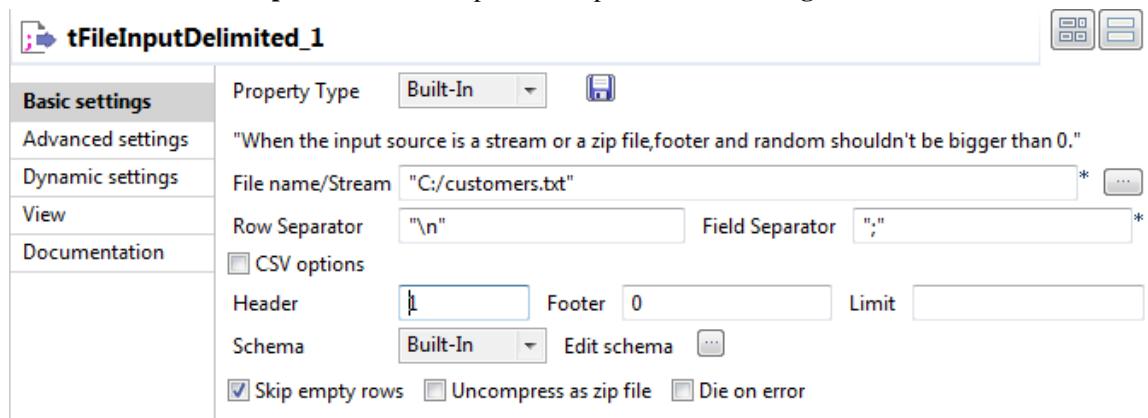
4.2.4. Configuring the components

Now that the components are linked, their properties should be defined.

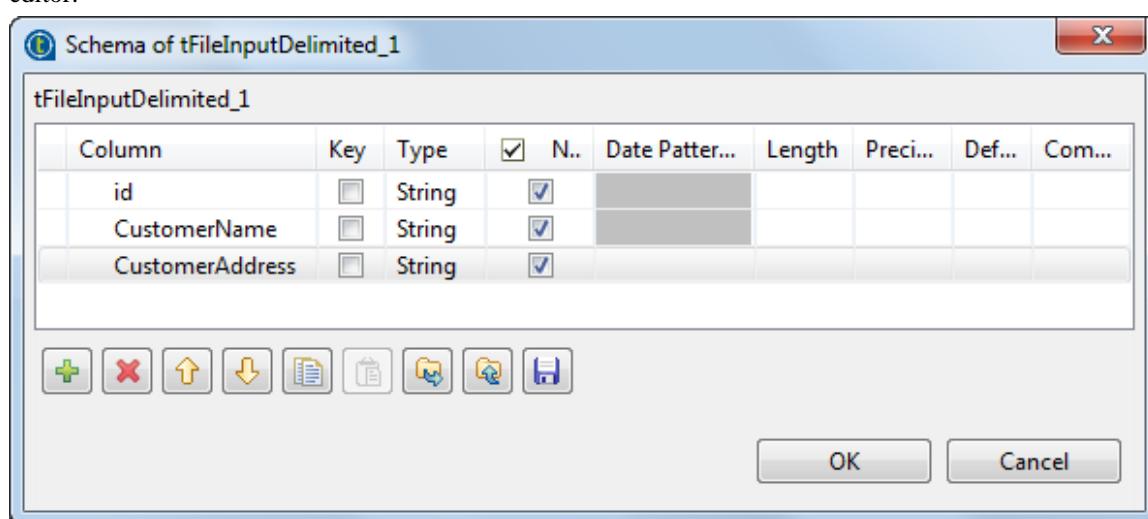
For more advanced details regarding the components properties, see [How to define component properties](#).

Configuring the tFileInputDelimited component

1. Double-click the **tFileInputDelimited** component to open its **Basic settings** view.

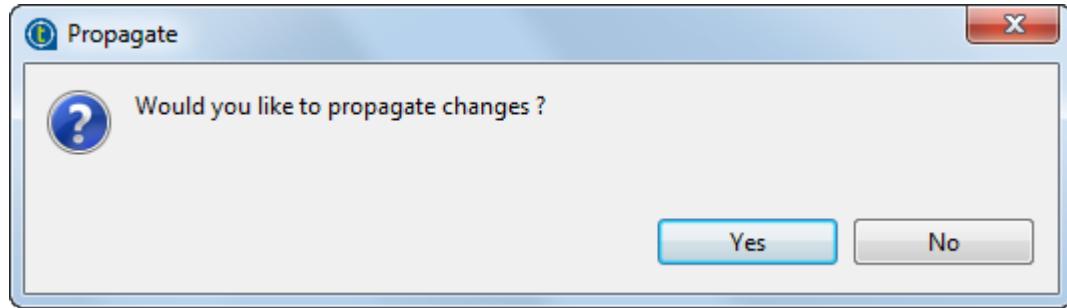


2. Click the [...] button next to the **File Name/Stream** field.
3. Browse your system or enter the path to the input file, *customers.txt* in this example.
4. In the **Header** field, enter *1*.
5. Click the [...] button next to **Edit schema**.
6. In the Schema Editor that opens, click three times the [+] button to add three columns.
7. Name the three columns *id*, *CustomerName* and *CustomerAddress* respectively and click **OK** to close the editor.



8. In the pop-up that opens, click **OK** accept the propagation of the changes.

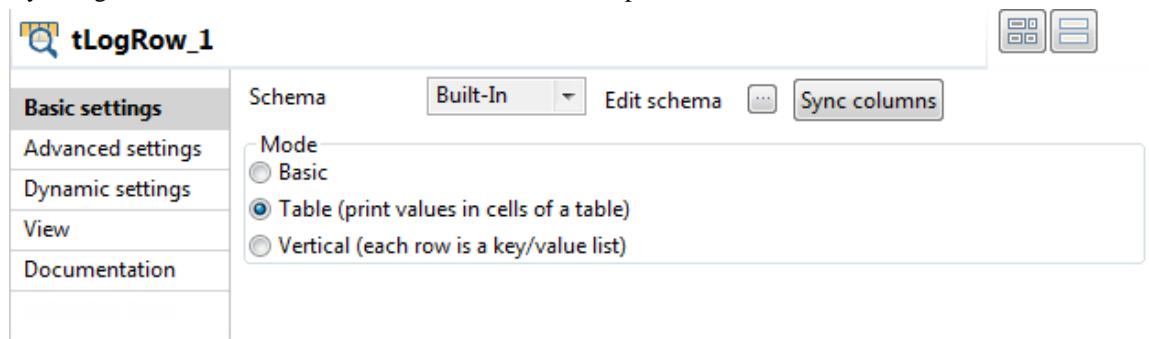
This allows you to copy the schema you created to the next component, **tLogRow** in this example.



Configuring the tLogRow component

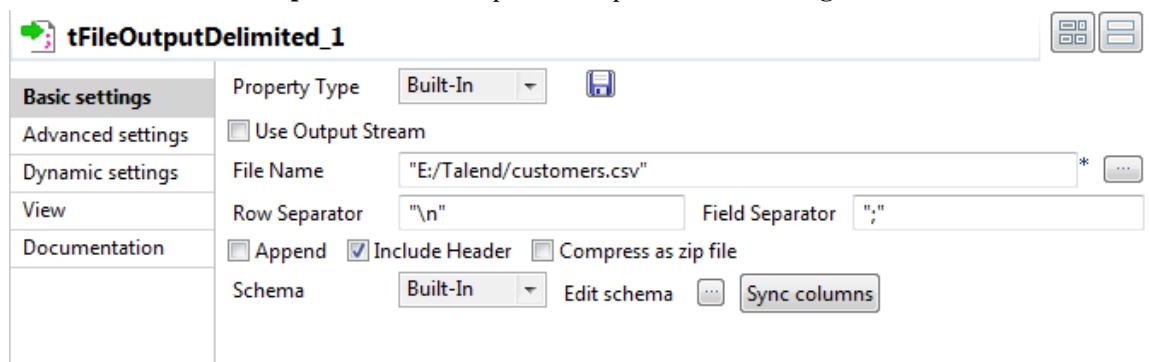
1. Double-click the **tLogRow** component to open its **Basic settings** view.
2. In the **Mode** area, select **Table (print values in cells of a table)**.

By doing so, the contents of the *customers.txt* file will be printed in a table and therefore more readable.



Configuring the tFileOutputDelimited component

1. Double-click the **tFileOutputDelimited** component to open its **Basic settings** view.



2. Click the [...] button next to the **File Name** field.
3. Browse your system or enter the path to the output file, *customers.csv* in this example.
4. Select the **Include Header** check box.
5. If needed, click the **Sync columns** button to retrieve the schema from the input component.

4.2.5. Executing the Job

Now that components are configured, the Job can be executed.

To do so, proceed as follows:

1. Press **Ctrl+S** to save the Job.
2. Go to **Run** tab, and click on **Run** to execute the Job.

The file is read row by row and the extracted fields are displayed on the **Run** console and written to the specified output file.

```

Starting job A_Basic_Job at 11:37 21/06/2015.
[statistics] connecting to socket on port 3648
[statistics] connected
-----+-----+
-----+-----+
| id|CustomerName          |CustomerAddress
|---+-----+-----+
| 1 | Griffith Paving and Sealcoatin | talend@apres91
| 2 | Bill's Dive Shop           | 511 Maple Ave. Apt. 1B
| 3 | Childress Child Day Care   | 662 Lyons Circle
| 4 | Facelift Kitchen and Bath  | 220 Vine Ave.
| 5 | Terrinni & Son Auto and Truck | 770 Exmoor Rd
| 6 | Kermitt the Pet Shop       | 1860 Parkside Ln.
| 7 | Tub's Furniture Store      | 807 Old Trail Rd.
| 8 | Toggle & Myerson Ltd     | 618 Sheridan rd.
| 9 | Childress Child Day Care   | 788 Tennyson Ave.
| 10| Elle Hypnosis and Therapy Cent | 2032 Northbrook Ct.
-----+-----+
[statistics] disconnected
Job A_Basic_Job ended at 11:37 21/06/2015. [exit code=0]

```

Line limit Wrap

4.3. Working with components

The sections below give detailed information about various subjects related to handling components in a data integration Job, including:

- [How to add a component between two connected components](#)
- [How to define component properties](#)
- [How to define the start component](#)
- [How to find Jobs containing a specific component](#)
- [How to set default values in the schema of a component](#)

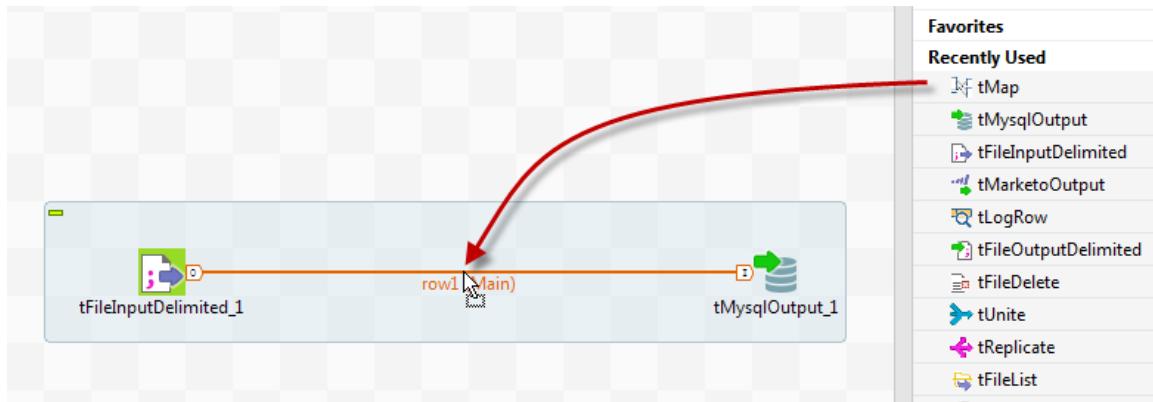
4.3.1. How to add a component between two connected components

When designing a Job, you can insert a component between two components linked by a **Row** connection, provided that the new component can serve as a middle component between the two.

The examples below show different options for you to insert a **tMap** between a **tFileInputDelimited** and **tMySqlOutput** linked by a **Row > Main** connection. For how to connect components in a Job, see [Connecting the components together](#). For more information about various types of connections, see [Connection types](#).

Dropping the component from the Palette onto the connection

1. From the **Palette**, locate and select **tMap**.
2. Drag the component and drop it onto the **Row** connection.

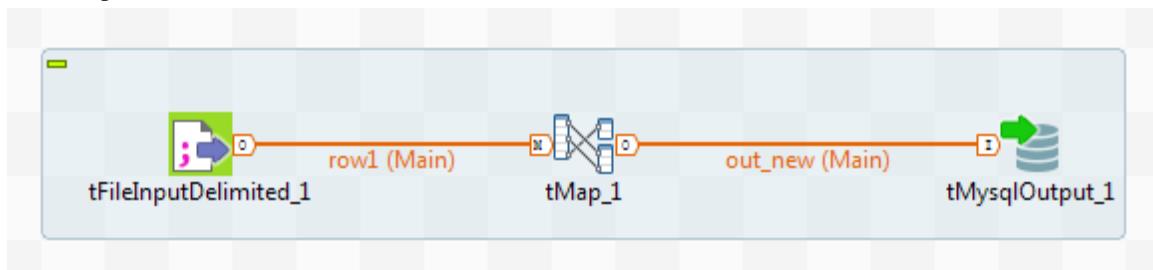


If you are prompted to give a name to the output connection from the newly added component, which is true in the case of a **tMap**, type in a name and click **OK** to close the dialog box.



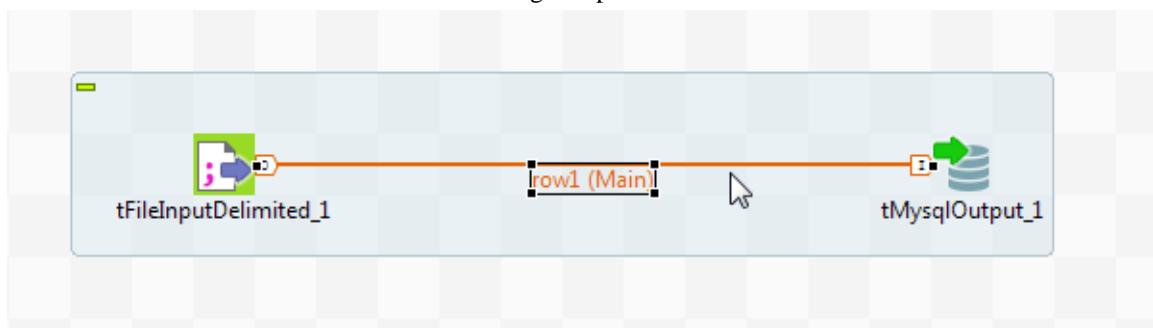
You may be asked to retrieve the schema of the target component. In that case, click **OK** to accept or click **No** to deny.

The component is inserted in the middle of the connection, which is now divided into two connections.

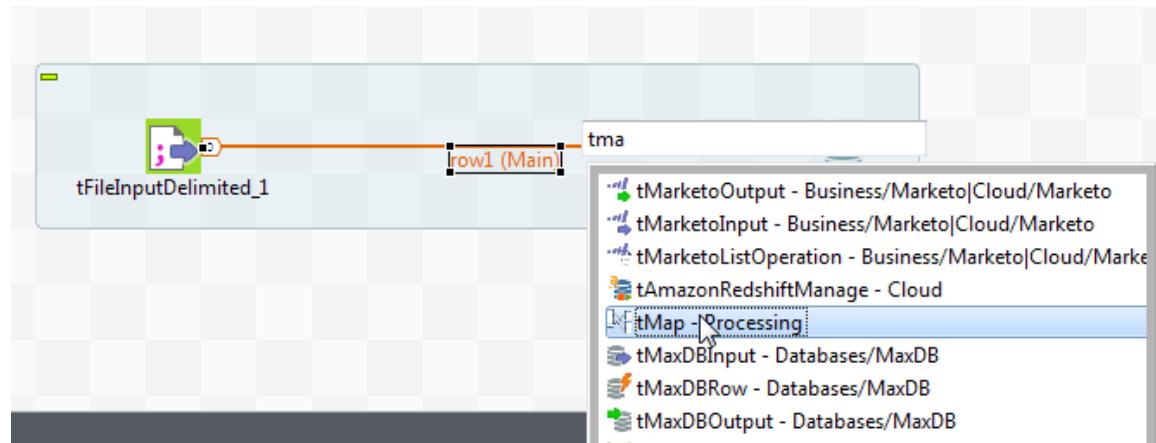


Adding the component by typing on the connection

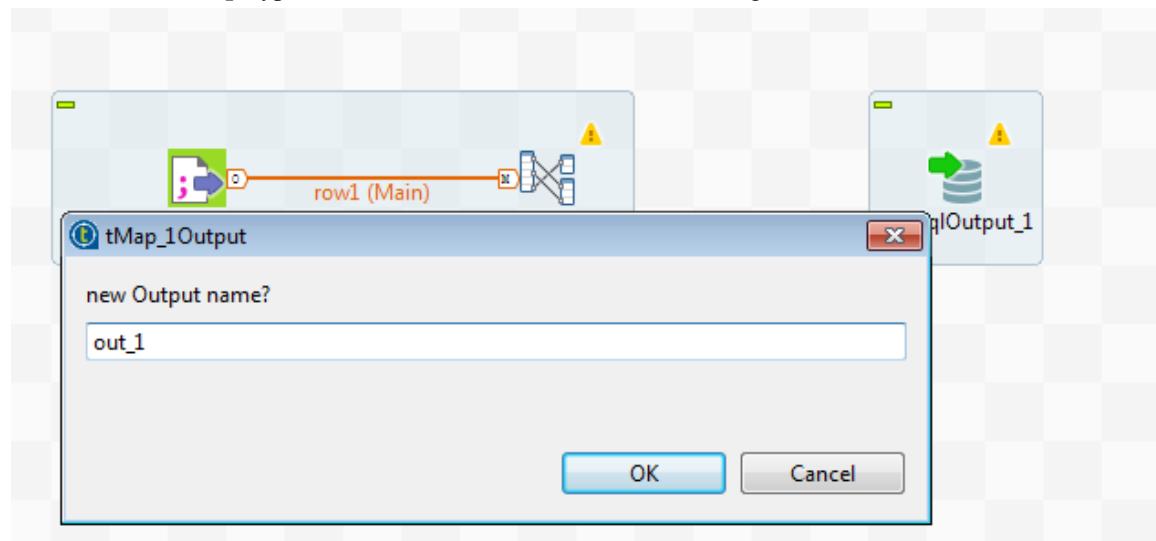
1. Click on the connection that links the two existing components to select it.



2. Type the name of the new component you want to add, **tMap** in this example, and double click the component on the suggested list to add it onto the connection.

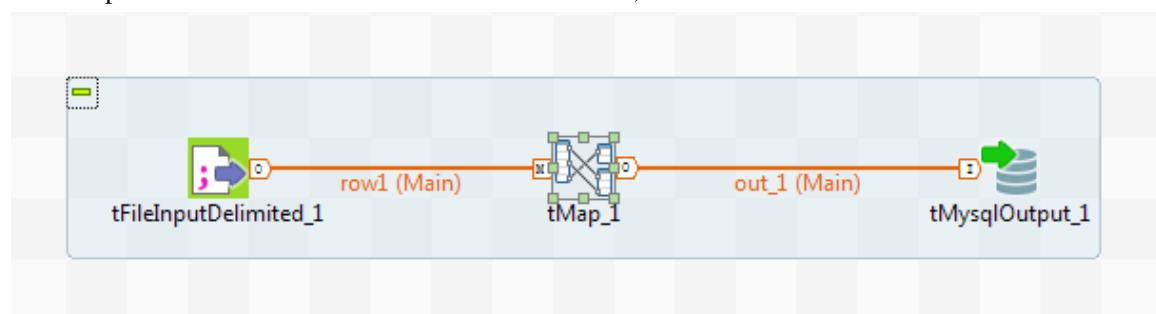


3. If you are prompted to give a name to the output connection from the newly added component, which is true in the case of a **tMap**, type in a name and click **OK** to close the dialog box.



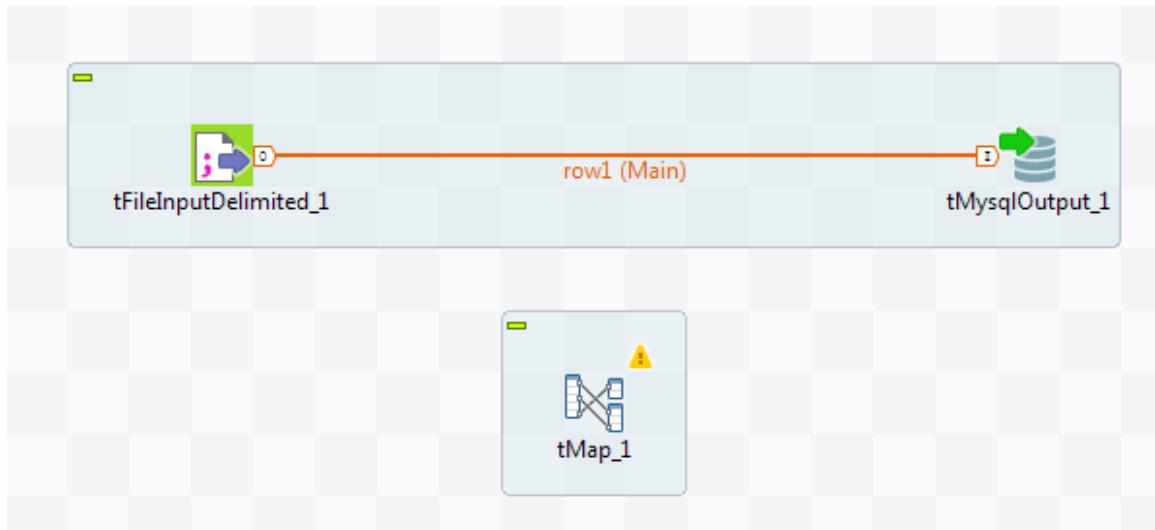
You may be asked to retrieve the schema of the target component. In that case, click **OK** to accept or click **No** to deny.

The component is inserted in the middle of the connection, which is now divided in two connections.



Adding the component to the design workspace and moving the existing connection

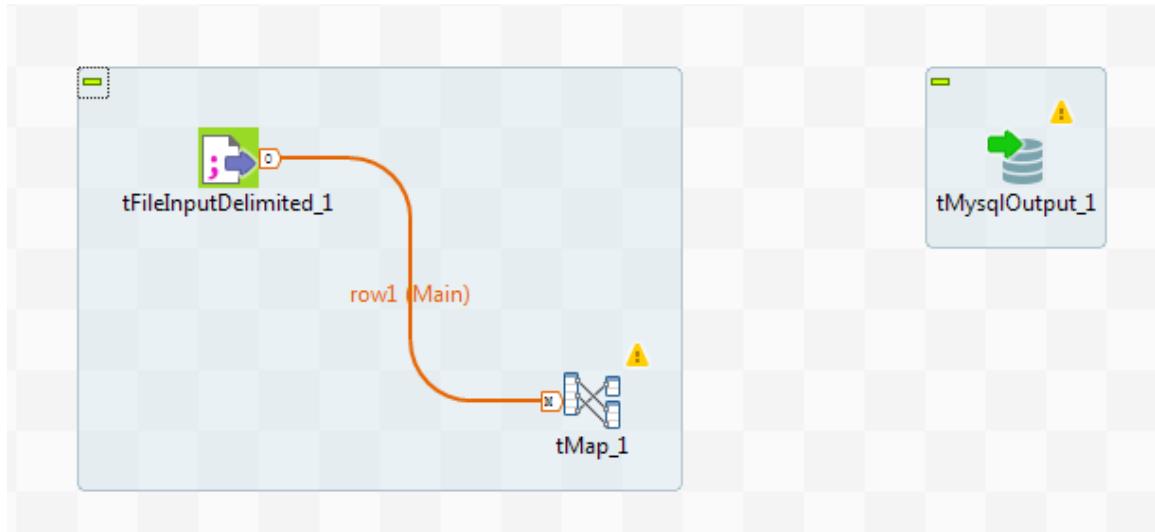
1. Add the new component, **tMap** in this example, onto the design workspace by either dropping it from the **Palette** or clicking in the design workspace and typing the component name.



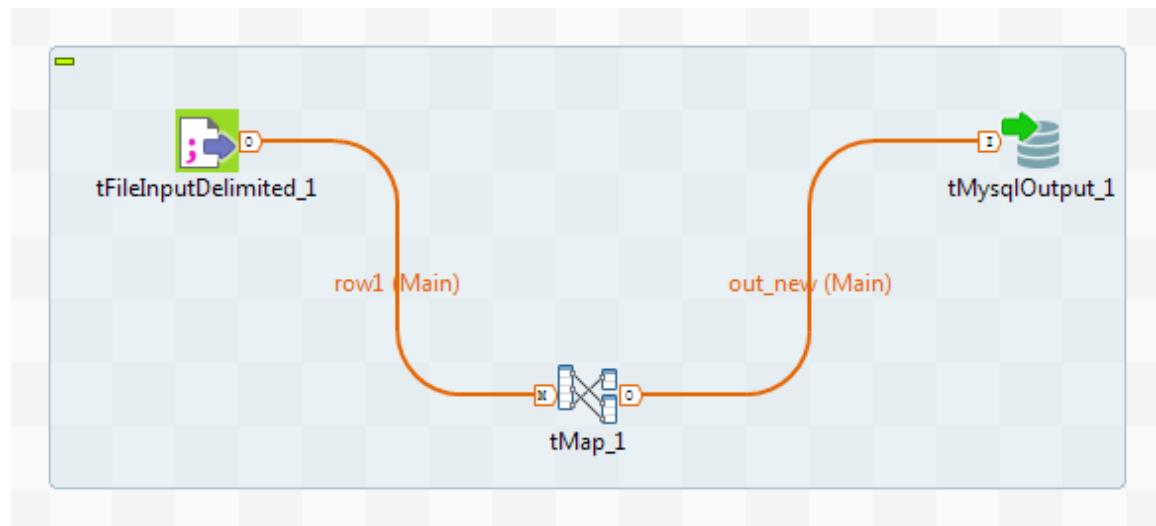
2. Select the connection and move your mouse pointer towards the end of the connection until the mouse pointer becomes a + symbol.



3. Drag the connection from the **tMySqlOutput** component and drop it onto the **tMap** component.



4. Connect the **tMap** component to the **tMySqlOutput** using a **Row > Main** connection.



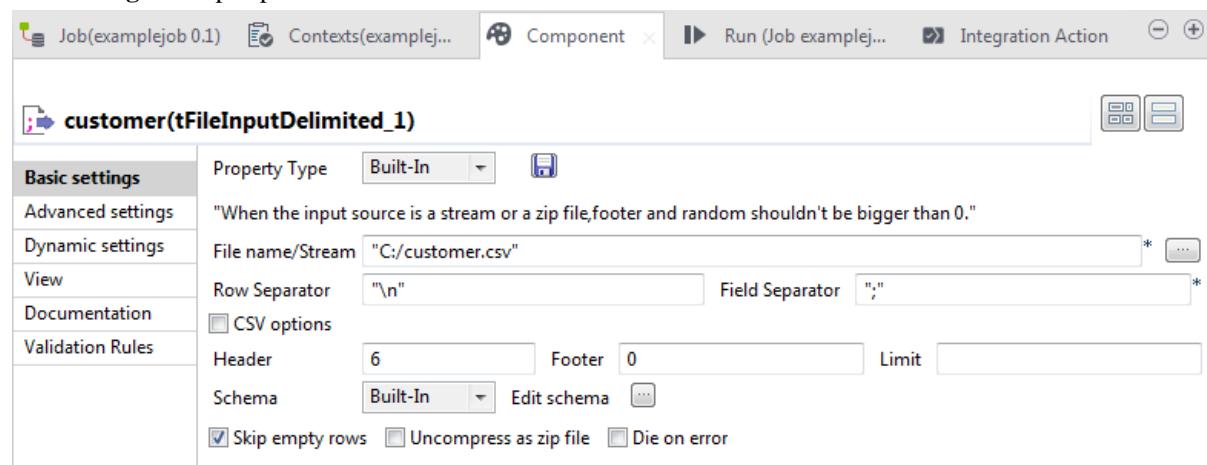
4.3.2. How to define component properties

The properties information for each component forming a Job or a subjob allows to set the actual technical implementation of the active Job.

Each component is defined by basic and advanced properties shown respectively on the **Basic Settings** tab and the **Advanced Settings** tab of the **Component** view of the selected component in the design workspace. The **Component** view gathers also other collateral information related to the component in use, including **View** and **Documentation** tabs.

4.3.2.1. Basic Settings tab

The **Basic Settings** tab is part of the **Component** view, which is located on the lower part of the designing editor of the **Integration** perspective of *Talend Studio*.



Each component has specific basic settings according to its function requirements within the Job.



Some components require code to be input or functions to be set. Make sure you use Java code in properties.

For **File** and **Database** components, you can centralize properties in metadata files located in the **Metadata** directory of the **Repository** tree view. This means that on the **Basic Settings** tab you can set properties on the

spot, using the **Built-In Property Type** or use the properties you stored in the **Metadata Manager** using the **Repository Property Type**. The latter option helps you save time.

Select **Repository** as **Property Type** and choose the metadata file holding the relevant information. Related topic: [Managing Metadata](#).

Alternatively, you can drop the **Metadata** item from the **Repository** tree view directly to the component already dropped on the design workspace, for its properties to be filled in automatically.

If you selected the **Built-in** mode and set manually the properties of a component, you can also save those properties as metadata in the **Repository**. To do so:

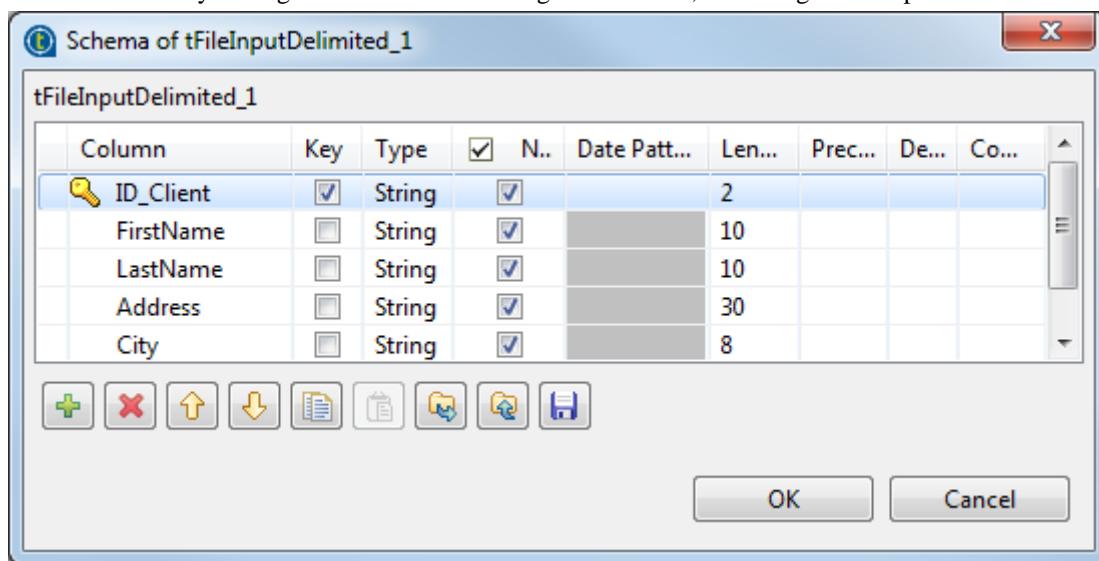
1. Click the floppy disk icon. The metadata creation wizard corresponding to the component opens.
2. Follow the steps in the wizard. For more information about the creation of metadata items, see [Managing Metadata](#).
3. The metadata displays under the **Metadata** node of the **Repository**.

For all components that handle a data flow (most components), you can define a **Talend** schema in order to describe and possibly select the data to be processed. Like the **Properties** data, this schema is either **Built-in** or stored remotely in the **Repository** in a metadata file that you created. A detailed description of the Schema setting is provided in the next sections.

How to set a built-in schema

A schema created as **Built-in** is meant for a single use in a Job, hence cannot be reused in another Job.

Select **Built-in** in the **Property Type** list of the **Basic settings** view, and click the **Edit Schema** button to create your built-in schema by adding columns and describing their content, according to the input file definition.



Make sure the data type in the **Type** column is correctly defined.

For more information regarding Java data types, including date pattern, see [Java API Specification](#).

Below are the commonly used **Talend** data types:

- Object: a generic **Talend** data type that allows processing data without regard to its content, for example, a data file not otherwise supported can be processed with a **tFileInputRaw** component by specifying that it has a data type of Object.

- List: a space-separated list of primitive type elements in an XML Schema definition, defined using the xsd:list element.
- Document: a data type that allows processing an entire XML document without regarding to its content.

In all output properties, you also have to define the schema of the output. To retrieve the schema defined in the input schema, click the **Sync columns** tab in the **Basic settings** view.



*When creating a database table, you are recommended to specify the **Length** field for all columns of type String, Integer or Long and specify the **Precision** field for all columns of type Double, Float or BigDecimal in the schema of the component used. Otherwise, unexpected errors may occur.*

How to set a repository schema

If you often use certain database connections or specific files when creating your data integration Jobs, you can avoid defining the same properties over and over again by creating metadata files and storing them in the **Metadata** node in the **Repository** tree view of the **Integration** perspective.

To recall a metadata file into your current Job, select **Repository** in the **Schema** list and then select the relevant metadata file. Or, drop the metadata item from the **Repository** tree view directly to the component already dropped on the design workspace. Then click **Edit Schema** to check that the data is appropriate.

The screenshot shows the configuration of a DBInput component named 'DBInput(tMysqlInput_1)'. The 'Basic settings' tab is active. The 'Property Type' dropdown is set to 'Repository' with 'DB (MYSQL):mysql' selected. The 'DB Version' dropdown is set to 'Mysql 5'. Under 'Use an existing connection', the host is 'localhost', port is '3306', database is 'employee', username is 'root', and password is masked. The 'Schema' dropdown is set to 'Repository' with 'DB (MYSQL):mysql - customer' selected. The 'Table Name' is 'customer', 'Query Type' is 'Built-In', and the 'Query' text area contains the following SQL:

```

"SELECT
`customer`.`id`,
`customer`.`CustomerName`,
`customer`.`CustomerAddress`,
`customer`.`idState`,
`customer`.`RegisterTime`,
`customer`.`Sum1`,
`customer`.`Sum2`,
`customer`.`Sum3`,
`customer`.`Sum4`,
`customer`.`Sum5`,
`customer`.`Sum6`,
`customer`.`Sum7`
FROM `customer`"

```

You can edit a repository schema used in a Job from the **Basic settings** view. However, note that the schema hence becomes **Built-in** in the current Job.

You can also use a repository schema partially. For more information, see [How to use a repository schema partially](#).



You cannot change the schema stored in the repository from this window. To edit the schema stored remotely, right-click it under the **Metadata** node and select the corresponding edit option (**Edit connection** or **Edit file**) from the contextual menu.

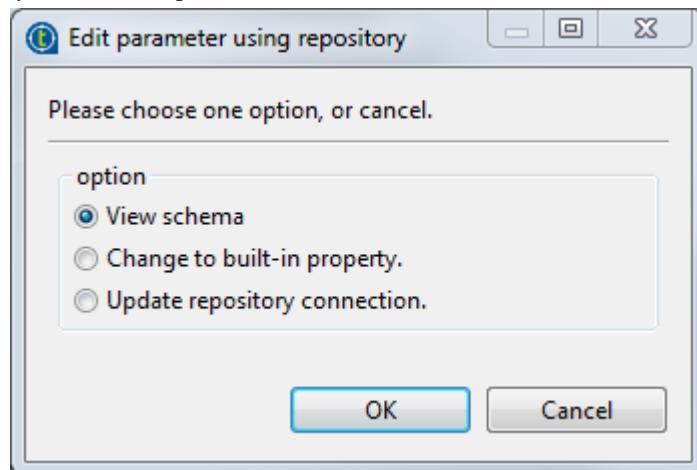
Related topics: [Managing Metadata](#).

How to use a repository schema partially

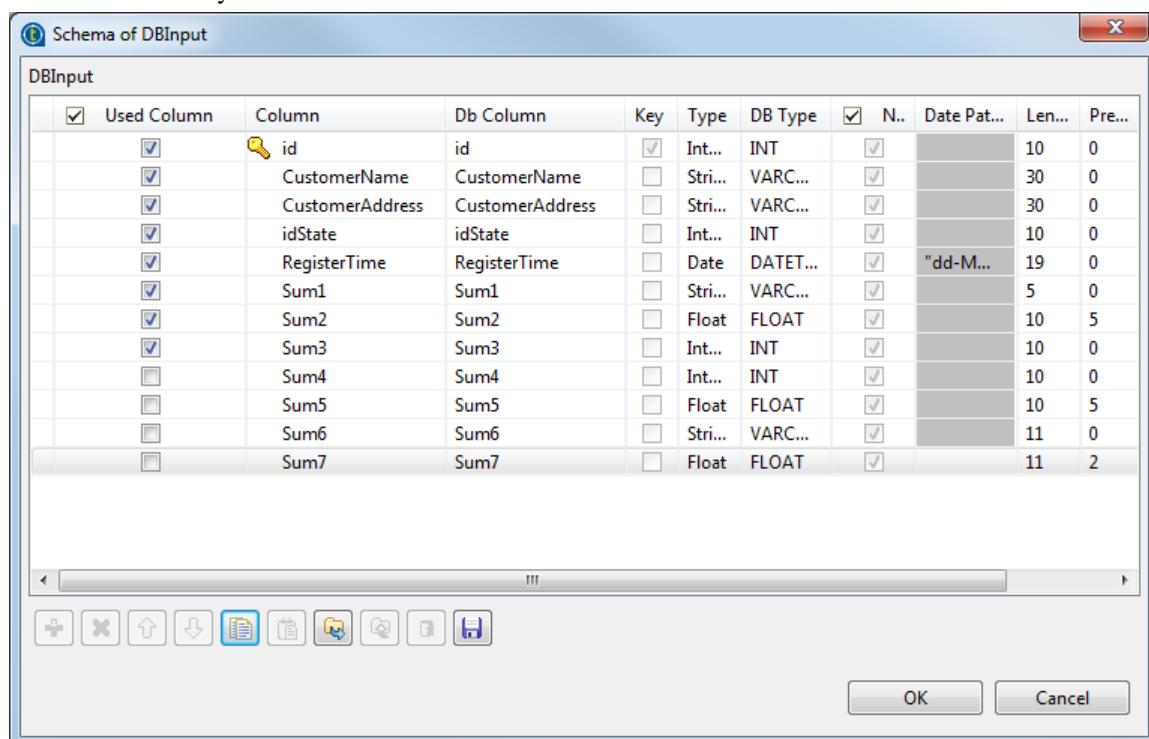
When using a repository schema, if you do not want to use all the predefined columns, you can select particular columns without changing the schema into a built-in one:

The following describes how to use a repository schema partially for a database input component. The procedure may vary slightly according to the component you are using.

1. Click the [...] button next to **Edit schema** on the **Basic settings** tab. The **[Edit parameter using repository]** dialog box appears. By default, the option **View schema** is selected.



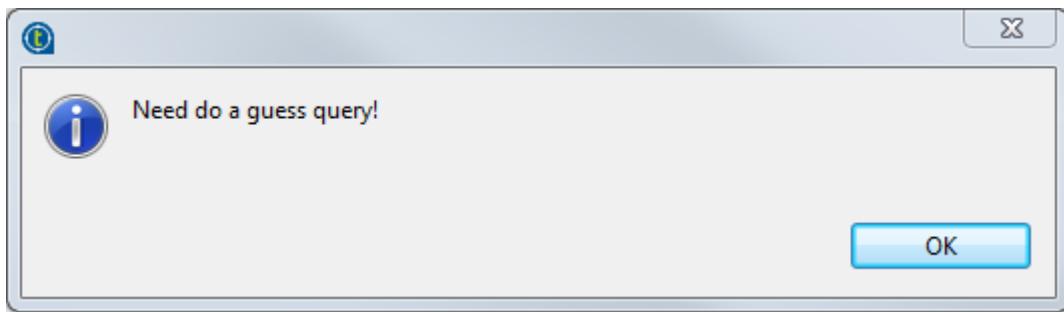
2. Click **OK**. The **[Schema]** dialog box pops up, which displays all columns in the schema. The **Used Column** check box before each column name indicates whether the column is used.
3. Select the columns you want to use.



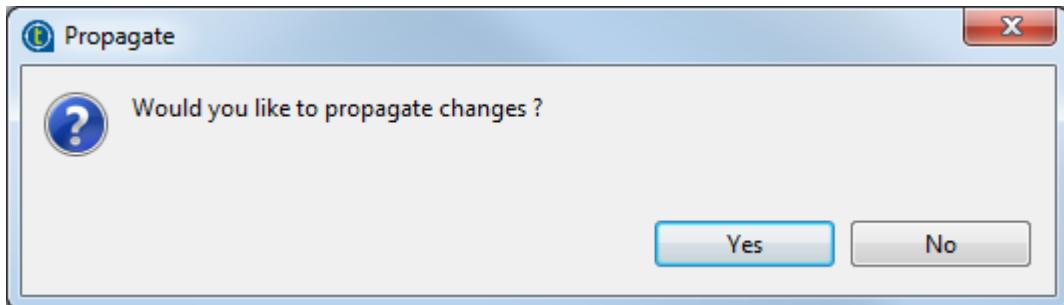
4. Click **OK**. A message box appears, which prompts you to do a guess query.



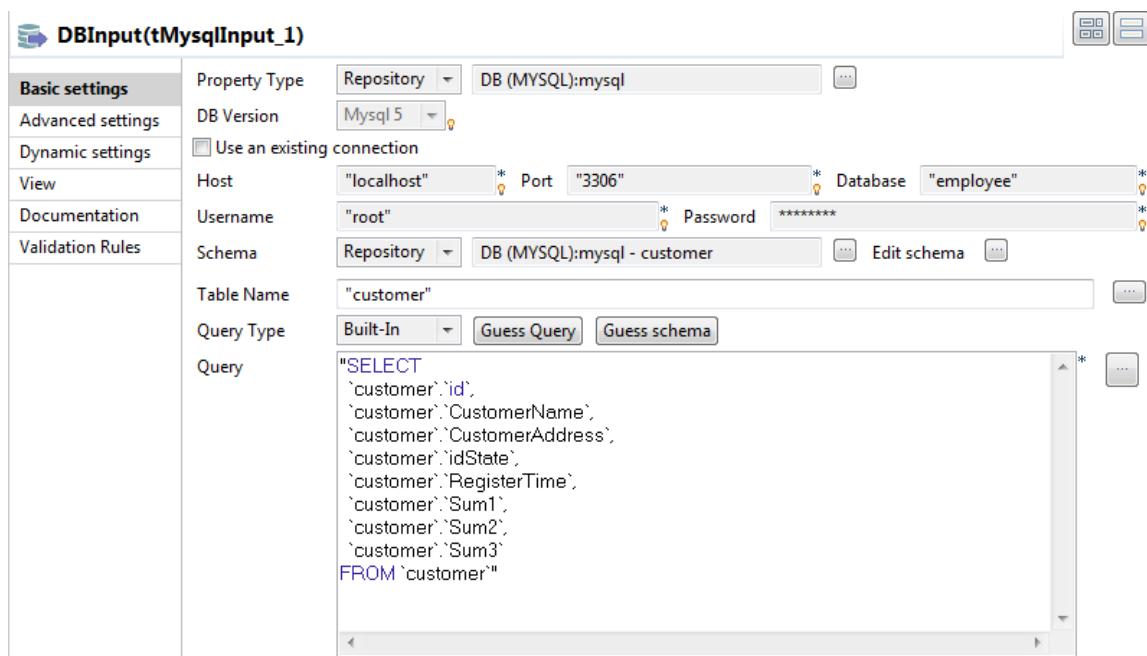
The guess query operation is needed only for the database metadata.



5. Click **OK** to close the message box. The [**Propagate**] dialog box appears. Click **Yes** to propagate the changes and close the dialog box.



6. On the **Basic settings** tab, click **Guess Query**. The selected column names are displayed in the **Query** area as expected.



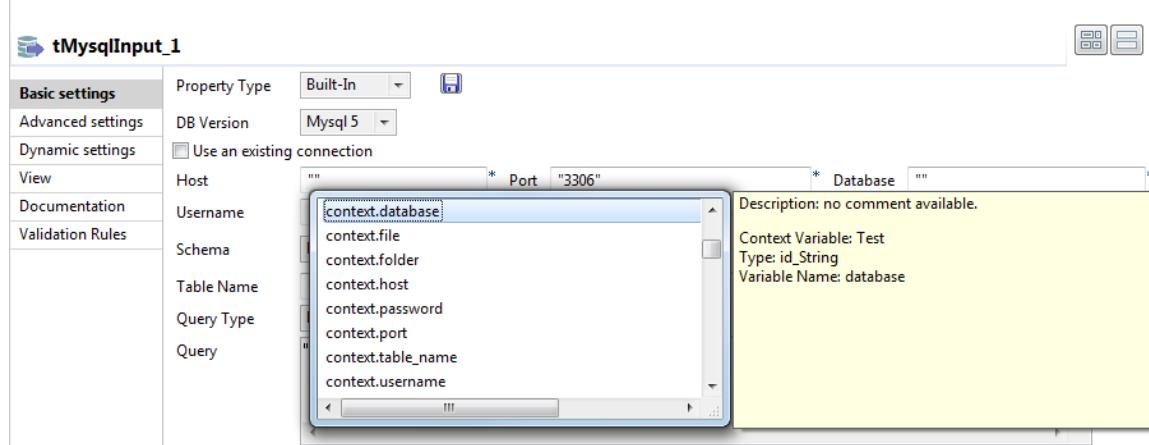
For more information about how to set a repository schema, see [How to set a repository schema](#).

How to set a field dynamically (Ctrl+Space bar)

On any field of your Job/component **Properties** view, you can use the **Ctrl+Space** bar to access the global and context variable list and set the relevant field value dynamically.

1. Place the cursor on any field of the **Component** view.

2. Press **Ctrl+Space bar** to access the proposal list.
3. Select on the list the relevant parameters you need. Appended to the variable list, a information panel provides details about the selected parameter.

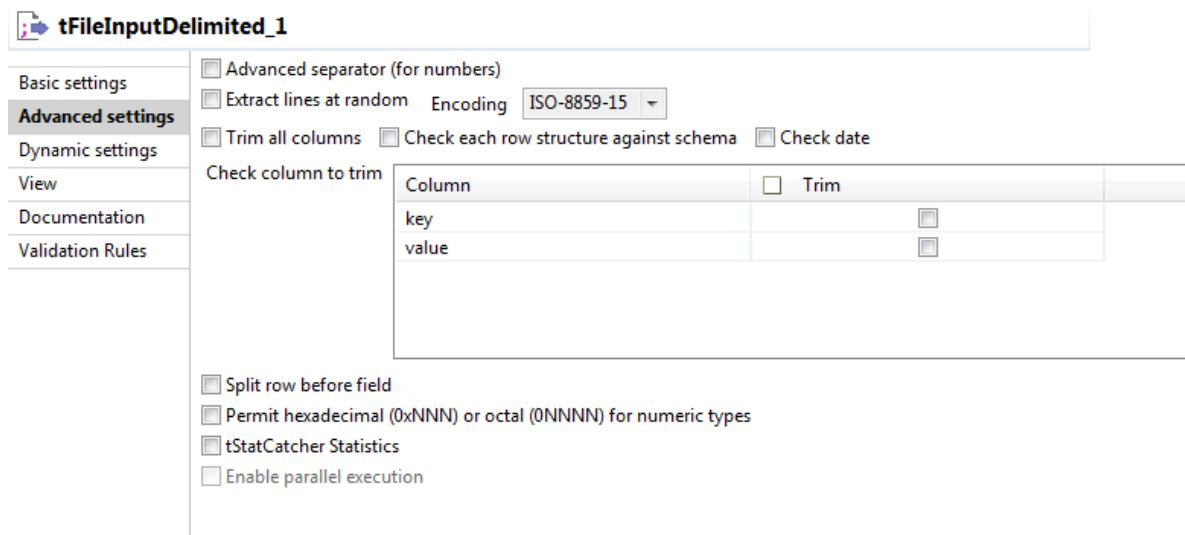


This can be any parameter including: error messages, number of lines processed, or else... The list varies according to the component in selection or the context you are working in.

Related topic: [Using contexts and variables](#).

4.3.2.2. Advanced settings tab

Some components, especially **File** and **Databases** components, provides numerous advanced use possibilities.



The content of the **Advanced settings** tab changes according to the selected component.

Generally you will find on this tab the parameters that are not required for a basic or usual use of the component but may be required for a use out of the standard scope.

How to measure data flows

You can also find in the **Advanced settings** view the option **tStatCatcher Statistics** that allows you, if selected, to display logs and statistics about the current Job without using dedicated components. For more information regarding the stats & log features, see [How to automate the use of statistics & logs](#).

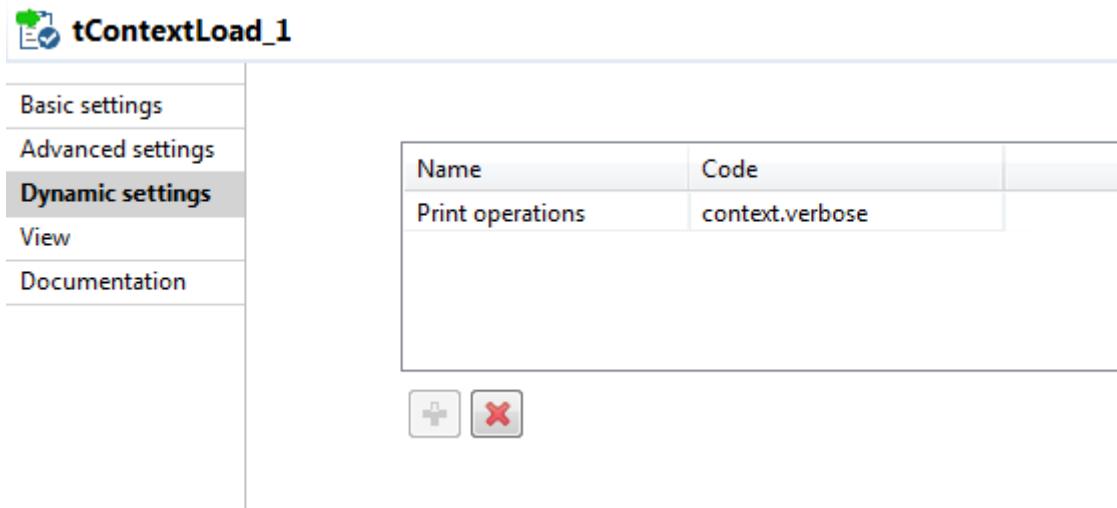
4.3.2.3. Dynamic settings tab

The **Basic settings** and **Advanced settings** tabs of all components display various check boxes and drop-down lists for component parameters. Usually, available values for these types of parameters can only be edited when designing your Job.

The **Dynamic settings** tab, on the **Component** view, allows you to customize these parameters into code or variable.

This feature allows you, for example, to define these parameters as variables and thus let them become context-dependent, whereas they are not meant to be by default.

Another benefit of this feature is that you can now change the context setting at execution time. This makes full sense when you intend to export your Job in order to deploy it onto a Job execution server for example.



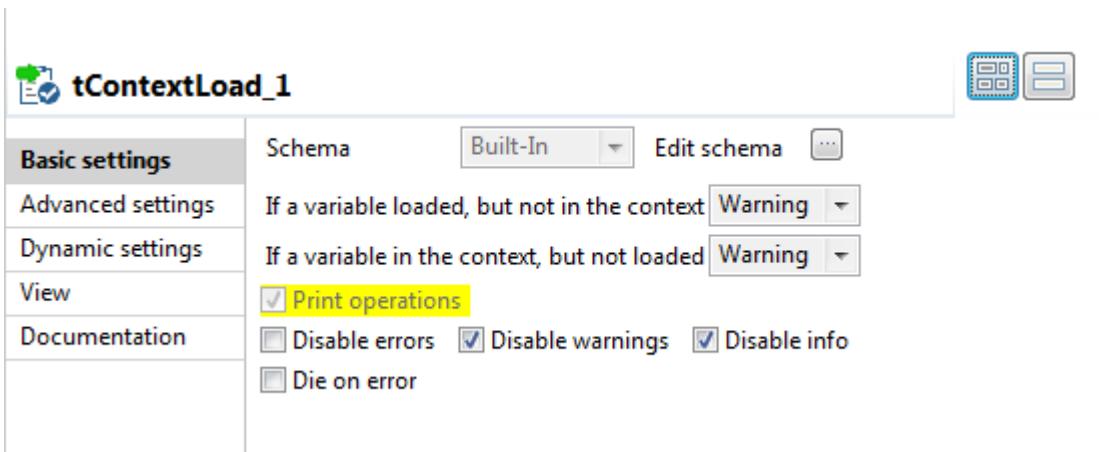
To customize these types of parameters, as context variables for example, follow the following steps:

1. Select the relevant component basic settings or advanced settings view that contains the parameter you want to define as a variable.
2. Click the **Dynamic settings** tab.
3. Click the **plus** button to display a new parameter line in the table.
4. Click the **Name** of the parameter displaying to show the list of available parameters. For example: *Print operations*
5. Then click in the facing **Code** column cell and set the code to be used. For example: *context.verbose* if you create the corresponding context variable, called *verbose*.



As code, you can input a context variable or a piece of Java code.

The corresponding lists or check boxes thus become unavailable and are highlighted in yellow in the **Basic settings** or **Advanced settings** tab.



If you want to set a parameter as context variable, make sure you create the corresponding variable in the **Contexts** view.

For more information regarding the context variable definition, see [How to define context variables in the Contexts view](#).

For use cases showing how to define a dynamic parameter, see the scenario of **tMysqlInput** about reading data from MySQL databases through context-based dynamic connections and the scenario of **tContextLoad** at <https://help.talend.com>.

4.3.2.4. View tab

The **View** tab of the **Component** view allows you to change the default display format of components on the design workspace.

Field	Description
Label format	Free text label showing on the design workspace. Variables can be set to retrieve and display values from other fields. The field tooltip usually shows the corresponding variable where the field value is stored.
Hint format	Hidden tooltip, showing only when you mouse over the component.
Connection format	Indicates the type of connection accepted by the component.

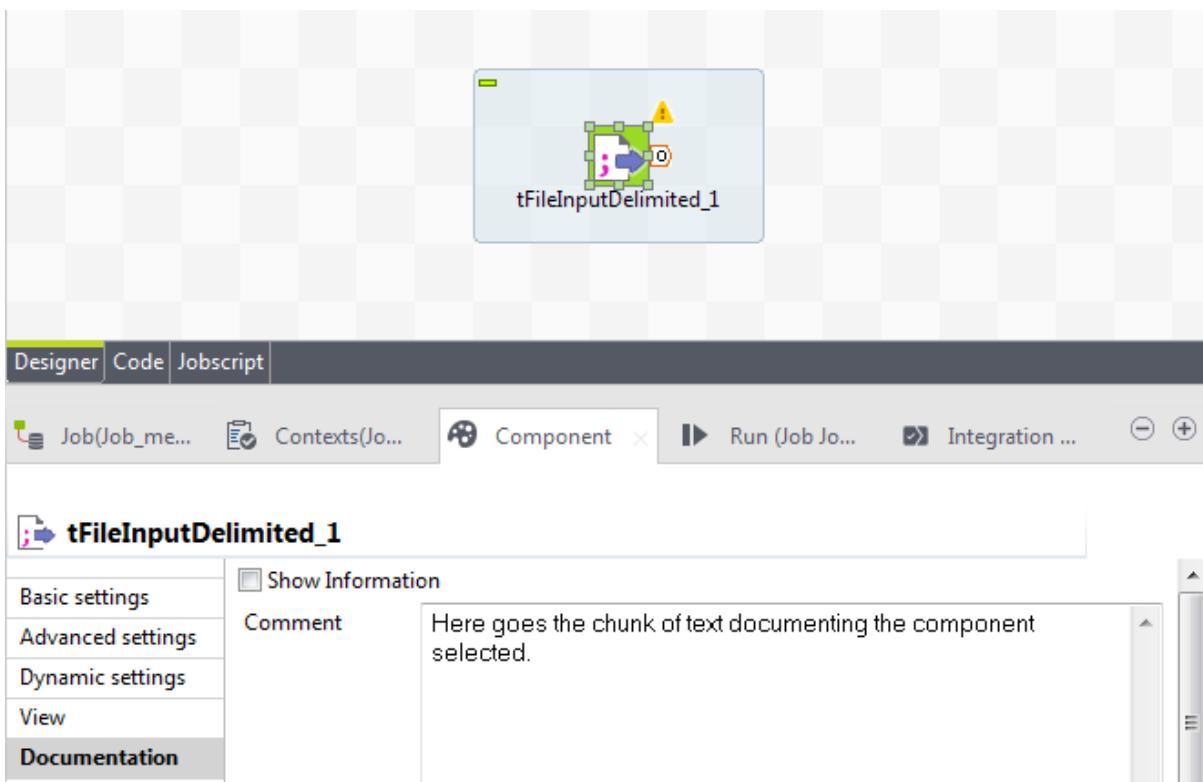
You can graphically highlight both **Label** and **Hint** text with HTML formatting tags:

- Bold: YourLabelOrHint
- Italic: <i> YourLabelOrHint </i>
- Return carriage: YourLabelOrHint
 ContdOnNextLine
- Color: YourLabelOrHint

To change your preferences of this **View** panel, click **Window>Preferences>Talend>Designer**.

4.3.2.5. Documentation tab

Feel free to add any useful comment or chunk of text or documentation to your component.



In the **Documentation** tab, you can add your text in the **Comment** field. Then, select the **Show Information** check box and an information icon display next to the corresponding component in the design workspace.

You can show the Documentation in your hint tooltip using the associated variable `_COMMENT_`, so that when you place your mouse on this icon, the text written in the **Comment** field displays in a tooltip box.

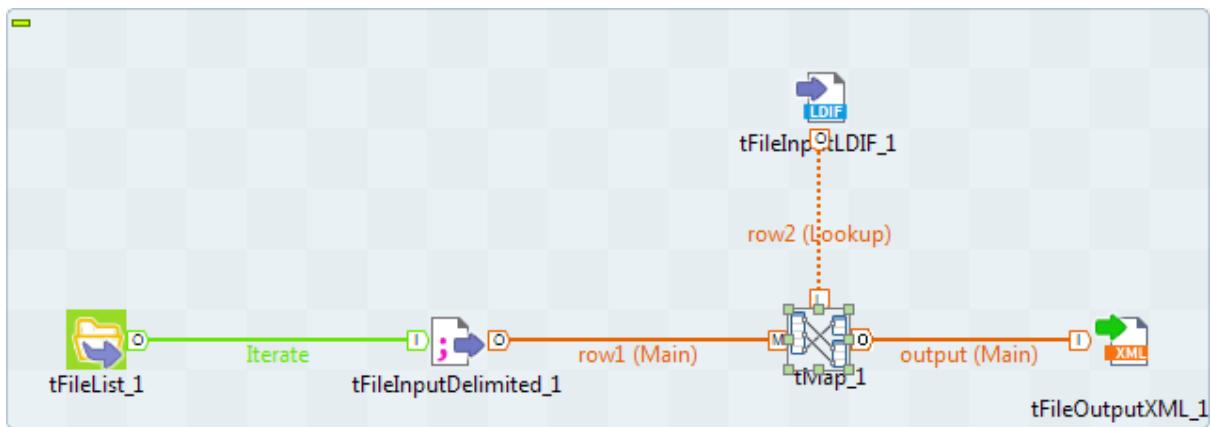
For advanced use of Documentations, you can use the **Documentation** view in order to store and reuse any type of documentation.

4.3.3. How to define the start component

The **Start** component is the trigger of a Job. There can be several Start components per Job design if there are several flows running in parallel. But for one flow and its *connected* subflows, only one component can be the Start component.

Drop a component to the design workspace, all possible start components take a distinctive bright green background color. Notice that most of the components, can be **Start** components.

Only components which do not make sense to trigger a flow, will not be proposed as Start components, such as the **tMap** component for example.



To distinguish which component is to be the **Start** component of your Job, identify the main flow and the secondary flows of your Job.

- The main flow should be the one connecting a component to the next component using a Row type link. The Start component is then automatically set on the first component of the main flow (icon with green background).
- The secondary flows are also connected using a Row-type link which is then called Lookup row on the design workspace to distinguish it from the main flow. This Lookup flow is used to enrich the main flow with more data.

Be aware that you can change the Start component hence the main flow by changing a main Row into a Lookup Row, simply through a right-click the row to be changed.

Related topics:

- [Connecting the components together](#)
- [Activating/Deactivating a component or a subjob](#)

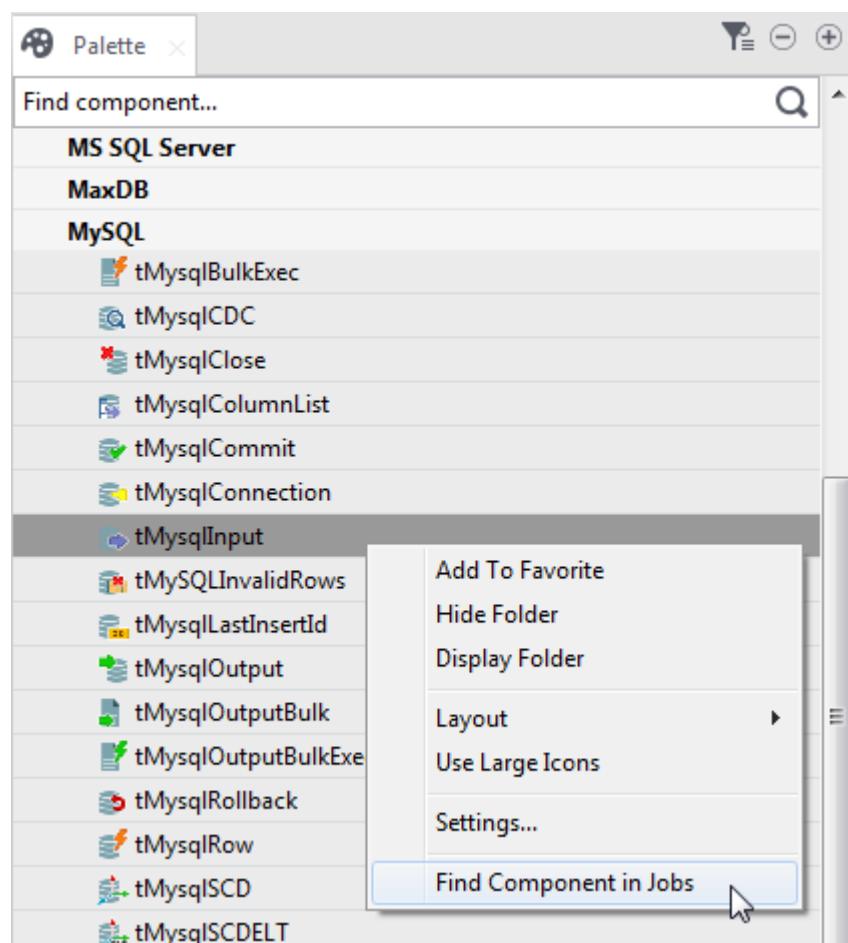
4.3.4. How to find Jobs containing a specific component



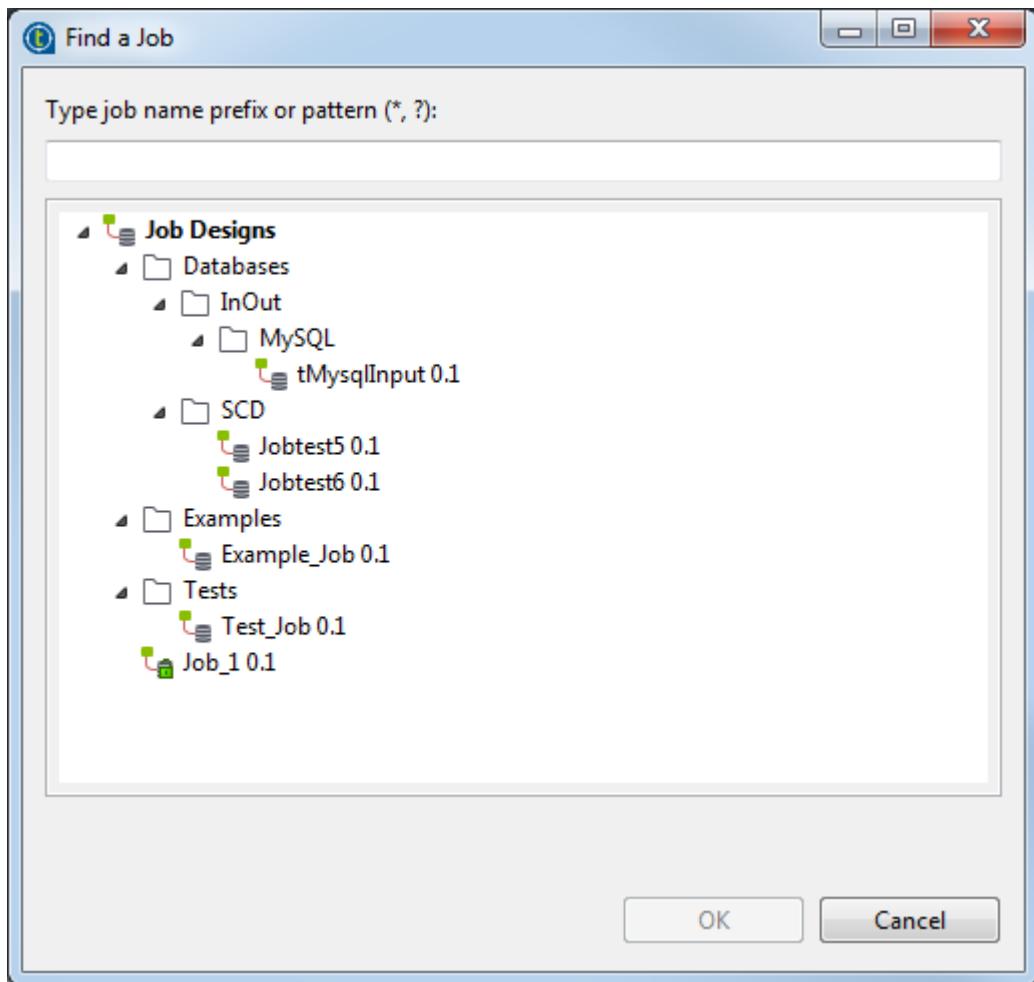
You should open one Job at least in the Studio to display the **Palette** to the right of the design workspace and thus start the search.

From the **Palette**, you can search for all the Jobs that use the selected component. To do so:

1. In the **Palette**, right-click the component you want to look for and select **Find Component in Jobs**.



A progress indicator displays to show the percentage of the search operation that has been completed then the **[Find a Job]** dialog box displays listing all the Jobs that use the selected component.



- From the list of Jobs, click the desired Job and then click **OK** to open it on the design workspace.

4.3.5. How to set default values in the schema of a component

You can set default values in the schema of certain components to replace null values retrieved from the data source.

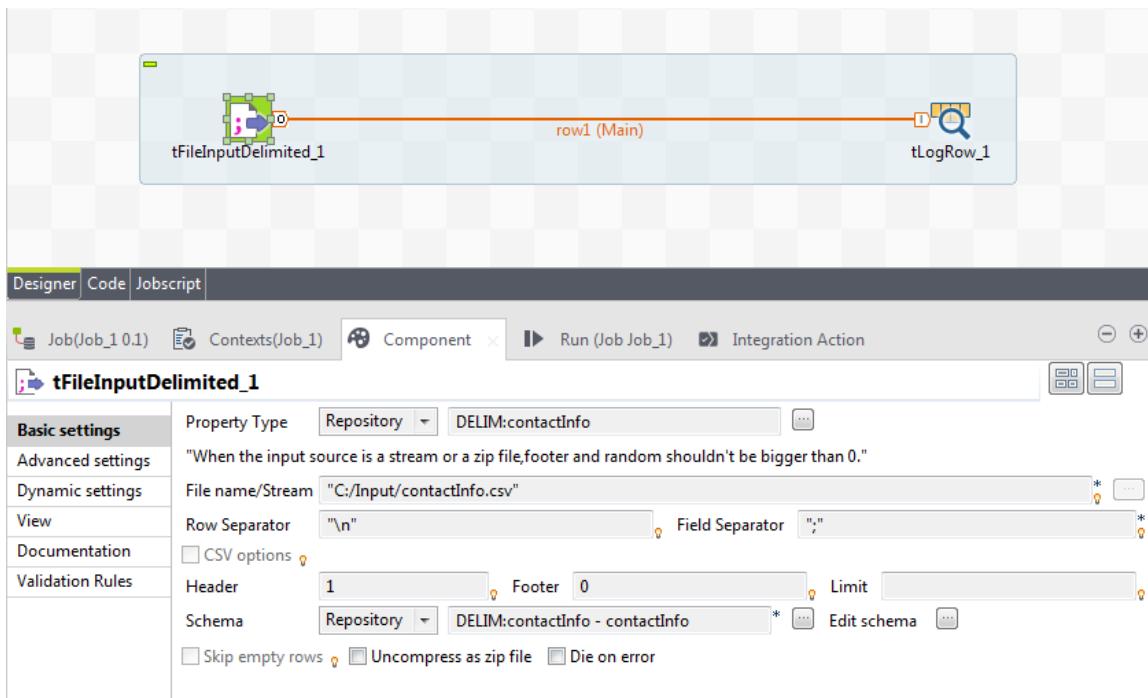
 At present, only **tFileInputDelimited**, **tFileInputExcel**, and **tFixedFlowInput** support default values in the schema.

In the following example, the *company* and *city* fields of some records of the source CSV file are left blank, as shown below. The input component reads data from the source file and completes the missing information using the default values set in the schema, *Talend* and *Paris* respectively.

```
id;firstName;lastName;company;city;phone
1;Michael;Jackson;IBM;Roma;2323
2;Elisa;Black;Microsoft;London;4499
3;Michael;Dujardin;;;8872
4;Marie;Dolvina;;;6655
5;Jean;Perfide;;;3344
6;Emilie;Taldor;Oracle;Madrid;2266
7;Anne-Laure;Paldufier;Apple;;4422
```

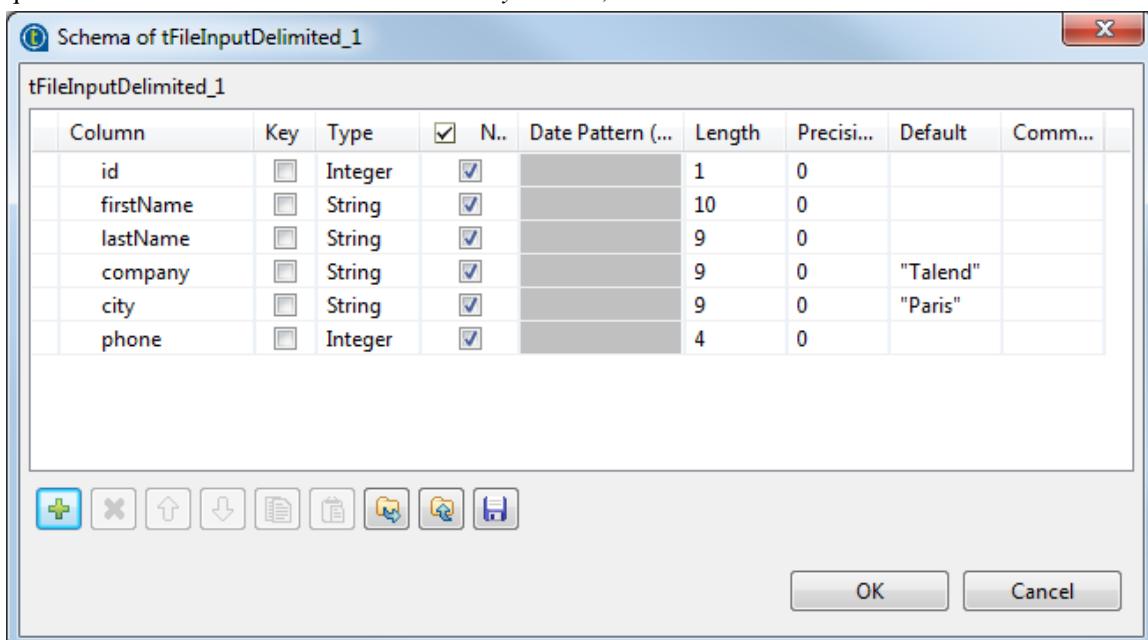
To set default values:

1. Double-click the input component **tFileInputDelimited** to show its **Basic settings** view.

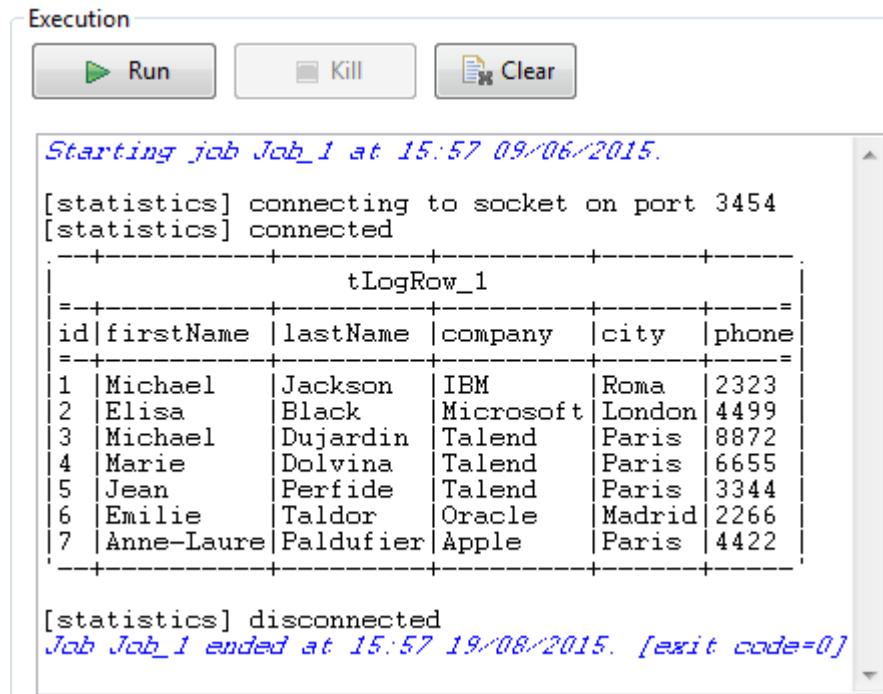


In this example, the metadata for the input component is stored in the Repository. For information about metadata creation in the Repository, see [Managing Metadata](#).

2. Click the [...] button next to **Edit schema**, and select the **Change to built-in property** option from the pop-up dialog box to open the schema editor.
3. Enter *Talend* between quotation marks in the **Default** field for the *company* column, enter *Paris* between quotation marks in the **Default** field for the *city* column, and click **OK** to close the schema editor.



4. Configure the output component **tLogRow** to display the execution result the way you want, and then run the Job.



In the output data flow, the missing information is completed according to the set default values.

4.4. Using connections

In *Talend Studio*, a Job or a subjob is composed of a group of components logically linked to one another via connections. You need to use the connections to define how the components in use are coordinated. This section will describe the types of connections and their related settings.

4.4.1. Connection types

There are various types of connections which define either the data to be processed, the data output, or the Job logical sequence.

Right-click a component on the design workspace to display a contextual menu that lists all available connections for the selected component.

The sections below describe all available connection types.

4.4.1.1. Row connection

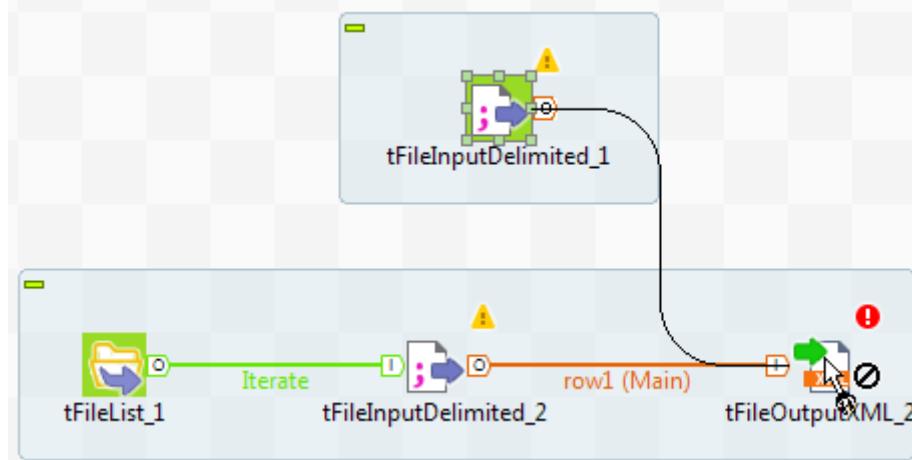
A **Row** connection handles the actual data. The **Row** connections can be **Main**, **Lookup**, **Reject**, **Output**, **Uniques/Duplicates**, or **Combine** according to the nature of the flow processed.

Main

This type of row connection is the most commonly used connection. It passes on data flows from one component to the other, iterating on each row and reading input data according to the component properties setting (schema).

Data transferred through main rows are characterized by a schema definition which describes the data structure in the input file.

 You cannot connect two Input components together using a **Row > Main** connection. Only *one* incoming **Row** connection is possible per component. You will not be able to link twice the same target component using a main **Row** connection. The second **Row** connection will be called **Lookup**.



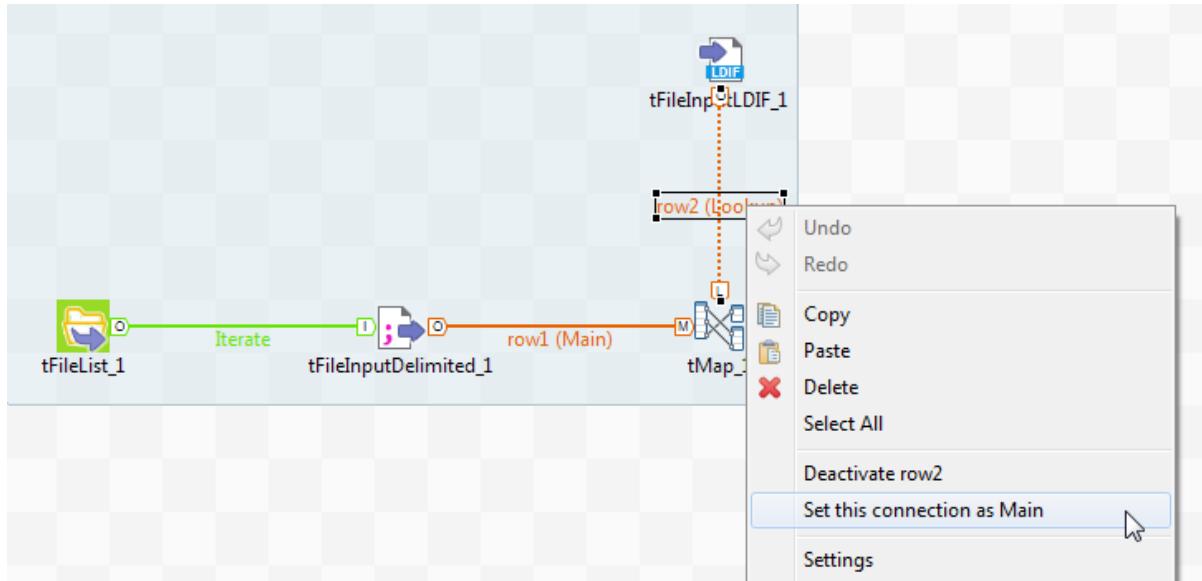
To connect two components using a Main connection, right-click the input component and select **Row > Main** on the connection list.

Alternatively, you can click the component to highlight it, then right-click it or click the **O** icon that appears on side of it and drag the cursor towards the destination component. This will automatically create a **Row > Main** type of connection.

For information on using multiple **Row** connections, see [Multiple Input/Output](#).

Lookup

This row connection connects a sub-flow component to a main flow component (which should be allowed to receive more than one incoming flow). This connection is used only in the case of multiple input flows.



A **Lookup** row can be changed into a main row at any time (and reversely, a main row can be changed to a lookup row). To do so, right-click the row to be changed, and on the pop-up menu, click **Set this connection as Main**.

Related topic: [Multiple Input/Output](#).

Filter

This row connection connects specifically a **tFilterRow** component to an output component. This row connection gathers the data matching the filtering criteria. This particular component offers also a **Reject** connection to fetch the non-matching data flow.

Rejects

This row connection connects a processing component to an output component. This row connection gathers the data that does NOT match the filter or are not valid for the expected output. This connection allows you to track the data that could not be processed for any reason (wrong type, undefined null value, etc.). On some components, this connection is enabled when the **Die on error** option is deactivated.

ErrorReject

This row connection connects a **tMap** component to an output component. This connection is enabled when you clear the **Die on error** check box in the **tMap editor** and it gathers data that could not be processed (wrong type, undefined null value, unparseable dates, etc.).

Related topic: [Handling errors](#).

Output

This row connection connects a **tMap** component to one or several output components. As the Job output can be multiple, you get prompted to give a name for each output row created.



The system also remembers deleted output connection names (and properties if they were defined). This way, you do not have to fill in again property data in case you want to reuse them.

Related topic: [Multiple Input/Output](#).

Uniques/Duplicates

These row connection connect a **tUniqRow** to output components.

The **Uniques** connection gathers the rows that are found first in the incoming flow. This flow of unique data is directed to the relevant output component or else to another processing subjob.

The **Duplicates** connection gathers the possible duplicates of the first encountered rows. This reject flow is directed to the relevant output component, for analysis for example.

Multiple Input/Output

Some components help handle data through multiple inputs and/or multiple outputs. These are often processing-type components such as the **tMap**.

If this requires a join or some transformation in one flow, you want to use the **tMap** component, which is dedicated to this use.

For further information regarding data mapping, see [Mapping data flows](#).

Combine

This type of row connection connects one CombinedSQL component to another.

When right-clicking the CombinedSQL component to be connected to the next one, select **Row > Combine**.

4.4.1.2. Iterate connection

The **Iterate** connection can be used to loop on files contained in a directory, on rows contained in a file or on DB entries.

A component can be the target of only one **Iterate** connection. The **Iterate** connection is mainly to be connected to the start component of a flow (in a subjob).

Some components such as the **tFileDialog** component are meant to be connected through an iterate connection with the next component. For how to set an **Iterate** connection, see [Iterate connection settings](#).

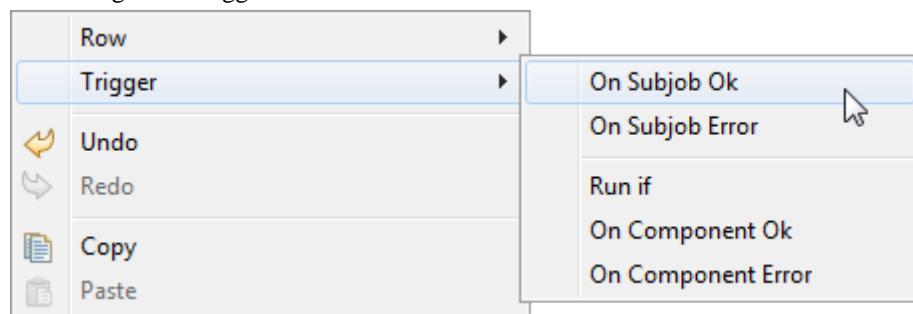


The name of the **Iterate** connection is read-only unlike other types of connections.

4.4.1.3. Trigger connections

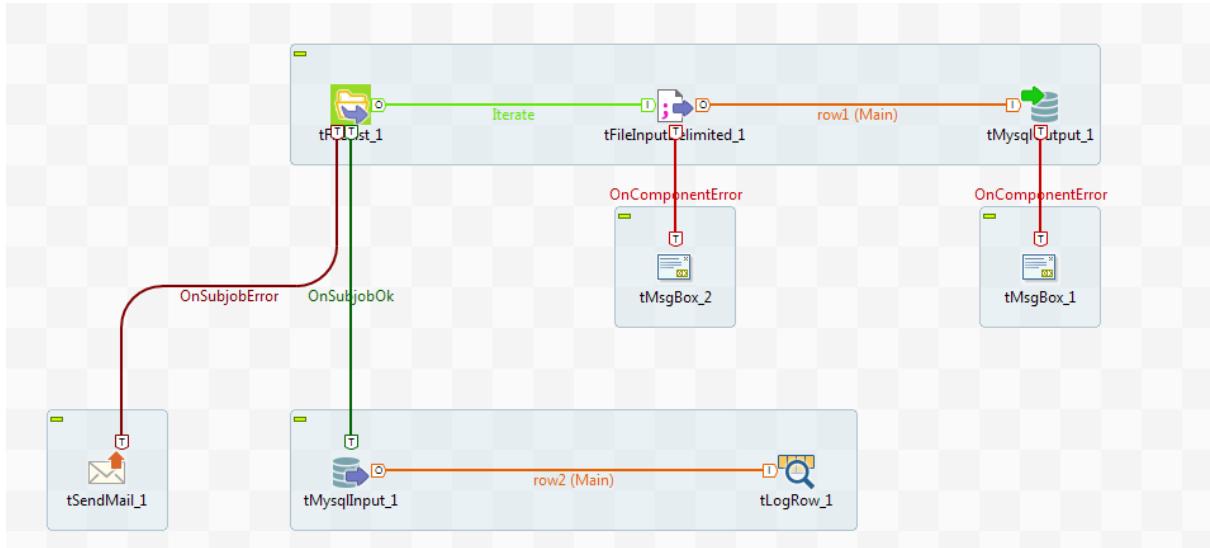
Trigger connections define the processing sequence, so no data is handled through these connections.

The connection in use will create a dependency between Jobs or subjobs which therefore will be triggered one after the other according to the trigger nature.



Trigger connections fall into two categories:

- subjob triggers: **On Subjob Ok**, **On Subjob Error** and **Run if**,
- component triggers: **On Component Ok**, **On Component Error** and **Run if**.



OnSubjobOK (previously **Then Run**): This connection is used to trigger the next subjob on the condition that the main subjob completed without error. This connection is to be used only from the start component of the Job.

These connections are used to orchestrate the subjobs forming the Job or to easily troubleshoot and handle unexpected errors.

OnSubjobError: This connection is used to trigger the next subjob in case the first (main) subjob do not complete correctly. This "on error" subjob helps flagging the bottleneck or handle the error if possible.

Related topic: [How to define the start component](#).

OnComponentOK and **OnComponentError** are component triggers. They can be used with any source component on the subjob.

OnComponentOK will only trigger the target component once the execution of the source component is complete without error. Its main use could be to trigger a notification subjob for example.

OnComponentError will trigger the sub-job or component as soon as an error is encountered in the primary Job.

Run if triggers a subjob or component in case the condition defined is met. For further information about **Run if**, see [Run if connection settings](#).

For how to set a trigger condition, see [Trigger connection settings](#).

4.4.1.4. Link connection

The **Link** connection can only be used with ELT components. These connections transfer table schema information to the ELT mapper component in order to be used in specific DB query statements.

The **Link** connection therefore does not handle actual data but only the metadata regarding the table to be operated on.

When right-clicking the ELT component to be connected, select **Link > New Output**.



Be aware that the name you provide to the connection must reflect the actual table name.

In fact, the connection name will be used in the SQL statement generated through the ETL Mapper, therefore the same name should never be used twice.

4.4.2. How to define connection settings

You can display the properties of a connection by selecting it and clicking the **Component** view tab, or by right-clicking the connection and selecting **Settings** from the contextual menu. This section summarizes connection property settings.

4.4.2.1. Row connection settings

The **Basic settings** vertical tab of the **Component** view of the connection displays the schema of the data flow handled by the connection. You can change the schema by clicking the **Edit schema** button. For more information, see [How to set a built-in schema](#).

Column	Key	Type	N..	Date Patter...	Length	Preci...	Def...	Com...
id	<input checked="" type="checkbox"/>	Integer	<input checked="" type="checkbox"/>		2			
CustomerName	<input type="checkbox"/>	String	<input checked="" type="checkbox"/>		255			
CustomerAddress	<input type="checkbox"/>	String	<input checked="" type="checkbox"/>		255			
idState	<input type="checkbox"/>	Integer	<input checked="" type="checkbox"/>		2			
id2	<input type="checkbox"/>	Integer	<input checked="" type="checkbox"/>		2			
RegTime	<input type="checkbox"/>	String	<input checked="" type="checkbox"/>		50			
RegisterTime	<input type="checkbox"/>	String	<input checked="" type="checkbox"/>		50			

The **Advanced settings** vertical tab lets you monitor the data flow over the connection in a Job without using a separate **tFlowMeter** component. The measured information will be interpreted and displayed in a monitoring tool such *Talend Activity Monitoring Console* (available with **Talend** subscription-based products).

Use input connection name as label

Mode: Absolute *

Thresholds

Label	Low end	Top end	Color

Monitor this connection

To monitor the data over the connection, perform the following settings in the **Advanced settings** vertical tab:

1. Select the **Monitor this connection** check box.
2. From the **Mode** list, select **Absolute** to log the actual number of rows passes over the connection, or **Relative** to log the ratio (%) of the number of rows passed over this connection against a reference connection. If you select **Relative**, you need to select a reference connection from the **Connections List** list.

- Click the plus button to add a line in the **Thresholds** table and define a range of the number of rows to be logged.

For more information about flow metrics, see the documentation of **tFlowMeterCatcher** at <https://help.talend.com>.

4.4.2.2. Iterate connection settings

When you configure an Iterate connection, you are actually enabling parallel iterations. For further information, see [How to launch parallel iterations to read data](#).

4.4.2.3. Trigger connection settings

Run if connection settings

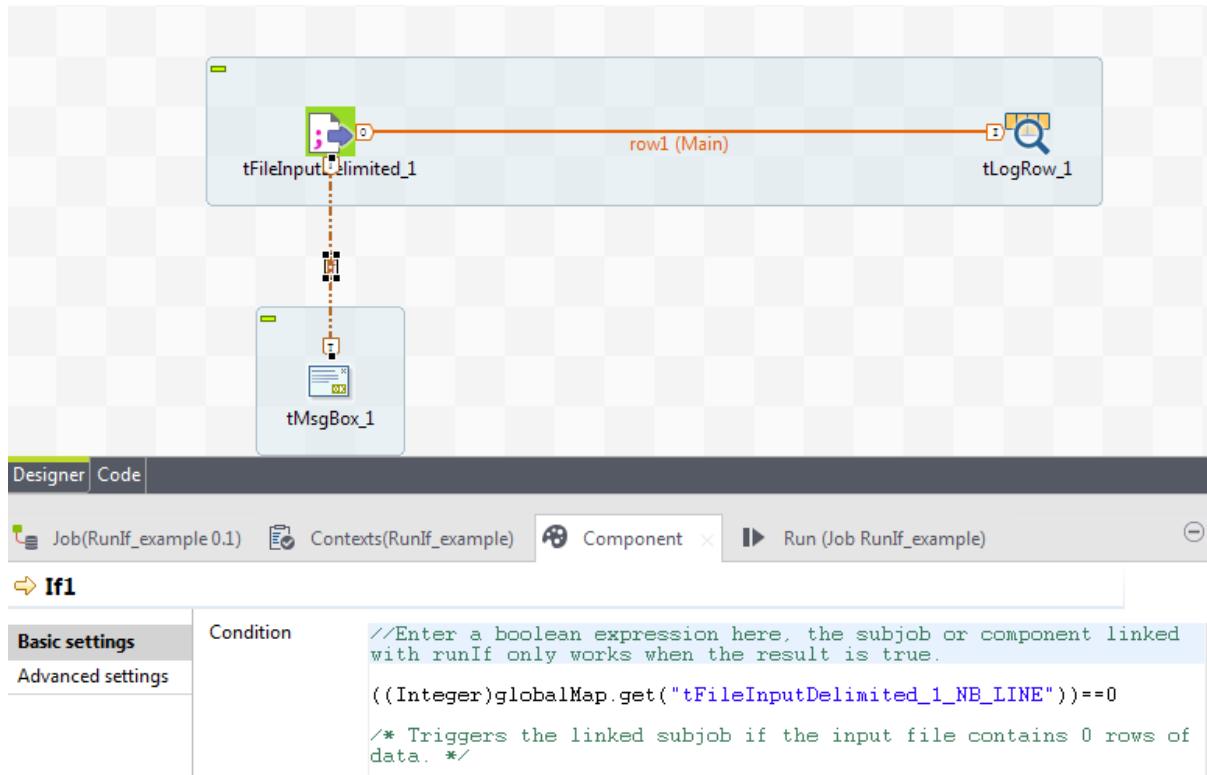
In the **Basic settings** view of a **Run if** connection, you can set the condition to the Subjob in Java.

You can use variables in your condition. Pressing **Ctrl+Space** allows you to access all global and context variables. For more information, see [How to use variables in a Job](#).



When adding a comment after the condition, be sure to enclose it between / and */ even if it is a single-line comment.*

In the following example, a message is triggered if the input file contains 0 rows of data.



- Create a Job and drop three components to the design workspace: a **tFileInputDelimited**, a **tLogRow**, and a **tMsgBox**.

2. Connect the components as follows:
 - Right-click the **tFileInputDelimited** component, select **Row > Main** from the contextual menu, and click the **tLogRow** component.
 - Right-click the **tFileInputDelimited** component, select **Trigger > Run if** from the contextual menu, and click the **tMsgBox** component.
 3. Configure the **tFileInputDelimited** component so that it reads a file that contains no data rows.
 4. Select the **Run if** connection between the **tFileInputDelimited** component and the **tMsgBox** component, and click the **Component** view. In the **Condition** field on the **Basic settings** tab, pressing **Ctrl+Space** to access the variable list, and select the *NB_LINE* variable of the **tFileInputDelimited** component. Edit the condition as follows:
- ```
((Integer)globalMap.get("tFileInputDelimited_1_NB_LINE"))==0
```
5. Go to the **Component** view of the **tMsgBox** component, and enter a message, "No data is read from the file" for example, in the **Message** field.
  6. Save and run the Job. You should see the message you defined in the **tMsgBox** component.

## 4.5. Using contexts and variables

Variables represent values which change throughout the execution of a program. A global variable is a system variable which can be accessed by any module or function. It retains its value after the function or program using it has completed execution. A context variable is a variable which is defined by the user for a particular context.

Depending on the circumstances the Job is being used in, you might want to manage it differently for various execution types, known as contexts (*Prod* and *Test* in the example given below). For instance, there might be various testing stages you want to perform and validate before a Job is ready to go live for production use.

A context is characterized by parameters. These parameters are mostly context-sensitive variables which will be added to the list of variables for reuse in the component-specific properties on the **Component** view through the **Ctrl+Space** keystrokes.

*Talend Studio* offers you the possibility to create multiple context data sets. Furthermore you can either create context data sets on a one-shot basis from the context tab of a Job, or you can centralize the context data sets in the **Contexts** node of the **Repository** tree view in order to reuse them in different Jobs.

You can define the values of your context variables when creating them, or load your context parameters dynamically, either explicitly using the **tContextLoad** component or implicitly using the Implicit Context Load feature, when your Jobs are executed.

This section describes how to create contexts and variables and define context parameter values. For an example of loading context parameters dynamically using the **tContextLoad** component, see the documentation of **tContextLoad** at <https://help.talend.com>. For an example of loading context parameters dynamically using the Implicit Context Load feature, see [Using the Implicit Context Load feature](#).

### 4.5.1. How to define context variables for a Job

You can define context variables for a particular Job in two ways:

- Using the **Contexts** view of the Job. See [How to define context variables in the Contexts view](#).

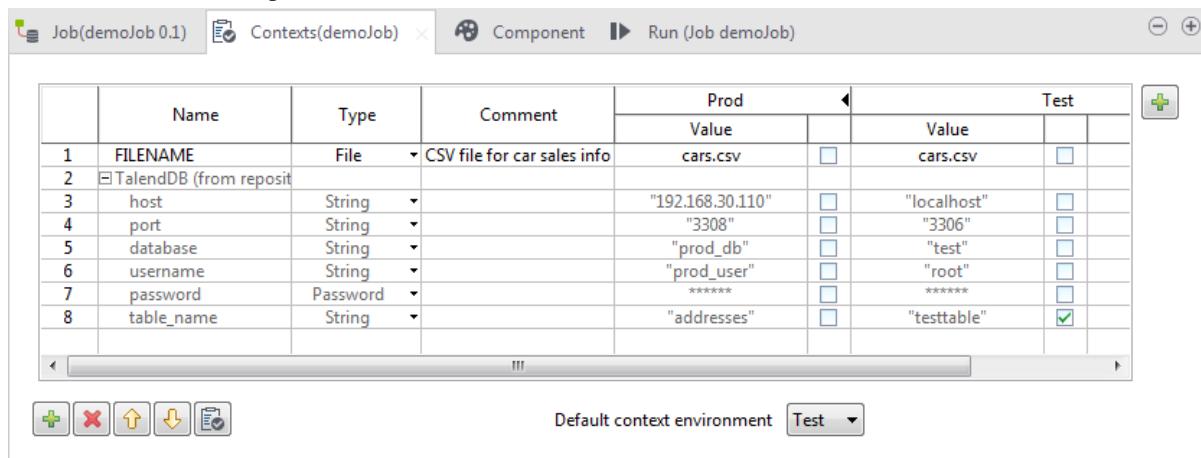
- Using the **F5** key from the **Component** view of a component. See [How to define variables from the Component view](#).

### 4.5.1.1. How to define context variables in the Contexts view

The **Contexts** view is positioned among the configuration tabs below design workspace.

 If you cannot find the **Contexts** view on the tab system of *Talend Studio*, go to **Window > Show view > Talend**, and select **Contexts**.

The **Contexts** tab view shows all of the variables that have been defined for each component in the current Job and context variables imported into the current Job.



The screenshot shows the Talend Studio interface with the 'Contexts(demoJob)' tab selected. The main area displays a table of context variables. The table has columns for Name, Type, Comment, Prod Value, and Test Value. The Prod column is bolded, and the Test column has a green plus sign icon. The table includes rows for FILENAME (File type, value cars.csv), host (String type, value "192.168.30.110"), port (String type, value "3308"), database (String type, value "prod\_db"), username (String type, value "prod\_user"), password (Password type, value \*\*\*\*), and table\_name (String type, value "addresses"). A row for TalendDB (from repository) is also present. At the bottom, there are icons for creating, deleting, and modifying contexts, and a dropdown for the default context environment set to 'Test'.

|   | Name                       | Type     | Comment                     | Prod             |                          | Test        |                                     |
|---|----------------------------|----------|-----------------------------|------------------|--------------------------|-------------|-------------------------------------|
|   |                            |          |                             | Value            |                          | Value       |                                     |
| 1 | FILENAME                   | File     | CSV file for car sales info | cars.csv         | <input type="checkbox"/> | cars.csv    | <input type="checkbox"/>            |
| 2 | TalendDB (from repository) |          |                             |                  |                          |             |                                     |
| 3 | host                       | String   |                             | "192.168.30.110" | <input type="checkbox"/> | "localhost" | <input type="checkbox"/>            |
| 4 | port                       | String   |                             | "3308"           | <input type="checkbox"/> | "3306"      | <input type="checkbox"/>            |
| 5 | database                   | String   |                             | "prod_db"        | <input type="checkbox"/> | "test"      | <input type="checkbox"/>            |
| 6 | username                   | String   |                             | "prod_user"      | <input type="checkbox"/> | "root"      | <input type="checkbox"/>            |
| 7 | password                   | Password |                             | *****            | <input type="checkbox"/> | *****       | <input type="checkbox"/>            |
| 8 | table_name                 | String   |                             | "addresses"      | <input type="checkbox"/> | "testtable" | <input checked="" type="checkbox"/> |

From this view, you can manage your built-in variables:

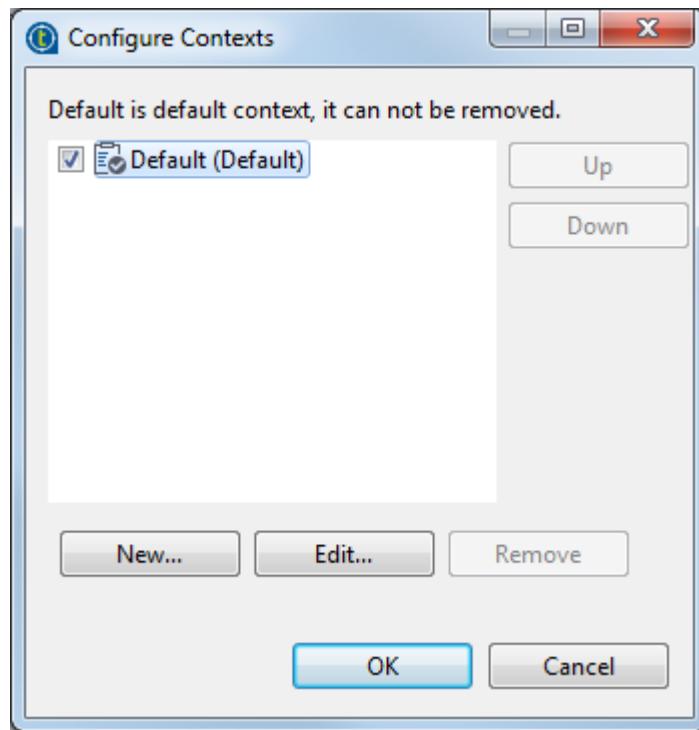
- Create and manage built-in contexts.
- Create, edit and delete built-in variables.
- Reorganize the context variables.
- Add built-in context variables to the Repository.
- Import variables from a Repository context source for use in the current Job.
- Edit Repository-stored context variables and update the changes to the Repository.
- Remove imported Repository variables from the current Job.

The following example will demonstrate how to define two contexts named *Prod* and *Test* and a set of variables - *host*, *port*, *database*, *username*, *password*, and *table\_name* - under the two contexts for a Job.

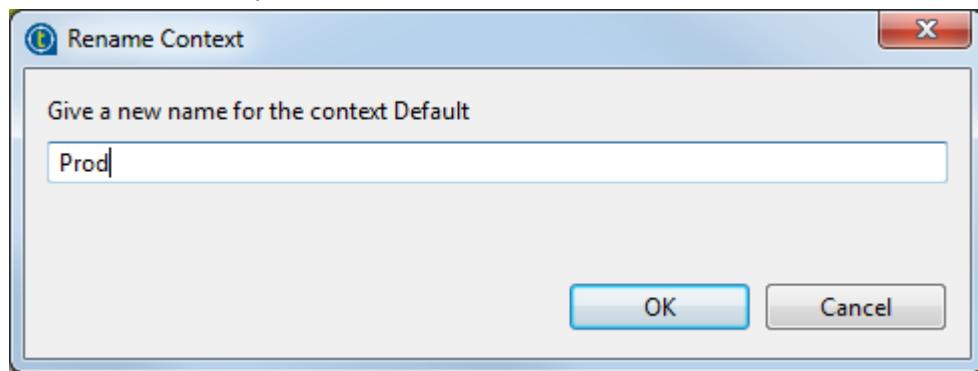
### Defining contexts

- Open the Job in the design workspace.
- Select the **Contexts** tab view and click the **[+]** button at the upper right corner of the view.

The **[Configure Contexts]** dialog box pops up. A context named *Default* has been created and set as the default one by the system.



3. Select the context *Default*, click the **Edit...** button and enter *Prod* in the **[Rename Context]** dialog box that opens to rename the context *Default* to *Prod*.

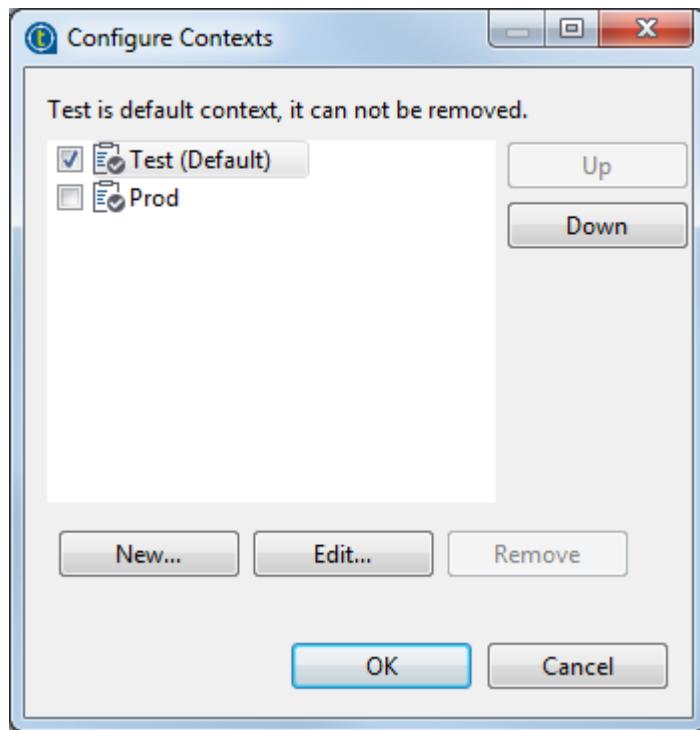


Then click **OK** to close the dialog box.

4. Click the **New...** button and enter *Test* in the **[New Context]** dialog box. Then click **OK** to close the dialog box.
5. Select the check box preceding the context you want to set it as the default context. You can also set the default context by selecting the context name from the **Default context environment** list in the **Contexts** tab view.

If needed, move a context up or down by selecting it and clicking the **Up** or **Down** button.

In this example, set *Test* as the default context and move it up.



- Click **OK** to validate your context definition and close the **[Configure Contexts]** dialog box.

The newly created contexts are shown in the context variables table of the **Contexts** tab view.

|  | Name | Type | Comment | Test<br>Value | Prod<br>Value |
|--|------|------|---------|---------------|---------------|
|  |      |      |         |               |               |

Below the table are standard table navigation buttons (+, -, up, down, search) and a 'Default context environment' dropdown set to 'Test'.

## Defining variables

- Click the **[+]** button at the bottom of the **Contexts** tab view to add a parameter line in the table.

|   | Name | Type   | Comment | Test<br>Value | Prod<br>Value |
|---|------|--------|---------|---------------|---------------|
| 1 | new1 | String |         |               |               |
|   |      |        |         |               |               |

Below the table are standard table navigation buttons (+, -, up, down, search) and a 'Default context environment' dropdown set to 'Test'.

- Click in the **Name** field and enter the name of the variable you are creating, *host* in this example.
- From the **Type** list, select the type of the variable corresponding to the component field where it will be used, **String** for the variable *host* in this example.

4. If needed, click in the **Comment** field and enter a comment to describe the variable.
5. Click in the **Value** field and enter the variable value under each context.

For different variable types, the **Value** field appear slightly different when you click in it and functions differently:

| Type                                                               | Value field                                                                                                              | Default value |
|--------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------|---------------|
| <b>String</b> (default type)                                       | Editable text field                                                                                                      | null          |
| <b>Boolean</b>                                                     | Drop-down list box with two options: <b>true</b> and <b>false</b>                                                        |               |
| <b>Character, Double, Integer, Long, Short, Object, BigDecimal</b> | Editable text field                                                                                                      |               |
| <b>Date</b>                                                        | Editable text field, with a button to open the <b>[Select Date &amp; Time]</b> dialog box.                               |               |
| <b>File</b>                                                        | Editable text field, with a button to open the <b>[Open]</b> dialog box for file selection.                              |               |
| <b>Directory</b>                                                   | Editable text field, with a button to open the <b>[Browse for Folder]</b> dialog box for folder selection.               |               |
| <b>List of Value</b>                                               | Editable text field, with a button to open the <b>[Configure Values]</b> dialog box for list creation and configuration. | (Empty)       |
| <b>Password</b>                                                    | Editable text field; text entered appears encrypted.                                                                     |               |



*It is recommended that you enclose the values of string type variables between double quotation marks to avoid possible errors during Job execution.*

6. If needed, select the check box next to the variable of interest and enter the prompt message in the corresponding **Prompt** field. This allows you to see a prompt for the variable value and to edit it at the execution time.

You can show/hide a **Prompt** column of the table by clicking the black right/left pointing triangle next to the relevant context name.

7. Repeat the steps above to define all the variables in this example.

- *port*, type **String**,
- *database*, type **String**,
- *username*, type **String**,
- *password*, type **Password**,
- *table\_name*, type **String**.

|   | Name       | Type     | Comment | Test        |                                     | Prod        |                                     |             |
|---|------------|----------|---------|-------------|-------------------------------------|-------------|-------------------------------------|-------------|
|   |            |          |         | Value       | Prompt                              | Value       | Prompt                              |             |
| 1 | host       | String   |         | "localhost" | <input type="checkbox"/>            | host?       | <input checked="" type="checkbox"/> | host?       |
| 2 | port       | String   |         | "3306"      | <input type="checkbox"/>            | port?       | <input type="checkbox"/>            | port?       |
| 3 | database   | String   |         | "test"      | <input type="checkbox"/>            | database?   | <input type="checkbox"/>            | database?   |
| 4 | username   | String   |         | "root"      | <input type="checkbox"/>            | username?   | <input type="checkbox"/>            | username?   |
| 5 | password   | Password |         | *****       | <input type="checkbox"/>            | password?   | <input type="checkbox"/>            | password?   |
| 6 | table_name | String   |         | "testtable" | <input checked="" type="checkbox"/> | table_name? | <input type="checkbox"/>            | table_name? |
|   |            |          |         |             |                                     |             |                                     |             |

Default context environment

All the variables created and their values under different contexts are displayed in the table and are ready for use in your Job. You can further edit the variables in this view if needed.

You can also add a built-in context variable to the Repository to make it reusable across different Jobs. For more information, see [How to add a built-in context variable to the Repository](#).

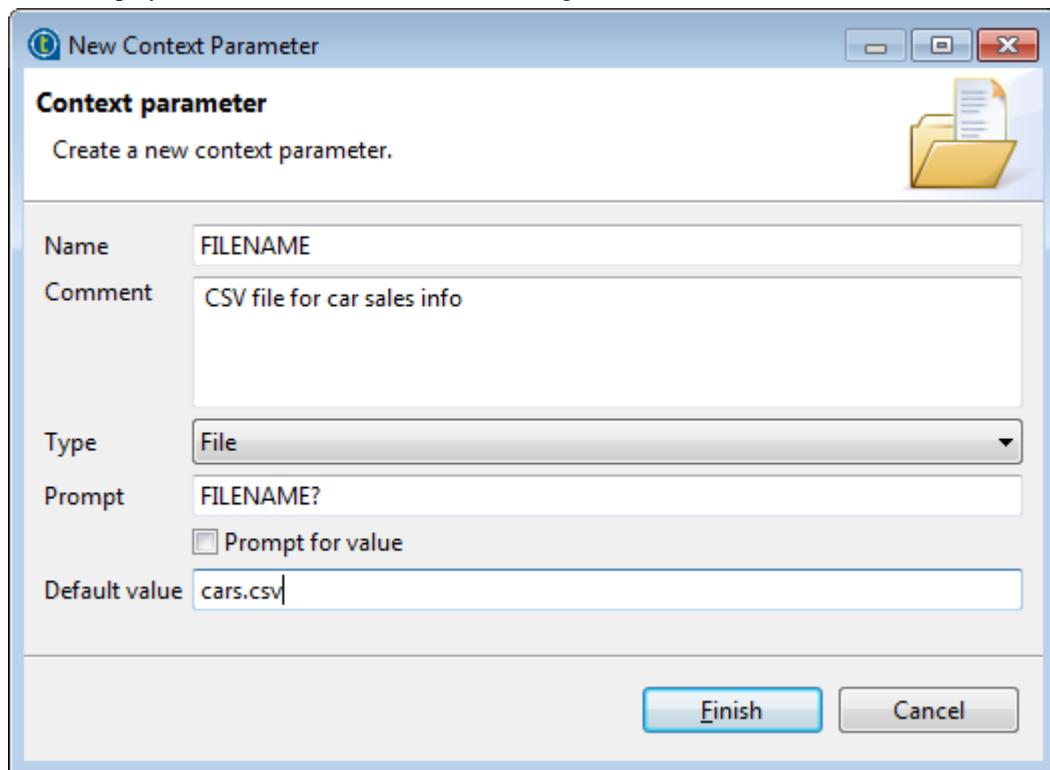
## Related topics:

- [How to define variables from the Component view](#)
- [How to centralize context variables in the Repository](#)
- [How to use variables in a Job](#)
- [How to run a Job in a selected context](#)

### 4.5.1.2. How to define variables from the Component view

The quickest way to create a single context variable is to use the **F5** key from the **Component** view:

1. On the relevant **Component** view, place your cursor in the field you want to parameterize.
2. Press **F5** to display the **[New Context Parameter]** dialog box:



3. Give a **Name** to this new variable, fill in the **Comment** field if needed, and choose the **Type**.
4. Enter a **Prompt** to be displayed to confirm the use of this variable in the current Job execution (generally used for test purpose only), select the **Prompt for value** check box to display the prompt message and an editable value field at the execution time.
5. If you filled in a value already in the corresponding properties field, this value is displayed in the **Default value** field. Else, type in the default value you want to use for one context.
6. Click **Finish** to validate.
7. Go to the **Contexts** view tab. Notice that the context variables tab lists the newly created variables.

The newly created variables are listed in the **Contexts** view.



The variable name should follow some typing rules and should not contain any forbidden characters, such as space character.

The variable created this way is automatically stored in all existing contexts, but you can subsequently change the value independently in each context. For more information on how to create or edit a context, see [Defining contexts](#).

### Related topics:

- [How to define context variables in the Contexts view](#)
- [How to centralize context variables in the Repository](#)
- [How to use variables in a Job](#)
- [How to run a Job in a selected context](#)

## 4.5.2. How to centralize context variables in the Repository

Context variables centrally stored in the Repository can be reused across various Jobs.

You can store context variables in the Repository in different ways:

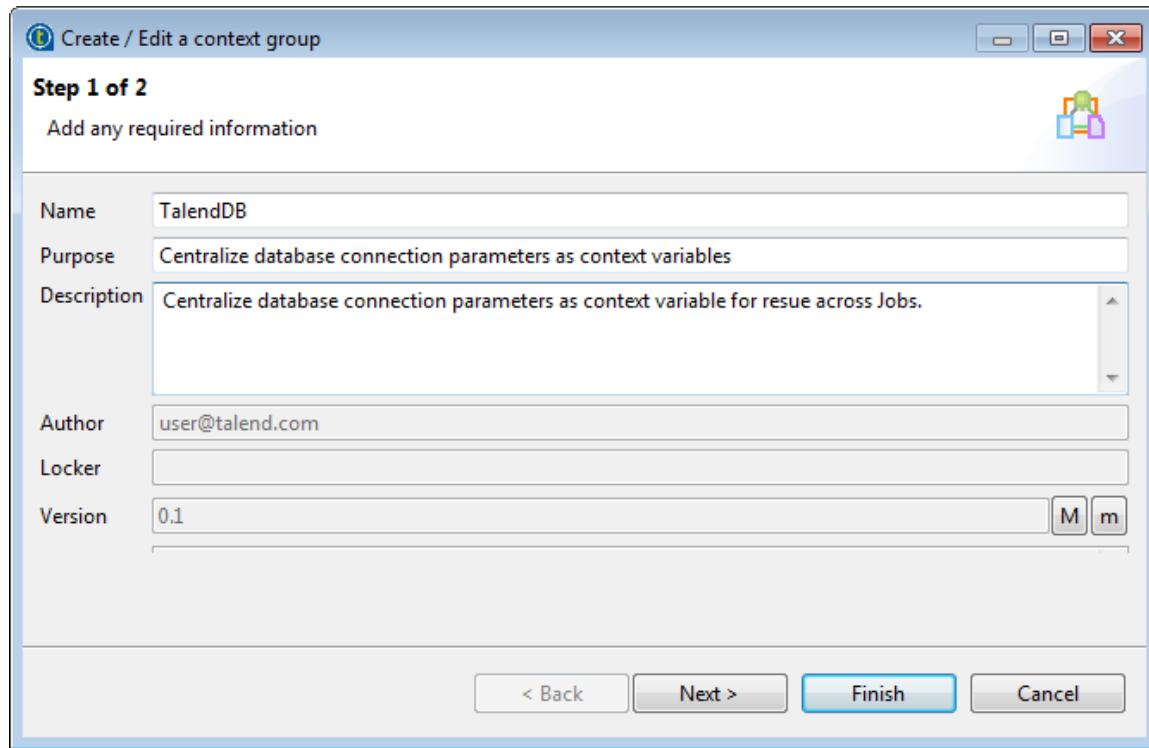
- Creating a context group using the **[Create / Edit a context group]** wizard. See [How to create a context group and define context variables in it](#) for details.
- Adding a built-in context variable to an existing or new context group in the Repository. See [How to add a built-in context variable to the Repository](#) for details.
- Saving a context from metadata. See [How to create a context from a Metadata](#) for more information.

### 4.5.2.1. How to create a context group and define context variables in it

The following example will demonstrate how to use the **[Create / Edit a context group]** wizard to create a context group named *TalendDB* that contains two contexts named *Prod* and *Test* and define a set of variables - *host*, *port*, *database*, *username*, *password*, and *table\_name* - under the two contexts in the Repository.

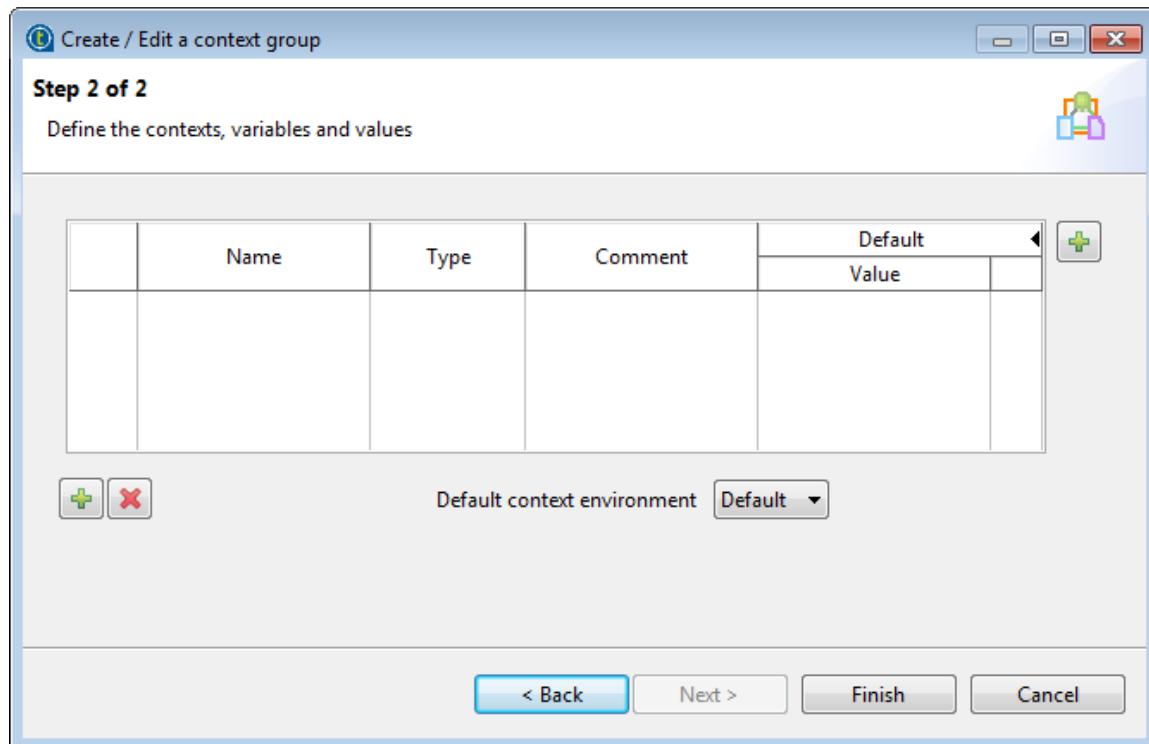
#### Create the context group and contexts

1. Right-click the **Contexts** node in the **Repository** tree view and select **Create context group** from the contextual menu.  
A 2-step wizard appears to help you define the various contexts and context parameters.
2. In Step 1 of 2, type in a name for the context group to be created, *TalendDB* in this example, and add any general information such as a description if required. The information you provide in the **Description** field will appear as a tooltip when you move your mouse over the context group in the Repository.

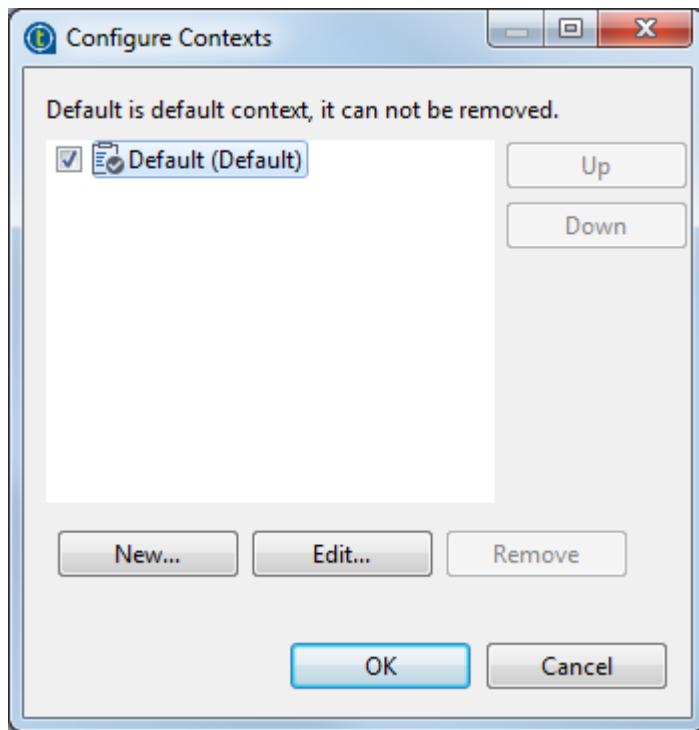


- Click **Next** to go to Step 2 of 2, which allows you to define the various contexts and variables that you need.

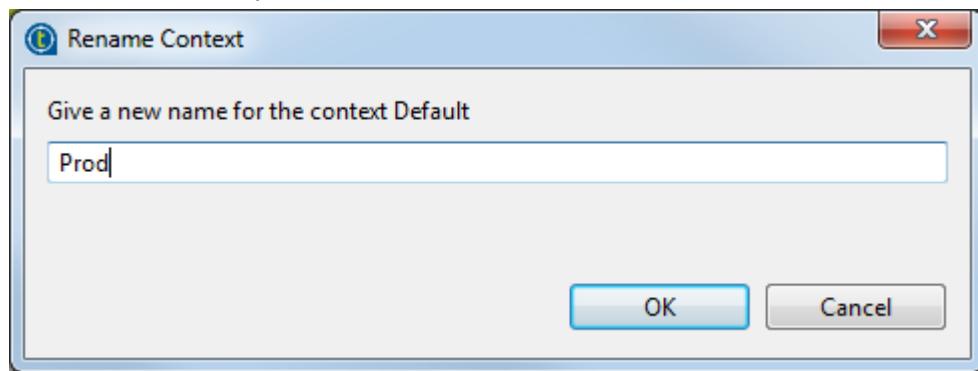
A context named *Default* has been created and set as the default one by the system.



- Click the [+] button at the upper right corner of the wizard to define contexts. The **[Configure Contexts]** dialog box pops up.



5. Select the context *Default*, click the **Edit...** button and enter *Prod* in the **[Rename Context]** dialog box that opens to rename the context *Default* to *Prod*.

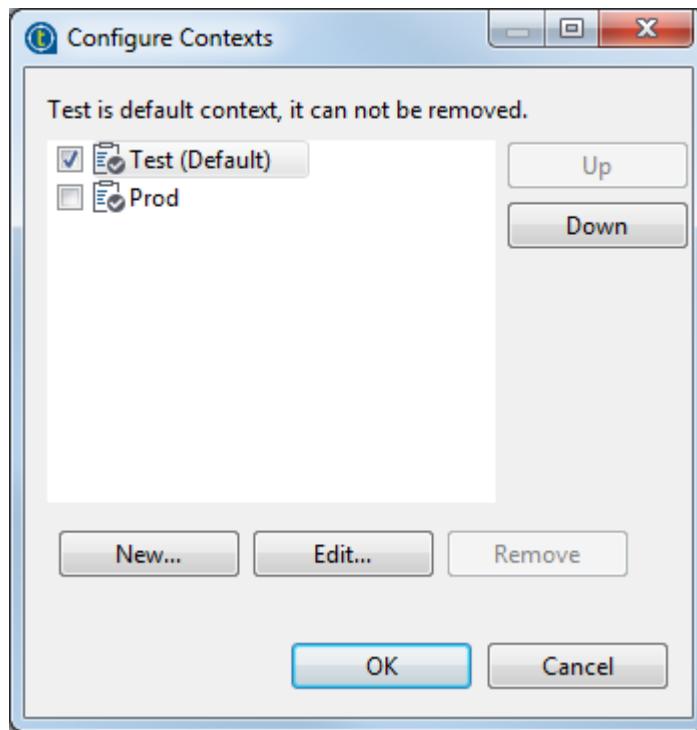


Then click **OK** to close the dialog box.

6. Click the **New...** button and enter *Test* in the **[New Context]** dialog box. Then click **OK** to close the dialog box.
7. Select the check box preceding the context you want to set as the default context. You can also set the default context by selecting the context name from the **Default context environment** list on the wizard.

If needed, move a context up or down by selecting it and clicking the **Up** or **Down** button.

In this example, set *Test* as the default context and move it up.



- Click **OK** to validate your context definition and close the **[Configure Contexts]** dialog box.

The newly created contexts are shown in the context variables table of the wizard.

A screenshot of the 'Create / Edit a context group' dialog box, specifically 'Step 2 of 2'. The title bar says 'Create / Edit a context group' and 'Step 2 of 2'. The main area is titled 'Define the contexts, variables and values' and contains a table for defining context variables. The table has columns: Name, Type, Comment, Test Value, and Prod Value. There is a '+' button to add new rows. Below the table is a 'Default context environment' dropdown set to 'Test'. At the bottom are '&lt; Back', 'Next &gt;', 'Finish', and 'Cancel' buttons.

## Define context variables

- Click the **[+]** button at the bottom of the wizard to add a parameter line in the table.
- Click in the **Name** field and enter the name of the variable you are creating, *host* in this example.

3. From the **Type** list, select the type of the variable corresponding to the component field where it will be used, **String** for the variable *host* in this example.
4. If needed, click in the **Comment** field and enter a comment to describe the variable.
5. Click in **Value** field and enter the variable value under each context.

For different variable types, the **Value** field appears slightly different when you click in it and functions differently:

| Type                                                               | Value field                                                                                                              | Default value |
|--------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------|---------------|
| <b>String</b> (default type)                                       | Editable text field                                                                                                      | null          |
| <b>Boolean</b>                                                     | Drop-down list box with two options: <b>true</b> and <b>false</b>                                                        |               |
| <b>Character, Double, Integer, Long, Short, Object, BigDecimal</b> | Editable text field                                                                                                      |               |
| <b>Date</b>                                                        | Editable text field, with a button to open the <b>[Select Date &amp; Time]</b> dialog box.                               |               |
| <b>File</b>                                                        | Editable text field, with a button to open the <b>[Open]</b> dialog box for file selection.                              |               |
| <b>Directory</b>                                                   | Editable text field, with a button to open the <b>[Browse for Folder]</b> dialog box for folder selection.               |               |
| <b>List of Value</b>                                               | Editable text field, with a button to open the <b>[Configure Values]</b> dialog box for list creation and configuration. | (Empty)       |
| <b>Password</b>                                                    | Editable text field; text entered appears encrypted.                                                                     |               |

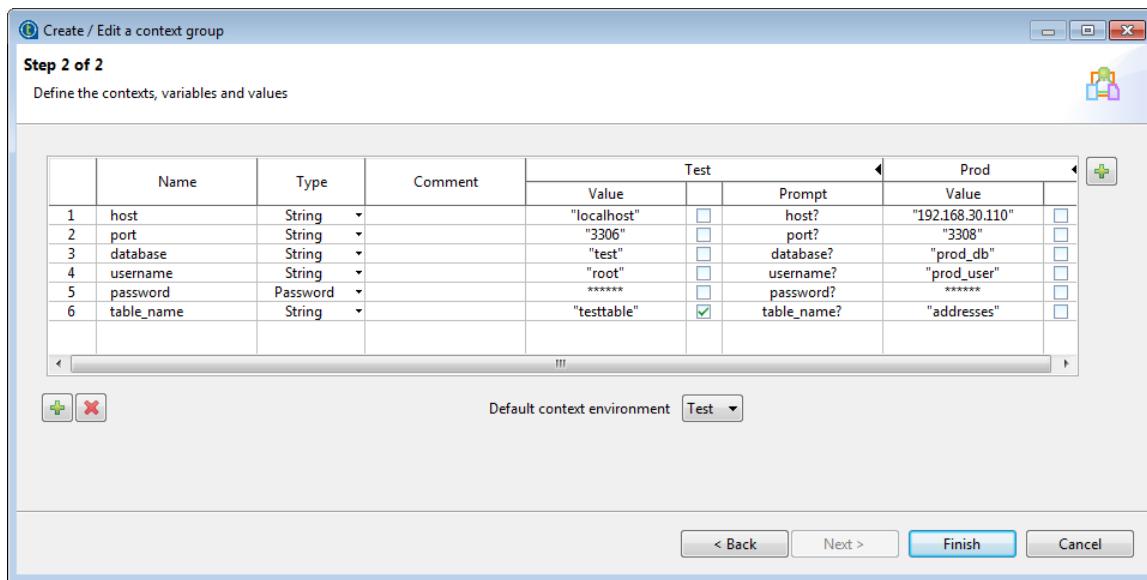


*It is recommended that you enclose the values of string type variables between double quotation marks to avoid possible errors during Job execution.*

6. If needed, select the check box next to the variable of interest and enter the prompt message in the corresponding **Prompt** field. This allows you to see a prompt for the variable value and to edit it at the execution time.

You can show/hide a **Prompt** column of the table by clicking the black right/left pointing triangle next to the relevant context name.

7. Repeat the steps above to define all the variables in this example.
  - *port*, type **String**,
  - *database*, type **String**,
  - *username*, type **String**,
  - *password*, type **Password**,
  - *table\_name*, type **String**



All the variables created and their values under different contexts are displayed in the table and are ready for use in your Job. You can further edit the variables if needed.

Once you created and adapted as many context sets as you want, click **Finish** to validate. The group of contexts thus displays under the **Contexts** node in the **Repository** tree view. You can further edit the context group, contexts, and context variables in the wizard by right-clicking the **Contexts** node and selecting **Edit context group** from the contextual menu.

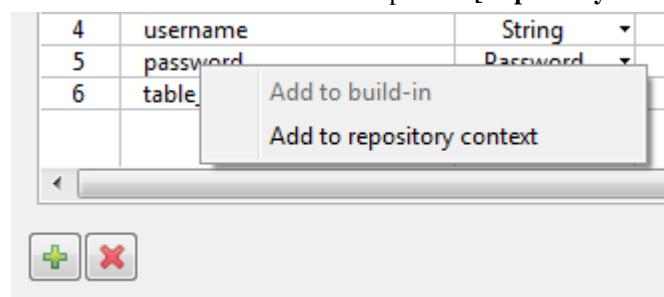
### Related topics:

- [How to add a built-in context variable to the Repository](#)
- [How to create a context from a Metadata](#)
- [How to apply Repository context variables to a Job](#)
- [How to define context variables for a Job.](#)
- [How to use variables in a Job](#)
- [How to run a Job in a selected context](#)

## 4.5.2.2. How to add a built-in context variable to the Repository

You can save a built-in context variable defined in a Job to a new context group, or an existing context group provided that the context variable does not already exist in the group.

1. In the **Context** tab view of a Job, right-click the context variable you want to add to the Repository and select **Add to repository context** from the contextual menu to open the **[Repository Content]** dialog box.



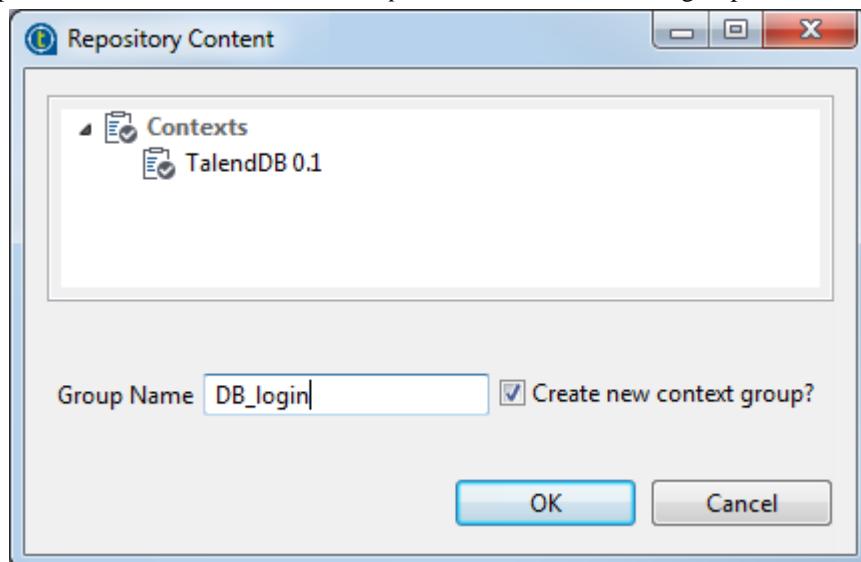
2. In the dialog box, do either of the following:

- to add your context variable to a new context group, select **Create new context group** and enter a name for the new context group in the **Group Name** field, and then click **OK**.
- to add your context variable to an existing context group, select the context group and click **OK**.



*When adding a built-in context variable to an existing context group, make sure that the variable does not already exist in the context group.*

In this example, add the built-in context variable *password* to a new context group named *DB\_login*.



The context variable is added to the Repository context group of your choice, along with the defined built-in contexts, and it appears as a Repository-stored context variable in the **Contexts** tab view.

|   | Name                       | Type     | Comment | Test        |                                     | Prod        |                  |
|---|----------------------------|----------|---------|-------------|-------------------------------------|-------------|------------------|
|   |                            |          |         | Value       | Prompt                              | Value       | Prompt           |
| 1 | host                       | String   |         | "localhost" | <input type="checkbox"/>            | host?       | "192.168.30.110" |
| 2 | port                       | String   |         | "3306"      | <input type="checkbox"/>            | port?       | "3308"           |
| 3 | database                   | String   |         | "test"      | <input type="checkbox"/>            | database?   | "prod_db"        |
| 4 | username                   | String   |         | "root"      | <input type="checkbox"/>            | username?   | "prod_user"      |
| 5 | DB_login (from repository) |          |         |             |                                     |             |                  |
| 6 | password                   | Password |         | *****       | <input type="checkbox"/>            | password?   | *****            |
| 7 | table_name                 | String   |         | "testtable" | <input checked="" type="checkbox"/> | table_name? | "addresses"      |
|   |                            |          |         |             |                                     |             |                  |



Default context environment

## Related topics:

- [How to create a context group and define context variables in it](#)
- [How to create a context from a Metadata](#)
- [How to apply Repository context variables to a Job](#)
- [How to define context variables for a Job.](#)
- [How to use variables in a Job](#)
- [How to run a Job in a selected context](#)

### 4.5.2.3. How to create a context from a Metadata

When creating or editing a metadata connection (through a File or DB metadata wizard), you have the possibility to save the connection parameters as context variables in a newly created context group under the **Contexts** node

of the Repository. To do so, complete your connection details and click the **Export as context** button in the second step of the wizard.

For more information about this feature, see [Exporting metadata as context and reusing context parameters to set up a connection](#).

## 4.5.3. How to apply Repository context variables to a Job

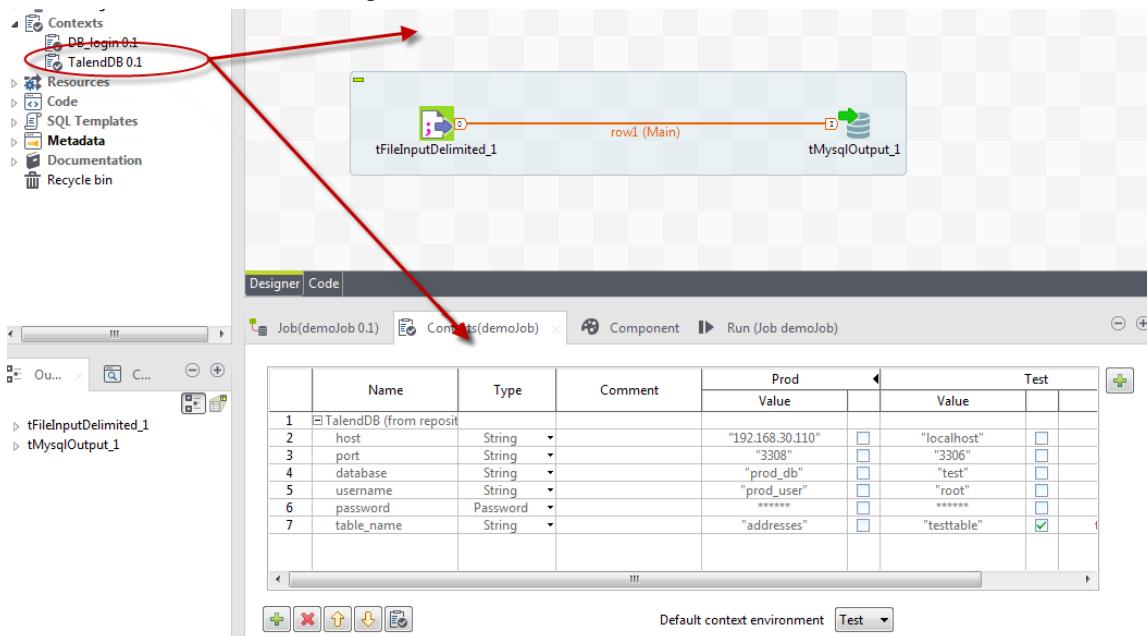
Once a context group is created and stored in the **Repository**, there are two ways of applying it to a Job:

- Drop a context group. This way, the group is applied as a whole. See [How to drop a context group onto a Job](#) for details.
- Use the  button. This way, the variables of a context group can be applied separately. See [How to apply context variables to a Job using the context button](#) for details.

### 4.5.3.1. How to drop a context group onto a Job

To drop a context group onto a Job, proceed as follows:

1. Double-click the Job to which a context group is to be added.
2. Once the Job is opened, drop the context group of your choice either onto the Job workspace or onto the **Contexts** view beneath the workspace.



The **Contexts** view shows all the contexts and variables of the group. You can:

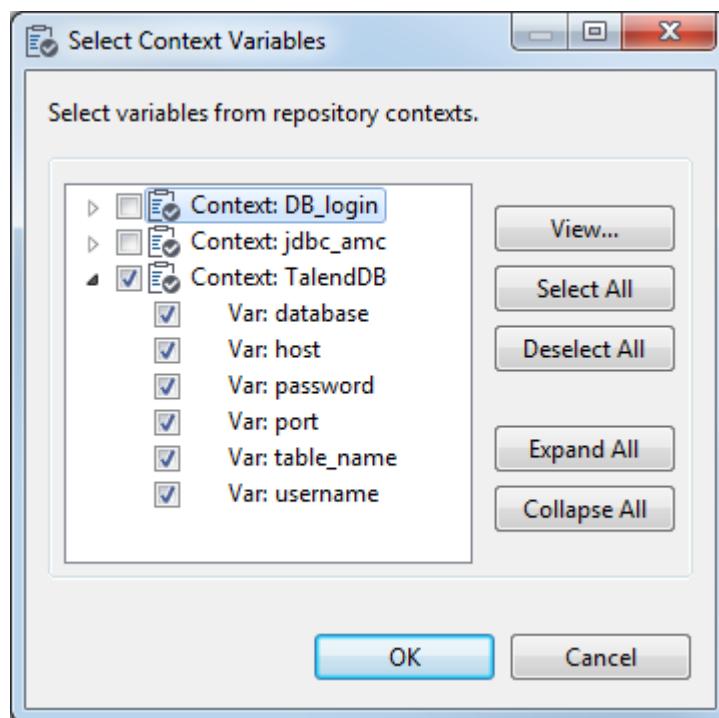
- edit the contexts by clicking the [+] button at the upper right corner of the **Contexts** view.
- delete the whole group or any variable by selecting the group name or the variable and clicking the [X] button.

- save any imported context variable as a built-in variable by right-click it and selecting **Add to built-in** from the contextual menu.
- double-click any context variable to open the context group in the [**Create / Edit a context group**] wizard and update changes to the Repository.

### 4.5.3.2. How to apply context variables to a Job using the context button

To use the context button to apply context variables to a Job, proceed as follows:

1. Double-click the Job to which a context variable is to be added.
2. Once the Job is opened in the workspace, click the **Contexts** view beneath the workspace to open it.
3. At the bottom of the **Contexts** view, click the  button to open the wizard to select the context variables to be applied.



4. In the wizard, select the context variables you need to apply or clear those you do not need to.



The context variables that have been applied are automatically selected and cannot be cleared.

5. Click **OK** to apply the selected context variables to the Job.

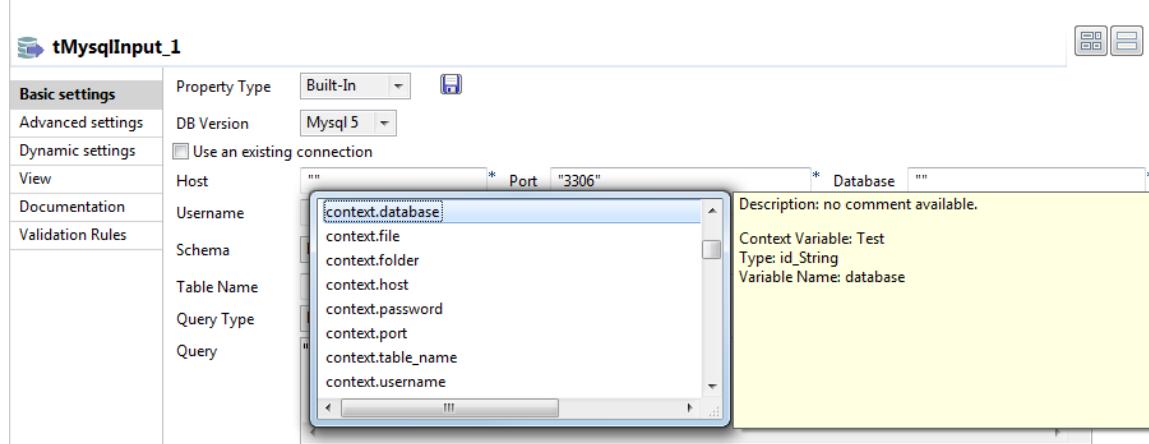
The **Contexts** view shows the context group and the selected context variables. You can edit the contexts by clicking the **[+]** button at the upper right corner of the **Contexts** view, delete the whole group or any variable by selecting the group name or the variable and clicking the **[X]** button, but you cannot edit Repository-stored variables in this view.

## 4.5.4. How to use variables in a Job

You can use an existing global variable, a context variable defined in your Job, or a Repository-stored context variable applied to your Job in any component properties field.

1. In the relevant **Component** view, place your mouse in the field you want to parameterize and press **Ctrl + Space** to display a full list of all the global variables and those context variables defined in or applied to your Job.

The list grows along with new user-defined variables (context variables).



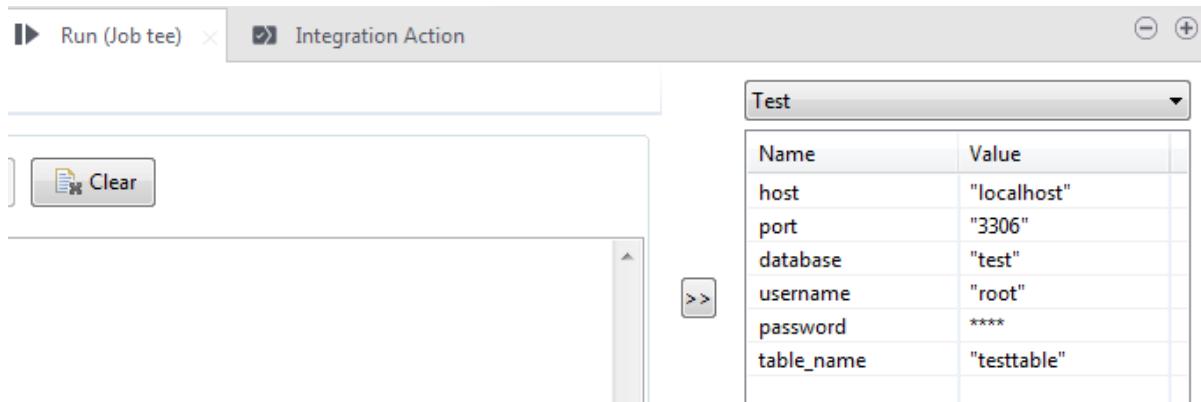
2. Double-click the variable of your choice to fill it in the field.

### Related topics:

- [How to define context variables for a Job](#)
- [How to centralize context variables in the Repository](#)
- [How to apply Repository context variables to a Job](#)
- [How to run a Job in a selected context](#)

## 4.5.5. How to run a Job in a selected context

You can select the context you want the Job design to be executed in.



Click the **Run Job** tab, and in the **Context** area, select the relevant context among the various ones you created.

If you did not create any context, only the **Default** context shows on the list.

All the context variables you created for the selected context display, along with their respective values, in a table underneath.

To make a change permanent in a variable value, you need to change it on the Context view if your variable is of type built-in or in the Context group of the repository.

### Related topics:

- [\*How to define context variables for a Job\*](#)
- [\*How to centralize context variables in the Repository\*](#)
- [\*How to apply Repository context variables to a Job\*](#)
- [\*How to use variables in a Job\*](#)

## 4.5.6. StoreSQLQuery

**StoreSQLQuery** is a user-defined variable and is mainly dedicated to debugging.

**StoreSQLQuery** is different from other context variables in the fact that its main purpose is to be used as parameter of the specific global variable called **Query**. It allows you to dynamically feed the global query variable.

The global variable **Query**, is available on the proposals list (**Ctrl+Space bar**) for some DB input components.

For further details on **StoreSQLQuery** settings, see the scenario of **tDBInput** at <https://help.talend.com>.

## 4.6. Using parallelization to optimize Job performance

Parallelization in terms of **Talend** Jobs means to accomplish technical processes through parallel executions. When properly designed, a parallelization-enabled technical process can be completed within a shorter time frame.

**Talend** Studio allows you to implement different types of parallelization depending on ranging circumstances. These circumstances could be:

1. Parallel executions of multiple Subjobs. For further information, see [\*How to execute multiple Subjobs in parallel\*](#).
2. Parallel iterations for reading data. For further information, see [\*How to launch parallel iterations to read data\*](#).

Parallelization is an advanced feature and requires basic knowledge about a **Talend** Job such as how to design and execute a Job or a Subjob, how to use components and how to use the different types of connections that link components or Jobs. If you feel that you need to acquire this kind of knowledge, see [\*Designing a Job\*](#).

### 4.6.1. How to execute multiple Subjobs in parallel

The **Multi thread execution** feature allows you to run multiple Subjobs that are active in the workspace in parallel.

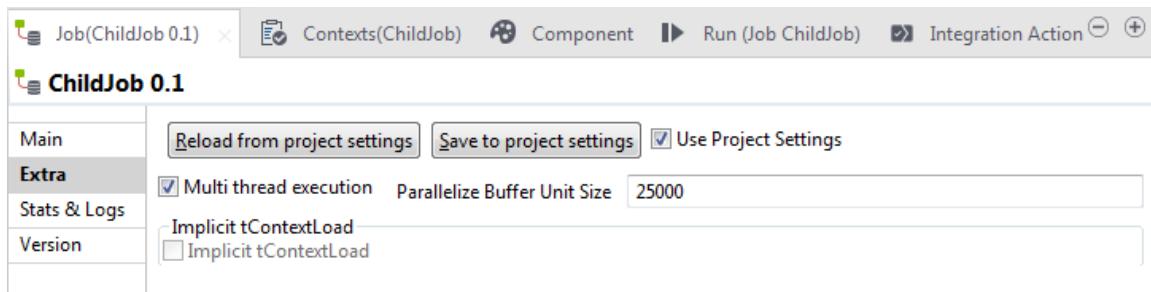
As explained in the previous sections, a Job opened in the workspace can contain several Subjobs and you are able to arrange their execution order using the trigger links such as **OnSubjobOK**. However, when the Subjobs do not have any dependencies between them, you might want to launch them at the same time. For example, the following image presents four Subjobs within a Job and with no dependencies in between.



The **tRunJob** component is used in this example to call each Subjob they represent. For further information about **tRunJob**, see the component documentation at <https://help.talend.com>.

Then with the Job opened in the workspace, you need simply proceed as follows to run the Subjobs in parallel:

1. Click the **Job** tab, then the **Extra** tab to display it.



2. Select the **Multi thread execution** check box to enable the parallel execution.

When the **Use project settings** check box is selected, the **Multi thread execution** check box could be greyed out and become unavailable. In this situation, clear the **Use project settings** check box to activate the **Multi thread execution** check box.

This feature is optimal when the number of threads (in general a Subjob count one thread) do not exceed the number of processors of the machine you use for parallel executions. Otherwise, some of the Subjobs have to wait until any processor is freed up.

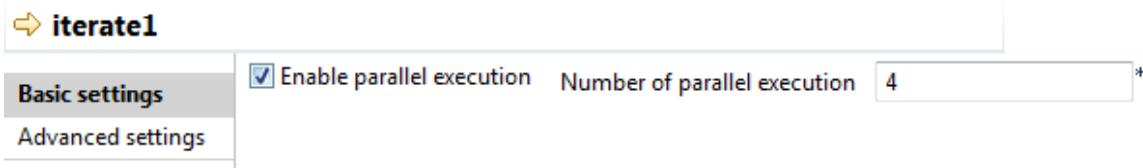
For a use case of using this feature to run Jobs in parallel, see [Using the Multi-thread Execution feature to run Jobs in parallel](#).

## 4.6.2. How to launch parallel iterations to read data

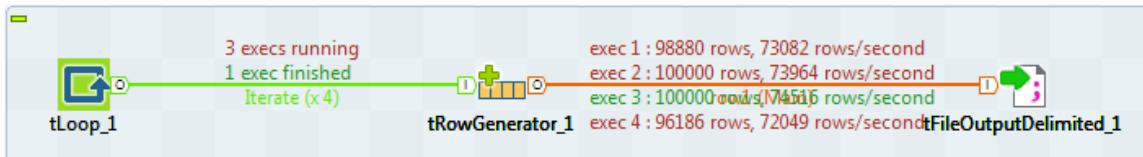
A parallelization-enabled Iterate connection allows the component that receives threads from the connection to read those threads in parallel.

You need to proceed as follows to set the parallel iterations:

1. Simply select the **Iterate** link of your subjob to display the related **Basic settings** view of the **Components** tab.
2. Select the **Enable parallel execution** check box and set the number of executions to be carried out in parallel.



When executing your Job, the number of parallel iterations will be distributed onto the available processors.



3. Select the **Statistics** check box of the **Run** view to show the real time parallel executions on the design workspace.

This feature is especially useful when you need to use the Iterate connection to pass context variables to a Subjob. In that situation, the variables will be read in parallel in the Subjob and thus the processes handled by the Subjob will be simultaneously run using those variables.

## 4.7. Handling Jobs: advanced subjects

The sections below give detail information about various advanced configuration situations of a data integration Job including handling multiple input and output flows, using SQL queries, using external components in the Job, scheduling a task to run your Job.

### 4.7.1. How to map data flows

The most common way to handle multiple input and output flows in your Job including transformations and data re-routing is to use the **tMap** component.

For more information about the principles of using this component, see [Designing a Job](#).

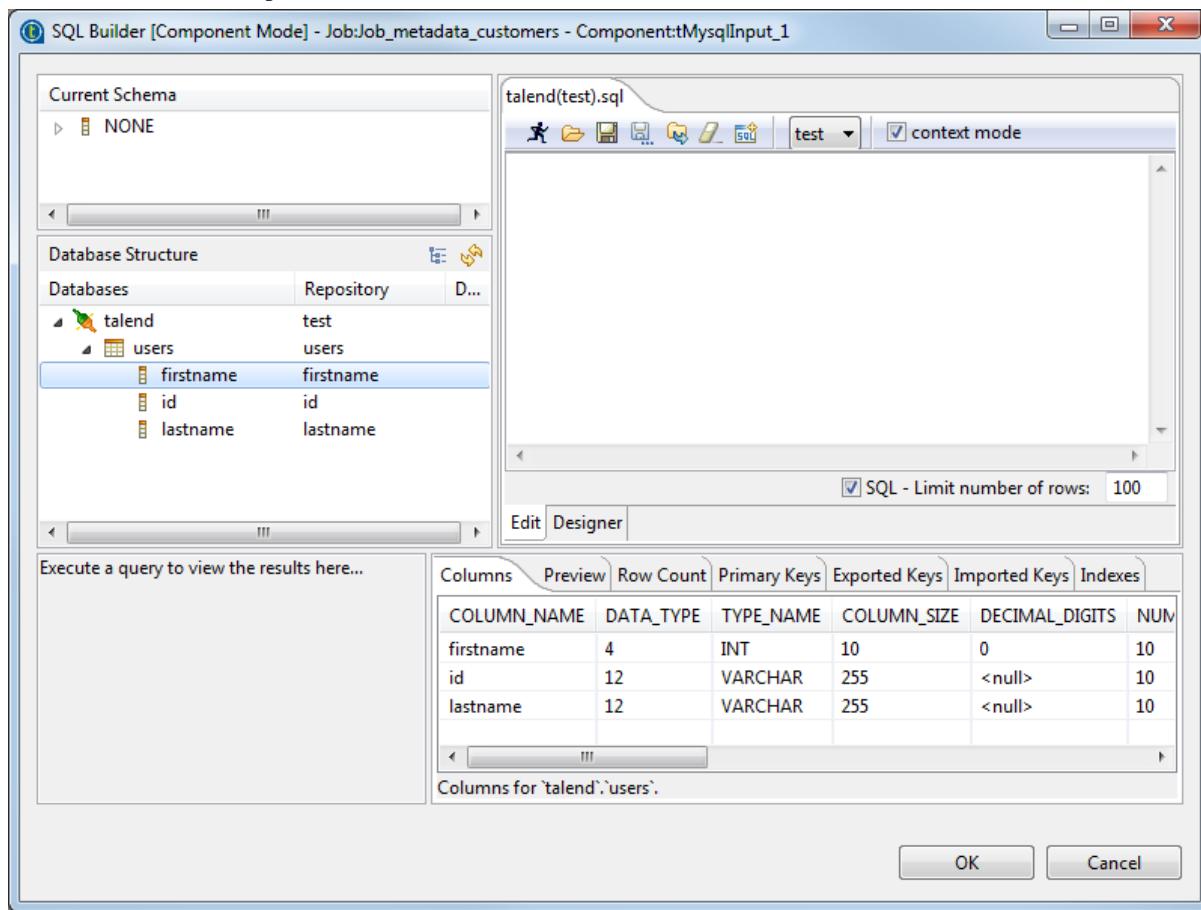
### 4.7.2. How to create queries using the SQLBuilder

SQLBuilder helps you create your SQL queries and monitor the changes between DB tables and metadata tables. This editor is available in all DBInput and DBSQLRow components (specific or generic).

You can create a query using the SQLbuilder whether your database table schema is stored in the **Repository** tree view or built-in directly in the Job.

Fill in the DB connection details and select the appropriate repository entry if you defined it.

Remove the default query statement in the **Query** field of the **Basic settings** view of the **Component** panel. Then click the [...] button to open the **[SQL Builder]** editor.



The **[SQL Builder]** editor is made of the following panels:

- Current Schema,
- Database structure,
- Query editor made of editor and designer tabs,
- Query execution view,
- Schema view.

The Database structure shows the tables for which a schema was defined either in the repository database entry or in your built-in connection.

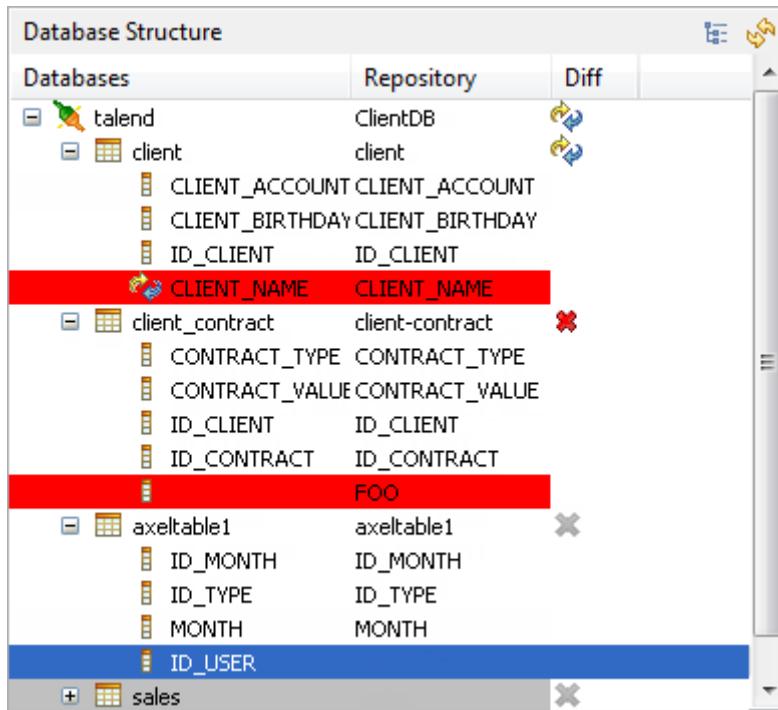
The schema view, in the bottom right corner of the editor, shows the column description.

#### 4.7.2.1. How to compare database structures

On the **Database Structure** panel, you can see all tables stored in the DB connection metadata entry in the **Repository** tree view, or in case of built-in schema, the tables of the database itself.

 The connection to the database, in case of built-in schema or in case of a refreshing operation of a repository schema might take quite some time.

Click the refresh icon to display the differences between the DB metadata tables and the actual DB tables.



The **Diff** icons point out that the table contains differences or gaps. Expand the table node to show the exact column containing the differences.

The red highlight shows that the content of the column contains differences or that the column is missing from the actual database table.

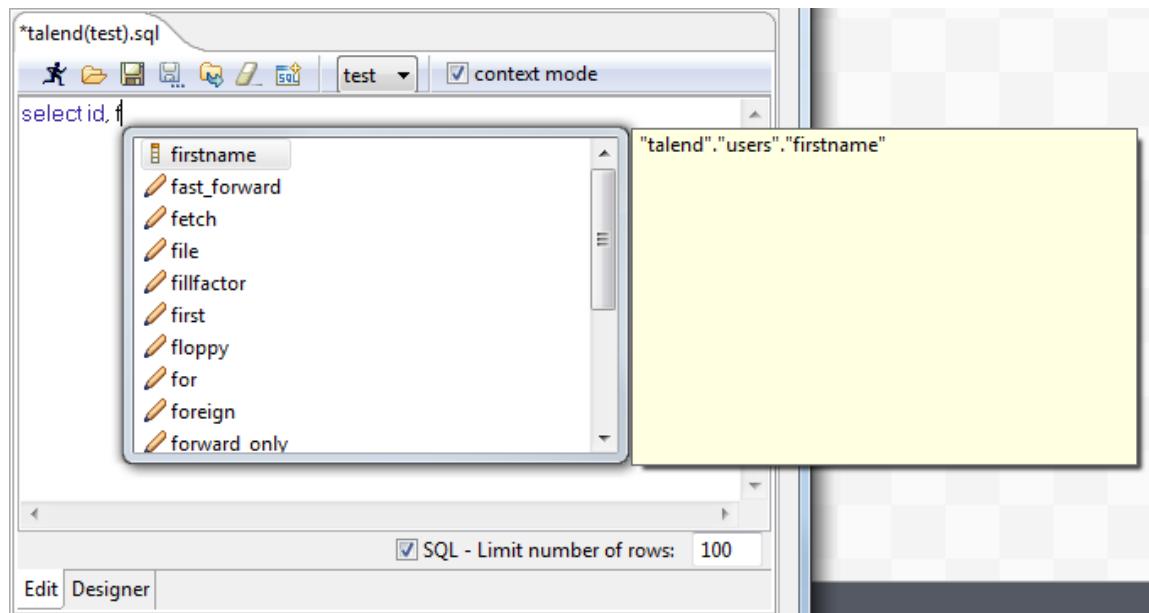
The blue highlight shows that the column is missing from the table stored in **Repository > Metadata**.

#### 4.7.2.2. How to create a query

The **[SQL Builder]** editor is a multiple-tab editor that allows you to write or graphically design as many queries as you want.

To create a new query, complete the following:

1. Right-click the table or on the table column and select **Generate Select Statement** on the pop-up list.
2. Click the empty tab showing by default and type in your SQL query or press **Ctrl+Space** to access the autocompletion list. The tooltip bubble shows the whole path to the table or table section you want to search in.



Alternatively, the graphical query **Designer** allows you to handle tables easily and have real-time generation of the corresponding query in the **Edit** tab.

3. Click the **Designer** tab to switch from the manual **Edit** mode to the graphical mode.

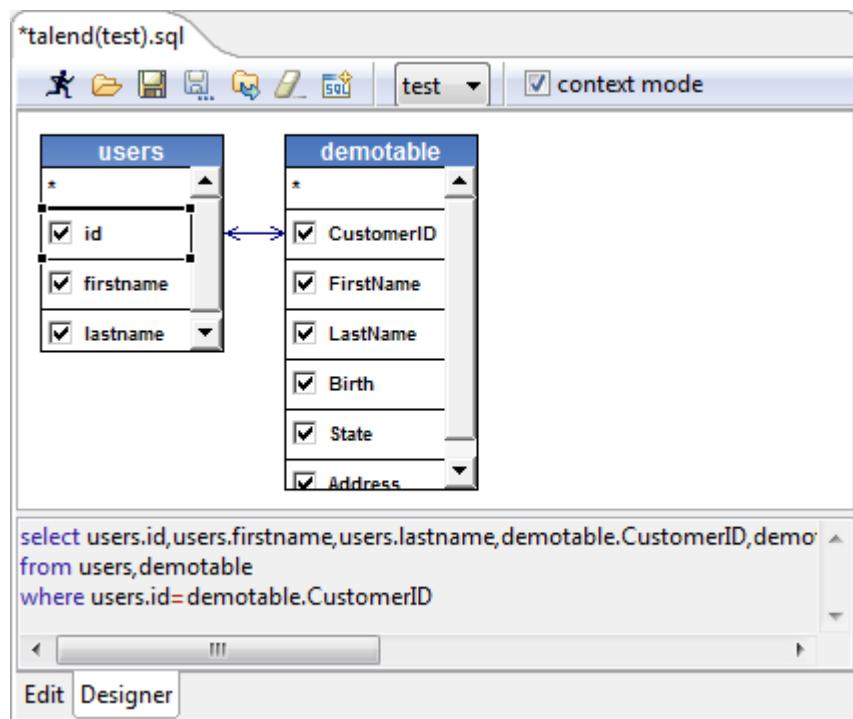


You may get a message while switching from one view to the other as some SQL statements cannot be interpreted graphically.

4. If you selected a table, all columns are selected by default. Clear the check box facing the relevant columns to exclude them from the selection.
5. Add more tables in a simple right-click. On the **Designer** view, right-click and select **Add tables** in the pop-up list then select the relevant table to be added.

If joins between these tables already exist, these joins are automatically set up graphically in the editor.

You can also create a join between tables very easily. Right-click the first table columns to be linked and select **Equal** on the pop-up list, to join it with the relevant field of the second table.



The SQL statement corresponding to your graphical handlings is also displayed on the viewer part of the editor or click the **Edit** tab to switch back to the manual **Edit** mode.



In the **Designer** view, you cannot include graphically filter criteria. You need to add these in the **Edit** view.

6. Once your query is complete, execute it by clicking the  icon on the toolbar.

The toolbar of the query editor allows you to access quickly usual commands such as: execute, open, save and clear.

The results of the active query are displayed on the **Results** view in the lower left corner.

7. If needed, you can select the **context mode** check box to keep the original query statement and customize it properly in the **Query** area of the component. For example, if a context parameter is used in the query statement, you cannot execute it by clicking the  icon on the toolbar.
8. Click **OK**. The query statement will be loaded automatically in the **Query** area of the component.

### 4.7.2.3. How to store a query in the repository

To be able to retrieve and reuse queries, we recommend you to store them in the repository.

In the **[SQL Builder]** editor, click the icon on the toolbar to bind the query with the DB connection and schema in case these are also stored in the repository.

The query can then be accessed from the **Database structure** view, on the left-hand side of the editor.

## 4.7.3. How to download/upload Talend Community components

*Talend Studio* enables you to access a list of all community components in **Talend Exchange** that are compatible with your current version of *Talend Studio*. You can then download and install these components to use them later in the Job designs you carry out in the Studio. From *Talend Studio*, you can also upload components you have created to **Talend Exchange** to share with other community users.

A click on the **Exchange** link on the toolbar of *Talend Studio* opens the **Exchange** tab view on the design workspace, where you can find lists of:

- components available in **Talend Exchange** for you to download and install,
- components you downloaded and installed in previous versions of *Talend Studio* but not installed yet in your current Studio,
- components you have created and uploaded to **Talend Exchange** to share with other **Talend** Community users.

Note that the approach explained in this section is to be used for the above-mentioned components only.



- Before you can download community components or upload your own components to the community, you need to sign in to **Talend Exchange** from your Studio first. If you did not sign in to **Talend Exchange** when launching the Studio, you still have a chance to sign in from the **Talend Exchange** preferences settings page. For more information, see [Exchange preferences \(Talend > Exchange\)](#).
- The community components available for download are not validated by **Talend**. This explains why you may encounter component loading errors sometimes when trying to install certain community components, why an installed community component may have a different name in the **Palette** than in the **Exchange** tab view, and why you may not be able to find a component in the **Palette** after it is seemingly installed successfully.

### 4.7.3.1. How to install community components from Talend Exchange

To install community components from **Talend Exchange** to the **Palette** of your current *Talend Studio*:

1. Click the **Exchange** link on the toolbar of *Talend Studio* to open the **Exchange** tab view on the design workspace.

| Available Extensions  |  |                                         |         |        |          |                               |
|-----------------------|--|-----------------------------------------|---------|--------|----------|-------------------------------|
| Downloaded Extensions |  |                                         |         |        |          |                               |
| My Extensions         |  | Extension Name                          | Version | Rating | Author   | View                          |
|                       |  | Facebook Application Insights Component | 0.1     | ★★★★★  | saburo   | <a href="#">view/download</a> |
|                       |  | tFileOutputDelimitedEx                  | 1.0     | ★★★★★  | Alezis   | <a href="#">view/download</a> |
|                       |  | tScriptRules                            | 1.0     | ★★★★★  | walkerca | <a href="#">view/download</a> |
|                       |  | tWorldBank components Demo              | 0.2     | ★★★★★  | saburo   | <a href="#">view/download</a> |
|                       |  | pUpdateMailOffer                        | V02     | ★★★★★  | PayZen   | <a href="#">view/download</a> |
|                       |  | pCreateMailOffer                        | V02     | ★★★★★  | PayZen   | <a href="#">view/download</a> |
|                       |  | tDBFOutput                              | 1.1     | ★★★★★  | BiSi     | <a href="#">view/download</a> |
|                       |  | tDBFInput                               | 1.1     | ★★★★★  | BiSi     | <a href="#">view/download</a> |
|                       |  | pCreatePayment                          | V06     | ★★★★★  | PayZen   | <a href="#">view/download</a> |

2. In the **Available Extensions** view, if needed, enter a full component name or part of it in the text field and click the fresh button to find quickly the component you are interested in.
3. Click the **view/download** link for the component of interest to display the component download page.

**tPDFToText**  
Version 1.1  
2011-05-17

Convert a PDF to text file. It's possible to extract a delimited area.





**Install**

#### User Reviews

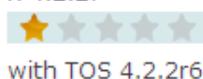
[write a review](#)



bien it's good



It works on 4.2.2 You can use it, it works on 4.2.2!



tPDFToText This does not seem compatible with TOS 4.2.2r63143. I have installed it on TOS and it does not generate an output file.

4. View the information about the component, including component description and review comments from community users, or write your own review comments and/or rate the component if you want. For more information on reviewing and rating a community component, see [How to review and rate a community component](#).

If needed, click the left arrow button to return to the component list page.

5. Click the **Install** button in the right part of the component download page to start the download and installation process.

A progress indicator appears to show the completion percentage of the download and installation process. Upon successful installation of the component, the **Downloaded Extensions** view opens and displays the status of the component, which is **Installed**.

| Available Extensions  | Extension Name         | Downloaded Version | Download Date | Install/Update          |
|-----------------------|------------------------|--------------------|---------------|-------------------------|
| Downloaded Extensions |                        |                    |               |                         |
| My Extensions         | tDBFInput              | 1.1                | 2011-11-17    | <a href="#">Install</a> |
|                       | tDBFOoutput            | 1.1                | 2011-10-31    | <a href="#">Install</a> |
|                       | bcLogbackConfig        | 1.2                | 2011-10-31    | <a href="#">Install</a> |
|                       | bcLogbackCatch         | 1.3                | 2011-10-31    | <a href="#">Install</a> |
|                       | tLog4J                 | 1.4                | 2011-11-17    | <a href="#">Install</a> |
|                       | tFileOutputDelimitedEx | 1.0                | 2011-11-17    | <a href="#">Install</a> |
|                       | null                   | null               | 2011-11-17    | <a href="#">Install</a> |
|                       | tScriptRules           | 1.0                | 2011-11-17    | <a href="#">Install</a> |
|                       | BRules                 | 1.1                | 2011-11-17    | <a href="#">Install</a> |
|                       | tPDFToText             | 1.1                | 2011-11-18    | Installed               |

### 4.7.3.2. How to reinstall or update community components

From the **Exchange** tab view, you can reinstall components you already downloaded and installed in your previous version of *Talend Studio* or install the updated version of *Talend Studio* or components in your current Studio.



By default, while you are connected to **Talend Exchange**, a dialog box appears to notify you whenever an update to an installed community component is available. If you often check for community component updates and you do not want that dialog box to appear again, you can turn it off in **Talend Exchange** preferences settings. For more information, see [Exchange preferences \(Talend > Exchange\)](#).

To reinstall a community component you already downloaded or update an installed one, do the following:

1. From the **Exchange** tab view, click **Downloaded Extensions** to display the list of components you have already downloaded from **Talend Exchange**.

In the **Downloaded Extensions** view, the components you have installed in your previous version of *Talend Studio* but not in your current Studio have an **Install** link in the **Install/Update** column, and those with updates available in **Talend Exchange** have an **Update** link.

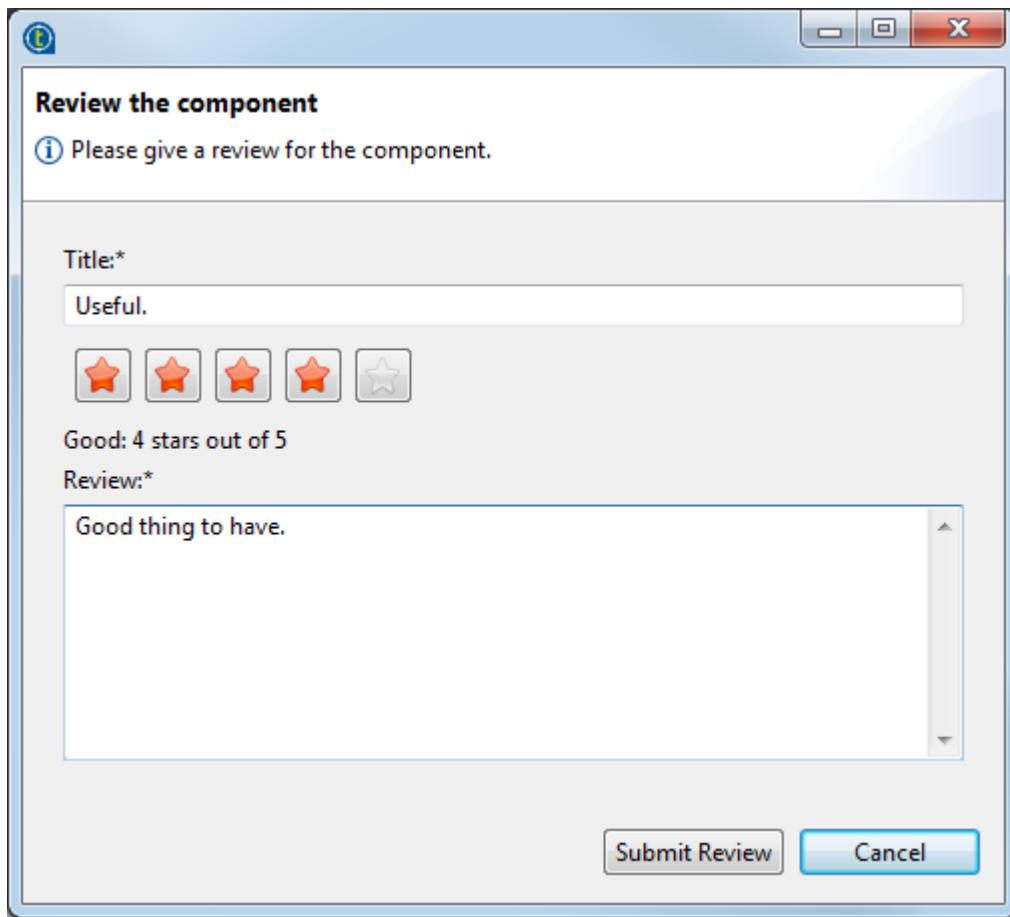
2. Click the **Install** or **Update** link for the component of interest to start the installation process.

A progress indicator appears to show the completion percentage of the installation process. Upon successful installation, the **Downloaded Extensions** view displays the status of the component, which is **Installed**.

### 4.7.3.3. How to review and rate a community component

To review and rate a community component:

1. From the **Available Extensions** view, click the **view/download** link for the component you want to review or rate to open the community component download page.
2. On the component download page, click the **write a review** link to open the **[Review the component]** dialog box.



- Fill in the required information, including a title and a review comment, click one of the five stars to rate the component, and click **Submit Review** to submit your review to the **Talend Exchange** server.

Upon validation by the **Talend Exchange** moderator, your review is published on **Talend Exchange** and displayed in the **User Review** area of the component download page.

#### 4.7.3.4. How to upload a component you created to Talend Exchange

You can create your own components for use in your Jobs in *Talend Studio* and upload them to **Talend Exchange** to share with other **Talend** Community users. For information on how to create your own components and deploy them in *Talend Studio*, see [How to define the user component folder \(Talend > Components\)](#).

To upload a component you created to **Talend Exchange**, complete the following:

- From the **Exchange** tab view, click **My Extensions** to open the **My Extensions** view.

| Available Extensions | Downloaded Extensions | Add New Extension |         |             |           |
|----------------------|-----------------------|-------------------|---------|-------------|-----------|
| My Extensions        |                       | Extension Name    | Version | Upload Date | Operation |
|                      |                       |                   |         |             |           |

- Click the **Add New Extension** link in the upper right part of the view to open the component upload page.

**Add New Extension**

Extension Title: tExEvaluate

Initial Version: 1.0

Compatibility:

- All versions
- Version and older: [ ]
- Versions and newer: 5.0
- All versions except: [ ]
- Only these versions: [ ]

Description: tExEvaluate evaluates the execution of a Job.

File: C:\Work\Components\tExEvaluate.zip

**Add Extension**

3. Complete the required information, including the component title, initial version, Studio compatibility information, and component description, fill in or browse to the path to the source package in the **File** field, and click the **Upload Extension** button.

Upon successful upload, the component is listed in the **My Extensions** view, where you can update, modify and delete any component you have uploaded to **Talend Exchange**.

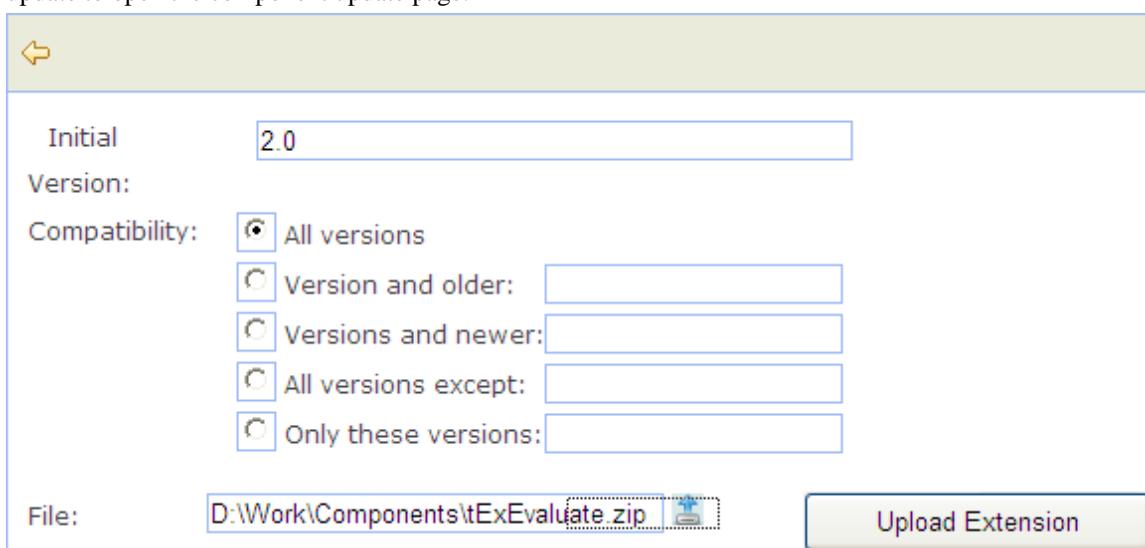
| Available Extensions  | Add New Extension |         |             |           |
|-----------------------|-------------------|---------|-------------|-----------|
| Downloaded Extensions |                   |         |             |           |
| My Extensions         | Extension Name    | Version | Upload Date | Operation |
|                       | tExEvaluate       | 1.0     | 2011-11-18  |           |

#### 4.7.3.5. How to manage components you uploaded to Talend Exchange

From the **Exchange** tab view, you can manage components you have uploaded to **Talend Exchange**, including updating component version, modifying component information, and deleting components from **Talend Exchange**.

To update the version of a component, complete the following:

- From the **My Extensions** view, click the  icon in the **Operation** column for the component you want to update to open the component update page.



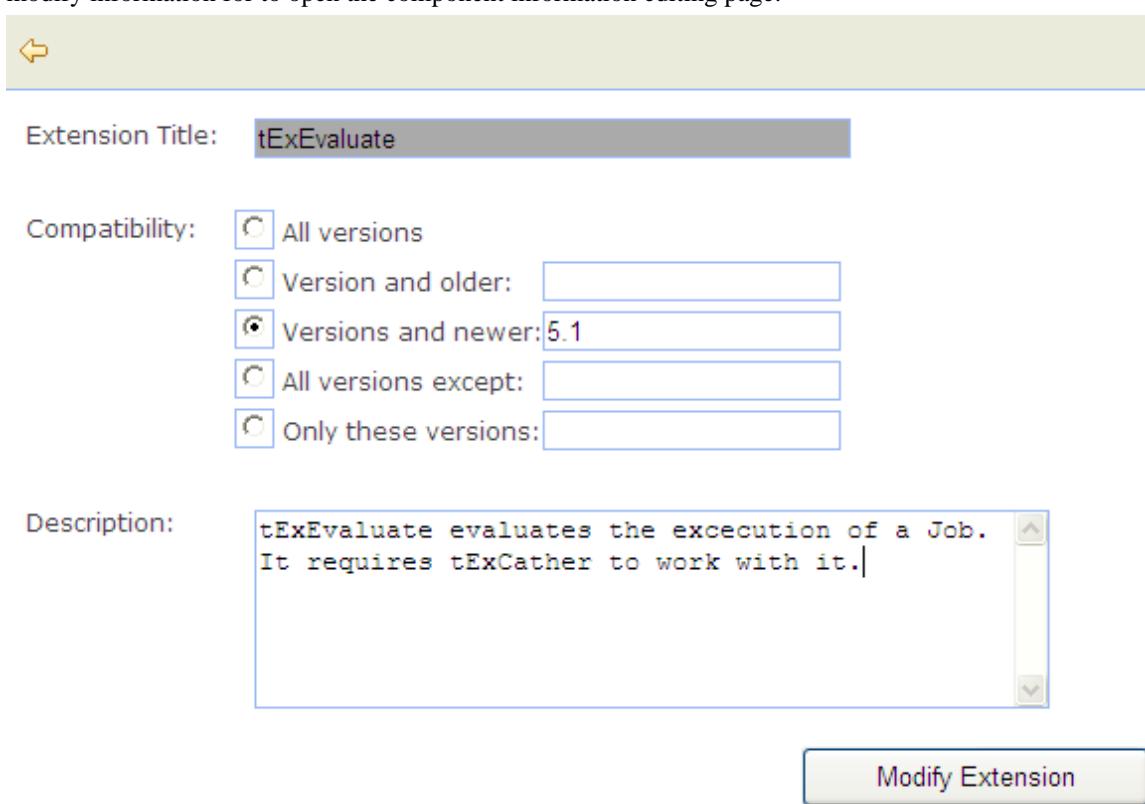
The screenshot shows a component update page. At the top, there's a back arrow icon. Below it, the 'Initial' version is set to '2.0'. Under 'Version:', the 'Compatibility:' section contains several radio button options: 'All versions' (selected), 'Version and older:', 'Versions and newer:', 'All versions except:', and 'Only these versions:'. Below this, the 'File:' field contains the path 'D:\Work\Components\tExEvaluate.zip' with a browse icon. To the right is a blue 'Upload Extension' button.

- Fill in the initial version and Studio compatibility information, fill in or browse to the path to the source package in the **File** field, and click the **Update Extension** button.

Upon successful upload of the updated component, the component is replaced with the new version on **Talend Exchange** and the **My Extension** view displays the component's new version and update date.

To modify the information of a component uploaded to **Talend Exchange**, complete the following:

- From the **My Extensions** view, click the  icon in the **Operation** column for the component you want to modify information for to open the component information editing page.



The screenshot shows a component information editing page. At the top, there's a back arrow icon. Below it, the 'Extension Title:' field is filled with 'tExEvaluate'. Under 'Compatibility:', the 'Versions and newer:' option is selected, with the value '5.1'. The other options are: 'All versions', 'Version and older:', 'All versions except:', and 'Only these versions:'. Below this, the 'Description:' field contains the text 'tExEvaluate evaluates the execution of a Job. It requires tExCather to work with it.' with scroll bars on the right. To the right is a blue 'Modify Extension' button.

2. Complete the Studio compatibility information and component description, and click the **Modify Extension** button to update the component information to **Talend Exchange**.

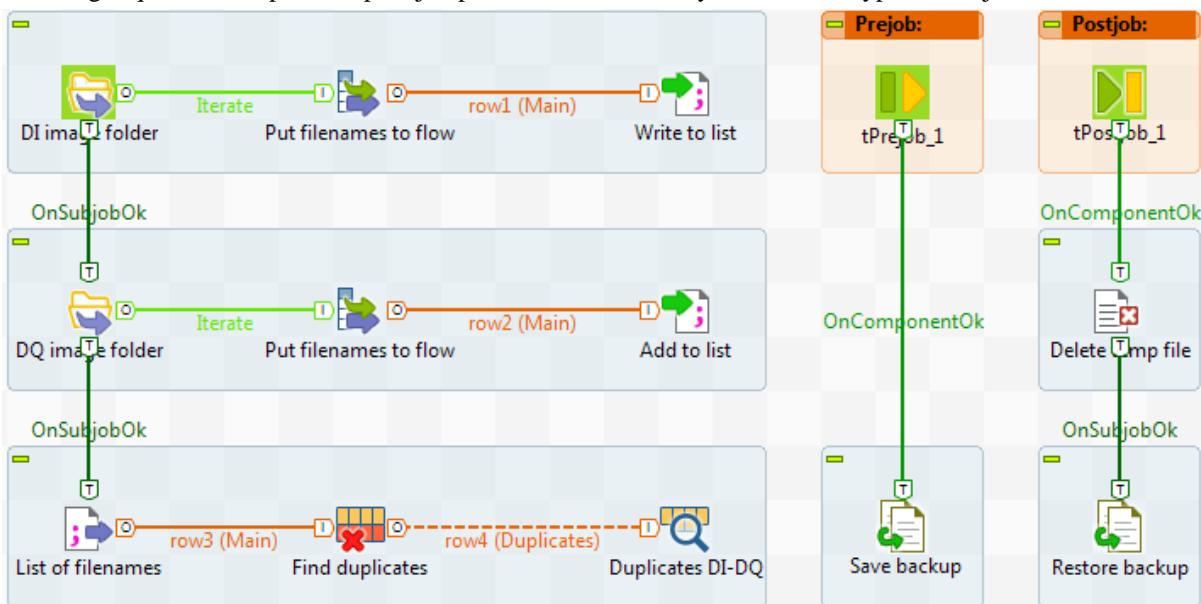
To delete a component you have uploaded to **Talend Exchange**, click  icon for the component from the **My Extensions** view. The component is then removed from **Talend Exchange** and is no longer displayed on the component list in the **My Extensions** view.

## 4.7.4. How to use the tPrejob and tPostjob components

The **tPrejob** and **tPostjob** components are designed to make the execution of tasks before and after a given job easier to manage. These components differ from other components in that they do not actually process data and they do not have any components properties to be configured. A key feature of these components is that they are always guaranteed to be executed, even if the main data Job fails. Therefore, they are very useful for setup and teardown actions for a given Job.

 As **tPrejob** and **tPostjob** are not meant to take part in any data processing, they cannot be part of a multi-thread execution. They are meant to help you make your Job design clearer.

To use these **tPrejob** and **tPostjob** components, simply drop them onto the design workspace as you would do with any other components, and then connect **tPrejob** to a component or subjob that is meant to perform a pre-job task, and **tPostjob** to a component or subjob that is meant to perform a post-job task, using **Trigger** connections. An orange square on the pre- and post-job parts indicates that they are different types of subjobs.



Tasks that require the use of a **tPrejob** component include:

- Loading context information required for the subjob execution.
- Opening a database connection.
- Making sure that a file exists.

Tasks that require the use of a **tPostjob** component include:

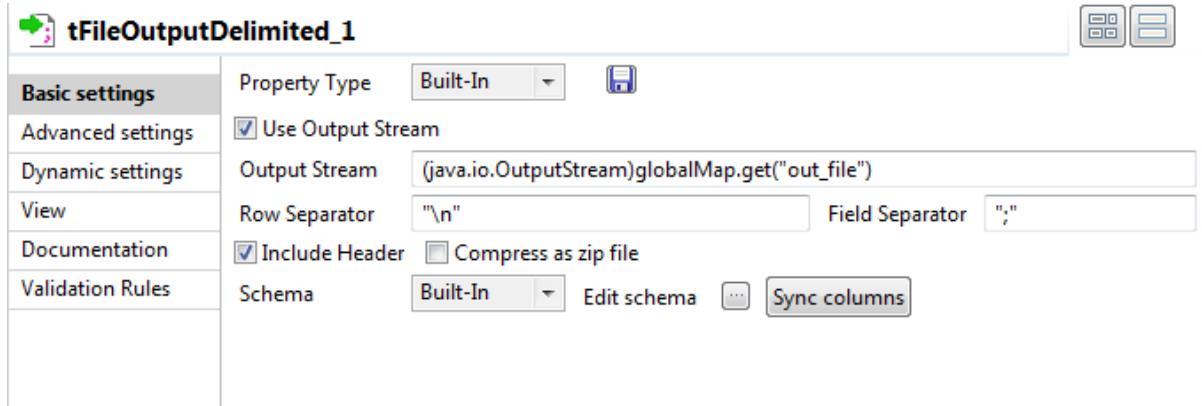
- Cleaning up temporary files created during the processing of the main data Job.
- Closing a database connection or a connection to an external service.

- Any task required to be executed, even if the preceding Job or subjobs failed.

For a use case that uses the **tPrejob** and **tPostjob** components, see **tPrejob** at <https://help.talend.com>.

## 4.7.5. How to use the Use Output Stream feature

The **Use Output Stream** feature allows you to process the data in byte-arrays using a `java.io.outputstream()` class which writes data using binary stream without data buffering. When processing data with a linear format, for example, when all data is of *String* format, this feature will help you improve the overall output performance.



The **Use Output Stream** feature can be found in the **Basic settings** view of a number of components such as **tFileOutputDelimited**.

To use this feature, select **Use Output Stream** check box in the **Basic settings** view of a component that has this feature. In the **Output Stream** field that is thus enabled, define your output stream using a command.



Prior to using the output stream feature, you have to open a stream. For a detailed example of the illustration of this prerequisite and the usage of the **Use Output Stream** feature, see [Using the output stream feature](#).

## 4.8. Handling Jobs: miscellaneous subjects

The sections below give detail information about various subjects related to the management of a data integration Job including:

- [How to use folders](#)
- [How to share a database connection](#)
- [How to add notes to a Job design](#)
- [How to display the code or the outline of your Job](#)
- [How to manage the subjob display](#)
- [How to define options on the Job view](#)

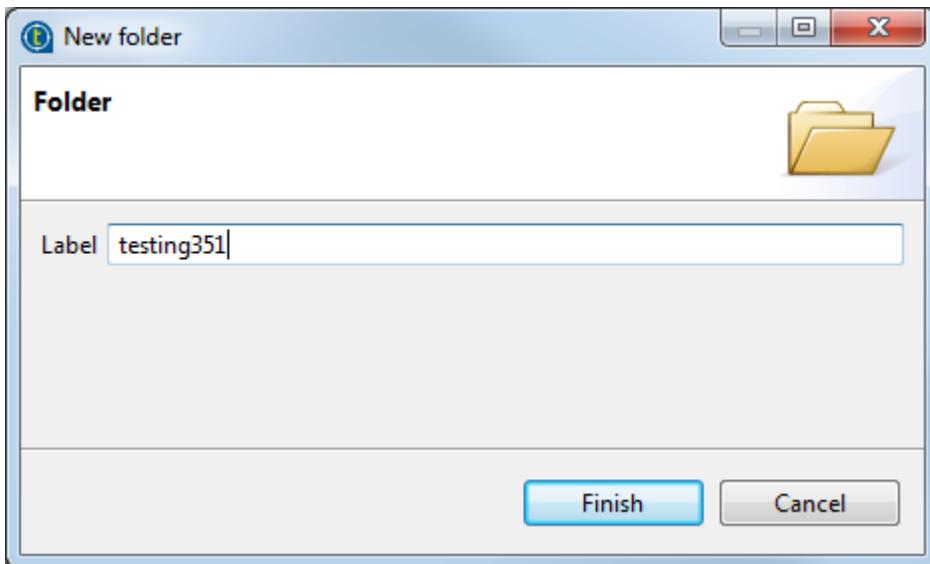
### 4.8.1. How to use folders

You can organize your Jobs into folders.

To create a folder, proceed as follows:

1. In the **Repository** tree view of the **Integration** perspective, right-click **Job Designs** and select **Create folder** from the contextual menu.

The **[New folder]** dialog box displays.



2. In the **Label** field, enter a name for the folder and then click **Finish** to confirm your changes and close the dialog box.

The created folder is listed under the **Job Designs** node in the **Repository** tree view.



If you have already created Jobs that you want to move into this new folder, simply drop them into the folder.

## 4.8.2. How to share a database connection

If you have various Jobs using the same database connection, you can factorize the connection by using the **Use or register a shared DB Connection** option so that the connection can be shared between parent and child Jobs.

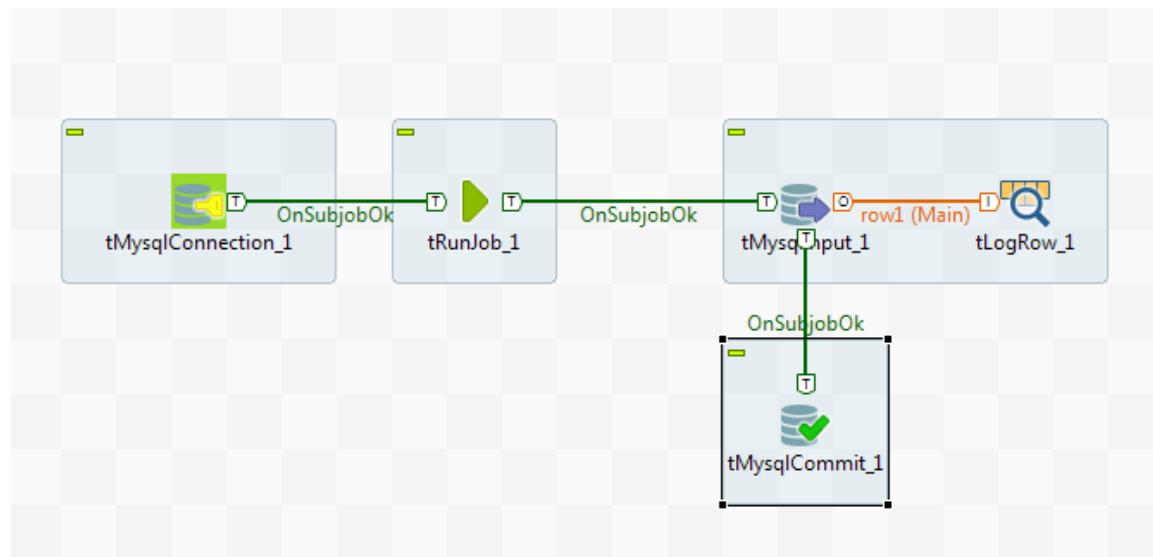
This option has been added to all database connection components in order to reduce the number of connections to open and close.



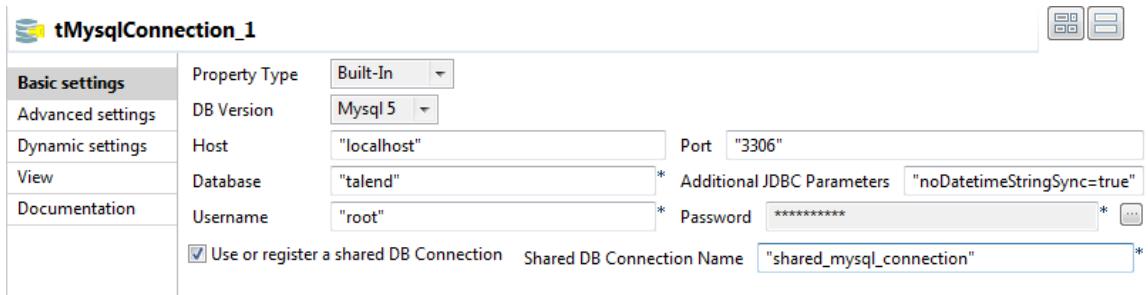
*The Use or register a shared DB Connection option of all database connection components is incompatible with the Use dynamic job and Use an independent process to run subjob options of the tRunJob component. Using a shared database connection together with a tRunJob component with either of these two options enabled will cause your Job to fail.*

Assume that you have two related Jobs (a parent Job and a child Job) that both need to connect to your remote MySQL database. To use a shared database connection in the two Jobs, to the following:

1. Add a **tMysqlConnection** (assuming that you work with a MySQL database) to both the parent and the child Job, if they are not using a database connection component.
2. Connect each **tMysqlConnection** to the relevant component in your Jobs using a **Trigger > On Subjob Ok** link.



3. In the **Basic settings** view of the **tMysqlConnection** component that will run first, fill in the database connection details if the database connection is not centrally stored in the **Repository**.
4. Select the **Use or register a shared DB Connection** check box, and give a name to the connection in the **Shared DB Connection Name** field.



You are now able to re-use the connection in your child Job.

5. In the **Basic settings** view of the other **tMysqlConnection** component, which is in the other Job, simply select **Use or register a shared DB Connection** check box, and fill the **Shared DB Connection Name** field with the same name as in the parent Job.



Among the different Jobs sharing the same database connection, you need to define the database connection details only in the first Job that needs to open the database connection.

For a complete use case, see the scenario of the **tMysqlConnection** component showing how to share a database connection between different Jobs at <https://help.talend.com>.

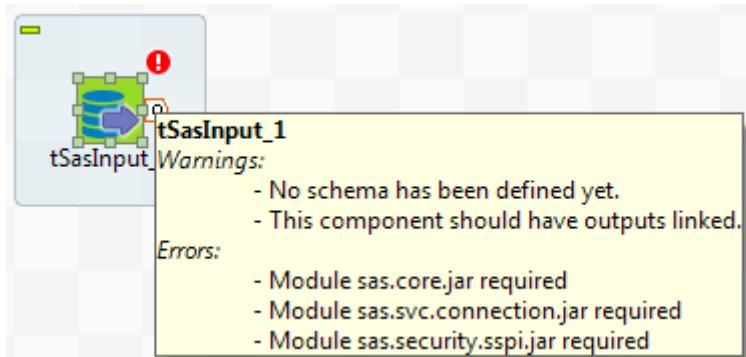
### 4.8.3. How to handle error icons on components or Jobs

When the properties of a component are not properly defined and contain one or several errors that can prevent the Job code to compile properly, error icons will automatically show next to the component icon on the design workspace and the Job name in the **Repository** tree view.

### 4.8.3.1. Warnings and error icons on components

When a component is not properly defined or if the link to the next component does not exist yet, a red checked circle or a warning sign is docked at the component icon.

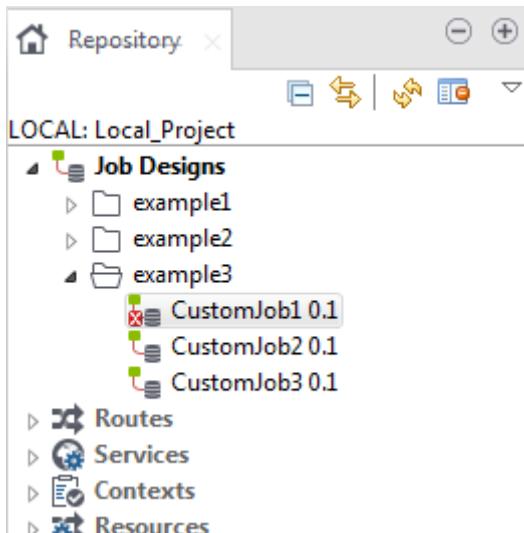
Mouse over the component, to display the tooltip messages or warnings along with the label. This context-sensitive help informs you about any missing data or component status.



 When the tooltip messages of a component indicate that a module is required, you must install this module for this component using the **Module** view. This view is hidden by default. For further information about how to install external modules using this view, see the *Talend Installation and Upgrade Guide*.

### 4.8.3.2. Error icons on Jobs

When the component settings contain one or several errors that can prevent the Job code to compile properly, an icon will automatically show next to the Job name in the **Repository** tree view.



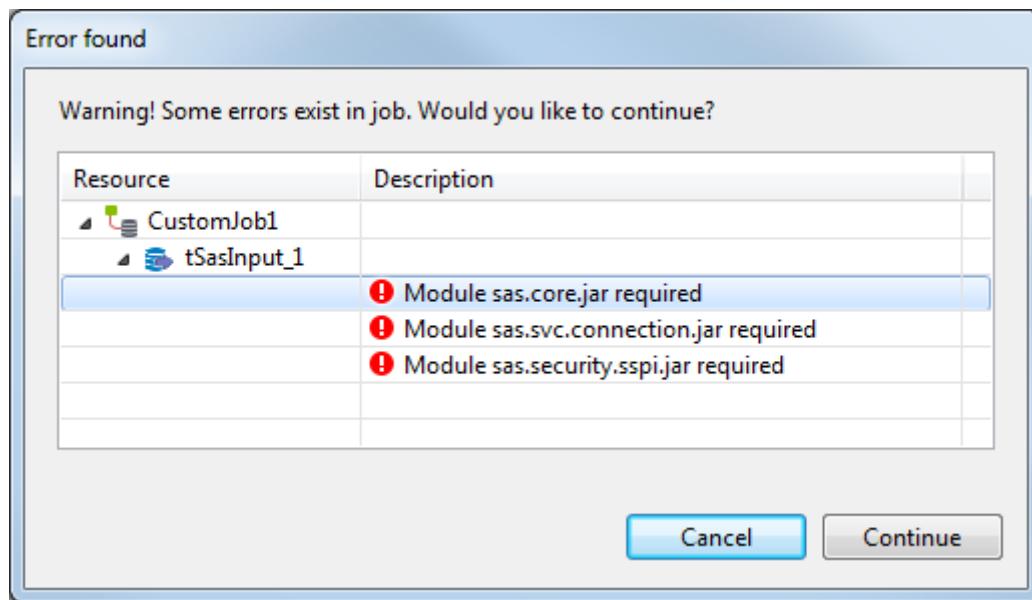
The error icon displays as well on the tab next to the Job name when you open the Job on the design workspace.

The compilation or code generation does only take place when carrying out one of the following operations:

- opening a Job,
- clicking on the **Code Viewer** tab,
- executing a Job (clicking on **Run Job**),
- saving the Job.

Hence, the red error icon will only show then.

When you execute the Job, a warning dialog box opens to list the source and description of any error in the current Job.



Click **Cancel** to stop your Job execution or click **Continue** to continue it.

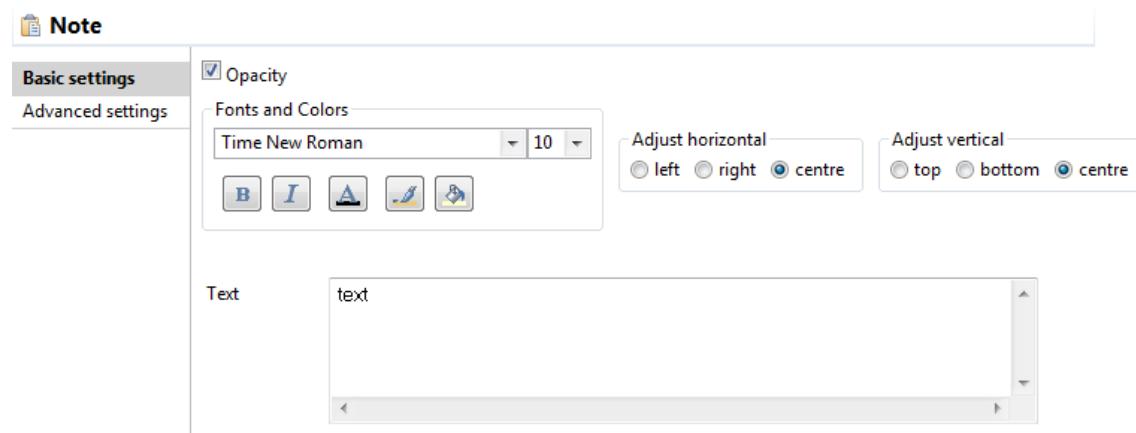
For information on errors on components, see [Warnings and error icons on components](#).

#### 4.8.4. How to add notes to a Job design

In the **Palette**, click the **Misc** family and then drop the **Note** element to the design workspace to add a text comment to a particular component or to the whole Job.



You can change the note format. To do so, select the note you want to format and click the **Basic setting** tab of the **Component** view.



Select the **Opacity** check box to display the background color. By default, this box is selected when you drop a note on the design workspace. If you clear this box, the background becomes transparent.

You can select options from the **Fonts and Colors** list to change the font style, size, color, and so on as well as the background and border color of your note.

You can select the **Adjust horizontal** and **Adjust vertical** boxes to define the vertical and horizontal alignment of the text of your note.

The content of the **Text** field is the text displayed on your note.

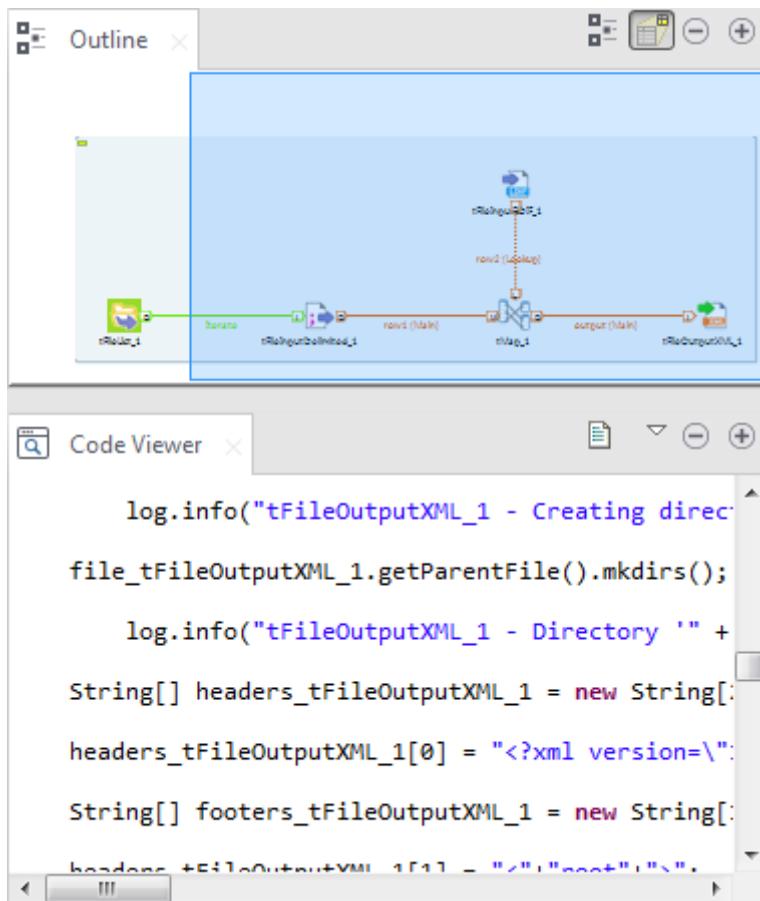
## 4.8.5. How to display the code or the outline of your Job

This panel is located below the **Repository** tree view. It displays detailed information about the open Job or Business Model in the design workspace.

The Information panel is composed of two tabs, **Outline** and **Code Viewer**, which provide information regarding the displayed diagram (either Job or Business Model).

### 4.8.5.1. Outline

The **Outline** tab offers a quick view of the business model or the open Job on the design workspace and also a tree view of all used elements in the Job or Business Model. As the design workspace, like any other window area, can be resized to suit your needs, the **Outline** view provides a convenient way for you to check out where on your design workspace you are located.



This graphical representation of the diagram highlights in a blue rectangle the diagram part showing in the design workspace.

Click the blue-highlighted view and hold down the mouse button. Then, move the rectangle over the Job.

The view in the design workspace moves accordingly.

The **Outline** view can also be displaying a folder tree view of components in use in the current diagram. Expand the node of a component, to show the list of variables available for this component.

To switch from the graphical outline view to the tree view, click either icon docked at the top right of the panel.

### 4.8.5.2. Code viewer

The **Code viewer** tab provides lines of code generated for the selected component, behind the active Job design view, as well the run menu including Start, Body and End elements.

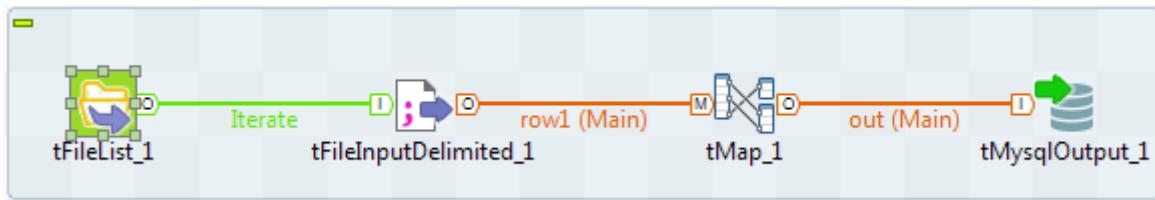


This view only concerns the Job design code, as no code is generated from Business Models.

Using a graphical colored code view, the tab shows the code of the component selected in the design workspace. This is a partial view of the primary Code tab docked at the bottom of the design workspace, which shows the code generated for the whole Job.

### 4.8.6. How to manage the subjob display

A subjob is graphically defined by a blue square gathering all connected components that belong to this subjob. Each individual component can be considered as a subjob when they are not yet connected to one another.



This blue highlight helps you easily distinguish one subjob from another.

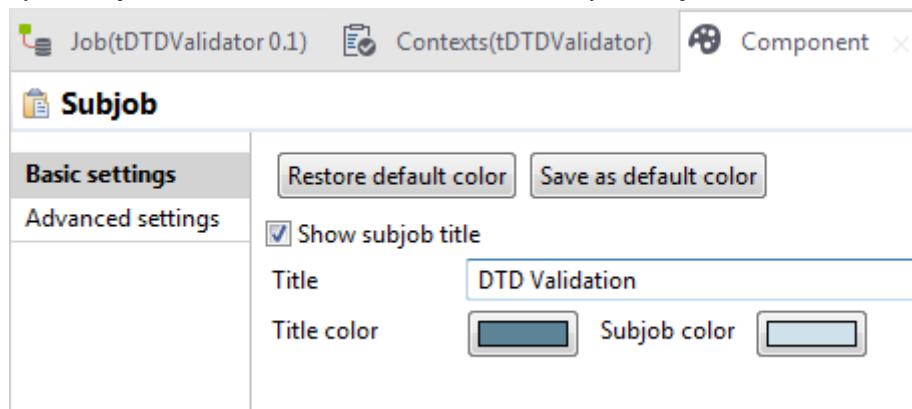


A Job can be made of one single subjob. An orange square shows the prejob and postjob parts which are different types of subjobs.

For more information about prejob and postjob, see [How to use the tPrejob and tPostjob components](#).

#### 4.8.6.1. How to format subjobs

You can modify the subjob color and its title color. To do so, select your subjob and click the **Component** view.



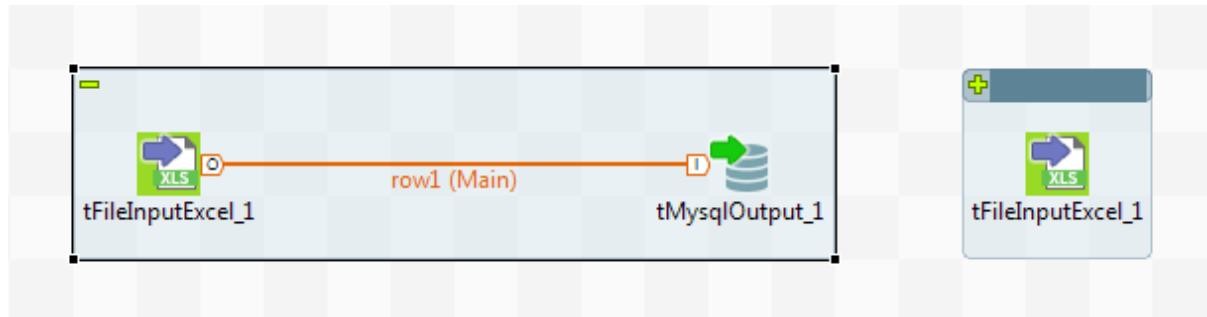
In the **Basic setting** view, select the **Show subjob title** check box if you want to add a title to your subjob, then fill in a title.

To modify the title color and the subjob color:

1. In the **Basic settings** view, click the **Title color/Subjob color** button to display the **[Colors]** dialog box.
2. Set your colors as desired. By default, the title color is blue and the subjob color is transparent blue.

#### 4.8.6.2. How to collapse the subjobs

If your Job is made of numerous subjobs, you can collapse them to improve the readability of the whole Job. The minus (**[ - ]**) and plus (**[ + ]**) signs on the top right-hand corner of the subjob allow you to collapse and restore the complete subjob.



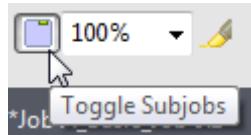
Click the minus sign (**[ - ]**) to collapse the subjob. When reduced, only the first component of the subjob is displayed.

Click the plus sign (**[ + ]**) to restore your subjob.

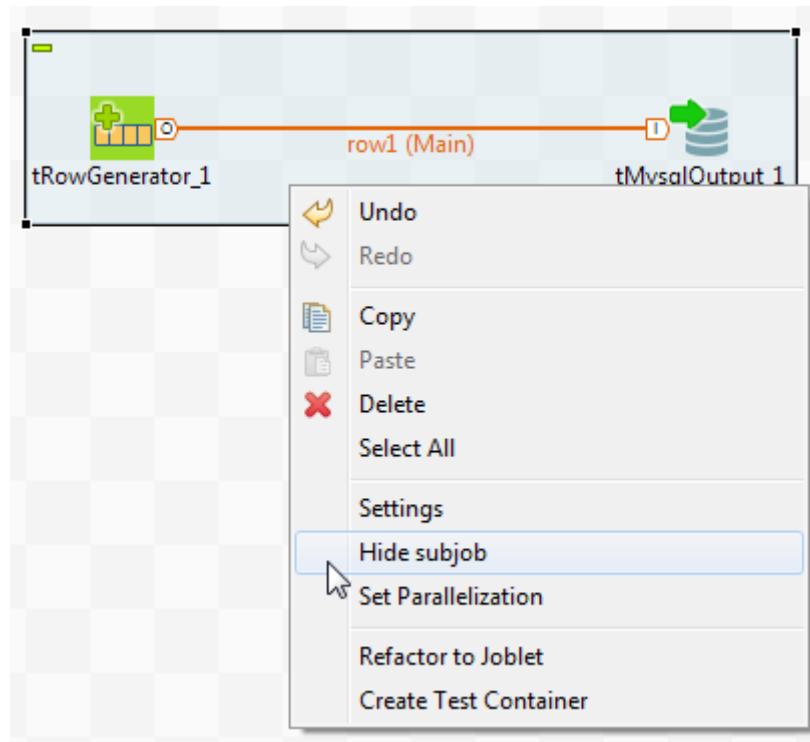
#### 4.8.6.3. How to remove the subjob background color

If you do not want your subjobs to be highlighted, you can remove the background color on all or specific subjobs.

To remove the background color of all your subjobs, click the **Toggle Subjobs** icon on the toolbar of *Talend Studio*.



To remove the background color of a specific subjob, right-click the subjob and select the **Hide subjob** option on the pop-up menu.



## 4.8.7. How to define options on the Job view

On the **Job** view located on the bottom part of the design workspace, you can define Job's optional functions. This view is made of two tabs: **Stats & Logs** tab and **Extra** tab.

The **Stats & Logs** tab allows you to automate the use of **Stats & Logs** features and the Context loading feature. For more information, see [How to automate the use of statistics & logs](#).

The **Extra** tab lists various options you can set to automate some features such as the context parameters use, in the **Implicit Context Loading** area. For more information, see [How to use the features in the Extra tab](#).

### 4.8.7.1. How to automate the use of statistics & logs

If you have a great need of log, statistics and other measurement of your data flows, you are facing the issue of having too many log-related components loading your Job Designs. You can automate the use of **tFlowMeterCatcher**, **tStatCatcher**, **tLogCatcher** component functionalities without using the components in your Job via the **Stats & Logs** tab.

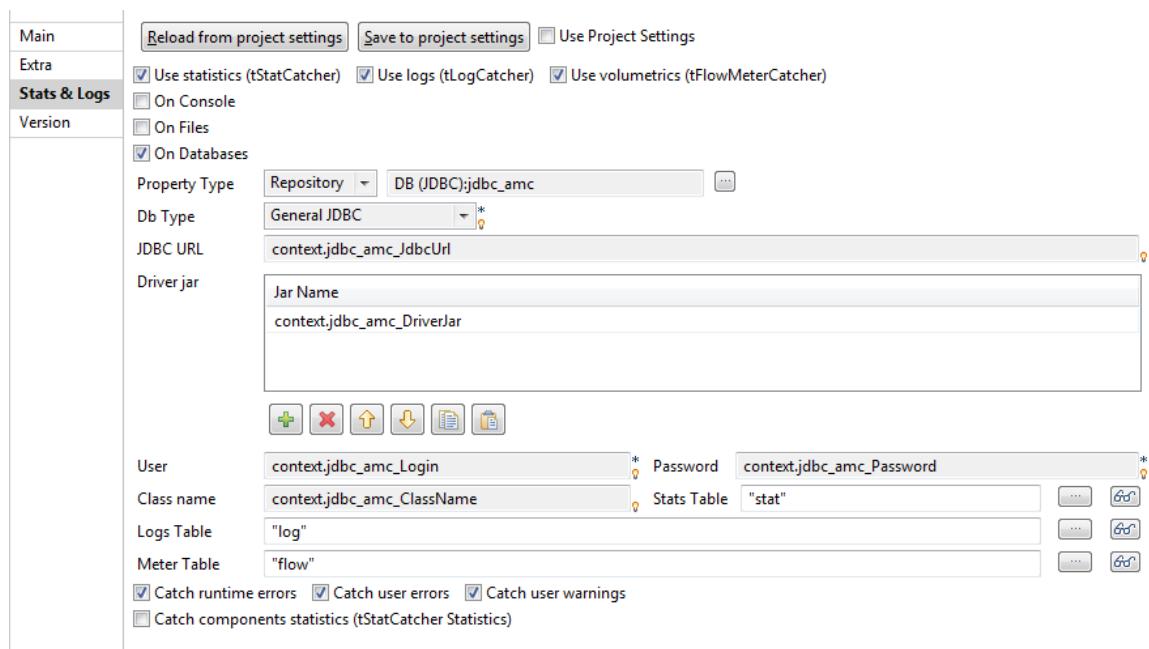
The **Stats & Logs** panel is located on the **Job** tab underneath the design workspace and prevents your Jobs Designs to be overloaded by components.



This setting supersedes the log-related components with a general log configuration.

To set the **Stats & Logs** properties:

1. Click the **Job** tab.
2. Select the **Stats & Logs** panel to display the configuration view.

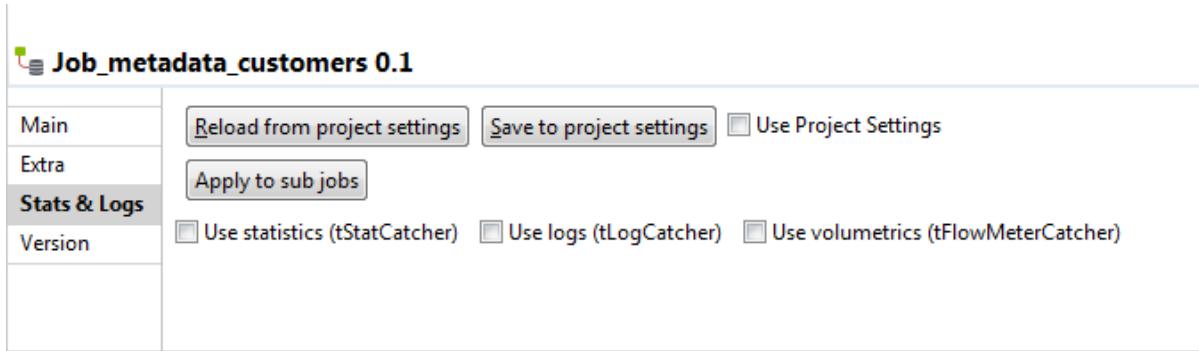


3. Set the relevant details depending on the output you prefer (console, file or database).
4. Select the relevant **Catch** check box according to your needs.



You can save the settings into your Project Settings by clicking the **Save to project settings** button. This way, you can access such settings via **File > Edit project settings > Job settings > Stats & Logs** or via the button on the toolbar.

When you use **Stats & Logs** functions in your Job, you can apply them to all its subjobs.



To do so, click the **Apply to subjobs** button in the **Stats & Logs** panel of the **Job** view and the selected stats & logs functions of the main Job will be selected for all of its subjobs.

#### 4.8.7.2. How to use the features in the Extra tab

The **Extra** tab offers some optional function parameters.

- Select the **Multithread execution** check box to allow two Job executions to start at the same time.
- Set the **Implicit tContextLoad** option parameters to avoid using the **tContextLoad** component on your Job and automate the use of context parameters.

Choose between **File** and **Database** as source of your context parameters and set manually the file or database access.

Set notifications (error/warning/info) for unexpected behaviors linked to context parameter setting.

For an example of loading context parameters dynamically using the Implicit Context Load feature, see [Using the Implicit Context Load feature](#).

- When you fill in **Implicit tContextLoad** manually, you can store these parameters in your project by clicking the **Save to project settings** button, and thus reuse these parameters for other components in different Jobs.
- Select the **Use Project Settings** check box to recuperate the context parameters you have already defined in the **Project Settings** view.

The **Implicit tContextLoad** option becomes available and all fields are filled in automatically.

For more information about context parameters, see [Context settings](#).

- Click **Reload from project settings** to update the context parameters list with the latest context parameters from the project settings.





## Chapter 5. Managing Jobs

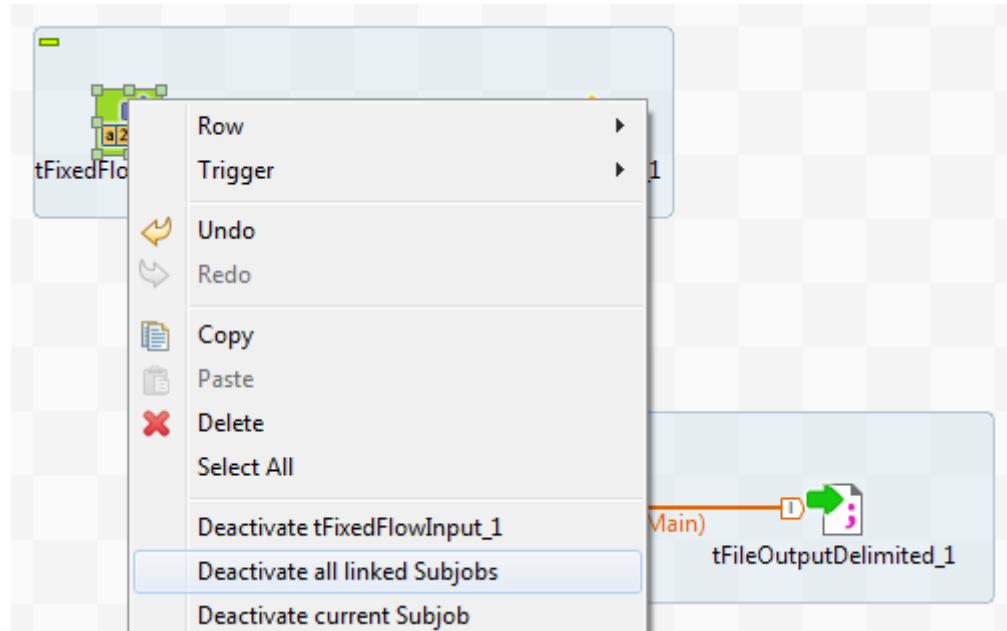
This chapter describes the management procedures you can carry out on the Jobs you design in *Talend Studio* or you can carry out on any of the items included in a project, for example routines or metadata.

These management procedures include importing and exporting Jobs and items between different projects or machines, scheduling Job execution, etc.

# 5.1. Activating/Deactivating a component or a subjob

You can activate or deactivate a subjob directly connected to the selected component. You can also activate or deactivate a single component as well as all the subjobs linked to a Start component. The Start component is the trigger of the Job. It has a green background. For more information about Start components, see [How to define the start component](#).

When a component or a subjob is deactivated, you are not able to create or modify links from or to it. Moreover, at runtime, no code is generated for the deactivated component or subjob.



## 5.1.1. Activate or deactivate a component

To activate or deactivate a component, proceed as follows:

1. Right-click the component you want to activate or deactivate, the **tFixedFlowInput** component for example.
2. Select the option corresponding to the action you want to perform:
  - **Activate tFixedFlowInput\_1** if you want to activate it.
  - **Deactivate tFixedFlowInput\_1** if you want to deactivate it.

## 5.1.2. Activate or deactivate a subjob

To activate or deactivate a subjob, proceed as follows:

1. Right-click any component composing the subjob.
2. Select the option corresponding to the action you want to perform:

- **Activate current Subjob** if you want to activate it.
- **Deactivate current Subjob** if you want to deactivate it.

### 5.1.3. Activate or deactivate all linked subjobs

To activate or deactivate all linked subjobs, proceed as follows:

1. Right-click the Start component.
2. Select the option corresponding to the action you want to perform:
  - **Activate all linked Subjobs** if you want to activate them.
  - **Deactivate all linked Subjobs** if you want to deactivate them.

## 5.2. Importing/exporting items and building Jobs

*Talend Studio* enables you to import/export your Jobs or items in your Jobs from/to various projects or various versions of the Studio. It enables you as well to build Jobs and thus deploy and execute those created in the Studio on any server.

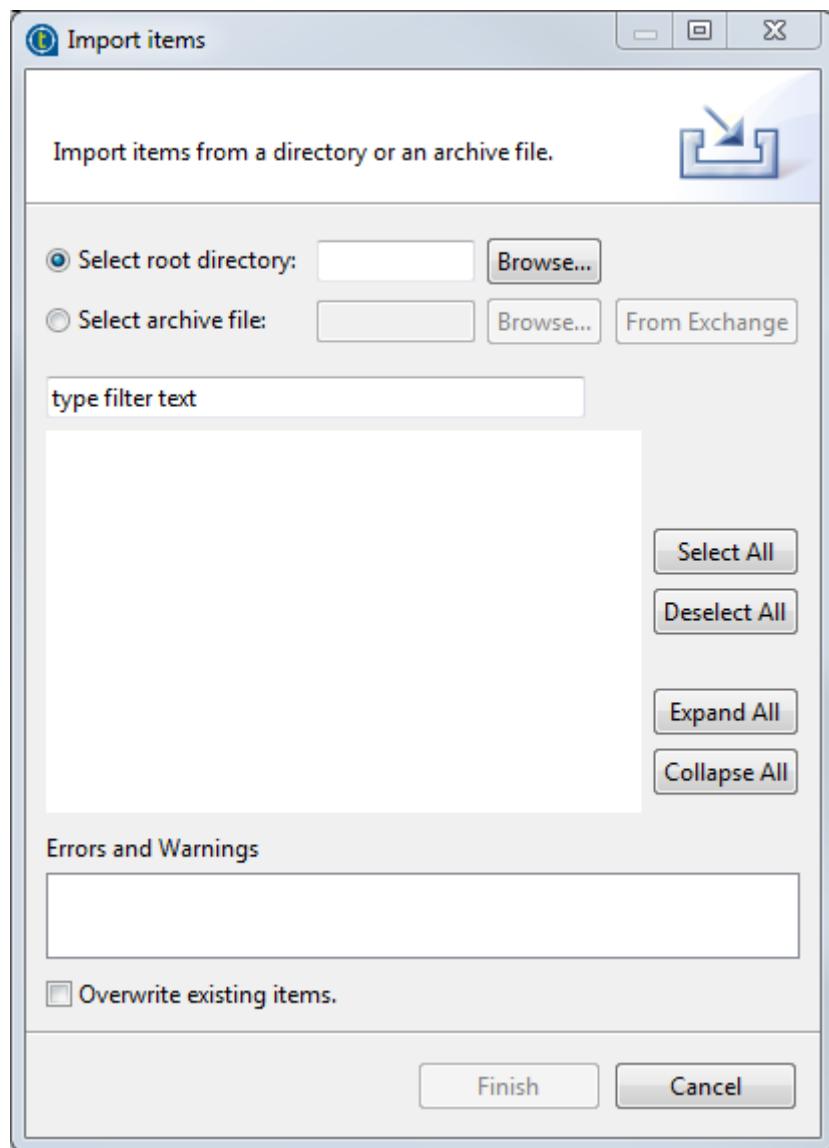
### 5.2.1. How to import items

You can import items from previous versions of *Talend Studio* or from a different project of your current version.

The items you can possibly import are multiple:

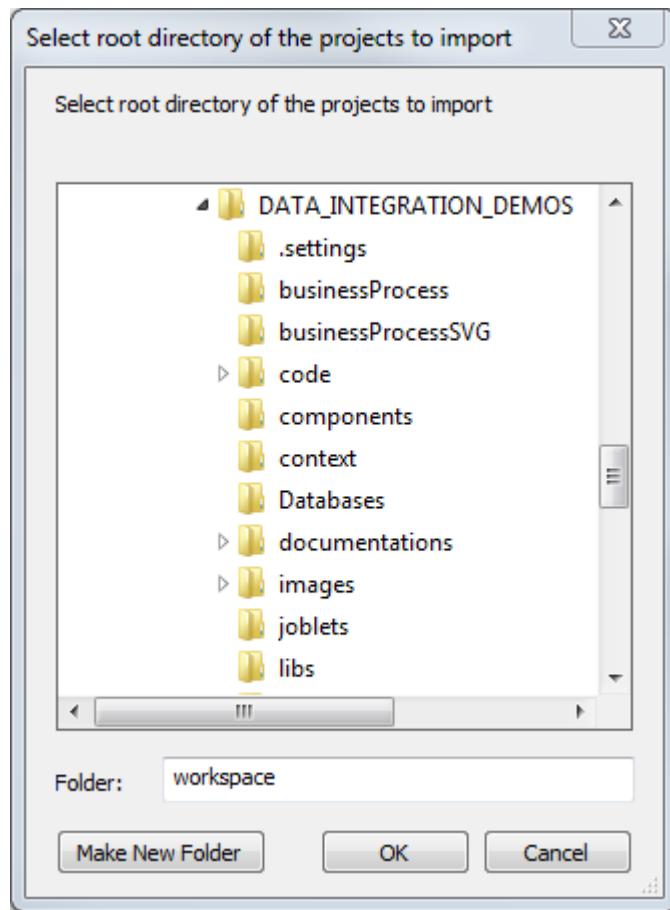
- Business Models
- Jobs Designs
- Routines
- Documentation
- Metadata

To import items, right-click any entry such as **Job Designs** or **Business Models** in the **Repository** tree view and select **Import Items** from the contextual menu or directly click the  icon on the toolbar to open the **[Import items]** dialog box and then select an import option.



To import items stored in a local directory, do the following:

1. Click the **Select root directory** option in the **[Import items]** dialog box.
2. Click **Browse** to browse down to the relevant project folder within the workspace directory. It should correspond to the project name you picked up.



3. If you only want to import very specific items such as some **Job Designs**, you can select the specific folder, such as Process where all the Job Designs for the project are stored. If you only have **Business Models** to import, select the specific folder: **BusinessProcess**, and click **OK**.

But if your project gathers various types of items (Business Models, Jobs Designs, Metadata, Routines...), we recommend you to select the project folder to import all items in one go, and click **OK**.

4. If needed, select the **overwrite existing items** check box to overwrite existing items with those having the same names to be imported. This will refresh the **Items List**.
5. From the **Items List** which displays all valid items that can be imported, select the items that you want to import by selecting the corresponding check boxes.
6. Click **Finish** to validate the import.

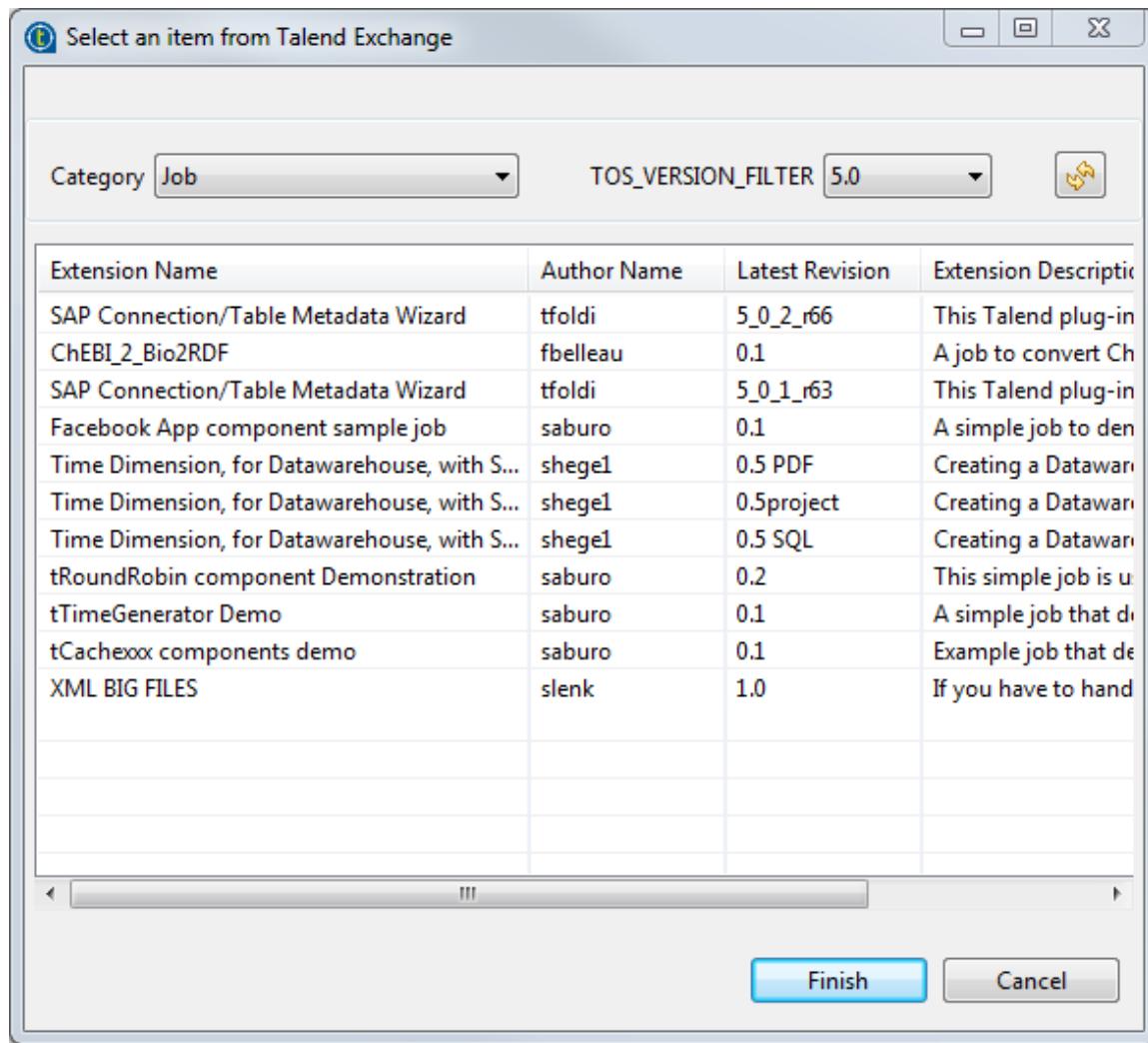
To import items from an archive file (including source files and scripts), do the following:

1. Click the **Select archive file** option in the **[Import items]** dialog box.
2. Browse to the desired archive file and click **Open**.
3. If needed, select the **overwrite existing items** check box to overwrite existing items with those having the same names to be imported. This will refresh the **Items List**.
4. From the **Items List** which displays all valid items that can be imported, select the items that you want to import by selecting the corresponding check boxes.
5. Click **Finish** to validate the import.

To import items from **Talend Exchange**, do the following:

1. Click the **Select archive file** option in the [Import items] dialog box. Then, click **BrowseTalendExchange** to open the [Select an item from Talend Exchange] dialog box.
2. Select the desired category from the **Category** list, and select the desired version from the **TOS\_VERSION\_FILTER** list.

A progress bar appears to indicate that the extensions are being downloaded. At last, the extensions for the selected category and version will be shown in the dialog box.



3. Select the extension that you want to import from the list.

Click **Finish** to close the dialog box.

4. If needed, select the **overwrite existing items** check box to overwrite existing items with those having the same names to be imported. This will refresh the **Items List**.
5. From the **Items List** which displays all valid items that can be imported, select the items that you want to import by selecting the corresponding check boxes.
6. Click **Finish** to validate the import.



If there are several versions of the same items, they will all be imported into the Project you are running, unless you already have identical items.

## 5.2.2. How to build Jobs

The **Build Job** feature allows you to deploy and execute a Job on any server, independent of *Talend Studio*.

By executing build scripts generated from the templates defined in Project Settings, the **Build Job** feature adds all of the files required to execute the Job to an archive, including the *.bat* and *.sh* along with any context-parameter files or other related files.

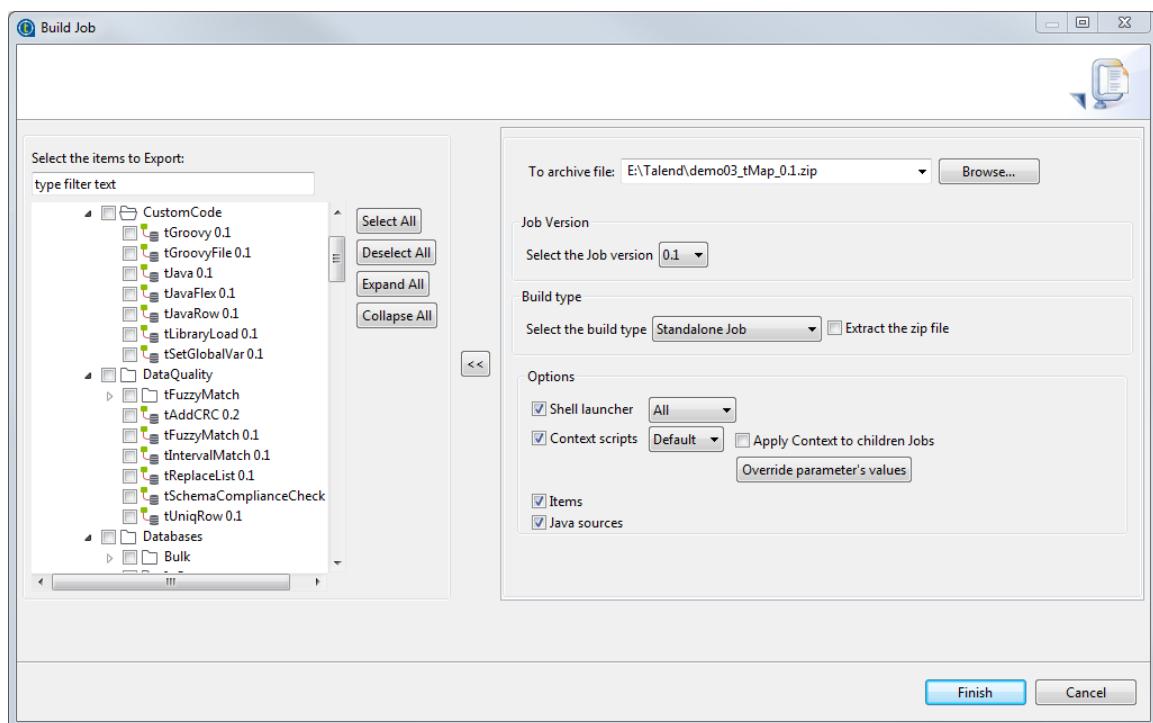
 Your *Talend Studio* provides a set of default build script templates. You can customize those templates to meet your actual needs. For more information, see [Customizing Maven build script templates](#).

By default, when a Job is built, all the required jars are included in the *.bat* or *.sh* command. For a complex Job that involves many Jars, the number of characters in the batch command may exceed the limitation of command length on certain operating systems. To avoid failure of running the batch command due to this limitation, before building your Job, go to **Window > Preferences**, select **Talend > Import/Export**, and then select the **Add classpath jar in exported jobs** check box to wrap the Jars in a *classpath.jar* file added to the built Job.

To build Jobs, complete the following:

1. In the **Repository** tree view, right-click the Job you want to build, and select **Build Job** to open the **[Build Job]** dialog box.

 You can show/hide a tree view of all created Jobs in *Talend Studio* directly from the **[Build Job]** dialog box by clicking the  and the  buttons respectively. The Jobs you earlier selected in the Studio tree view display with selected check boxes. This accessibility helps to modify the selected items to be exported directly from the dialog box without having to close it and go back to the **Repository** tree view in *Talend Studio* to do that.



2. In the **To archive file** field, browse to the directory where you want to save your built Job.
3. From the **Select the Job version** area, select the version number of the Job you want to build if you have created more than one version of the Job.
4. Select the **Build Type** from the list between **Standalone Job**, **Axis Webservice (WAR) (Deprecated)**, **Axis Webservice (ZIP) (Deprecated)** and **OSGI Bundle For ESB**.

If the data service Job includes the **tRESTClient** or **tESBConsumer** component, and none of the Service Registry, Service Locator or Service Activity Monitor is enabled in the component, the data service Job can

be built as **OSGI Bundle For ESB** or **Standalone Job**. With the Service Registry, Service Locator or Service Activity Monitor enabled, the data service Job including the **tRESTClient** or **tESBConsumer** component can only be built as **OSGI Bundle For ESB**.

5. Select the **Extract the zip file** check box if you want the archive file to be automatically extracted in the target directory.
6. In the **Options** area, select the file type(s) you want to add to the archive file. The check boxes corresponding to the file types necessary for the execution of the Job are selected by default. You can clear these check boxes depending on what you want to build.

| Option                   | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|--------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Shell launcher</b>    | Select this check box to export the <i>.bat</i> and/or <i>.sh</i> files necessary to launch the built Job. <ul style="list-style-type: none"> <li>• <b>All</b>: exports the <i>.bat</i> and <i>.sh</i> files.</li> <li>• <b>Unix</b> exports the <i>.sh</i> file.</li> <li>• <b>Windows</b> exports the <i>.bat</i> file.</li> </ul>                                                                                                                                                                                     |
| <b>Context scripts</b>   | Select this check box to export ALL context parameters files and not just those you select in the corresponding list.<br> To export only one context, select the context that fits your needs from the <b>Context scripts</b> list, including the <i>.bat</i> or <i>.sh</i> files holding the appropriate context parameters. Then you can, if you wish, edit the <i>.bat</i> and <i>.sh</i> files to manually modify the context type. |
| <b>Apply to children</b> | Select this check box if you want to apply the context selected from the list to all child Jobs.                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <b>Items</b>             | Select this check box to export the sources used by the Job during its execution including the <i>.item</i> and <i>.properties</i> files, Java and <b>Talend</b> sources.<br> If you select the <b>Items</b> or <b>Source files</b> check box, you can reuse the built Job in a <i>Talend Studio</i> installed on another machine. These source files are only used in <i>Talend Studio</i> .                                         |
| <b>Java sources</b>      | Select this check box to export the <i>.java</i> file holding Java classes generated by the Job when designing it.                                                                                                                                                                                                                                                                                                                                                                                                       |

7. Click the **Override parameters' values** button, if necessary.

In the window which opens you can update, add or remove context parameters and values of the Job context you selected in the list.

8. Click **Finish** to validate your changes, complete the build operation and close the dialog box.

A zipped file for the Jobs is created in the defined place.



If the Job to be built calls a user routine that contains one or more extra Java classes in parallel with the public class named the same as the user routine, the extra class or classes will not be included in the exported file. To export such classes, you need to include them within the class with the routine name as inner classes. For more information about user routines, see [Managing user routines](#). For more information about classes and inner classes, see relevant Java manuals.

### 5.2.2.1. How to build a Job as a standalone Job

In the case of a Plain Old Java Object export, if you want to reuse the Job in *Talend Studio* installed on another machine, make sure you selected the **Items** check box. These source files (*.item* and *.properties*) are only needed within *Talend Studio*.

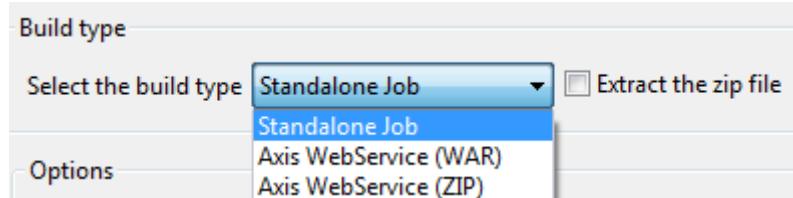
Select a context from the list when offered. Then once you click the **Override parameters' values** button below the **Context scripts** check box, the opened window will list all of the parameters of the selected context. In this window, you can configure the selected context as needs.

All contexts parameter files are exported along in addition to the one selected in the list.

-  After being exported, the context selection information is stored in the `.bat` or `.sh` file and the context settings are stored in the context `.properties` file.

## 5.2.2.2. How to build a Job as a Webservice

In the **[Build Job]** dialog box, you can change the build type in order to build the Job selection as Webservice archive.



Select the type of archive you want to use in your Web application.

| Archive type | Description                                                                                                                                                                                                        |
|--------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| WAR          | The options are read-only. Indeed, the WAR archive generated includes all configuration files necessary for the execution or deployment from the Web application.                                                  |
| ZIP          | All options are available. In the case the files of your Web application config are all set, you have the possibility to only set the Context parameters if relevant and export only the Classes into the archive. |

Once the archive is produced, place the WAR or the relevant Class from the ZIP (or unzipped files) into the relevant location, of your Web application server.

The URL to be used to deploy the Job, typically reads as follow:

```
http://localhost:8080/Webappname/services/JobName?method=runJob&args=null
```

where the parameters stand as follow:

| URL parameters           | Description                                                    |
|--------------------------|----------------------------------------------------------------|
| http://localhost:8080/   | Type in the Webapp host and port.                              |
| /Webappname/             | Type in the actual name of your web application.               |
| /services/               | Type in "services" as the standard call term for web services. |
| /JobName                 | Type in the exact name of the Job you want to execute.         |
| ?method=runJob&args=null | The method is RunJob to execute the Job.                       |

The call return from the Web application is 0 when there is no error and different from 0 in case of error. For a real-life example of creating and building a Job as a Webservice and calling the built Job from a browser, see [An example of building a Job as a Web service](#).

The **tBufferOutput** component was especially designed for this type of deployment. For more information regarding this component, see the **tBufferOutput** component documentation.

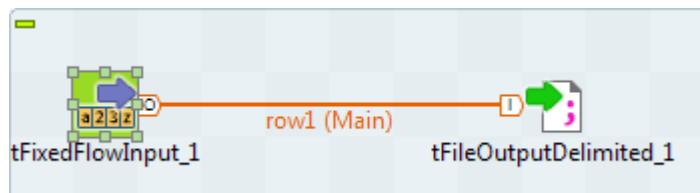
## 5.2.2.3. An example of building a Job as a Web service

This scenario describes first a simple Job that creates a `.txt` file and writes in it the current date along with first and last names. Secondly, it shows how to build this Job as a Webservice. And finally, it calls the Job built as

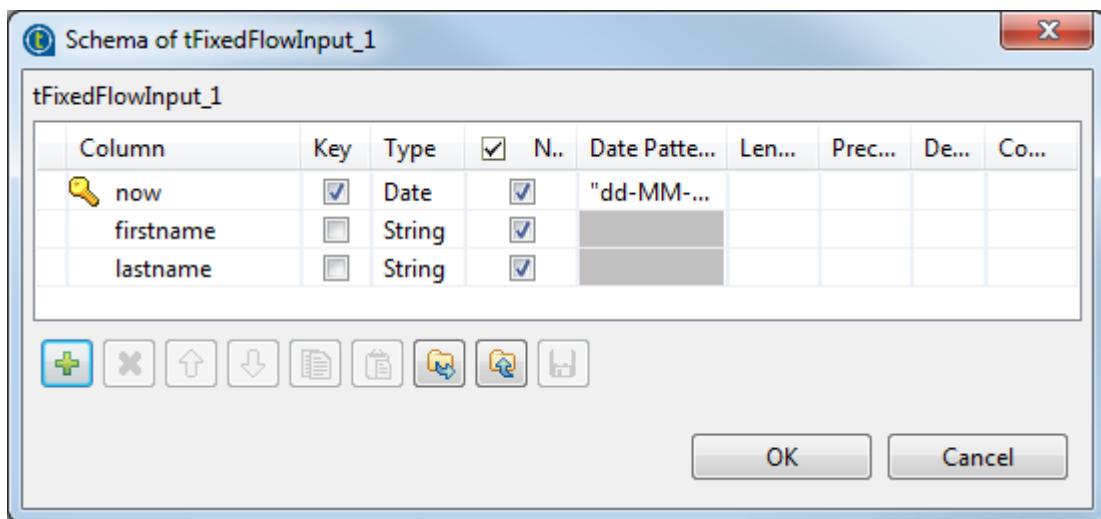
a Webservice from a browser. The built Job as a Webservice will simply return the "return code" given by the operating system.

## Creating the Job:

1. Drop the following components from the **Palette** onto the design workspace: **tFixedFlowInput** and **tFileOutputDelimited**.
2. Connect **tFixedFlowInput** to **tFileOutputDelimited** using a **Row > Main** link.



3. In the design workspace, select **tFixedFlowInput**, and click the **Component** tab to define the basic settings for **tFixedFlowInput**.
4. Set the **Schema** to **Built-In** and click the [...] button next to **Edit Schema** to describe the data structure you want to create from internal variables. In this scenario, the schema is made of three columns, *now*, *firstname*, and *lastname*.

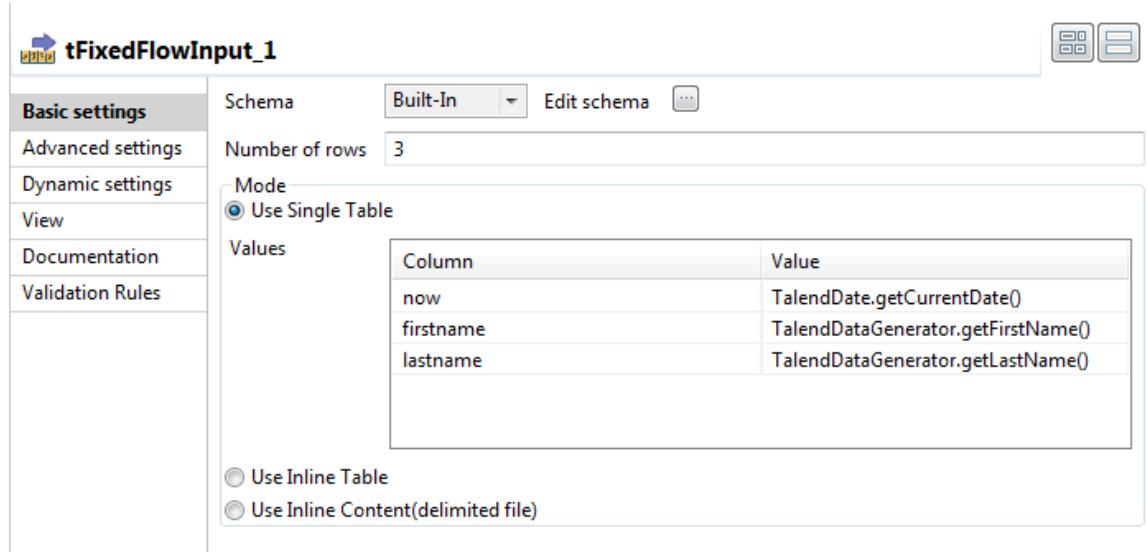


5. Click the [+] button to add the three parameter lines and define your variables, and then click **OK** to close the dialog box and accept propagating the changes when prompted by the system.

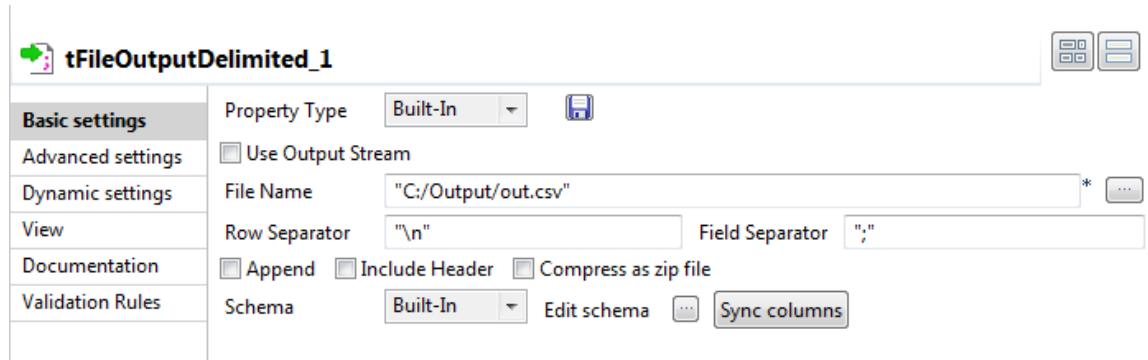
The three defined columns display in the **Values** table of the **Basic settings** view of **tFixedFlowInput**.

| Values | Column    | Value |
|--------|-----------|-------|
|        | now       |       |
|        | firstname |       |
|        | lastname  |       |

6. In the **Value** cell of each of the three defined columns, press **Ctrl+Space** to access the global variable list, and select *TalendDate.getCurrentDate()*, *talendDatagenerator.getFirstName*, and *talendDataGenerator.getLastname* for the *now*, *firstname*, and *lastname* columns respectively.
7. In the **Number of rows** field, enter the number of lines to be generated.



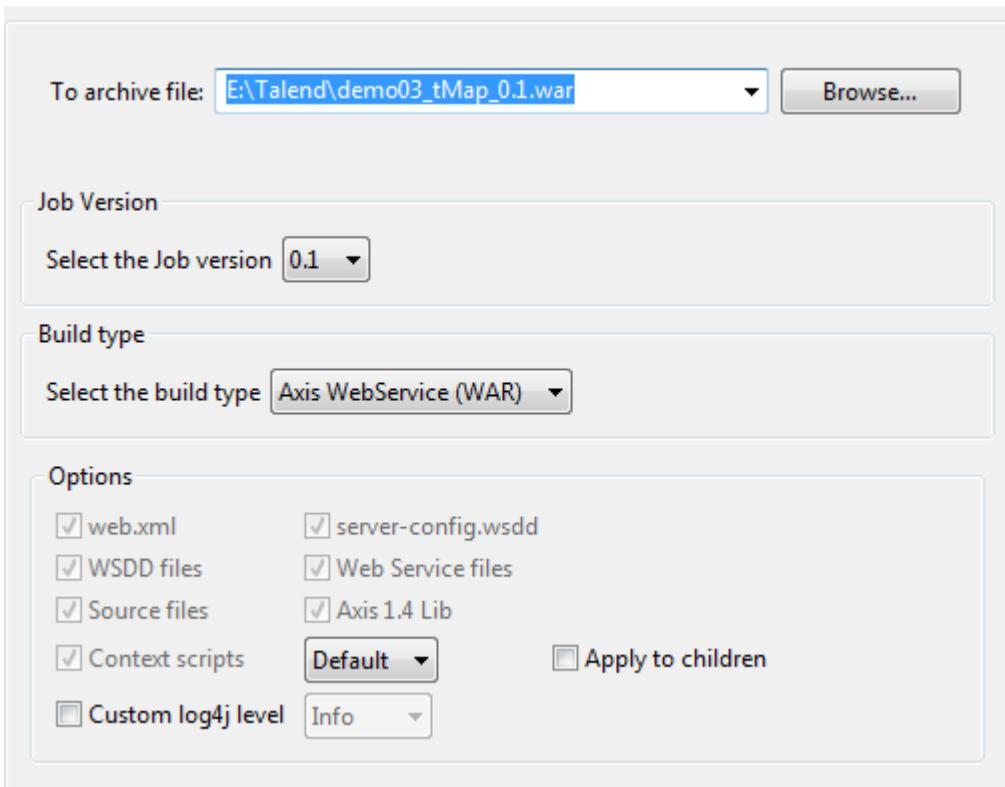
- In the design workspace, select **tFileOutputDelimited**, click the **Component** tab for **tFileOutputDelimited**, and browse to the output file to set its path in the **File name** field. Define other properties as needed.



If you press **F6** to execute the Job, three rows holding the current date and first and last names will be written to the set output file.

### Building the Job as a Webservice:

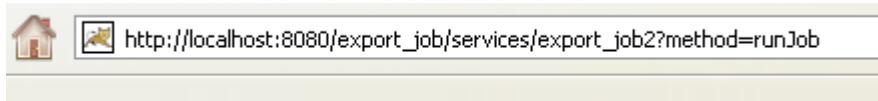
- In the **Repository** tree view, right-click the above created Job and select **Build Job**. The **[Build Job]** dialog box appears.



2. Click the **Browse...** button to select a directory to archive your Job in.
3. In the **Job Version** area, select the version of the Job you want to build as a web service.
4. In the **Build type** area, select the build type you want to use in your Web application (WAR in this example) and click **Finish**. The **[Build Job]** dialog box disappears.
5. Copy the War folder and paste it in the Tomcat webapp directory.

### Calling the Job from a browser:

1. Type the following URL into your browser: `http://localhost:8080//export_job/services/export_job2?method=runJob` where "export\_job" is the name of the webapp directory deployed in Tomcat and "export\_job2" is the name of the Job.



2. Click **Enter** to execute the Job from your browser.

Ce fichier XML ne semble pas avoir d'information de style lui

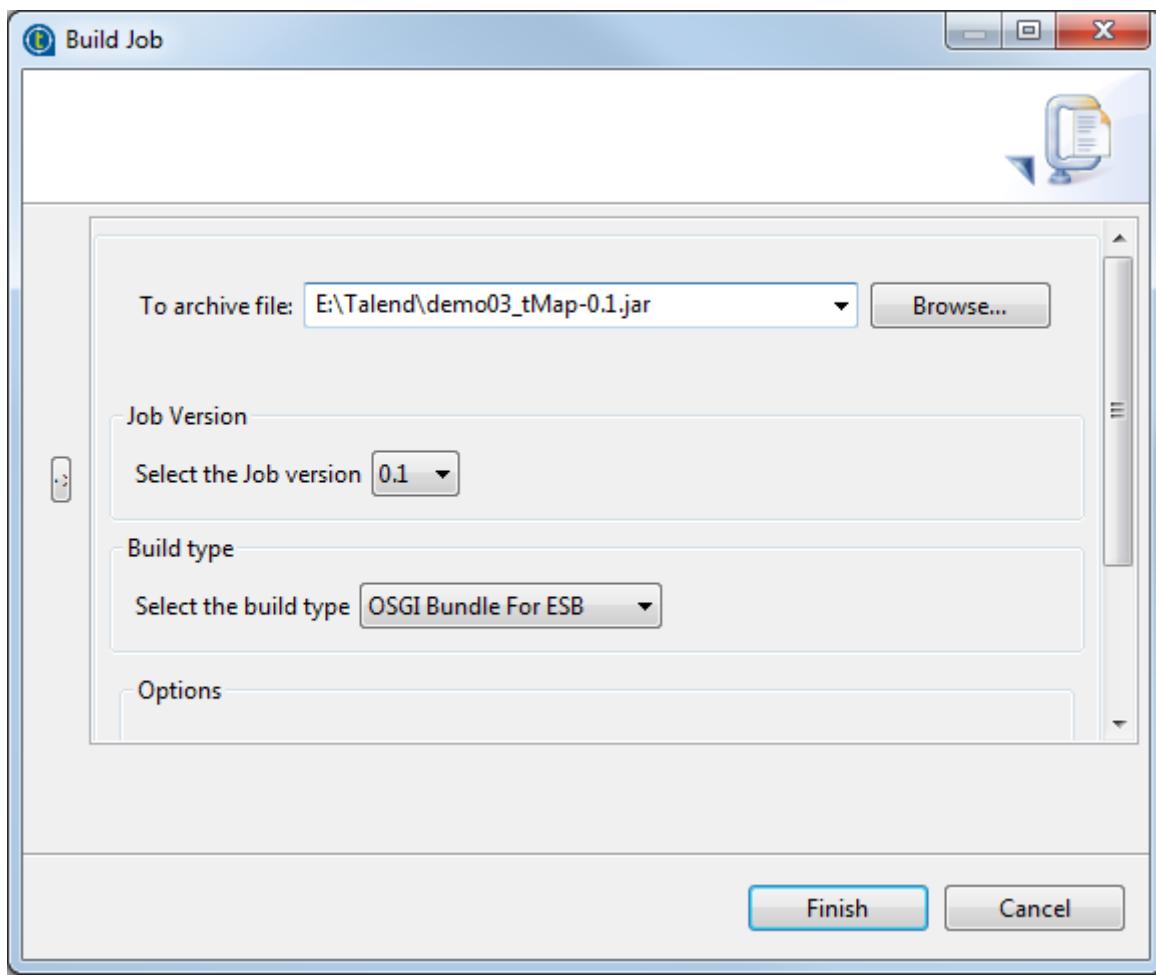
```
- <soapenv:Envelope>
 - <soapenv:Body>
 - <runJobReturn xsi:type="ns1:runJobReturn">
 - <ns1:item xsi:type="ns1:ArrayOf_xsd_string">
 <ns1:item xsi:type="xsd:string">0</ns1:item>
 </ns1:item>
 </runJobReturn>
 </soapenv:Body>
</soapenv:Envelope>
```

The return code from the Web application is 0 when there is no error and 1 if an error occurs.

For a real-life example of creating and building a Job as a Webservices using the **tBufferOutput** component, see the the **tBufferOutput** component documentation.

#### 5.2.2.4. How to build a Job as an OSGI Bundle For ESB

In the **[Build Job]** dialog box, you can change the build type in order to build the Job selection as an OSGI Bundle in order to deploy your Job in **Talend ESB Container**.



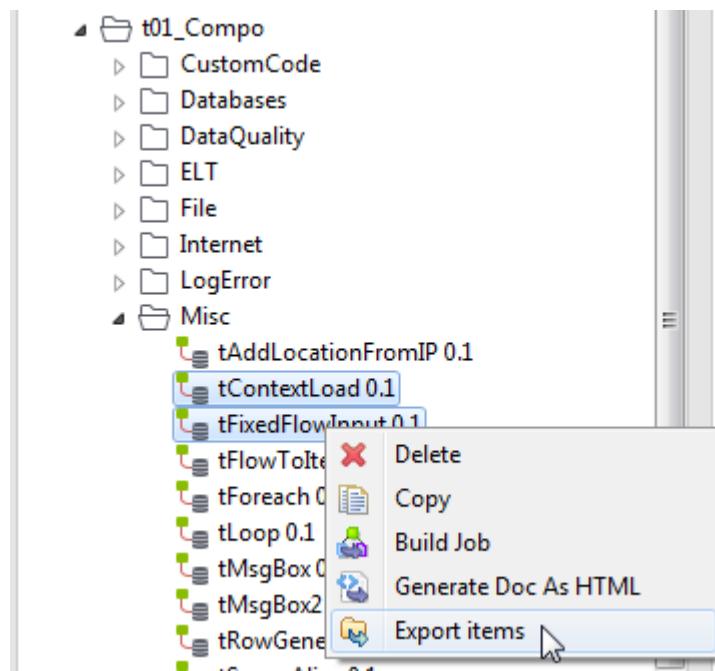
1. In the **Job Version** area, select the version number of the Job you want to build if you have created more than one version of the Job.
2. In the **Build type** area, select **OSGI Bundle For ESB** to build your Job as an OSGI Bundle.  
The extension of your build automatically change to *.jar* as it is what **Talend ESB Container** is expecting.
3. Click the **Browse...** button to specify the folder in which building your Job.
4. Click **Finish** to build it.

### 5.2.3. How to export items

You can export multiple items from the repository onto a directory or an archive file. Hence you have the possibility to export metadata information such as DB connection or Documentation along with your Job or your Business Model, for example.

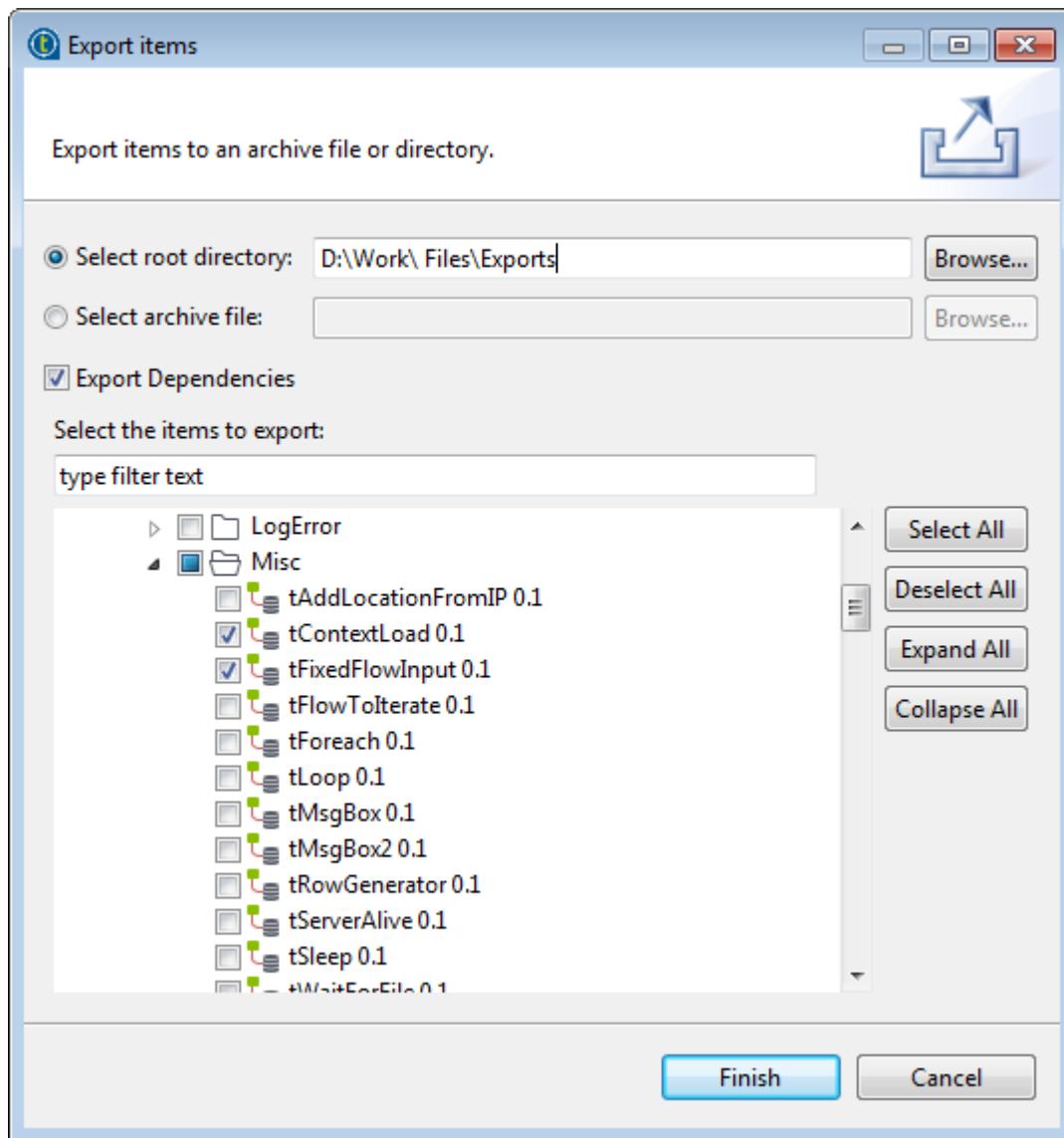
To do so:

1. In the **Repository** tree view, select the items you want to export.
2. To select several items at a time, press the **Ctrl** key and select the relevant items.



If you want to export a database table metadata entry, make sure you select the whole DB connection, and not only the relevant table as this will prevent the export process to complete correctly.

3. Right-click while maintaining the **Ctrl** key down and select **Export items** on the pop-up menu:



You can select additional items on the tree for exportation if required.

4. Click **Browse** to browse to where you want to store the exported items. Alternatively, define the archive file where to compress the files for all selected items.



If you have several versions of the same item, they will all be exported.



Select the **Export Dependencies** check box if you want to set and export routine dependencies along with Jobs you are exporting. By default, all of the user routines are selected. For further information about routines, see [What are routines](#).

5. Click **Finish** to close the dialog box and export the items.

## 5.2.4. How to change context parameters in Jobs

As explained in [How to build Jobs](#), you can edit the context parameters:

If you want to change the context selection, simply edit the .bat/.sh file and change the following setting: --context=Prod to the relevant context.

If you want to change individual parameters in the context selection, edit the .bat/.sh file and add the following setting according to your need:

| Operation                                                                                       | Setting                                                 |
|-------------------------------------------------------------------------------------------------|---------------------------------------------------------|
| To change <i>value1</i> for parameter <i>key1</i>                                               | --context_param key1=value1                             |
| To change <i>value1</i> and <i>value2</i> for respective parameters <i>key1</i> and <i>key2</i> | --context_param key1=value1 --context_param key2=value2 |
| To change a value containing space characters such as in a file path                            | --context_param key1="path to file"                     |

## 5.3. Managing repository items

*Talend Studio* enables you to edit the items centralized in the repository and to update the Jobs that use these items accordingly.

### 5.3.1. How to handle updates in repository items

You can update the metadata, context parameters that are centralized in the **Repository** tree view any time in order to update the database connection or the context group details, for example.

When you modify any of the parameters of an entry in the **Repository** tree view, all Jobs using this repository entry will be impacted by the modification. This is why the system will prompt you to propagate these modifications to all the Jobs that use the repository entry.

The following sections explain how to modify the parameters of a repository entry and how to propagate the modifications to all or some of the Jobs that use the entry in question.

#### 5.3.1.1. How to modify a repository item

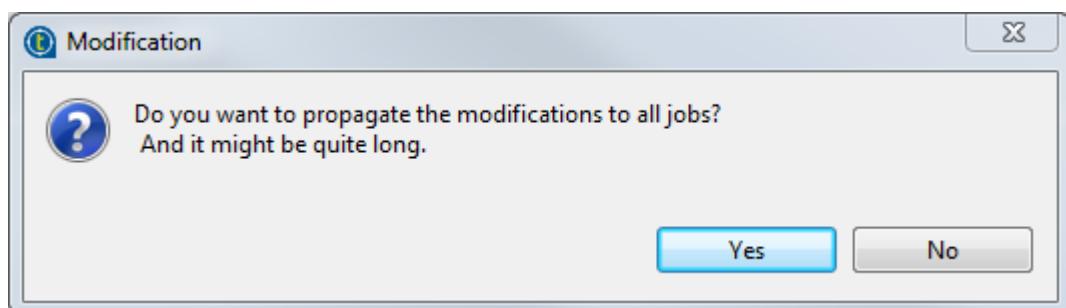
To update the parameters of a repository item, complete the following:

1. Expand the **Metadata**, or **Contexts** node in the **Repository** tree view and browse to the relevant entry that you need to update.
2. Right-click this entry and select the corresponding edit option in the contextual menu.

A respective wizard displays where you can edit each of the definition steps for the entry parameters.

When updating the entry parameters, you need to propagate the changes throughout numerous Jobs or all your Jobs that use this entry.

A prompt message pops up automatically at the end of your update/modification process when you click the **Finish** button in the wizard.



- Click **Yes** to close the message and implement the changes throughout all Jobs impacted by these changes. For more information about the first way of propagating all your changes, see [How to update impacted Jobs automatically](#).

Click **No** if you want to close the message without propagating the changes. This will allow you to propagate your changes on the impacted Jobs manually on one by one basis. For more information on another way of propagating changes, see [How to update impacted Jobs manually](#).

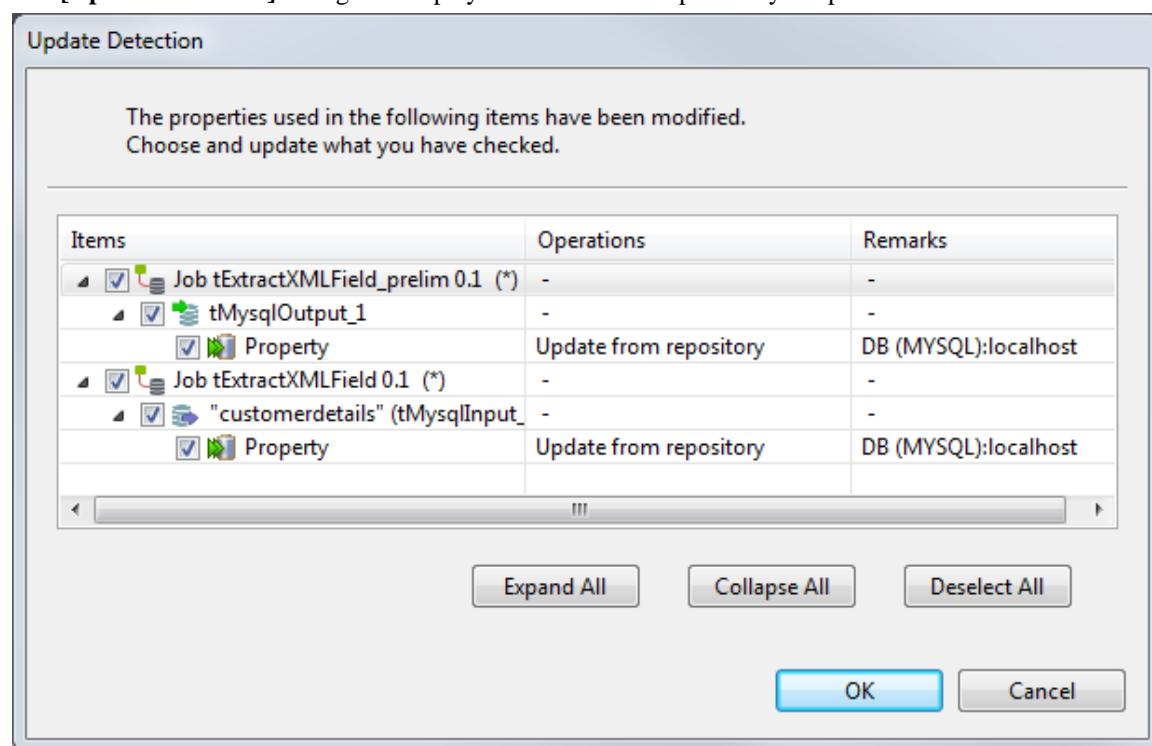
### 5.3.1.2. How to update impacted Jobs automatically

After you update the parameters of any item already centralized in the **Repository** tree view and used in different Jobs, a message will prompt you to propagate the modifications you did to all Jobs that use these parameters.

To update impacted Jobs, complete the following:

- In the [**Modification**] dialog box, click **Yes** to let the system scan your **Repository** tree view for the Jobs that get impacted by the changes you just made. This aims to automatically propagate the update throughout all your Jobs (open or not) in one click.

The [**Update Detection**] dialog box displays to list all Jobs impacted by the parameters that are modified.



You can open the [**Update Detection**] dialog box any time if you right-click the item centralized in the **Repository** tree view and select **Manage Dependencies** from the contextual menu. For more information, see [How to update impacted Jobs manually](#).

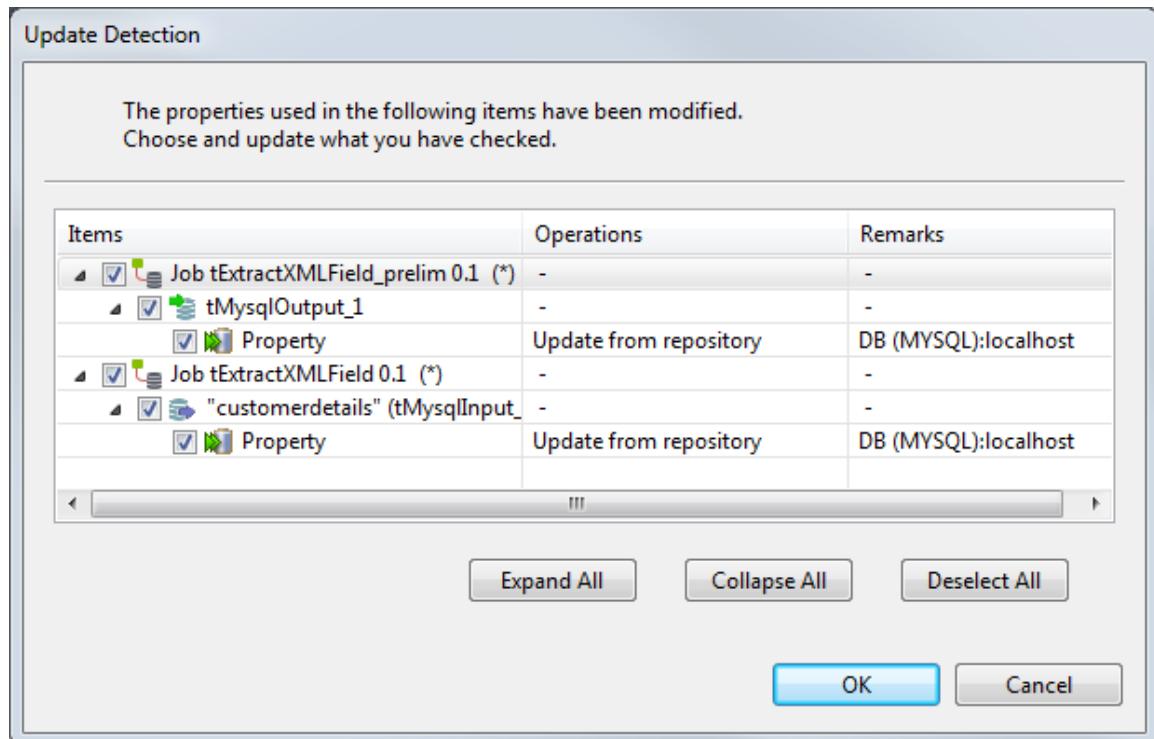
- If needed, clear the check boxes that correspond to the Jobs you do not wish to update. You can update them any time later through the **Detect Dependencies** menu. For more information, see [How to update impacted Jobs manually](#).
- Click **OK** to close the dialog box and update all selected Jobs.

### 5.3.1.3. How to update impacted Jobs manually

Before propagating changes in the parameters of an item centralized in the tree view throughout the Jobs using this entry, you might want to view all Jobs that are impacted by the changes. To do that, complete the following:

1. In the **Repository** tree view, expand the node holding the entry you want to check what Jobs use it.
2. Right-click the entry and select **Detect Dependencies**.

A progress bar indicates the process of checking for all Jobs that use the modified metadata or context parameter. Then a dialog box displays to list all Jobs that use the modified item.



3. Select the check boxes corresponding to the Jobs you want to update with the modified metadata or context parameter and clear those corresponding to the Jobs you do not want to update.
4. Click **OK** to validate and close the dialog box.



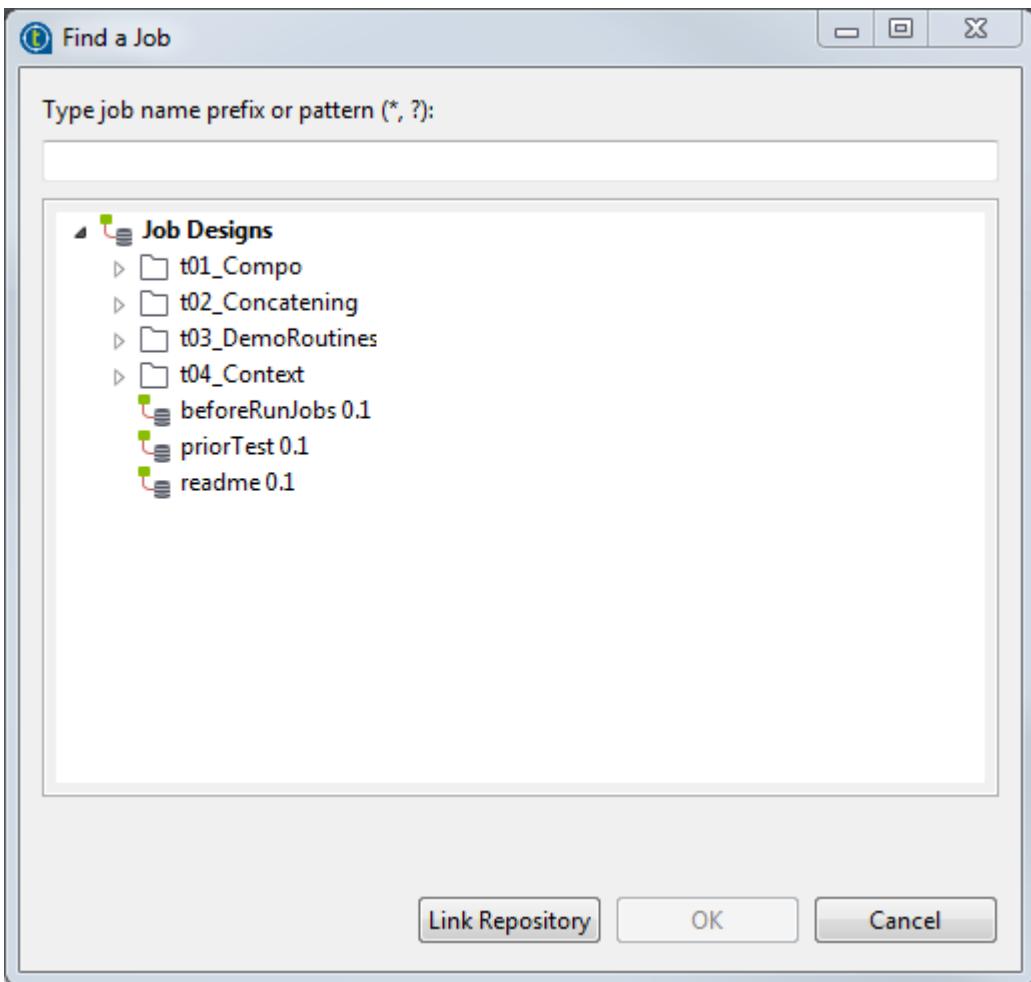
The Jobs that you choose not to update will be switched back to **Built-in**, as the link to the Repository cannot be maintained. It will thus keep their setting as it was before the change.

## 5.4. Searching a Job in the repository

If you want to open a specific Job in the **Repository** tree view of the current **Integration** perspective of *Talend Studio* and you can not find it for one reason or another, you can simply click  on the quick access toolbar.

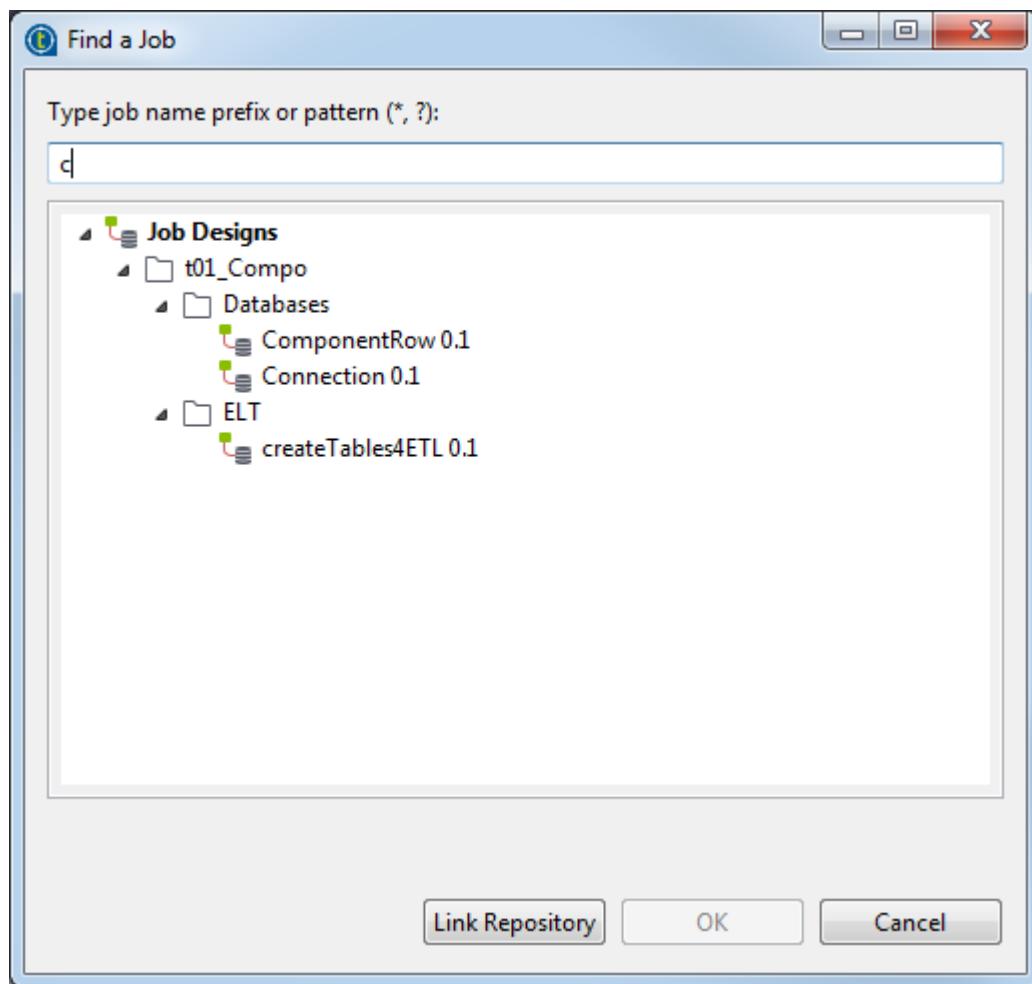
To find a Job in the **Repository** tree view, complete the following:

1. On *Talend Studio* toolbar, click  to open the **[Find a Job]** dialog box that lists automatically all the Jobs you created in the current Studio.



2. Enter the Job name or part of the Job name in the upper field.

When you start typing your text in the field, the Job list is updated automatically to display only the Job(s) which name(s) match(es) the letters you typed in.



3. Select the desired Job from the list and click **Link Repository** to automatically browse to the selected Job in the **Repository** tree view.
  4. If needed, click **Cancel** to close the dialog box and then right-click the selected Job in the **Repository** tree view to perform any of the available operations in the contextual menu.
- Otherwise, click **OK** to close the dialog box and open the selected Job on the design workspace.

## 5.5. Managing Job versions

When you create a Job in *Talend Studio*, by default its version is 0.1, where 0 stands for the major version and 1 for the minor version.

You can create as many versions of the same Job as you want. To do that:

1. Close your Job if it is open on the design workspace. Otherwise, its properties will be read-only and thus you cannot modify them.
2. In the **Repository** tree view, right-click your Job and select **Edit properties** from the drop-down list to open the **[Edit properties]** dialog box.
3. Next to the **Version** field, click the **M** button to increment the major version and the **m** button to increment the minor version.
4. Click **Finish** to validate the modification.



By default, when you open a Job, you open its last version.

Any previous version of the Job is read-only and thus cannot be modified.

To change the version of your Job, you can also:

1. Close your Job if it is open on the design workspace. Otherwise, its properties will be read-only and thus you cannot modify them.
2. In the **Repository** tree view, right-click your Job and select **Open another version** from the drop-down list.
3. In the dialog box, select the **Create new version and open it** check box and click the **M** button to increment the major version and the **m** button to increment the minor version.
4. Click **Finish** to validate the modification and open this new version of your Job.

You can also save your currently active Job and increment its version at the same time, by clicking **File > Save As...** and setting a new version in the **[Save As]** dialog box.



If you give your Job a new name, this option does not overwrite your current Job, but it saves your Job as a new one with the same version of the current Job or with a new version if you specify one.

You can access a list of the different versions of a Job and perform certain operations. To do that:

1. In the **Repository** tree view, select the Job you want to consult the versions of.
2. On the configuration tabs panel, click the **Job** tab and then click **Version** to display the version list of the selected Job.
3. Right-click the Job version you want to consult.
4. Do one of the followings:

| Select                      | To...                                                                                                                                                                                                                  |
|-----------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Edit Job</b>             | open the last version of the Job.<br><br>This option is available only when you select the last version of the Job.                                                                                                    |
| <b>Read job</b>             | consult the Job in read-only mode.                                                                                                                                                                                     |
| <b>Open Job Hierarchy</b>   | consult the hierarchy of the Job.                                                                                                                                                                                      |
| <b>Edit properties</b>      | edit Job properties.<br><br><b>Note:</b> The Job should not be open on the design workspace, otherwise it will be in read-only mode.<br><br>This option is available only when you select the last version of the Job. |
| <b>Run job</b>              | execute the Job.                                                                                                                                                                                                       |
| <b>Generate Doc As HTML</b> | generate details documentation about the Job.                                                                                                                                                                          |

You can also manage the version of several Jobs and/or metadata at the same time, as well as Jobs and their dependencies and/or child Jobs from the Project Settings. For more information, see [Version management](#).

## 5.6. Documenting a Job

*Talend Studio* enables you to generate documentation that gives general information about your projects, Jobs or joblets. You can automate the generation of such documentation and edit any of the generated documents.

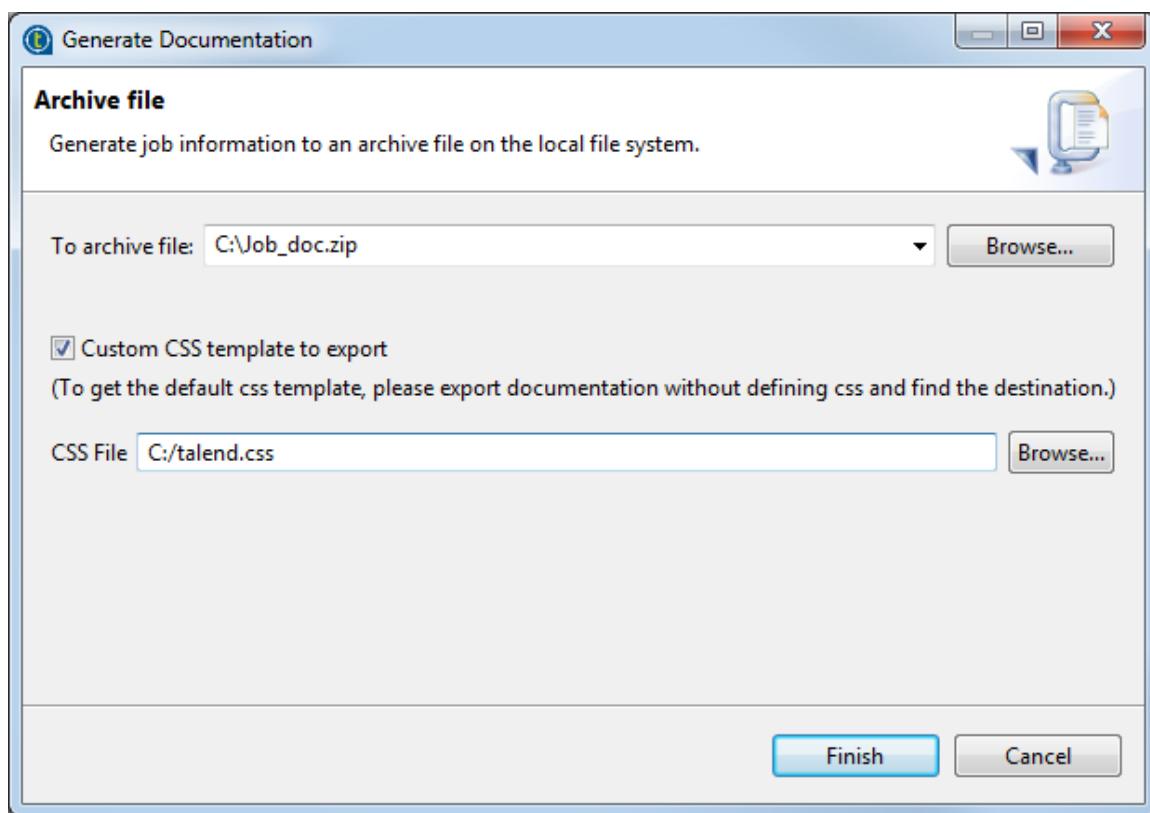
## 5.6.1. How to generate HTML documentation

*Talend Studio* allows you to generate detailed documentation in HTML of the Job(s) you select in the **Repository** tree view of your Studio in the **Integration** perspective. This auto-documentation offers the following:

- The properties of the project where the selected Jobs have been created,
- The properties and settings of the selected Jobs along with preview pictures of each of the Jobs,
- The list of all the components used in each of the selected Jobs and component parameters.

To generate an HTML document for a Job, complete the following:

1. In the **Repository** tree view, right-click a **Job** entry or select several items to produce multiple documentations.
2. Select **Generate Doc as HTML** on the contextual menu.



3. Browse to the location where the generated documentation archive should be stored.
4. In the same field, type in a name for the archive gathering all generated documents.
5. Select the **Use CSS file as a template to export** check box to activate the **CSS File** field if you need to use a CSS file.
6. In the **CSS File** field, browse to, or enter the path to the CSS file to be used.
7. Click **Finish** to validate the generation operation.

The archive file is generated in the defined path. It contains all required files along with the Html output file. You can open the HTML file in your favorite browser.

## 5.6.2. How to update the documentation on the spot

You can choose to manually update your documentation on the spot.

To update a single document, right-click the relevant documentation entry and select **Update documentation**.

# 5.7. Handling Job execution

You can execute a Job in several ways. This mainly depends on the purpose of your Job execution and on your user level.

This section describes:

- [\*How to run a Job in normal mode\*](#)
- [\*How to run a Job in Java Debug mode\*](#)
- [\*How to run a Job in Traces Debug mode\*](#)
- [\*How to set advanced execution settings\*](#)
- [\*How to show JVM resource usage during Job execution\*](#)
- [\*How to deploy a Job on SpagoBI server\*](#)

## 5.7.1. How to run a Job in normal mode



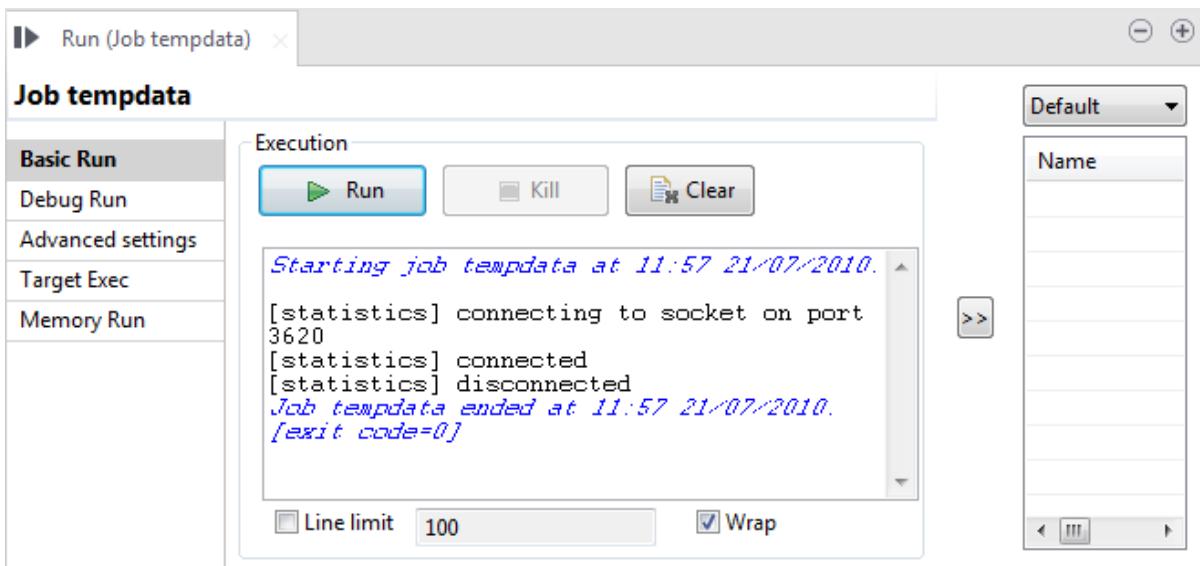
Make sure you saved your **Job** before running it in order for all properties to be taken into account.

To run your Job in a normal mode, do the following:

1. Click the **Run** view to access it.
2. Click the **Basic Run** tab to access the normal execution mode.
3. In the **Context** area to the right of the view, select in the list the proper context for the Job to be executed in. You can also check the variable values.

If you have not defined any particular execution context, the context parameter table is empty and the context is the default one. Related topic: [\*Using contexts and variables\*](#).

1. Click **Run** to start the execution.
2. On the same view, the console displays the progress of the execution. The log includes any error message as well as start and end messages. It also shows the Job output in case of a **tLogRow** component is used in the Job design.
3. To define the lines of the execution progress to be displayed in the console, select the **Line limit** check box and type in a value in the field.
4. Select the **Wrap** check box to wrap the text to fit the console width. This check box is selected by default. When it is cleared, a horizontal scrollbar appears, allowing you to view the end of the lines.



Before running again a Job, you might want to remove the execution statistics and traces from the designing workspace. To do so, click the **Clear** button.

If for any reason, you want to stop the Job in progress, simply click the **Kill** button. You will need to click the **Run** button again, to start again the Job.

*Talend Studio* offers various informative features displayed during execution, such as statistics and traces, facilitating the Job monitoring and debugging work. For more information, see the following sections.

## 5.7.2. How to run a Job in Java Debug mode

To follow step by step the execution of a Job to identify possible bugs, you can run it in Debug mode.

To access the Debug mode:

1. Click the **Run** view to access it.
2. Click the **Debug Run** tab to access the debug execution modes.

Before running your Job in Debug mode, add breakpoints to the major steps of your Job flow.



This will allow you to get the Job to automatically stop at each breakpoint. This way, components and their respective variables can be verified individually and debugged if required.

To add breakpoints to a component, right-click it on the design workspace, and select **Add breakpoint** on the contextual menu.

A pause icon displays next to the component where the break is added.

To switch to debug mode, click the **Java Debug** button on the **Debug Run** tab of the **Run** panel. *Talend Studio's* main window gets reorganized for debugging.

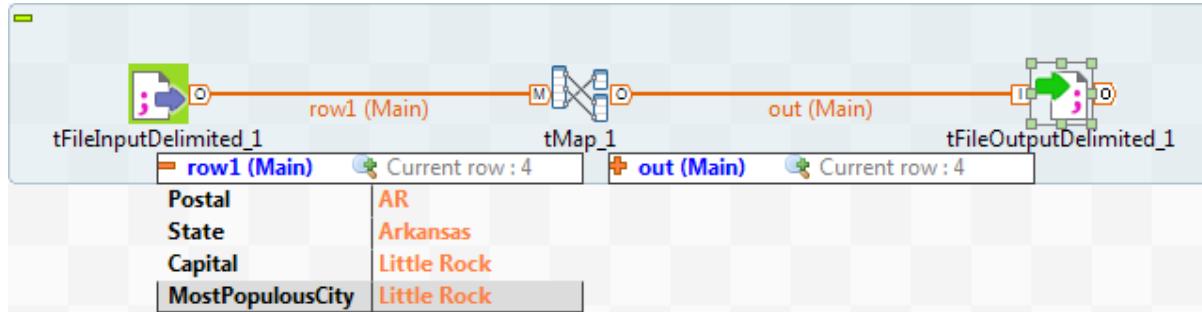
You can then run the Job step by step and check each breakpoint component for the expected behavior and variable values.

To switch back to *Talend Studio* designer mode, click **Window**, then **Perspective** and select **Integration**.

### 5.7.3. How to run a Job in Traces Debug mode

The traces feature allows you to monitor data processing when running a Job in the **Integration** perspective of *Talend Studio*.

It provides a row by row view of the component behavior and displays the dynamic result next to the **Row** link on the design workspace.



This feature allows you to monitor all the components of a Job, without switching to the debug mode, hence without requiring advanced Java knowledge.

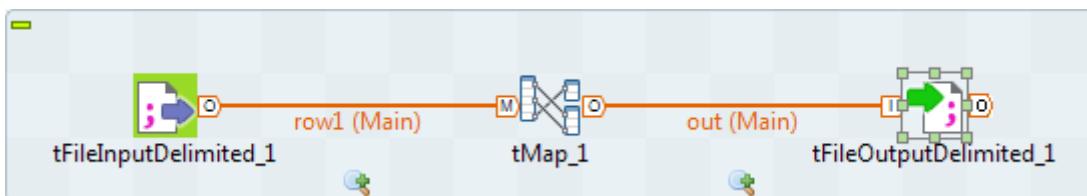
The **Traces** function displays the content of processed rows in a table.



Exception is made for external components which cannot offer this feature if their design does not include it.

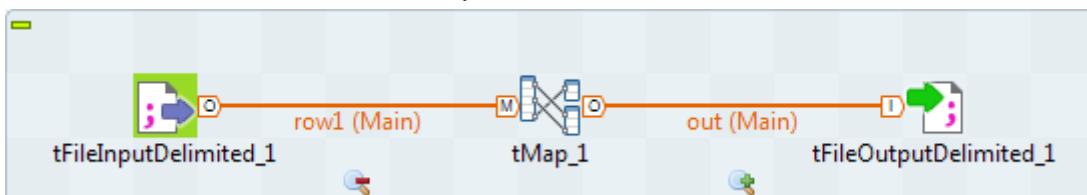
You can activate or deactivate **Traces** or decide what processed columns to display in the traces table that displays on the design workspace when launching the current Job.

To activate the **Traces** mode in a Job:



1. Click the **Run** view.
2. Click the **Debug Run** tab to access the debug and traces execution modes.
3. Click the down arrow of the **Java Debug** button and select the **Traces Debug** option. An icon displays under every flow of your Job to indicate that process monitoring is activated.
4. Click the **Traces Debug** to execute the Job in Traces mode.

To deactivate the **Traces** on one of the flows in your Job:

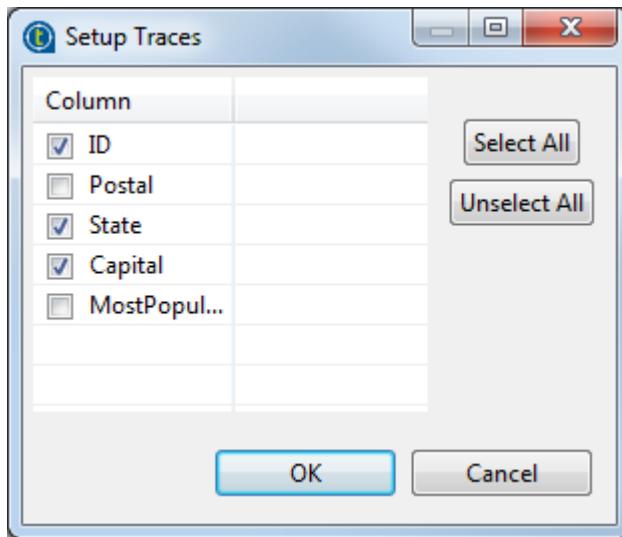


1. Right-click the **Traces** icon under the relevant flow.

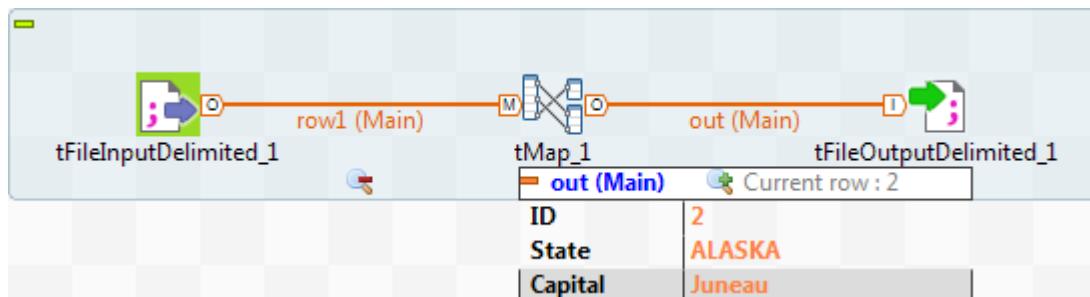
- Select **Disable Traces** from the list. A red minus sign replaces the green plus sign on the icon to indicate that the **Traces** mode has been deactivated for this flow.

To choose which columns of the processed data to display in the traces table, do the following:

- Right-click the **Traces** icon for the relevant flow, then select **Setup Traces** from the list. The **[Setup Traces]** dialog box appears.



- In the dialog box, clear the check boxes corresponding to the columns you do not want to display in the Traces table.
- Click **OK** to close the dialog box.



Monitoring data processing starts when you execute the Job and stops at the end of the execution.

To remove the displayed monitoring information, click the **Clear** button in the **Debug Run** tab.

## 5.7.4. How to set advanced execution settings

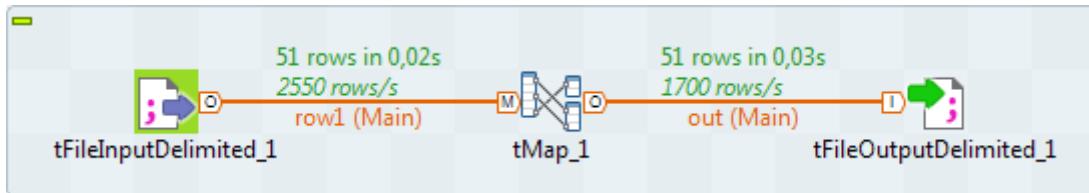
In the **Advanced settings** tab of the **Run** view, several advanced execution settings are available to make the execution of the Jobs handier:

- Statistics**, this feature displays processing performance rate. For more information, see [How to display Statistics](#).
- Exec time**, this feature displays the execution time in the console at the end of the execution. For more information, see [How to display the execution time and other options](#).
- Save Job before execution**, this feature allows to automatically save the Job before its execution.
- Clear before run**, this feature clears all the results of a previous execution before re-executing the Job.

- **JVM Setting**, this feature allows you to define the parameters of your JVM according to your needs. For an example of how this can be used, see [How to display special characters in the console](#).

### 5.7.4.1. How to display Statistics

The **Statistics** feature displays each component performance rate, under the flow links on the design workspace.



It shows the number of rows processed and the processing time in row per second, allowing you to spot straight away any bottleneck in the data processing flow.

For trigger links like **OnComponentOK**, **OnComponentError**, **OnSubjobOK**, **OnSubjobError** and **If**, the **Statistics** option displays the state of this trigger during the execution time of your Job: Ok or Error and True or False.



Exception is made for external components which cannot offer this feature if their design does not include it.

In the **Run** view, click the **Advanced settings** tab and select the **Statistics** check box to activate the Stats feature and clear the box to disable it.

The calculation only starts when the Job execution is launched, and stops at the end of it.

Click the **Clear** button from the **Basic** or **Debug Run** views to remove the calculated stats displayed. Select the **Clear before Run** check box to reset the Stats feature before each execution.



The statistics thread slows down Job execution as the Job must send these stats data to the design workspace in order to be displayed.

You can also save your Job before the execution starts. Select the relevant option check box.

### 5.7.4.2. How to display the execution time and other options

To display the Job total execution time after Job execution, select in the **Advanced settings** tab of the **Run** view the **Exec time** check box before running the Job.

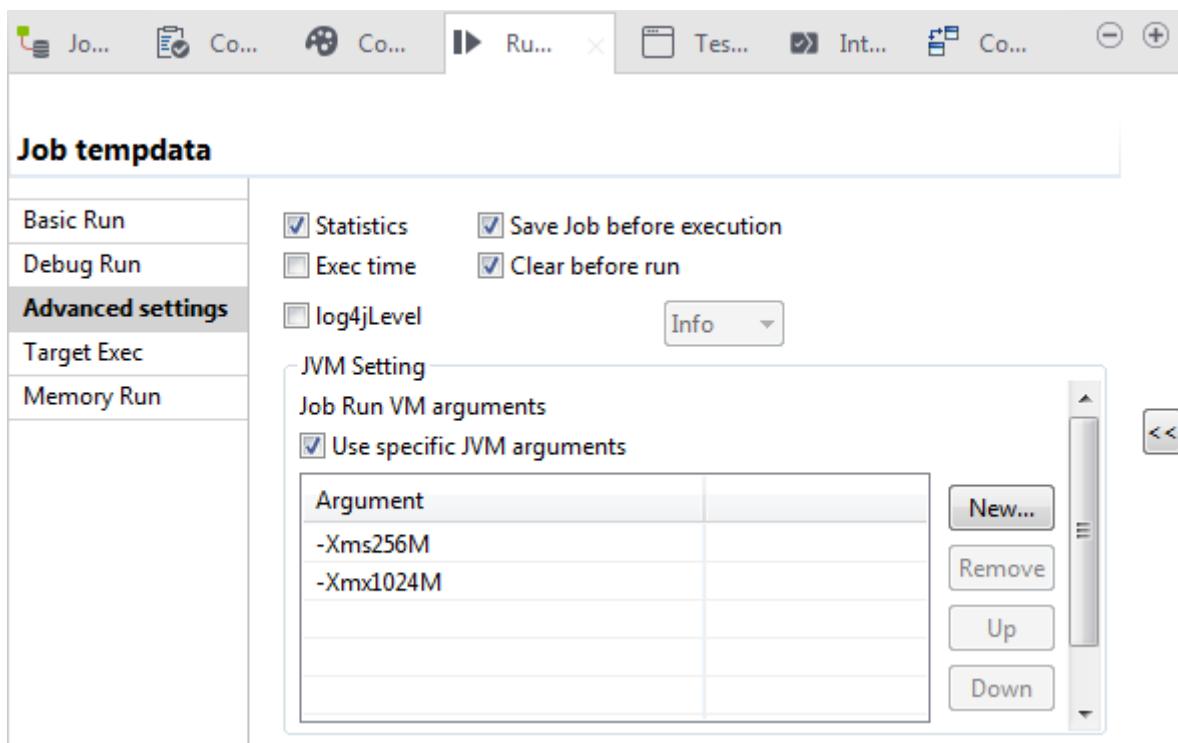
This way you can test your Job before going to production.

You can also clear the design workspace before each Job execution by selecting the check box **Clear before Run**.

You can also save your Job before the execution starts. Select the relevant option check box.

### 5.7.4.3. How to display special characters in the console

*Talend Studio* can display special characters in the console. To enable the display of Chinese, Japanese or Korean characters, for example, proceed as follows before executing the Job:



1. Select the **Advanced settings** tab.
2. In the **JVM settings** area of the tab view, select the **Use specific JVM arguments** check box to activate the **Argument** table.
3. Next to the **Argument** table, click the **New...** button to pop up the **[Set the VM argument]** dialog box.
4. In the dialog box, type in `-Dfile.encoding=UTF-8`.
5. Click **OK** to close the dialog box.

This argument can be applied for all of your Job executions in *Talend Studio*. For further information about how to apply this JVM argument for all of the Job executions, see [Debug and Job execution preferences \(Talend > Run/Debug\)](#).

## 5.7.5. How to show JVM resource usage during Job execution

The **Memory Run** vertical tab of the **Run** view of your *Talend Studio* allows you to monitor real-time JVM resource usage during Job execution, including memory consumption and host CPU usage, so that you can take appropriate actions when the resource usage is too high and results in low performance of your *Talend Studio*, such as increasing the memory allocated to the JVM, stopping unnecessary Jobs, and so on.

To monitor JVM resource usage at Job execution, do the following:

1. Open your Job.

In the **Run** view, click the **Memory Run** tab.

2. Click **Run** to run the Job.

You can click **Run** on the **Memory Run** tab to monitor the JVM resource usage by your Job at any time even after you launch your Job from the **Basic Run** tab.

The Studio console displays curve graphs showing the JVM heap usage and CPU usage respectively during the Job execution. Warning messages are shown in red on the **Job execution information** area when the relevant thresholds are reached.



3. To view the information about resources used at a certain point of time during the Job execution, move the mouse onto that point of time on the relevant graph. Depending on the graph on which you move your mouse pointer, you can see the information about allocated heap size, the 90% heap threshold, and the 70% heap threshold, or the CPU usage, at the point of time.
4. To run the Garbage Collector at a particular interval, select the **With Garbage Collector pace set to** check box and select an interval in seconds. The Garbage Collector automatically runs at the specified interval.  
To run the Garbage Collector once immediately, click the **Trigger GC** button.
5. To export the log information into a text file, click the **Export** button and select a file to save the log.
6. To stop the Job, click the **Kill** button.

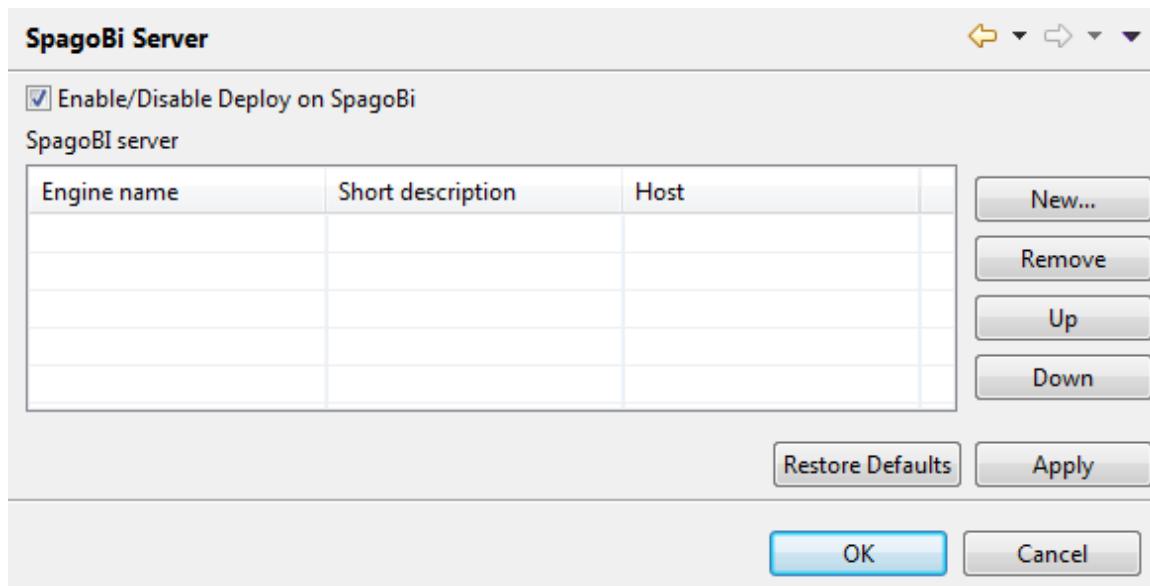
## 5.7.6. How to deploy a Job on SpagoBI server

From *Talend Studio* interface, you can deploy your Jobs easily on a SpagoBI server in order to execute them from your SpagoBI administrator.

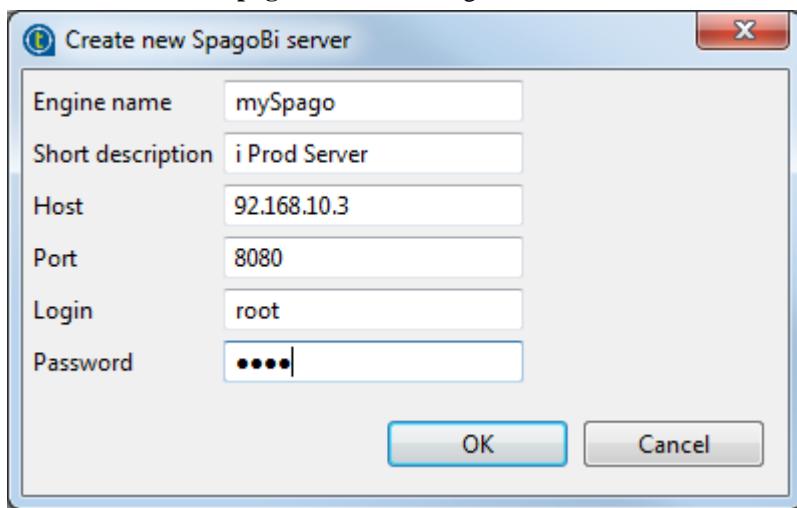
### 5.7.6.1. How to create a SpagoBI server entry

Beforehand, you need to set up your single or multiple SpagoBI server details in *Talend Studio*.

1. On the menu bar, click **Window > Preferences** to open the **[Preferences]** dialog box.
2. Expand the **Talend > Import/Export** nodes in succession and select **SpagoBI Server** to display the relevant view.



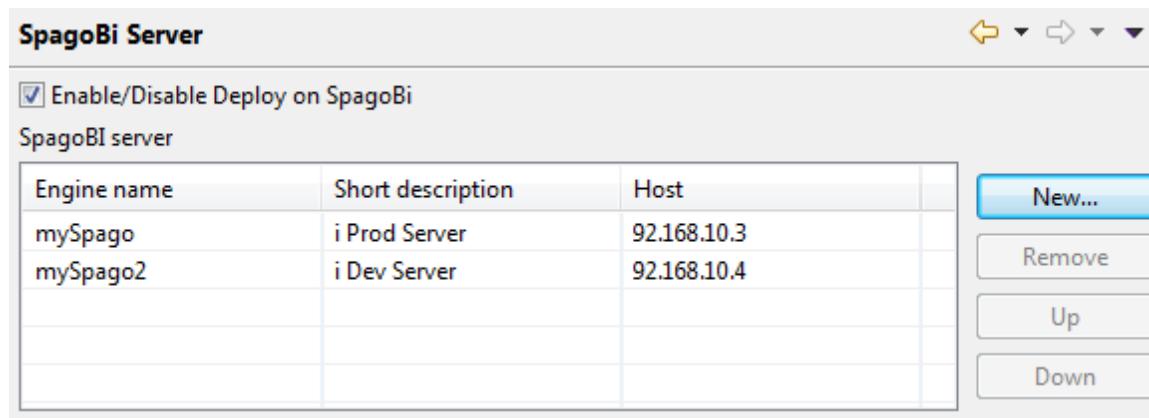
3. Select the **Enable/Disable Deploy on SpagoBI** check box to activate the deployment operation.
4. Click **New** to open the **[Create new SpagoBI server]** dialog box and add a new server to the list.



5. Enter your SpagoBI server details, as described below:

| Field             | Description                                                                                      |
|-------------------|--------------------------------------------------------------------------------------------------|
| Engine Name       | Internal engine name used in <i>Talend Studio</i> . This name is not used in the generated code. |
| Short description | Free text to describe the server entry you are recording.                                        |
| Host              | IP address or host name of the machine running the SpagoBI server.                               |
| Login             | User name required to log on to the SpagoBI server.                                              |
| Password          | Password for SpagoBI server logon authentication.                                                |

6. Click **OK** to validate the details of the new server entry and close the dialog box.



The newly created entry is added to the table of available servers. You can add as many SpagoBI entries as you need.

- Click **Apply** and then **OK** to close the **[Preferences]** dialog box.

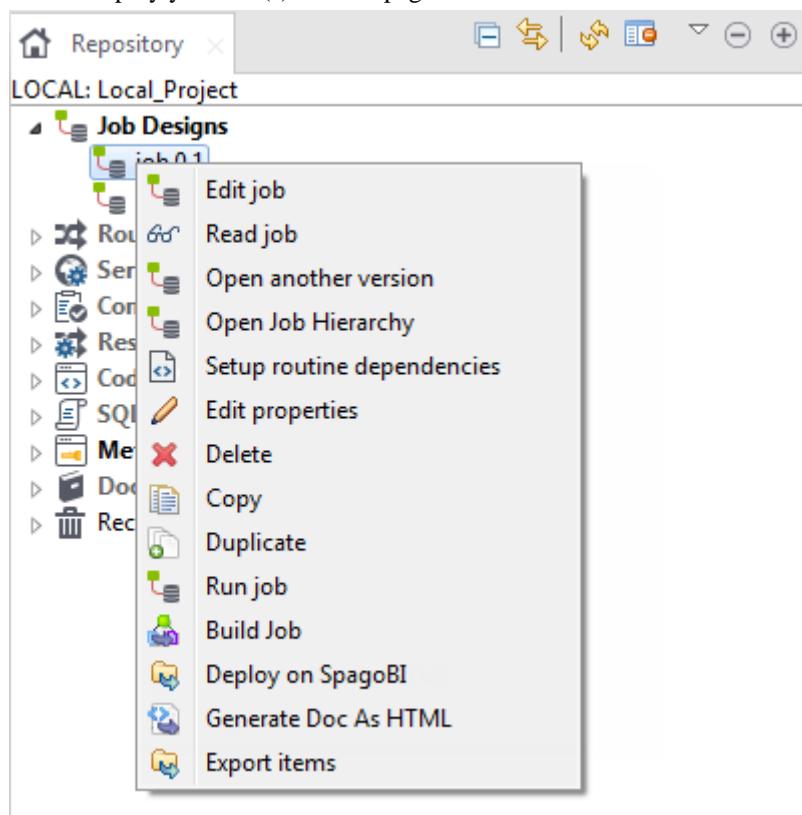
### 5.7.6.2. How to edit or remove a SpagoBI server entry

Select the relevant entry in the table, click the **Remove** button next to the table to first delete the outdated entry.

Then if required, simply create a new entry including the updated details.

### 5.7.6.3. How to deploy your Jobs on a SpagoBI server

Follow the steps below to deploy your Job(s) onto a SpagoBI server.



1. In the **Repository** tree view, expand **Job Designs** and right-click the Job to deploy.
2. In the drop-down list, select **Deploy on SpagoBI**.
3. As for any Job export, select a **Name** for the Job archive that will be created and fill it in the **To archive file** field.
4. Select the relevant **SpagoBI server** on the drop-down list.
5. The **Label**, **Name** and **Description** fields come from the Job main properties.
6. Select the relevant context in the list.
7. Click **OK** once you have completed the setting operation.

The Jobs are now deployed onto the relevant SpagoBI server. Open your SpagoBI administrator to execute your Jobs.





## Chapter 6. Mapping data flows

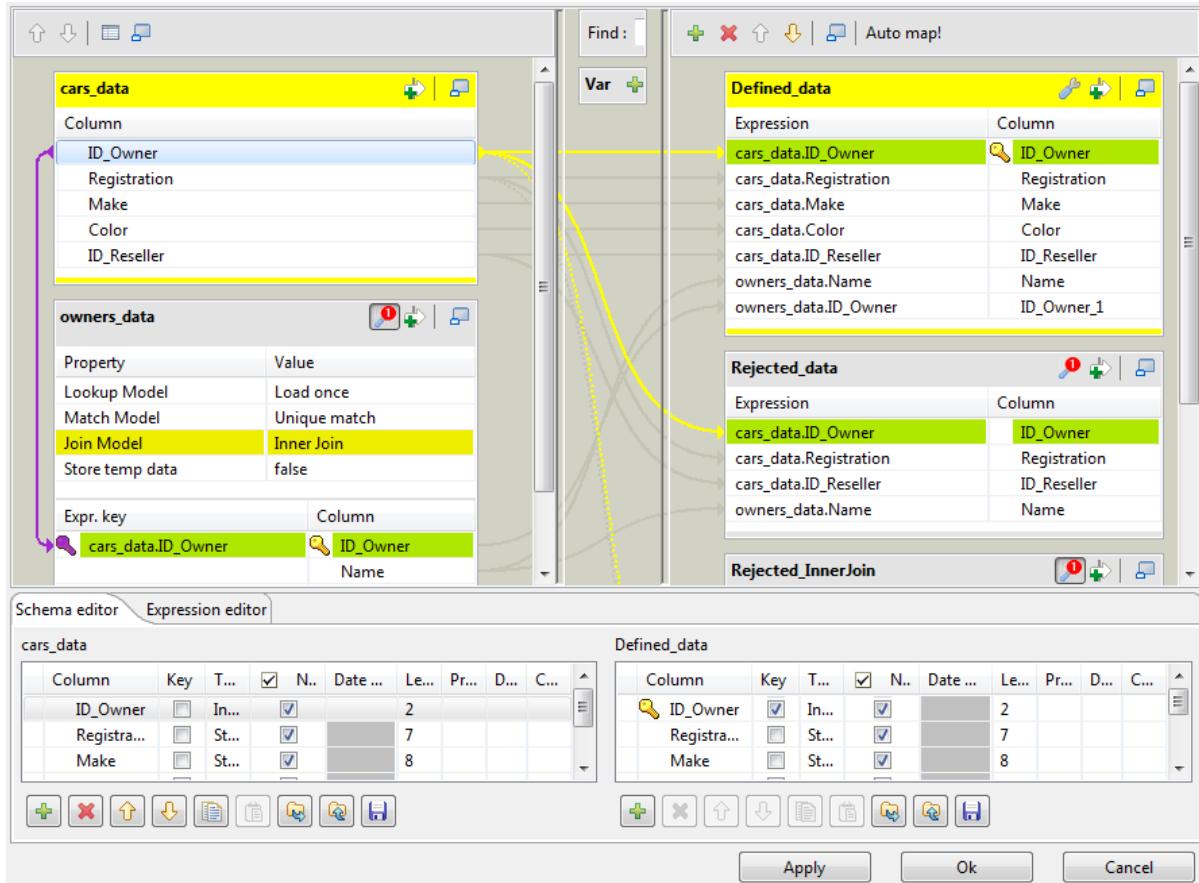
The most common way to handle multiple input and output flows including transformations and data re-routing is to use dedicated mapping components.

This chapter explains the theory behind how those mapping components can be used, by taking as example the typical ones which you can refer to for the use of the other mapping components.

## 6.1. Map editor interfaces

Mapping components are advanced components which require more detailed explanation than other **Talend Open Studio Components**. The **Map Editor** is an "all-in-one" tool allowing you to define all parameters needed to map, transform and route your data flows via a convenient graphical interface.

You can minimize and restore the **Map Editor** and all tables in the **Map Editor** using the window icons.



This figure presents the interface of **tMap**. Those of the other mapping components differ slightly in appearance. For example, in addition to the **Schema editor** and the **Expression editor** tabs on the lower part of this interface, **tXMLMap** has a third tab called **Tree schema editor**. For further information about **tXMLMap**, see [tXMLMap operation](#).

The **Map Editor** is made of several panels:

- The **Input panel** is the top left panel on the editor. It offers a graphical representation of all (main and lookup) incoming data flows. The data are gathered in various columns of input tables. Note that the table name reflects the main or lookup row from the Job design on the design workspace.
- The **Variable panel** is the central panel in the **Map Editor**. It allows the centralization of redundant information through the mapping to variable and allows you to carry out transformations.
- The **Search panel** is above the **Variable panel**. It allow you to search in the editor for columns or expressions that contain the text you enter in the **Find** field.
- The **Output panel** is the top right panel on the editor. It allows mapping data and fields from Input tables and Variables to the appropriate Output rows.
- Both bottom panels are the Input and Output schemas description. The **Schema editor** tab offers a schema view of all columns of input and output tables in selection in their respective panel.

- **Expression editor** is the edition tool for all expression keys of Input/Output data, variable expressions or filtering conditions.

The name of input/output tables in the **Map Editor** reflects the name of the incoming and outgoing flows (row connections).

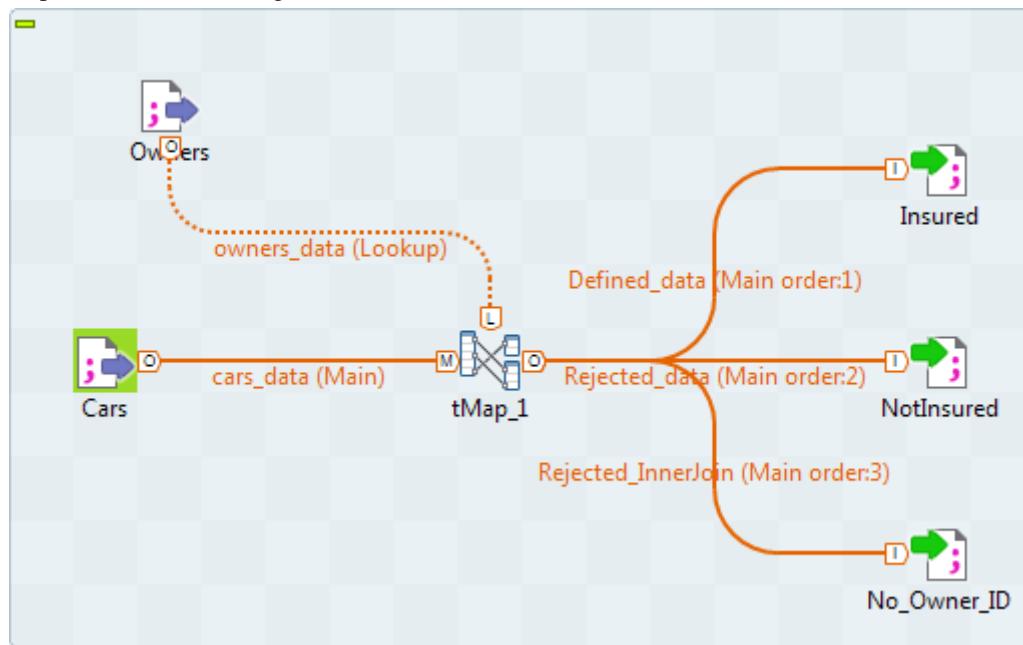
The following sections present separately different mapping components of which each is able to map flows of a specific nature.

## 6.2. tMap operation

**tMap** allows the following types of operations:

- data multiplexing and demultiplexing,
- data transformation on any type of fields,
- fields concatenation and interchange,
- field filtering using constraints,
- data rejecting.

As all these operations of transformation and/or routing are carried out by **tMap**, this component cannot be a start or end component in the Job design.



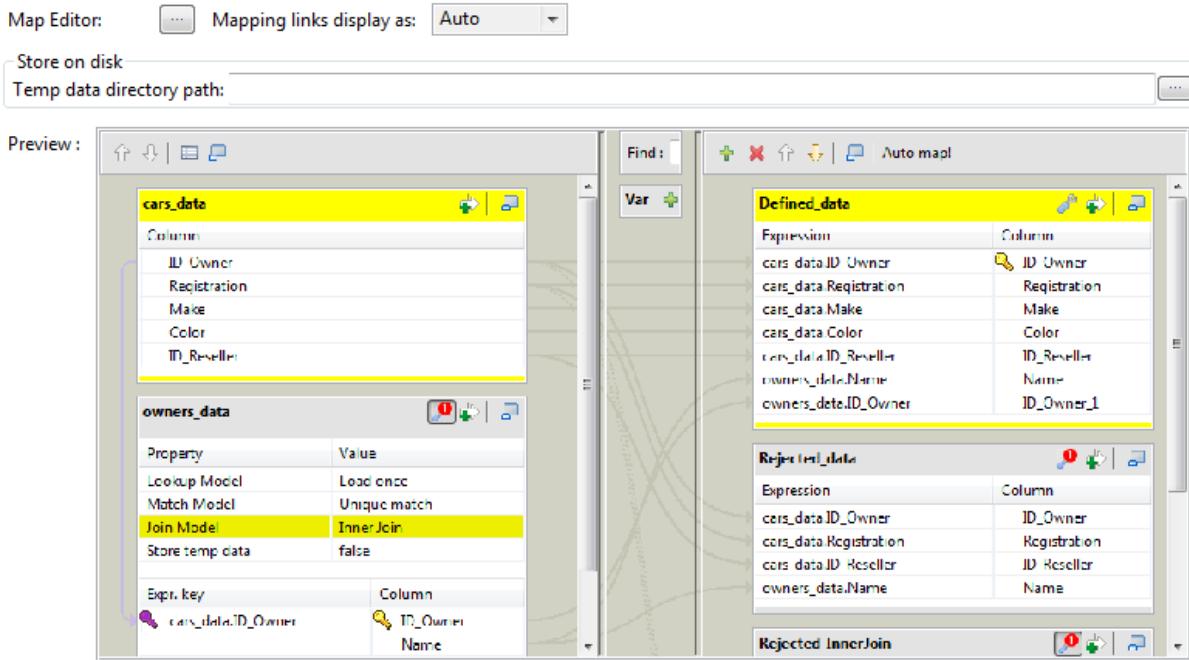
**tMap** uses incoming connections to pre-fill input schemas with data in the **Map Editor**. Therefore, you cannot create new input schemas directly in the **Map Editor**. Instead, you need to implement as many **Row** connections incoming to **tMap** component as required, in order to create as many input schemas as needed.

The same way, create as many output row connections as required. However, you can fill in the output with content directly in the **Map Editor** through a convenient graphical editor.

Note that there can be only one **Main** incoming rows. All other incoming rows are of **Lookup** type. Related topic: [Row connection](#).

Lookup rows are incoming connections from secondary (or reference) flows of data. These reference data might depend directly or indirectly on the primary flow. This dependency relationship is translated with a graphical mapping and the creation of an expression key.

The **Map Editor** requires the connections to be implemented in your Job in order to be able to define the input and output flows in the **Map Editor**. You also need to create the actual mapping in your Job in order to display the **Map Editor** in the **Preview** area of the **Basic settings** view of the **tMap** component.



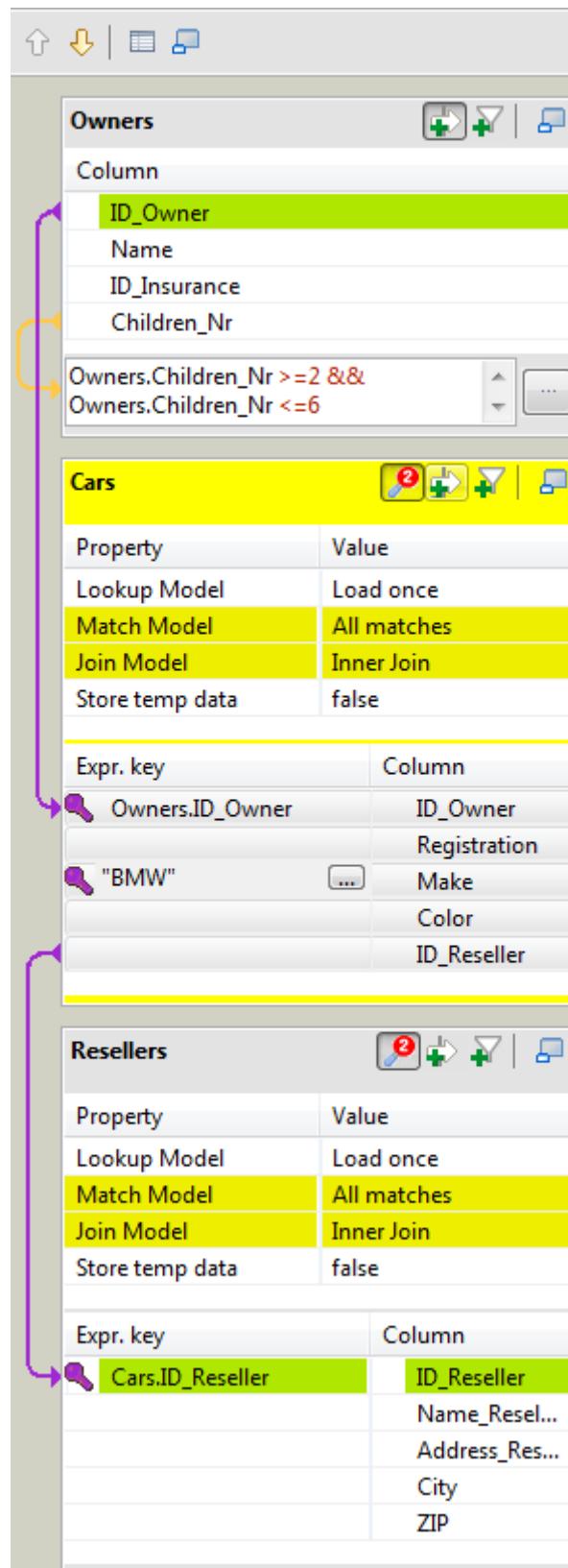
To open the **Map Editor** in a new window, double-click the **tMap** icon in the design workspace or click the three-dot button next to the **Map Editor** in the **Basic settings** view of the **tMap** component.

The following sections give the information necessary to use the **tMap** component in any of your Job designs.

## 6.2.1. Setting the input flow in the Map Editor

The order of the **Input** tables is essential. The top table reflects the **Main** flow connection, and for this reason, is given priority for reading and processing through the **tMap** component.

For this priority reason, you are not allowed to move up or down the **Main** flow table. This ensures that no Join can be lost.



Although you can use the up and down arrows to interchange **Lookup** tables order, be aware that the **Joins** between two lookup tables may then be lost.

Related topic: [How to use Explicit Join](#).

### 6.2.1.1. How to fill in Input tables with a schema

To fill in the input tables, you need to define either the schemas of the input components connected to the **tMap** component on your design workspace, or the input schemas within the **Map Editor**.

For more information about setting a component schema, see [How to define component properties](#).

For more information about setting an input schema in the **Map Editor**, see [Setting schemas in the Map Editor](#).

### Main and Lookup table content

The order of the **Input** tables is essential.

The **Main Row** connection determines the **Main** flow table content. This input flow is reflected in the first table of the **Map Editor's** Input panel.

The **Lookup** connections' content fills in all other (secondary or subordinate) tables which displays below the **Main** flow table. If you have not define the schema of an input component yet, the input table displays as empty in the Input area.

The key is also retrieved from the schema defined in the Input component. This **Key** corresponds to the key defined in the input schema where relevant. It has to be distinguished from the hash key that is internally used in the **Map Editor**, which displays in a different color.

### Variables

You can use global or context variables or reuse the variable defined in the **Variables** area. Press **Ctrl+Space bar** to access the list of variables. This list gathers together global, context and mapping variables.

The list of variables changes according to the context and grows along new variable creation. Only valid mappable variables in the context show on the list.

The screenshot shows the Talend Map Editor interface. At the top, there is a search bar labeled "Expr. key" containing "Owners.ID\_Owner". Below it is a table with columns "Column" and "Expr. key". The table contains the following rows:

| Column       | Expr. key       |
|--------------|-----------------|
| ID_Owner     | Owners.ID_Owner |
| Registration |                 |
| Make         |                 |
| Color        |                 |
| ID_Reseller  |                 |

At the bottom left, there is a list of variables: Owners.ID\_Owner, Owners.Name, Owners.ID\_Insurance, Owners.Children\_Nr, Cars.ID\_Owner, Cars.Registration, Cars.Make, Cars.Color, Cars.ID\_Reseller, ABS(double tParam) : id\_Double - Mathematical, and ACOS(double tParam) : id\_Double - Mathematical. The variable "Owners.ID\_Owner" is highlighted with a yellow background. To the right of the list, a tooltip provides metadata for the selected column:

Metadata column 'ID\_Owner' properties :

- Column:ID\_Owner
- Key :false
- Type:id\_String
- Length :5
- Precision :
- Default :
- Comment :
- Expression key :

Docked at the **Variable** list, a metadata tip box display to provide information about the selected column.

Related topic: [Mapping variables](#)

### 6.2.1.2. How to use Explicit Join

In fact, **Joins** let you select data from a table depending upon the data from another table. In the **Map Editor** context, the data of a **Main** table and of a **Lookup** table can be bound together on **expression keys**. In this case, the order of table does fully make sense.

Simply drop column names from one table to a subordinate one, to create a **Join** relationship between the two tables. This way, you can retrieve and process data from multiple inputs.

The join displays graphically as a purple link and creates automatically a key that will be used as a hash key to speed up the match search.

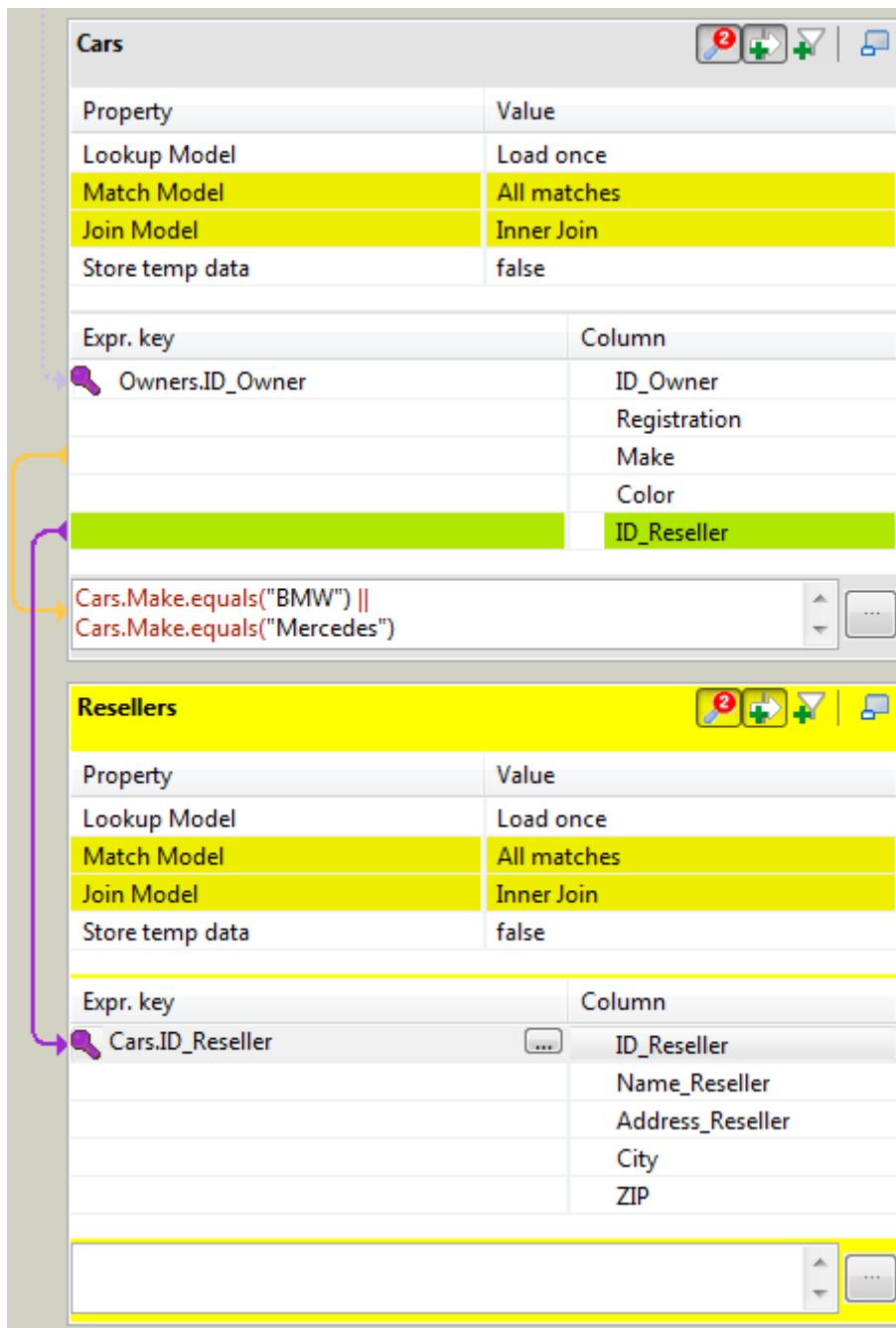
You can create direct joins between the main table and lookup tables. But you can also create indirect joins from the main table to a lookup table, via another lookup table. This requires a direct join between one of the **Lookup** table to the **Main** one.



You cannot create a **Join** from a subordinate table towards a superior table in the **Input** area.

The **Expression key** field which is filled in with the dragged and dropped data is editable in the input schema, whereas the column name can only be changed from the **Schema editor** panel.

You can either insert the dragged data into a new entry or replace the existing entries or else concatenate all selected data into one cell.



For further information about possible types of drag and drops, see [Mapping the Output setting](#).

If you have a big number of input tables, you can use the minimize/maximize icon to reduce or restore the table size in the **Input** area. The Join binding two tables remains visible even though the table is minimized.

Creating a Join automatically assigns a hash key onto the joined field name. The key symbol displays in violet on the input table itself and is removed when the Join between the two tables is removed.

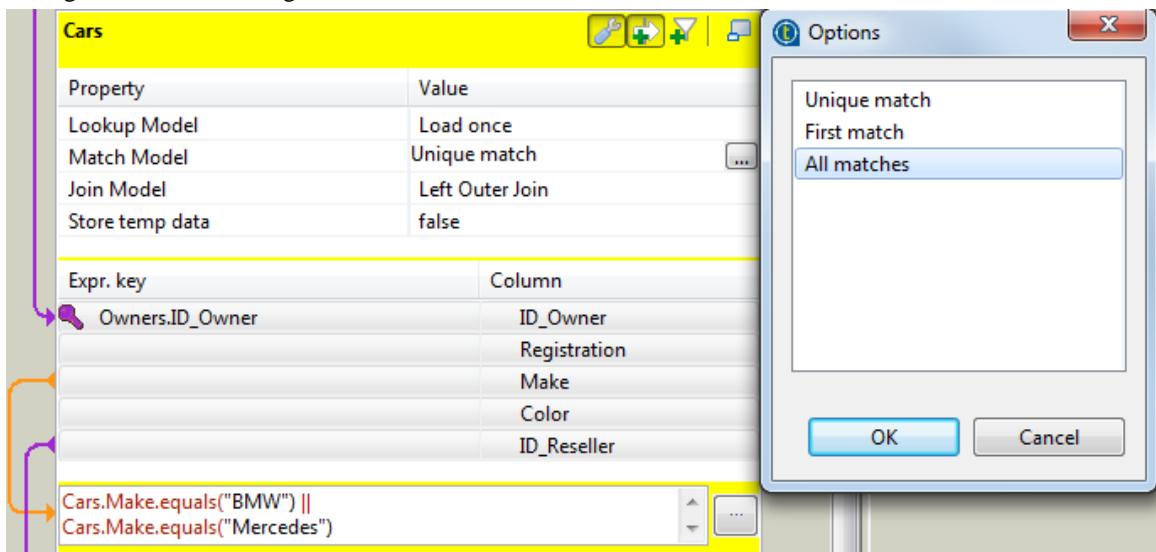
Related topics:

- [Setting schemas in the Map Editor](#)
- [How to use Inner Join](#)

Along with the explicit Join you can select whether you want to filter down to a unique match or if you allow several matches to be taken into account. In this last case, you can choose to consider only the first or the last match or all of them.

To define the match model for an explicit Join:

1. Click the **tMap settings** button at the top of the table to which the Join links to display the table properties.
2. Click in the **Value** field corresponding to **Match Model** and then click the three-dot button that appears to open the **[Options]** dialog box.
3. In the **[Options]** dialog box, double-click the wanted match model, or select it and click **OK** to validate the setting and close the dialog box.



## Unique Match

This is the default selection when you implement an explicit Join. This means that only the last match from the Lookup flow will be taken into account and passed on to the output.

The other matches will be then ignored.

## First Match

This selection implies that several matches can be expected in the lookup. The First Match selection means that in the lookup only the first encountered match will be taken into account and passed onto the main output flow.

The other matches will then be ignored.

## All Matches

This selection implies that several matches can be expected in the lookup flow. In this case, all matches are taken into account and passed on to the main output flow.

### 6.2.1.3. How to use Inner Join

The **Inner join** is a particular type of Join that distinguishes itself by the way the rejection is performed.

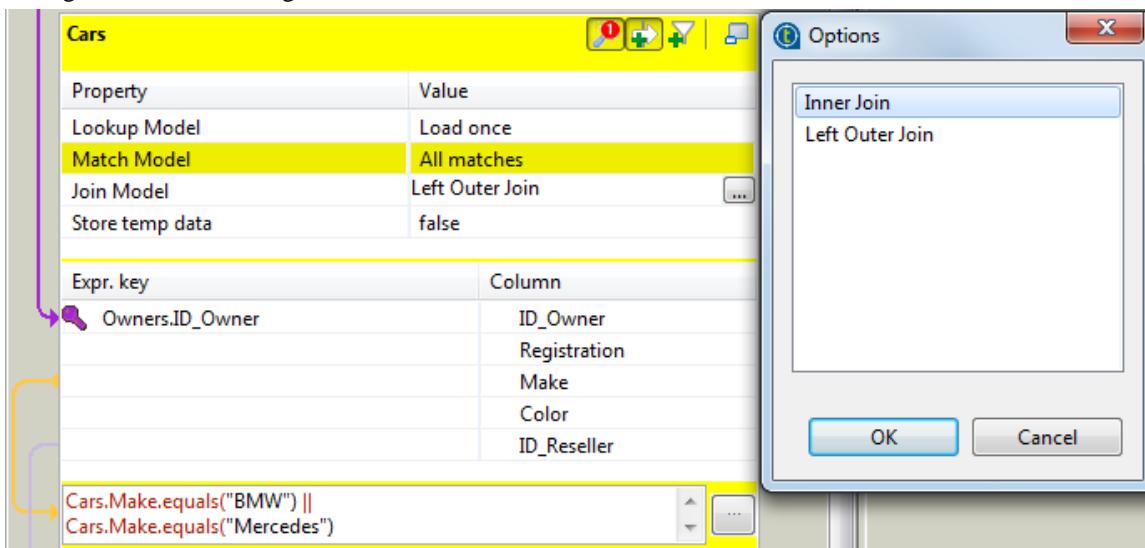
This option avoids that null values are passed on to the main output flow. It allows also to pass on the rejected data to a specific table called **Inner Join Reject** table.

If the data searched cannot be retrieved through the explicit Join or the filter Join, in other words, the Inner Join cannot be established for any reason, then the requested data will be rejected to the Output table defined as **Inner Join Reject** table if any.

Simply drop column names from one table to a subordinate one, to create a **Join** relationship between the two tables. The Join is displayed graphically as a purple link and creates automatically a key that will be used as a hash key to speed up the match search.

To define the type of an explicit Join:

1. Click the **tMap settings** button at the top of the table to which the Join links to display the table properties.
2. Click in the **Value** field corresponding to **Join Model** and then click the three-dot button that appears to open the **[Options]** dialog box.
3. In the **[Options]** dialog box, double-click the wanted Join type, or select it and click **OK** to validate the setting and close the dialog box.



An **Inner Join** table should always be coupled to an **Inner Join Reject** table. For how to define an output table as an **Inner Join Reject** table, see [Lookup Inner Join rejection](#).

You can also use the filter button to decrease the number of rows to be searched and improve the performance (in Java).

Related topics:

- [Lookup Inner Join rejection](#)
- [How to filter an input flow](#)

#### 6.2.1.4. How to use the All Rows option

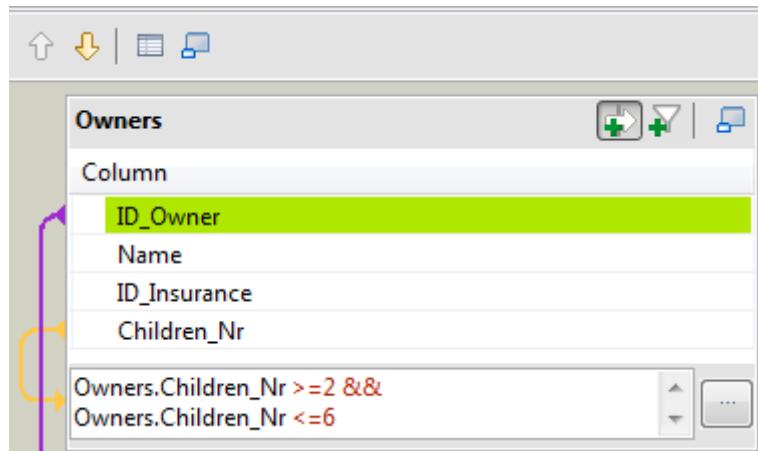
By default, without a Join set up, in each input table of the input area of the **Map Editor**, the **All rows** match model option is selected. This **All rows** option means that all the rows are loaded from the **Lookup** flow and searched against the **Main** flow.

The output corresponds to the Cartesian product of both table (or more tables if need be).

 If you create an explicit or an inner Join between two tables, the **All rows** option is no longer available. You then have to select **Unique match**, **First match** or **All matches**. For more information, see [How to use Explicit Join](#) and [How to use Inner Join](#).

### 6.2.1.5. How to filter an input flow

Click the **Filter** button next to the **tMap settings** button to add a **Filter** field.



In the **Filter** field, type in the condition to be applied. This allows to reduce the number of rows parsed against the main flow, enhancing the performance on long and heterogeneous flows.

You can use the Auto-completion tool via the **Ctrl+Space bar** keystrokes in order to reuse schema columns in the condition statement.

### 6.2.1.6. How to remove input entries from table

To remove input entries, click the red cross sign on the Schema Editor of the selected table. Press **Ctrl** or **Shift** and click fields for multiple selection to be removed.

 If you remove Input entries from the **Map Editor** schema, this removal also occurs in your component schema definition.

## 6.2.2. Mapping variables

The **Var** table (variable table) regroups all mapping variables which are used numerous times in various places.

You can also use the **Expression** field of the **Var** table to carry out any transformation you want to, using Java Code.

Variables help you save processing time and avoid you to retype many times the same data.

| Var                              | +      | X        | Up | Down |  | Schema |
|----------------------------------|--------|----------|----|------|--|--------|
| Expression                       | Type   | Variable |    |      |  |        |
| StringHandling.UPCASE(Cars.Make) | String | var1     |    |      |  |        |

There are various possibilities to create variables:

- Type in freely your variables in Java. Enter the strings between quotes or concatenate functions using the relevant operator.
- Add new lines using the plus sign and remove lines using the red cross sign. And press **Ctrl+Space** to retrieve existing global and context variables.
- Drop one or more **Input** entries to the **Var** table.

| Var         |        |          |
|-------------|--------|----------|
| Expression  | Type   | Variable |
| StringH     | String | var1     |
| Color       |        |          |
| ID_Reseller |        |          |

Select an entry on the Input area or press Shift key to select multiple entries of one Input table.

Press **Ctrl** to select either non-appended entries in the same input table or entries from various tables. When selecting entries in the second table, notice that the first selection displays in grey. Hold the **Ctrl** key down to drag all entries together. A tooltip shows you how many entries are in selection.

Then various types of drag-and-drops are possible depending on the action you want to carry out.

| To...                                                                                              | You need to...                                                                                                                                                                                                                         |
|----------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Insert all selected entries as separated variables.                                                | Simply drag & drop to the Var table. Arrows show you where the new Var entry can be inserted. Each Input is inserted in a separate cell.                                                                                               |
| Concatenate all selected input entries together with an existing Var entry.                        | Drag & drop onto the Var entry which gets highlighted. All entries gets concatenated into one cell. Add the required operators using Java operations signs. The dot concatenates string variables.                                     |
| Overwrite a Var entry with selected concatenated Input entries.                                    | Drag & drop onto the relevant Var entry which gets highlighted then press <b>Ctrl</b> and release. All selected entries are concatenated and overwrite the highlighted Var.                                                            |
| Concatenate selected input entries with highlighted Var entries and create new Var lines if needed | Drag & drop onto an existing Var then press <b>Shift</b> when browsing over the chosen Var entries. First entries get concatenated with the highlighted Var entries. And if necessary new lines get created to hold remaining entries. |

### 6.2.2.1. How to access global or context variables

Press **Ctrl+Space** to access the global and context variable list.

Appended to the variable list, a metadata list provides information about the selected column.

### 6.2.2.2. How to remove variables

To remove a selected **Var** entry, click the red cross sign. This removes the whole line as well as the link.

Press **Ctrl** or **Shift** and click fields for multiple selection then click the red cross sign.

## 6.2.3. Working with expressions

All expressions (**Input**, **Var** or **Output**) and constraint statements can be viewed and edited directly in the expression fields, in the expression editor, and in the Expression Builder.

### 6.2.3.1. How to access the expression editor

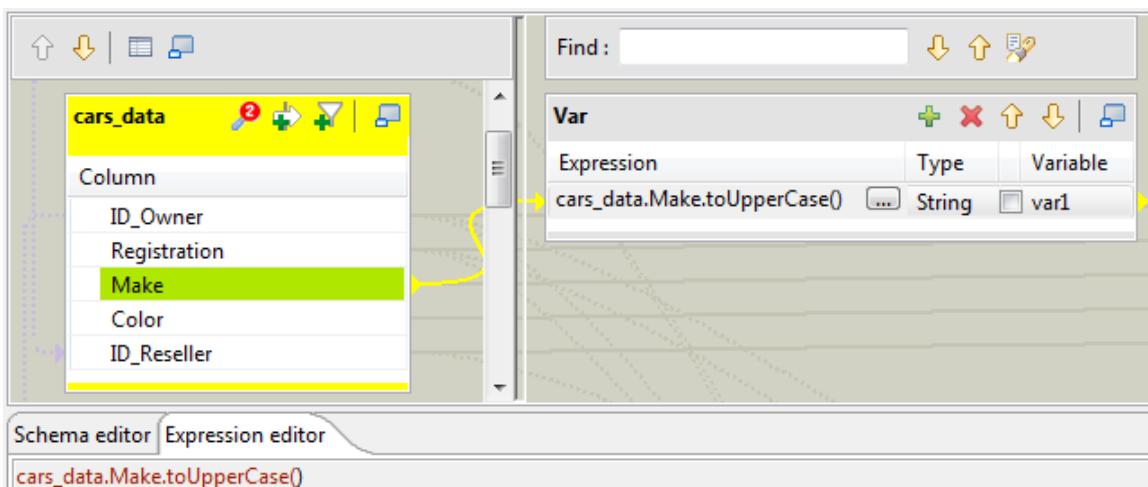
The expression editor provides visual comfort to write any function or transformation in a handy dedicated view.

You can write the expressions necessary for the data transformation directly in the **Expression editor** view located in the lower half of the expression editor.

To open the **Expression editor** view, complete the following:

1. Double-click the **tMap** component in your Job design to open the **Map Editor**.
2. In the lower half of the editor, click the **Expression editor** tab to open the corresponding view.

 To edit an expression, select it in the **Input** panel and then click the **Expression editor** tab and modify the expression as required.



3. Enter the Java code according to your needs. The corresponding expression in the output panel is synchronized.



Refer to the Java documentation for more information regarding functions and operations.

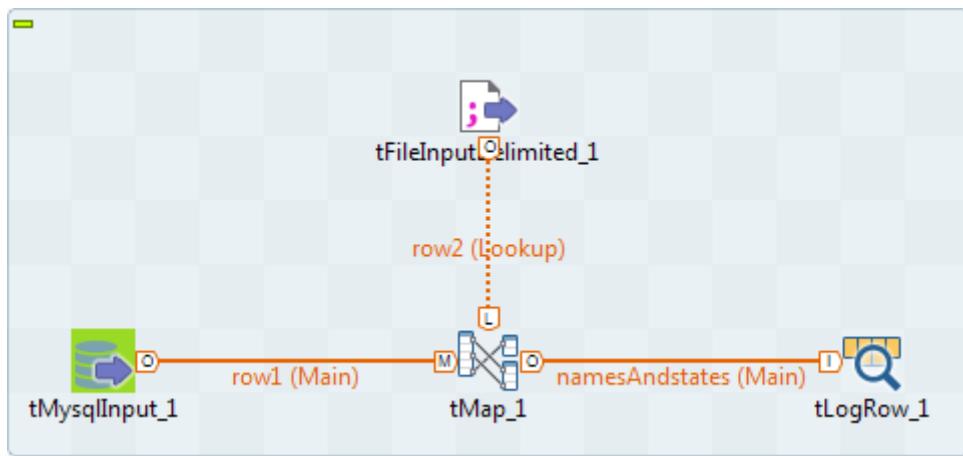
### 6.2.3.2. How to write code using the Expression Builder

Some Jobs require pieces of code to be written in order to provide components with parameters. In the **Component** view of some components, an **Expression Builder** interface can help you write such pieces of code (in Java), known as expressions.

Using the Expression Builder of **tMap**, you can edit the expression for an input column, an output column, or a variable, or change the expressions for multiple output columns at the same time.

#### Editing individual expressions

The following example shows how to use the **Expression Builder** to edit two individual expressions.

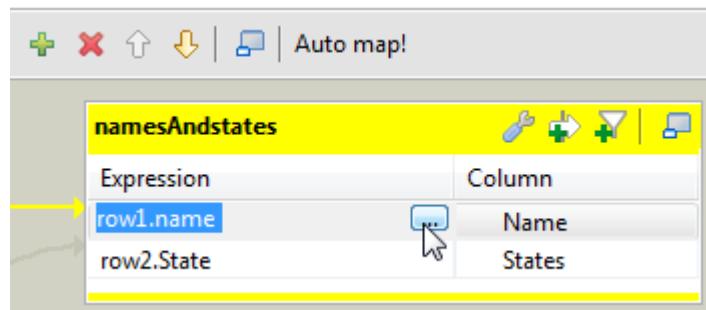


In this example, two input flows are connected to the **tMap** component.

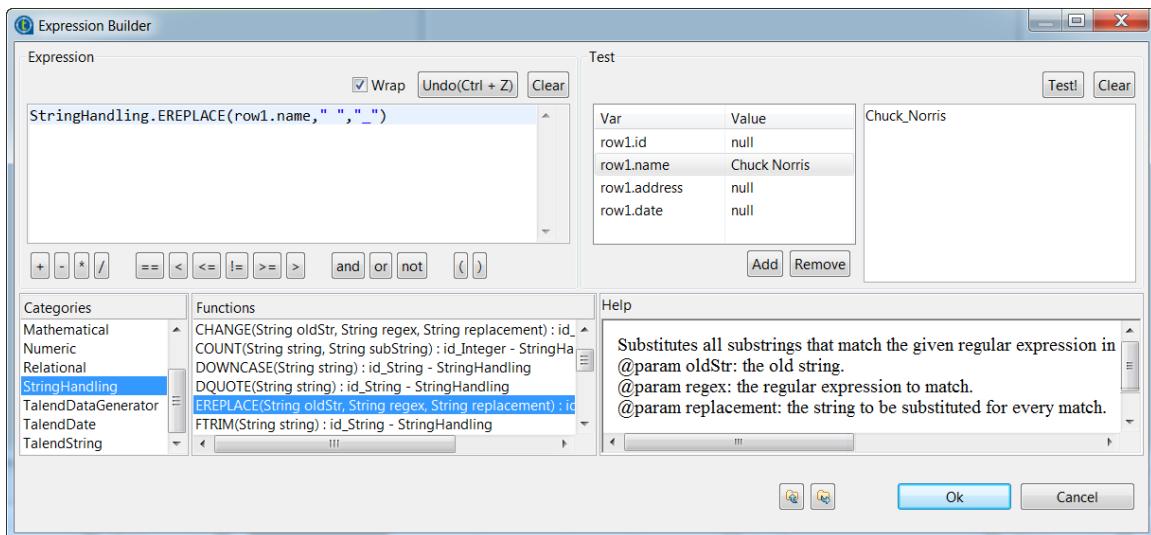
- From the DB input, comes a list of names made of a first name and a last name separated by a space char.
- From the File input, comes a list of US states, in lower case.

In the **tMap**, use the expression builder to: First, replace the blank char separating the first and last names with an underscore char, and second, change the states from lower case to upper case.

1. In the **tMap**, set the relevant inner join to set the reference mapping. For more information regarding **tMap**, see [tMap operation](#) and [Map editor interfaces](#).
2. From the main (*row1*) input, drop the *Names* column to the output area, and the *State* column from the lookup (*row2*) input towards the same output area.
3. Click in the first **Expression** field (*row1.Name*), and then click the [...] button that appears next to the expression.



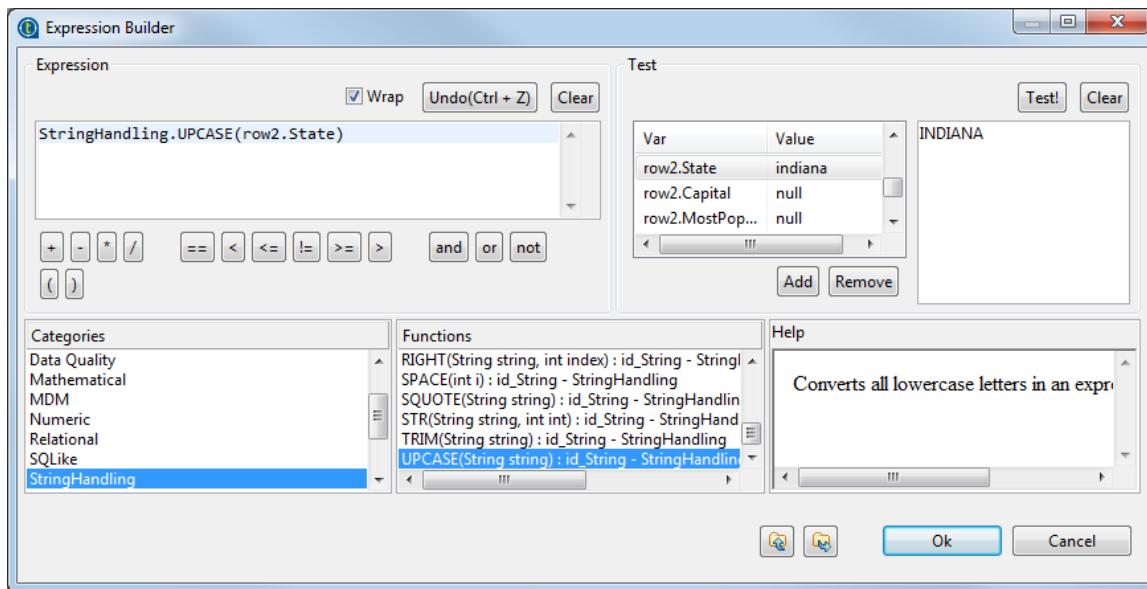
The **[Expression Builder]** dialog box opens up.



4. In the **Category** area, select the relevant action you want to perform. In this example, select **StringHandling** and select the **EREPLACE** function.
5. In the **Expression** area, paste *row1.Name* in place of the text expression, in order to get: **StringHandling.EREPLACE(row1.Name, " ", "\_")**. This expression will replace the separating space char with an underscore char in the char string given.

Note that the **CHANGE** and **EREPLACE** functions in the **StringHandling** category are used to substitute all substrings that match the given regular expression in the given old string with the given replacement and returns a new string. Their three parameters are:

- **oldStr**: the old string.
  - **newStr**: the regular expression to match.
  - **replacement**: the string to be substituted for every match.
6. Now check that the output is correct, by typing in the relevant **Value** field of the **Test** area, a dummy value, e.g: *Chuck Norris* and clicking **Test!**. The correct change should be carried out, for example, *Chuck\_Norris*.
  7. Click **OK** to validate the changes, and then proceed with the same operation for the second column (*State*).
  8. In the **tMap** output, select the *row2.State* Expression and click the [...] button to open the **Expression builder** again.



This time, the `StringHandling` function to be used is `UPCASE`. The complete expression says: `StringHandling.UPCASE(row2.State)`.

- Once again, check that the expression syntax is correct using a dummy **Value** in the **Test** area, for example *indiana*. The **Test!** result should display *INDIANA* for this example. Then, click **OK** to validate the changes.

Both expressions are now displayed in the **tMap Expression** field.

| Expression                                                | Column |
|-----------------------------------------------------------|--------|
| <code>StringHandling.EREPLACE(row1.name, " ", "_")</code> | Name   |
| <code>StringHandling.UPCASE(row2.State)</code>            | States |

These changes will be carried out along the flow processing. The output of this example is as shown below.

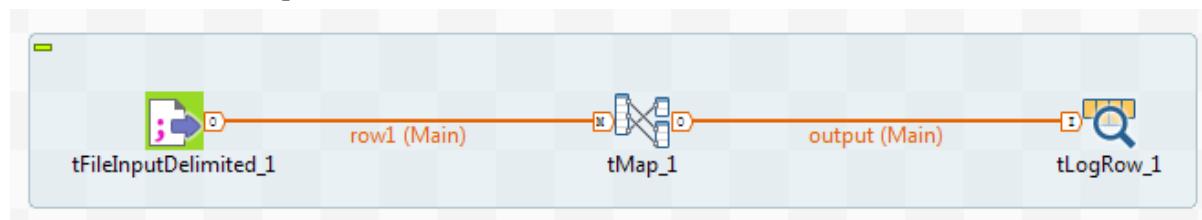
*Starting job NamesAndStates at 10:02 10/10/2007.*

| tLogRow_1          |               |
|--------------------|---------------|
| Name               | RandomStates  |
| William_Grant      | IOWA          |
| William_Hoover     | NEW YORK      |
| Grover_Lincoln     | NORTH DAKOTA  |
| Lyndon_Jefferson   | OHIO          |
| Gerald_Hayes       | WASHINGTON    |
| Benjamin_Grant     | MAINE         |
| George_Pierce      | CONNECTICUT   |
| Jimmy_Reagan       | ALASKA        |
| Martin_Hayes       | WASHINGTON    |
| Franklin_Jefferson | IOWA          |
| Andrew_Nixon       | NEW HAMPSHIRE |

## Setting expressions for multiple output columns simultaneously

**tMap** allows you to define the transformation behavior for multiple output columns at the same time.

Using a simple transformation Job, the following example shows how to define expressions on multiple columns in a batch manner in **tMap**.



Here is the content of the input CSV file used in this example:

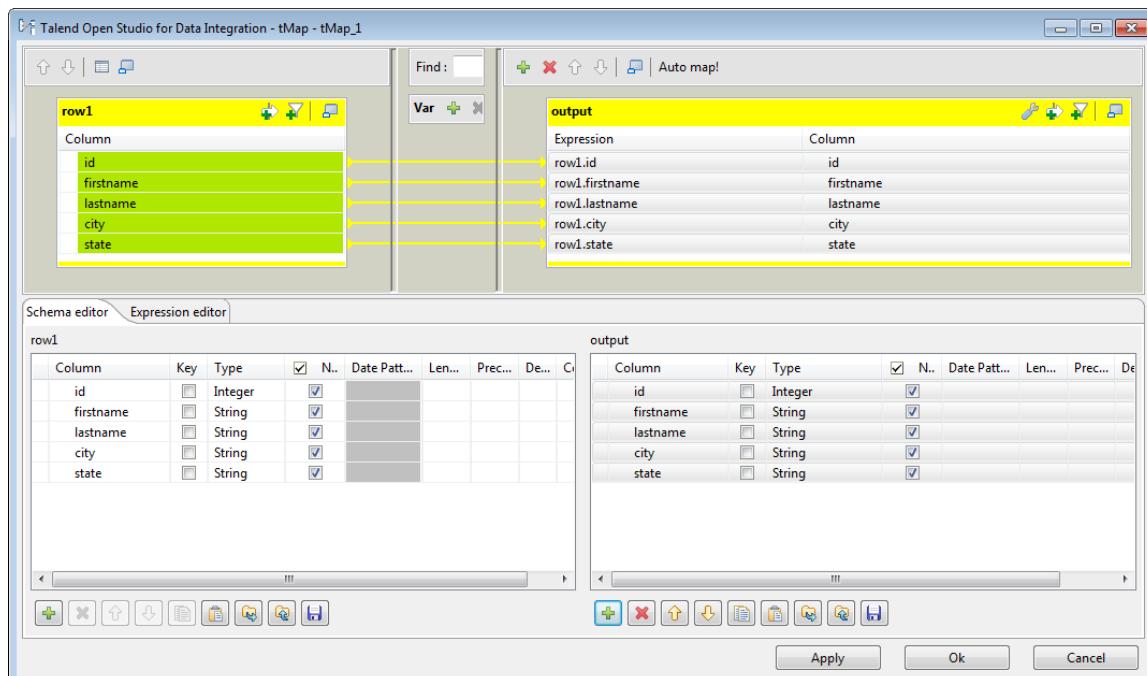
```

id;firstname;lastname;city;state
1; Andrew; Adams; Madison; Rhode Island
2; Andrew; Garfield; Saint Paul; Colorado
3; Woodrow; Eisenhower ; Juneau; New Hampshire
4; Woodrow; Jackson; Denver; Maine
5; Lyndon; Buchanan; Pierre; Kentucky
6; Bill; Tyler; Helena; New York
7; George; Adams; Oklahoma City ; Alaska
8; Ulysses; Garfield; Santa Fe; Massachusetts
9; Thomas; Coolidge ; Charleston; Mississippi
10; John; Polk; Carson City; Louisiana

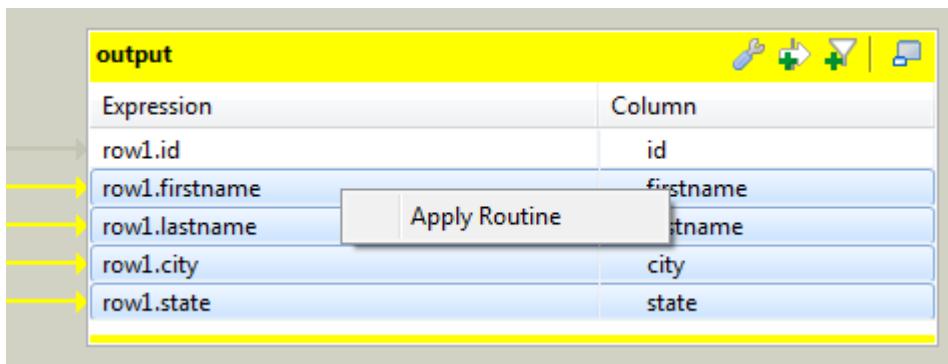
```

In this example, all the output columns of type String will be trimmed to remove preceding and trailing whitespace and the last names and state names will be transformed to upper case.

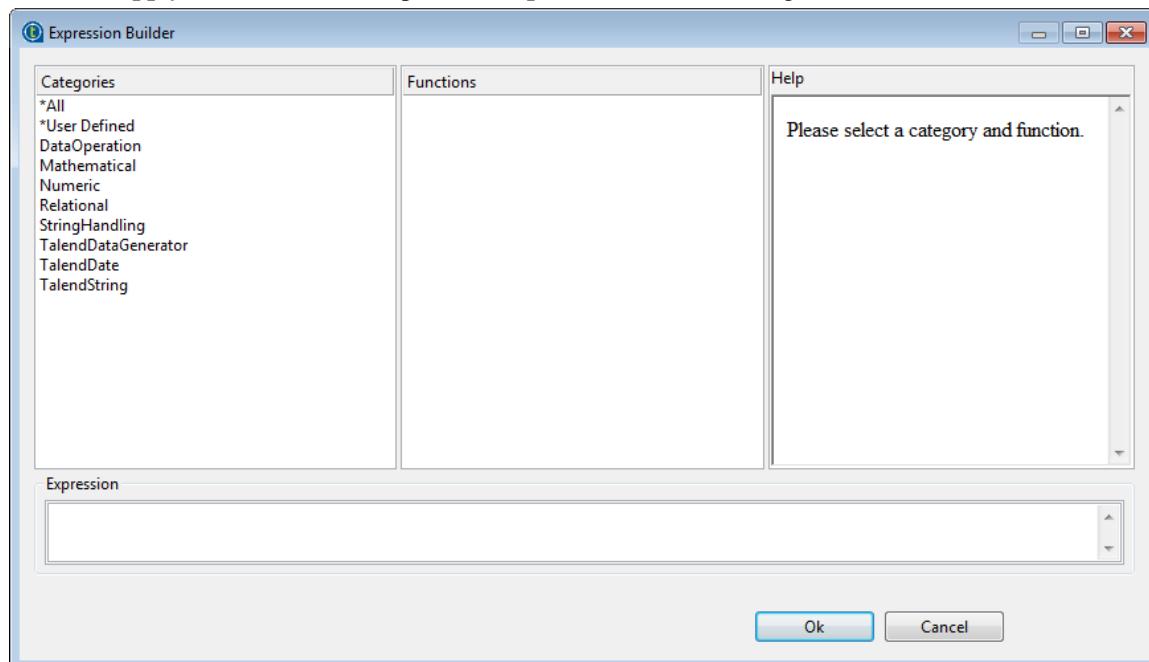
1. In the Map Editor, complete the input-output mappings.



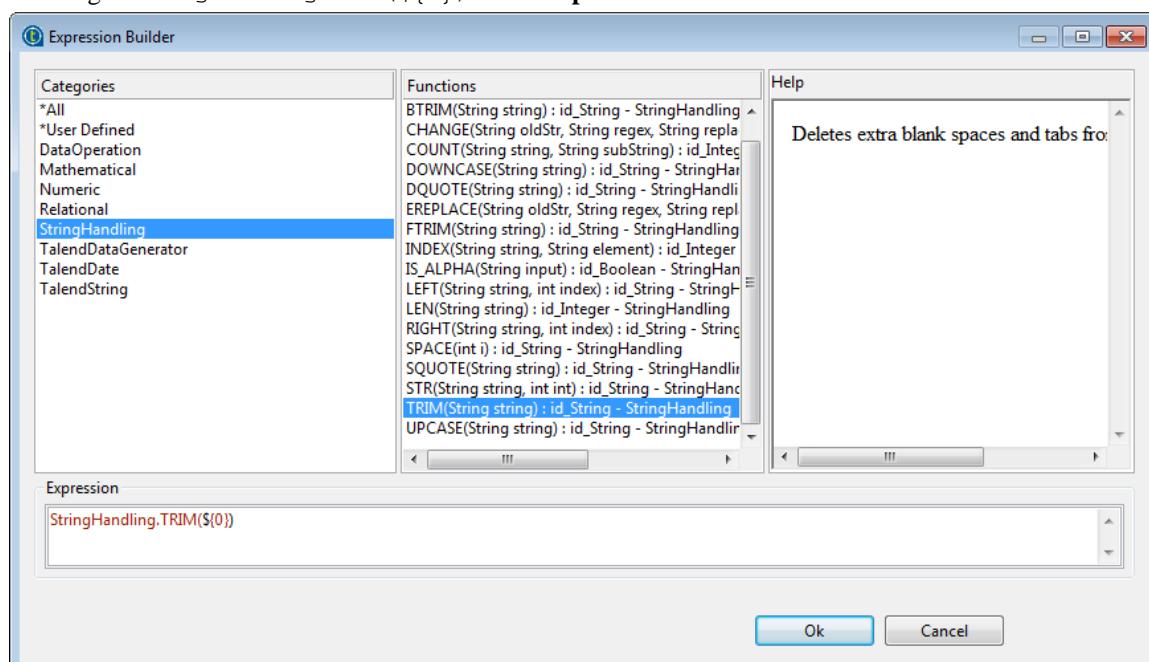
2. Select the columns of type String in the output table, namely *firstname*, *lastname*, *city*, and *state* in this example, and right-click the selection so that the **Apply Routine** button shows up.



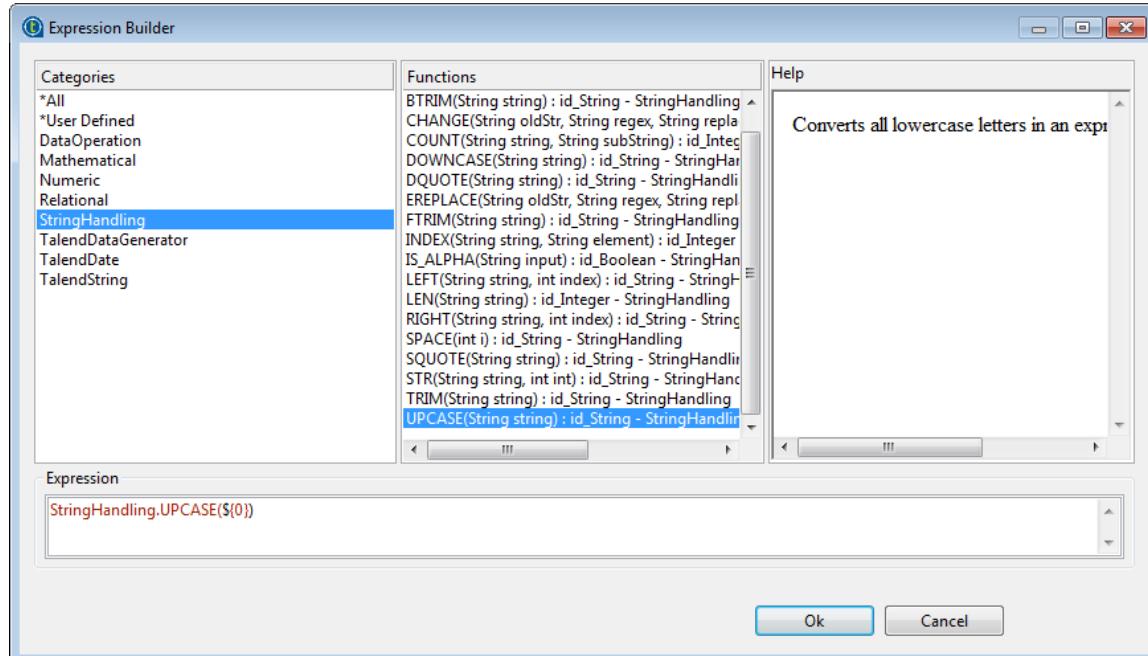
3. Click the **Apply Routine** button to open the **[Expression Builder]** dialog box.



4. Select **StringHandling** in the **Categories** area, and then double-click the **TRIM** function in the **Functions** area to get **StringHandling.TRIM(\${0})** in the **Expression** field.



5. Click **OK** to close the **[Expression Builder]** dialog box.
6. Select the *lastname* and *state* columns in the output table of the Map Editor, right-click the selection, and then click the **Apply Routine** button to open the **[Expression Builder]** dialog box.
7. Select **StringHandling** in the **Categories** area, and then double-click the **UPPERCASE** function in the **Functions** area to get **StringHandling.UPCASE(\${0})** in the **Expression** field.



8. Click **OK** to close the **[Expression Builder]** dialog box.

Now the expressions on those output columns look like below:

| output                                                     |           |
|------------------------------------------------------------|-----------|
| Expression                                                 | Column    |
| row1.id                                                    | id        |
| StringHandling.TRIM(row1.firstname )                       | firstname |
| StringHandling.UPCASE(StringHandling.TRIM(row1.lastname )) | lastname  |
| StringHandling.TRIM(row1.city )                            | city      |
| StringHandling.UPCASE(StringHandling.TRIM(row1.state ))    | state     |

The functions will be carried out along the flow processing. The output of this example is as shown below.

Execution

[statistics] connected

| tLogRow_1 |    |           |            |               |               |
|-----------|----|-----------|------------|---------------|---------------|
| =         | id | firstname | lastname   | city          | state         |
| =         | 1  | Andrew    | ADAMS      | Madison       | RHODE ISLAND  |
| =         | 2  | Andrew    | GARFIELD   | Saint Paul    | COLORADO      |
| =         | 3  | Woodrow   | EISENHOWER | Juneau        | NEW HAMPSHIRE |
| =         | 4  | Woodrow   | JACKSON    | Denver        | MAINE         |
| =         | 5  | Lyndon    | BUCHANAN   | Pierre        | KENTUCKY      |
| =         | 6  | Bill      | TYLER      | Helena        | NEW YORK      |
| =         | 7  | George    | ADAMS      | Oklahoma City | ALASKA        |
| =         | 8  | Ulysses   | GARFIELD   | Santa Fe      | MASSACHUSETTS |
| =         | 9  | Thomas    | COOLIDGE   | Charleston    | MISSISSIPPI   |
| =         | 10 | John      | POLK       | Carson City   | LOUISIANA     |

[statistics] disconnected

## 6.2.4. Mapping the Output setting

On the design workspace, the creation of a **Row** connection from the **tMap** component to the output components adds Output schema tables in the **Map Editor**.

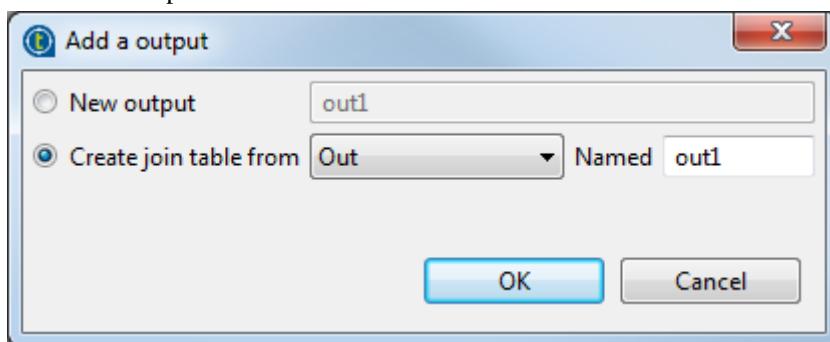
You can also add an Output schema in your **Map Editor**, using the plus sign from the tool bar of the Output area.

You have as well the possibility to create a join between your output tables. The join on the tables enables you to process several flows separately and unite them in a single output.



The join table retrieves the schema of the source table.

When you click the **[+]** button to add an output schema or to make a join between your output tables, a dialog box opens. You have then two options.



| Select...                     | To...                                                                                                                                                                                                          |
|-------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>New output</b>             | Add an independent table.                                                                                                                                                                                      |
| <b>Create join table from</b> | Create a join between output tables. In order to do so, select in the drop down list the table from which you want to create the join. In the <b>Named</b> field, type in the name of the table to be created. |

Unlike the **Input** area, the order of output schema tables does not make such a difference, as there is no subordination relationship between outputs (of Join type).

Once all connections, hence output schema tables, are created, you can select and organize the output data via drag & drops.

You can drop one or several entries from the **Input** area straight to the relevant output table.

Press **Ctrl** or **Shift**, and click entries to carry out multiple selection.

Or you can drag expressions from the **Var** area and drop them to fill in the output schemas with the appropriate reusable data.

Note that if you make any change to the Input column in the Schema Editor, a dialog prompts you to decide to propagate the changes throughout all Input/Variable/Output table entries, where concerned.

| Action                                 | Result                                                                                     |
|----------------------------------------|--------------------------------------------------------------------------------------------|
| Drag & Drop onto existing expressions. | Concatenates the selected expression with the existing expressions.                        |
| Drag & Drop to insertion line.         | Inserts one or several new entries at start or end of table or between two existing lines. |
| Drag & Drop + Ctrl.                    | Replaces highlighted expression with selected expression.                                  |
| Drag & Drop + Shift.                   | Adds the selected fields to all highlighted expressions. Inserts new lines if needed.      |
| Drag & Drop + Ctrl + Shift.            | Replaces all highlighted expressions with selected fields. Inserts new lines if needed.    |

You can add filters and rejections to customize your outputs.

### 6.2.4.1. Creating complex expressions

If you have complex expressions to create, or advanced changes to be carried out on the output flow, then the Expression Builder interface can help in this task.

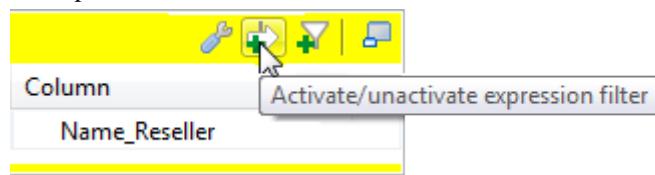
Click the **Expression** field of your input or output table to display the [...] button. Then click this three-dot button to open the **Expression Builder**.

For more information regarding the Expression Builder, see [How to write code using the Expression Builder](#).

### 6.2.4.2. Filters

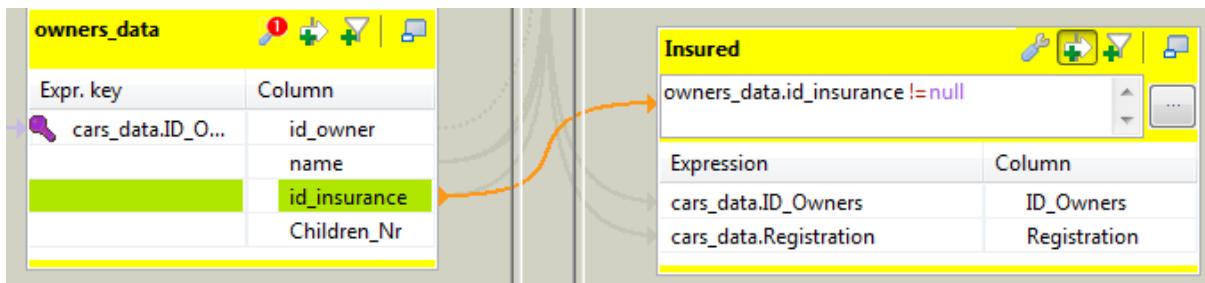
Filters allow you to make a selection among the input fields, and send only the selected fields to various outputs.

Click the  button at the top of the table to add a filter line.



You can enter freely your filter statements using Java operators and functions.

Drop expressions from the **Input** area or from the **Var** area to the Filter row entry of the relevant Output table.



An orange link is then created. Add the required Java operator to finalize your filter formula.

You can create various filters on different lines. The AND operator is the logical conjunction of all stated filters.

### 6.2.4.3. Output rejection

Reject options define the nature of an output table.

It groups data which do not satisfy one or more filters defined in the standard output tables. Note that as standard output tables, are meant all non-reject tables.

This way, data rejected from other output tables, are gathered in one or more dedicated tables, allowing you to spot any error or unpredicted case.

The Reject principle concatenates all non Reject tables filters and defines them as an ELSE statement.

To define an output table as the Else part of the regular tables:

1. Click the **tMap settings** button at the top of the output table to display the table properties.
2. Click in the **Value** field corresponding to **Catch output reject** and then click the [...] button that appears to display the [Options] dialog box.
3. In the [Options] dialog box, double-click **true**, or select it and click **OK** to validate the setting and close the dialog box.

The screenshot shows the 'Rejected\_data' table properties and the 'Options' dialog box. The 'Rejected\_data' table has a 'Catch output reject' value of 'true'. The 'Options' dialog box shows 'true' selected. The table also has columns: Expression (Column), cars\_data.ID\_Owners (ID\_Owner), cars\_data.Registration (Registration), cars\_data.ID\_Reseller (ID\_Reseller), and owners\_data.Name (Name). The Schema Type is set to Built-In.

You can define several Reject tables, to offer multiple refined outputs. To differentiate various Reject outputs, add filter lines, by clicking on the plus arrow button.

Once a table is defined as Reject, the verification process will be first enforced on regular tables before taking in consideration possible constraints of the Reject tables.

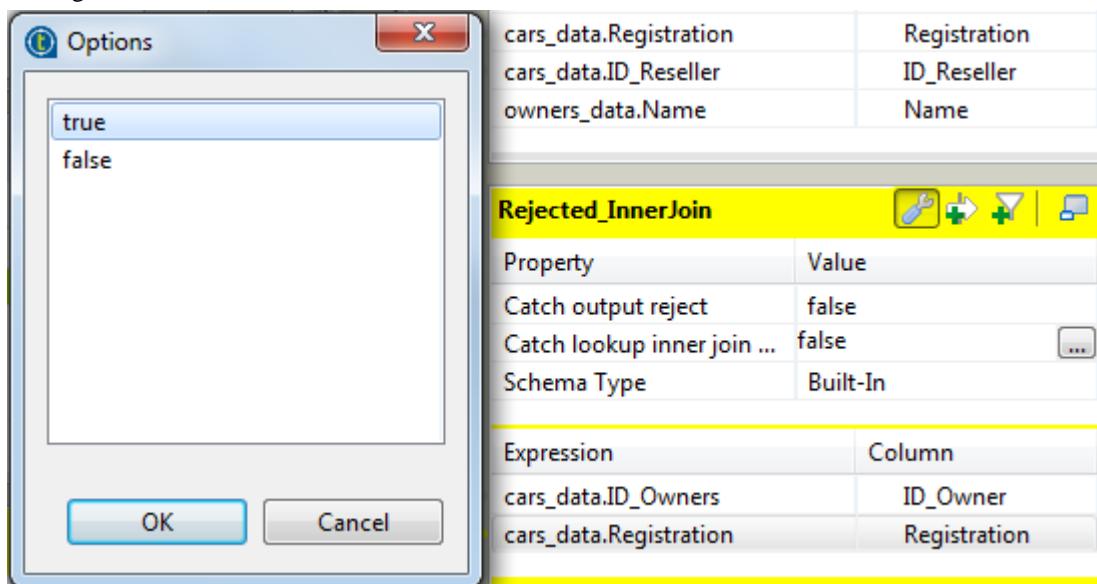
Note that data are not exclusively processed to one output. Although a data satisfied one constraint, hence is routed to the corresponding output, this data still gets checked against the other constraints and can be routed to other outputs.

#### 6.2.4.4. Lookup Inner Join rejection

The Inner Join is a Lookup Join. The Inner Join Reject table is a particular type of Rejection output. It gathers rejected data from the main row table after an Inner Join could not be established.

To define an Output flow as container for rejected Inner Join data, create a new output component on your Job that you connect to the **Map Editor**. Then in the **Map Editor**, follow the steps below:

1. Click the **tMap settings** button at the top of the output table to display the table properties.
2. Click in the **Value** field corresponding to **Catch lookup inner join reject** and then click the [...] button that appears to display the **[Options]** dialog box.
3. In the **[Options]** dialog box, double-click **true**, or select it and click **OK** to validate the setting and close the dialog box.



#### 6.2.4.5. Removing Output entries

To remove Output entries, click the cross sign on the Schema Editor of the selected table.

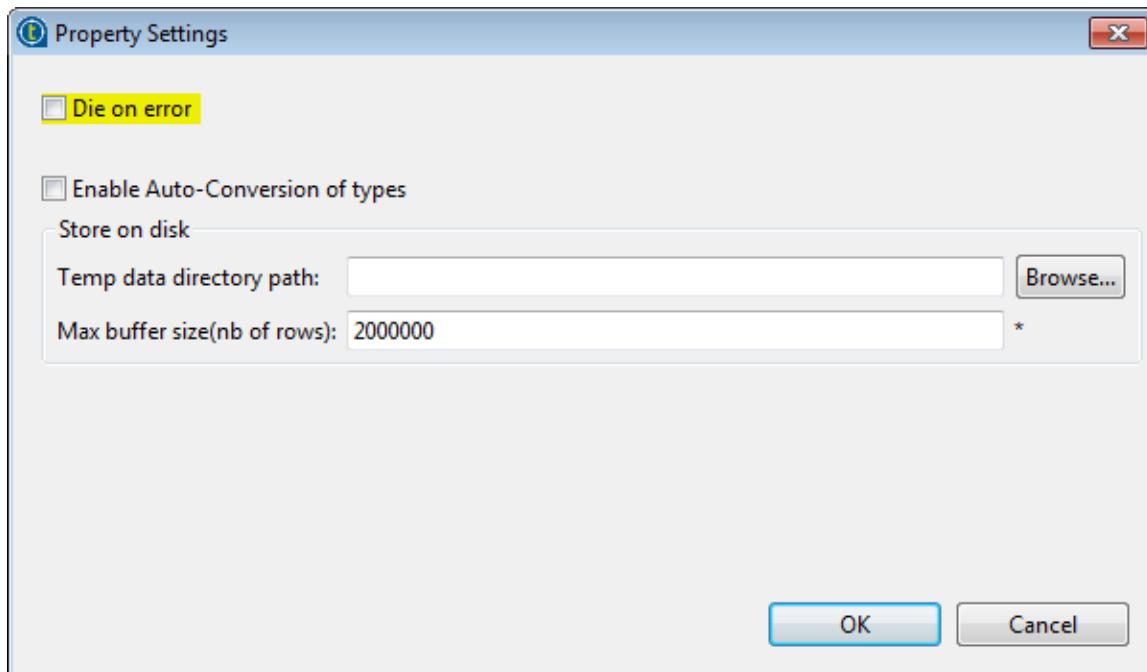
#### 6.2.4.6. Handling errors

The **Die on error** option prevents error to be processed. To do so, it stops the Job execution as soon as an error is encountered. The **tMap** component provides this option to prevent processing erroneous data. The **Die on error** option is activated by default in **tMap**.

Deactivating the **Die on error** option will allow you to skip the rows on error and complete the process for error-free rows on one hand, and to retrieve the rows on error and manage them if needed.

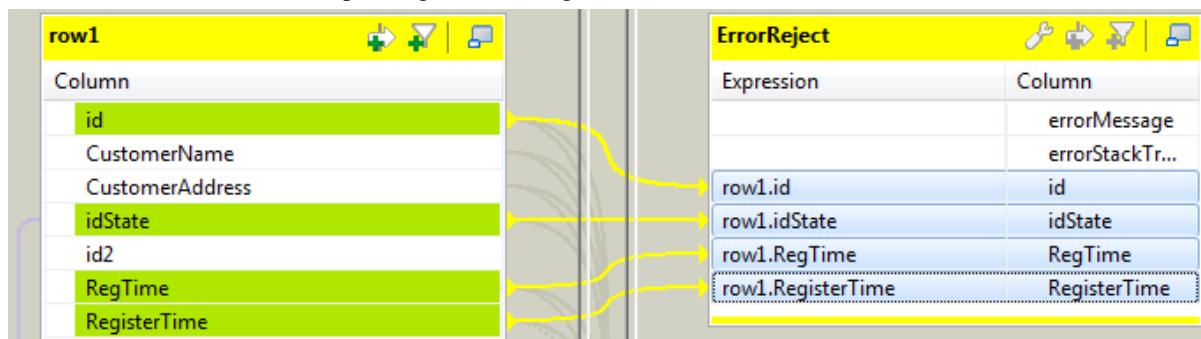
To deactivate the **Die on error** option:

1. Double-click the **tMap** component on the design workspace to open the **Map Editor**.
2. Click the **Property Settings** button at the top of the input area to display the **[Property Settings]** dialog box.
3. In **[Property Settings]** dialog box, clear the **Die on error** check box and click **OK**.

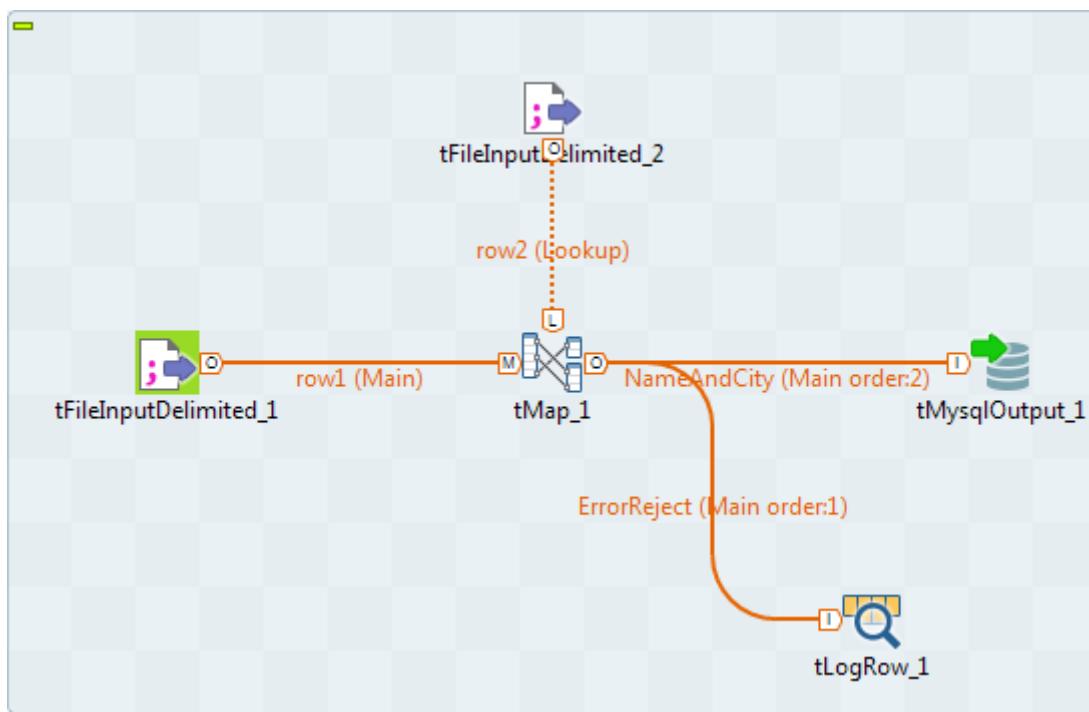


A new table called **ErrorReject** appears in the output area of the **Map Editor**. This output table automatically comprises two columns: **errorMessage** and **errorStackTrace**, retrieving the message and stack trace of the error encountered during the Job execution. Errors can be unparseable dates, null pointer exceptions, conversion issues, etc.

You can also drag and drop columns from the input tables to this error reject output table. Those erroneous data can be retrieved with the corresponding error messages and thus be corrected afterward.



Once the error reject table is set, its corresponding flow can be sent to an output component.



To do so, on the design workspace, right-click the **tMap** component, select **Row > ErrorReject** in the menu, and click the corresponding output component, here **tLogRow**.

When you execute the Job, errors are retrieved by the **ErrorReject** flow.

```

Starting job Die_on_error at 17:30 01/09/2010.

java.text.ParseException: Unparseable date: "08 01
1980" | java.lang.RuntimeException:
java.text.ParseException: Unparseable date: "08 01 1980"
 at routines.TalendDate.parseDate(TalendDate.java:503)
 at
doc.die_on_error_0_1.Die_on_error.tFileInputDelimited_2Pro
cess(Die_on_error.java:1409)
 at
doc.die_on_error_0_1.Die_on_error.runJobInTOS(Die_on_error.
java:2262)
 at
doc.die_on_error_0_1.Die_on_error.main(Die_on_error.java:2
160)
Caused by: java.text.ParseException: Unparseable date: "08
01 1980"
 at java.text.DateFormat.parse(Unknown Source)
 at routines.TalendDate.parseDate(TalendDate.java:501)
 ... 3 more
|1|08 01 1980
Job Die_on_error ended at 17:30 01/09/2010. [exit code=0]

```

The result contains the error message, its stack trace, and the two columns, *id* and *date*, dragged and dropped to the **ErrorReject** table, separated by a pipe "|".

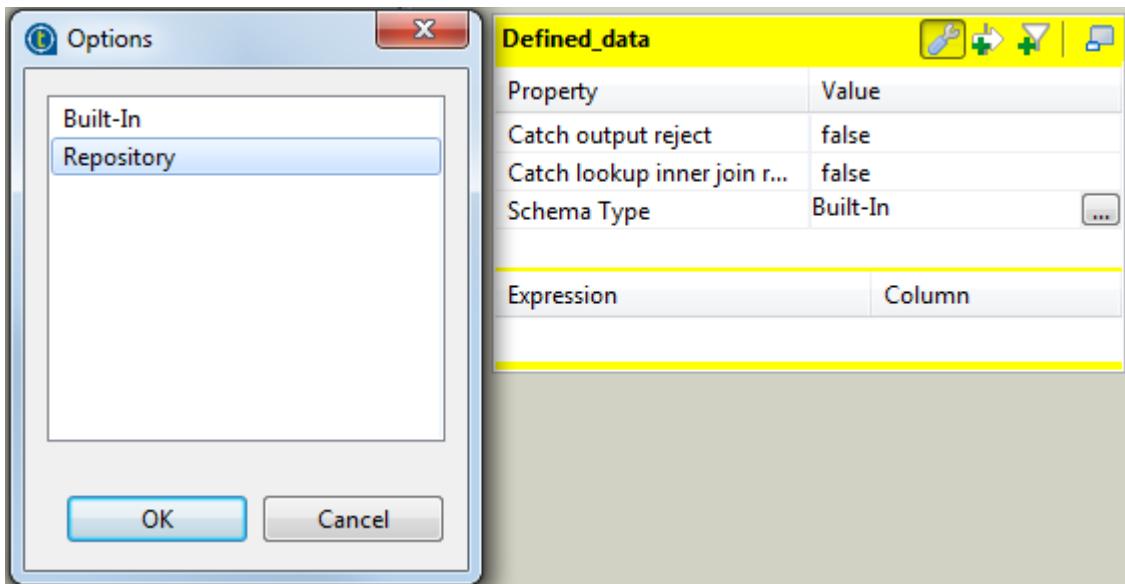
## 6.2.5. Setting schemas in the Map Editor

In the **Map Editor**, you can define the type of a table schema as **Built-In** so that you can modify the data structure in the **Schema editor** panel, or **Repository** and retrieve the data structure from the Repository. By default, the schema type is set to **Built-In** for all tables.

### 6.2.5.1. Retrieving the schema structure from the Repository

To retrieve the schema structure of the selected table from the Repository:

1. Click the **tMap Settings** button at the top of the table to display the table properties.
2. Click in the **Value** field of **Schema Type**, and then click the three-dot button that appears to open the **[Options]** dialog box.



3. In the **[Options]** dialog box, double-click **Repository**, or select it and click **OK**, to close the dialog box and display the **Schema Id** property beneath **Schema Type**.



If you close the **Map Editor** now without specifying a Repository schema item, the schema type changes back to **Built-In**.

4. Click in the **Value** field of **Schema Id**, and then click the [...] button that appears to display the **[Repository Content]** dialog box.
5. In the **[Repository Content]** dialog box, select your schema as you define a centrally stored schema for any component, and then click **OK**.

The **Value** field of **Schema Id** is filled with the schema you just selected, and everything in the **Schema editor** panel for this table becomes read-only.

| Defined_data                   |  |                         |  |  |  |  |  |  |  |
|--------------------------------|--|-------------------------|--|--|--|--|--|--|--|
| Property                       |  | Value                   |  |  |  |  |  |  |  |
| Catch output reject            |  | false                   |  |  |  |  |  |  |  |
| Catch lookup inner join reject |  | false                   |  |  |  |  |  |  |  |
| Schema Type                    |  | Repository              |  |  |  |  |  |  |  |
| Schema Id                      |  | DELIM:owners - metadata |  |  |  |  |  |  |  |
| Defined_data                   |  |                         |  |  |  |  |  |  |  |
| Expression                     |  | Column                  |  |  |  |  |  |  |  |
|                                |  | ID_Owner                |  |  |  |  |  |  |  |
|                                |  | Name                    |  |  |  |  |  |  |  |
|                                |  | ID_Insurance            |  |  |  |  |  |  |  |
|                                |  | Children_Nr             |  |  |  |  |  |  |  |

| Column      | Key                      | T... | <input checked="" type="checkbox"/> | N.. | Date ... | L... | Pr... | D... | C... |
|-------------|--------------------------|------|-------------------------------------|-----|----------|------|-------|------|------|
| ID_Owner    | <input type="checkbox"/> | I... | <input checked="" type="checkbox"/> |     |          | 2    | 0     |      |      |
| Name        | <input type="checkbox"/> | S... | <input checked="" type="checkbox"/> |     |          | 19   | 0     |      |      |
| ID_Insur... | <input type="checkbox"/> | S... | <input checked="" type="checkbox"/> |     |          | 3    | 0     |      |      |
| Childre...  | <input type="checkbox"/> | I... | <input checked="" type="checkbox"/> |     |          | 2    | 0     |      |      |



*Changing the schema type of the subordinate table across a Join from **Built-In** to **Repository** causes the Join to get lost.*



Changes to the schema of a table made in the **Map Editor** are automatically synchronized to the schema of the corresponding component connected with the **tMap** component.

### 6.2.5.2. Searching schema columns

The schema column filter of **tMap** allows you to quickly search an input or output schema column or multiple columns among hundreds of them in one go.

The following example shows how to find columns containing the string "customer" in the output table of the Map Editor.

1. Open the Map Editor, and click the button at the top of the table to open the filter area.

| Expression            | Column           |
|-----------------------|------------------|
| row2.id               | id               |
| row1.CustomerName     | customerName     |
| row2.age              | age              |
| row1.CustomerAddress  | customerAddress  |
| row2.city             | city             |
| row1.idState          | idState          |
| row2.carModel         | carModel         |
| row2.madeIn           | madeIn           |
| row2.dealer           | dealer           |
| row2.shopName         | shopName         |
| row2.discount         | discount         |
| row2.totalPrice       | totalPrice       |
| row2.salesManager     | salesManager     |
| row2.insuranceCompany | insuranceCompany |
| row2.insureTvinc      | insureTvinc      |

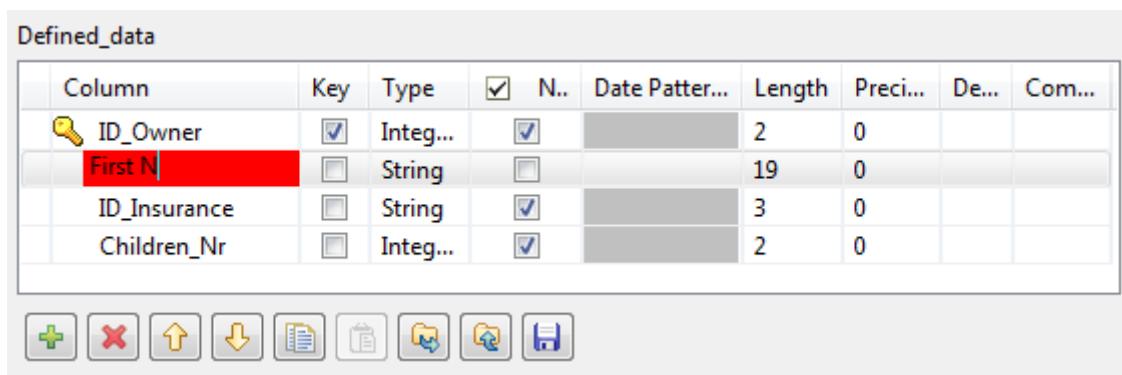
2. In the filter area, type in your search string, *customer* in this example.

As you start to type, the table displays the columns that match the characters.

| Expression                  | Column                 |
|-----------------------------|------------------------|
| row1.CustomerName           | customerName           |
| row1.CustomerAddress        | customerAddress        |
| row2.customerServiceHistory | customerServiceHistory |

### 6.2.5.3. Using the Schema Editor

The **Schema Editor** details all fields of the selected table. With the schema type of the table set to **Built-In**, you can modify the schema of the table.



Use the tool bar below the schema table, to add, move or remove columns from the schema.

You can also load a schema from the repository or export it into a file.

| Metadata         | Description                                                                                                                                  |
|------------------|----------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Column</b>    | Column name as defined on the <b>Map Editor</b> schemas and on the Input or Output component schemas.                                        |
| <b>Key</b>       | The Key shows if the expression key data should be used to retrieve data through the Join link. If unchecked, the Join relation is disabled. |
| <b>Type</b>      | Type of data: String, Integer, Date, etc.<br><br>💡 This column should always be defined in a Java version.                                   |
| <b>Length</b>    | -1 shows that no length value has been defined in the schema.                                                                                |
| <b>Precision</b> | Defines the number of digits to the right of the decimal point.                                                                              |
| <b>Nullable</b>  | Clear this check box if the field value should not be null.                                                                                  |
| <b>Default</b>   | Shows any default value that may be defined for this field.                                                                                  |
| <b>Comment</b>   | Free text field. Enter any useful comment.                                                                                                   |



Input metadata and output metadata are independent from each other. You can, for instance, change the label of a column on the output side without the column label of the input schema being changed.

However, any change made to the metadata are immediately reflected in the corresponding schema on the **tMap** relevant (Input or Output) area, but also on the schema defined for the component itself on the design workspace.

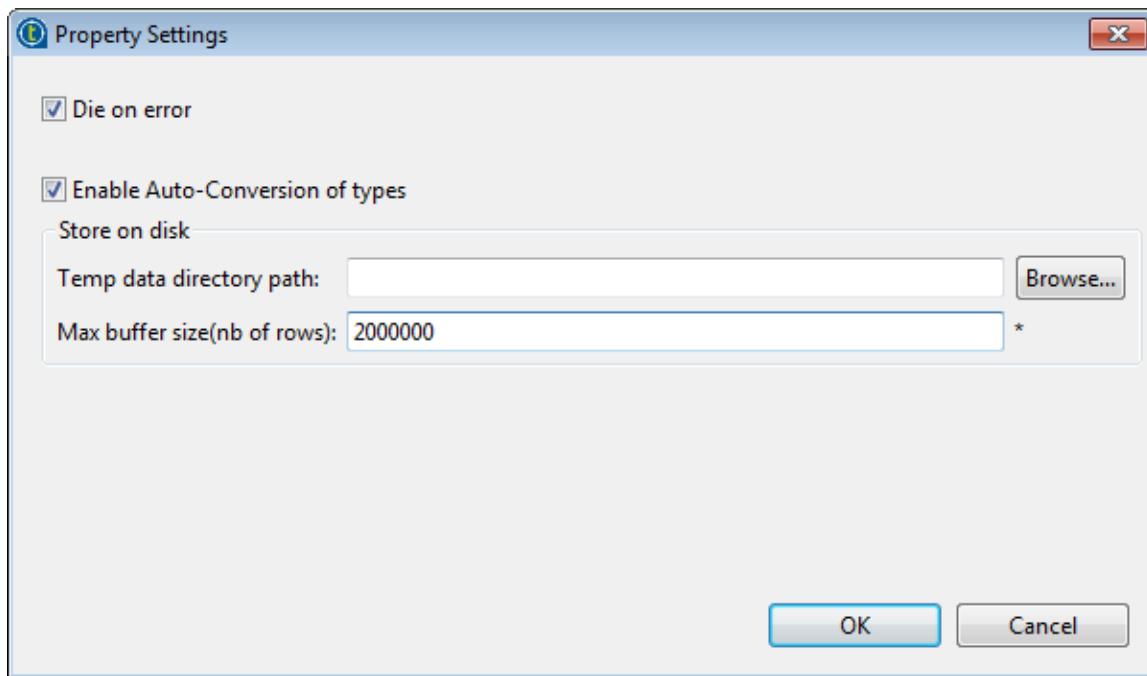
A Red colored background shows that an invalid character has been entered. Most special characters are prohibited in order for the Job to be able to interpret and use the text entered in the code. Authorized characters include lower-case, upper-case, figures except as start character.

## 6.2.6. Enabling automatic data type conversion

When processing data flows using a **tMap**, if the input and output columns across a mapping are of different data types, compiling errors may occur at the Job execution time. The **Enable Auto-Conversion of types** option in the **tMap** helps avoid such errors.

To enable this feature in **tMap** in a Job:

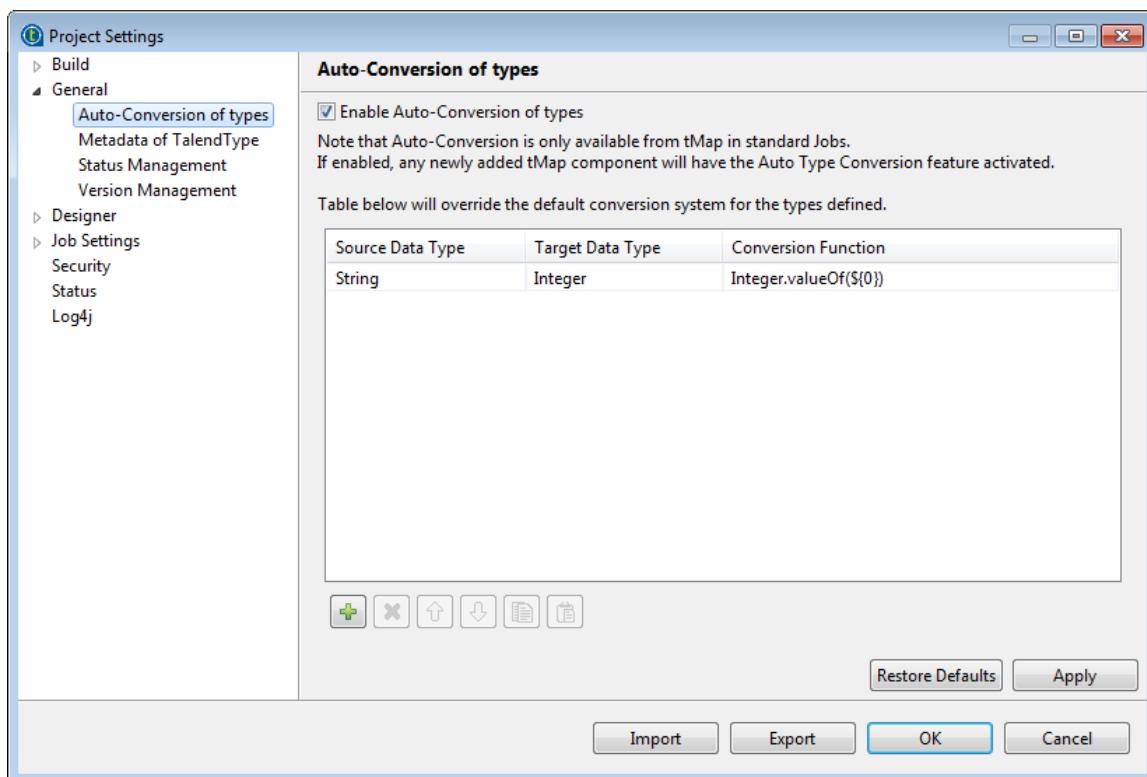
1. Click the  button at the top of the Map Editor to open the **[Property Settings]** dialog box.
2. Select the **Enable Auto-Conversion of types** check box and then click **OK**.



You can activate the automatic conversion option at the project level so that any **tMap** component added afterwards in the project will have this feature enabled.

If needed, you can also define conversion rules to override the default conversion behavior of **tMap**.

1. On the toolbar of the Studio main window, click or click **File > Edit Project Properties** from the menu bar to open the **[Project Settings]** dialog box.
2. In the tree view of the dialog box, expand **General** and select **Auto-Conversion of types** to open the relevant view.



3. Select the **Enable Auto-Conversion of types** check box to activate the automatic type conversion feature for all **tMap** components added afterwards in the project.
4. If needed, click the [+] button to add a line, select the source and target data types, and define a Java function for data type conversion to create a conversion rule to override the default conversion behavior of **tMap** for data that matches the rule.

You can press **Ctrl+Space** in the **Conversion Function** field to access a list of available Java functions.

The rule shown in this example will match mappings with the input data type of String and output data type of Integer.

You can create as many conversion rules as you want.

5. Click **Apply** to apply your changes and then **OK** to close the dialog box.

## 6.2.7. Solving memory limitation issues in tMap use

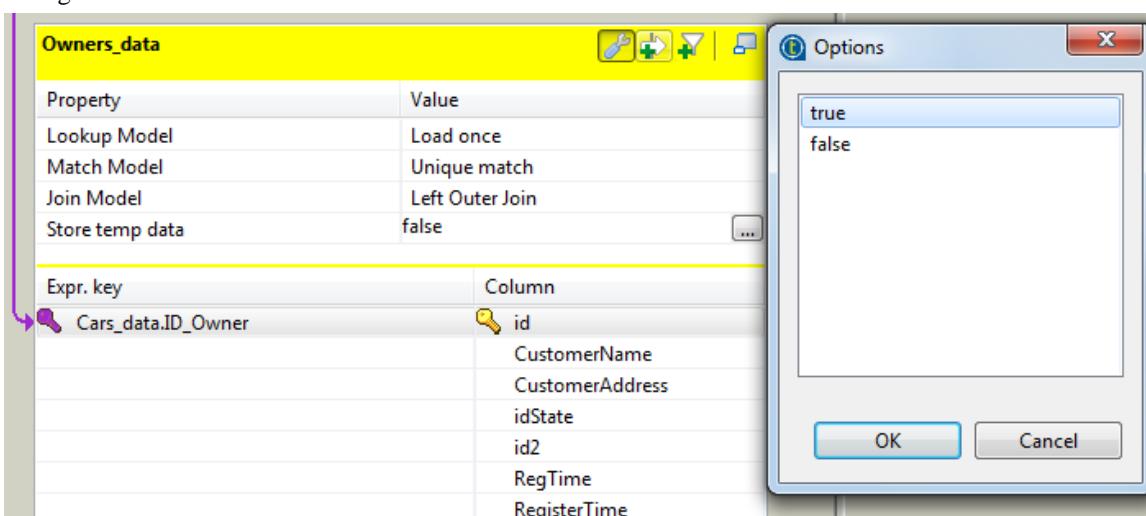
When handling large data sources, including for example, numerous columns, large number of lines or of column types, your system might encounter memory shortage issues that prevent your Job, to complete properly, in particular when using a **tMap** component for your transformation.

A feature has been added (in Java only for the time being) to the **tMap** component, in order to reduce the memory in use for lookup loading. In fact, rather than storing the temporary data in the system memory and thus possibly reaching the memory limitation, the **Store temp data** option allows you to choose to store the temporary data onto a directory of your disk instead.

This feature comes as an option to be selected in the Lookup table of the input data in the **Map Editor**.

To enable the **Store temp data** option:

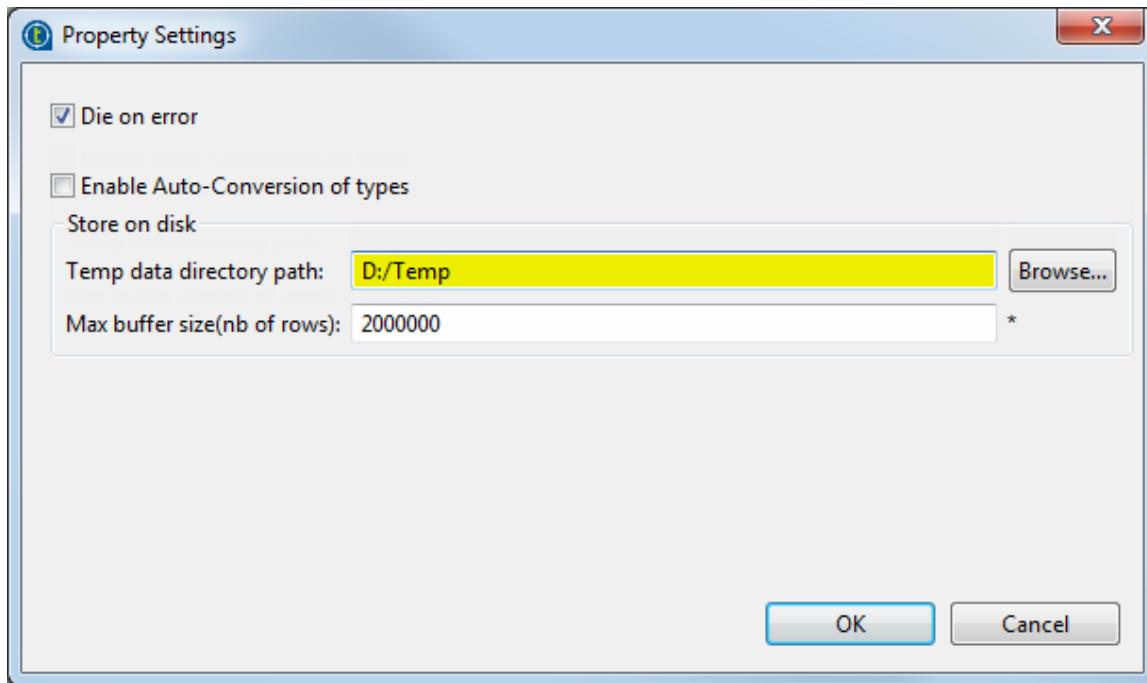
1. Double-click the **tMap** component in your Job to launch the **Map Editor**.
2. In input area, click the Lookup table describing the temporary data you want to be loaded onto the disk rather than in the memory.
3. Click the **tMap settings** button to display the table properties.
4. Click in the **Value** field corresponding to **Store temp data**, and then click the [...] button to display the **[Options]** dialog box.
5. In the **[Options]** dialog box, double-click **true**, or select it and click **OK**, to enable the option and close the dialog box.



For this option to be fully activated, you also need to specify the directory on the disk, where the data will be stored, and the buffer size, namely the number of rows of data each temporary file will contain. You can set the temporary storage directory and the buffer size either in the **Map Editor** or in the **tMap** component property settings.

To set the temporary storage directory and the buffer size in the **Map Editor**:

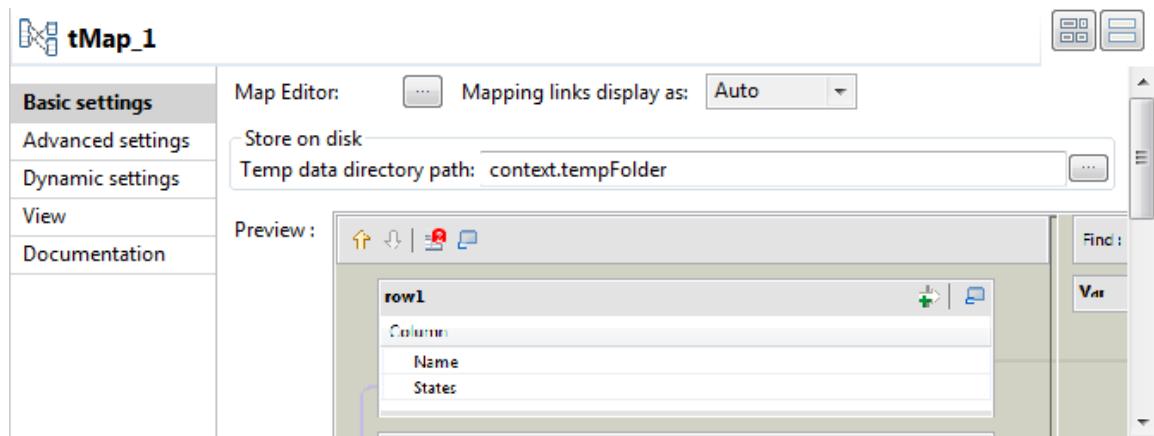
1. Click the **Property Settings** button at the top of the input area to display the **[Property Settings]** dialog box.
2. In **[Property Settings]** dialog box, fill the **Temp data directory path** field with the full path to the directory where the temporary data should be stored.
3. In the **Max buffer size (nr of rows)** field, specify the maximum number of rows each temporary file can contain. The default value is **2,000,000**.
4. Click **OK** to validate the settings and close the **[Property Settings]** dialog box.



To set the temporary storage directory in the **tMap** component property settings without opening the **Map Editor**:

1. Click the **tMap** component to select it on the design workspace, and then select the **Component** tab to show the **Basic settings** view.
2. In the **Store on disk** area, fill the **Temp data directory path** field with the full path to the directory where the temporary data should be stored.

Alternatively, you can use a context variable through the **Ctrl+Space** bar if you have set the variable in a Context group in the repository. For more information about contexts, see [Using contexts and variables](#).



At the end of the subjob, the temporary files are cleared.

This way, you will limit the use of allocated memory per reference data to be written onto temporary files stored on the disk.

 As writing the main flow onto the disk requires the data to be sorted, note that the order of the output rows cannot be guaranteed.

On the **Advanced settings** view, you can also set a buffer size if needed. Simply fill out the field **Max buffer size (nb of rows)** in order for the data stored on the disk to be split into as many files as needed.

## 6.2.8. Handling Lookups

When implementing a join (including **Inner Join** and **Left Outer Join**) in a **tMap** between different data sources, there is always only one main flow and one or more lookup flows connected to the **tMap**. All the records of the lookup flow need to be loaded before processing each record of the main flow. Three types of lookup loading models are provided suiting various types of business requirement and the performance needs: **Load once**, **Reload at each row**, and **Reload at each row (cache)**.

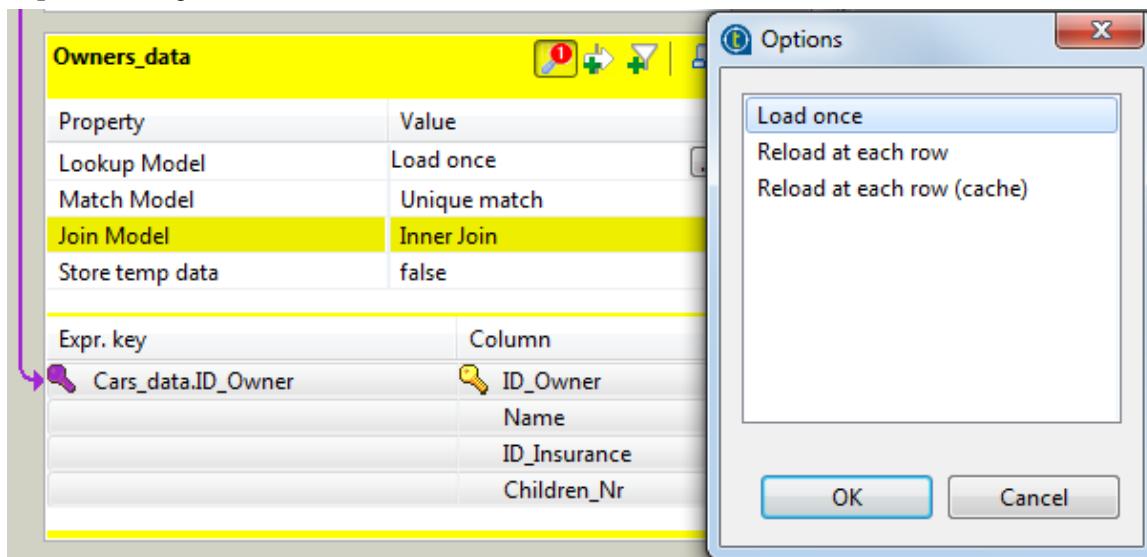
- **Load once:** it loads once (and only once) all the records from the lookup flow either in the memory or in a local file before processing each record of the main flow in case the **Store temp data** option is set to true. This is the default setting and the preferred option if you have a large set of records in the main flow to be processed using a join to the lookup flow.
- **Reload at each row:** it loads all the records of the lookup flow for each record of the main flow. Generally, this option increases the Job execution time due to the repeated loading of the lookup flow for each main flow record. However, this option is preferred in the following situations:
  - The lookup data flow is constantly updated and you want to load the latest lookup data for each record of the main flow to get the latest data after the join execution;
  - There are very few data from the main flow while a large amount of data from a database table in the lookup flow. In this case, it might cause an *OutOfMemory* exception if you use the **Load once** option. You can use dynamic variable settings such as where clause to update the lookup flow on the fly as it gets loaded, before the main flow join is processed. For an example, refer to [Reloading data at each row](#).
- **Reload at each row (cache):** it functions like the **Reload at each row** model, all the records of the lookup flow are loaded for each record of the main flow. However, this model can't be used with the **Store temp data on disk** option. The lookup data are cached in memory, and when a new loading occurs, only the records that are not already exist in the cache will be loaded, in order to avoid loading the same records twice. This option optimizes the processing time and helps improve processing performance of the **tMap** component. Note that you can not use **Reload at each row (cache)** and **Store temp data** at the same time.

Note that when your lookup is a database table, the best practise is to open the connection to the database in the beginning of your Job design in order to optimize performance.

### 6.2.8.1. Setting the loading mode of a lookup flow

To set the loading mode of a lookup flow:

1. Click the **tMap settings** button at the top right of the lookup table to display the table properties.
2. Click in the **Value** field corresponding to **Lookup Model**, and then click the [...] button to display the **[Options]** dialog box.



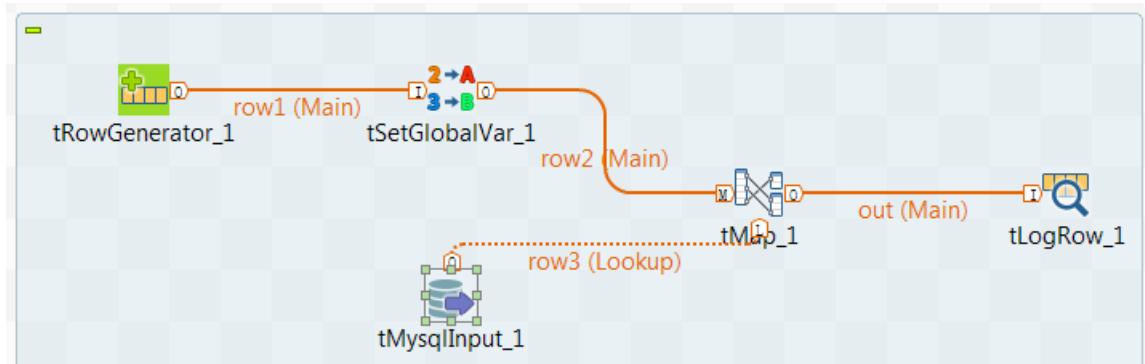
3. In the **[Options]** dialog box, double-click the wanted loading mode, or select it and then click **OK**, to validate the setting and close the dialog box.

For use cases using these options, see the related documentation of the **tMap** component.

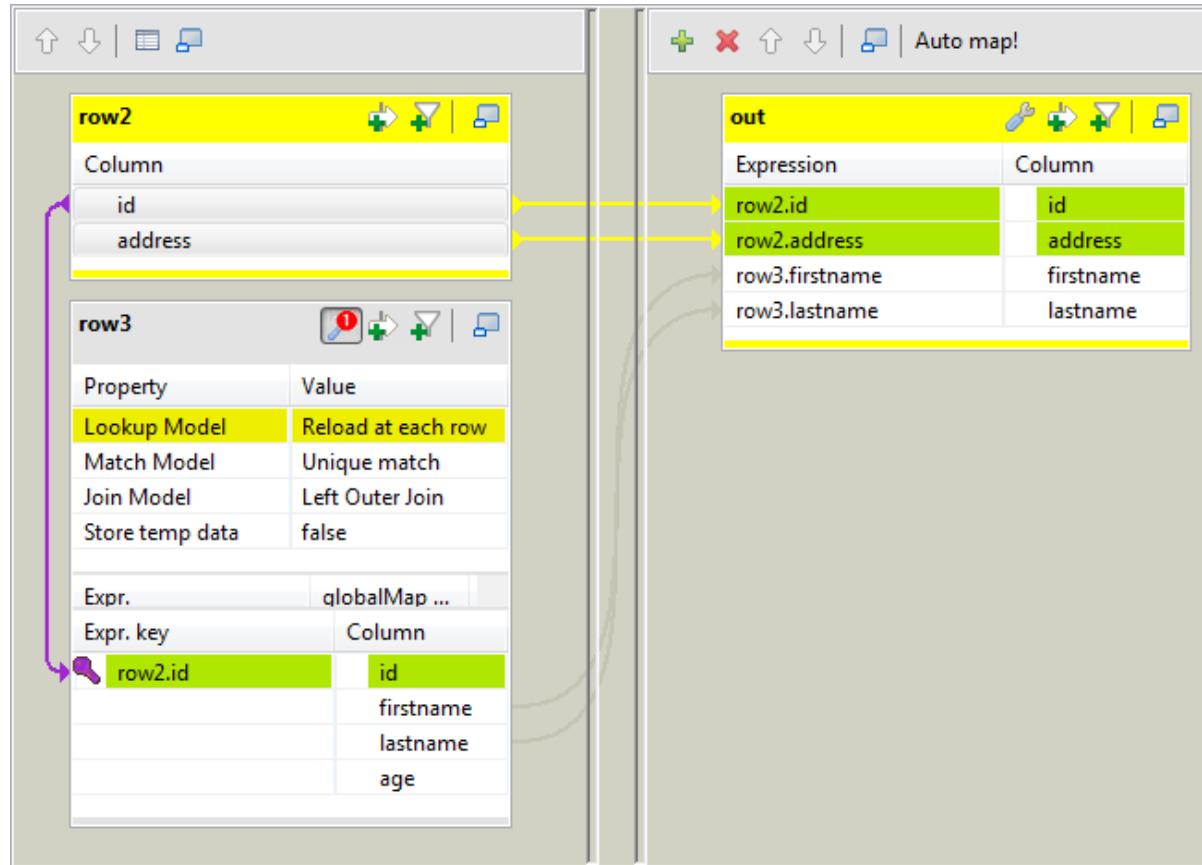
### 6.2.8.2. Reloading data at each row

The **Reload at each row** option is used to load all the records of a lookup flow for each record of the main flow.

When the main flow has much less rows than the lookup flow (for example, with a ratio of 1000 or more) and the lookup input is a database component, the advantage of this approach is that it helps deal with the fact that the amount of lookup data increases over time, since you can run queries against the data from the main flow in the database component to select only the lookup data that is relevant for each record in the main flow, such as in the following example which uses lookup data from a MySQL database.



The schemas of the main flow, the lookup flow and the output flow read as follows:



You can select from the MySQL database only the data that matches the values of the `id` column of the main flow. To do this, proceed as follows:

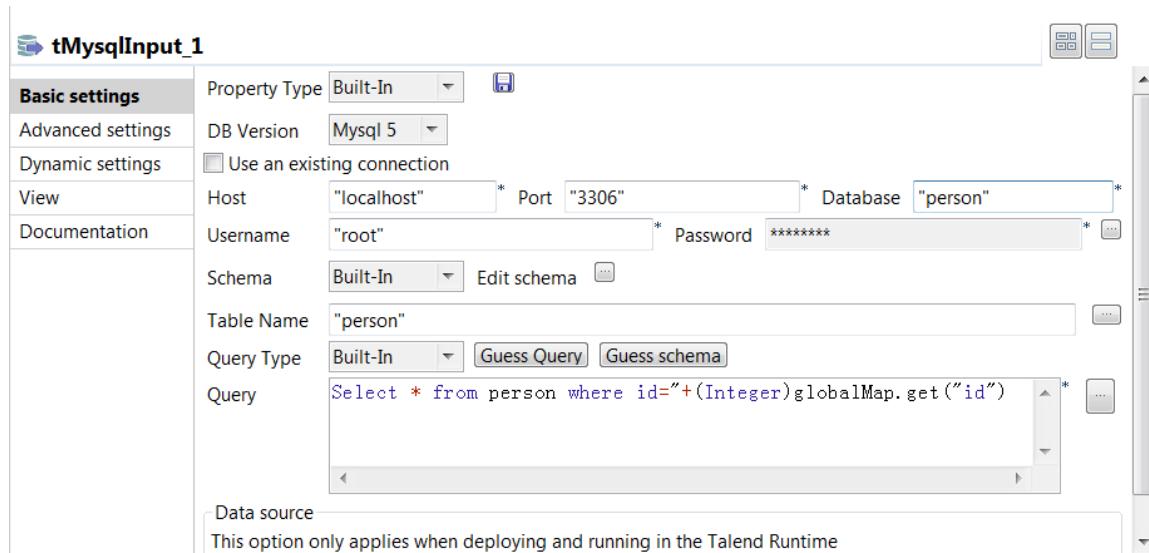
1. Double-click **tSetGlobalVar** to open its **Component** view.

The **tSetGlobalVar\_1** component view shows:

- Basic settings:** Advanced settings, Dynamic settings, View, Documentation.
- Variables:** A table with one row:
 

| Key  | Value     |
|------|-----------|
| "id" | "row1.id" |

2. Click the **[+]** button to add one row and name the **Key** to `id` and the **Value** to `row1.id`.
3. Double-click **tMysqlInput** to open its **Component** view.



4. In the **Query** field, enter the query to select the data that matches the *id* column of the main flow. In this example, this query reads:

```
Select * from person where id="+(Integer)globalMap.get("id")"
```

Refer to the related documentation of the components used in this example for more information.

## 6.3. tXMLMap operation



Before starting this section, we recommend reading the previous **tMap** sections for the basic knowledge of a **Talend** mapping component.

**tXMLMap** is fine-tuned to leverage the **Document** data type for processing XML data, a case of transformation that often mixes hierarchical data (XML) and flat data together. This **Document** type carries a complete user-specific XML flow. In using **tXMLMap**, you are able to add as many input or output flows as required into a visual map editor to perform, on these flows, the operations as follows:

- data multiplexing and demultiplexing,
- data transformation on any type of fields, particularly on the **Document** type,
- data matching via different models, for example, the **Unique match** mode (related topic: [How to use Explicit Join](#)),
- Automated XML tree construction on both of the input and the output sides,
- inner join and left outer join (related topic: [How to use Inner Join](#))
- lookup between data sources whatever they are flat or XML data using models like **Load once** (related topic: [Handling Lookups](#)),
- fields concatenation and interchange,
- field filtering using constraints,
- data rejecting.

Like **tMap**, a map editor is required to configure these operations. To open this map editor, you can double-click the **tXMLMap** icon in the design workspace, or alternatively, click the three-dot button next to the **Map Editor** in the **Basic settings** view of the **tXMLMap** component.

**tXMLMap** and **tMap** use the common approaches to accomplish most of these operations. Therefore, the following sections explain only the particular operations to which **tXMLMap** is dedicated for processing the hierarchical XML data.

The operations focusing on hierarchical data are:

- using the **Document** type to create the XML tree;
- managing the output XML data;
- editing the XML tree schema.

The following sections present more relevant details.



Different from **tMap**, **tXMLMap** does not provide the **Store temp data** option for storing temporary data onto the directory of your disk. For further information about this option of **tMap**, see [Solving memory limitation issues in tMap use](#).

## 6.3.1. Using the document type to create the XML tree

The **Document** data type fits perfectly the conception of defining XML structure as easily as possible. When you need the XML tree structure to map the input or output flow or both, use this type. Then you can import the XML tree structure from various XML sources and edit the tree directly in the mapping editor, thus saving the manual efforts.

### 6.3.1.1. How to set up the Document type

The **Document** data type is one of the data types provided by **Talend**. This **Document** type is set up when you edit the schema for the corresponding data in the **Schema editor**. For further information about the schema editor, see [Using the Schema Editor](#).

The following figure presents an example in which the input flow, *Customer*, is set up as the **Document** type. To replicate it, in the Map editor, you can simply click the **[+]** button to add one row on the input side of the **Schema editor**, rename it and select **Document** from the drop-down list of the given data types.

The screenshot shows the Talend Schema editor interface. At the top, there are three tabs: "Schema editor" (selected), "Tree schema editor", and "Expression editor". Below the tabs, the schema is defined under a section named "row1". A table lists columns with their properties. In the "Customer" column, the "Type" dropdown menu is open, displaying a list of data types. The "Document" option is highlighted with a blue selection bar and a cursor arrow pointing at it. Other options in the list include boolean | Boolean, byte | Byte, byte[], char | Character, Date, double | Double, float | Float, BigDecimal, int | Integer, long | Long, Object, short | Short, String, List, and Dynamic.

In practice for most cases, **tXMLMap** retrieves the schema of its preceding or succeeding components, for example, from a **tFileInputXML** component or in the ESB use case, from a **tESBProviderRequest** component. This avoids many manual efforts to set up the **Document** type for the XML flow to be processed. However, to continue to modify the XML structure as the content of a Document row, you need still to use the given Map editor.

-  Be aware that a **Document** flow carries a user-defined XML tree and is no more than one single field of a schema, which, same as the other schemas, may contain different data types between each field. For further information about how to set a schema, see [Basic Settings tab](#).

Once the **Document** type is set up for a row of data, in the corresponding data flow table in the map editor, a basic XML tree structure is created automatically to reflect the details of this structure. This basic structure represents the minimum element required by a valid XML tree in using **tXMLMap**:

- The root element: it is the minimum element required by an XML tree to be processed and when needs be, the foundation to develop a sophisticated XML tree.
- The loop element: it determines the element over which the iteration takes place to read the hierarchical data of an XML tree. By default, the root element is set as loop element.



This figure gives an example with the input flow, *Customer*. Based on this generated XML root tagged as **root** by default, you can develop the XML tree structure of interest.

To do this, you need to:

- Import the custom XML tree structure from one of the following types of sources:
  - XML or XSD files (related topic: [How to import the XML tree structure from XML and XSD files](#))
    -  When you import an XSD file, you will create the XML structure this XSD file describes.
  - file XML connections created and stored in the **Repository** of your Studio (related topic: [How to import the XML tree structure from the Repository](#)).
    -  If needs be, you can develop the XML tree of interest manually using the options provided on the contextual menu.
- Reset the loop element for the XML tree you are creating, if needs be. You can set as many loops as you need to. At this step, you may have to consider the following situations:
  - If you have to create several XML trees, you need to define the loop element for each of them.
  - If you import the XML tree from the **Repository**, the loop element will have been set depending on the set of the source structure. But you can still reset the loop element.

For further details, see [How to set or reset a loop element for an imported XML structure](#)

If needed, you can continue to modify the imported XML tree using the options provided in the contextual menu. The following table presents the operations you can perform through the available options.

| Options                                        | Operations                                                                                                                                          |
|------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Create Sub-element and Create Attribute</b> | Add elements or attributes to develop an XML tree. Related topic: <a href="#">How to add a sub-element or an attribute to an XML tree structure</a> |

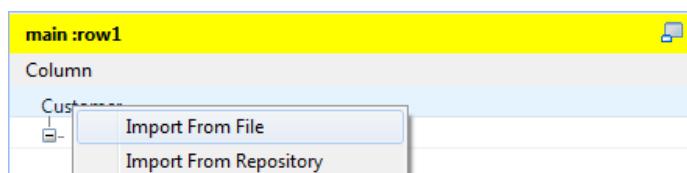
| Options                     | Operations                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|-----------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Set a namespace</b>      | Add and manage given namespaces on the imported XML tree. Related topic: <a href="#">How to manage a namespace</a>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| <b>Delete</b>               | Delete an element or an attribute. Related topic: <a href="#">How to delete an element or an attribute from the XML tree structure</a>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| <b>Rename</b>               | Rename an element or an attribute.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| <b>As loop element</b>      | Set or reset an element as loop element. Multiple loop elements and optional loop element are supported.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>As optional loop</b>     | This option is not available unless to the loop element you have defined.<br><br>When the corresponding element exists in the source file, an optional loop element works the same way as a normal loop element; otherwise, it resets automatically its parent element as loop element or in absence of parent element in the source file, it takes the element of the higher level until the root element. But in the real-world practice, with such differences between the XML tree and the source file structure, we recommend adapting the XML tree to the source file for better performance.                                                                                |
| <b>As group element</b>     | On the XML tree of the output side, set an element as group element. Related topic: <a href="#">How to group the output data</a>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>As aggregate element</b> | On the XML tree of the output side, set an element as aggregate element. Related topic: <a href="#">How to aggregate the output data</a>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>Add Choice</b>           | Set the Choice element. Then all of its child elements developed underneath will be contained in this declaration. This Choice element originates from one of the XSD concepts. It enables <b>tXMLMap</b> to perform the function of the XSD Choice element to read or write a Document flow.<br><br>When <b>tXMLMap</b> processes a choice element, the elements contained in its declaration will not be outputted unless their mapping expressions are appropriately defined.<br><br> The <b>tXMLMap</b> component declares automatically any Choice element set in the XSD file it imports. |
| <b>Set as Substitution</b>  | Set the Substitution element to specify the element substitutable for a given head element defined in the corresponding XSD. The Substitution element enables <b>tXMLMap</b> to perform the function of the XSD Substitution element to read or write a Document flow<br><br>When <b>tXMLMap</b> processes a substitution element, the elements contained in its declaration will not be outputted unless their mapping expressions are appropriately defined.<br><br> The <b>tXMLMap</b> component declares automatically any Substitution element set in the XSD file it imports.             |

The following sections present more details about the process of creating the XML tree.

### 6.3.1.2. How to import the XML tree structure from XML and XSD files

To import the XML tree structure from an XML file, proceed as follows:

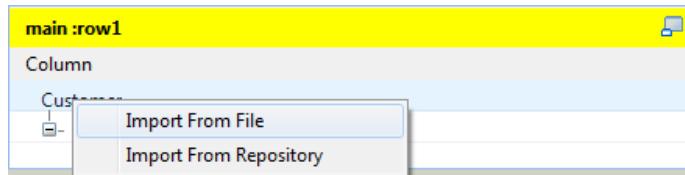
1. In the input flow table of interest, right-click the column name to open the contextual menu. In this example, it is *Customer*.



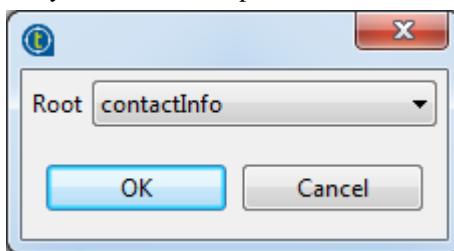
2. From this menu, select **Import From File**.
3. In the pop-up dialog box, browse to the XML file you need to use to provide the XML tree structure of interest and double-click the file.

To import the XML tree structure from an XSD file, proceed as follows:

1. In the input flow table of interest, right-click the column name to open the contextual menu. In this example, it is *Customer*.



2. From this menu, select **Import From File**.
3. In the pop-up dialog box, browse to the XSD file you need to use to provide the XML tree structure of interest and double-click the file.
4. In the dialog box that appears, select an element from the **Root** list as the root of your XML tree, and click **OK**. Then the XML tree described by the XSD file imported is established.



The root of the imported XML tree is adaptable:

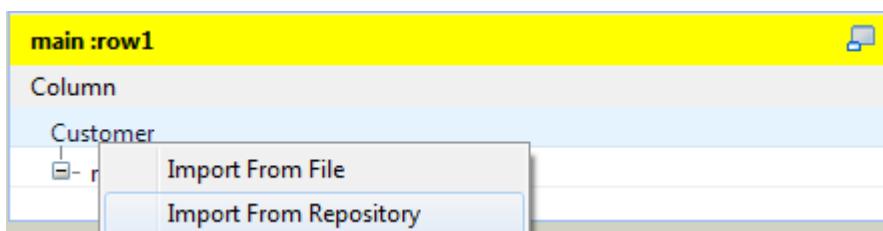
- When importing either an input or an output XML tree structure from an XSD file, you can choose an element as the root of your XML tree.
- Once an XML structure is imported, the **root** tag is renamed automatically with the name of the XML source. To change this root name manually, you need to use the tree schema editor. For further information about this editor, see [Editing the XML tree schema](#).

Then, you need to define the loop element in this XML tree structure. For further information about how to define a loop element, see [How to set or reset a loop element for an imported XML structure](#).

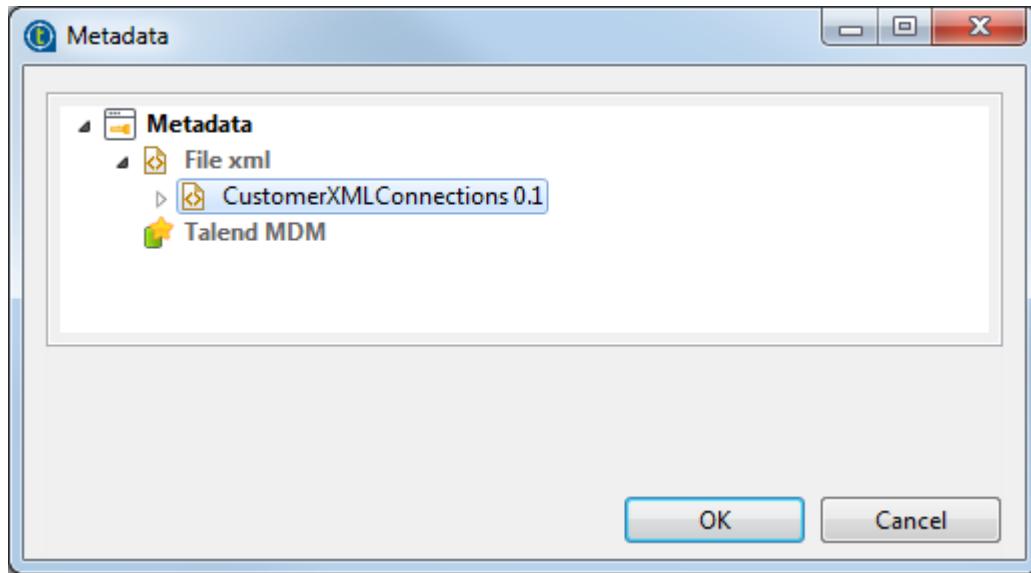
### 6.3.1.3. How to import the XML tree structure from the Repository

To do this, proceed as follows:

1. In any input flow table, right click the column name to open the contextual menu. In this example, it is *Customer*.



2. From this menu, select **Import From Repository**.
3. In the pop-up repository content list, select the XML connection or the MDM connection of interest to import the corresponding XML tree structure.



This figure presents an example of this **Repository**-stored XML connection.



To import an XML tree structure from the **Repository**, the corresponding XML connection should have been created. For further information about how to create a file XML connection in the Repository, see [Centralizing XML file metadata](#).

4. Click **OK** to validate this selection.

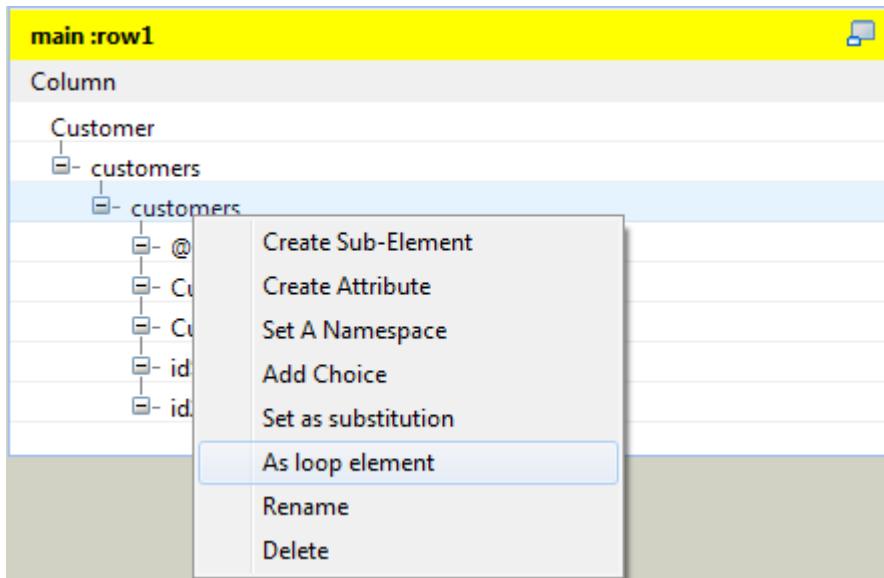
The XML tree structure is created and a loop is defined automatically as this loop was already defined during the creation of the current **Repository**-stored XML connection.

#### **6.3.1.4. How to set or reset a loop element for an imported XML structure**

You need to set at least one loop element for each XML tree if it does not have any. If it does, you may have to reset the existing loop element when needs be.

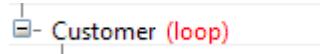
Whatever you need to set or reset a loop element, proceed as follows:

1. In the created XML tree structure, right-click the element you need to define as loop. For example, you need to define the *Customer* element as loop in the following figure.



2. From the pop-up contextual menu, select **As loop element** to define the selected element as loop.

Once done, this selected element is marked with the text: **loop**.



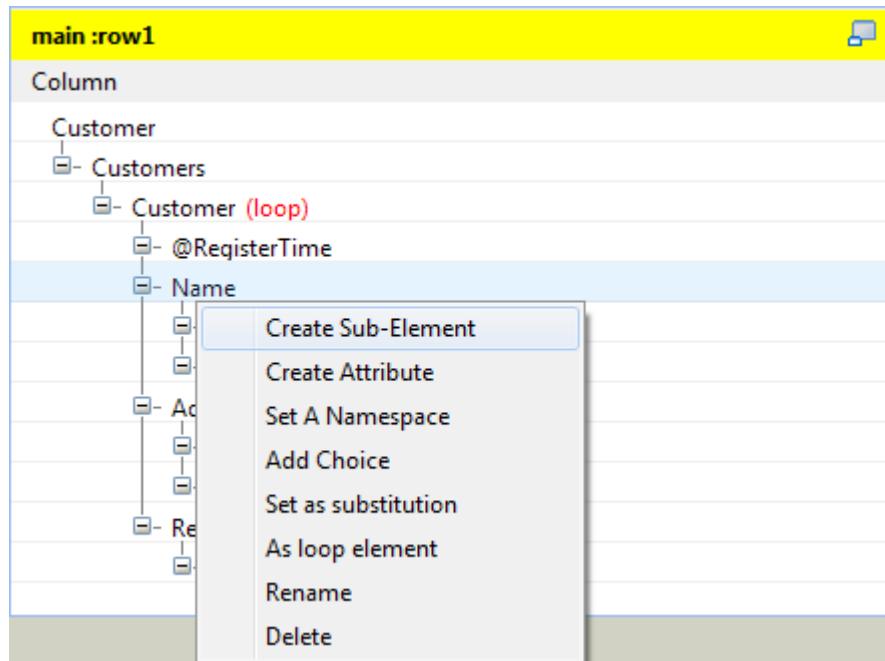
 If you close the **Map Editor** without having set the required loop element for a given XML tree, its root element will be set automatically as loop element.

### 6.3.1.5. How to add a sub-element or an attribute to an XML tree structure

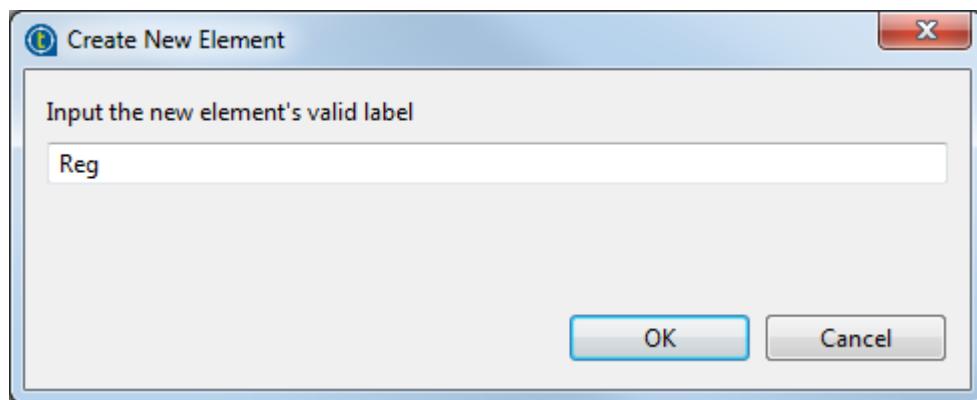
In the XML tree structure view, you are able to manually add a sub-element or an attribute to the root or to any of the existing elements when needs be.

To do either of these operations, proceed as follows:

1. In the XML tree you need to edit, right-click the element to which you need to add a sub-element or an attribute underneath and select **Create Sub-Element** or **Create Attribute** according to your purpose.



2. In the pop-up [Create New Element] wizard, type in the name you need to use for the added sub-element or attribute.

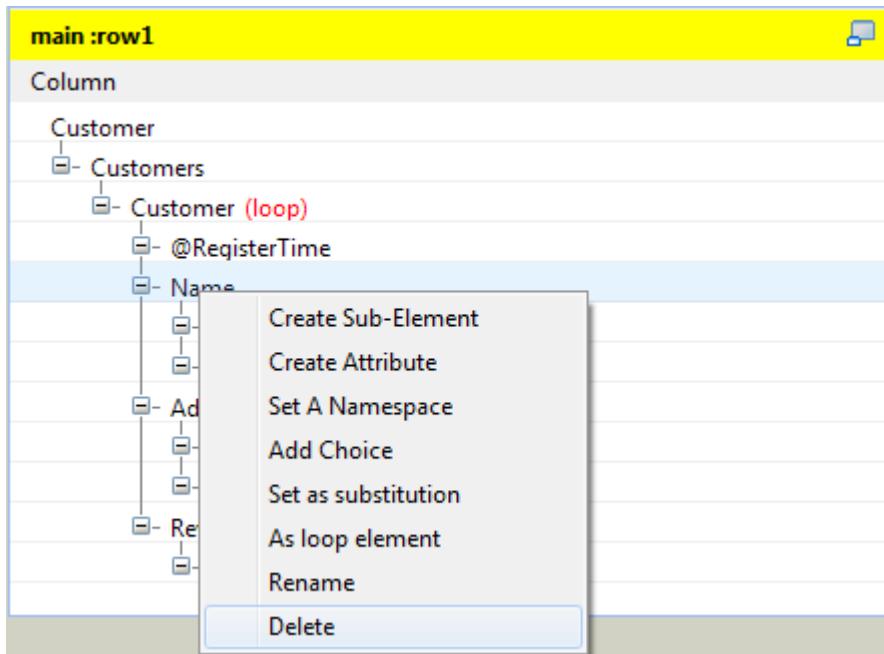


3. Click **OK** to validate this creation. The new sub-element or attribute displays in the XML tree structure you are editing.

### 6.3.1.6. How to delete an element or an attribute from the XML tree structure

From an established XML tree, you may need to delete an element or an attribute. To do this, proceed as follows:

1. In the XML tree you need to edit, right-click the element or the attribute you need to delete.



2. In the pop-up contextual menu, select **Delete**.

Then the selected element or attribute is deleted, including all of the sub-elements or the attributes attached to it underneath.

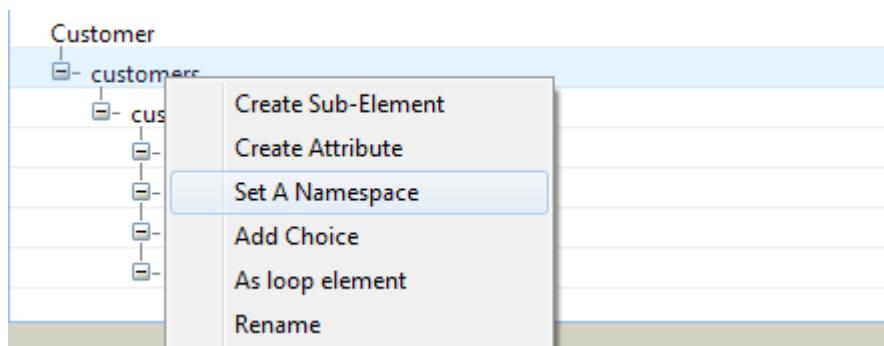
### 6.3.1.7. How to manage a namespace

When necessary, you are able to set and edit namespace for each of the element in the a created XML tree of the input or the output data flow.

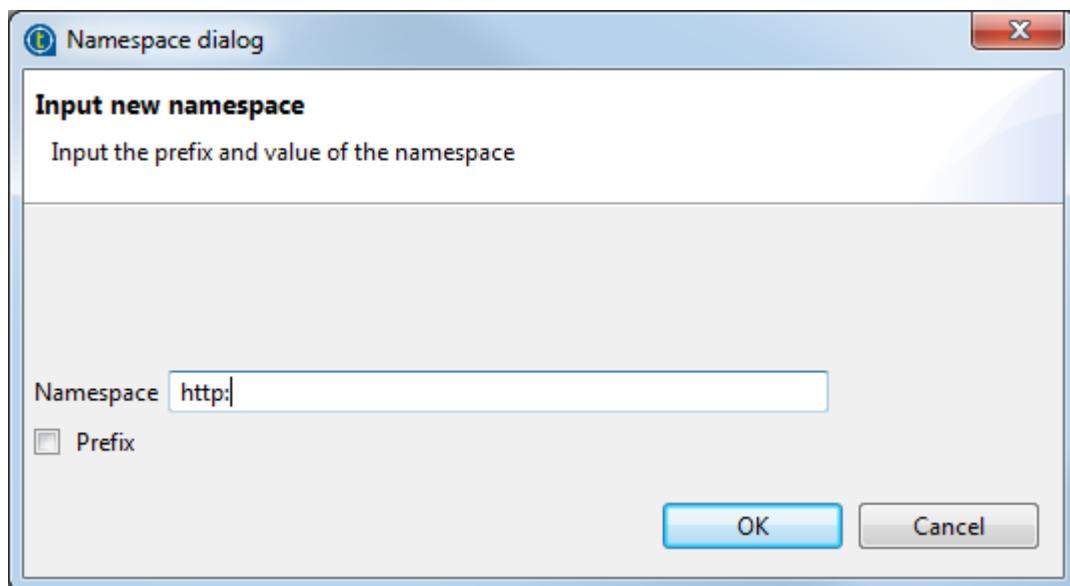
#### Defining a namespace

To do this, proceed as follows:

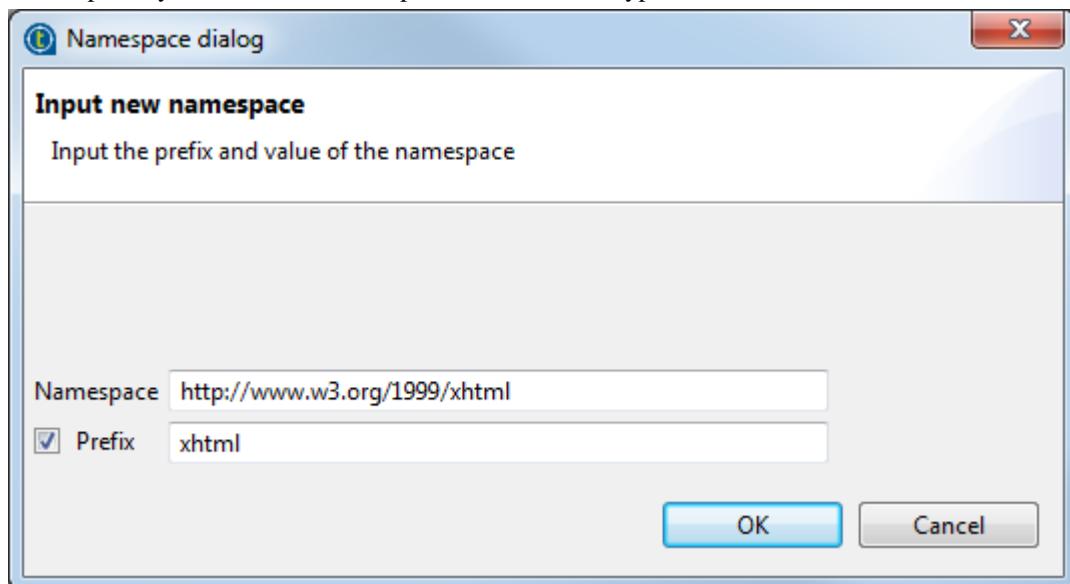
1. In the XML tree of the input or the output data flow you need to edit, right click the element for which you need to declare a namespace. For example, in a *Customer* XML tree of the output flow, you need to set a namespace for the root.



2. In the pop-up contextual menu, select **Set a namespace**. Then the [Namespace dialog] wizard displays.
3. In this wizard, type in the URI you need to use.



- If you need to set a prefix for this namespace you are editing, select the **Prefix** check box in this wizard and type in the prefix you need. In this example, we select it and type in *xhtml*.

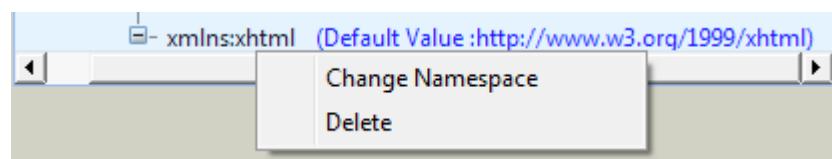


- Click **OK** to validate this declaration.

## Modifying the default value of a namespace

To do this, proceed as follows:

- In the XML tree that the namespace you need to edit belongs to, right-click this namespace to open the contextual menu.



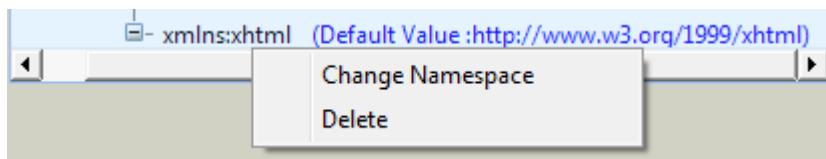
- In this menu, select **Change Namespace** to open the corresponding wizard.

3. Type in the new default value you need in this wizard.
4. Click **OK** to validate this modification.

## Deleting a namespace

To do this, proceed as follows:

1. In the XML tree that the namespace you need to edit belongs to, right-click this namespace to open the contextual menu.



2. In this menu, click **Delete** to validate this deletion

### 6.3.1.8. How to group the output data

The **tXMLMap** component uses a group element to group the output data according to a given grouping condition. This allows you to wrap elements matching the same condition with this group element.

To set a group element, two restrictions must be respected:

1. the root node cannot be set as group element;
2. the group element must be the parent of the loop element.

 The option of setting group element is not visible until you have set the loop element; this option is also invisible if an element is not allowed to be set as group element.

Once the group element is set, all of its sub-elements except the loop one are used as conditions to group the output data.

You have to carefully design the XML tree view for the optimized usage of a given group element. For further information about how to use a group element, see **tXMLMap** at <https://help.talend.com>.

 **tXMLMap** provides group element and aggregate element to classify data in the XML tree structure. When handling a row of XML data flow, the behavioral difference between them is:

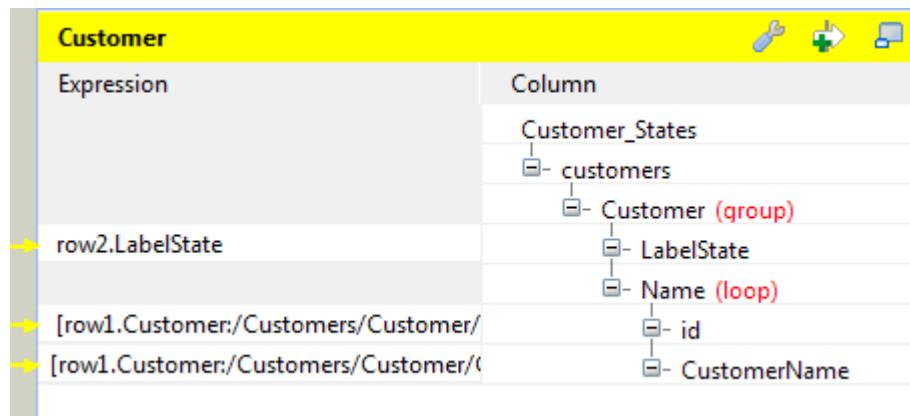
- The group element processes the data always within one single flow.
- The aggregate element splits this flow into separate and complete XML flows.

## Setting a group element

To set a group element, proceed as follows:

1. In the XML tree view on the output side of the **Map editor**, right-click the element you need to set as group element.
2. From the opened contextual menu, select **As group element**.

Then this element of selection becomes the group element. The following figure presents an example of an XML tree with the group element.



## Revoking a defined group element

To revoke a defined group element, proceed as follows:

1. In the XML tree view on the output side of the **Map editor**, right-click the element you have defined as **group element**.
2. From the opened contextual menu, select **Remove group element**.

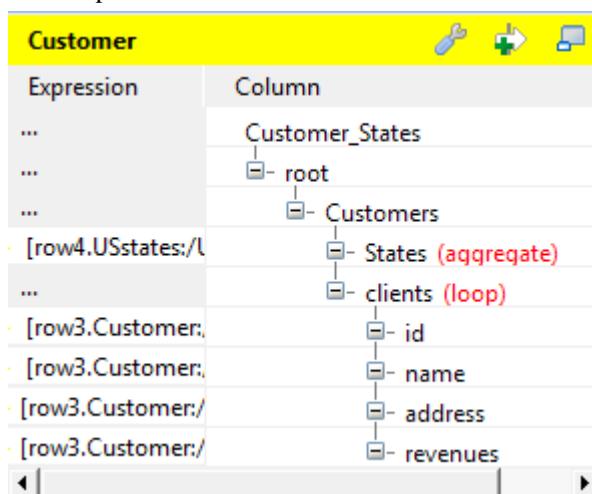
Then the defined group element is revoked.

### 6.3.1.9. How to aggregate the output data

With **tXMLMap**, you can define as many aggregate elements as required in the output XML tree to class the XML data accordingly. Then this component outputs these classes, each as one complete XML flow.

1. To define an element as aggregate element, simply right-click this element of interest in the XML tree view on the output side of the **Map editor** and from the contextual menu, select **As aggregate element**.

Then this element becomes the aggregate element. Texts in red are added to it, reading **aggregate**. The following figure presents an example.



2. To revoke the definition of the aggregate element, simply right-click the defined aggregate element and from the contextual menu, select **Remove aggregate element**.

 To define an element as aggregate element, ensure that this element has no child element and the **All in one** feature is being disabled. The **As aggregate element** option is not available in the contextual menu until both of the conditions are respected. For further information about the **All in one** feature, see [How to output elements into one document](#).

For an example about how to use the aggregate element with **tXMLMap**, see the **tXMLMap** documentation at <https://help.talend.com>.

 **tXMLMap** provides **group element** and **aggregate element** to classify data in the XML tree structure. When handling one row of data (one complete XML flow), the behavioral difference between them is:

- The **group element** processes the data always within one single flow.
- The **aggregate element** splits this flow into separate and complete XML flows.

## 6.3.2. Defining the output mode

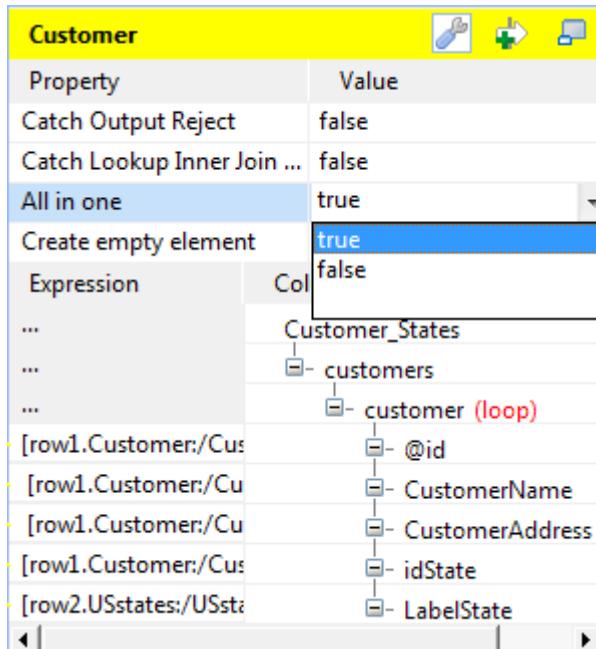
To define the output mode of the document-type data, you are defining whether to put all of the XML elements into one single XML flow and when empty element exist, whether to output them. By doing this, you do not change the structure of the XML tree you have created.

### 6.3.2.1. How to output elements into one document

Unless you are using the aggregate element which always classifies the output elements and splits an output XML flow, you are able to determine whether an XML flow is output as one single flow or as separate flows, using the **All in one** feature in the **tXMLMap** editor.

To do this, on the output side of the **Map editor**, proceed as follows:

1. Click the pincer icon to open the map setting panel. The following figure presents an example.



2. Click the **All in one** field and from the drop-down list, select **true** or **false** to decide whether the output XML flow should be one single flow.
  - If you select **true**, the XML data is output all in one single flow. In this example, the single flow reads as follows:

```
Starting job tXMLMap at 10:16 09/11/2011.
[statistics] connecting to socket on port 3643
[statistics] connected
<?xml version="1.0" encoding="UTF-8"?>
<customers><customer id="1"><CustomerName>Griffith Paving
and
Sealcoatin</CustomerName><CustomerAddress>talend@apres91</C
ustomerAddress><idState>7</idState><LabelState>Connecticut<
<LabelState></customer><customer
id="56"><CustomerName>Glenn Oaks Office
Supplies</CustomerName><CustomerAddress>1859 Green Bay
Rd.</CustomerAddress><idState>7</idState><LabelState>Conne
cticut</LabelState></customer><customer
id="2"><CustomerName>Bill's Dive
Shop</CustomerName><CustomerAddress>511 Maple Ave. Apt.
1B</CustomerAddress><idState>35</idState><LabelState>Ohio</
<LabelState></customer><customer id="61"><CustomerName>DBN
Bank</CustomerName><CustomerAddress>456 Grossman
Ln.</CustomerAddress><idState>35</idState><LabelState>Ohio<
<LabelState></customer><customer
id="63"><CustomerName>Pivot Point
College</CustomerName><CustomerAddress>1547 Knolwood
Rd.</CustomerAddress><idState>9</idState><LabelState>Flori
da</LabelState></customer></customers>
[statistics] disconnected
Job tXMLMap ended at 10:16 09/11/2011. [exit code=0]
```

The structure of this flow reads:

```
<?xml version="1.0" encoding="UTF-8"?>
<customers>
 <customer id="1">
 <CustomerName>Griffith Paving and Sealcoatin</CustomerName>
 <CustomerAddress>talend@apres91</CustomerAddress>
 <idState>7</idState>
 <LabelState>Connecticut</LabelState>
 </customer>
 <customer id="56">
 <CustomerName>Glenn Oaks Office Supplies</CustomerName>
 <CustomerAddress>1859 Green Bay Rd.</CustomerAddress>
 <idState>7</idState>
 <LabelState>Connecticut</LabelState>
 </customer>
 <customer id="2">
 <CustomerName>Bill's Dive Shop</CustomerName>
 <CustomerAddress>511 Maple Ave. Apt. 1B</CustomerAddress>
 <idState>35</idState>
 <LabelState>Ohio</LabelState>
 </customer>
 <customer id="61">
 <CustomerName>DBN Bank</CustomerName>
 <CustomerAddress>456 Grossman Ln.</CustomerAddress>
 <idState>35</idState>
 <LabelState>Ohio</LabelState>
 </customer>
 <customer id="63">
 <CustomerName>Pivot Point College</CustomerName>
 <CustomerAddress>1547 Knolwood Rd.</CustomerAddress>
 <idState>9</idState>
 <LabelState>Florida</LabelState>
 </customer>
</customers>
```

- If you select **false**, the XML data is output in separate flows, each loop being one flow, neither grouped nor aggregated. In this example, these flows read as follows:

```

Starting job tXMLMap at 10:25 09/11/2011.

[statistics] connecting to socket on port 4036
[statistics] connected
<?xml version="1.0" encoding="UTF-8"?>
<customers><customer id="1"><CustomerName>Griffith Paving
and
Sealcoatin</CustomerName><CustomerAddress>talend@apres91</C
ustomerAddress><idState>7</idState><LabelState>Connecticut<
/LabelState></customer></customers>
<?xml version="1.0" encoding="UTF-8"?>
<customers><customer id="56"><CustomerName>Glenn Oaks
Office Supplies</CustomerName><CustomerAddress>1859 Green
Bay
Rd.</CustomerAddress><idState>7</idState><LabelState>Conne
cticut</LabelState></customer></customers>
<?xml version="1.0" encoding="UTF-8"?>
<customers><customer id="2"><CustomerName>Bill's Dive
Shop</CustomerName><CustomerAddress>511 Maple Ave. Apt.
1B</CustomerAddress><idState>35</idState><LabelState>Ohio<
/LabelState></customer></customers>
<?xml version="1.0" encoding="UTF-8"?>
<customers><customer id="61"><CustomerName>DBN
Bank</CustomerName><CustomerAddress>456 Grossman
Ln.</CustomerAddress><idState>35</idState><LabelState>Ohio<
/LabelState></customer></customers>
<?xml version="1.0" encoding="UTF-8"?>
<customers><customer id="63"><CustomerName>Pivot Point
College</CustomerName><CustomerAddress>1547 Knolwood
Rd.</CustomerAddress><idState>9</idState><LabelState>Flori
da</LabelState></customer></customers>
```

Each flow contains one complete XML structure. To take the first flow as example, its structure reads:

```

<?xml version="1.0" encoding="UTF-8"?>
<customers>
 <customer id="1">
 <CustomerName>Griffith Paving and Sealcoatin</CustomerName>
 <CustomerAddress>talend@apres91</CustomerAddress>
 <idState>7</idState>
 <LabelState>Connecticut</LabelState>
 </customer>
</customers>
```



The **All in one** feature is disabled if you are using the aggregate element. For further information about the aggregate element, see [How to aggregate the output data](#)

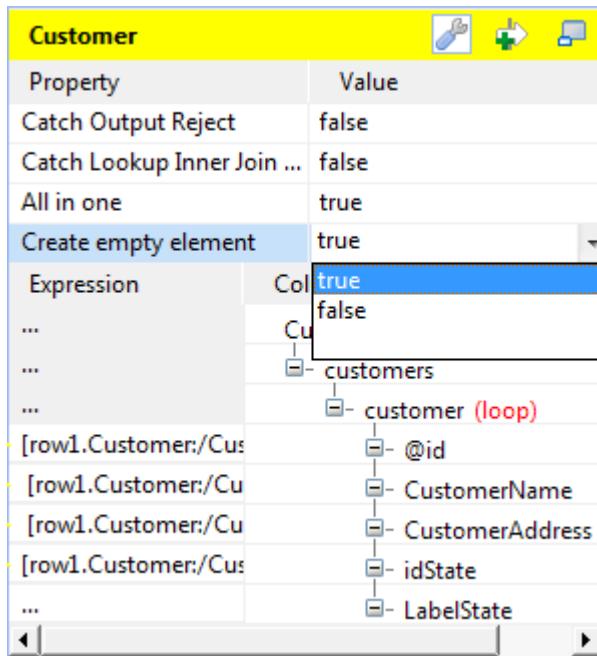
### 6.3.2.2. How to manage empty element in Map editor

It may be necessary to create and output empty elements during the process of transforming data into XML flow, such as, when **tXMLMap** works along with **tWriteXMLField** that creates empty elements or when there is no input column associated with certain XML node in the output XML data flow.

By contrast, in some scenarios, you do not need to output the empty element while you have to keep them in the output XML tree for some reasons.

**tXMLMap** allows you to set the boolean for the creation of empty element. To do this, on the output side of the **Map editor**, perform the following operations:

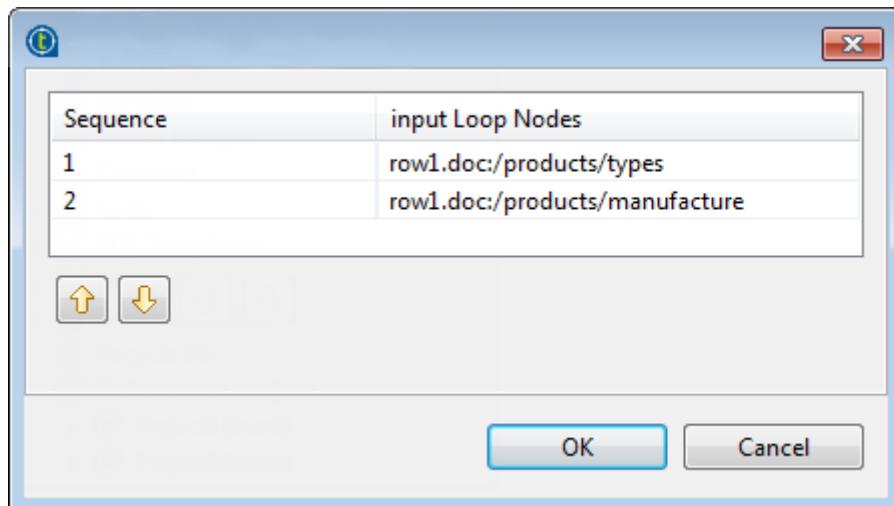
1. Click the pincer icon to open the map setting panel.



2. In the panel, click the **Create empty element** field and from the drop-down list, select **true** or **false** to decide whether to output the empty element.
  - If you select **true**, the empty element is created in the output XML flow and output, for example, `<customer><LabelState/></customer>`.
  - If you select **false**, the empty element is not output.

### 6.3.2.3. How to define the sequence of multiple input loops

If a loop element, or the flat data flow, receives mappings from more than one loop element of the input flow, you need to define the sequence of the input loops. The first loop element of this sequence will be the primary loop, so the transformation process related to this sequence will first loop over this element such that the data outputted will be sorted with regard to its element values.



For example, in this figure, the *types* element is the primary loop and the outputted data will be sorted by the values of this element.

```

<types>
 <type>DELL123</type>
 <manufacture_id>manu_1</manufacture_id>
</types>
<types>
 <type>DELL123</type>
 <manufacture_id>manu_2</manufacture_id>
</types>
<types>
 <type>DELL456</type>
 <manufacture_id>manu_1</manufacture_id>
</types>
<types>
 <type>DELL456</type>
 <manufacture_id>manu_2</manufacture_id>
</types>
<types>
 <type>HP123</type>
 <manufacture_id>manu_1</manufacture_id>
</types>
<types>
 <type>HP123</type>
 <manufacture_id>manu_2</manufacture_id>
</types>
<types>
 <type>HP456</type>
 <manufacture_id>manu_1</manufacture_id>
</types>
<types>
 <type>HP456</type>
 <manufacture_id>manu_2</manufacture_id>
</types>
</manufactures>

```

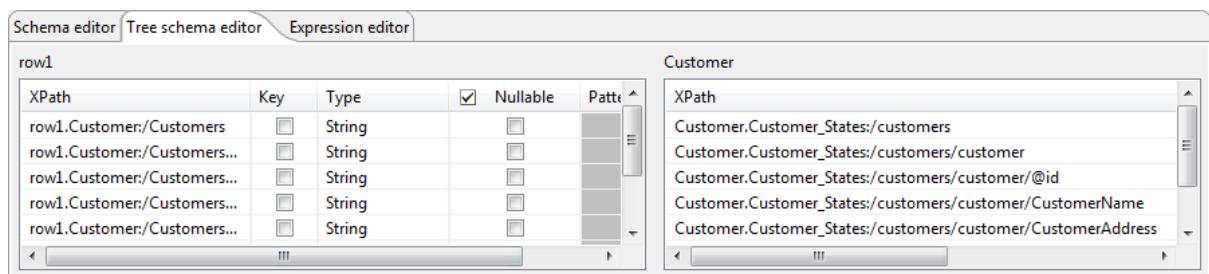
In this case in which one output loop element receives several input loop elements, a [...] button appears next to this receiving loop element or for the flat data, appears on the head of the table representing the flat data flow. To define the loop sequence, do the following:

1. Click this [...] button to open the sequence arrangement window as presented by the figure used earlier in this section.
2. Use the up or down flash button to arrange this sequence.

### 6.3.3. Editing the XML tree schema

In addition to the **Schema editor** and the **Expression editor** views that **tMap** is also equipped with, a **Tree schema editor** view is provided in the map editor of **tXMLMap** for you to edit the XML tree schema of an input or output data flow.

To access this schema editor, click the **Tree schema editor** tab on the lower part of the map editor.



The left half of this view is used to edit the tree schema of the input flow and the right half to edit the tree schema of the output flow.

The following table presents further information about this schema editor.

Metadata	Description
<b>XPath</b>	Use it to display the absolute paths pointing to each element or attribute in a XML tree and edit the name of the corresponding element or attribute.
<b>Key</b>	Select the corresponding check box if the expression key data should be used to retrieve data through the Join link. If unchecked, the Join relation is disabled.
<b>Type</b>	Type of data: String, Integer, Document, etc.  💡 This column should always be defined in a Java version.
<b>Nullable</b>	Select this check box if the field value could be null.
<b>Pattern</b>	Define the pattern for the Date data type.

Input metadata and output metadata are independent from each other. You can, for instance, change the label of a column on the output side without the column label of the input schema being changed.

However, any change made to the metadata are immediately reflected in the corresponding schema on the **tXMLMap** relevant (Input or Output) area, but also on the schema defined for the component itself on the design workspace.



# Chapter 7. Managing Metadata

Metadata in *Talend Studio* is definitional data that provides information about or documentation of other data managed within *Talend Studio*.

In the **Integration** perspective of the studio, the **Metadata** folder stores reusable information on files, databases, and/or systems that you need to create your Jobs.

Various corresponding wizards help you store these pieces of information and use them later to set the connection parameters of the relevant input or output components, but you can also store the data description called "schemas" in your studio.

Wizards' procedures slightly differ depending on the type of connection chosen.

This chapter provides procedures to create and manage various metadata items in the **Repository** that can be used in all your Job designs. For how to use a **Repository** metadata item, see [How to use centralized metadata in a Job](#) and [How to set a repository schema](#).

Before starting any metadata management processes, you need to be familiar with the Graphical User Interface (GUI) of your studio. For more information, see the appendix describing GUI elements.

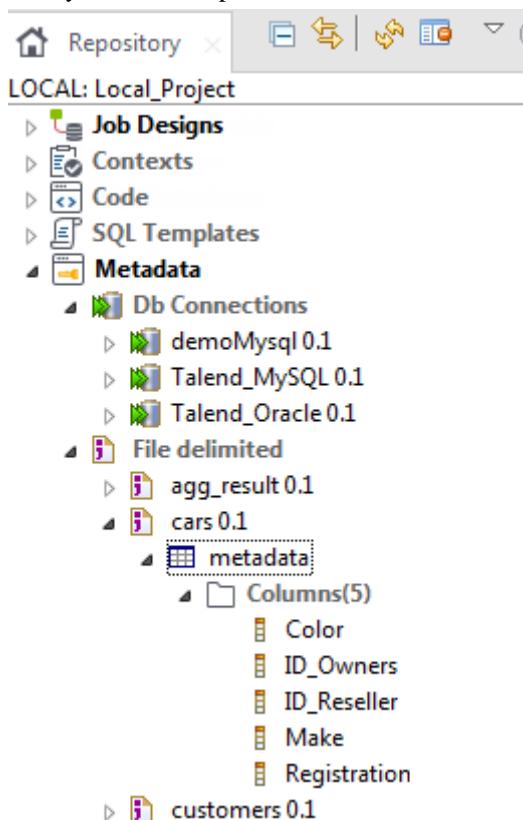
## 7.1. Objectives

The **Metadata** folder in the **Repository** tree view stores reusable information on files, databases, and/or systems that you need to create your Jobs.

Various corresponding wizards help you store these pieces of information that can be used later to set the connection parameters of the relevant input or output components and the data description called "schemas" in a centralized manner in *Talend Studio*.

The procedures of different wizards slightly differ depending on the type of connection chosen.

Click **Metadata** in the **Repository** tree view to expand the folder tree. Each of the connection nodes will gather the various connections and schemas you have set up.



From *Talend Studio*, you can set up the following, amongst others:

- a DB connection,
- a JDBC schema,
- a SAS connection,
- a file schema,
- an LDAP schema,
- a Salesforce schema,
- a generic schema,
- a MDM connection,

- a WSDL schema,
- a FTP connection,

## 7.2. Centralizing database metadata

If you often need to connect to database tables of any kind, then you may want to centralize the connection information details in the **Metadata** folder in the **Repository** tree view.

This setup procedure is made of two separate but closely related major tasks:

1. Set up a database connection,
2. Retrieve the table schemas.

The sections below describe how to complete the tasks in detail.

**Prerequisites:** *Talend Studio* requires specific third-party Java libraries or database drivers (*.jar* files) to be installed in order to connect to sources or targets. Due to license restrictions, **Talend** may not be able to ship certain required libraries or drivers; in that situation, the connection wizard to be presented in the following sections displays related information to help you identify and install the libraries or drivers in question. For more information, see the *Talend Installation and Upgrade Guide*.

### 7.2.1. Setting up a database connection

To create a database connection from scratch, expand **Metadata** in the **Repository** tree view, right-click **Db Connections** and select **Create connection** from the contextual menu to open the database connection setup wizard.

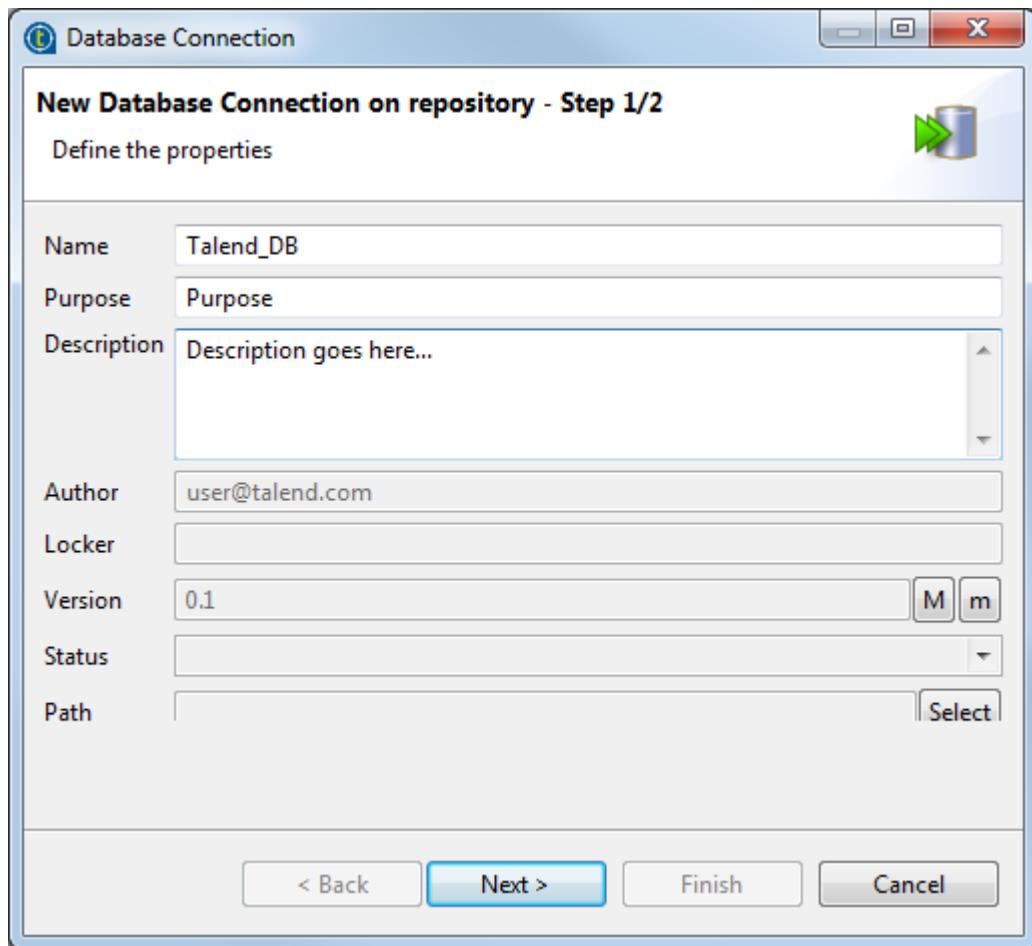
To centralize database connection parameters you have defined in a Job, click the  icon in the **Basic settings** view of the relevant database component with its **Property Type** set to **Built-in** to open the database connection setup wizard.

To modify an existing database connection, right-click the connection item from the **Repository** tree view, and select **Edit connection** to open the connection setup wizard.

Then define the general properties and parameters of the connection in the wizard.

#### 7.2.1.1. Defining general properties

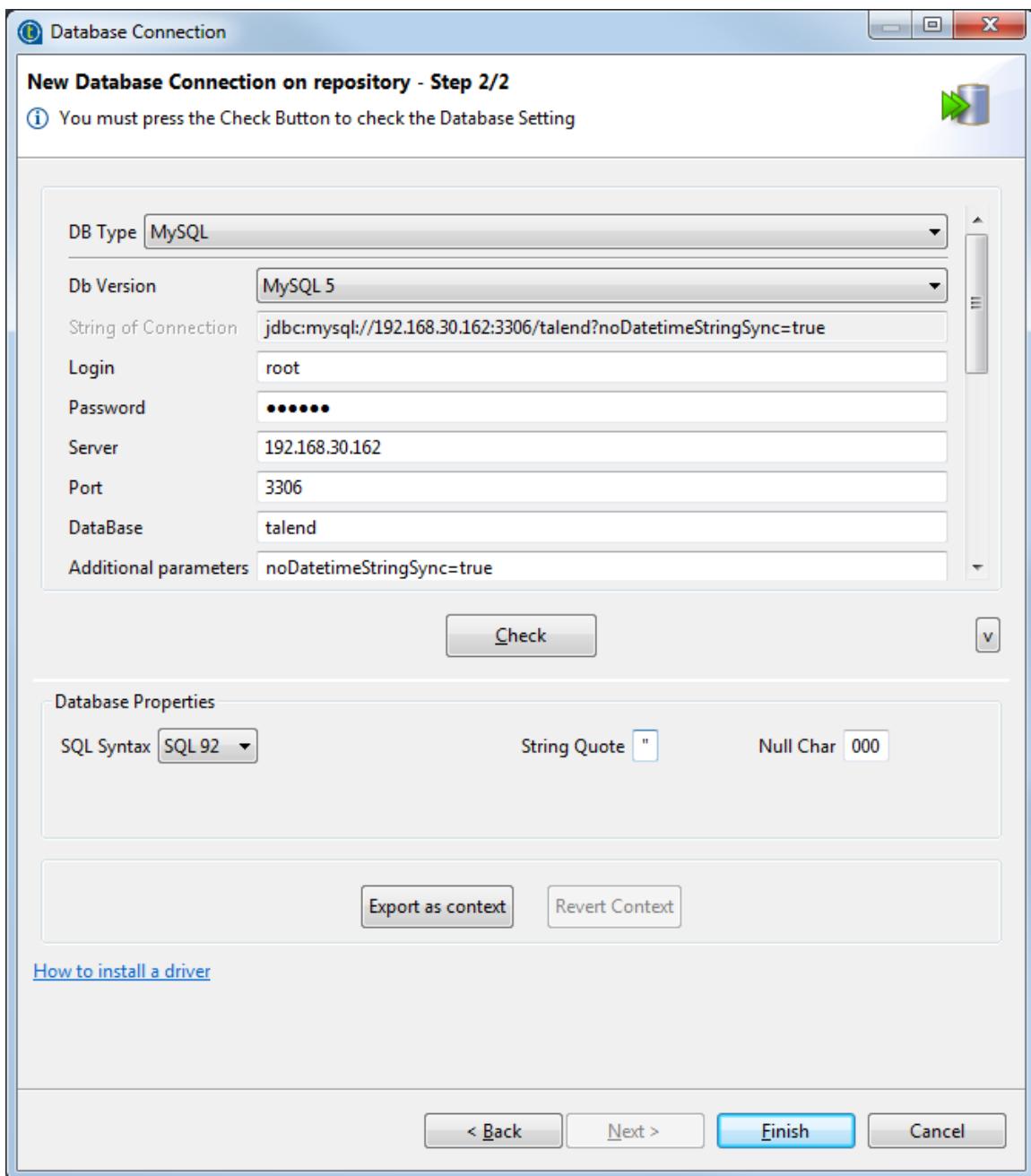
1. In the connection setup wizard, give your connection a name in the **Name** field. This name will appear as the database connection name under the **Metadata** node of the **Repository** tree view.



2. Fill in the optional **Purpose** and **Description** fields as required. The information you fill in the **Description** field will appear as a tooltip when you move your mouse pointer over the connection.
3. If needed, set the connection version and status in the **Version** and **Status** fields respectively. You can also manage the version and status of a repository item in the **[Project Settings]** dialog box. For more information, see [Version management](#) and [Status management](#) respectively.
4. If needed, click the **Select** button next to the **Path** field to select a folder under the **Db connections** node to hold your newly created database connection. Note that you cannot select a folder if you are editing an existing database connection, but you can drag and drop a connection to a new folder whenever you want.
5. Click **Next** when completed. The second step requires you to fill in or edit database connection data.

### 7.2.1.2. Defining connection parameters

1. Select the type of the database to which you want to connect and fill in the connection details. The fields you need to fill vary depending on the database type you select.



When you are creating the database connection of some databases like AS/400, HSQDB, Informix, Microsoft SQL, MySQL, Oracle, Sybase, or Teradata, you can specify additional connection properties through the **Additional parameters** field in the **Database Settings** area.

In *Talend Studio* 6.0 and onwards, due to limitations of Java 8, ODBC is no longer supported for Access database connections, and the only supported database driver type is JDBC.

Also due to Java 8 limitations, you cannot create Generic ODBC or Microsoft SQL Server (ODBC) connections in *Talend Studio* 6.0 and onwards unless you import such connections created in an earlier version of *Talend Studio* - in that case, you can create Generic ODBC and Microsoft SQL Server (ODBC) connections but they work only with Java 7.

For an MS SQL Server (JDBC) connection, when **Microsoft** is selected from the **Db Version** list, you need to download the Microsoft JDBC driver for SQL Server on [Microsoft Download Center](#), unpack the downloaded zip file, choose a jar in the unzipped folder based on your JRE version, rename the jar to *mssql-jdbc.jar* and install it manually. For more information about choosing the jar, see the System Requirements information on [Microsoft Download Center](#).

If you need to connect to Hive, we recommend using one of the **Talend** solutions with Big Data.



If you are creating an MSSQL connection, in order to be able to retrieve all table schemas in the database, be sure to:

- enter dbo in the **Schema** field if you are connecting to MSSQL 2000,
- remove dbo from the **Schema** field if you are connecting to MSSQL 2005/2008.

2. Click **Check** to check your connection.

If the connection fails, a message box is displayed to indicate this failure and from that message box. From that message box, click the **Details** button to read further information.

If a missing library or driver (.jar file) has provoked this failure, you can read that from the **Details** panel and then install the specified library or driver.

The Studio provides multiple approaches to automate the installation. For further information, see the chapter describing how to install external modules of the *Talend Installation and Upgrade Guide*.

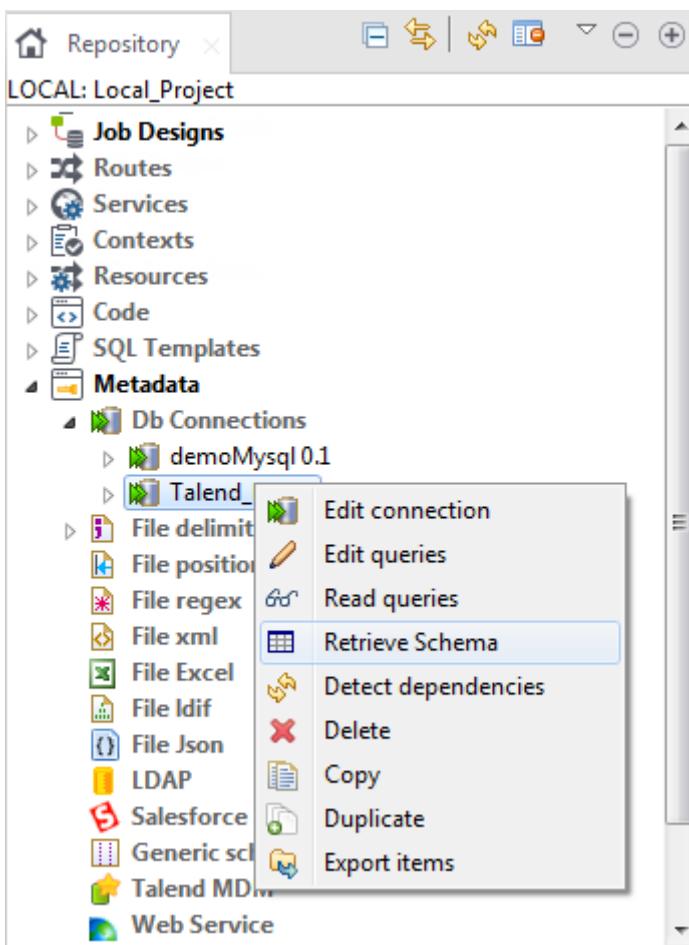
3. If needed, fill in the database properties information. That is all for the first operation on database connection setup. Click **Finish** to close the connection setup wizard.

The newly created database connection is now available in the **Repository** tree view and it displays several folders including **Queries** (for SQL queries you save) and **Table schemas** that will gather all schemas linked to this database connection upon table schema retrieval.

Now you can drag and drop the database connection onto the design workspace as a database component to reuse the defined database connection details in your Job.

## 7.2.2. Retrieving table schemas

To retrieve table schemas from the database connection you have just set up, right-click the connection item from the **Repository** tree view, and select **Retrieve schema** from the contextual menu.

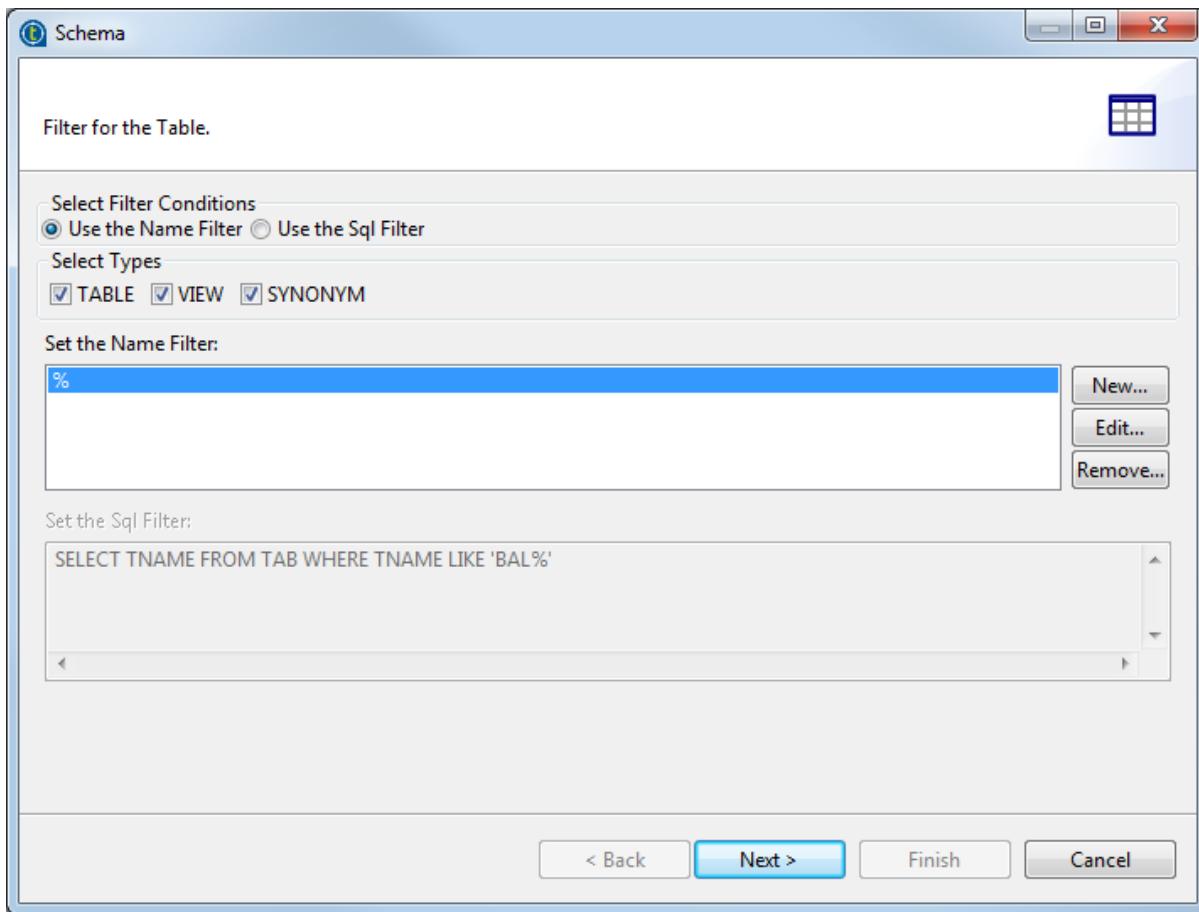


An error message will appear if there are no tables to retrieve from the selected database or if you do not have the correct rights to access this database.

A new wizard opens up where you can filter and show different objects (tables, views and synonyms) in your database connection, select tables of interest, and define table schemas.



For the time being, synonyms option works for Oracle, IBM DB2 and MSSQL only.



### 7.2.2.1. Filtering database objects

In the **Select Filter Conditions** area, you can filter the database objects using either of the two options: **Set the Name Filter** or **Use the Sql Filter** to filter tables based on objects names or using SQL queries respectively.

To filter database tables based on their names, do the following:

1. In the **Select Filter Conditions** area, select the **Use the Name Filter** option.
2. In the **Select Types** area, select the check box(es) of the database object(s) you want to filter or display.



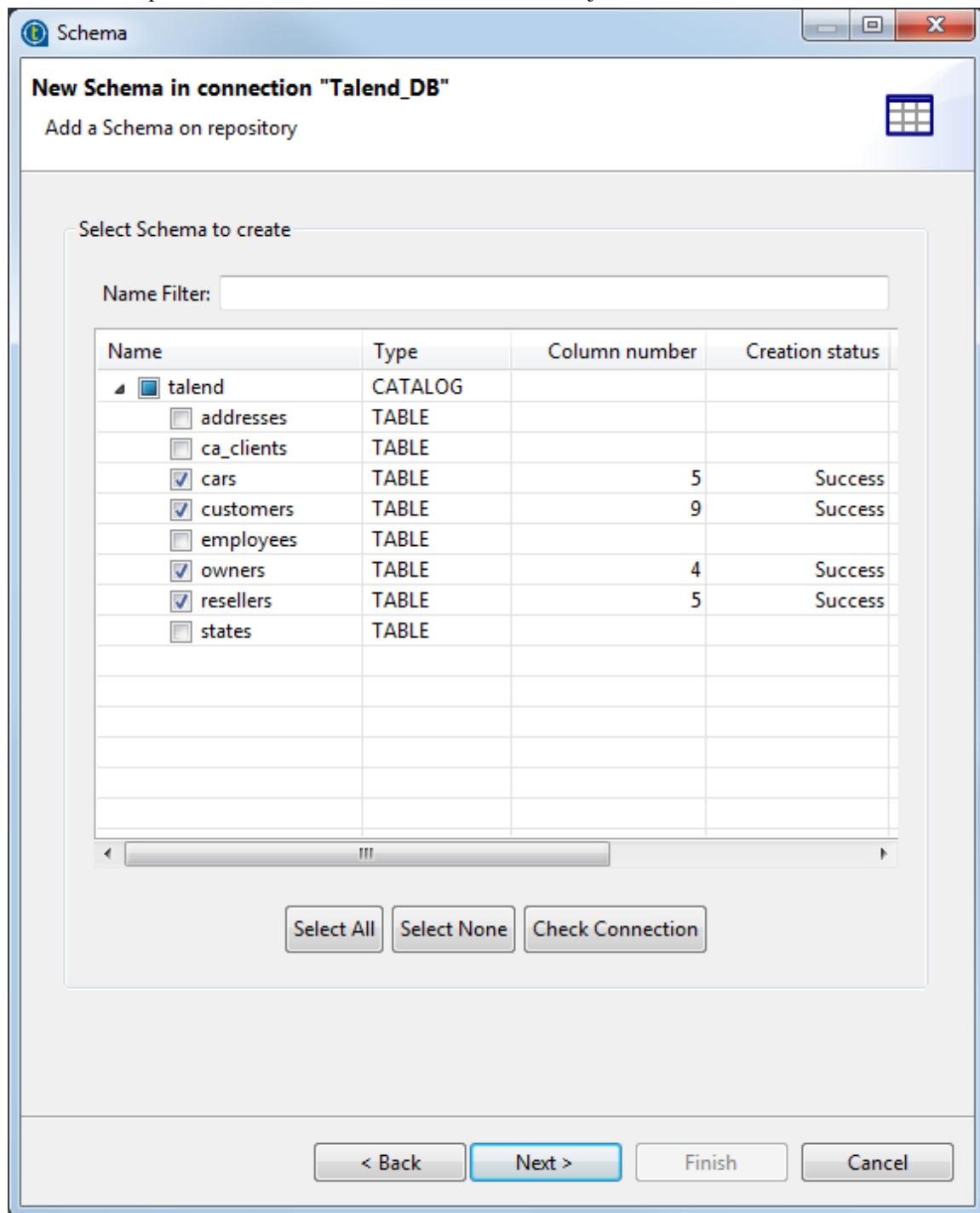
Available options can vary according to the selected database.

3. In the **Set the Name Filter** area, click **Edit...** to open the **[Edit Filter Name]** dialog box.
4. Enter the filter you want to use in the dialog box. For example, if you want to recuperate the database objects which names start with "A", enter the filter "A%", or if you want to recuperate all database objects which names end with "type", enter "%type" as your filter.
5. Click **OK** to close the dialog box.
6. Click **Next** to open a new view on the wizard that lists the filtered database objects.

To filter database objects using an SQL query, do the following:

1. In the **Select Filter Conditions** area, select the **Use Sql Filter** option.

2. In the **Set the Sql Filter** field, enter the SQL query you want to use to filter database objects.
3. Click **Next** to open a new view that lists the filtered database objects.



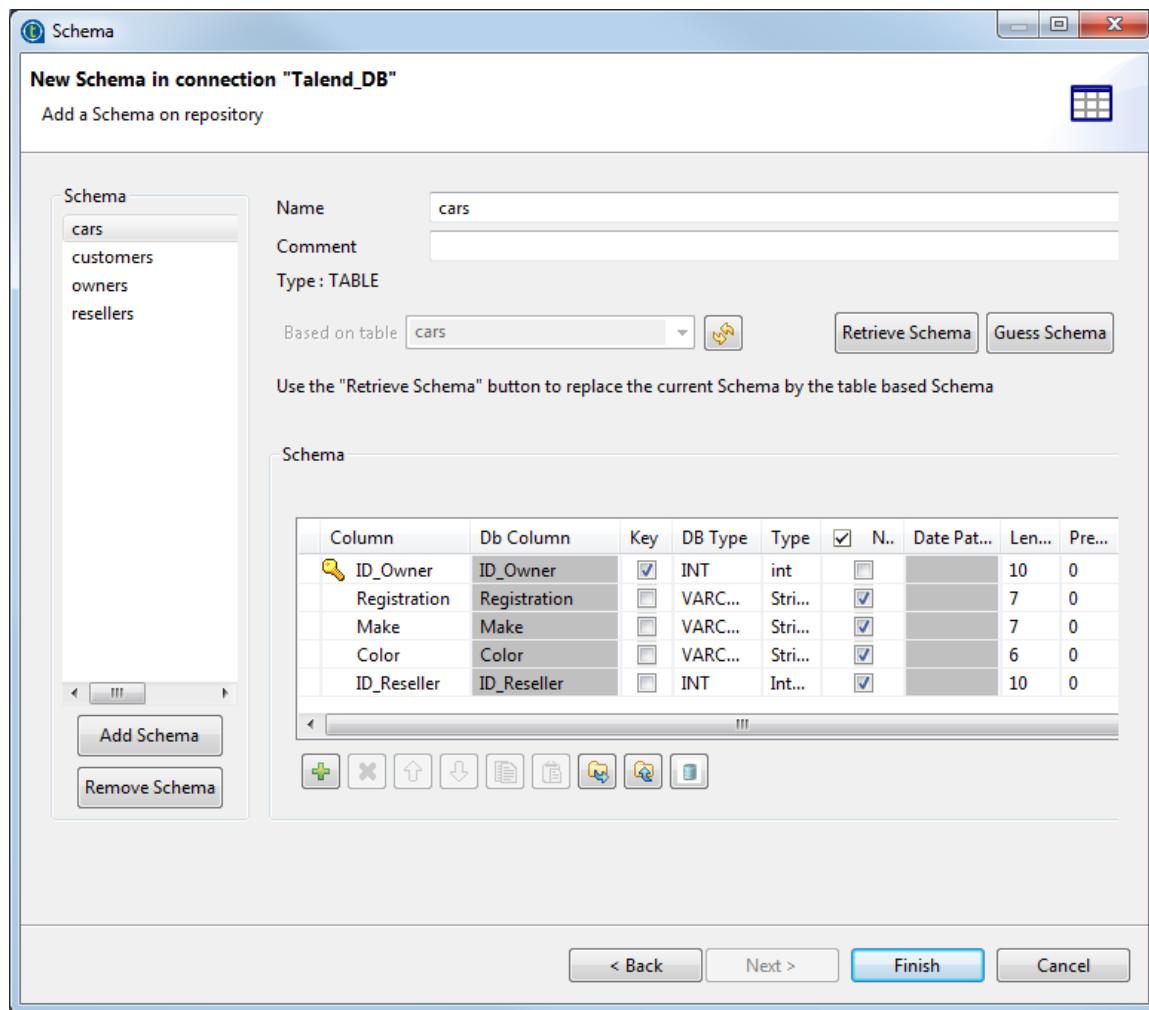
### 7.2.2.2. Selecting tables and defining table schemas

Once you have the filtered list of the database objects (tables, views and synonyms), do the following to load the schemas of the desired objects onto your Repository:

1. Select one or more database objects on the list and click **Next** to open a new view on the wizard where you can see the schemas of the selected object.



If no schema is visible on the list, click the **Check connection** button below the list to verify the database connection status.



## 2. Modify the schemas if needed.

Make sure the data type in the **Type** column is correctly defined.

For more information regarding Java data types, including date pattern, see [Java API Specification](#).

Below are the commonly used **Talend** data types:

- Object: a generic **Talend** data type that allows processing data without regard to its content, for example, a data file not otherwise supported can be processed with a **tFileInputRaw** component by specifying that it has a data type of Object.
- List: a space-separated list of primitive type elements in an XML Schema definition, defined using the xsd:list element.
- Document: a data type that allows processing an entire XML document without regarding to its content.



*If your source database table contains any default value that is a function or an expression rather than a string, be sure to remove the single quotation marks, if any, enclosing the default value in the end schema to avoid unexpected results when creating database tables using this schema.*

By default, the schema displayed on the **Schema** panel is based on the first table selected in the list of schemas loaded (left panel). You can change the name of the schema and according to your needs. You can also customize the schema structure in the schema panel.

The tool bar allows you to add, remove or move columns in your schema. In addition, you can load an XML schema from a file or export the current schema as XML.

To retrieve a schema based on one of the loaded table schemas, select the DB table schema name in the drop-down list and click **Retrieve schema**. Note that the retrieved schema then overwrites any current schema and does not retain any custom edits.

When done, click **Finish** to complete the database schema creation. All the retrieved schemas are displayed in the **Table schemas** sub-folder under the relevant database connection node.

Now you can drag and drop any table schema of the database connection from the **Repository** tree view onto the design workspace as a new database component or onto an existing component to reuse the metadata. For more information, see [How to use centralized metadata in a Job](#) and [How to set a repository schema](#).

## 7.3. Centralizing JDBC metadata

To centralize DB table based metadata into a JDBC connection under the **Metadata** node of the **Repository** tree view, the procedure is made of two separate but closely related tasks:

1. Set up a JDBC connection,
2. Retrieve the table schemas.

The sections below describe how to complete the tasks in detail.

### 7.3.1. Setting up a JDBC connection

1. To create a JDBC connection from scratch, expand **Metadata** in the **Repository** tree view, right-click **Db Connections** and select **Create connection** from the contextual menu to open the database connection setup wizard.

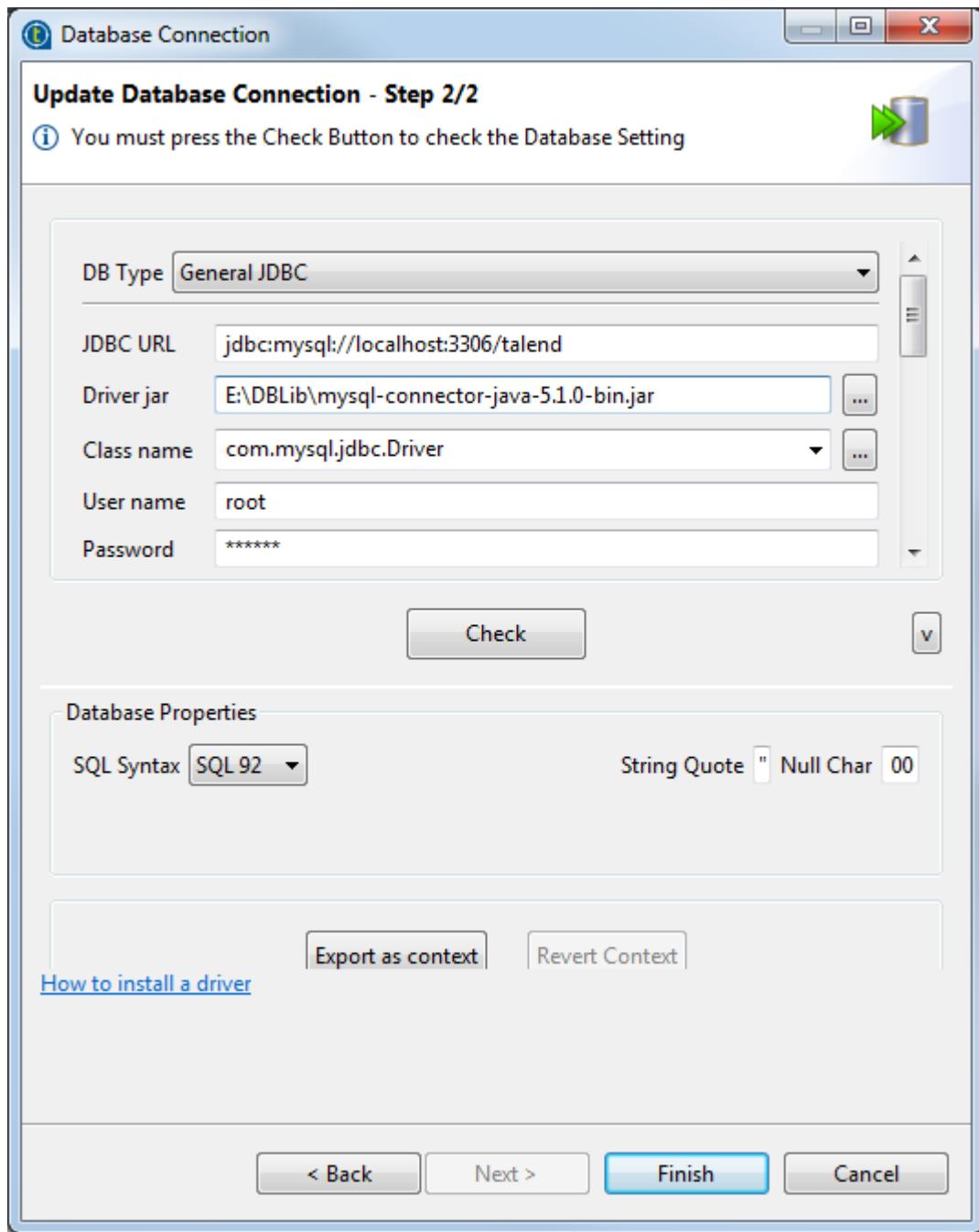
To centralize database connection parameters you have defined in a Job into a JDBC connection, click the  icon in the **Basic settings** view of the relevant database component with its **Property Type** set to **Built-in** to open the database connection setup wizard.

To modify an existing JDBC connection, right-click the connection item from the **Repository** tree view, and select **Edit connection** to open the connection setup wizard.

2. Fill in the schema generic information, such as the connection **Name** and **Description**, and then click **Next** to proceed to define the connection details.

For further information, see the section on defining general properties in [Setting up a database connection](#).

3. Select **General JDBC** from the **DB Type** list.



4. Fill in the connection details as follows:

- Fill in the **JDBC URL** used to access the database server.
- In the **Driver jar** field, select the jar driver validating your connection to the database.
- In the **Class name** field, fill in the main class of the driver allowing to communicate with the database.
- Fill in your **User name** and **Password**.
- Fill the **Mapping File** field with the mapping allowing the database Type to match the Java type of data on the schema by clicking the [...] button to open a dialog box and selecting the mapping file from the **Mapping list** area according to the type of database you are connecting to.



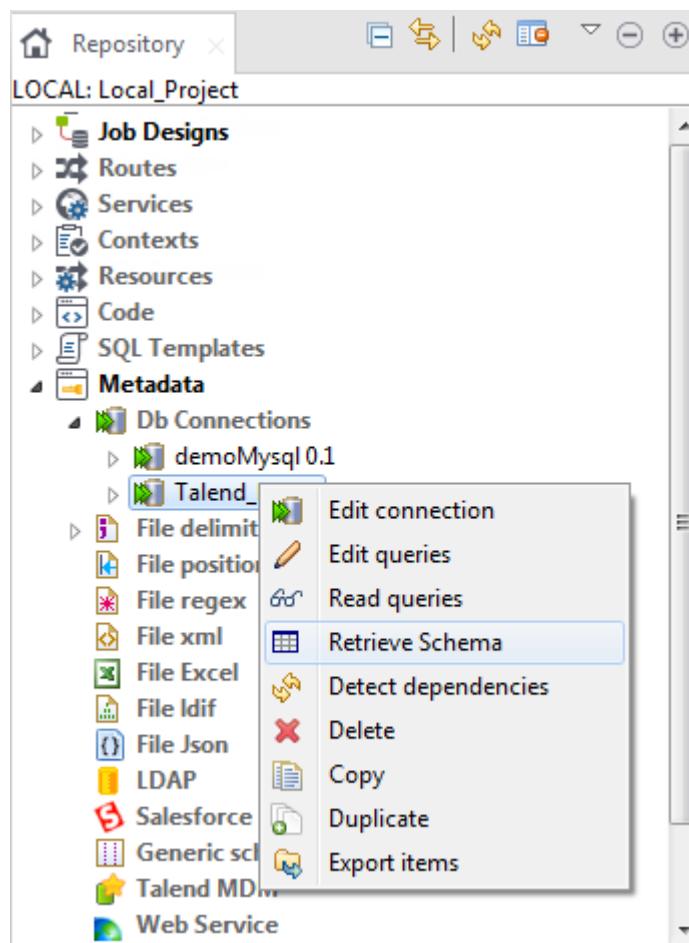
The mapping files are XML files that you can manage via **Window > Preferences > Talend > Specific Settings > Metadata of TalendType**. For more information, see [Type mapping](#).

5. Click **Check** to check your connection.
6. Fill in, if needed, the database properties information. Click **Finish** to close the connection setup wizard.

The newly created JDBC connection is now available in the **Repository** tree view and it displays several folders including **Queries** (for the SQL queries you save) and **Table schemas** that will gather all schemas linked to this DB connection upon schema retrieval.

### 7.3.2. Retrieving table schemas

1. To retrieve table schemas from the database connection you have just set up, right-click the connection item from the **Repository** tree view and select **Retrieve schema** from the contextual menu.



A new wizard opens up where you can filter and show different objects (tables, views and synonyms) in your database connection, select tables of interest, and define table schemas.

2. Define a filter to filter databases objects according to your need. For details, see [Filtering database objects](#).

Click **Next** to open a view that lists your filtered database objects. The list offers all the databases with all their tables present on the database connection that meet your filter conditions.

If no database is visible on the list, click **Check connection** to verify the database connection.

3. Select one or more tables on the list to load them onto your repository file system. Your repository schemas will be based on these tables.
4. Click **Next**. On the next window, four setting panels help you define the schemas to create. Modify the schemas if needed.

Make sure the data type in the **Type** column is correctly defined.

For more information regarding Java data types, including date pattern, see [Java API Specification](#).

Below are the commonly used **Talend** data types:

- Object: a generic **Talend** data type that allows processing data without regard to its content, for example, a data file not otherwise supported can be processed with a **tFileInputRaw** component by specifying that it has a data type of Object.
- List: a space-separated list of primitive type elements in an XML Schema definition, defined using the xsd:list element.
- Document: a data type that allows processing an entire XML document without regarding to its content.



*If your source database table contains any default value that is a function or an expression rather than a string, be sure to remove the single quotation marks, if any, enclosing the default value in the end schema to avoid unexpected results when creating database tables using this schema.*

By default, the schema displayed on the Schema panel is based on the first table selected in the list of schemas loaded (left panel). You can change the name of the schema and according to your needs, you can also customize the schema structure in the schema panel.

The tool bar allows you to add, remove or move columns in your schema. In addition, you can load an XML schema from a file or export the current schema as XML.

To retrieve a schema based on one of the loaded table schemas, select the database table schema name in the drop-down list and click **Retrieve schema**. Note that the retrieved schema then overwrites any current schema and does not retain any custom edits.

When done, click **Finish** to complete the database schema creation. All the retrieved schemas are displayed in the **Table schemas** sub-folder under the relevant database connection node.

Now you can drag and drop any table schema of the database connection from the **Repository** tree view onto the design workspace as a new database component or onto an existing component to reuse the metadata. For more information, see [How to use centralized metadata in a Job](#) and [How to set a repository schema](#).

## 7.4. Centralizing SAS metadata

If you often need to connect to a remote SAS system, you can centralize the connection information in the **Repository**.

To centralize the metadata information of a SAS connection in the **Repository**, you need to complete two major tasks:

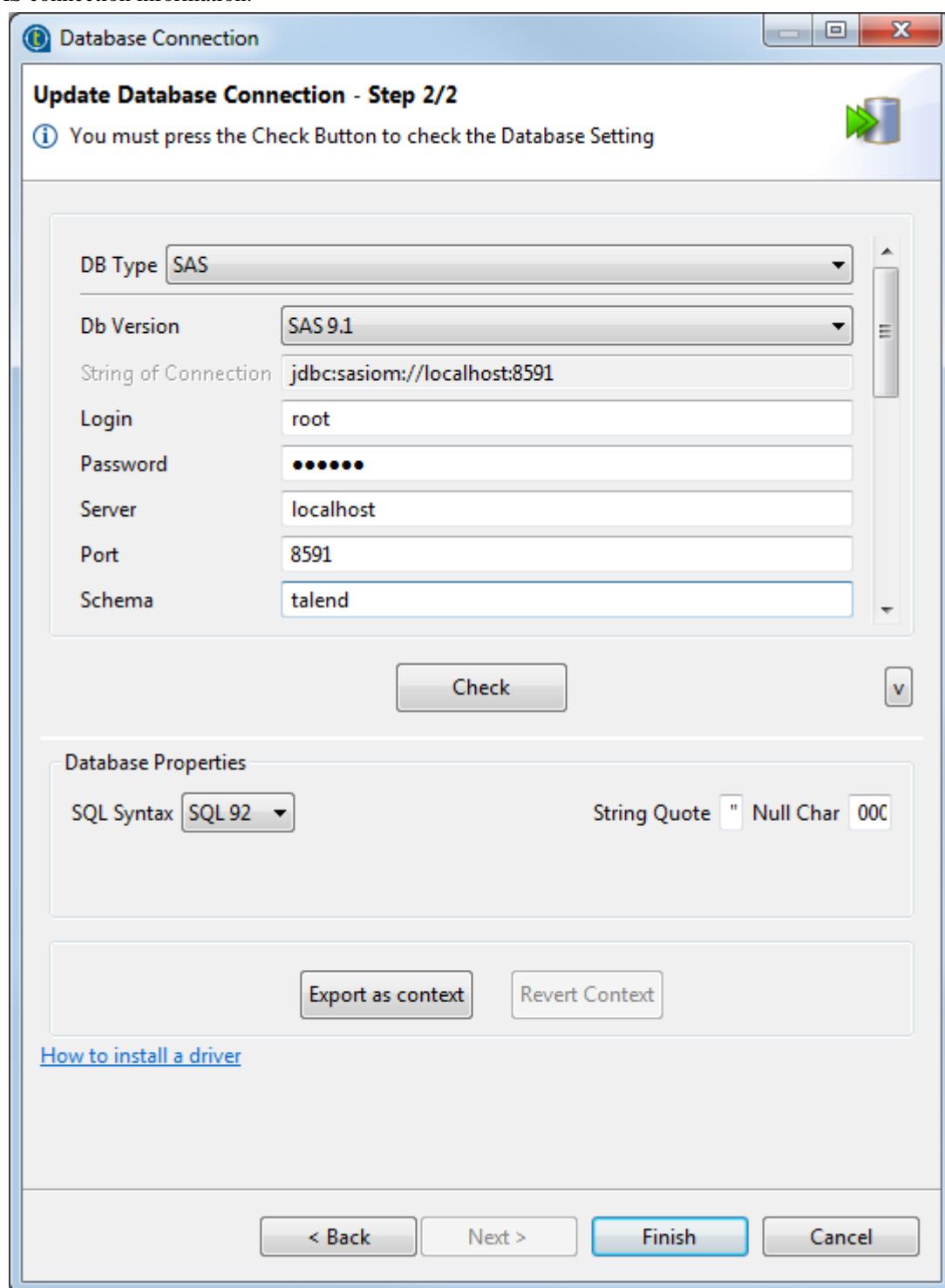
1. Set up a SAS connection,
2. Retrieve the database schemas.

### Prerequisites:

- *Talend Studio* requires specific third-party Java libraries or database drivers (*.jar* files) to be installed in order to connect to sources or targets. Due to license restrictions, **Talend** may not be able to ship certain required libraries or drivers; in that situation, the connection wizard to be presented in the following sections displays related information to help you identify and install the libraries or drivers in question. For more information, see the *Talend Installation and Upgrade Guide*.
- Before carrying on the procedure below to configure your SAS connection, make sure that you retrieve your metadata from the SAS server and export it in XML format.

## 7.4.1. Setting up a SAS connection

1. In the **Repository** tree view of *Talend Studio*, right-click **DB Connections** under the **Metadata** node and select **Create connection** from the contextual menu to open the **[Database Connection]** wizard.
2. Fill in the general properties of the connection, such as **Name** and **Description** and click **Next** to open a new view on the wizard to define the connection details.  
For further information, see the section on defining general properties in [Setting up a database connection](#).
3. In the **DB Type** field of the **[Database Connection]** wizard, select **SAS** and fill in the fields that follow with SAS connection information.

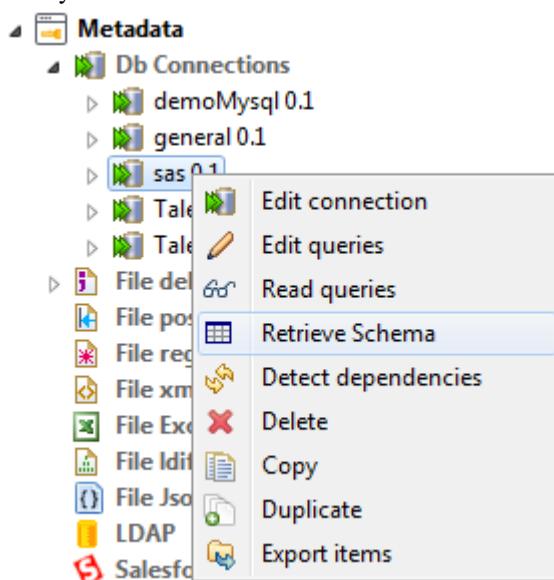


4. If needed, click the **Check** tab to verify if your connection is successful.
5. If needed, define the properties of the database in the corresponding fields in the **Database Properties** area.
6. Click **Finish** to validate your changes and close the wizard.

The newly set connection to the defined database displays under the **DB Connections** folder in the **Repository** tree view. This connection has several sub-folders among which **Table schemas** will group all schemas relative to this connection after schema retrieval.

## 7.4.2. Retrieving table schemas

1. Right-click the SAS connection you created and then select **Retrieve Schema** from the contextual menu.



A new wizard opens up where you can filter and show different objects (tables, views) in your database connection, select tables of interest, and define table schemas.

2. Filter databases objects according to your need, select one or more tables of interest, and modify the table schemas if needed. For details, see [Retrieving table schemas](#).

Make sure the data type in the **Type** column is correctly defined.

For more information regarding Java data types, including date pattern, see [Java API Specification](#).

Below are the commonly used **Talend** data types:

- Object: a generic **Talend** data type that allows processing data without regard to its content, for example, a data file not otherwise supported can be processed with a **tFileInputRaw** component by specifying that it has a data type of Object.
- List: a space-separated list of primitive type elements in an XML Schema definition, defined using the xsd:list element.
- Document: a data type that allows processing an entire XML document without regarding to its content.

When done, you can drag and drop any table schema of the SAS connection from the **Repository** tree view onto the design workspace as a new component or onto an existing component to reuse the metadata. For more information, see [How to use centralized metadata in a Job](#) and [How to set a repository schema](#).

## 7.5. Centralizing File Delimited metadata

If you often need to read data from and/or write data to delimited files, you may want to centralize their metadata in the **Repository** for easy reuse. File Delimited metadata can be used to define the properties of **tFileInputDelimited**, **tFileOutputDelimited**, and **t\*OutputBulk** components.



The file schema creation is very similar for all types of file connections: Delimited, Positional, Regex, XML, or Ldif.

Unlike the database connection wizard, the **[New Delimited File]** wizard gathers both file connection and schema definitions in a four-step procedure.

To create a File Delimited connection from scratch, expand **Metadata** in the **Repository** tree view, right-click **File Delimited** and select **Create file delimited** from the contextual menu to open the file metadata setup wizard.

To centralize a file connection and its schema you have defined in a Job, click the icon in the **Basic settings** view of the relevant component with its **Property Type** set to **Built-in** to open the file metadata setup wizard.

Then define the general properties and file schema in the wizard.

### Defining the general properties

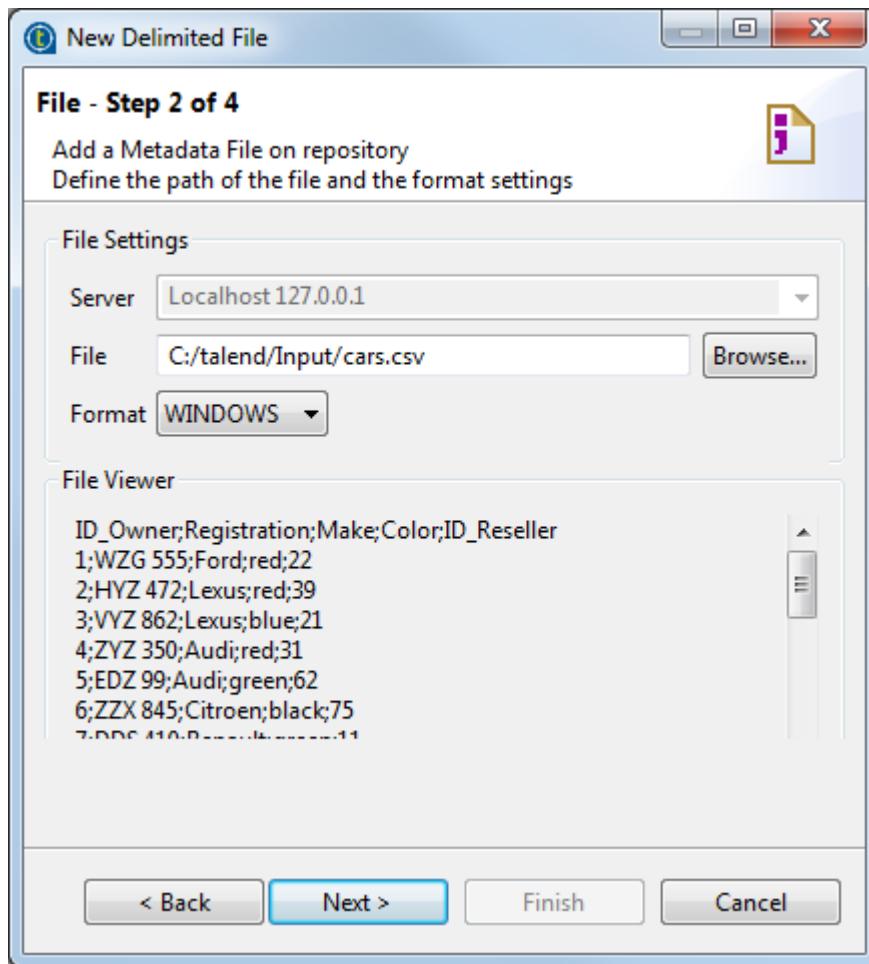
- In the file metadata setup wizard, fill in the **Name** field, which is mandatory, and the **Purpose** and **Description** fields if you choose to do so. The information you provide in the **Description** field will appear as a tooltip when you move your mouse pointer over the file connection.

Name	Cars
Purpose	Centralize cars.csv metadata for reuse.
Description	File connection and schema of cars.csv
Author	user@talend.com
Locker	
Version	0.1
Status	
Path	Select

2. If needed, set the version and status in the **Version** and **Status** fields respectively. You can also manage the version and status of a repository item in the **[Project Settings]** dialog box. For more information, see [Version management](#) and [Status management](#) respectively.
3. If needed, click the **Select** button next to the **Path** field to select a folder under the **File delimited** node to hold your newly created file connection. Note that you cannot select a folder if you are editing an existing connection, but you can drag and drop it to a new folder whenever you want.
4. Click **Next** when completed with the general properties.

## Defining the file path and format

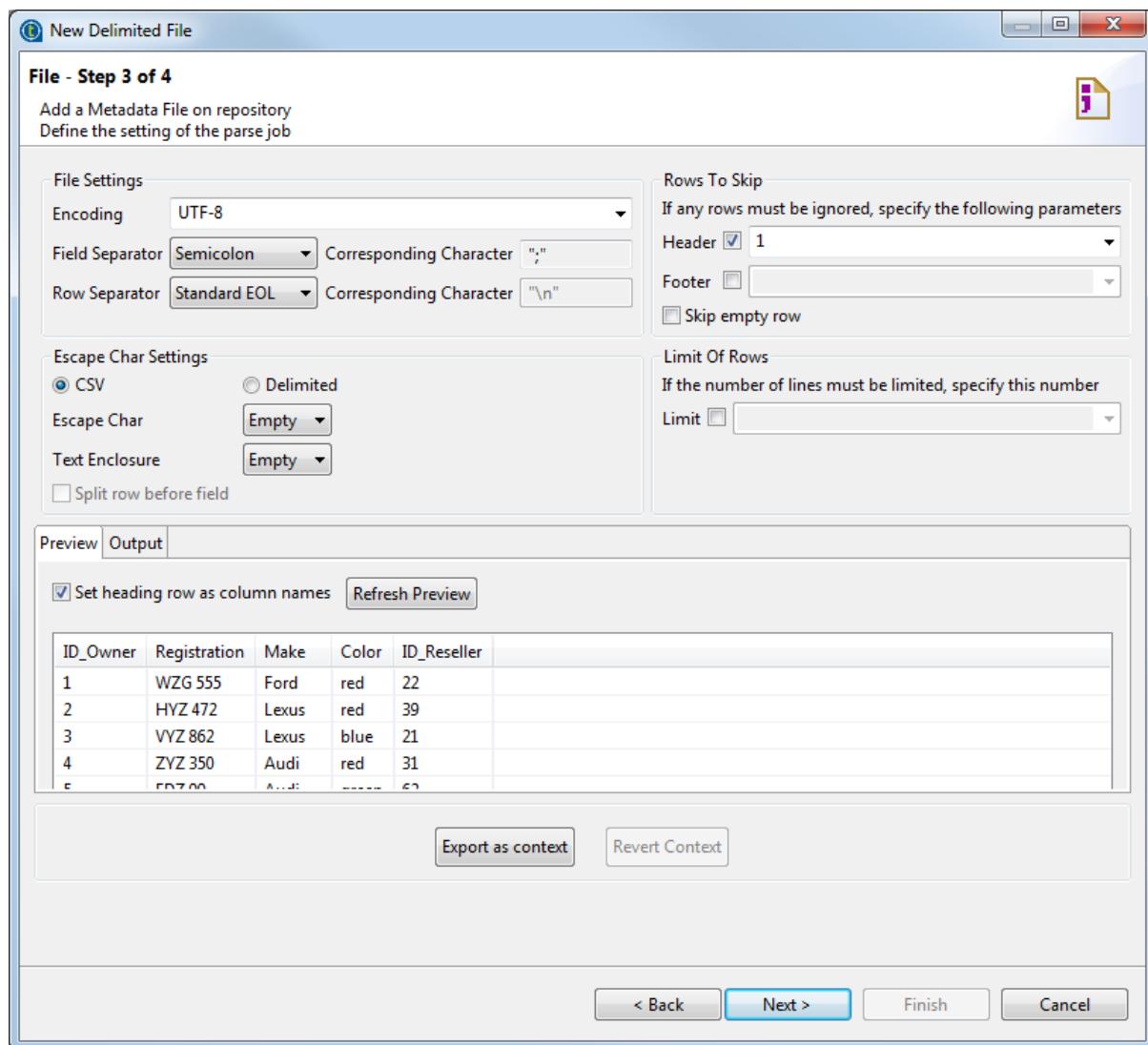
1. Click the **Browse...** button to search for the file on the local host or a LAN host.



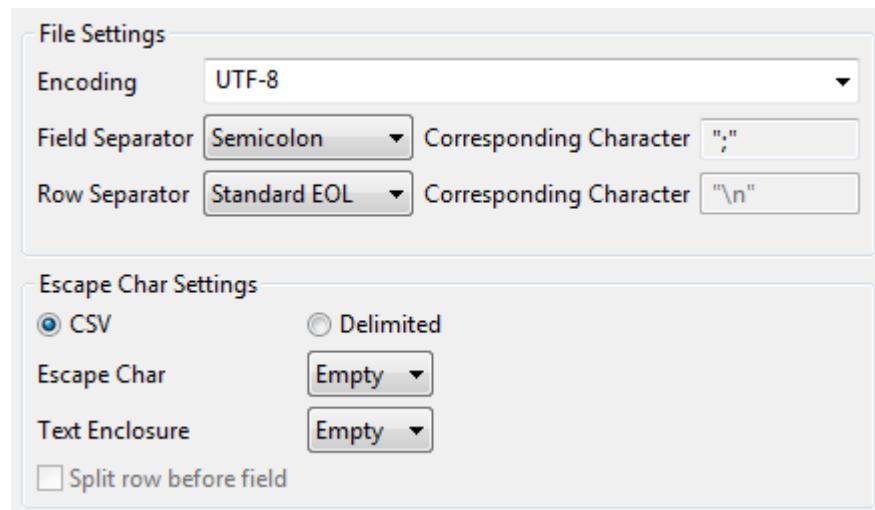
2. Select the OS **Format** the file was created in. This information is used to prefill subsequent step fields. If the list doesn't include the appropriate format, ignore it.
3. The **File viewer** gives an instant picture of the file loaded. Check the file consistency, the presence of header and more generally the file structure.
4. Click **Next** to proceed to the next step.

## Defining the file parsing parameters

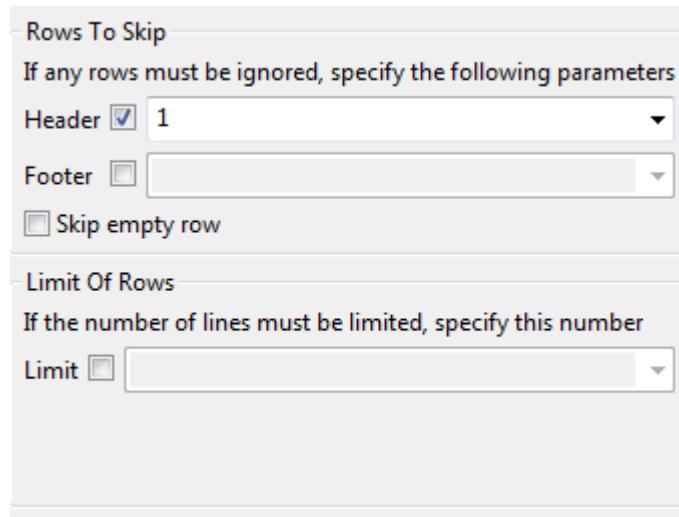
On this view, you can refine the various settings of your file so that the file schema can be properly retrieved.



1. Set the Encoding type, and the Field and Row separators in the **File Settings** area.



2. Depending on your file type (csv or delimited), set the Escape and Enclosure characters to be used.
3. If the file preview shows a header message, exclude the header from the parsing. Set the number of header rows to be skipped. Also, if you know that the file contains footer information, set the number of footer lines to be ignored.



4. The **Limit of Rows** allows you to restrict the extend of the file being parsed. If needed, select the **Limit** check box and set or select the desired number of rows.
5. In the **File Preview** panel, view the new settings impact.
6. Check the **Set heading row as column names** box to transform the first parsed row as labels for schema columns. Note that the number of header rows to be skipped is then incremented by 1.

**Preview** **Output**

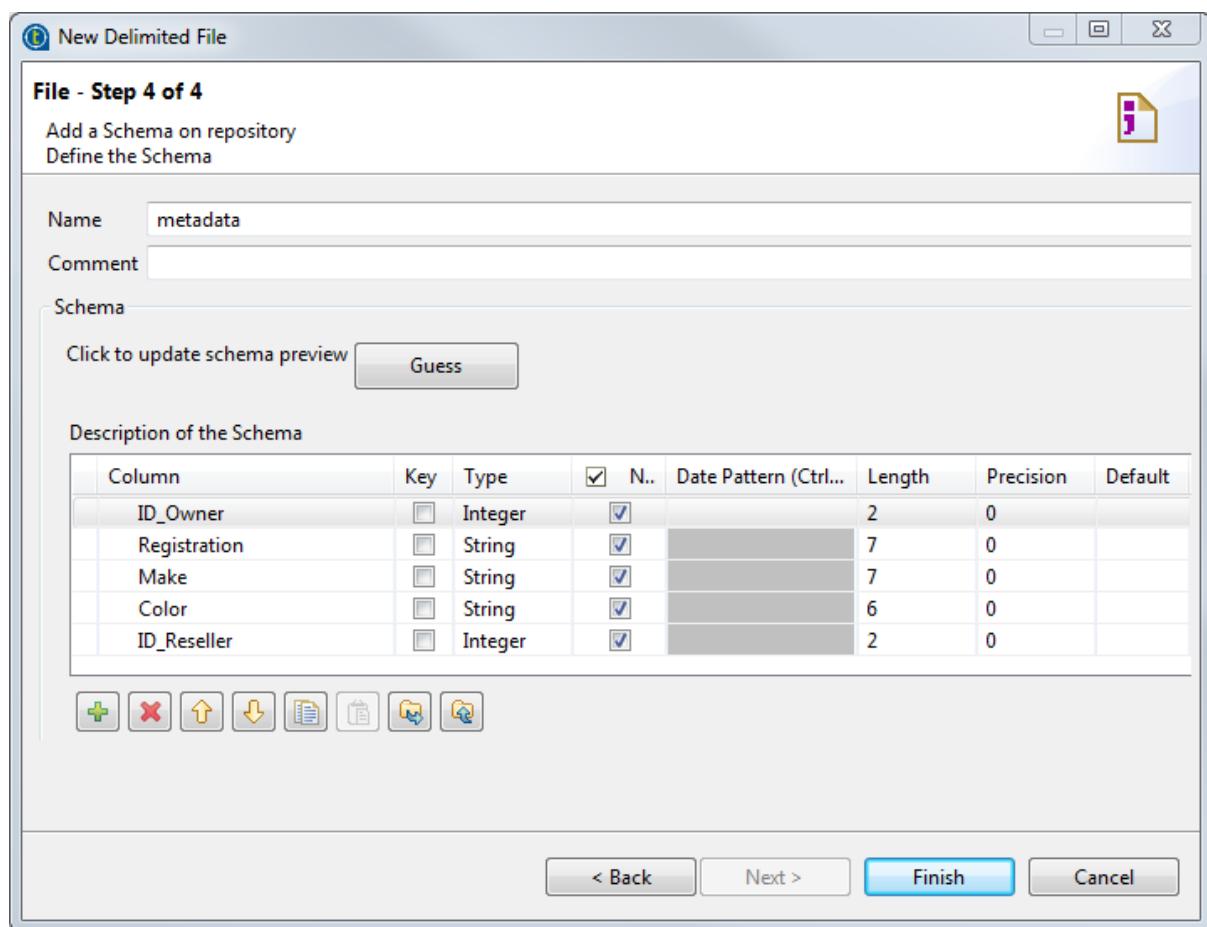
Set heading row as column names **Refresh Preview**

ID_Owner	Registration	Make	Color	ID_Reseller
1	WZG 555	Ford	red	22
2	HYZ 472	Lexus	red	39
3	VYZ 862	Lexus	blue	21
4	ZYZ 350	Audi	red	31
5	EDZ 99	Audi	green	62
6	ZZX 845	Citroen	black	75

7. Click **Refresh** on the preview panel for the settings to take effect and view the result on the viewer.
8. Click **Next** to proceed to the final step to check and customize the generated file schema.

## Checking and customizing the file schema

The last step shows the Delimited File schema generated. You can customize the schema using the toolbar underneath the table.



Make sure the data type in the **Type** column is correctly defined.

For more information regarding Java data types, including date pattern, see [Java API Specification](#).

Below are the commonly used **Talend** data types:

- Object: a generic **Talend** data type that allows processing data without regard to its content, for example, a data file not otherwise supported can be processed with a **tFileInputRaw** component by specifying that it has a data type of Object.
  - List: a space-separated list of primitive type elements in an XML Schema definition, defined using the xsd:list element.
  - Document: a data type that allows processing an entire XML document without regarding to its content.
1. If the Delimited file which the schema is based on has been changed, use the **Guess** button to generate again the schema. Note that if you customized the schema, the **Guess** feature does not retain these changes.
  2. Click **Finish**. The new schema is displayed under the relevant **File Delimited** connection node in the **Repository** tree view.

Now you can drag and drop the file connection or any schema of it from the **Repository** tree view onto the design workspace as a new component or onto an existing component to reuse the metadata. For further information about how to use the centralized metadata in a Job, see [How to use centralized metadata in a Job](#) and [How to set a repository schema](#).

To modify an existing file connection, right-click it from the **Repository** tree view, and select **Edit file delimited** to open the file metadata setup wizard.

To add a new schema to an existing file connection, right-click the connection from the **Repository** tree view and select **Retrieve Schema** from the contextual menu.

To edit an existing file schema, right-click the schema from the **Repository** tree view and select **Edit Schema** from the contextual menu.

## 7.6. Centralizing File Positional metadata

If you often need to read data from and/or write data to certain positional files, you may want to centralize their metadata in the **Repository** for easy reuse. File Positional metadata can be used to define the properties of **tFileInputPositional**, **tFileOutputPositional**, and **tFileInputMSPositional** components.

The **[New Positional File]** wizard gathers both file connection and schema definitions in a four-step procedure.

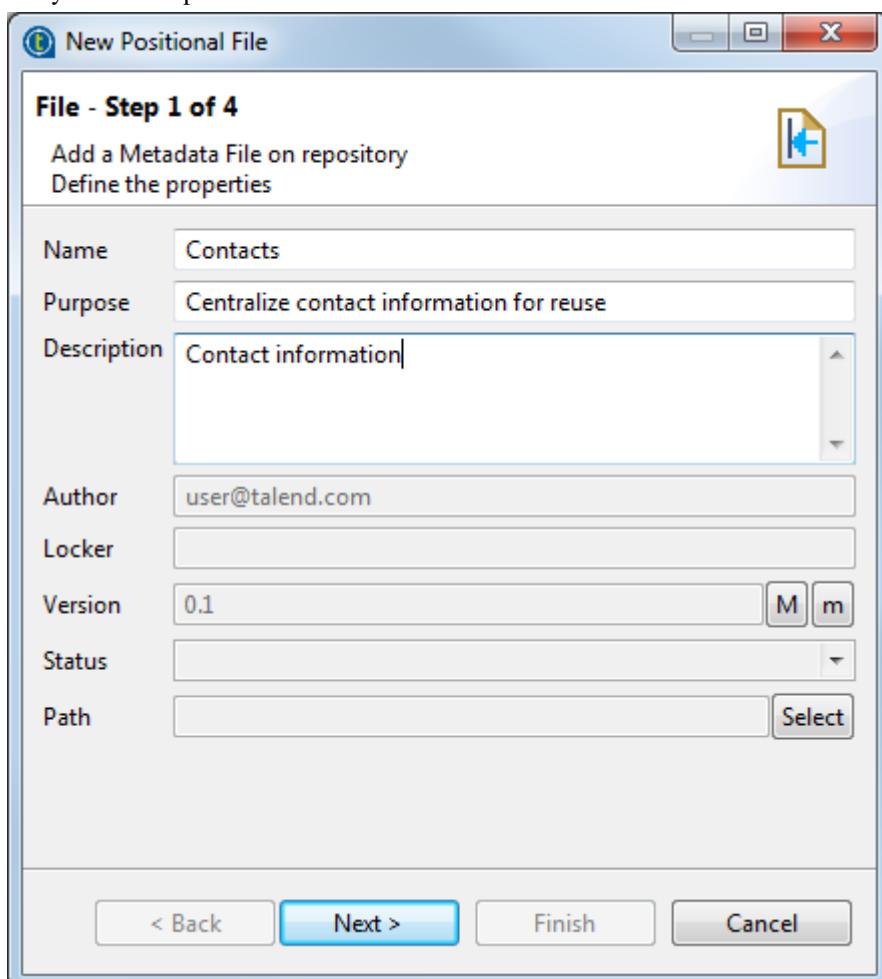
To create a File Positional connection from scratch, expand **Metadata** in the **Repository** tree view, right-click **File positional** and select **Create file positional** from the contextual menu to open the file metadata setup wizard.

To centralize a file connection and its schema you have defined in a Job, click the  icon in the **Basic settings** view of the relevant component with its **Property Type** set to **Built-in** to open the file metadata setup wizard.

Then define the general properties and file schema in the wizard.

### Defining the general properties

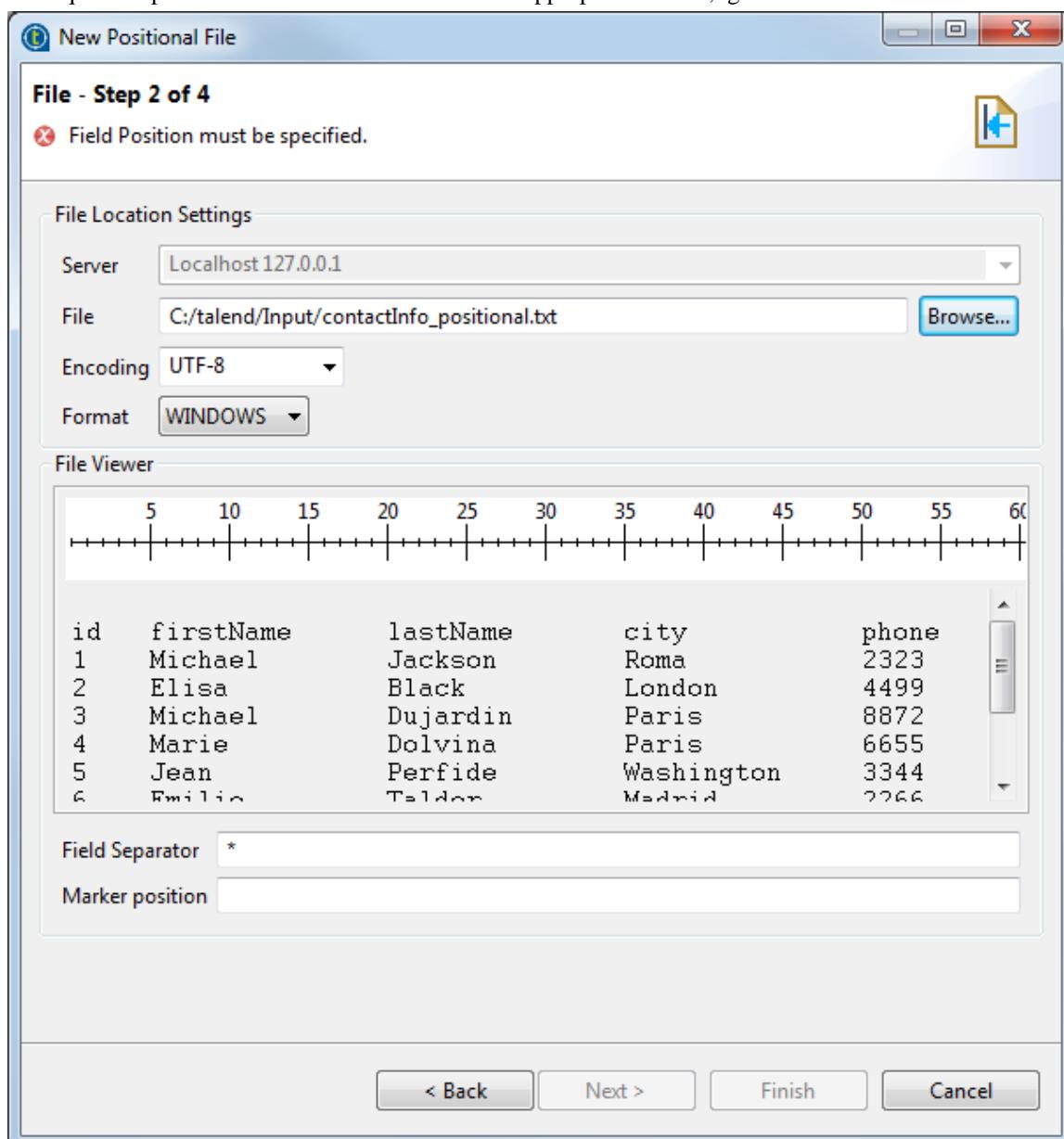
1. In the file metadata setup wizard, fill in the **Name** field, which is mandatory, and the **Purpose** and **Description** fields if you choose to do so. The information you provide in the **Description** field will appear as a tooltip when you move your mouse pointer over the file connection.



2. If needed, set the version and status in the **Version** and **Status** fields respectively. You can also manage the version and status of a **Repository** item in the **[Project Settings]** dialog box. For more information, see [Version management](#) and [Status management](#) respectively.
3. If needed, click the **Select** button next to the **Path** field to select a folder under the **File positional** node to hold your newly created file connection. Note that you cannot select a folder if you are editing an existing connection, but you can drag and drop it to a new folder whenever you want.
4. Click **Next** when completed with the general properties.

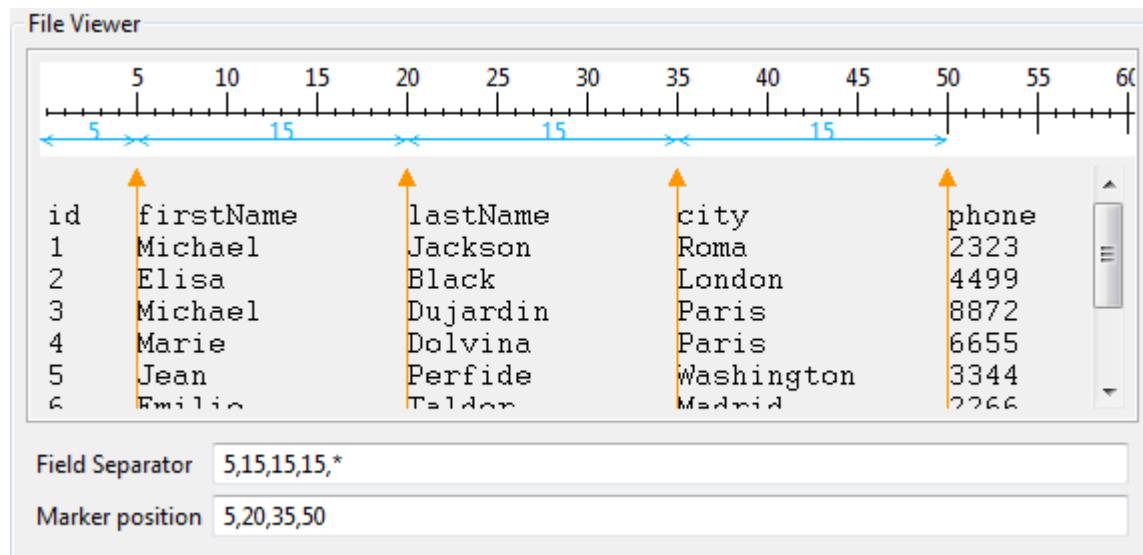
## Defining the file path, format and marker positions

1. Click the **Browse...** button to search for the file on the local host or a LAN host.
2. Select the **Encoding** type and the **OS Format** the file was created in. This information is used to prefill subsequent step fields. If the list doesn't include the appropriate format, ignore the OS format.



The file is loaded and the **File Viewer** area shows a file preview and allows you to place your position markers.

3. Click the file preview and set the markers against the ruler to define the file column properties. The orange arrow helps you refine the position.



The **Field Separator** and **Marker Position** fields are automatically filled with a series of figures separated by commas.

The figures in the **Field Separator** are the number of characters between the separators, which represent the lengths of the columns of the loaded file. The asterisk symbol means all remaining characters on the row, starting from the preceding marker position. You can change the figures to specify the column lengths precisely.

The **Marker Position** field shows the exact position of each marker on the ruler, in units of characters. You can change the figures to specify the positions precisely.

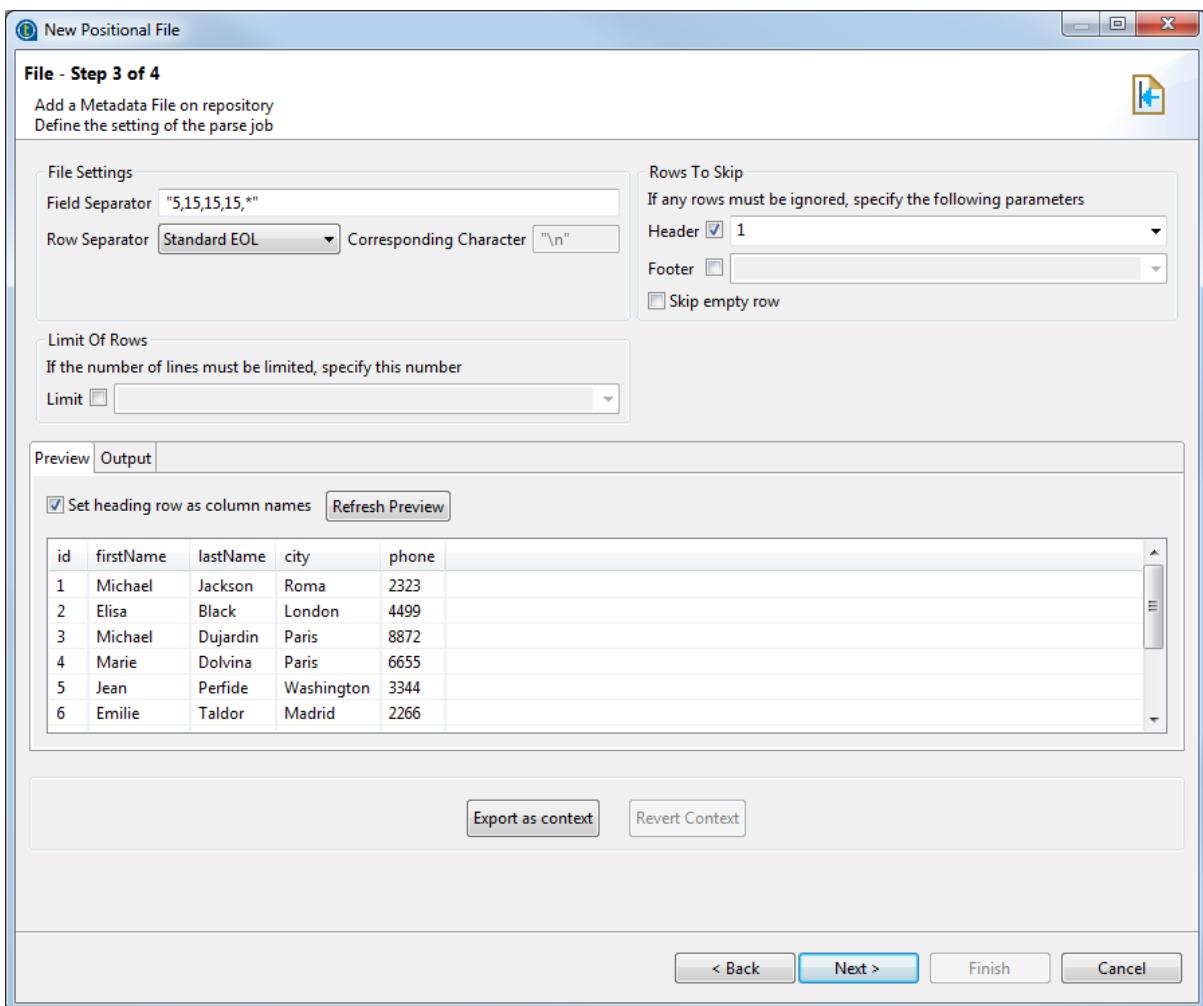
To move a marker, press its arrow and drag it to the new position. To remove a marker, press its arrow and drag it towards the ruler until a icon appears.

4. Click **Next** to continue.

## Defining the file parsing parameters

On this view, you define the file parsing parameters so that the file schema can be properly retrieved.

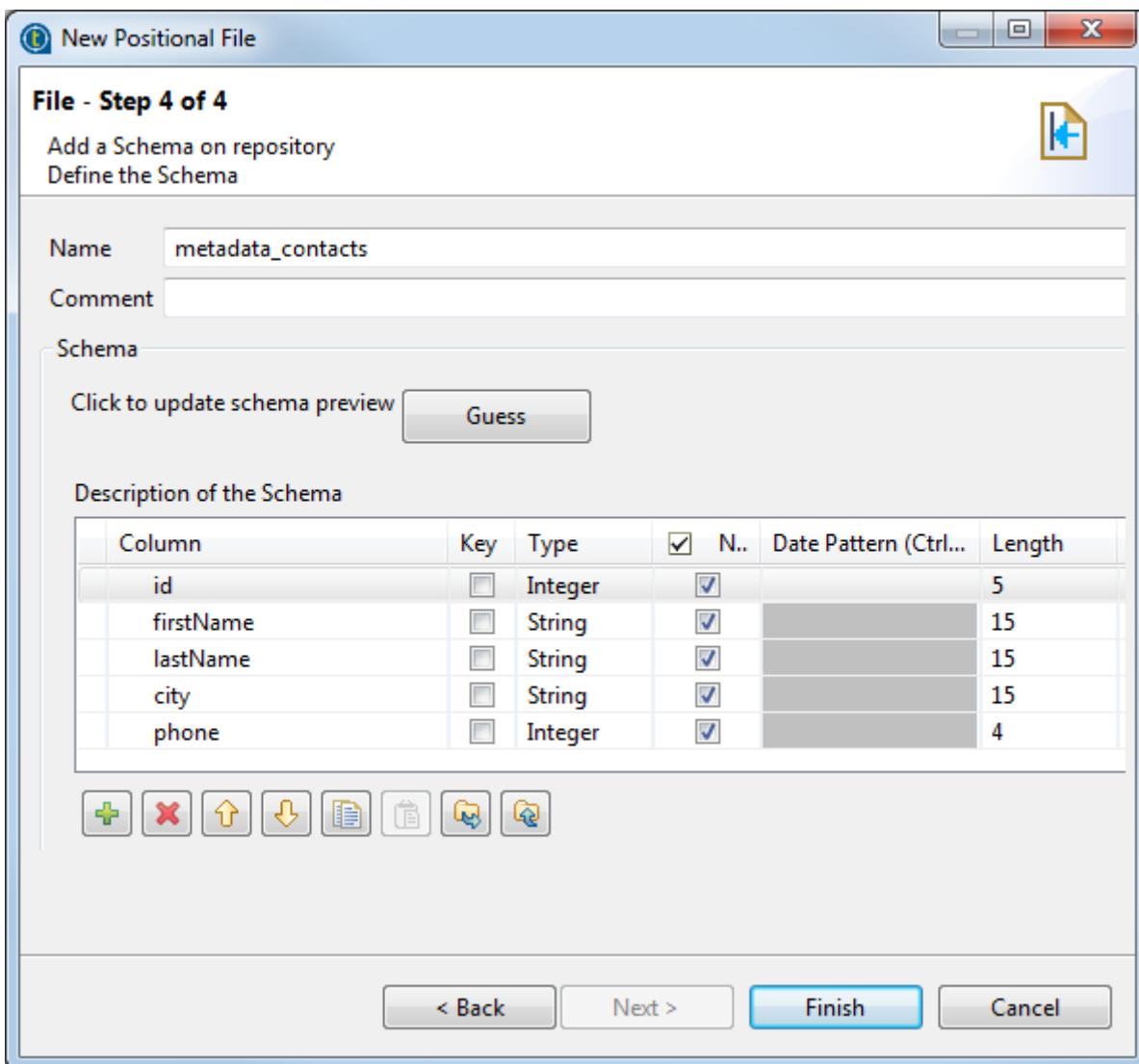
At this stage, the preview shows the file columns upon the markers' positions.



1. Set the Field and Row separators in the **File Settings** area.
  - If needed, change the figures in the **Field Separator** field to specify the column lengths precisely.
  - If the row separator of your file is not the standard EOL (end of line), select **Custom String** from the **Row Separator** list and specify the character string in the **Corresponding Character** field.
2. If your file has any header rows to be excluded from the data content, select the **Header** check box in the **Rows To Skip** area and define the number of rows to be ignored in the corresponding field. Also, if you know that the file contains footer information, select the **Footer** check box and set the number of rows to be ignored.
3. The **Limit of Rows** area allows you to restrict the extend of the file being parsed. If needed, select the **Limit** check box and set or select the desired number of rows.
4. If the file contains column labels, select the **Set heading row as column names** check box to transform the first parsed row to labels for schema columns. Note that the number of header rows to be skipped is then incremented by 1.
5. Click **Refresh Preview** on the **Preview** panel for the settings to take effect and view the result on the viewer.
6. Click **Next** to proceed to the next view to check and customize the generated file schema.

## Checking and customizing the file schema

Step 4 shows the end schema generated. Note that any character which could be misinterpreted by the program is replaced by neutral characters. Underscores replace asterisks, for example.



1. Rename the schema (by default, *metadata*) and edit the schema columns as needed.

Make sure the data type in the **Type** column is correctly defined.

For more information regarding Java data types, including date pattern, see [Java API Specification](#).

Below are the commonly used **Talend** data types:

- Object: a generic **Talend** data type that allows processing data without regard to its content, for example, a data file not otherwise supported can be processed with a **tFileInputRaw** component by specifying that it has a data type of Object.
  - List: a space-separated list of primitive type elements in an XML Schema definition, defined using the xsd:list element.
  - Document: a data type that allows processing an entire XML document without regarding to its content.
2. To generate the Positional File schema again, click the **Guess** button. Note that, however, any edits to the schema might be lost after "guessing" the file-based schema.
  3. When done, click **Finish** to close the wizard.

The new schema is displayed under the relevant **File positional** connection node in the **Repository** tree view. You can drop the defined metadata from the **Repository** onto the design workspace as a new component or onto an

existing component to reuse the metadata. For further information about how to use the centralized metadata in a Job, see [How to use centralized metadata in a Job](#) and [How to set a repository schema](#).

To modify an existing file connection, right-click it from the **Repository** tree view, and select **Edit file positional** to open the file metadata setup wizard.

To add a new schema to an existing file connection, right-click the connection from the **Repository** tree view and select **Retrieve Schema** from the contextual menu.

To edit an existing file schema, right-click the schema from the **Repository** tree view and select **Edit Schema** from the contextual menu.

## 7.7. Centralizing File Regex metadata

Regex file schemas are used for files made of regular expressions, such as log files. If you often need to connect to a regex file, you may want to centralize its connection and schema information in the Repository for easy reuse.

The **[New RegEx File]** wizard gathers both file connection and schema definitions in a four-step procedure.



This procedure requires some advanced knowledge on regular expression syntax.

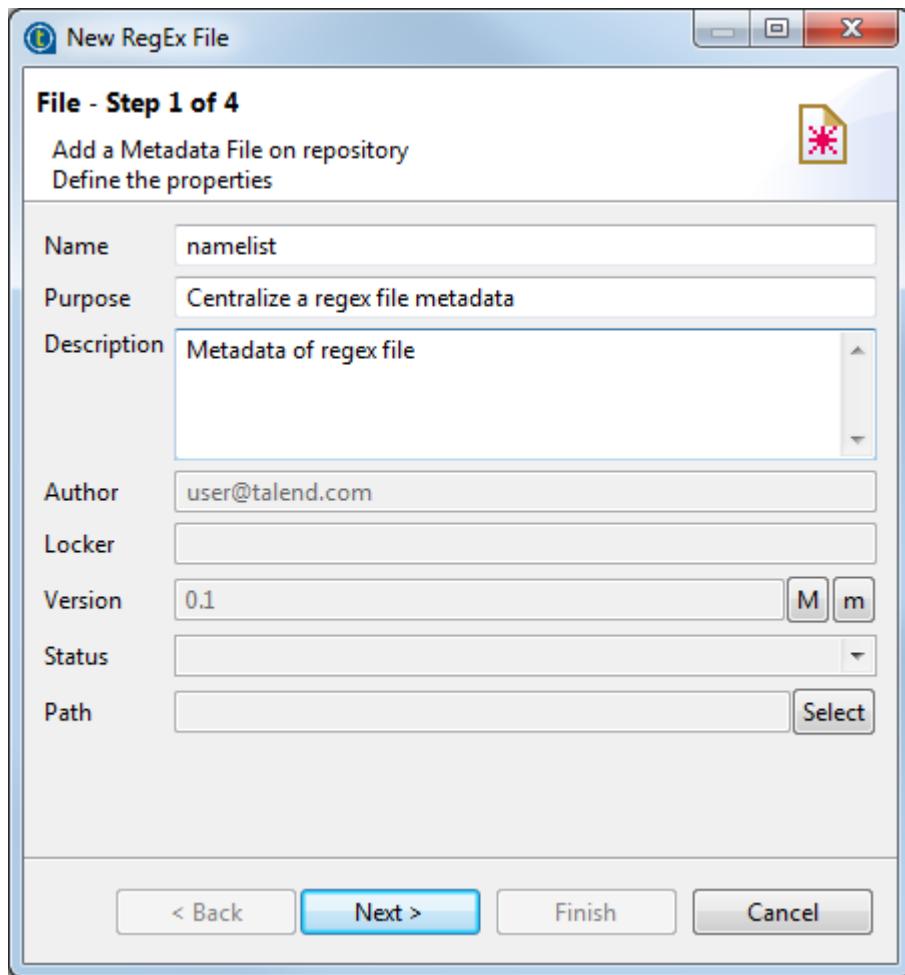
To create a File Regex connection from scratch, expand the **Metadata** node in the **Repository** tree view, right-click **File Regex** and select **Create file regex** from the contextual menu to open the file metadata setup wizard.

To centralize a file connection and its schema you have defined in a Job, click the icon in the **Basic settings** view of the relevant component with its **Property Type** set to **Built-in** to open the file metadata setup wizard.

Then define the general properties and file schema in the wizard.

### Defining the general properties

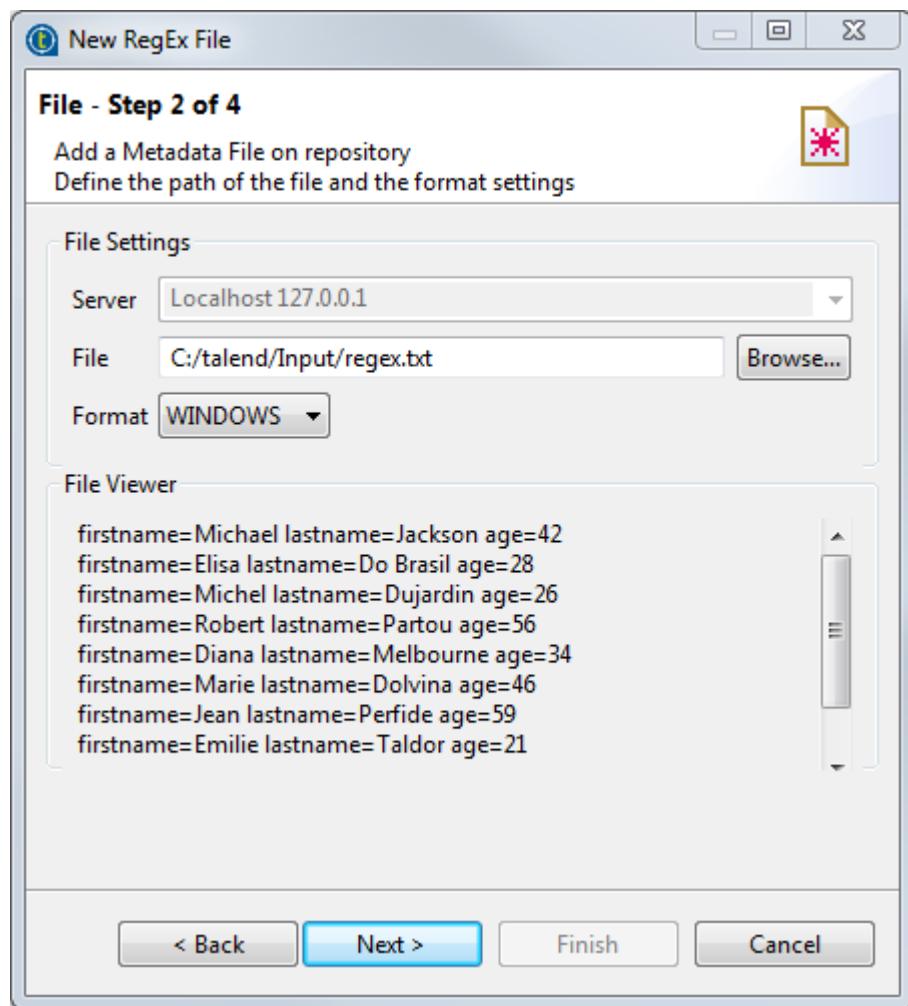
1. In the file metadata setup wizard, fill in the **Name** field, which is mandatory, and the **Purpose** and **Description** fields if you choose to do so. The information you provide in the **Description** field will appear as a tooltip when you move your mouse pointer over the file connection.



2. If needed, set the version and status in the **Version** and **Status** fields respectively. You can also manage the version and status of a repository item in the **[Project Settings]** dialog box. For more information, see [Version management](#) and [Status management](#) respectively.
3. If needed, click the **Select** button next to the **Path** field to select a folder under the **File regex** node to hold your newly created file connection. Note that you cannot select a folder if you are editing an existing connection, but you can drag and drop it to a new folder whenever you want.
4. Click **Next** when completed with the general properties.

## Defining the file path and format

1. Click the **Browse...** button to search for the file on the local host or a LAN host.
2. Select the **Encoding** type and the **OS Format** the file was created in. This information is used to prefill subsequent step fields. If the list doesn't include the appropriate format, ignore the OS format.



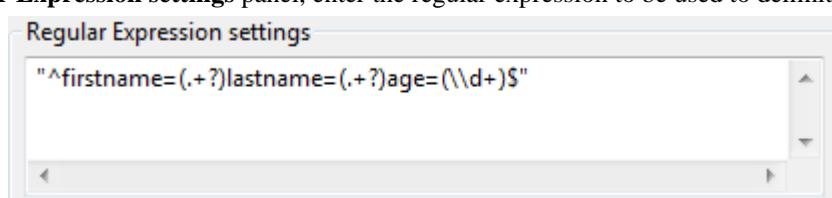
The file viewer gives an instant picture of the loaded file.

- Click **Next** to define the schema structure.

### Defining the file parsing parameters

On this view, you define the file parsing parameters so that the file schema can be properly retrieved.

- Set the Field and Row separators in the **File Settings** area.
  - If needed, change the figures in the **Field Separator** field to specify the column lengths precisely.
  - If the row separator of your file is not the standard EOL, select **Custom String** from the **Row Separator** list and specify the character string in the **Corresponding Character** field.
- In the **Regular Expression settings** panel, enter the regular expression to be used to delimit the file.



*Make sure to include the Regex code in single or double quotes accordingly.*

3. If your file has any header rows to be excluded from the data content, select the **Header** check box in the **Rows To Skip** area and define the number of rows to be ignored in the corresponding field. Also, if you know that the file contains footer information, select the **Footer** check box and set the number of rows to be ignored.
  4. The **Limit of Rows** allows you to restrict the extend of the file being parsed. If needed, select the **Limit** check box and set or select the desired number of rows.
  5. If the file contains column labels, select the **Set heading row as column names** check box to transform the first parsed row to labels for schema columns. Note that the number of header rows to be skipped is then incremented by 1.
  6. Then click **Refresh preview** to take the changes into account. The button changes to **Stop** until the preview is refreshed.

Column 0	Column 1	Column 2
Michael	Jackson	42
Elisa	Do Brasil	28
Michel	Dujardin	26
Robert	Partou	56
Diana	Melbourne	34
Marie	Dolvina	46
.	.	.

7. Click **Next** to proceed to the next view where you can check and customize the generated Regex File schema.

## Checking and customizing the file schema

1. Rename the schema (by default, *metadata*) and edit the schema columns as needed.

Make sure the data type in the **Type** column is correctly defined.

For more information regarding Java data types, including date pattern, see [Java API Specification](#).

Below are the commonly used **Talend** data types:

- Object: a generic **Talend** data type that allows processing data without regard to its content, for example, a data file not otherwise supported can be processed with a **tFileInputRaw** component by specifying that it has a data type of Object.
  - List: a space-separated list of primitive type elements in an XML Schema definition, defined using the xsd:list element.
  - Document: a data type that allows processing an entire XML document without regarding to its content.

2. To retrieve or update the Regex File schema, click **Guess**. Note however that any edits to the schema might be lost after guessing the file based schema.

3. When done, click **Finish** to close the wizard.

The new schema is displayed under the relevant **File regex** node in the **Repository** tree view. You can drop the defined metadata from the **Repository** onto the design workspace as a new component or onto an existing component to reuse the metadata. For further information about how to use the centralized metadata in a Job, see [How to use centralized metadata in a Job](#) and [How to set a repository schema](#).

To modify an existing file connection, right-click it from the **Repository** tree view, and select **Edit file regex** to open the file metadata setup wizard.

To add a new schema to an existing file connection, right-click the connection from the **Repository** tree view and select **Retrieve Schema** from the contextual menu.

To edit an existing file schema, right-click the schema from the **Repository** tree view and select **Edit Schema** from the contextual menu.

## 7.8. Centralizing XML file metadata

If you often need to connect to an XML file, you may want to use the **[New Xml File]** wizard to centralize your connection to the file and the schema retrieved from it in your Repository for easy reuse.

Depending on the option you select, the wizard helps you create either an input or an output file connection. In a Job, the **tFileInputXML** and **tExtractXMLField** components use the input connection created to read XML files, whereas **tAdvancedFileOutputXML** uses the output schema created to either write an XML file, or to update an existing XML file.

For further information about reading an XML file, see [Setting up XML metadata for an input file](#).

For further information about writing an XML file, see [Setting up XML metadata for an output file](#).

To create an XML file connection from scratch, expand the **Metadata** node in the **Repository** tree view, right-click **File XML** and select **Create file XML** from the contextual menu to open the file metadata setup wizard.

To centralize a file connection and its schema you have defined in a Job, click the  icon in the **Basic settings** view of the relevant component with its **Property Type** set to **Built-in** to open the file metadata setup wizard.

Then define the general properties and file schema in the wizard.

### 7.8.1. Setting up XML metadata for an input file

This section describes how to define a file connection and upload an XML schema for an input file. To define and upload an output file, see [Setting up XML metadata for an output file](#).

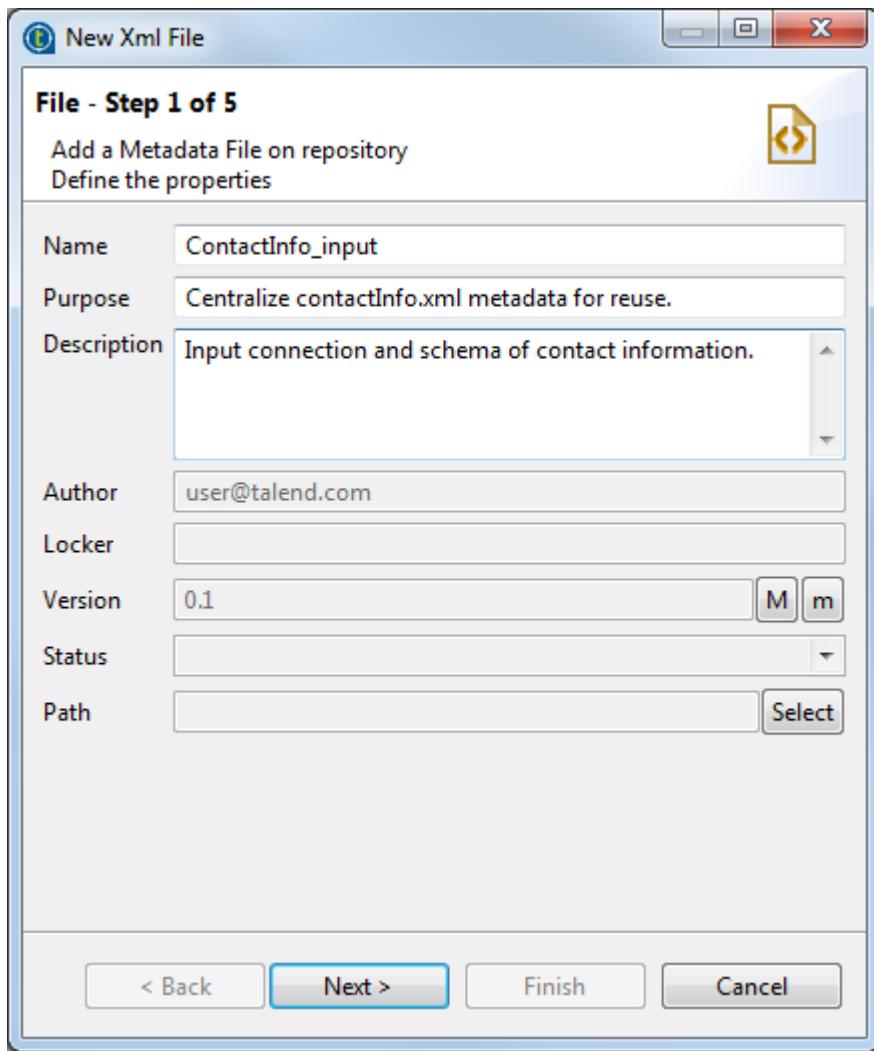
#### Defining the general properties

In this step, the general metadata properties such as the **Name**, **Purpose** and **Description** are set.

1. In the file metadata setup wizard, fill in the **Name** field, which is mandatory, and the **Purpose** and **Description** fields if you choose to do so. The information you provide in the **Description** field will appear as a tooltip when you move your mouse pointer over the file connection.



When you enter the general properties of the metadata to be created, you need to define the type of connection as either input or output. It is therefore advisable to enter information that will help you distinguish between your input and output schemas.

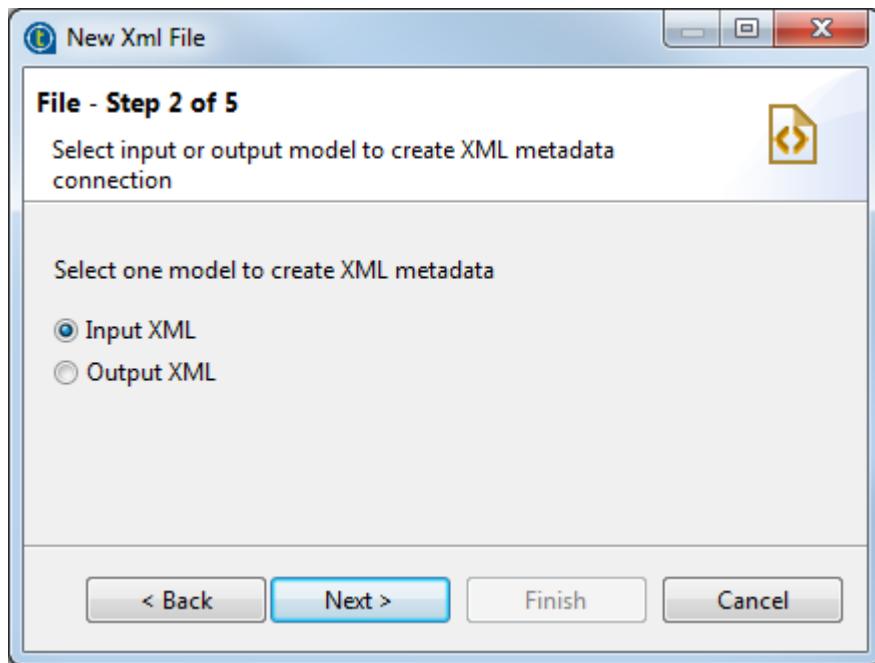


2. If needed, set the version and status in the **Version** and **Status** fields respectively. You can also manage the version and status of a **Repository** item in the **[Project Settings]** dialog box. For more information, see [Version management](#) and [Status management](#) respectively.
3. If needed, click the **Select** button next to the **Path** field to select a folder under the **File XML** node to hold your newly created file connection. Note that you cannot select a folder if you are editing an existing connection, but you can drag and drop it to a new folder whenever you want.
4. Click **Next** to select the type of metadata.

### Setting the type of metadata (input)

In this step, the type of metadata is set as either input or output. For this procedure, the metadata of interest is input.

1. In the dialog box, select **Input XML**.



2. Click **Next** to upload the input file.

## Uploading an XML file

This procedure describes how to upload an XML file to obtain the XML tree structure. To upload an XML Schema Definition (XSD) file, see [Uploading an XSD file](#).

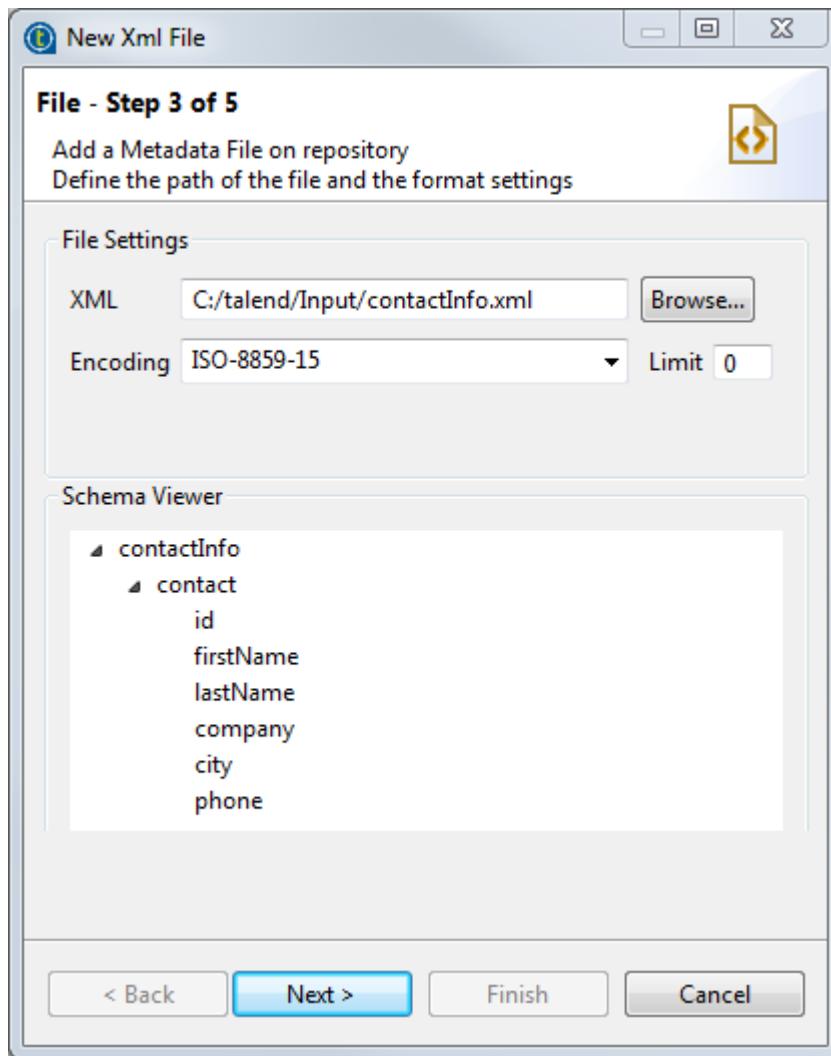
The example input XML file used to demonstrate this step contains some contact information, and the structure is like the following:

```
<contactInfo>
 <contact>
 <id>1</id>
 <firstName>Michael</firstName>
 <lastName>Jackson</lastName>
 <company>Talend</company>
 <city>Paris</city>
 <phone>2323</phone>
 </contact>
 <contact>
 <id>2</id>
 <firstName>Elisa</firstName>
 <lastName>Black</lastName>
 <company>Talend</company>
 <city>Paris</city>
 <phone>4499</phone>
 </contact>
 ...
</contactInfo>
```

To upload an XML file, do the following:

1. Click **Browse...** and browse your directory to the XML file to be uploaded. Alternatively, enter the access path to the file.

The **Schema Viewer** area displays a preview of the XML structure. You can expand and visualize every level of the file's XML tree structure.



2. Enter the **Encoding** type in the corresponding field if the system does not detect it automatically.
3. In the **Limit** field, enter the number of columns on which the XPath query is to be executed, or *0* if you want to run it against all of the columns.
4. Click **Next** to define the schema parameters.

## Uploading an XSD file

This procedure describes how to upload an XSD file to obtain the XML tree structure. To upload an XML file, see [Uploading an XML file](#).

An XSD file is used to define the schema of XML files. The structure and element data types of the example XML file above can be described using the following XSD, which is used as the example XSD input in this section.

```
<xss:schema xmlns:xss="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
 <xss:element name="contactInfo">
 <xss:complexType>
 <xss:sequence>
 <xss:element maxOccurs="unbounded" ref="contact" />
 </xss:sequence>
 </xss:complexType>
 </xss:element>
 <xss:element name="contact">
 <xss:complexType>
 <xss:sequence>
```

```

<xs:element ref="id"/>
<xs:element ref="firstName"/>
<xs:element ref="lastName"/>
<xs:element ref="company"/>
<xs:element ref="city"/>
<xs:element ref="phone"/>
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="id" type="xs:integer" />
<xs:element name="firstName" type="xs:NCName" />
<xs:element name="lastName" type="xs:NCName" />
<xs:element name="company" type="xs:NCName" />
<xs:element name="city" type="xs:NCName" />
<xs:element name="phone" type="xs:integer" />
</xs:schema>

```

For more information on XML Schema, see <http://www.w3.org/XML/Schema>.

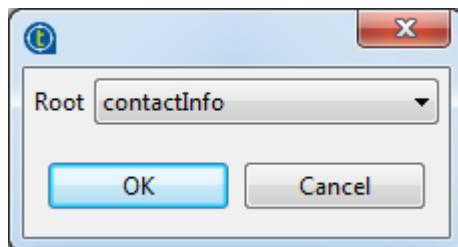


When loading an XSD file,

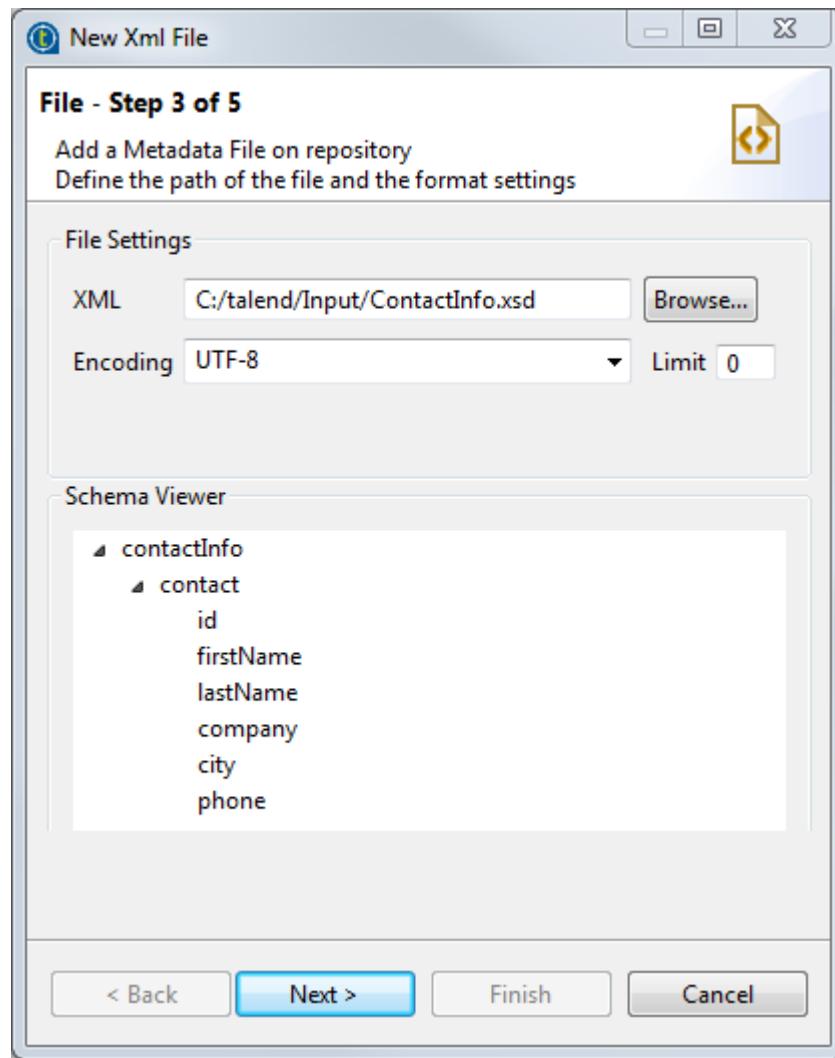
- the data will be saved in the **Repository**, and therefore the metadata will not be affected by the deletion or displacement of the file.
- you can choose an element as the root of your XML tree.

To load an XSD file, do the following:

1. Click **Browse...** and browse your directory to the XSD file to be uploaded. Alternatively, enter the access path to the file.
2. In the dialog box that appears, select an element from the **Root** list as the root of your XML tree, and click **OK**.



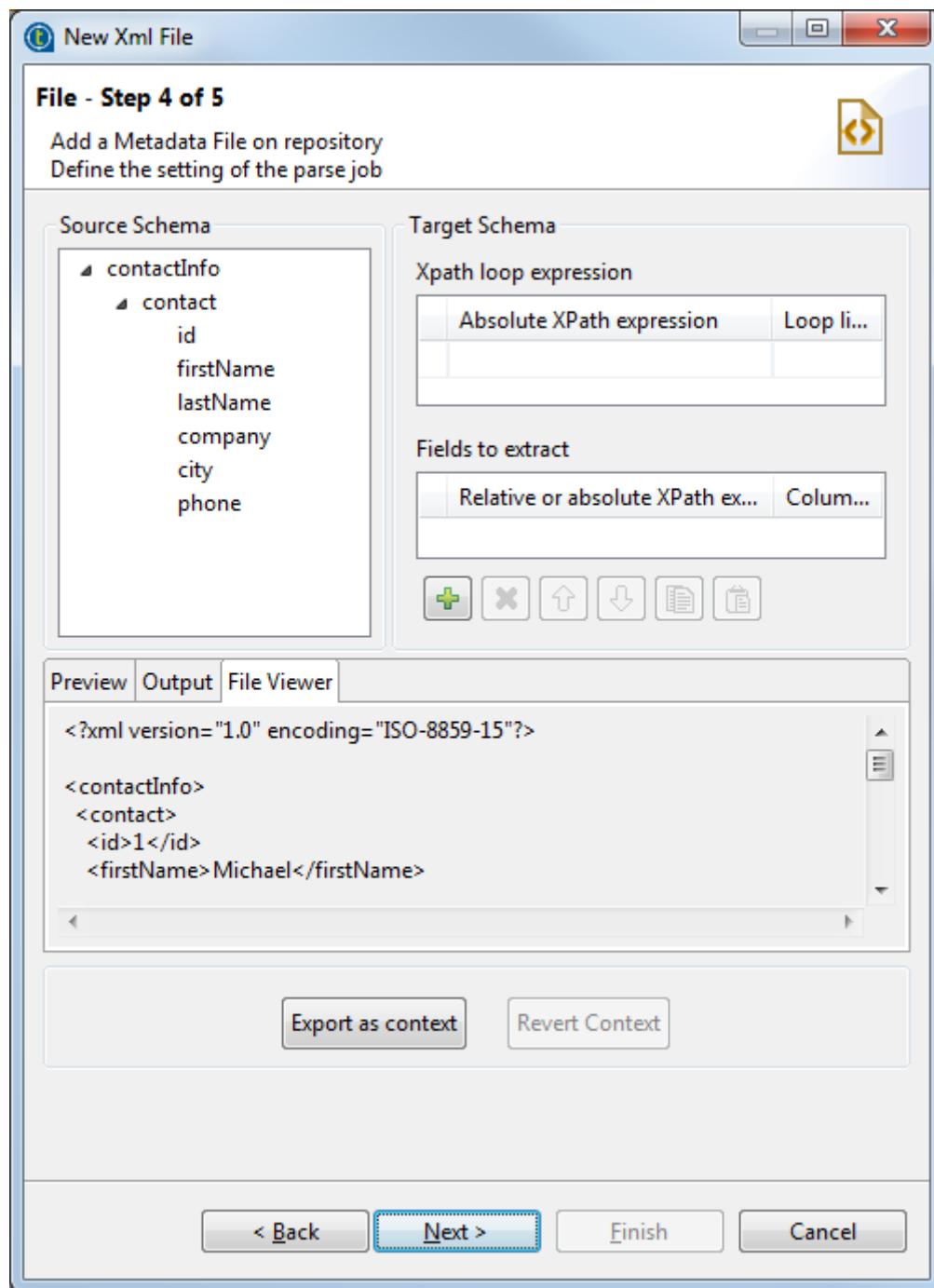
The **Schema Viewer** area displays a preview of the XML structure. You can expand and visualize every level of the file's XML tree structure.



3. Enter the **Encoding** type in the corresponding field if the system does not detect it automatically.
4. In the **Limit** field, enter the number of columns on which the XPath query is to be executed, or *0* if you want to run it against all of the columns.
5. Click **Next** to define the schema parameters.

## Defining the schema

In this step the schema parameters are set.



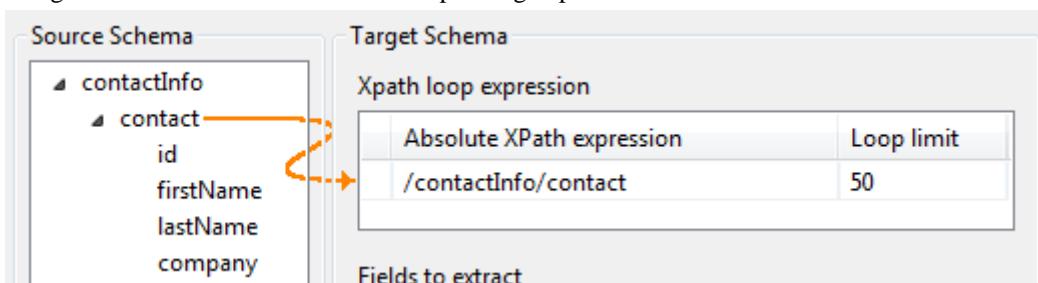
The schema definition window is composed of four views:

View	Description
<b>Source Schema</b>	Tree view of the XML file.
<b>Target Schema</b>	Extraction and iteration information.
<b>Preview</b>	Preview of the target schema, together with the input data of the selected columns displayed in the defined order.  The preview functionality is not available if you loaded an XSD file.
<b>File Viewer</b>	Preview of the brute data.

First define an Xpath loop and the maximum number of times the loop can run. To do so:

1. Populate the **XPath loop expression** field with the absolute XPath expression for the node to be iterated upon. There are two ways to do this, either:
  - enter the absolute XPath expression for the node to be iterated upon (Enter the full expression or press **Ctrl+Space** to use the autocompletion list),
  - drop a node from the tree view under **Source schema** onto the **Absolute XPath expression** field.

An orange arrow links the node to the corresponding expression.

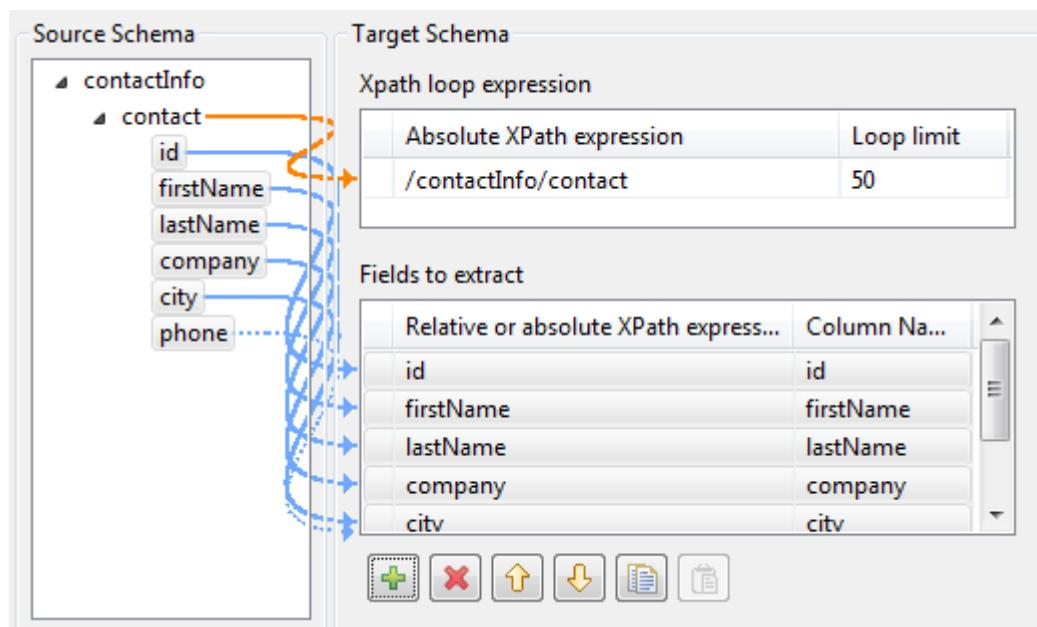


The **Xpath loop expression** field is mandatory.

2. In the **Loop limit** field, specify the maximum number of times the selected node can be iterated, or **-1** if you want to run it against all of the rows.
3. Define the fields to be extracted dragging the node(s) of interest from the **Source Schema** tree into the **Relative or absolute XPath expression** fields.



You can select several nodes to drop on the table by pressing **Ctrl** or **Shift** and clicking the nodes of interest. The arrow linking an individual node selected on the **Source Schema** to the **Fields to extract** table are blue in colour. The other ones are gray.



4. If needed, you can add as many columns to be extracted as necessary, delete columns or change the column order using the toolbar:
  - Add or delete a column using the and buttons.
  - Change the order of the columns using the and buttons.

5. In the **Column name** fields, enter labels for the columns to be displayed in the schema **Preview** area.
6. Click **Refresh Preview** to display a preview of the target schema. The fields are consequently displayed in the schema according to the defined order.



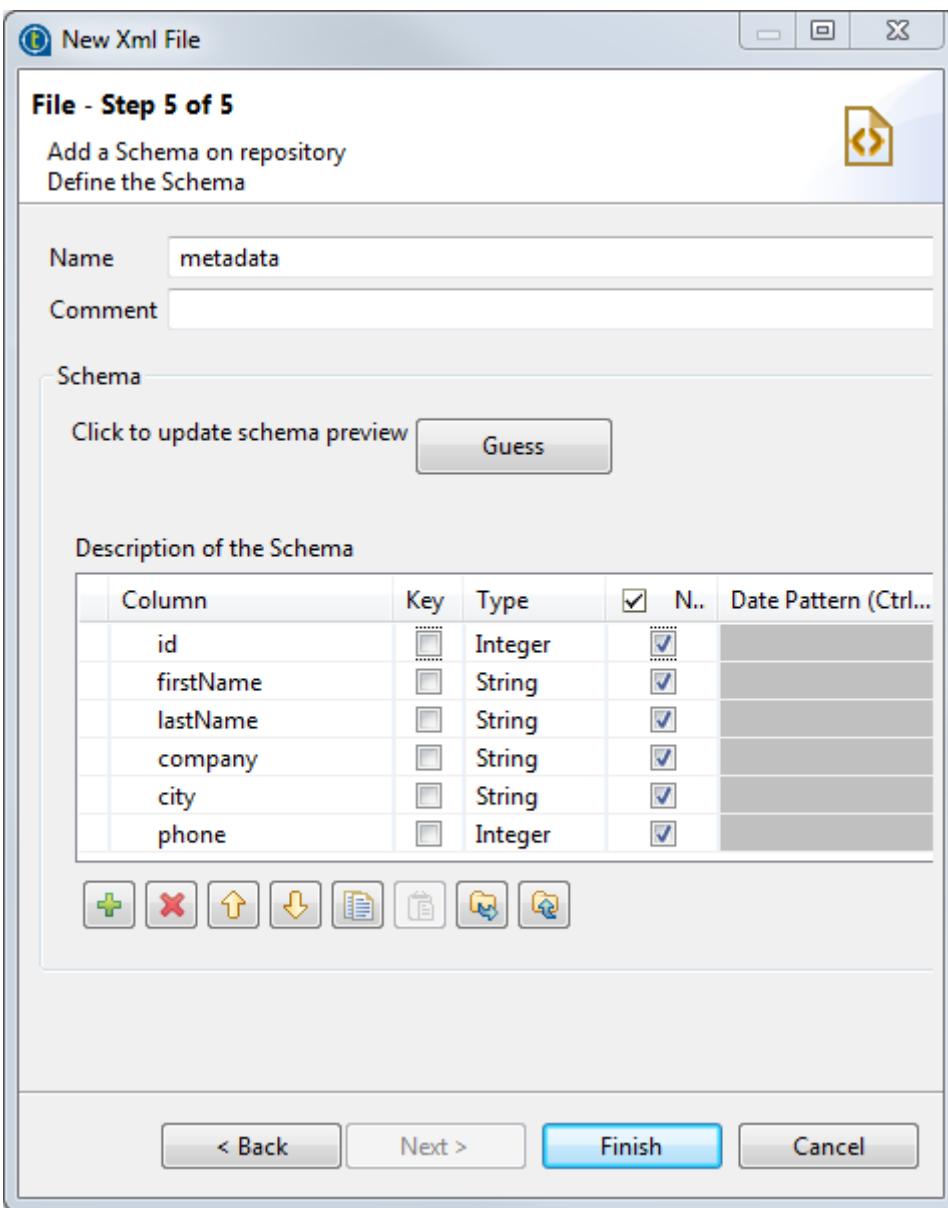
The preview functionality is not available if you loaded an XSD file.

Preview					
Output					
File Viewer					
<b>Refresh Preview</b>					
Preview successful...					
id					
1	Michael	Jackson	Talend	Paris	2323
2	Elisa	Black	Talend	Paris	4499
3	Michael	Dujardin	Talend	Paris	8872
4	Marie	Dolvina	Talend	Paris	6655
5	Jean	Perfide	Talend	Paris	3344
6	Emilie	Taldor	Talend	Paris	2266
7	Anne-Laure	Paldufier	Talend	Paris	4422

7. Click **Next** to check and edit the end schema.

### Finalizing the end schema

The schema generated displays the columns selected from the XML file and allows you to further define the schema.



1. If needed, rename the metadata in the **Name** field (*metadata*, by default), add a **Comment**, and make further modifications, for example:

- Redefine the columns by editing the relevant fields.
- Add or delete a column using the and buttons.
- Change the order of the columns using the and buttons.

Make sure the data type in the **Type** column is correctly defined.

For more information regarding Java data types, including date pattern, see [Java API Specification](#).

Below are the commonly used **Talend** data types:

- Object: a generic **Talend** data type that allows processing data without regard to its content, for example, a data file not otherwise supported can be processed with a **tFileInputRaw** component by specifying that it has a data type of Object.

- List: a space-separated list of primitive type elements in an XML Schema definition, defined using the xsd:list element.
  - Document: a data type that allows processing an entire XML document without regarding to its content.
2. If the XML file which the schema is based on has been changed, click the **Guess** button to generate the schema again. Note that if you have customized the schema, the **Guess** feature does not retain these changes.
  3. Click **Finish**. The new file connection, along with its schema, appears under the **File XML** node in the **Repository** tree view.

Now you can drag and drop the file connection or any schema of it from the **Repository** tree view onto the design workspace as a new **tFileInputXML** or **tExtractXMLField** component or onto an existing component to reuse the metadata. For further information about how to use the centralized metadata in a Job, see [How to use centralized metadata in a Job](#) and [How to set a repository schema](#).

To modify an existing file connection, right-click it from the **Repository** tree view, and select **Edit file xml** to open the file metadata setup wizard.

To add a new schema to an existing file connection, right-click the connection from the **Repository** tree view and select **Retrieve Schema** from the contextual menu.

To edit an existing file schema, right-click the schema from the **Repository** tree view and select **Edit Schema** from the contextual menu.

## 7.8.2. Setting up XML metadata for an output file

This section describes how to define a file connection and upload an XML schema for an output file. To define and upload an XML schema for an input file, see [Setting up XML metadata for an input file](#).

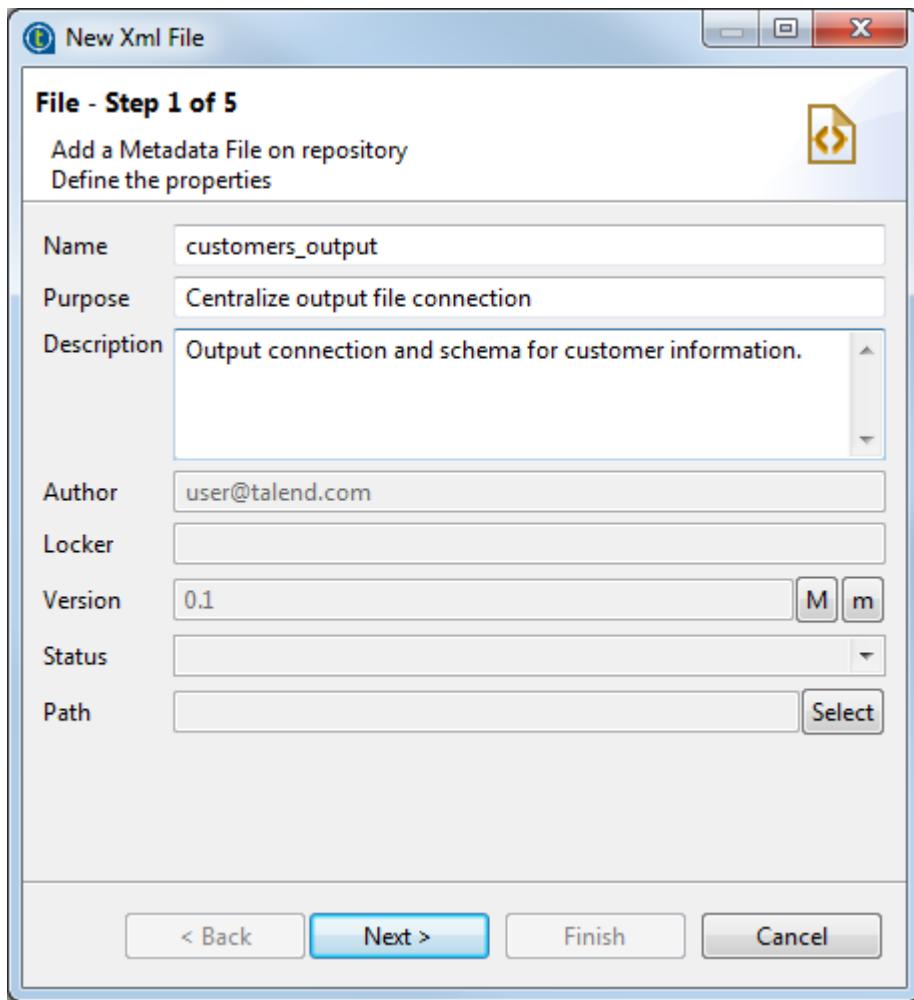
### Defining the general properties

In this step, the general metadata properties such as the **Name**, **Purpose** and **Description** are set.

1. In the file metadata setup wizard, fill in the **Name** field, which is mandatory, and the **Purpose** and **Description** fields if you choose to do so. The information you provide in the **Description** field will appear as a tooltip when you move your mouse pointer over the file connection.



When you enter the general properties of the metadata to be created, you need to define the type of connection as either input or output. It is therefore advisable to enter information that will help you distinguish between your input and output schemas.

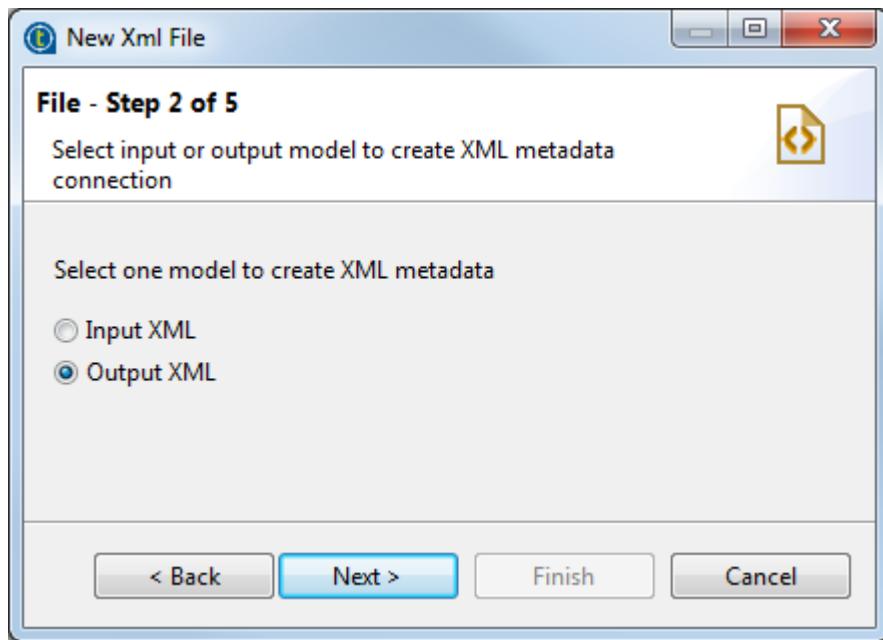


2. If needed, set the version and status in the **Version** and **Status** fields respectively. You can also manage the version and status of a repository item in the **[Project Settings]** dialog box. For more information, see [Version management](#) and [Status management](#) respectively.
3. If needed, click the **Select** button next to the **Path** field to select a folder under the **File XML** node to hold your newly created file connection. Note that you cannot select a folder if you are editing an existing connection, but you can drag and drop it to a new folder whenever you want.
4. Click **Next** to select the type of metadata.

### Setting the type of metadata (output)

In this step, the type of metadata is set as either input or output. For this procedure, the metadata of interest is output.

1. From the dialog box, select **Output XML**.



2. Click **Next** to define the output file, either from an XML or XSD file or from scratch.

### Defining the output file structure using an existing XML file

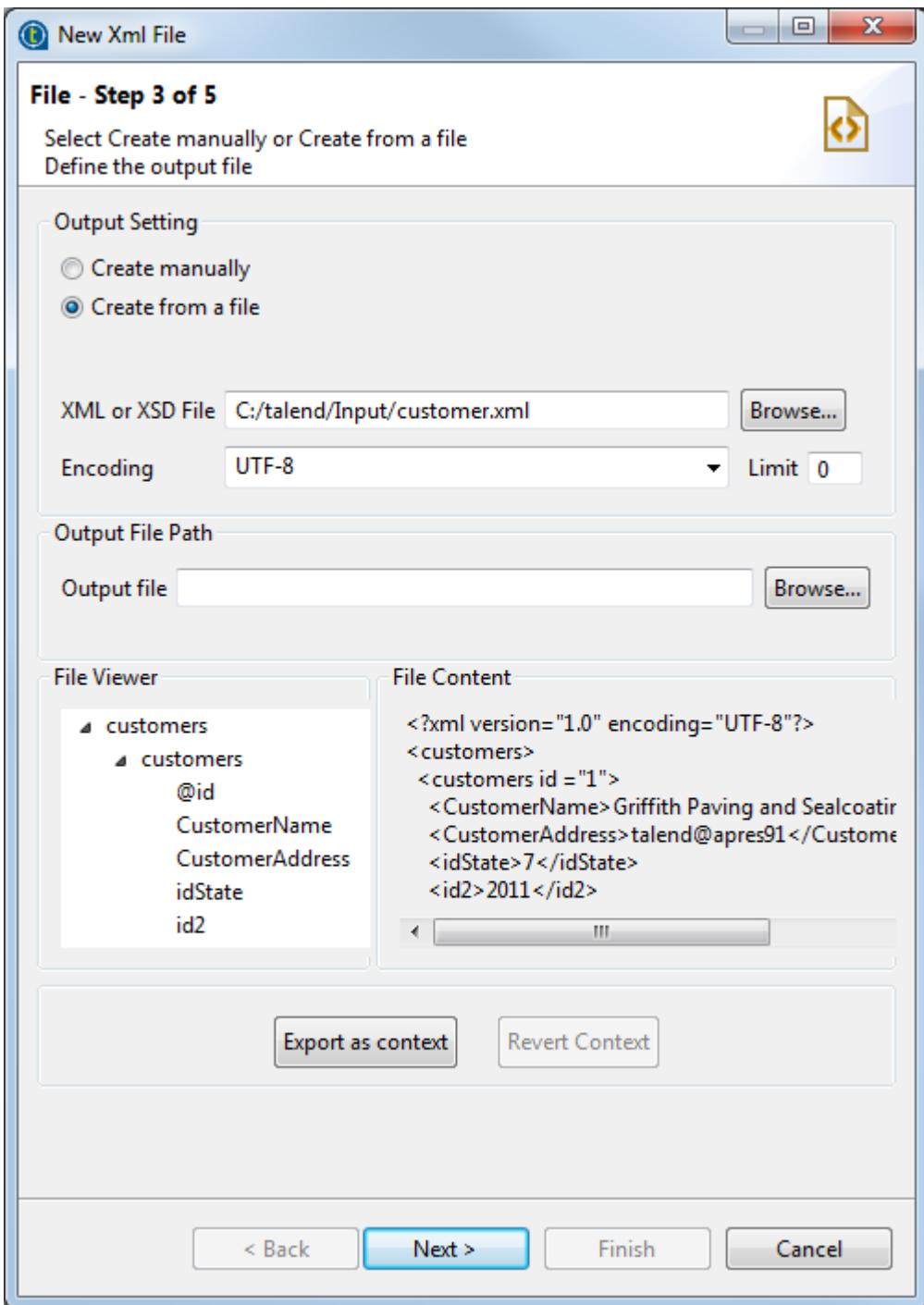
In this step, you will choose whether to create your file manually or from an existing XML or XSD file. If you choose the **Create manually** option you will have to configure your schema, source and target columns yourself at step 4 in the wizard. The file will be created in a Job using a an XML output component such as **tAdvancedFileOutputXML**.

In this procedure, we will create the output file structure by loading an existing XML. To create the output XML structure from an XSD file, see [Defining the output file structure using an XSD file](#).

To create the output XML structure from an XML file, do the following:

1. Select the **Create from a file** option.
2. Click the **Browse...** button next to the **XML or XSD File** field, browse to the access path to the XML file the structure of which is to be applied to the output file, and double-click the file.

The **File Viewer** area displays a preview of the XML structure, and the **File Content** area displays a maximum of the first 50 rows of the file.



3. Enter the **Encoding** type in the corresponding field if the system does not detect it automatically.
4. In the **Limit** field, enter the number of columns on which the XPath query is to be executed, or enter *0* if you want it to be run against all of the columns.
5. In the **Output File** field, in the **Output File Path** zone, browse to or enter the path to the output file. If the file does not exist as yet, it will be created during the execution of a Job using a **tAdvancedFileOutputXML** component. If the file already exists, it will be overwritten.
6. Click **Next** to define the schema.

## Defining the output file structure using an XSD file

This procedure describes how to define the output XML file structure from an XSD file. To define the XML structure from an XML file, see [Defining the output file structure using an existing XML file](#).



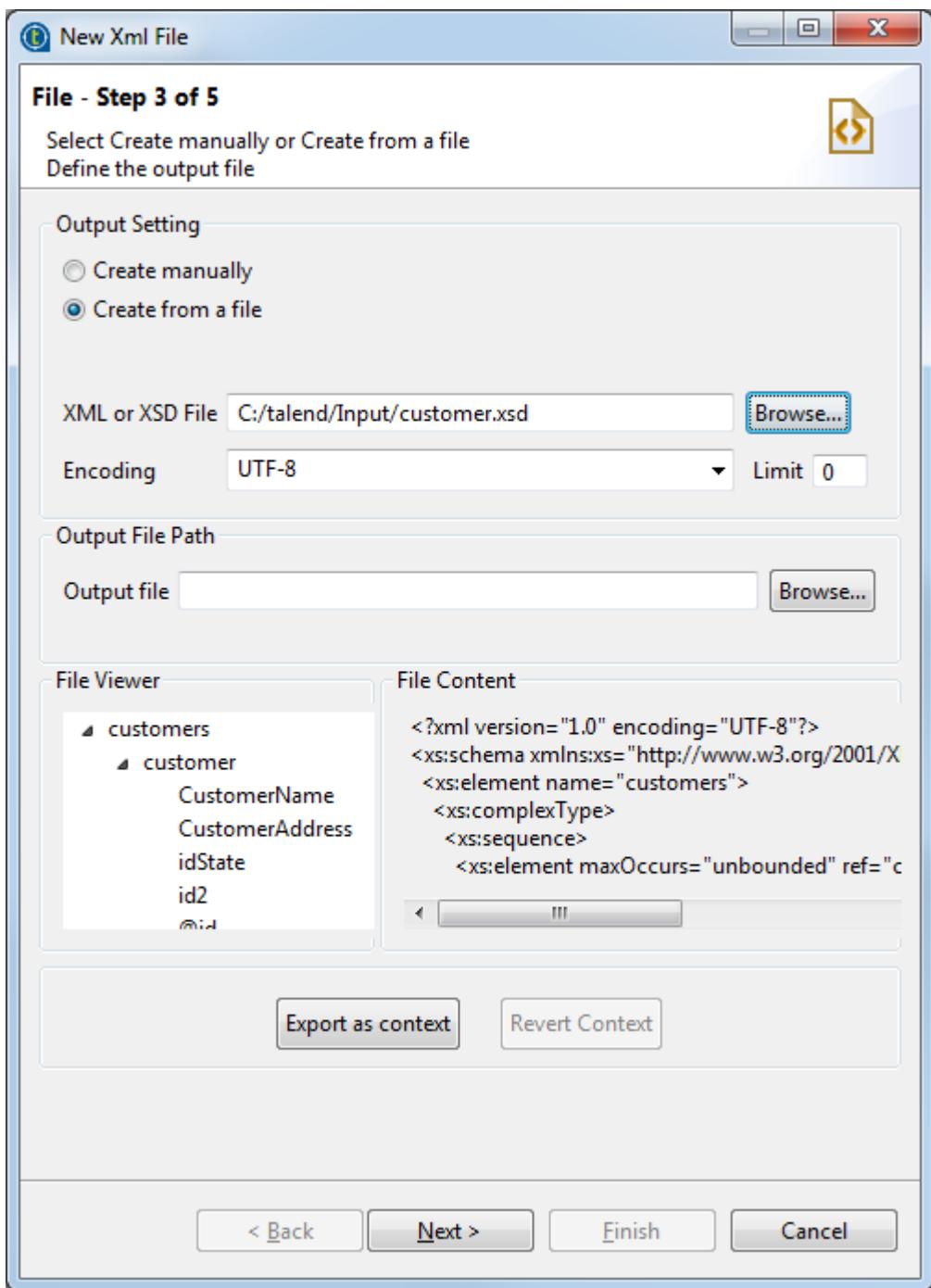
When loading an XSD file,

- the data will be saved in the **Repository**, and therefore the metadata will not be affected by the deletion or displacement of the file.
- you can choose an element as the root of your XML tree.

To create the output XML structure from an XSD file, do the following:

1. Select the **Create from a file** option.
2. Click the **Browse...** button next to the **XML or XSD File** field, browse to the access path to the XSD file the structure of which is to be applied to the output file, and double-click the file.
3. In the dialog box that appears, select an element from the **Root** list as the root of your XML tree, and click **OK**.

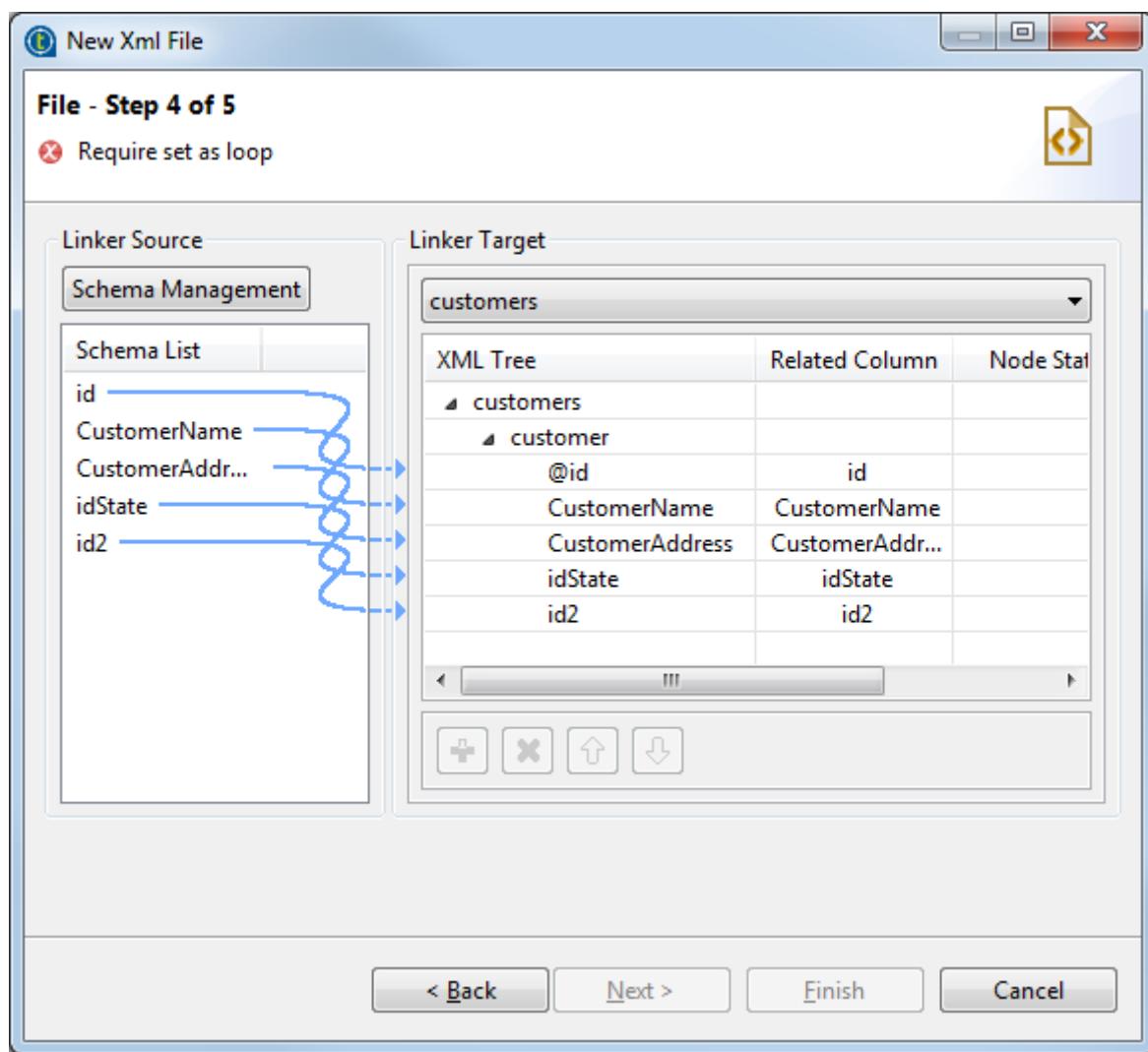
The **File Viewer** area displays a preview of the XML structure, and the **File Content** area displays a maximum of the first 50 rows of the file.



4. Enter the **Encoding** type in the corresponding field if the system does not detect it automatically.
5. In the **Limit** field, enter the number of columns on which the XPath query is to be executed, or enter *0* if you want it to be run against all of the columns.
6. In the **Output File** field, in the **Output File Path** zone, browse to or enter the path to the output file. If the file does not exist as yet, it will be created during the execution of a Job using a **tAdvancedFileOutputXML** component. If the file already exists, it will be overwritten.
7. Click **Next** to define the schema.

## Defining the schema

Upon completion of the previous operations, the columns in the **Linker Source** area are automatically mapped to the corresponding ones in the **Linker Target** area, as indicated by blue arrow links.



In this step, you need to define the output schema. The following table describes how:

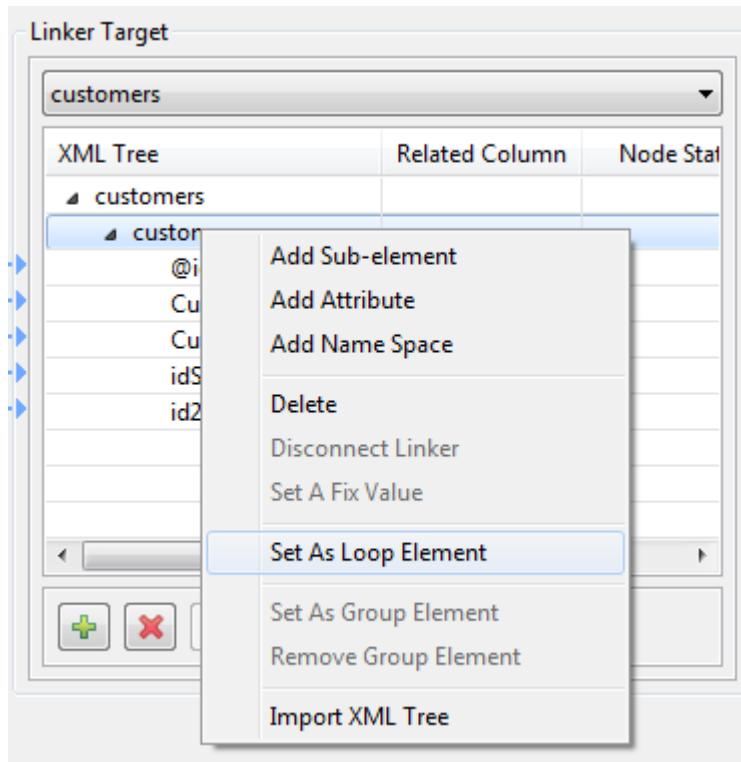
To...	Perform the following...
Create a schema from scratch or edit the source schema columns to pass to the output schema	In the <b>Linker Source</b> area, click the <b>Schema Management</b> button to open the schema editor.
Define a loop element	In the <b>Linker Target</b> area, right-click the element of interest and select <b>Set As Loop Element</b> from the contextual menu.  It is a mandatory operation to define an element to run a loop on.
Define a group element	In the <b>Linker Target</b> area, right-click the element of interest and select <b>Set As Group Element</b> from the contextual menu.  You can set a parent element of the loop element as a group element on the condition that the parent element is not the root of the XML tree.
Create a child element for an element	In the <b>Linker Target</b> area, <ul style="list-style-type: none"> <li>• Right-click the element of interest and select <b>Add Sub-element</b> from the contextual menu, enter a name for the sub-element in the dialog box that appears, and click <b>OK</b>,</li> <li>• Select the element of interest, click the <b>[+]</b> button at the bottom, select <b>Create as sub-element</b> in the dialog box that appears, and click <b>OK</b>. Then, enter a name for the sub-element in the next dialog box and click <b>OK</b>.</li> </ul>
Create an attribute for an element	In the <b>Linker Target</b> area,

To...	Perform the following...
	<ul style="list-style-type: none"> <li>Right-click the element of interest and select <b>Add Attribute</b> from the contextual menu, enter a name for the attribute in the dialog box that appears, and click <b>OK</b>,</li> <li>Select the element of interest, click the [+] button at the bottom, select <b>Create as attribute</b> in the dialog box that appears, and click <b>OK</b>. Then, enter a name for the attribute in the next dialog box and click <b>OK</b>.</li> </ul>
Create a name space for an element	<p>In the <b>Linker Target</b> area,</p> <ul style="list-style-type: none"> <li>Right-click the element of interest and select <b>Add Name Space</b> from the contextual menu, enter a name for the name space in the dialog box that appears, and click <b>OK</b>,</li> <li>Select the element of interest, click the [+] button at the bottom, select <b>Create as name space</b> in the dialog box that appears, and click <b>OK</b>. Then, enter a name for the name space in the next dialog box and click <b>OK</b>.</li> </ul>
Delete one or more elements/attributes/name spaces	<p>In the <b>Linker Target</b> area,</p> <ul style="list-style-type: none"> <li>Right-click the element(s)/attribute(s)/name space(s) of interest and select <b>Delete</b> from the contextual menu</li> <li>Select the element(s)/attribute(s)/name space(s) of interest and click the [x] button at the bottom</li> <li>Select the element(s)/attribute(s)/name space(s) of interest and press the <b>Delete</b> key.</li> </ul> <p> Deleting an element will also delete its children, if any.</p>
Adjust the order of one or more elements	<p>In the <b>Linker Target</b> area, select the element(s) of interest and click the  and  buttons.</p>
Set a static value for an element/attribute/name space	<p>In the <b>Linker Target</b> area, right-click the element/attribute/name space of interest and select <b>Set A Fix Value</b> from the contextual menu.</p> <p></p> <ul style="list-style-type: none"> <li>The value you set will replace any value retrieved for the corresponding column from the incoming data flow in your Job.</li> <li>You can set a static value for a child element of the loop element only, on the condition that the element does not have its own children and does not have a source-target mapping on it.</li> </ul>
Create a source-target mapping	<p>Select the column of interest in the <b>Linker Source</b> area, drop it onto the node of interest in the <b>Linker Target</b> area, and select <b>Create as sub-element of target node</b>, <b>Create as attribute of target node</b>, or <b>Add linker to target node</b> according to your need in the dialog box that appears, and click <b>OK</b>.</p> <p>If you choose an option that is not permitted for the target node, you will see a warning message and your operation will fail.</p>
Remove a source-target mapping	<p>In the <b>Linker Target</b> area, right-click the node of interest and select <b>Disconnect Linker</b> from the contextual menu.</p>
Create an XML tree from another XML or XSD file	<p>Right-click any schema item in the <b>Linker Target</b> area and select <b>Import XML Tree</b> from the contextual menu to load another XML or XSD file. Then, you need to create source-target mappings manually and define the output schema all again.</p>



You can select and drop several fields at a time, using the **Ctrl + Shift** technique to make multiple selections, therefore making mapping faster. You can also make multiple selections for right-click operations.

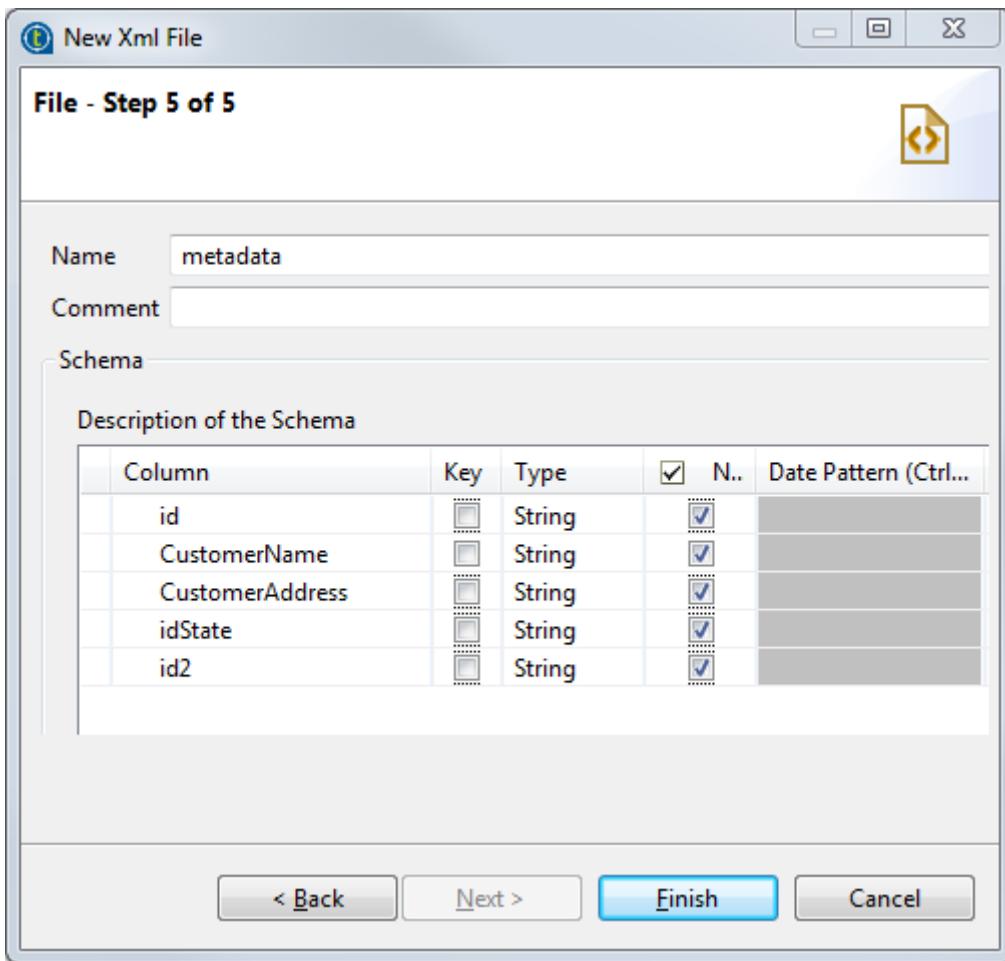
1. In the **Linker Target** area, right-click the element you want to run a loop on and select **Set As Loop Element** from the contextual menu.



2. Define other output file properties as needed, and then click **Next** to view and customize the end schema.

### Finalizing the end schema

Step 5 of the wizard displays the end schema generated and allows you to further define the schema.



- If needed, rename the metadata in the **Name** field (*metadata*, by default), add a **Comment**, and make further modifications, for example:
  - Redefine the columns by editing the relevant fields.
  - Add or delete a column using the **[+]** and **[x]** buttons.
  - Change the order of the columns using the **↑** and **↓** buttons.
- If the XML file which the schema is based on has been changed, click the **Guess** button to generate the schema again. Note that if you have customized the schema, the **Guess** feature does not retain these changes.
- Click **Finish**. The new file connection, along with its schema, is displayed under the relevant **File XML** metadata node in the **Repository** tree view.

Now you can drag and drop the file connection or any schema of it from the **Repository** tree view onto the design workspace as a new **tAdvancedFileOutputXML** component or onto an existing component to reuse the metadata.

To modify an existing file connection, right-click it from the **Repository** tree view, and select **Edit file xml** to open the file metadata setup wizard.

To add a new schema to an existing file connection, right-click the connection from the **Repository** tree view and select **Retrieve Schema** from the contextual menu.

To edit an existing file schema, right-click the schema from the **Repository** tree view and select **Edit Schema** from the contextual menu.

## 7.9. Centralizing File Excel metadata

If you often need to read data from and/or write data to a certain Excel spreadsheet file, you may want to centralize the connection to the file, along with its data structure, in the **Repository** for easy reuse. This will save you much effort because you will not have to define the metadata details manually in the relevant components each time you use the file.

You can centralize an Excel file connection either from an existing Excel file, or from Excel file property settings defined in a Job.

To centralize a File Excel connection and its schema from an Excel file, expand **Metadata** in the **Repository** tree view, right-click **File Excel** and select **Create file Excel** from the contextual menu to open the file metadata setup wizard.

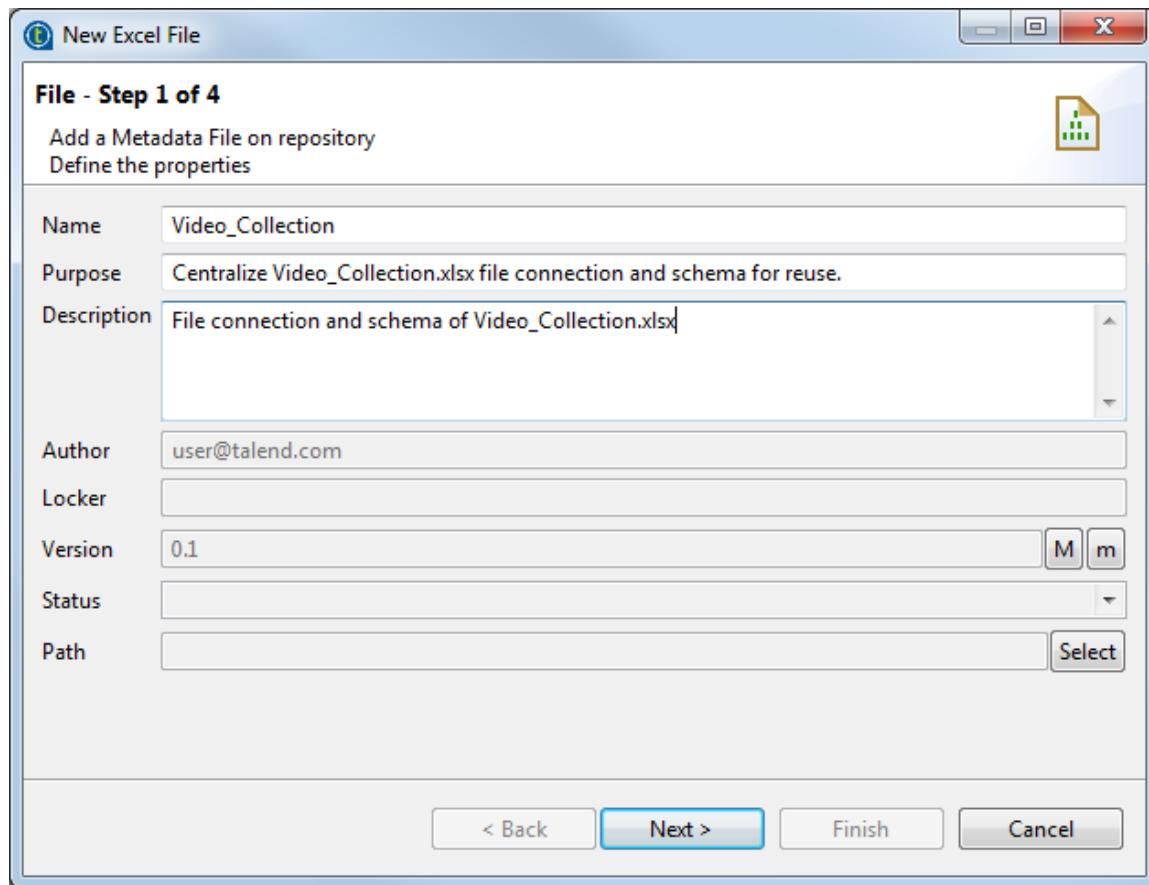
To centralize a file connection and its schema you have already defined in a Job, click the  icon in the **Basic settings** view of the relevant component, with its **Property Type** set to **Built-in**, to open the file metadata setup wizard.

Then complete these tasks step by step following the wizard:

- Define the general information that will identify the file connection. See [Defining the general properties](#).
- Load the file of interest. See [Loading the file](#).
- Parse the file to retrieve the file schema. See [Parsing the file](#).
- Finalize the file schema. See [Finalizing the end schema](#).

### Defining the general properties

1. In the file metadata setup wizard, fill in the **Name** field, which is mandatory, and the **Purpose** and **Description** fields if needed. The information you provide in the **Description** field will appear as a tooltip when you move your mouse pointer over the file connection.

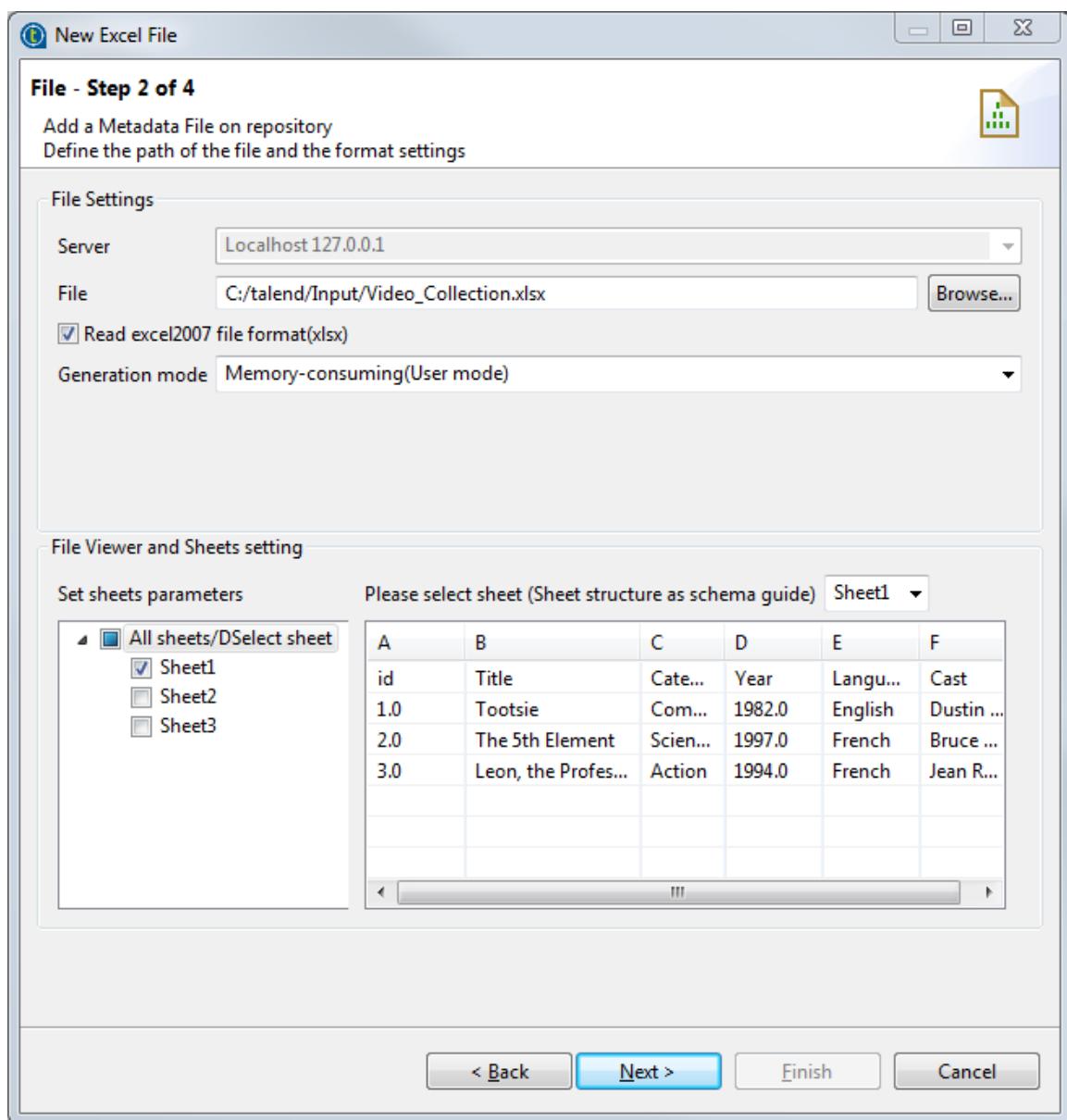


2. If needed, set the version and status in the **Version** and **Status** fields respectively. You can also manage the version and status of a repository item in the **[Project Settings]** dialog box. For more information, see [Version management](#) and [Status management](#) respectively.
3. If needed, click the **Select** button next to the **Path** field to select a folder under the **File Excel** node to hold your newly created file connection.
4. Click **Next** to proceed with file settings.

## Loading the file

1. Click the **Browse...** button to browse to the file and fill out the **File** field.

Skip this step if you are saving an Excel file connection defined in a component because the file path is already filled in the **File** field.



2. If the uploaded file is an Excel 2007 file, make sure that the **Read excel2007 file format(xlsx)** check box is selected.
3. By default, user mode is selected. If the uploaded xlsx file is extremely large, select **Less memory consumed for large excel(Event mode)** from the **Generation mode** list to prevent out-of-memory errors.
4. In the **File viewer and sheets setting** area, view the file content and the select the sheet or sheets of interest.
  - From the **Please select sheet** drop-down list, select the sheet you want to view. The preview table displays the content of the selected sheet.

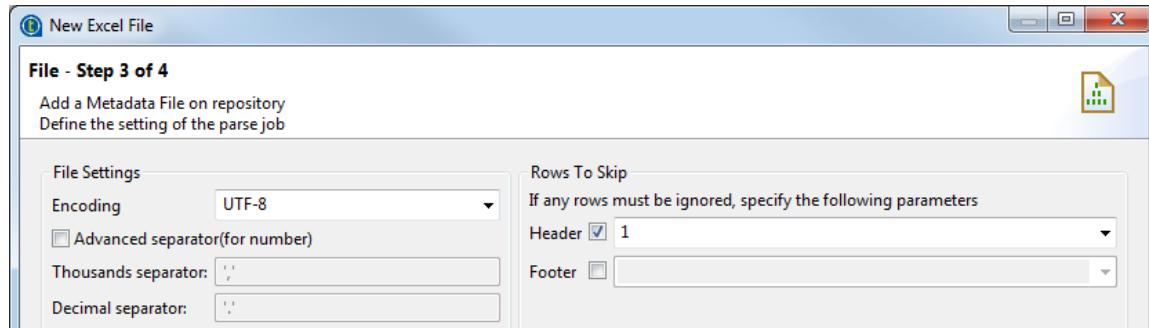
By default the file preview table displays the first sheet of the file.

  - From the **Set sheets parameters** list, select the check box next to the sheet or sheets you want to upload. If you select more than one sheet, the result schema will be the combination of the structures of all the selected sheets.
5. Click **Next** to continue.

## Parsing the file

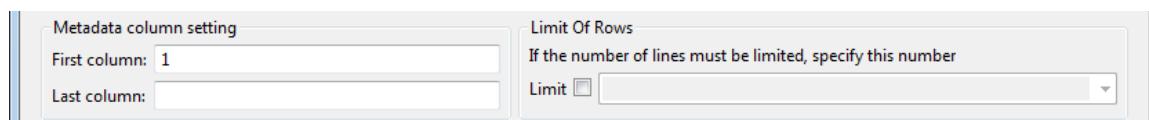
In this step of the wizard, you can define the various settings of your file so that the file schema can be properly retrieved.

- Specify the encoding, advanced separator for numbers, and the rows that should be skipped as they are header or footer, according to your Excel file.



- If needed, fill the **First column** and **Last column** fields with integers to set precisely the columns to be read in the file. For example, if you want to skip the first column as it may not contain proper data to be processed, fill the **First column** field with 2 to set the second column of the file as the first column of the schema.

To retrieve the schema of an Excel file you do not need to parse all the rows of the file, especially when you have uploaded a large file. To limit the number of rows to parse, select the **Limit** check box in the **Limit Of Rows** area and set or select the desired number of rows.



- If your Excel file has a header row, select the **Set heading row as column names** check box to take into account the heading names. Click **Refresh** to view the result of all the previous changes in the preview table.

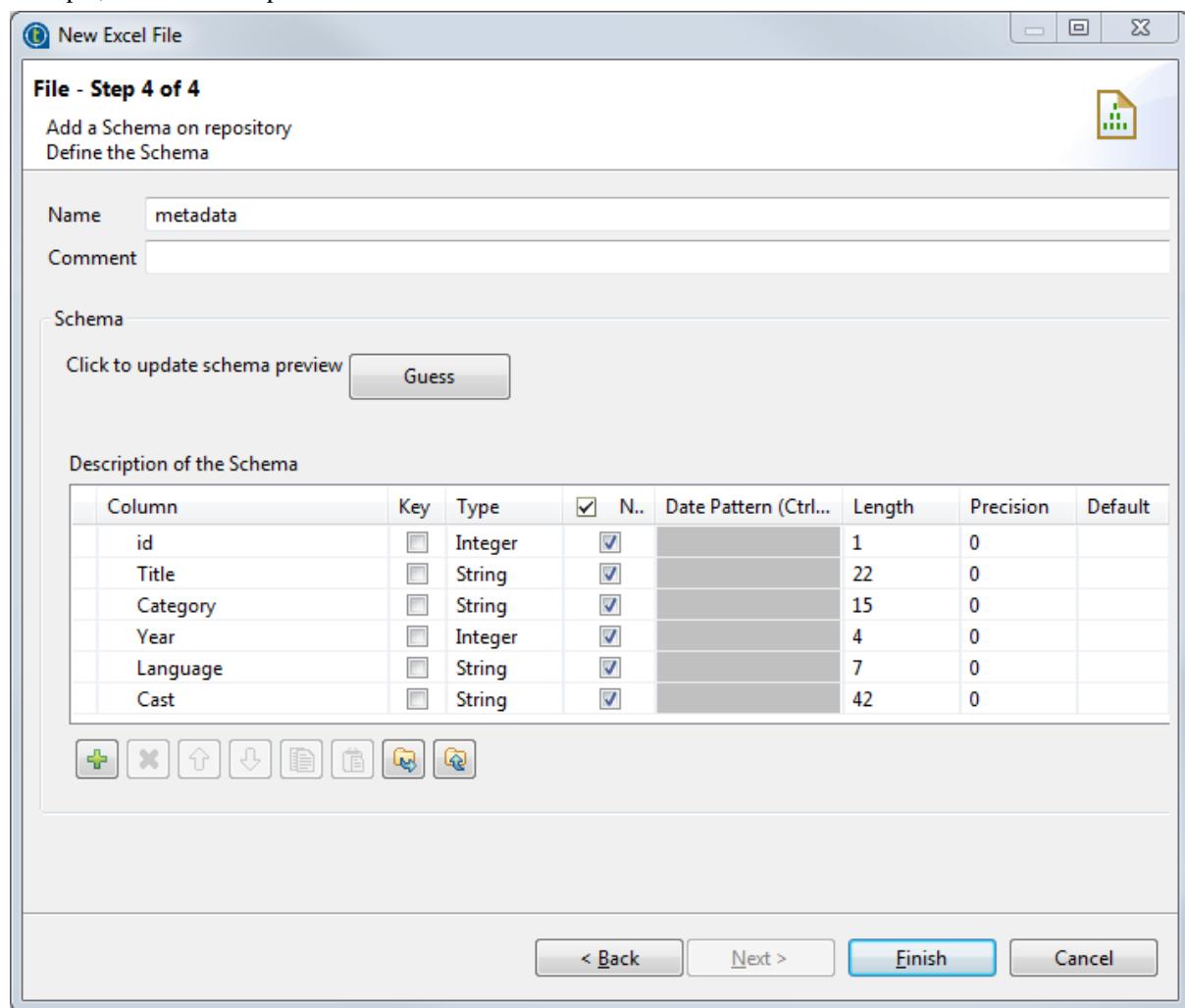
<b>id</b>	<b>Title</b>	<b>Category</b>	<b>Year</b>	<b>Language</b>	<b>Cast</b>
1	Tootsie	Comedy	1982	English	Dustin Hoffman, Jessica Lange, Sydney
2	The 5th Element	Science fiction	1997	French	Bruce Willis, Gary Oldman, Milla Jovovitch
3	Leon, the Professor	Action drama	1994	French	Jean Reno, Gary Oldman, Nathalie Portman

- Then click **Next** to continue.

## Finalizing the end schema

The last step of the wizard shows the end schema generated and allows you to customize the schema according to your needs.

Note that any character which could be misinterpreted by the program is replaced by neutral characters. For example, asterisks are replaced with underscores.



1. If needed, rename the schema (by default, *metadata*) and leave a comment.

Customize the schema if needed: add, remove or move schema columns, export the schema to an XML file, or replace the schema by importing an schema definition XML file using the tool bar.

Make sure the data type in the **Type** column is correctly defined.

For more information regarding Java data types, including date pattern, see [Java API Specification](#).

Below are the commonly used **Talend** data types:

- Object: a generic **Talend** data type that allows processing data without regard to its content, for example, a data file not otherwise supported can be processed with a **tFileInputRaw** component by specifying that it has a data type of Object.
- List: a space-separated list of primitive type elements in an XML Schema definition, defined using the xsd:list element.
- Document: a data type that allows processing an entire XML document without regarding to its content.

2. If the Excel file which the schema is based on has been changed, click the **Guess** button to generate the schema again. Note that if you have customized the schema, the **Guess** feature does not retain these changes.
3. Click **Finish**. The new schema is displayed under the relevant **File Excel** connection node in the **Repository** tree view.

Now you can drag and drop the file connection or the schema of it from the **Repository** tree view onto the design workspace as a new component or onto an existing component to reuse the metadata. For further information about how to use the centralized metadata in a Job, see [How to use centralized metadata in a Job](#) and [How to set a repository schema](#).

To modify an existing file connection, right-click it from the **Repository** tree view, and select **Edit file Excel** to open the file metadata setup wizard.

To add a new schema to an existing file connection, right-click the connection from the **Repository** tree view and select **Retrieve Schema** from the contextual menu.

To edit an existing file schema, right-click the schema from the **Repository** tree view and select **Edit Schema** from the contextual menu.

## 7.10. Centralizing File LDIF metadata

LDIF files are directory files described by attributes. If you often need to read certain LDIF files, you may want to centralize the connections to these LDIF-type files and their attribute descriptions in the **Repository** for easy reuse. This way you will not have to define the metadata details manually in the relevant components each time you use the files.

You can centralize an LDIF file connection either from an existing LDIF file, or from the LDIF file property settings defined in a Job.

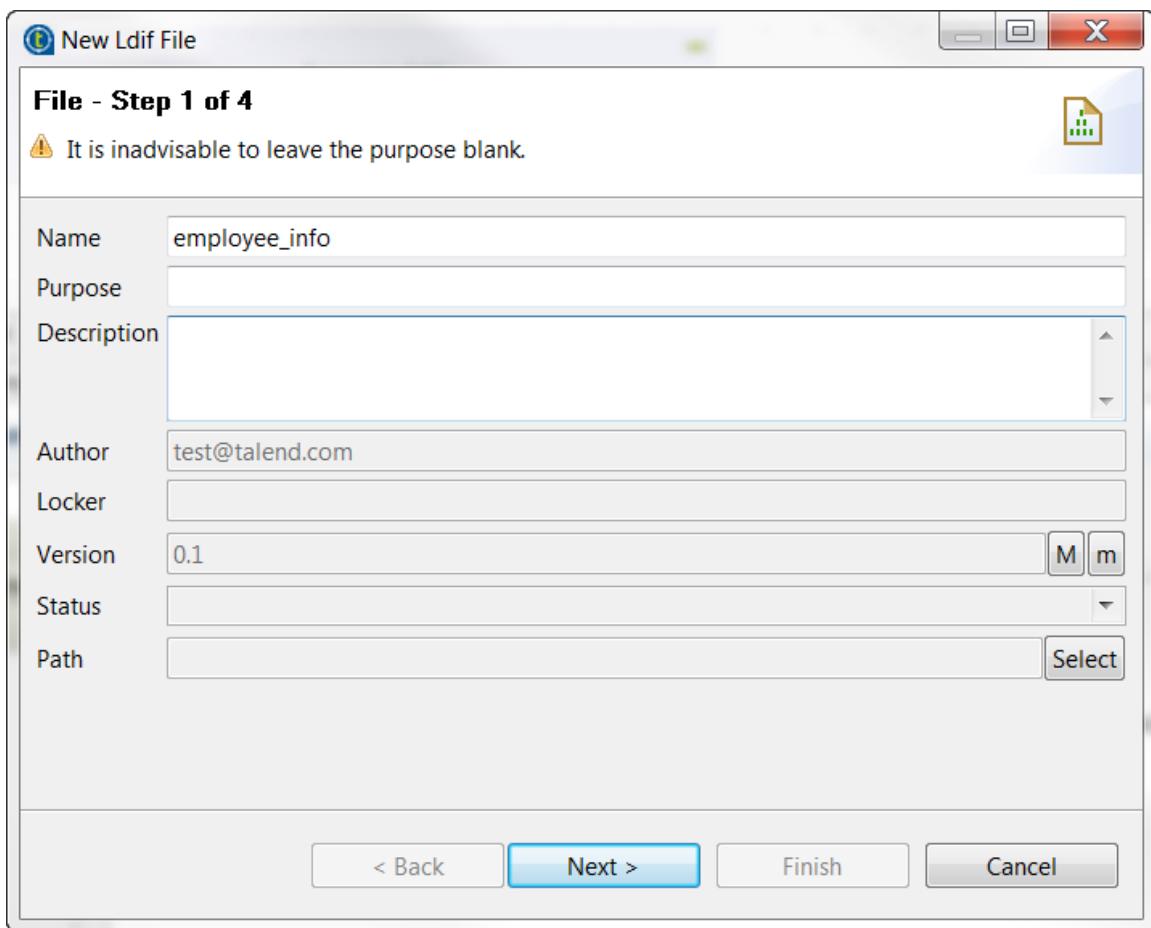
To centralize an LDIF connection and its schema from an LDIF file, expand **Metadata** in the **Repository** tree view, right-click **File ldif** and select **Create file ldif** from the contextual menu to open the file metadata setup wizard.

To centralize a file connection and its schema you have already defined in a Job, click the  icon in the **Basic settings** view of the relevant component, with its **Property Type** set to **Built-in**, to open the file metadata setup wizard.

Then complete these steps following the wizard:

-  Make sure that you have installed the required third-party module as described in the *Talend Installation and Upgrade Guide*.
1. Fill in the general information in the relevant fields to identify the LDIF file metadata, including **Name**, **Purpose** and **Description**.

The **Name** field is required, and the information you provide in the **Description** field will appear as a tooltip when you move your mouse pointer over the file connection.

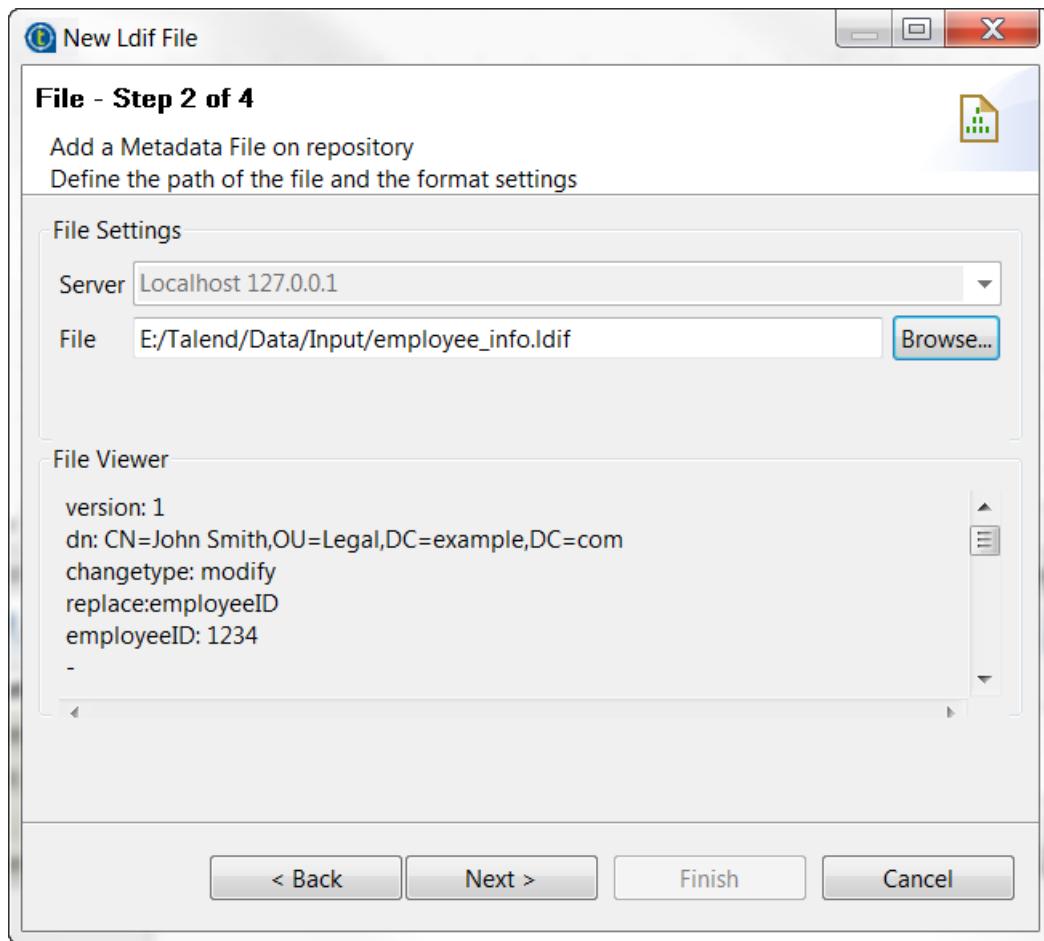


2. If needed, set the version and status in the **Version** and **Status** fields respectively. You can also manage the version and status of a repository item in the **[Project Settings]** dialog box. For more information, see [Version management](#) and [Status management](#) respectively.
3. If needed, click the **Select** button next to the **Path** field to select a folder under the **File Ldif** node to hold your newly created file connection.

Click **Next** to proceed with file settings.

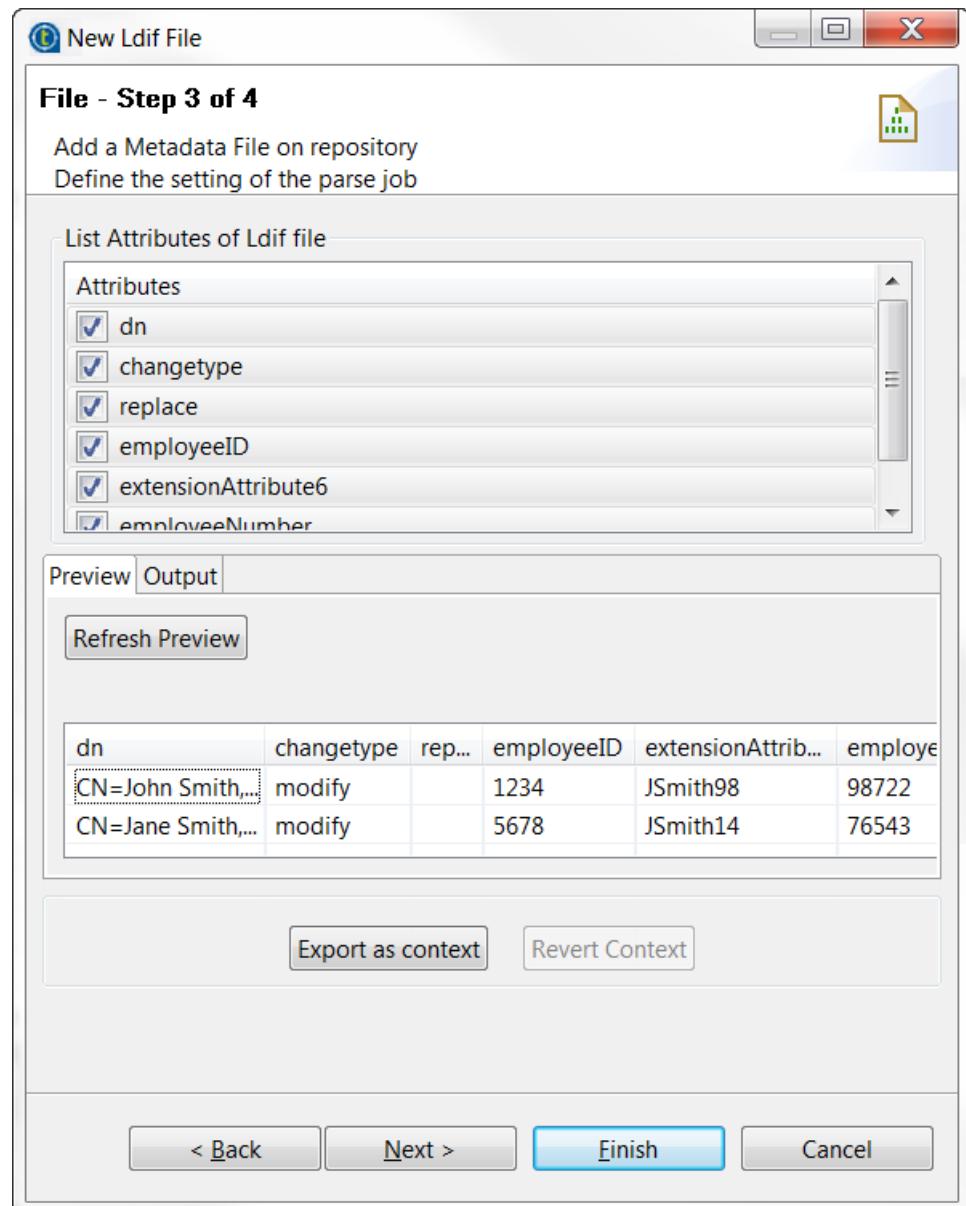
4. Click the **Browse...** button to browse to the file and fill out the **File** field.

Skip this step if you are saving an LDIF file connection defined in a component because the file path is already filled in the **File** field.



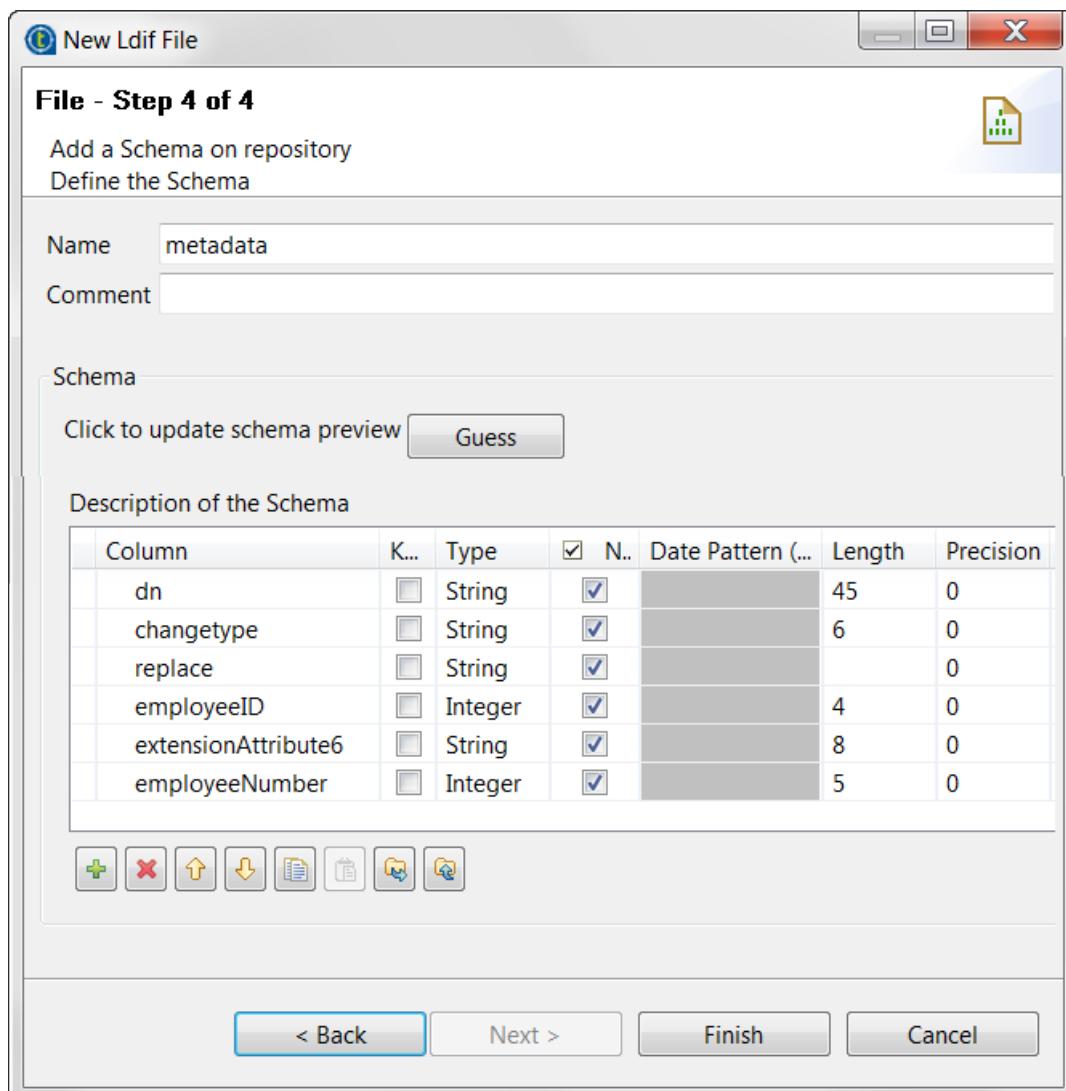
5. Check the first 50 rows of the file in the **File Viewer** area and click **Next** to continue.
6. From the list of attributes of the loaded file, select the attributes you want to include the file schema, and click **Refresh Preview** to preview the selected attributes.

Then click **Next** to proceed with schema finalization.



7. If needed, customize the generated schema:

- Rename the schema (by default, *metadata*) and leave a comment.
- Add, remove or move schema columns, export the schema to an XML file, or replace the schema by importing an schema definition XML file using the tool bar.



Make sure the data type in the **Type** column is correctly defined.

For more information regarding Java data types, including date pattern, see [Java API Specification](#).

Below are the commonly used **Talend** data types:

- Object: a generic **Talend** data type that allows processing data without regard to its content, for example, a data file not otherwise supported can be processed with a **tFileInputRaw** component by specifying that it has a data type of Object.
  - List: a space-separated list of primitive type elements in an XML Schema definition, defined using the xsd:list element.
  - Document: a data type that allows processing an entire XML document without regarding to its content.
8. If the LDIF file on which the schema is based has been changed, click the **Guess** button to generate the schema again. Note that if you have customized the schema, the **Guess** feature does not retain these changes.
  9. Click **Finish**. The new schema is displayed under the relevant Ldif file connection node in the **Repository** tree view.

Now you can drag and drop the file connection or the schema of it from the **Repository** tree view onto the design workspace as a new component or onto an existing component to reuse the metadata. For further information

about how to use the centralized metadata in a Job, see [How to use centralized metadata in a Job](#) and [How to set a repository schema](#).

To modify an existing file connection, right-click it from the **Repository** tree view, and select **Edit file Idif** to open the file metadata setup wizard.

To add a new schema to an existing file connection, right-click the connection from the **Repository** tree view and select **Retrieve Schema** from the contextual menu.

To edit an existing file schema, right-click the schema from the **Repository** tree view and select **Edit Schema** from the contextual menu.

## 7.11. Centralizing JSON file metadata

If you often need to use a JSON file, you may want to use the **[New Json File]** wizard to centralize the file connection, XPath query statements, and data structure in the **Repository** for easy reuse.

Depending on the option you select, the wizard helps you create either an input or an output file connections. In a Job, the **tFileInputJSON** and **tExtractJSONFields** components use the input schema created to read JSON files/fields, whereas **tWriteJSONField** uses the output schema created to write a JSON field, which can be saved in a file by **tFileOutputJSON** or extracted by **tExtractJSONFields**.

For information about setting up input JSON file metadata, see [Setting up JSON metadata for an input file](#).

For information about setting up output JSON metadata, see [Setting up JSON metadata for an output file](#).

In the **Repository** view, expand the **Metadata** node, right click **File JSON**, and select **Create JSON Schema** from the contextual menu to open the **[New Json File]** wizard.

### 7.11.1. Setting up JSON metadata for an input file

This section describes how to define a file connection and upload a JSON schema for an input file. To define an output JSON file connection and schema, see [Setting up JSON metadata for an output file](#).

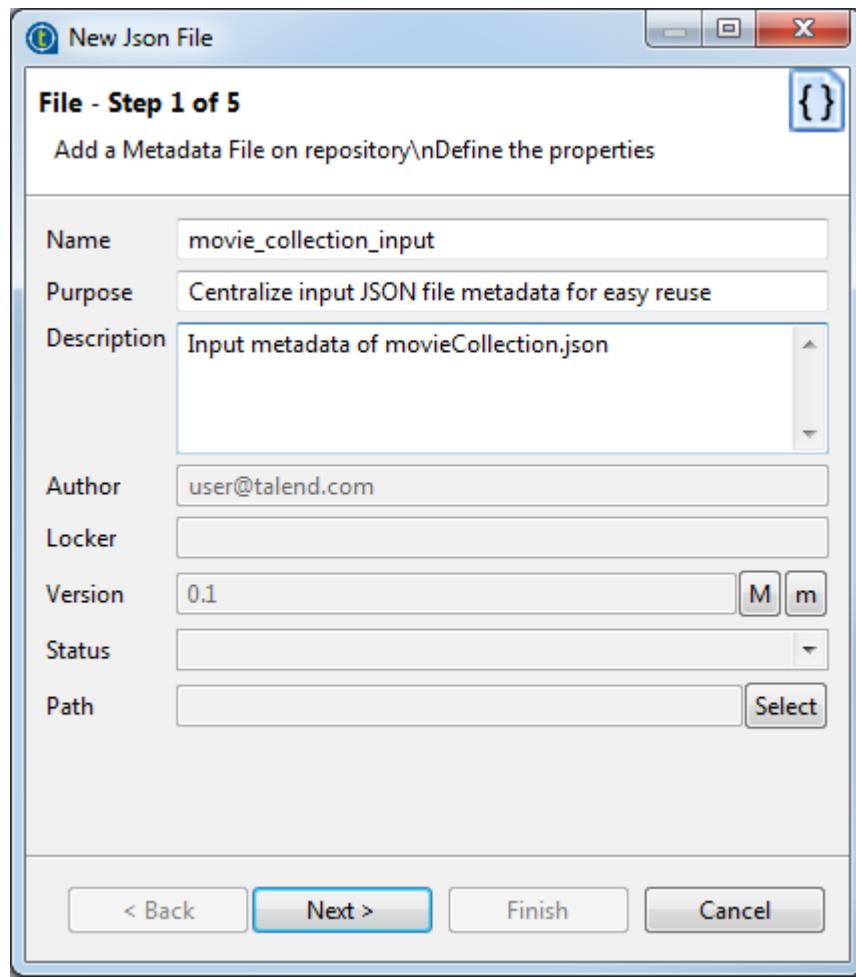
#### Defining the general properties

1. In the wizard, fill in the general information in the relevant fields to identify the JSON file metadata, including **Name**, **Purpose** and **Description**.

The **Name** field is required, and the information you provide in the **Description** field will appear as a tooltip when you move your mouse pointer over the file connection.



In this step, it is advisable to enter information that will help you distinguish between your input and output connections, which will be defined in the next step.



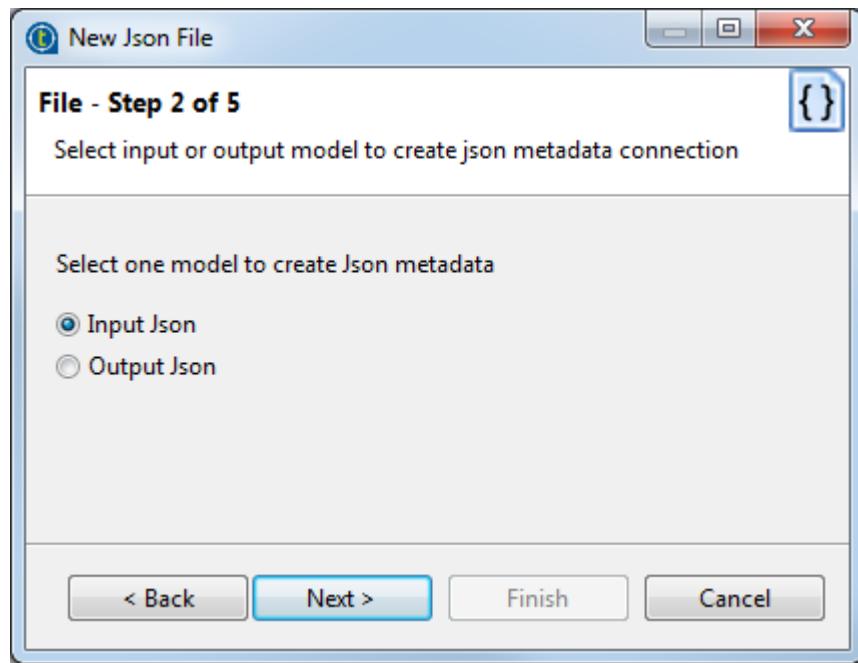
2. If needed, set the version and status in the **Version** and **Status** fields respectively.

You can also manage the version and status of a repository item in the **[Project Settings]** dialog box. For more information, see [Version management](#) and [Status management](#) respectively.

3. If needed, click the **Select** button next to the **Path** field to select a folder under the **File Json** node to hold your newly created file connection.
4. Click **Next** to select the type of metadata.

## Setting the type of metadata and loading the input file

1. In the dialog box, select **Input Json** and click **Next** to proceed to the next step of the wizard to load the input file.



2. From the **Read By** list box, select the type of query to read the source JSON file.

- **JsonPath:** read the JSON data based on a JsonPath query.

This is the default and recommended query type to read JSON data in order to gain performance and to avoid problems that you may encounter when reading JSON data based on an XPath query.

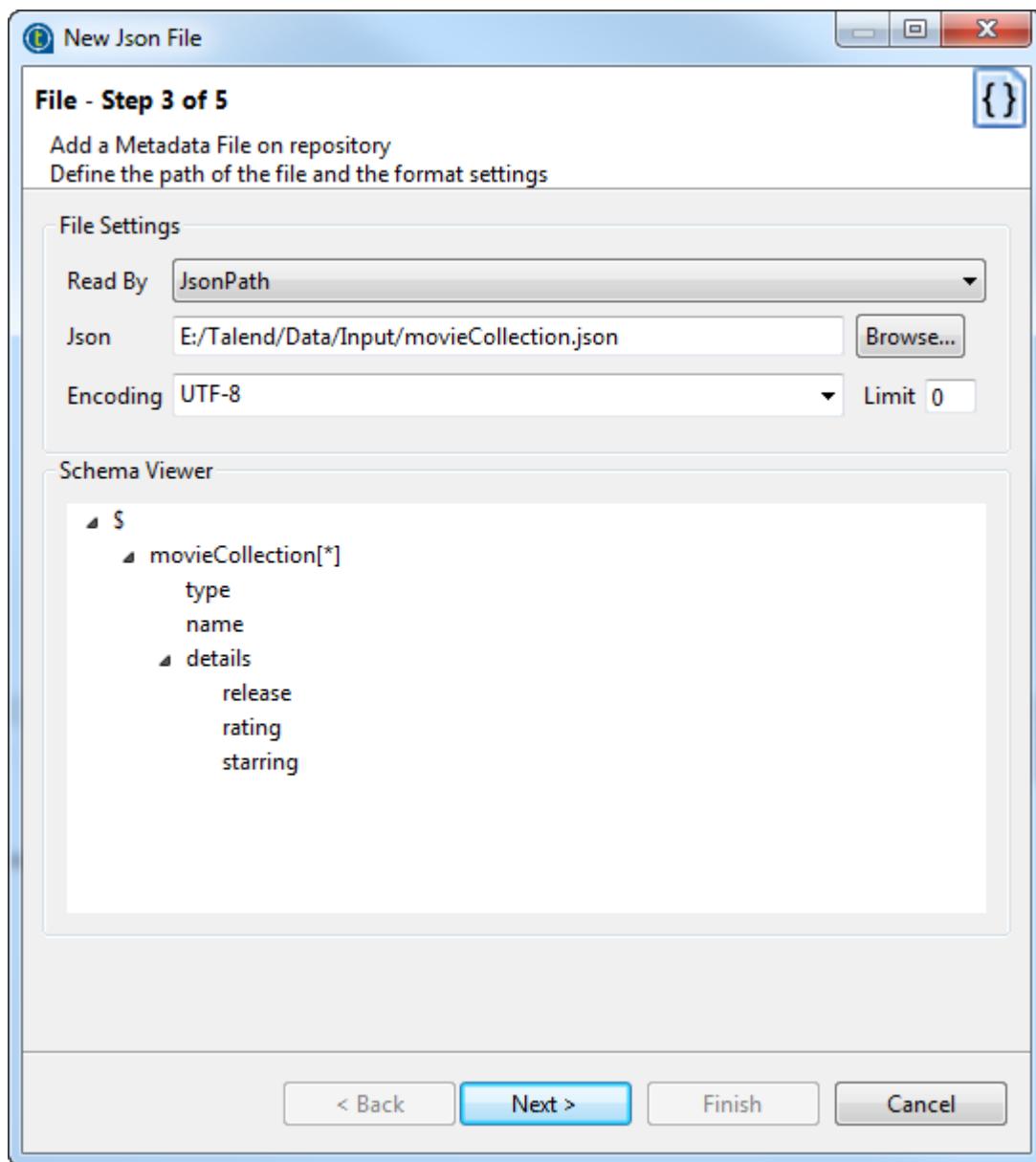
- **Xpath:** read the JSON data based on an XPath query.

3. Click **Browse...** and browse your directory to the JSON file to be uploaded. Alternatively, enter the full path to the file or the URL that links to the JSON file.

In this example, the input JSON file has the following content:

```
{
 "movieCollection": [
 {
 "type": "Action Movie",
 "name": "Brave Heart",
 "details": {
 "release": "1995",
 "rating": "5",
 "starring": "Mel Gibson"
 }
 },
 {
 "type": "Action Movie",
 "name": "Edge of Darkness",
 "details": {
 "release": "2010",
 "rating": "5",
 "starring": "Mel Gibson"
 }
 }
]
}
```

The **Schema Viewer** area displays a preview of the JSON structure. You can expand and visualize every level of the file's JSON tree structure.

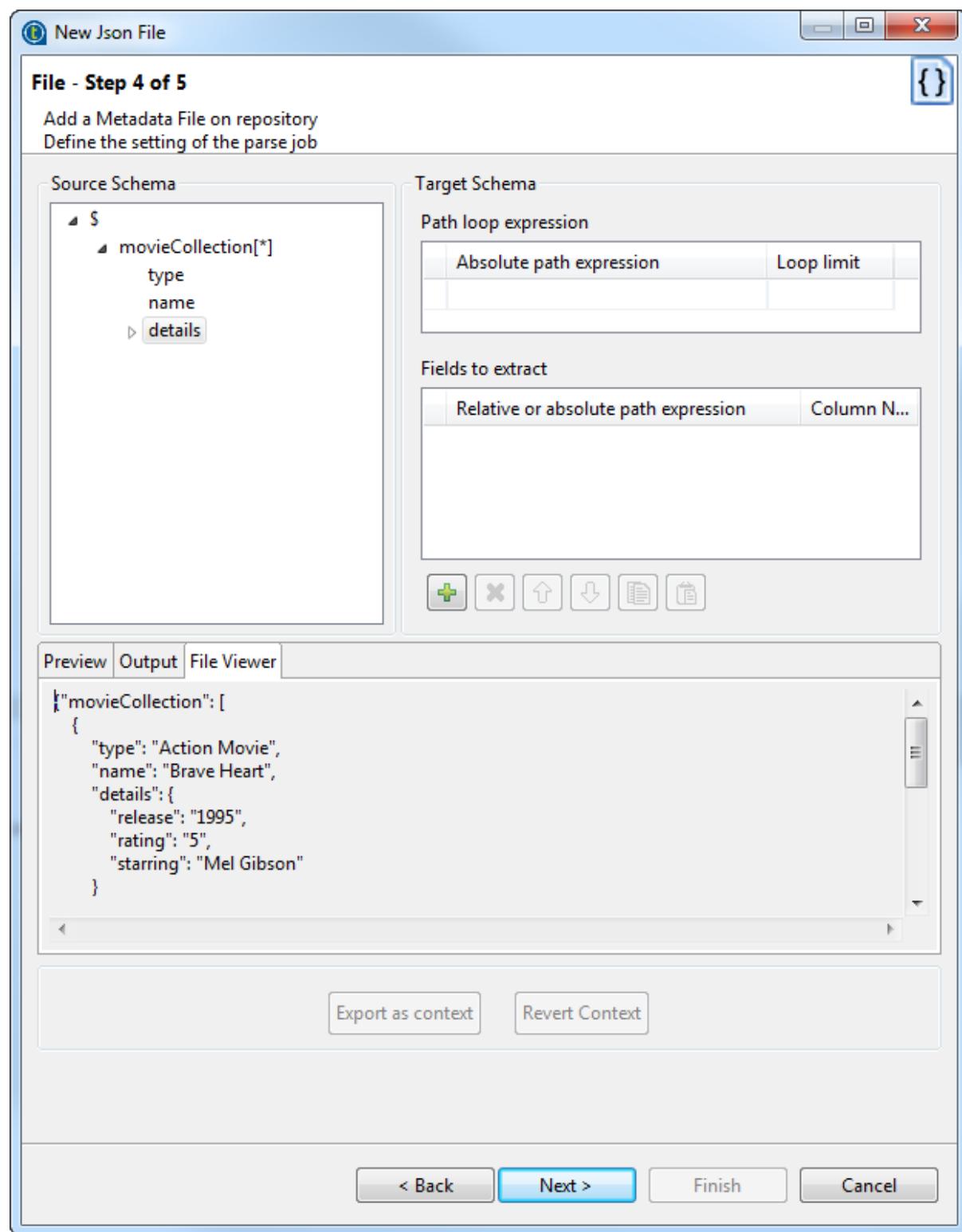


4. Enter the **Encoding** type in the corresponding field if the system does not detect it automatically.
5. In the **Limit** field, enter the number of levels in the JSON hierarchical depth to which you want to limit the JsonPath or XPath query, 0 for no limits.

Setting this parameter to a value less than 5 can help prevent the wizard from hanging in case of a large JSON file.
6. Click **Next** to define the schema parameters.

## Defining the schema

In this step you will set the schema parameters.



The schema definition window is composed of four views:

View	Description
<b>Source Schema</b>	Tree view of the JSON file.
<b>Target Schema</b>	Extraction and iteration information.
<b>Preview</b>	Preview of the target schema, together with the input data of the selected columns displayed in the defined order.
<b>File Viewer</b>	Preview of the JSON file's data.

1. Populate the **Path loop expression** field with the absolute JsonPath or XPath expression, depending on the type of query you have selected, for the node to be iterated upon. There are two ways to do this, either:
  - enter the absolute JsonPath or XPath expression for the node to be iterated upon (enter the full expression or press **Ctrl+Space** to use the autocompletion list),
  - drag the loop element node from the tree view under **Source schema** into the **Absolute path expression** field of the **Path loop expression** table.

An orange arrow links the node to the corresponding expression.

Absolute path expression	Loop limit
<code>\$ movieCollection[*]</code>	50



The **Path loop expression** definition is mandatory.

2. In the **Loop limit** field, specify the maximum number of times the selected node can be iterated.
3. Define the fields to be extracted by dragging the nodes from the **Source Schema** tree into the **Relative or absolute path expression** fields of the **Fields to extract** table.

Relative or absolute path expression	Column Name
<code>type</code>	<code>type</code>
<code>name</code>	<code>name</code>
<code>details.release</code>	<code>release</code>
<code>details.rating</code>	<code>rating</code>
<code>details.starring</code>	<code>starring</code>



You can select several nodes to drop onto the table by pressing **Ctrl** or **Shift** and clicking the nodes of interest.

4. If needed, you can add as many columns to be extracted as necessary, delete columns or change the column order using the toolbar:
  - Add or delete a column using the **[+]** and **[x]** buttons.
  - Change the order of the columns using the **↑** and **↓** buttons.
5. If you want your file schema to have different column names than those retrieved from the input file, enter new names in the corresponding **Column name** fields.

6. Click **Refresh Preview** to preview the target schema. The fields are consequently displayed in the schema according to the defined order.

type	name	release	rating	starring
Action Movie	Brave Heart	1995	5	Mel Gibson
Action Movie	Edge of Darkness	2010	5	Mel Gibson

7. Click **Next** to finalize the schema.

## Finalizing the schema

The last step of the wizard shows the end schema generated and allows you to customize the schema according to your needs.

Column	Key	Type	✓	N..	Date Pattern (Ctrl...)	Length	Precision
type	<input type="checkbox"/>	String	<input checked="" type="checkbox"/>			12	0
name	<input type="checkbox"/>	String	<input checked="" type="checkbox"/>			16	0
release	<input type="checkbox"/>	Integer	<input checked="" type="checkbox"/>			4	0
rating	<input type="checkbox"/>	Integer	<input checked="" type="checkbox"/>			1	0
starring	<input type="checkbox"/>	String	<input checked="" type="checkbox"/>			10	0

+
×
↑
↓
☰
↶
↷

< Back
Next >
Finish
Cancel

1. If needed, rename the schema (by default, *metadata*) and leave a comment.

Customize the schema if needed: add, remove or move schema columns, export the schema to an XML file, or replace the schema by importing an schema definition XML file using the tool bar.

Make sure the data type in the **Type** column is correctly defined.

For more information regarding Java data types, including date pattern, see [Java API Specification](#).

Below are the commonly used **Talend** data types:

- Object: a generic **Talend** data type that allows processing data without regard to its content, for example, a data file not otherwise supported can be processed with a **tFileInputRaw** component by specifying that it has a data type of Object.
  - List: a space-separated list of primitive type elements in an XML Schema definition, defined using the xsd:list element.
  - Document: a data type that allows processing an entire XML document without regarding to its content.
2. If the JSON file which the schema is based on has been changed, click the **Guess** button to generate the schema again. Note that if you have customized the schema, the **Guess** feature does not retain these changes.
  3. Click **Finish**. The new file connection, along with its schema, is displayed under the relevant **File Json** metadata node in the **Repository** tree view.

Now you can drag and drop the file connection or the schema of it from the **Repository** tree view onto the design workspace as a new **tFileInputJSON** or **tExtractJSONFields** component or onto an existing component to reuse the metadata. For further information about how to use the centralized metadata in a Job, see [How to use centralized metadata in a Job](#) and [How to set a repository schema](#).

To modify an existing file connection, right-click it from the **Repository** tree view, and select **Edit JSON** to open the file metadata setup wizard.

To add a new schema to an existing file connection, right-click the connection from the **Repository** tree view and select **Retrieve Schema** from the contextual menu.

To edit an existing file schema, right-click the schema from the **Repository** tree view and select **Edit Schema** from the contextual menu.

## 7.11.2. Setting up JSON metadata for an output file

This section describes how to define JSON metadata for an output file. To define JSON metadata for an input file, see [Setting up JSON metadata for an input file](#).

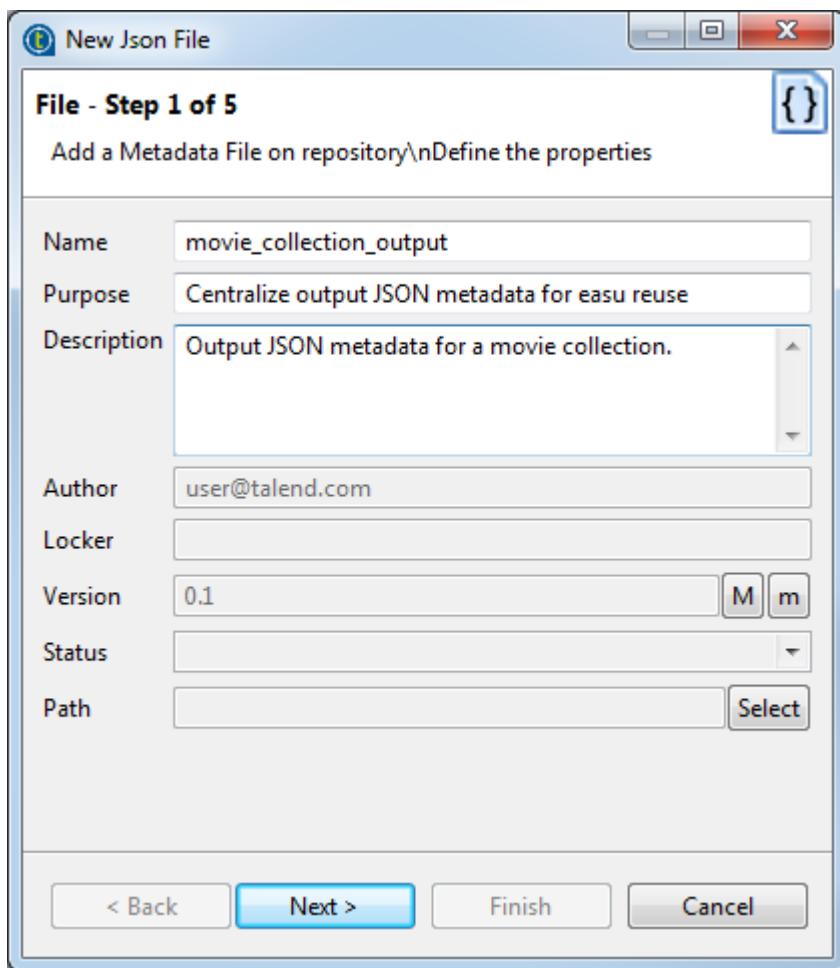
### Defining general properties

1. In the wizard, fill in the general information in the relevant fields to identify the JSON file metadata, including **Name**, **Purpose** and **Description**.

The **Name** field is required, and the information you provide in the **Description** field will appear as a tooltip when you move your mouse pointer over the file connection.



In this step, it is advisable to enter information that will help you distinguish between your input and output connections, which will be defined in the next step.



2. If needed, set the version and status in the **Version** and **Status** fields respectively.

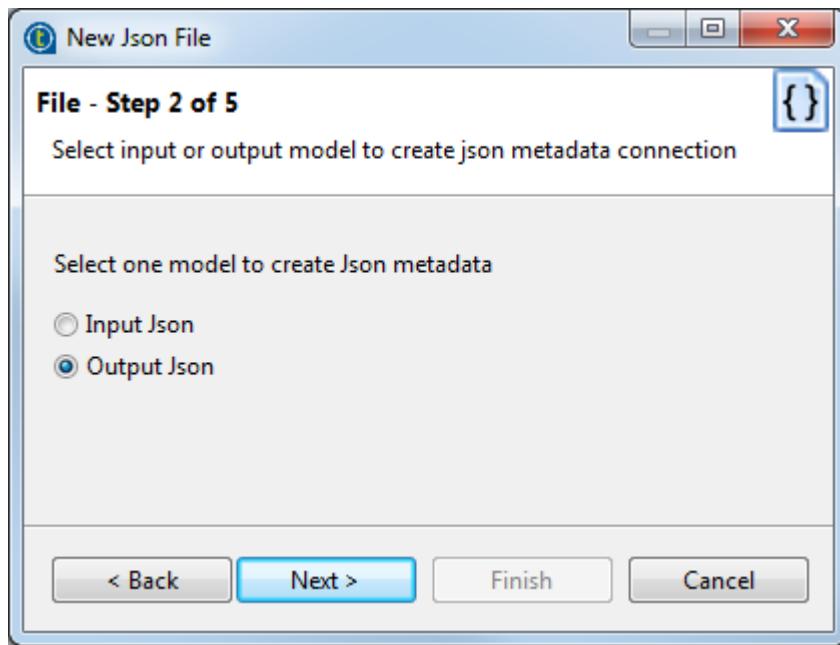
You can also manage the version and status of a repository item in the **[Project Settings]** dialog box. For more information, see [Version management](#) and [Status management](#) respectively.

3. If needed, click the **Select** button next to the **Path** field to select a folder under the **File Json** node to hold your newly created file connection.
4. Click **Next** to set the type of metadata.

### Setting the type of metadata and loading the template JSON file

In this step, the type of schema is set as either input or output. For this procedure, the schema of interest is output.

1. From the dialog box, select **Output JSON** click **Next** to proceed to the next step of the wizard.



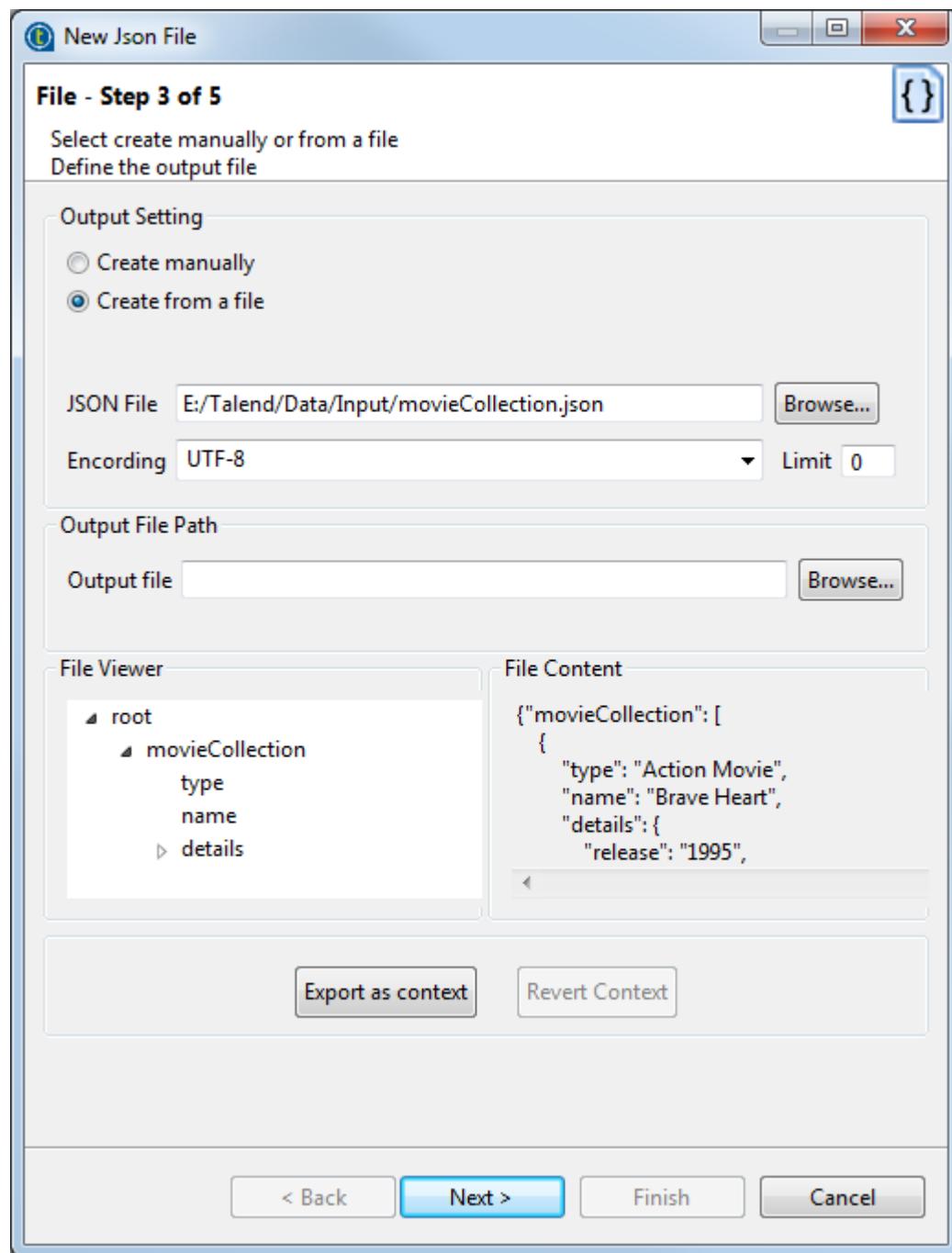
2. Choose whether to create the output metadata manually or from an existing JSON file as a template.

If you choose the **Create manually** option you will have to configure the schema and link the source and target columns yourself. The output JSON file/field is created via a Job using a JSON output component such as **tWriteJSONField**.

In this example, we will create the output metadata by loading an existing JSON file. Therefore, select the **Create from a file** option.

3. Click the **Browse...** button next to the **JSON File** field, browse to the access path to the JSON file the structure of which is to be applied to the output JSON file/field, and double-click the file. Alternatively, enter the full path to the file or the URL which links to the template JSON file.

The **File Viewer** area displays a preview of the JSON structure, and the **File Content** area displays a maximum of the first 50 rows of the file.



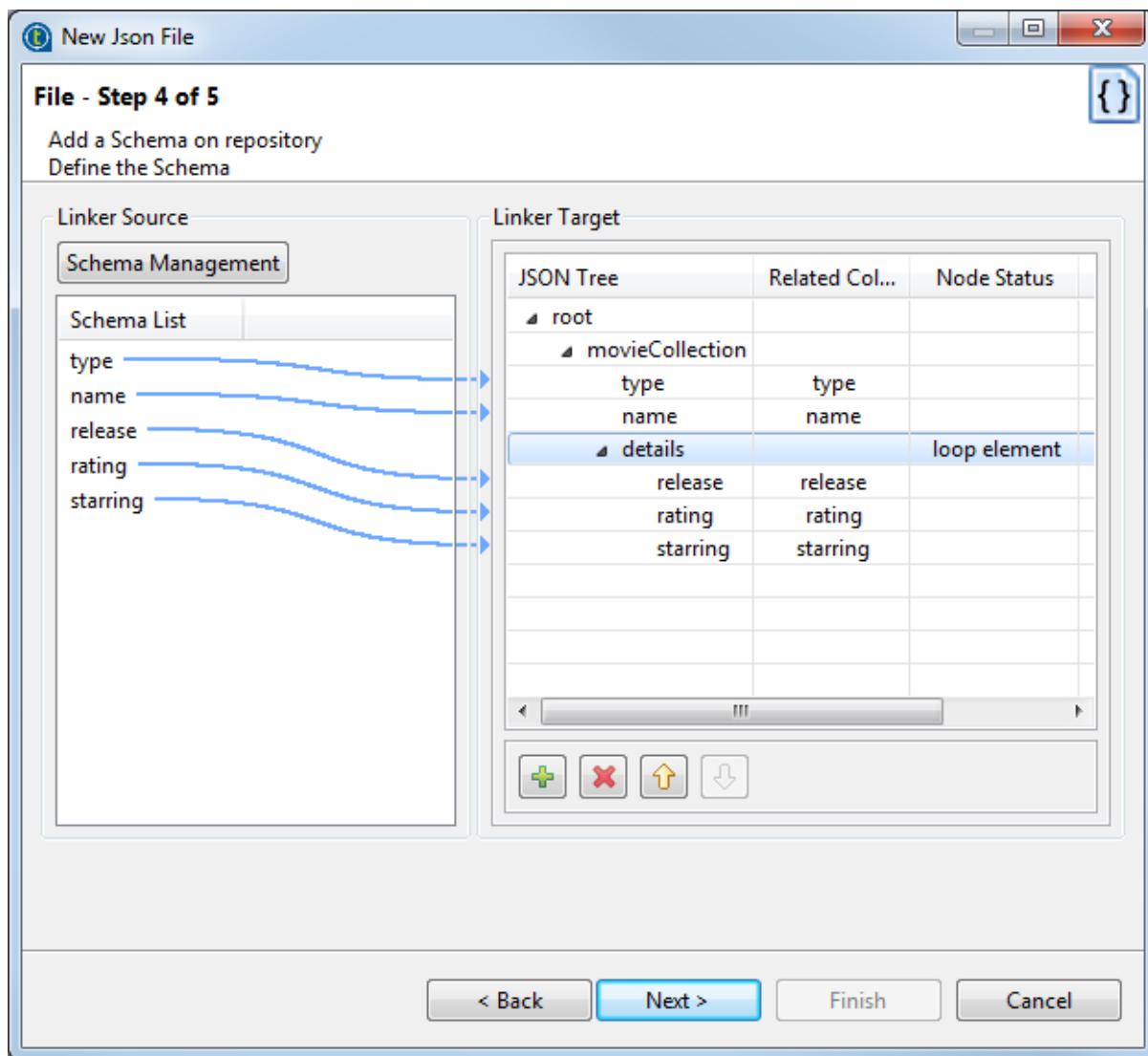
4. Enter the **Encoding** type in the corresponding field if the system does not detect it automatically.
5. In the **Limit** field, enter the number of levels in the JSON hierarchical depth to which you want to limit the JsonPath or XPath query, 0 for no limits.

Setting this parameter to a value less than 5 can help prevent the wizard from hanging in case of a large JSON file.

6. Optionally, specify an output file path.
7. Click **Next** to define the schema.

## Defining the schema

Upon completion of the previous operations, the columns in the **Linker Source** area are automatically mapped to the corresponding ones in the **Linker Target** area, as indicated by blue arrow links..



In this step, you need to define the output schema. The following table describes how:

To...	Perform the following...
Define a loop element	<p>In the <b>Linker Target</b> area, right-click the element of interest and select <b>Set As Loop Element</b> from the contextual menu.</p> <p> It is a mandatory operation to define an element to run a loop on.</p>
Define a group element	<p>In the <b>Linker Target</b> area, right-click the element of interest and select <b>Set As Group Element</b> from the contextual menu.</p> <p> You can set a parent element of the loop element as a group element on the condition that the parent element is not the root of the JSON tree.</p>
Create a child element for an element	<p>In the <b>Linker Target</b> area,</p> <ul style="list-style-type: none"> <li>Right-click the element of interest and select <b>Add Sub-element</b> from the contextual menu, enter a name for the sub-element in the dialog box that appears, and click <b>OK</b>.</li> <li>Select the element of interest, click the <b>[+]</b> button at the bottom, select <b>Create as sub-element</b> in the dialog box that appears, and click <b>OK</b>. Then, enter a name for the sub-element in the next dialog box and click <b>OK</b>.</li> </ul>
Create an attribute for an element	<p>In the <b>Linker Target</b> area,</p> <ul style="list-style-type: none"> <li>Right-click the element of interest and select <b>Add Attribute</b> from the contextual menu, enter a name for the attribute in the dialog box that appears, and click <b>OK</b>.</li> </ul>

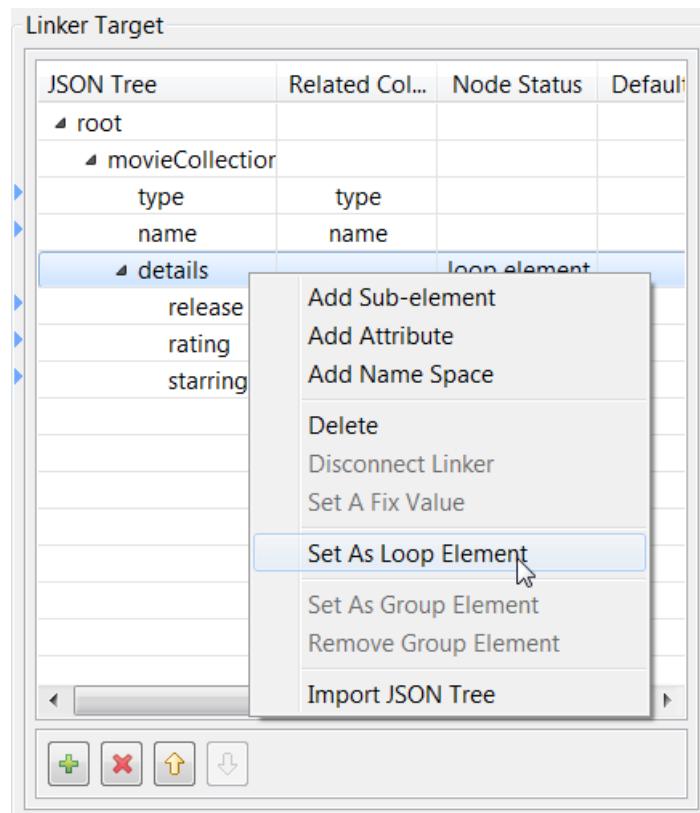
To...	Perform the following...
	<ul style="list-style-type: none"> <li>Select the element of interest, click the [+] button at the bottom, select <b>Create as attribute</b> in the dialog box that appears, and click <b>OK</b>. Then, enter a name for the attribute in the next dialog box and click <b>OK</b>.</li> </ul>
Create a name space for an element	<p>In the <b>Linker Target</b> area,</p> <ul style="list-style-type: none"> <li>Right-click the element of interest and select <b>Add Name Space</b> from the contextual menu, enter a name for the name space in the dialog box that appears, and click <b>OK</b>.</li> <li>Select the element of interest, click the [+] button at the bottom, select <b>Create as name space</b> in the dialog box that appears, and click <b>OK</b>. Then, enter a name for the name space in the next dialog box and click <b>OK</b>.</li> </ul>
Delete one or more elements/attributes/name spaces	<p>In the <b>Linker Target</b> area,</p> <ul style="list-style-type: none"> <li>Right-click the element(s)/attribute(s)/name space(s) of interest and select <b>Delete</b> from the contextual menu.</li> <li>Select the element(s)/attribute(s)/name space(s) of interest and click the [x] button at the bottom.</li> <li>Select the element(s)/attribute(s)/name space(s) of interest and press the <b>Delete</b> key.</li> </ul> <p> Deleting an element will also delete its children, if any.</p>
Adjust the order of one or more elements	<p>In the <b>Linker Target</b> area, select the element(s) of interest and click the  and  buttons.</p>
Set a static value for an element/attribute/name space	<p>In the <b>Linker Target</b> area, right-click the element/attribute/name space of interest and select <b>Set A Fix Value</b> from the contextual menu.</p> <p></p> <ul style="list-style-type: none"> <li>The value you set will replace any value retrieved for the corresponding column from the incoming data flow in your Job.</li> <li>You can set a static value for a child element of the loop element only, on the condition that the element does not have its own children and does not have a source-target mapping on it.</li> </ul>
Create a source-target mapping	<p>Select the column of interest in the <b>Linker Source</b> area, drop it onto the node of interest in the <b>Linker Target</b> area, and select <b>Create as sub-element of target node</b>, <b>Create as attribute of target node</b>, or <b>Add linker to target node</b> according to your need in the dialog box that appears, and click <b>OK</b>.</p> <p>If you choose an option that is not permitted for the target node, you will see a warning message and your operation will fail.</p>
Remove a source-target mapping	<p>In the <b>Linker Target</b> area, right-click the node of interest and select <b>Disconnect Linker</b> from the contextual menu.</p>
Create a JSON tree from another JSON file	<p>Right-click any schema item in the <b>Linker Target</b> area and select <b>Import JSON Tree</b> from the contextual menu to load another JSON file. Then, you need to create source-target mappings manually and define the output schema all again.</p>



You can select and drop several fields at a time, using the **Ctrl + Shift** technique to make multiple selections, therefore making mapping faster. You can also make multiple selections for right-click operations.

1. In the **Linker Target** area, right-click the element you want to set as the loop element and select **Set As Loop Element** from the contextual menu.

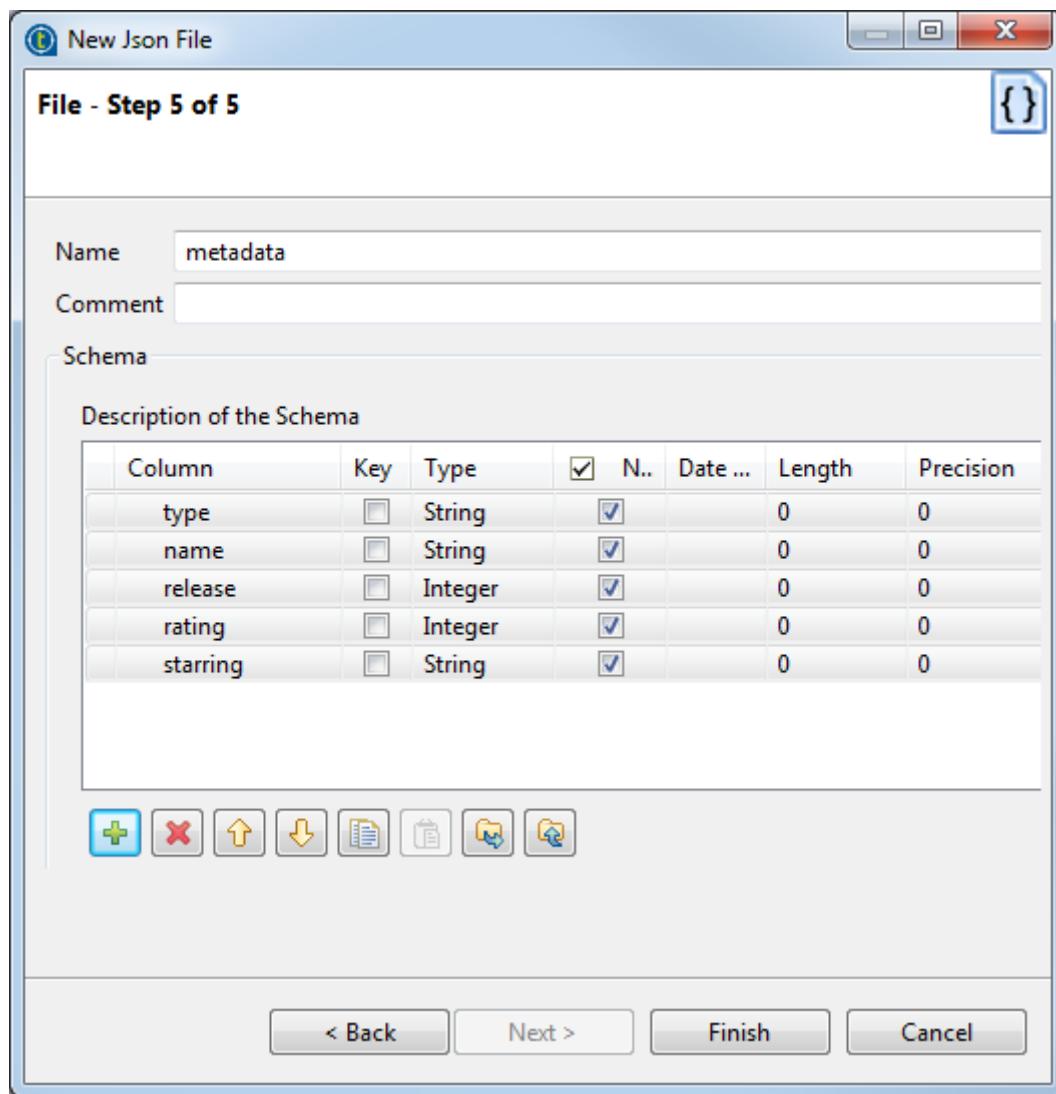
In this example, define a loop to run on the *details* element.



2. Customize the mappings if needed.
3. Click **Next** to finalize the schema.

## Finalizing the end schema

The last step of the wizard shows the end schema generated and allows you to customize the schema according to your needs.



1. If needed, rename the schema (by default, *metadata*) and leave a comment.

Customize the schema if needed: add, remove or move schema columns, export the schema to an XML file, or replace the schema by importing an schema definition XML file using the tool bar.

Make sure the data type in the **Type** column is correctly defined.

For more information regarding Java data types, including date pattern, see [Java API Specification](#).

Below are the commonly used **Talend** data types:

- Object: a generic **Talend** data type that allows processing data without regard to its content, for example, a data file not otherwise supported can be processed with a **tFileInputRaw** component by specifying that it has a data type of Object.
- List: a space-separated list of primitive type elements in an XML Schema definition, defined using the xsd:list element.
- Document: a data type that allows processing an entire XML document without regarding to its content.

2. If the JSON file which the schema is based on has been changed, click the **Guess** button to generate the schema again. Note that if you have customized the schema, the **Guess** feature does not retain these changes.

3. Click **Finish**. The new file connection, along with its schema, is displayed under the relevant **File Json** metadata node in the **Repository** tree view.

Now you can drag and drop the file connection or the schema of it from the **Repository** tree view onto the design workspace as a new **tWriteJSONField** component or onto an existing component to reuse the metadata. For further information about how to use the centralized metadata in a Job, see [How to use centralized metadata in a Job](#) and [How to set a repository schema](#).

To modify an existing file connection, right-click it from the **Repository** tree view, and select **Edit JSON** to open the file metadata setup wizard.

To add a new schema to an existing file connection, right-click the connection from the **Repository** tree view and select **Retrieve Schema** from the contextual menu.

To edit an existing file schema, right-click the schema from the **Repository** tree view and select **Edit Schema** from the contextual menu.

## 7.12. Centralizing LDAP connection metadata

If you often need to access an LDAP directory, you want to centralize your LDAP server connection in the **Repository** tree view for easy reuse.

You can create an LDAP connection either from an accessible LDAP directory, or by saving the LDAP settings defined in a Job.

To create an LDAP connection from an accessible LDAP directory, expand the **Metadata** node in the **Repository** tree view, right-click the **LDAP** tree node, and select **Create LDAP schema** from the contextual menu to open the **[Create new LDAP schema]** wizard.

To centralize an LDAP connection and its schema you have already defined in a Job, click the  icon in the **Basic settings** view of the relevant component, with its **Property Type** set to **Built-In**, to open the **[Create new LDAP schema]** wizard.

Unlike the DB connection wizard, the LDAP wizard gathers both LDAP server connection and schema definition in a five-step procedure.

### Defining the general properties

1. Fill in the general information in the relevant fields to identify the LDAP connection to be created, including **Name**, **Purpose** and **Description**.

The **Name** field is required, and the information you provide in the **Description** field will appear as a tooltip when you move your mouse pointer over the LDAP connection.

2. If needed, set the version and status in the **Version** and **Status** fields respectively. You can also manage the version and status of a **Repository** item in the **[Project Settings]** dialog box. For more information, see [Version management](#) and [Status management](#) respectively.
3. If needed, click the **Select** button next to the **Path** field to select a folder under the **LDAP** node to hold your newly created LDAP connection.
4. Click **Next** to define your LDAP server connection details.

### Defining the server connection

1. Fill the connection details.

**File - Step 2 of 5**

Add a Metadata File on repository  
Define the path of the file and the format settings

Network Parameter

Hostname:	Your-LDAP-IP
Port:	389
Encryption method:	LDAP

Click the button below to check the connection.

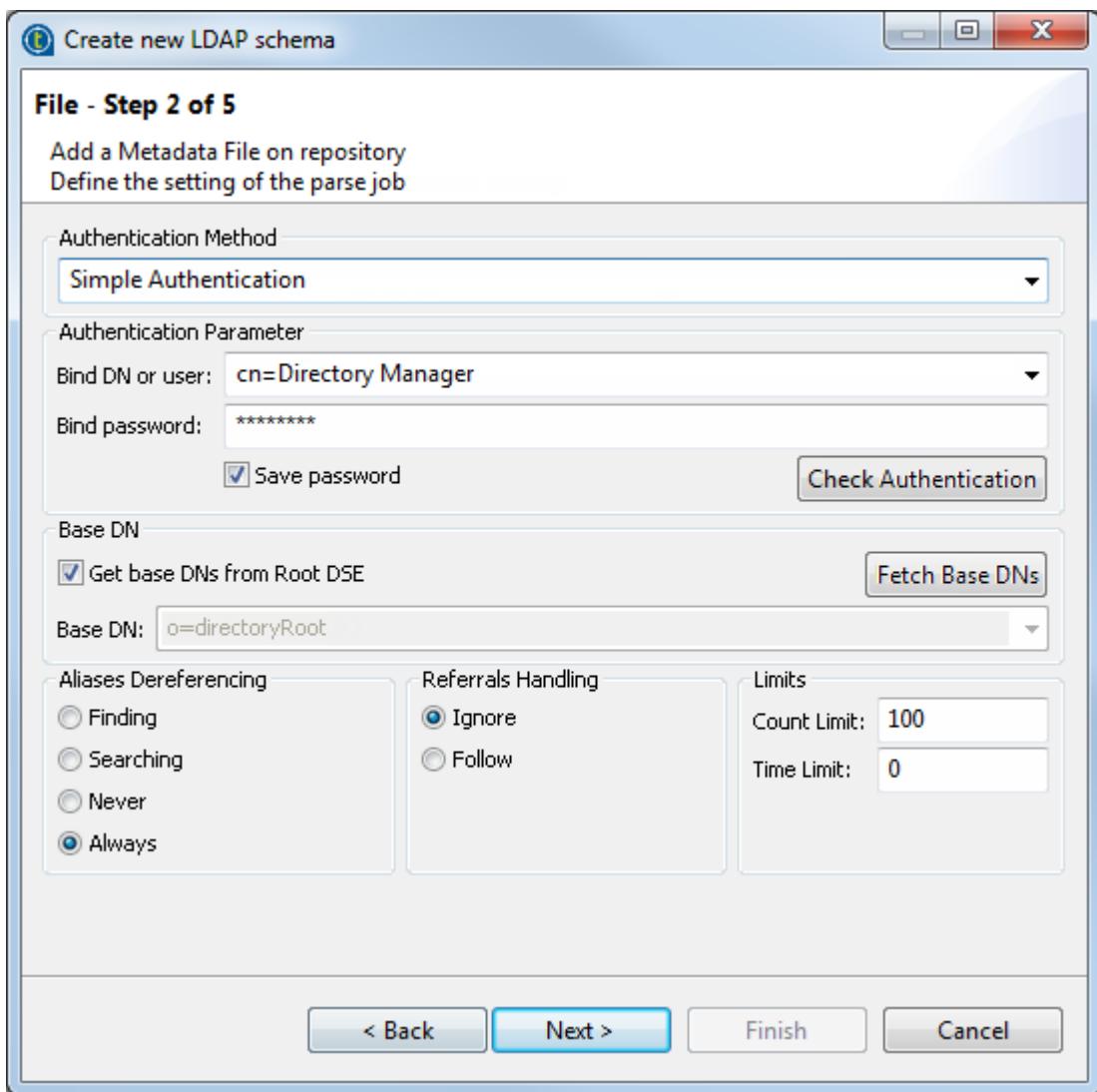
**Check Network Parameter**

Field	Description
Host	LDAP Server host name or IP address
Port	Listening port to the LDAP directory
Encryption method	<b>LDAP</b> : no encryption is used <b>LDAPS</b> : secured LDAP <b>TLS</b> : certificate is used

2. Then check your connection using **Check Network Parameter** to verify the connection and activate the **Next** button.
3. Click **Next** to continue.

### Configuring LDAP access parameters

1. In this view, set the authentication and data access mode.



Field	Description
<b>Authentication method</b>	<b>Simple authentication:</b> requires <b>Authentication Parameters</b> field to be filled in <b>Anonymous authentication:</b> does not require authentication parameters
<b>Authentication Parameters</b>	<b>Bind DN or User:</b> login as expected by the LDAP authentication method <b>Bind password:</b> expected password <b>Save password:</b> remembers the login details.
<b>Get Base DN from Root DSE / Base DN</b>	Path to user's authorized tree leaf <b>Fetch Base DNs</b> button retrieves the DN automatically from Root.
<b>Alias Dereferencing</b>	<b>Never</b> : allows to improve search performance if you are sure that no aliases is to be dereferenced. By default, <b>Always</b> is to be used. <b>Always</b> : Always dereference aliases <b>Never</b> : Never dereferences aliases. <b>Searching</b> : Dereferences aliases only after name resolution. <b>Finding</b> : Dereferences aliases only during name resolution
<b>Referral Handling</b>	Redirection of user request: <b>Ignore</b> : does not handle request redirections

Field	Description
	<b>Follow:</b> does handle request redirections
<b>Limit</b>	Limited number of records to be read

2. Click **Check authentication** to verify your access rights.
3. Click **Fetch Base DNs** to retrieve the DN and click the **Next** button to continue.
4. If any third-party libraries required for setting up an LDAP connection are found missing, an external module installation wizard appears. Install the required libraries as guided by the wizard. For more information on installing third-party modules, see the *Talend Installation and Upgrade Guide*.

## Defining the schema

1. Select the attributes to be included in the schema structure.

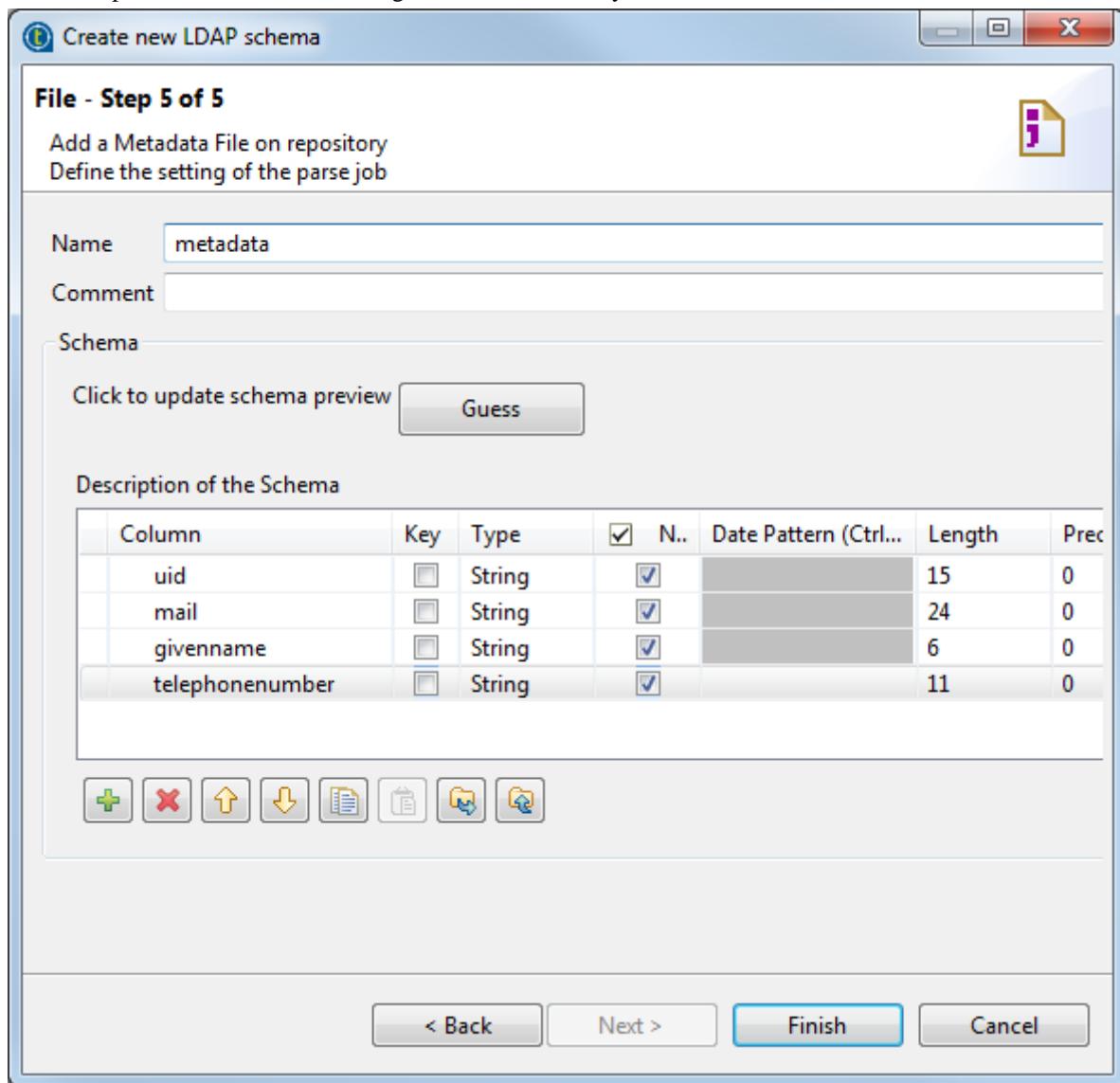
Add a filter if you want selected data only.

The screenshot shows the 'Create new LDAP schema' dialog box, specifically Step 4 of 5. The title bar says 'Create new LDAP schema'. The main area is titled 'File - Step 4 of 5' and has the sub-instruction 'Add a Metadata File on repository Define the setting of the parse job'. On the left, there's a list titled 'List attributes of LDAP Schema' with an 'Attributes' column containing checkboxes for 'description', 'userpassword', 'uid', 'sn', and 'cn'. To the right of this is a 'Filter' field containing the expression '(&(objectClass=\*))'. Below these sections is a 'Preview' area with a 'Refresh Preview' button. A table preview shows four columns: 'uid', 'mail', 'givenname', and 'telephonenumber'. The data rows are: PIERRE DUPONT, Pierre.Dupont@talend.com, PIERRE, 00149684750; PIERRE DUPON..., mhirt78@talend.com, PIERRE; mhirt; greg. At the bottom of the dialog are buttons for '< Back', 'Next >', 'Finish', and 'Cancel'.

2. Click **Refresh Preview** to display the selected column and a sample of the data.
3. Click **Next** to continue.

## Finalizing the end schema

The last step shows the LDAP schema generated and allows you to further customize the end schema.



1. If needed, rename the metadata in the **Name** field (*metadata*, by default), add a **Comment**, and make further modifications, for example:

- Redefine the columns by editing the relevant fields.
- Add or delete a column using the and buttons.
- Change the order of the columns using the and buttons.

Make sure the data type in the **Type** column is correctly defined.

For more information regarding Java data types, including date pattern, see [Java API Specification](#).

Below are the commonly used **Talend** data types:

- Object: a generic **Talend** data type that allows processing data without regard to its content, for example, a data file not otherwise supported can be processed with a **tFileInputRaw** component by specifying that it has a data type of Object.

- List: a space-separated list of primitive type elements in an XML Schema definition, defined using the xsd:list element.
  - Document: a data type that allows processing an entire XML document without regarding to its content.
2. If the LDAP directory which the schema is based on has changed, use the **Guess** button to generate again the schema. Note that if you customized the schema, your changes will not be retained after the **Guess** operation.
  3. Click **Finish**. The new schema is displayed under the relevant LDAP connection node in the **Repository** tree view.

Now you can drag and drop the file connection or any schema of it from the **Repository** tree view onto the design workspace as a new component or onto an existing component to reuse the metadata.

To modify an existing file connection, right-click it from the **Repository** tree view, and select **Edit LDAP schema** to open the file metadata setup wizard.

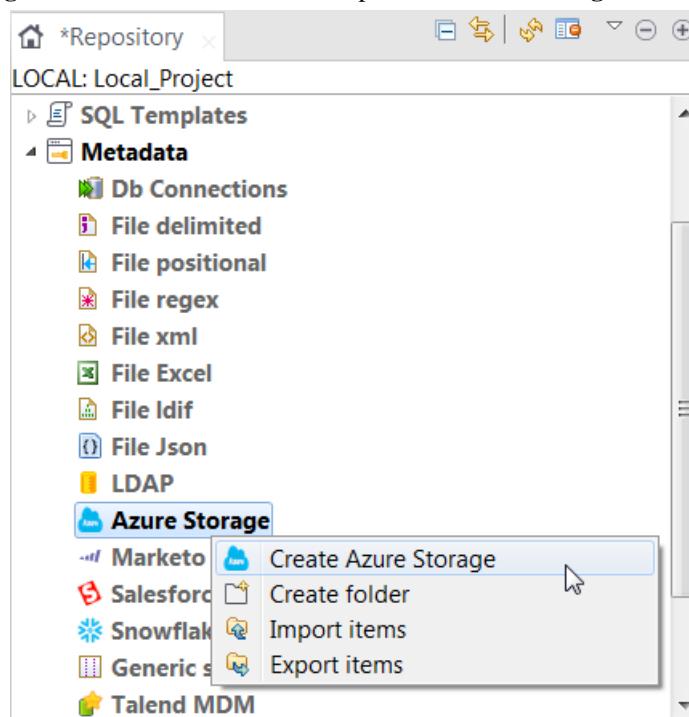
To add a new schema to an existing file connection, right-click the connection from the **Repository** tree view and select **Retrieve Schema** from the contextual menu.

To edit an existing file schema, right-click the schema from the **Repository** tree view and select **Edit Schema** from the contextual menu.

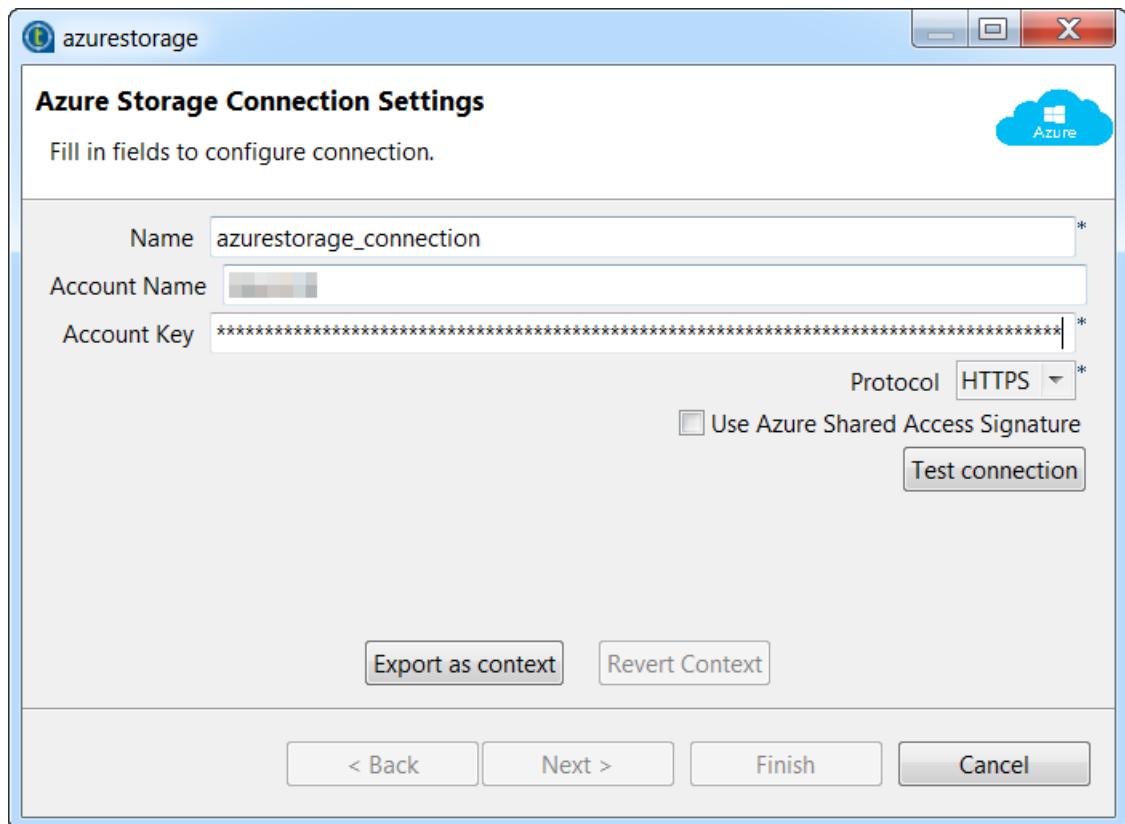
## 7.13. Centralizing Azure Storage metadata

You can use the Azure Storage metadata wizard provided by *Talend Studio* to set up quickly a connection to Azure Storage and retrieve the schema of your interested container(s), queue(s), and table(s).

1. In the **Repository** tree view, expand the **Metadata** node, right-click the **Azure Storage** tree node, and select **Create Azure Storage** from the contextual menu to open the **[Azure Storage]** wizard.



2. In the **Azure Storage Connection Settings** dialog box, specify (or update if needed) the values for the properties listed in the following table.

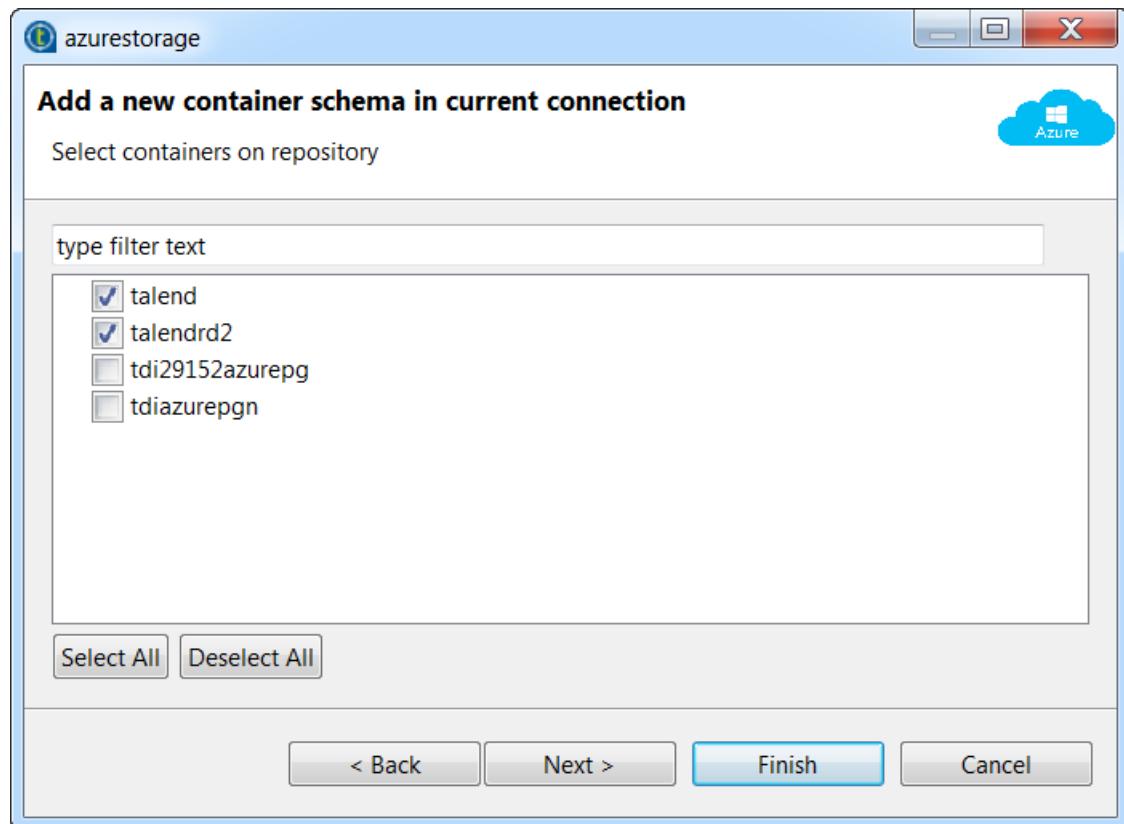


Property	Description
<b>Name</b>	Enter the name for the connection to be created.
<b>Account Name</b>	Enter the name of the storage account you need to access. A storage account name can be found in the Manage Access Keys dashboard of the Microsoft Azure Storage system to be used.
<b>Account Key</b>	Enter the key associated with the storage account you need to access. Two keys are available for each account and by default, either of them can be used for this access.
<b>Protocol</b>	Select the protocol for this connection to be created.
<b>Use Azure Shared Access Signature</b>	Select this check box to use a shared access signature to access the storage resources without need for the account key. In the Azure Shared Access Signature field displayed, enter your shared access signature between double quotation marks. For more information, see <a href="#">Using Shared Access Signatures (SAS)</a> .

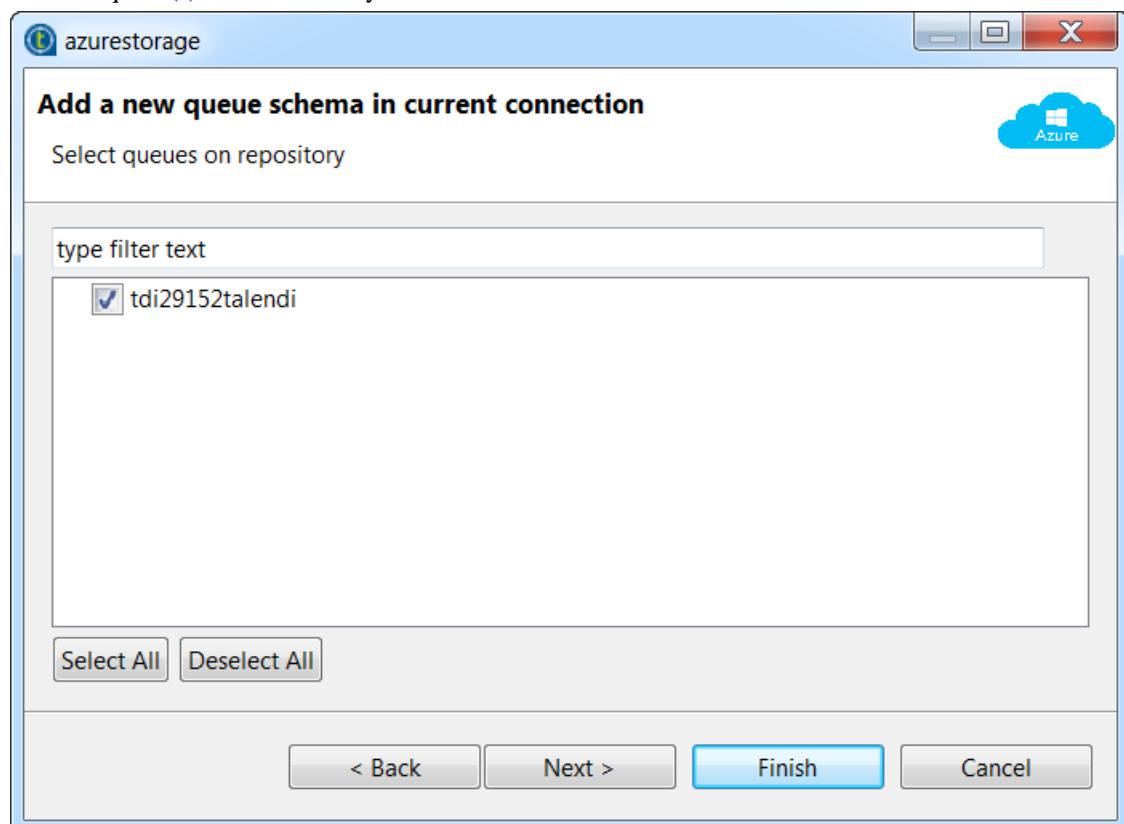
- Click **Test connection** to verify the configuration.

A connection successful dialog box will prompt up if the connection information provided is correct. Then click **OK** to close the dialog box. The **Next** button will be available to use.

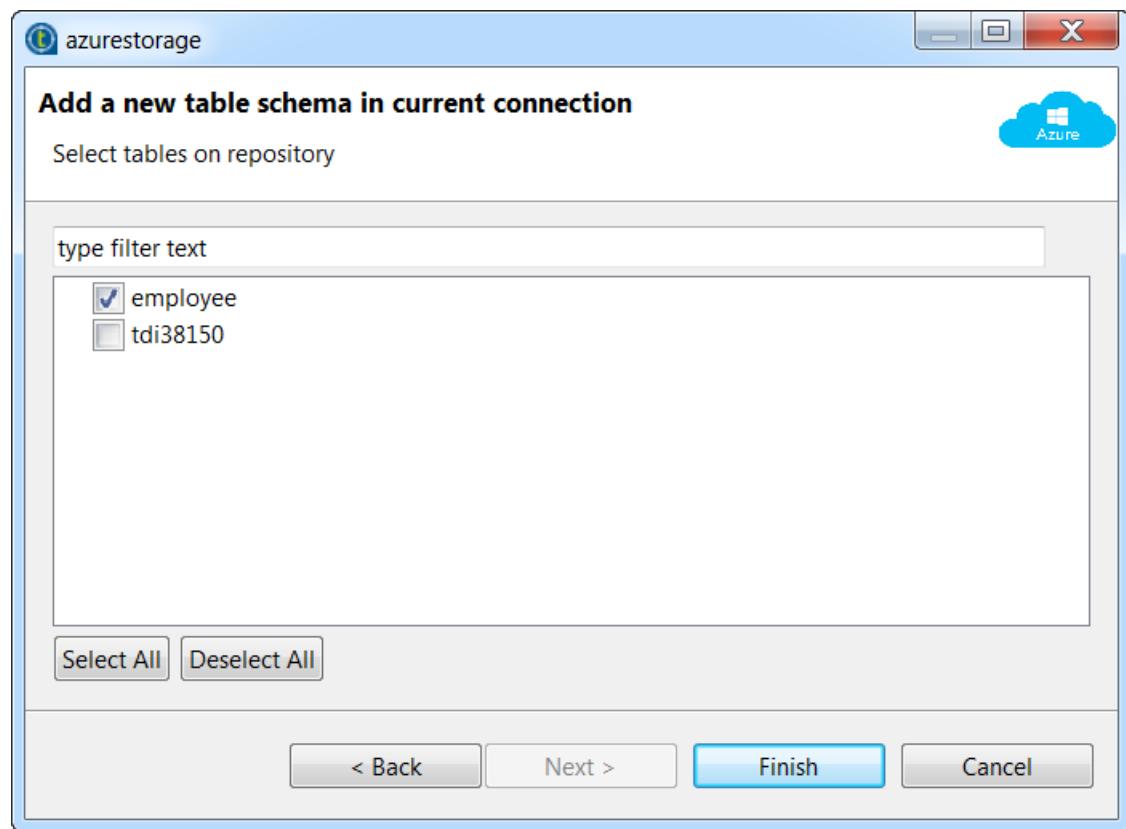
- Click **Next** and in the **[Add a new container schema in current connection]** dialog box displayed, select your interested container(s) whose schema you want to retrieve.



5. Click **Next** and in the **[Add a new queue schema in current connection]** dialog box displayed, select your interested queue(s) whose schema you want to retrieve.

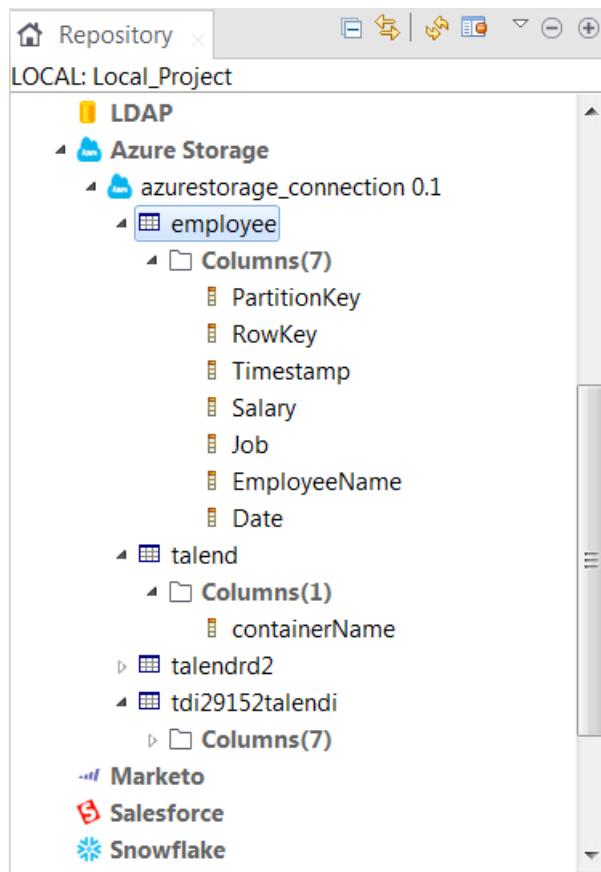


6. Click **Next** and in the **[Add a new table schema in current connection]** dialog box displayed, select your interested table(s) whose schema you want to retrieve.



7. Click **Finish** to complete the procedure.

The newly created Azure Storage connection is displayed under the **Azure Storage** node in the **Repository** tree view, along with the schema of your interested container(s), queue(s), and table(s).



You can now add a Azure Storage component onto the design workspace by dragging and dropping the Azure Storage connection created or any container/queue/table retrieved from the **Repository** view to reuse the connection and/or schema information. For more information about dropping component metadata in the design workspace, see [How to use centralized metadata in a Job](#). For more information about the usage of the Azure Storage components, see the related documentation for the Azure Storage components.

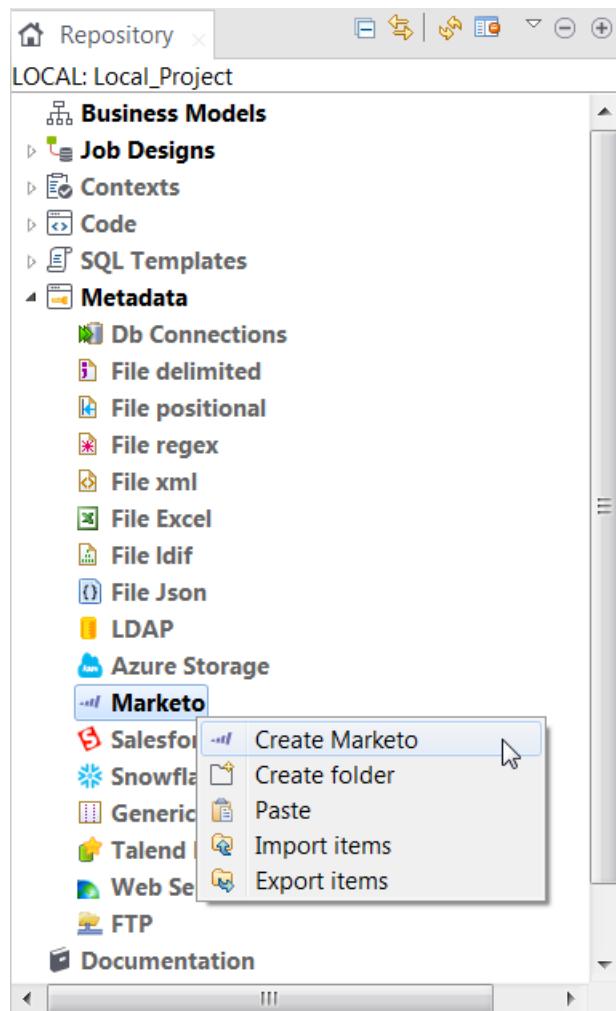
To modify the Azure Storage connection metadata created, right-click the connection node in the **Repository** tree view and select **Edit Azure Storage** from the contextual menu to open the metadata setup wizard.

To edit the schema of an interested container/queue/table, right-click the container/queue/table node in the **Repository** tree view and select **Edit Schema** from the contextual menu to open the update schema wizard.

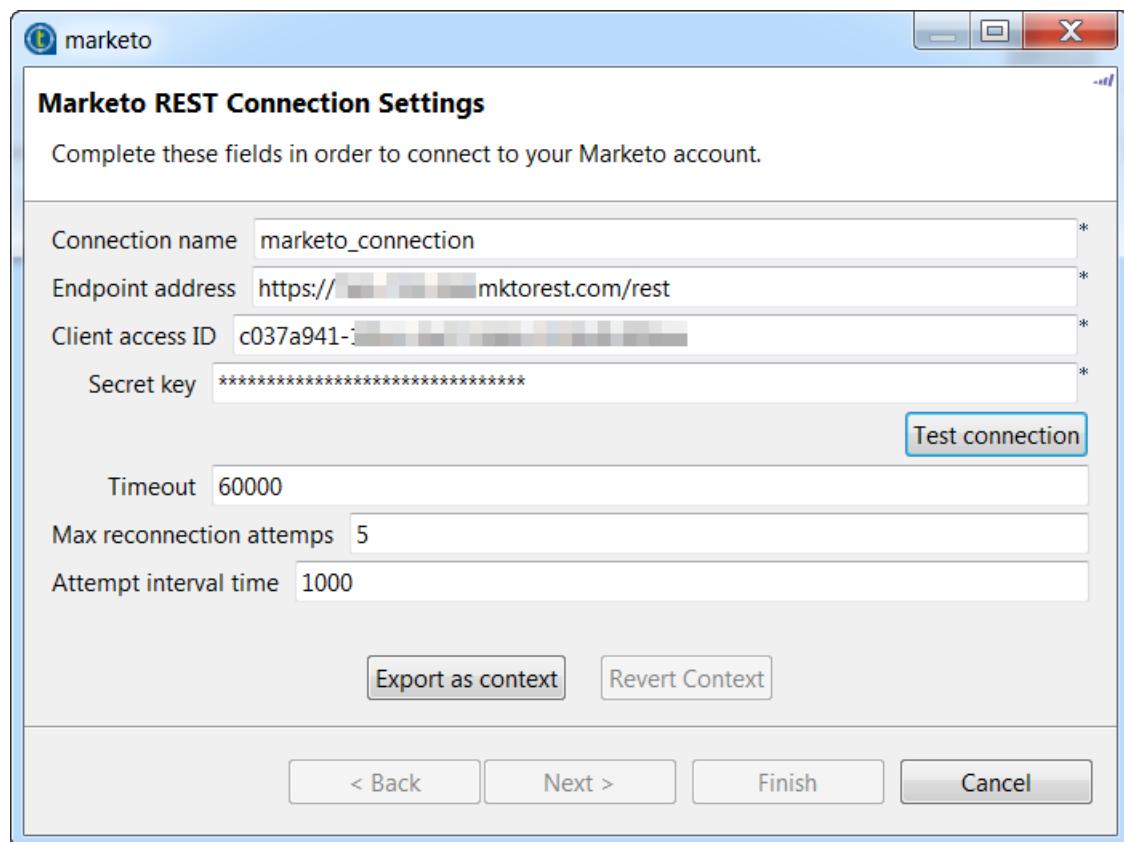
## 7.14. Centralizing Marketo metadata

You can use the Marketo metadata wizard provided by *Talend Studio* to set up quickly a connection to Marketo and retrieve the schema of your interested custom objects using REST API.

1. In the **Repository** tree view, expand the **Metadata** node, right-click the **Marketo** tree node, and select **Create Marketo** from the contextual menu to open the **[Marketo]** wizard.



2. In the **Marketo REST Connection Settings** dialog box, specify (or update if needed) the values for the properties listed in the following table.

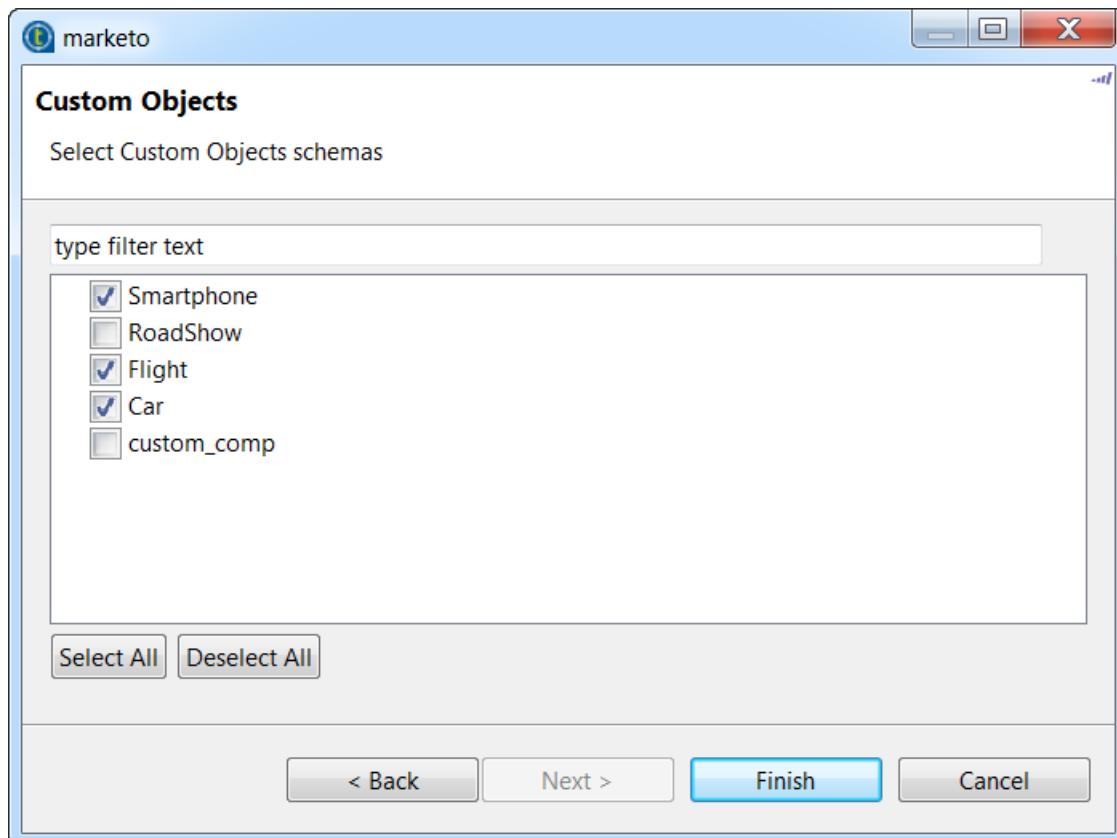


Property	Description
<b>Connection name</b>	Enter the name for the connection to be created.
<b>Endpoint address</b>	Enter the API Endpoint URL of the Marketo Web Service. The API Endpoint URL can be found on the Marketo Admin > <b>Web Services</b> panel.
<b>Client access ID</b>	Enter the client Id for the access to the Marketo Web Service.
<b>Secret key</b>	Enter the client secret for the access to the Marketo Web Service.
<b>Timeout</b>	Enter the timeout value (in milliseconds) for the connection to the Marketo Web Service before terminating the attempt.
<b>Max reconnection attempts</b>	Enter the maximum number of reconnect attempts to the Marketo Web Service before giving up.
<b>Attempt interval time</b>	Enter the time period (in milliseconds) between subsequent reconnection attempts.

3. Click **Test connection** to verify the configuration.

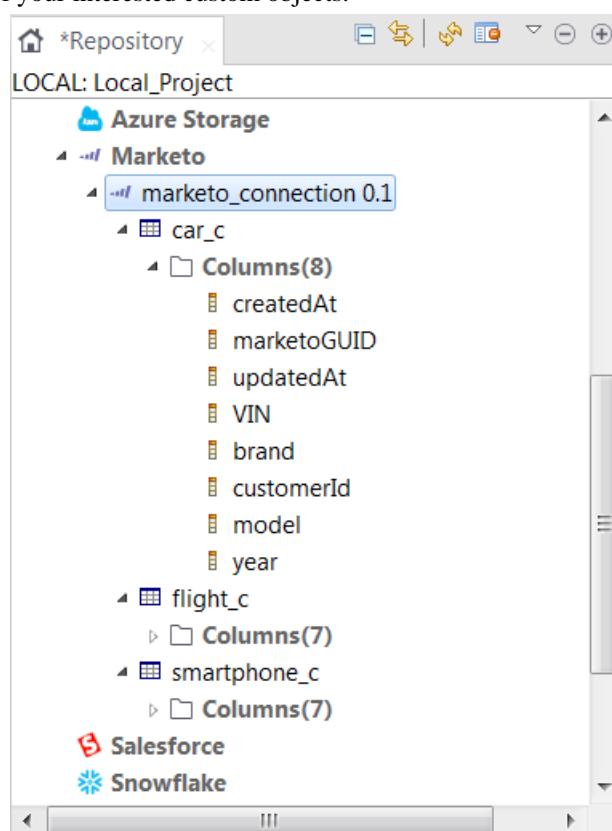
A connection successful dialog box will prompt up if the connection information provided is correct. Then click **OK** to close the dialog box. The **Next** button will be available to use.

4. Click **Next** to go to the next step to select your interested custom objects.



5. Select the custom objects whose schema you want to retrieve, and then click **Finish**.

The newly created Marketo connection is displayed under the **Marketo** node in the **Repository** tree view, along with the schema of your interested custom objects.



You can now add a Marketo component onto the design workspace by dragging and dropping the Marketo connection created or any custom object retrieved from the **Repository** view to reuse the connection and/or schema information. For more information about dropping component metadata in the design workspace, see [How to use centralized metadata in a Job](#). For more information about the usage of the Marketo components, see the related documentation for the Marketo components.

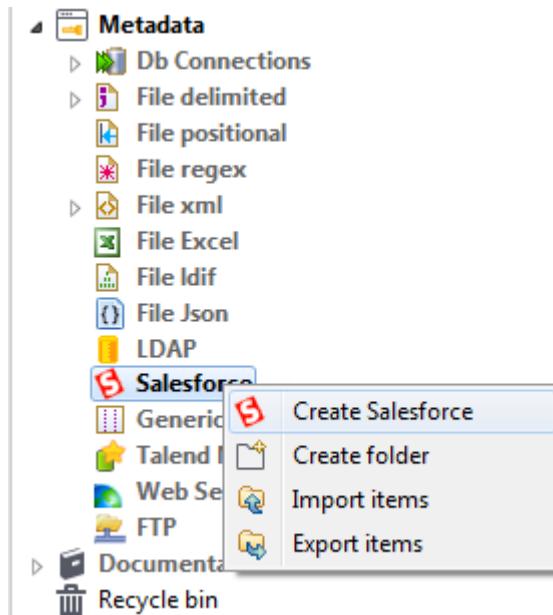
To modify the Marketo connection metadata created, right-click the connection node in the **Repository** tree view and select **Edit Marketo** from the contextual menu to open the metadata setup wizard.

To edit the schema of an interested custom object, right-click the custom object node in the **Repository** tree view and select **Edit Schema** from the contextual menu to open the update schema wizard.

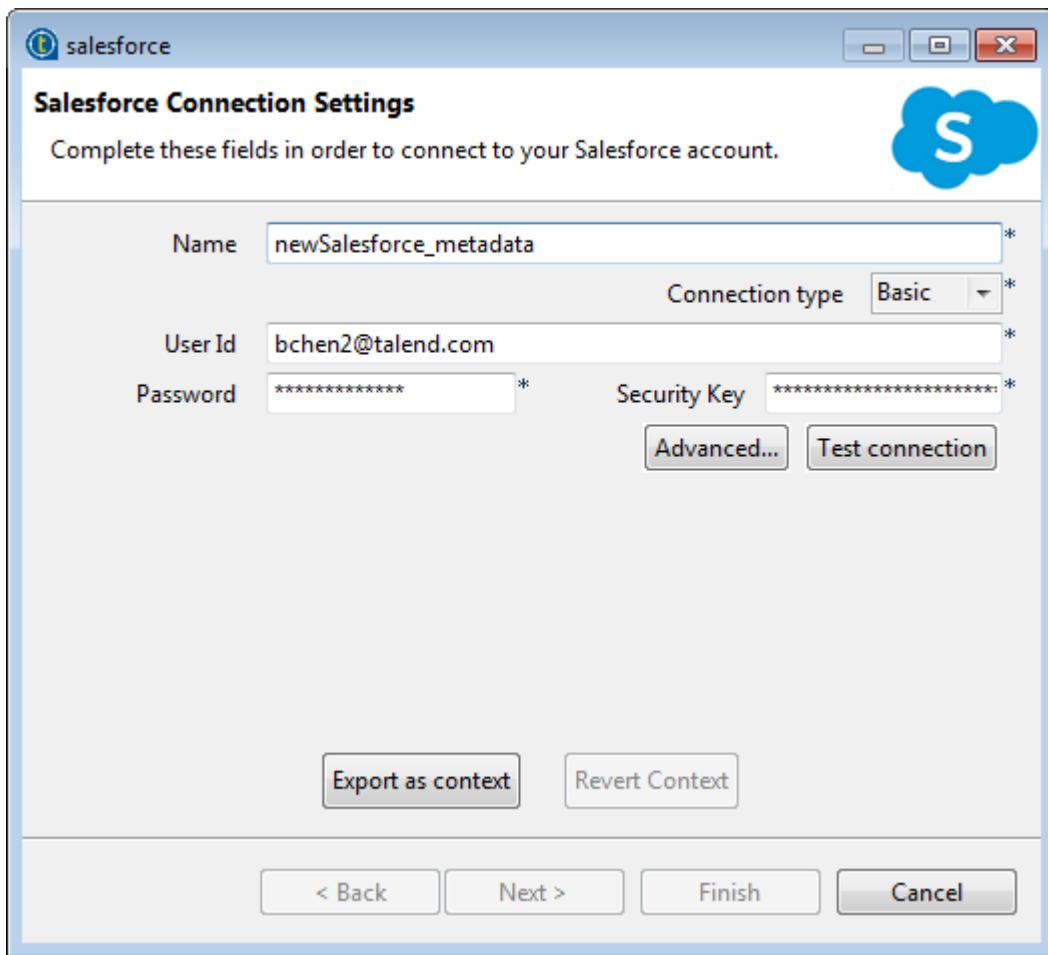
## 7.15. Centralizing Salesforce metadata

You can use the Salesforce metadata wizard provided by *Talend Studio* to set up quickly a connection to a Salesforce system so that you can reuse your Salesforce metadata across Jobs.

1. In the **Repository** tree view, expand the **Metadata** node, right-click the **Salesforce** tree node, and select **Create Salesforce** from the contextual menu to open the [Salesforce] wizard.



2. Enter a name for your connection in the **Name** field, select **Basic** or **OAuth** from the **Connection type** list, and provide the connection details according to the connection type you selected.



- With the **Basic** option selected, you need to specify the following details:
    - User Id:** the ID of the user in Salesforce.
    - Password:** the password associated with the user ID.
    - Security Key:** the security token.
  - With the **OAuth** option selected, you need to specify the following details:
    - Client Id and Client Secret:** the OAuth consumer key and consumer secret, which are available in the **OAuth Settings** area of the Connected App that you have created at Salesforce.com.
    - Callback Host and Callback Port:** the OAuth authentication callback URL. This URL (both host and port) is defined during the creation of a Connected App and will be shown in the **OAuth Settings** area of the Connected App.
    - Token File:** the path to the token file that stores the refresh token used to get the access token without authorization.
3. If needed, click **Advanced...** to open the [Salesforce Advanced Connection Settings] dialog box, do the following and then click **OK**:
- enter the Salesforce Webservice URL required to connect to the Salesforce system.
  - select the **Bulk Connection** check box if you need to use bulk data processing function.
  - select the **Need compression** check box to activate SOAP message compression, which can result in increased performance levels.

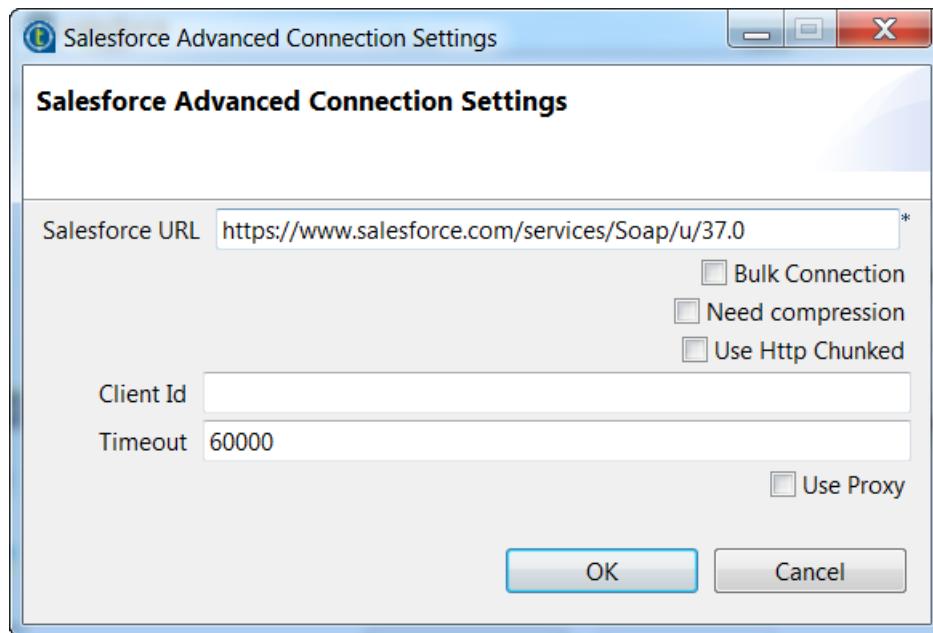
- select the **Trace HTTP message** check box to output the HTTP interactions on the console.

This option is available if the **Bulk Connection** check box is selected.

- select the **Use HTTP Chunked** check box to use the HTTP chunked data transfer mechanism.

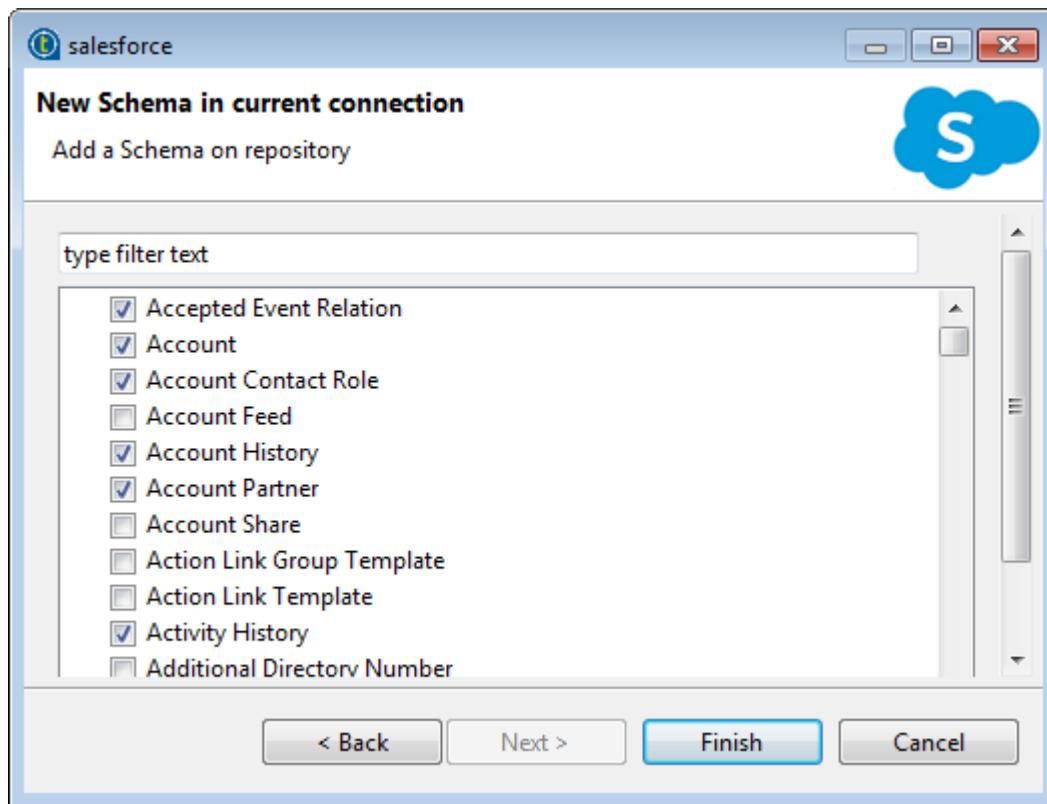
This option is not available if the **Bulk Connection** check box is selected.

- enter the ID of the real user in the **Client Id** field to differentiate between those who use the same account and password to access the Salesforce website.
- fill the **Timeout** field with the Salesforce connection timeout value, in milliseconds.

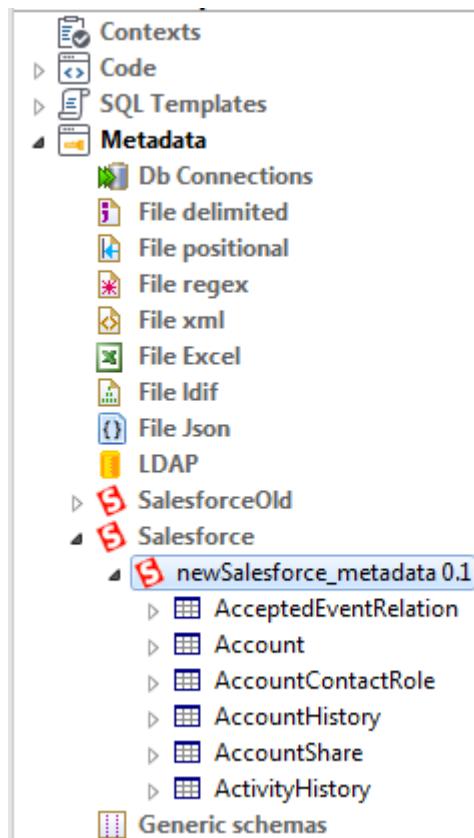


- Click **Test connection** to verify the connection settings, and when the connection check success message appears, click **OK** for confirmation. Then click **Next** to go to the next step to select the modules you want to retrieve the schema of.
- Select the check boxes for the modules of interest and click **Finish** to retrieve the schemas of the selected modules.

You can type in filter text to narrow down your selection.



The newly created Salesforce connection is displayed under the **Salesforce** node in the **Repository** tree view, along with the schemas of the selected modules.



You can now drag and drop the Salesforce connection or any schema of it from the **Repository** onto the design workspace, and from the dialog box that opens choose a Salesforce component to use in your Job. You can also

drop the Salesforce connection or a schema of it onto an existing component to reuse the connection or metadata details in the component. For more information about dropping component metadata in the design workspace, see [How to use centralized metadata in a Job](#).

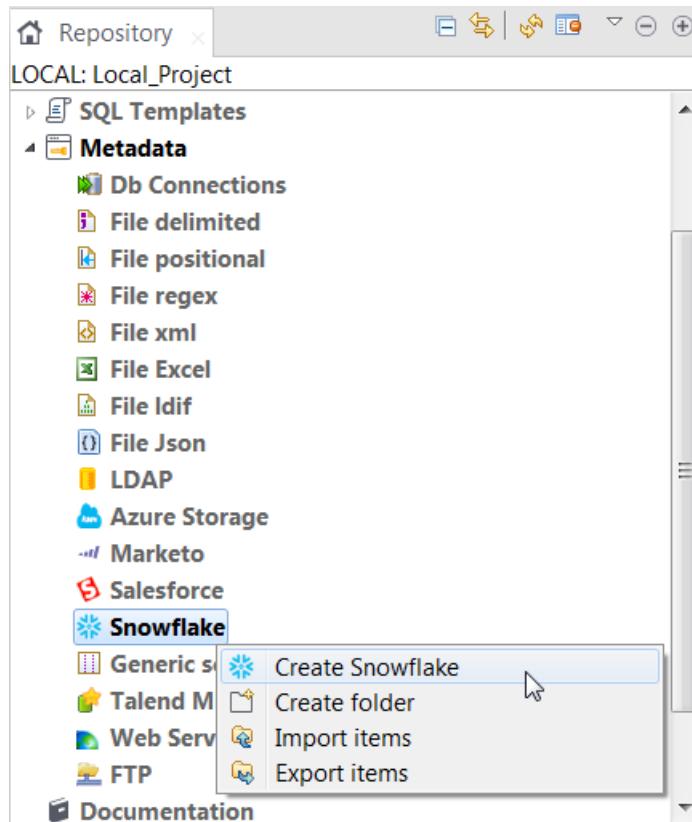
To modify the Salesforce metadata entry, right-click it from the **Repository** tree view, and select **Edit Salesforce** to open the file metadata setup wizard.

To edit an existing Salesforce schema, right-click the schema from the **Repository** tree view and select **Edit Schema** from the contextual menu.

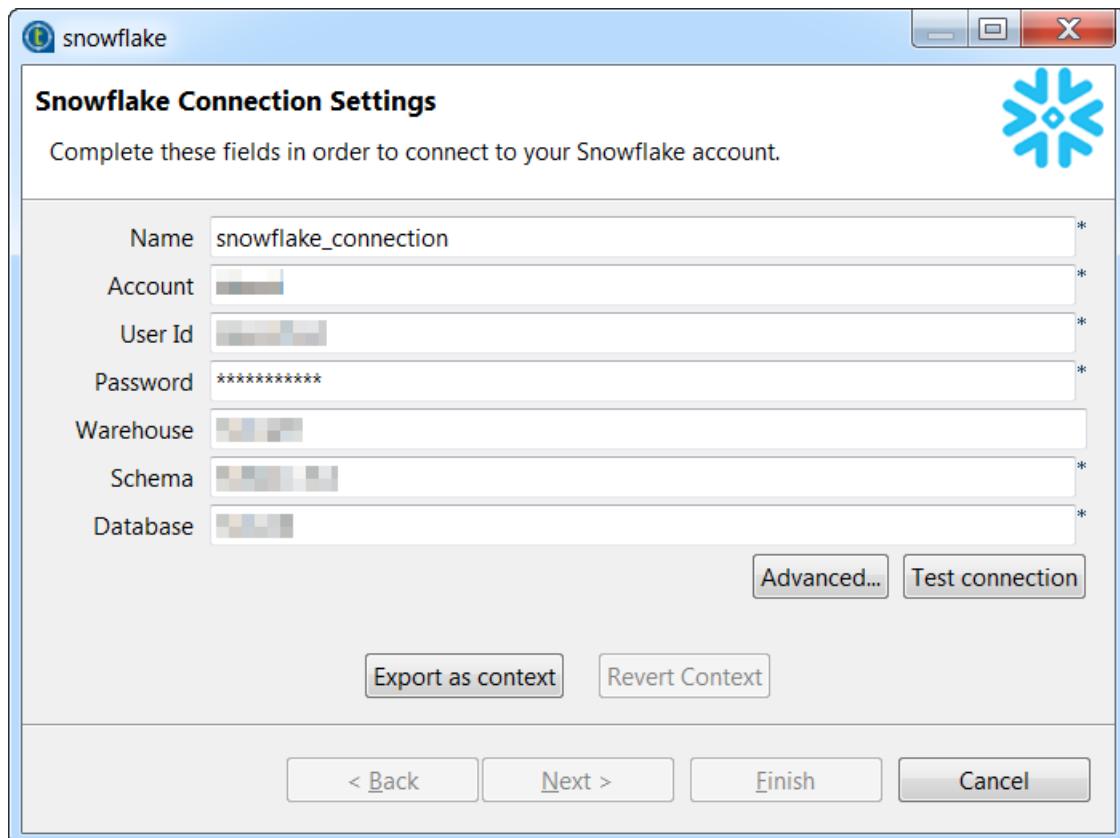
## 7.16. Centralizing Snowflake metadata

You can use the Snowflake metadata wizard provided by *Talend Studio* to set up quickly a connection to Snowflake and retrieve the schema of your interested tables.

1. In the **Repository** tree view, expand the **Metadata** node, right-click the **Snowflake** tree node, and select **Create Snowflake** from the contextual menu to open the **[Snowflake]** wizard.

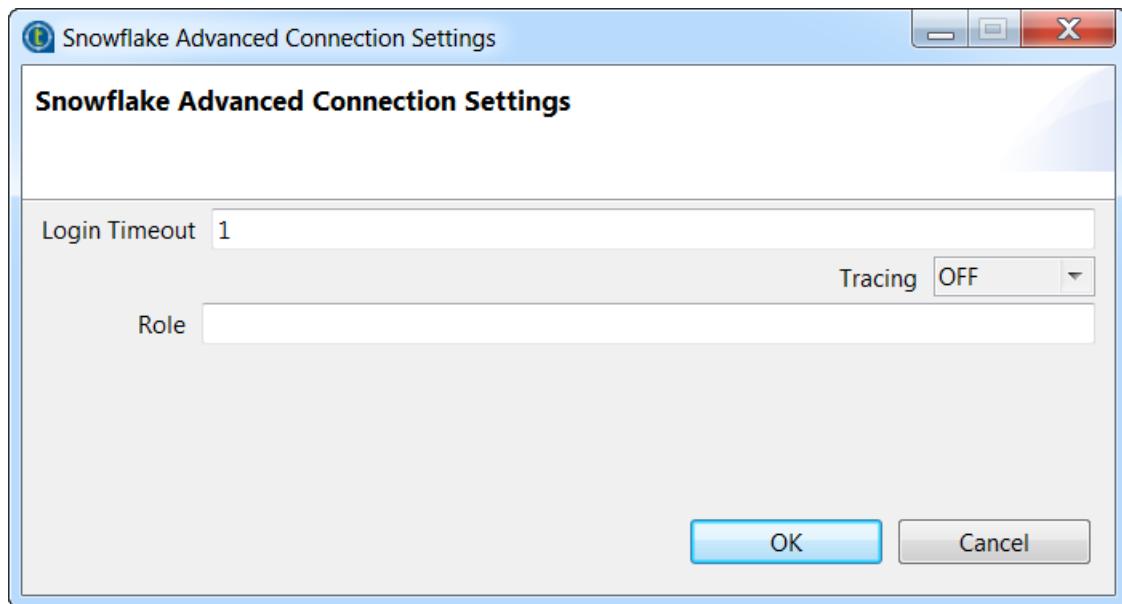


2. In the **Snowflake Connection Settings** dialog box, specify the values for the properties listed in the following table.



Property	Description
Name	Enter the name for the connection to be created.
Account	Enter the account name that has been assigned to you by Snowflake.
User Id	Enter your login name that has been defined in Snowflake using the LOGIN_NAME parameter of Snowflake. For details, ask the administrator of your Snowflake system.
Password	Enter the password associated with the user ID.
Warehouse	Enter the name of the Snowflake warehouse to be used. This name is case-sensitive and is normally upper case in Snowflake.
Schema	Enter the name of the database schema to be used. This name is case-sensitive and is normally upper case in Snowflake.
Database	Enter the name of the Snowflake database to be used. This name is case-sensitive and is normally upper case in Snowflake.

- Click **Advanced...** and in the **[Snowflake Advanced Connection Settings]** dialog box displayed, specify or update the values for the advanced properties listed in the following table and click **OK** to close the dialog box.

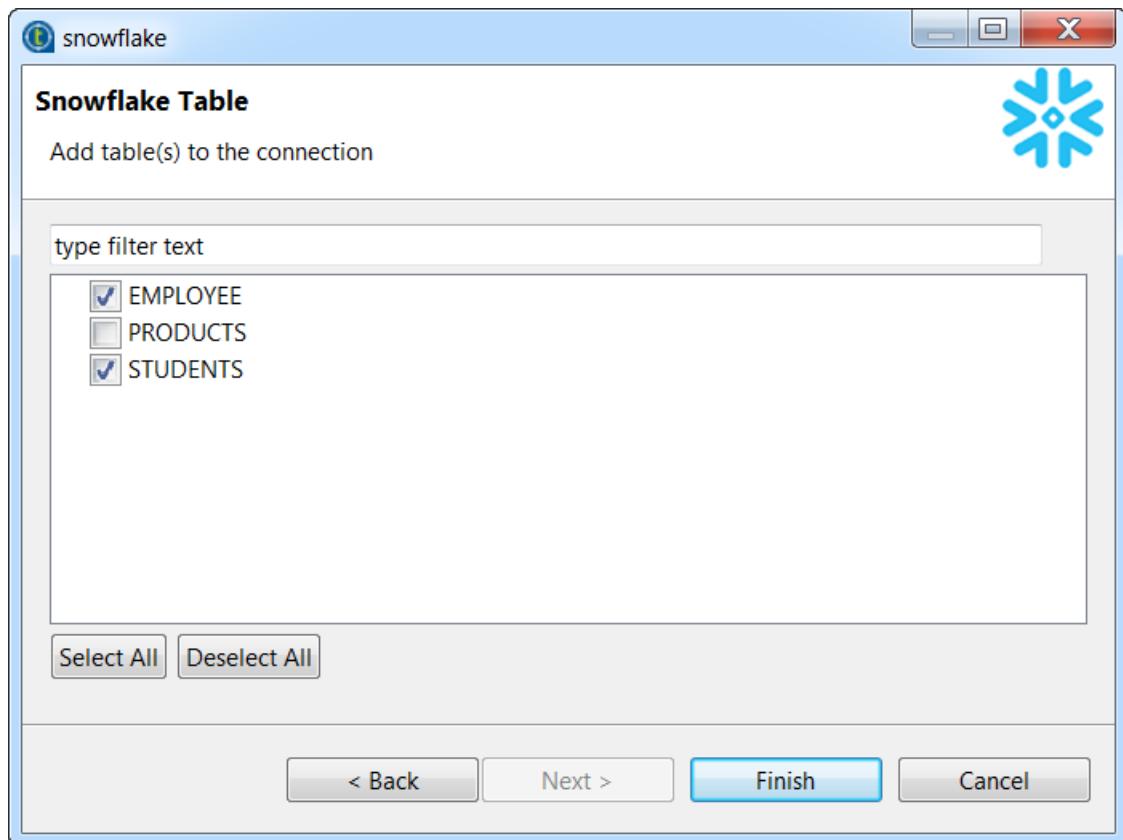


Property	Description
<b>Login Timeout</b>	Specify how long to wait for a response when connecting to Snowflake before returning an error.
<b>Tracing</b>	Select the log level for the Snowflake JDBC driver. If enabled, a standard Java log is generated.
<b>Role</b>	Enter the default access control role to use to initiate the Snowflake session.  This role must already exist and has been granted to the user ID you are using to connect to Snowflake. If this field is left empty, the PUBLIC role is automatically granted. For further information about the Snowflake access control model, see Snowflake documentation at <a href="#">Understanding the Access Control Model</a> .

- Click **Test connection** to verify the configuration.

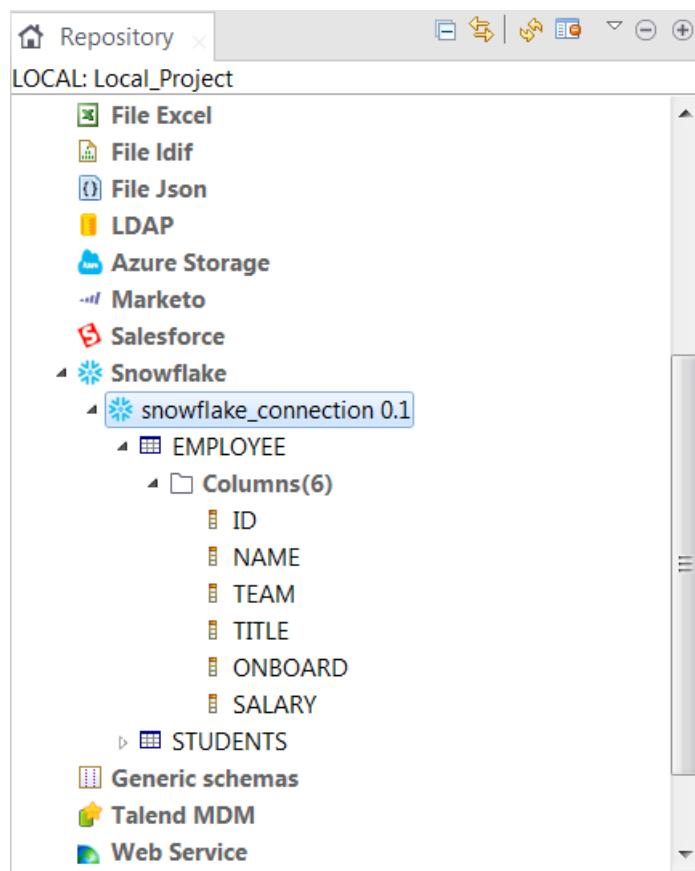
A connection successful dialog box will prompt up if the connection information provided is correct. Then click **OK** to close the dialog box. The **Next** button will be available to use.

- Click **Next** to go to the next step to select your interested tables.



6. Select the tables whose schema you want to retrieve, and then click **Finish**.

The newly created Snowflake connection is displayed under the **Snowflake** node in the **Repository** tree view, along with the schema of your interested tables.



You can now add a Snowflake component onto the design workspace by dragging and dropping the Snowflake connection created or any table retrieved from the **Repository** view to reuse the connection and/or schema information. For more information about dropping component metadata in the design workspace, see [How to use centralized metadata in a Job](#). For more information about the usage of the Snowflake components, see the related documentation for the Snowflake components.

To modify the Snowflake connection metadata created, right-click the connection node in the **Repository** tree view and select **Edit Snowflake** from the contextual menu to open the metadata setup wizard.

To edit the schema of an interested table, right-click the table node in the **Repository** tree view and select **Edit Schema** from the contextual menu to open the update schema wizard.

## 7.17. Setting up a generic schema

*Talend Studio* allows you to create a generic schema to use in your Jobs if none of the specific metadata wizards matches your need or if you do not have any source file to take the schema from.

You can create a generic schema:

- from scratch. For details, see [Setting up a generic schema from scratch](#),
- from a schema definition XML file. For details, see [Setting up a generic schema from an XML file](#), and
- from the schema defined in a component. For details, see [Saving a component schema as a generic schema](#).

To use a generic schema on a component, use either of the following methods:

- Select **Repository** from the **Schema** drop-down list in the component **Basic settings** view.

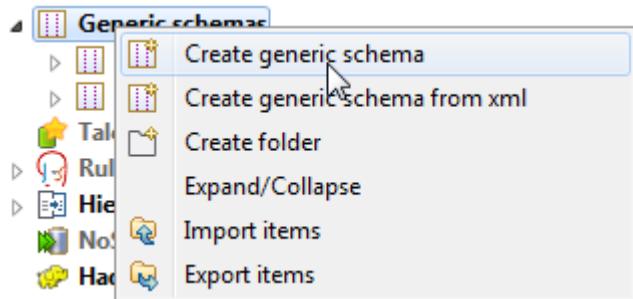
Click the [...] button to open the [Repository Content] dialog box, select the generic schema under the **Generic schemas** node and click **OK**.

- Select the **metadata** node of the generic schema from the **Repository** tree view and drop it onto the component.

## 7.17.1. Setting up a generic schema from scratch

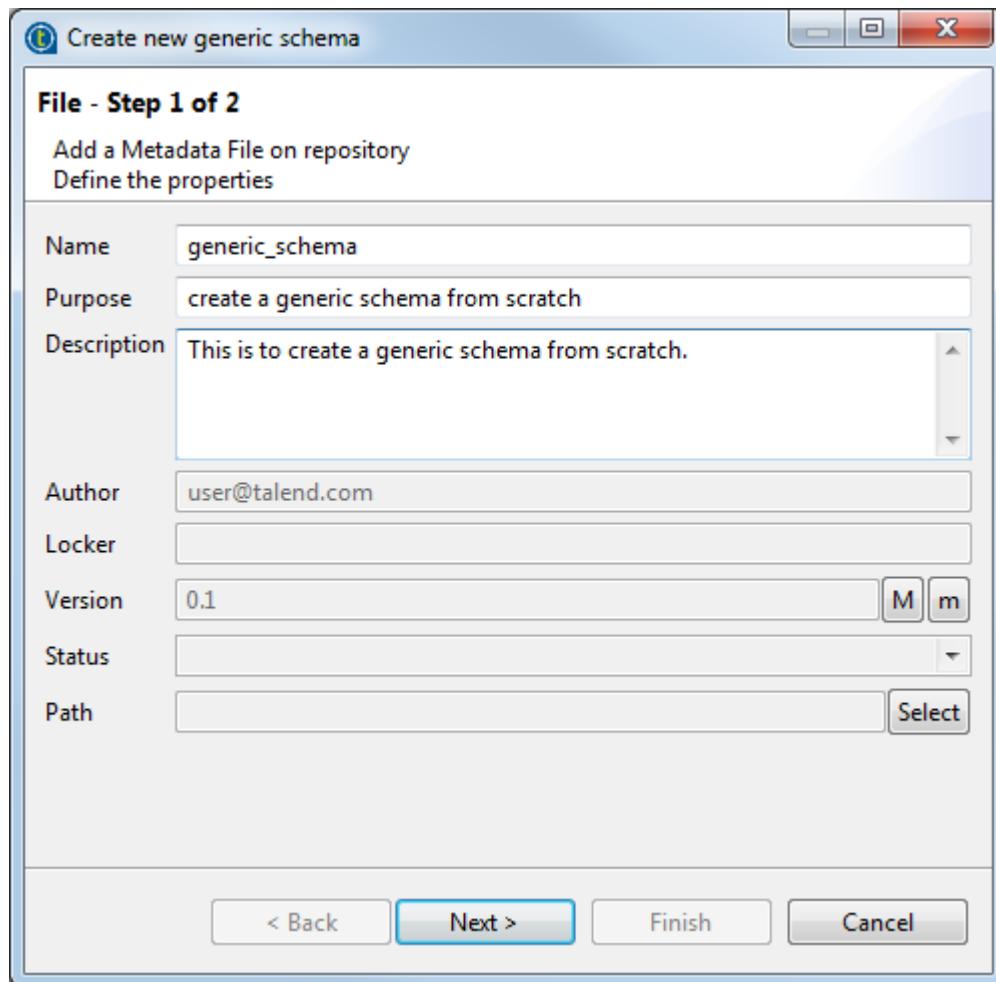
To create a generic schema from scratch, proceed as follows:

1. Right-click **Generic schemas** under the **Metadata** node in the **Repository** tree view, and select **Create generic schema**.



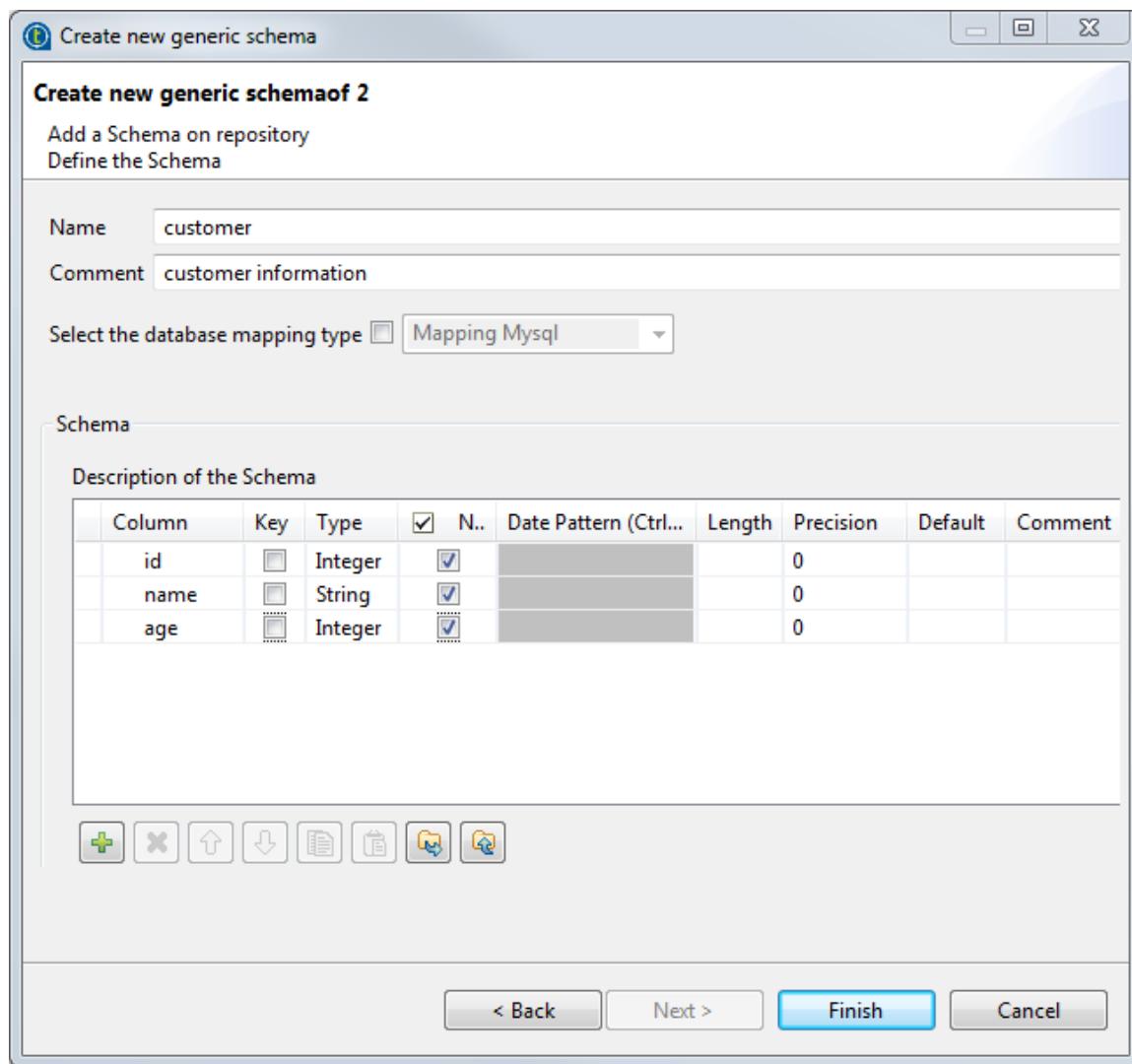
2. In the schema creation wizard that appears, fill in the generic schema properties such as schema **Name** and **Description**. The **Status** field is a customized field. For more information about how to define the field, see [Status settings](#).

Click **Next** to continue.



3. Give a name to the schema or use the default one (metadata) and add a comment if needed. Customize the schema structure in the **Schema** panel according to your needs.

The tool bar allows you to add, remove or move columns in your schema. You can also export the current schema as an XML file, or import a schema from an XML file, which must be an export of schema from the Studio, to replace the current schema.



Make sure the data type in the **Type** column is correctly defined.

For more information regarding Java data types, including date pattern, see [Java API Specification](#).

Below are the commonly used **Talend** data types:

- Object: a generic **Talend** data type that allows processing data without regard to its content, for example, a data file not otherwise supported can be processed with a **tFileInputRaw** component by specifying that it has a data type of Object.
  - List: a space-separated list of primitive type elements in an XML Schema definition, defined using the xsd:list element.
  - Document: a data type that allows processing an entire XML document without regarding to its content.
4. Click **Finish** to complete the generic schema creation. The created schema is displayed under the relevant **Generic schemas** node.

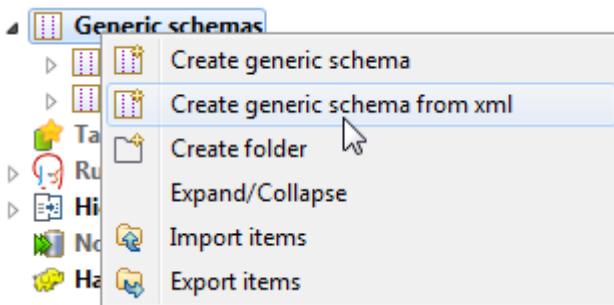
## 7.17.2. Setting up a generic schema from an XML file



*The source XML file from which you can create a generic schema must be an export of schema from the Studio or an XML with the same XML tree structure, not any other kind of XML.*

To create a generic schema from a source XML file, proceed as follows:

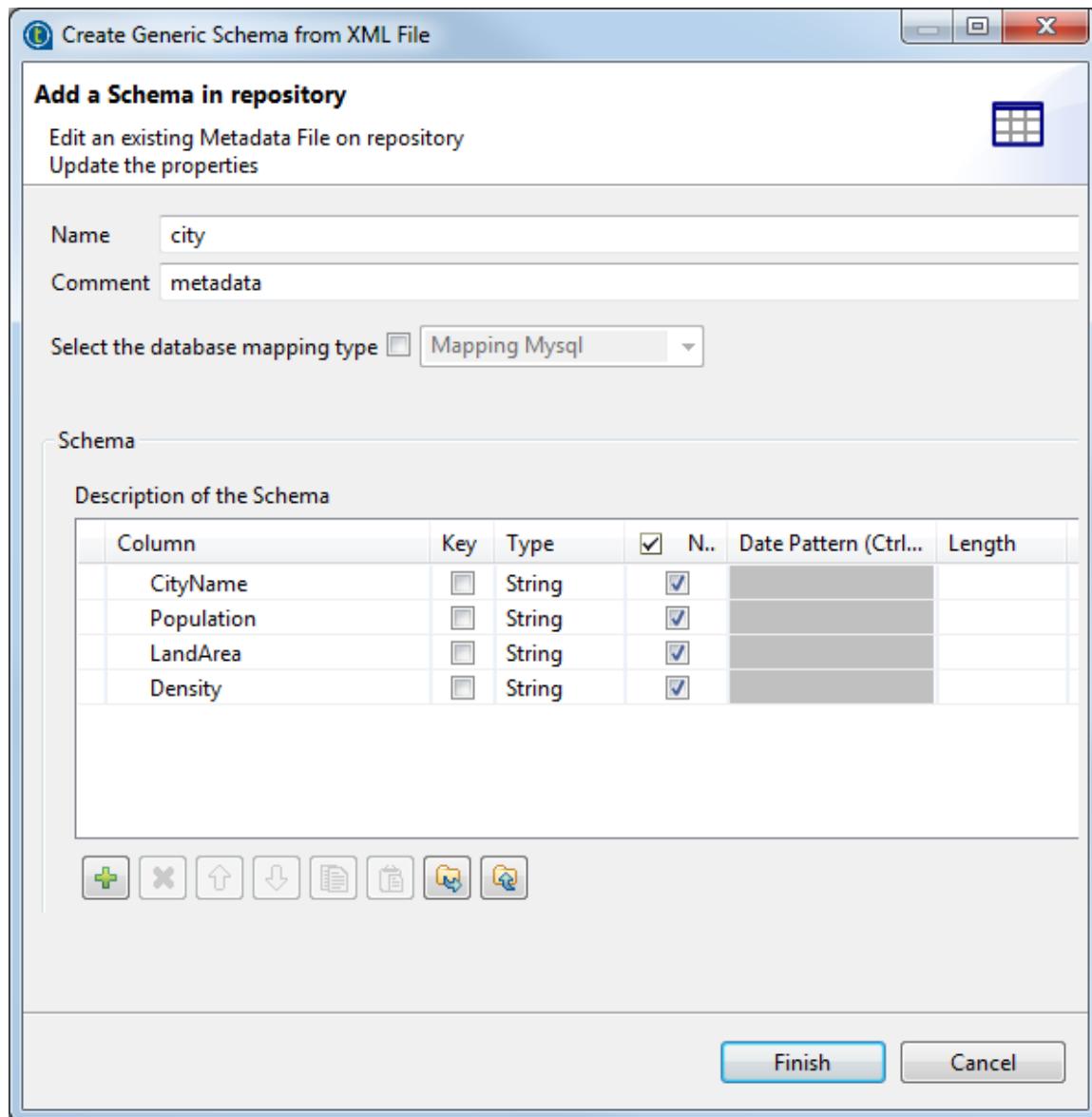
1. Right-click **Generic schemas** in the **Repository** tree view, and select **Create generic schema from xml**.



2. In the dialog box that appears, choose the source XML file from which the schema is taken and click **Open**.
3. In the schema creation wizard that appears, define the schema **Name** or use the default one (metadata) and give a **Comment** if any.

The schema structure from the source file is displayed in the **Schema** panel. You can customize the columns in the schema as needed.

The tool bar allows you to add, remove or move columns in your schema. You can also export the current schema as an XML file, or import a schema from an XML file, which must be an export of schema from the Studio, to replace the current schema.



Make sure the data type in the **Type** column is correctly defined.

For more information regarding Java data types, including date pattern, see [Java API Specification](#).

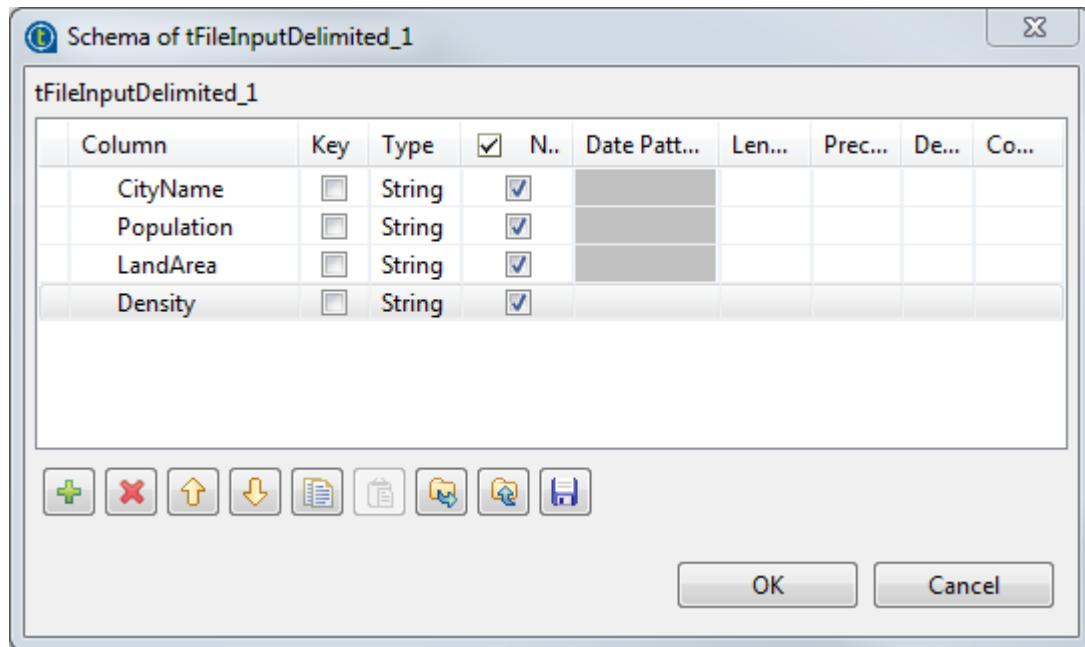
Below are the commonly used **Talend** data types:

- Object: a generic **Talend** data type that allows processing data without regard to its content, for example, a data file not otherwise supported can be processed with a **tFileInputRaw** component by specifying that it has a data type of Object.
  - List: a space-separated list of primitive type elements in an XML Schema definition, defined using the xsd:list element.
  - Document: a data type that allows processing an entire XML document without regarding to its content.
4. Click **Finish** to complete the generic schema creation. The created schema is displayed under the relevant **Generic schemas** node.

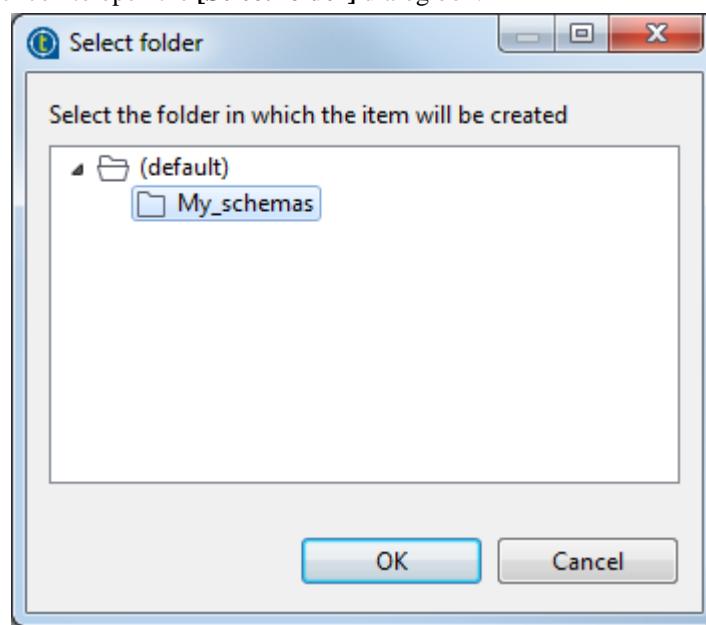
### 7.17.3. Saving a component schema as a generic schema

You can create a generic schema by saving the schema defined in a component. To do so, follow the steps below:

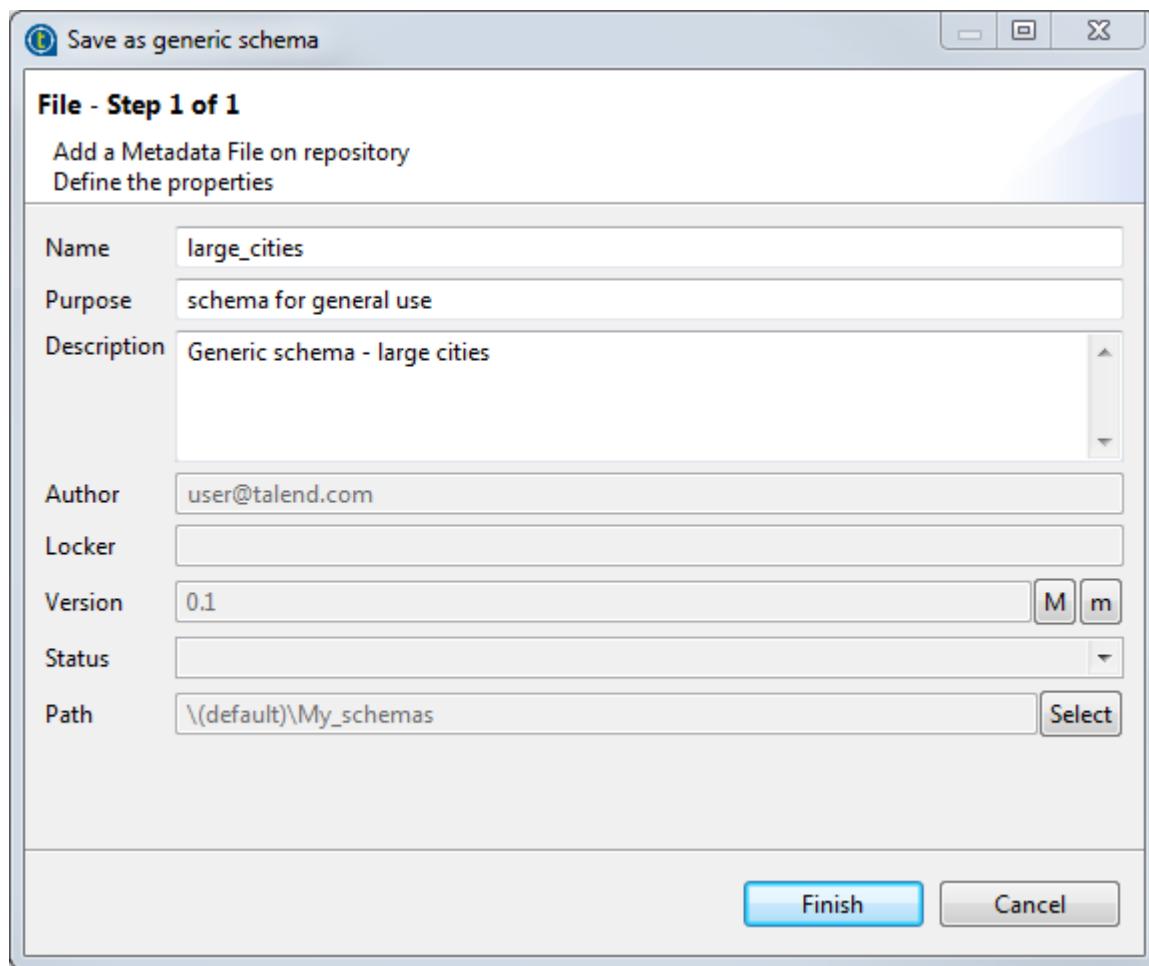
1. Open the **Basic settings** view of the component that has the schema you want to create a generic schema from, and click the [...] button next to **Edit schema** to open the **[Schema]** dialog box.



2. Click the floppy disc icon to open the **[Select folder]** dialog box.



3. Select a folder if needed, and click **OK** to close the dialog box and open the **[Save as generic schema]** creation wizard.



- Fill in the **Name** field (required) and the other fields if needed, and click **Finish** to save the schema. Then close the **[Schema]** dialog box opened from the component **Basic settings** view.

The schema is saved in the selected folder under the **Generic schemas** node in the **Repository** tree view.

## 7.18. Centralizing MDM metadata

*Talend Studio* enables you to centralize the details of one or more MDM connections under the **Metadata** folder in the **Repository** tree view. You can then use any of these established connections to connect to the MDM server.



You can also set up an MDM connection the same way by clicking the  icon in the **Basic settings** view of the **tMDMInput** and **tMDMOutput** components. For more information, see **tMDMInput** and **tMDMOutput** at <https://help.talend.com>.

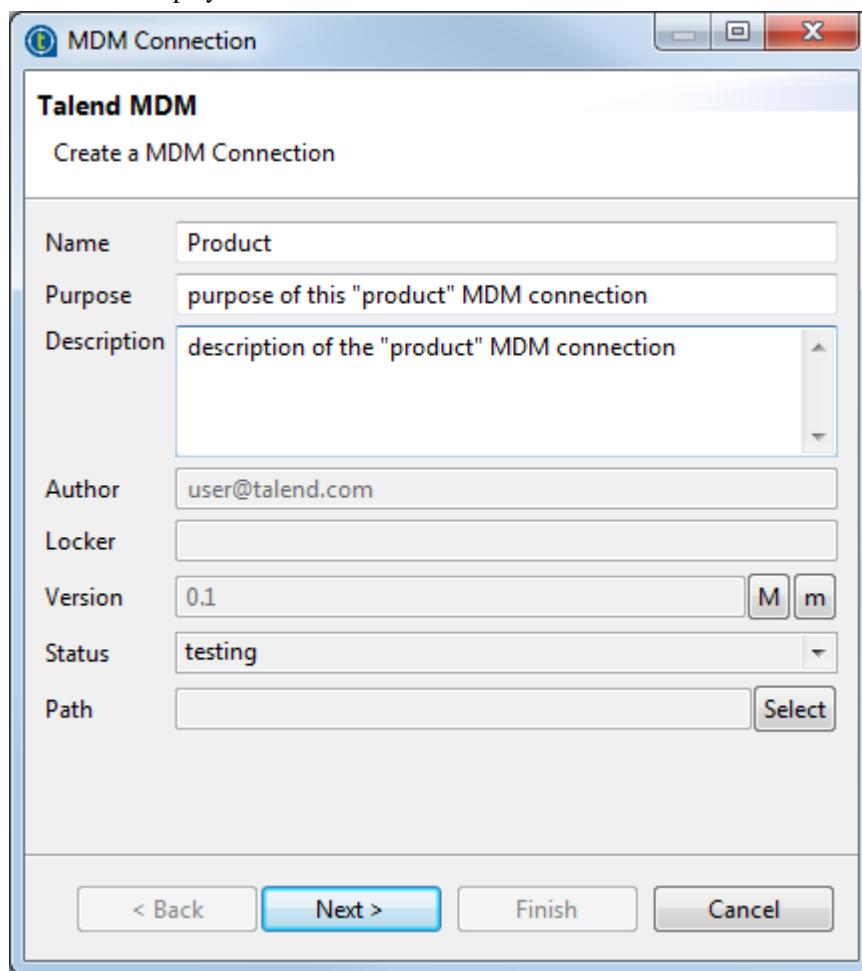
According to the option you select, the wizard helps you create an input XML, an output XML or a receive XML schema. Later, in a **Talend** Job, the **tMDMInput** component uses the defined input schema to read master data stored in XML documents, **tMDMOutput** uses the defined output schema to either write master data in an XML document on the MDM server, or to update existing XML documents and finally the **tMDMReceive** component uses the defined XML schema to receive an MDM record in XML from MDM triggers and processes.

### 7.18.1. Setting up the connection

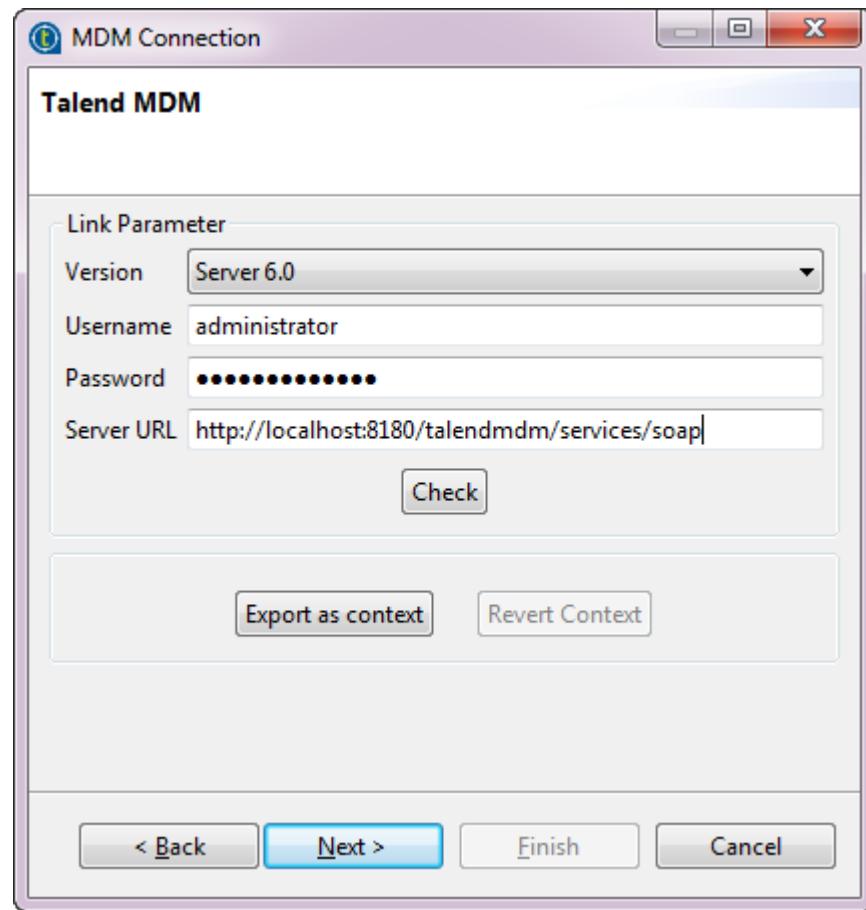
To establish an MDM connection, complete the following:

1. In the **Repository** tree view, expand **Metadata** and right-click **Talend MDM**.
2. Select **Create MDM Connection** from the contextual menu.

The connection wizard is displayed.



3. Fill in the connection properties such as **Name**, **Purpose** and **Description**. The **Status** field is a customized field that can be defined. For more information, see [Status settings](#).
4. Click **Next** to proceed to the next step.



- From the **Version** list, select the version of the MDM server to which you want to connect.



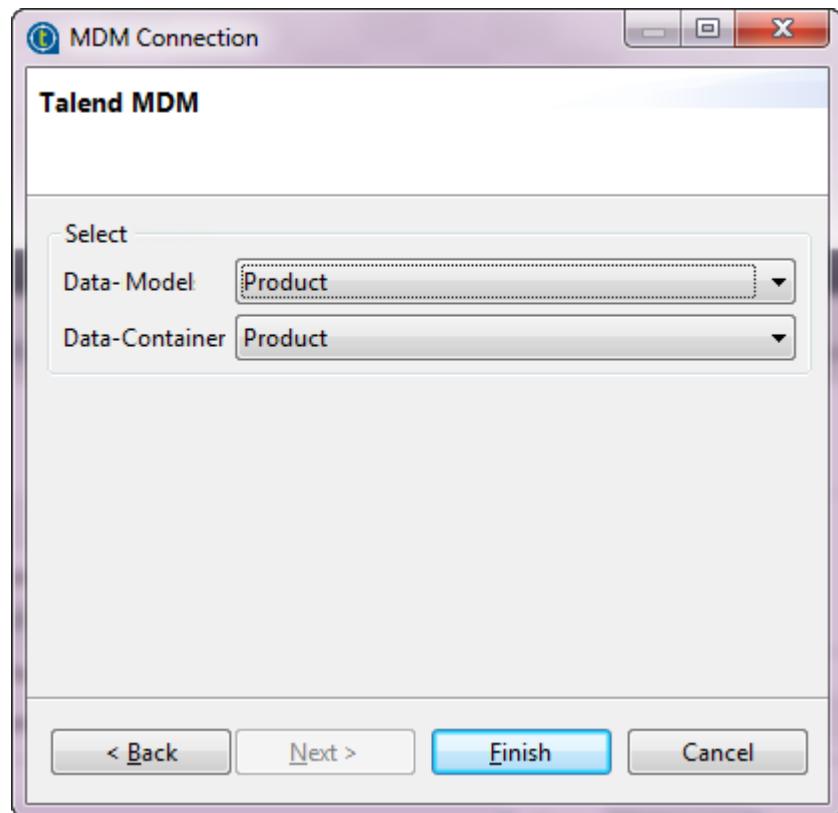
The default value in the **Server URL** field varies depending on what you selected in the **Version** list.

- Fill in the connection details including the authentication information to the MDM server and then click **Check** to check the connection you have created.

A dialog box pops up to show that your connection is successful. Click **OK** to close it.

If needed, you can click **Export as context** to export this **Talend MDM** connection details to a new context group in the Repository or reuse variables of an existing context group to set up your metadata connection. For more information, see [Exporting metadata as context and reusing context parameters to set up a connection](#).

- Click **Next** to proceed to the next step.



8. From the **Data-Model** list, select the data model against which the master data is validated.
9. From the **Data-Container** list, select the data container that holds the master data you want to access.
10. Click **Finish** to validate your changes and close the dialog box.

The newly created connection is listed under **Talend MDM** under the **Metadata** folder in the **Repository** tree view.

You need now to retrieve the XML schema of the business entities linked to this MDM connection.

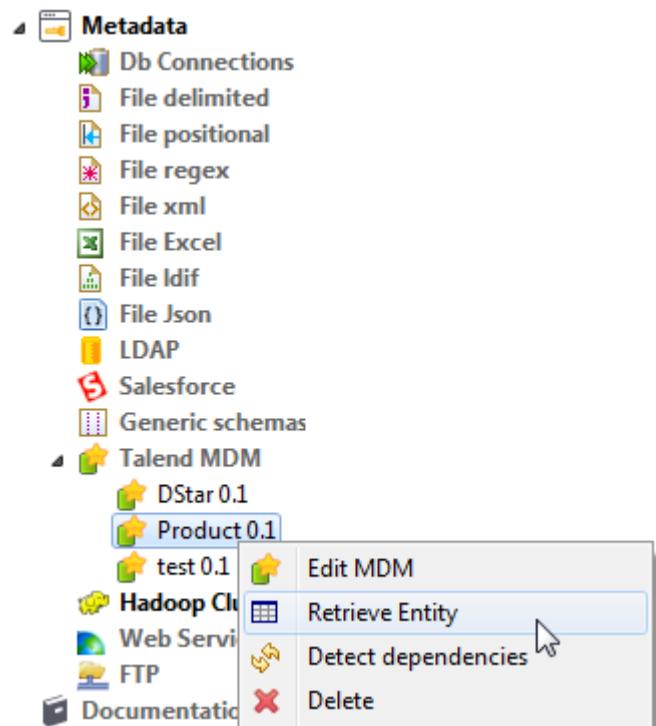
## 7.18.2. Defining MDM schema

### 7.18.2.1. Defining Input MDM schema

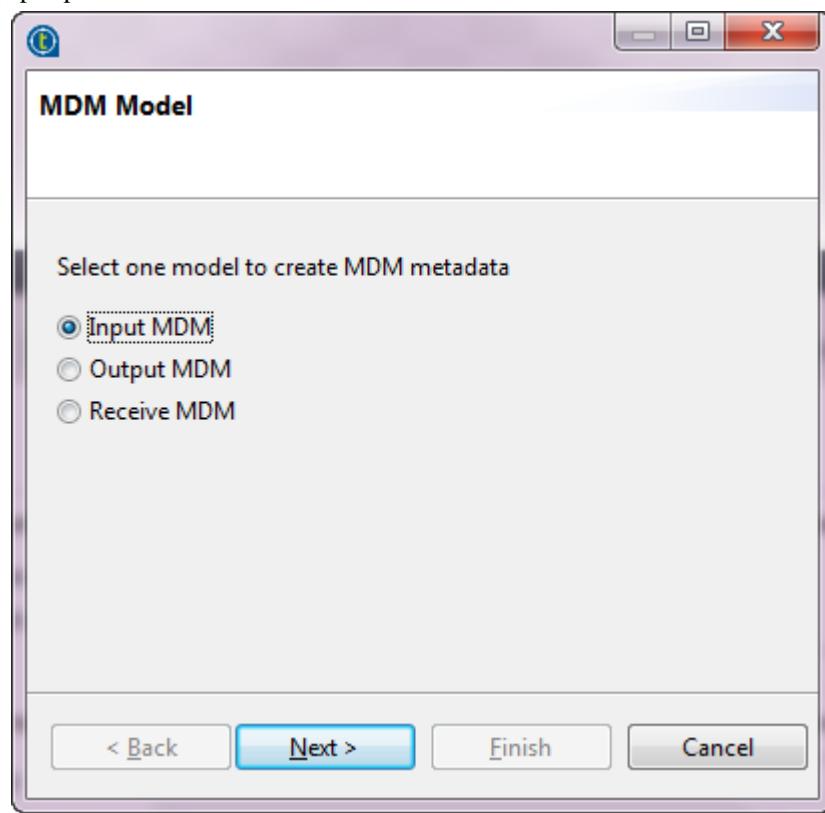
This section describes how to define and download an input MDM XML schema. To define and download an output MDM XML schema, see [Defining output MDM schema](#).

To set the values to be fetched from one or more entities linked to a specific MDM connection, complete the following:

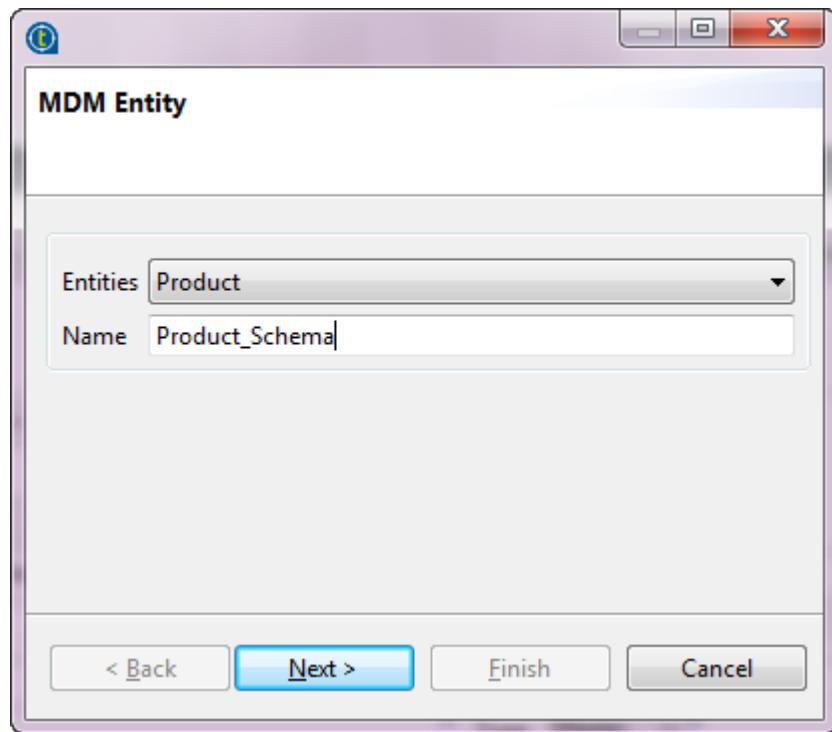
1. In the **Repository** tree view, expand **Metadata** and right-click the MDM connection for which you want to retrieve the entity values.
2. Select **Retrieve Entity** from the contextual menu.



A dialog box pops up.



3. Select the **Input MDM** option in order to download an input XML schema and then click **Next** to proceed to the following step.



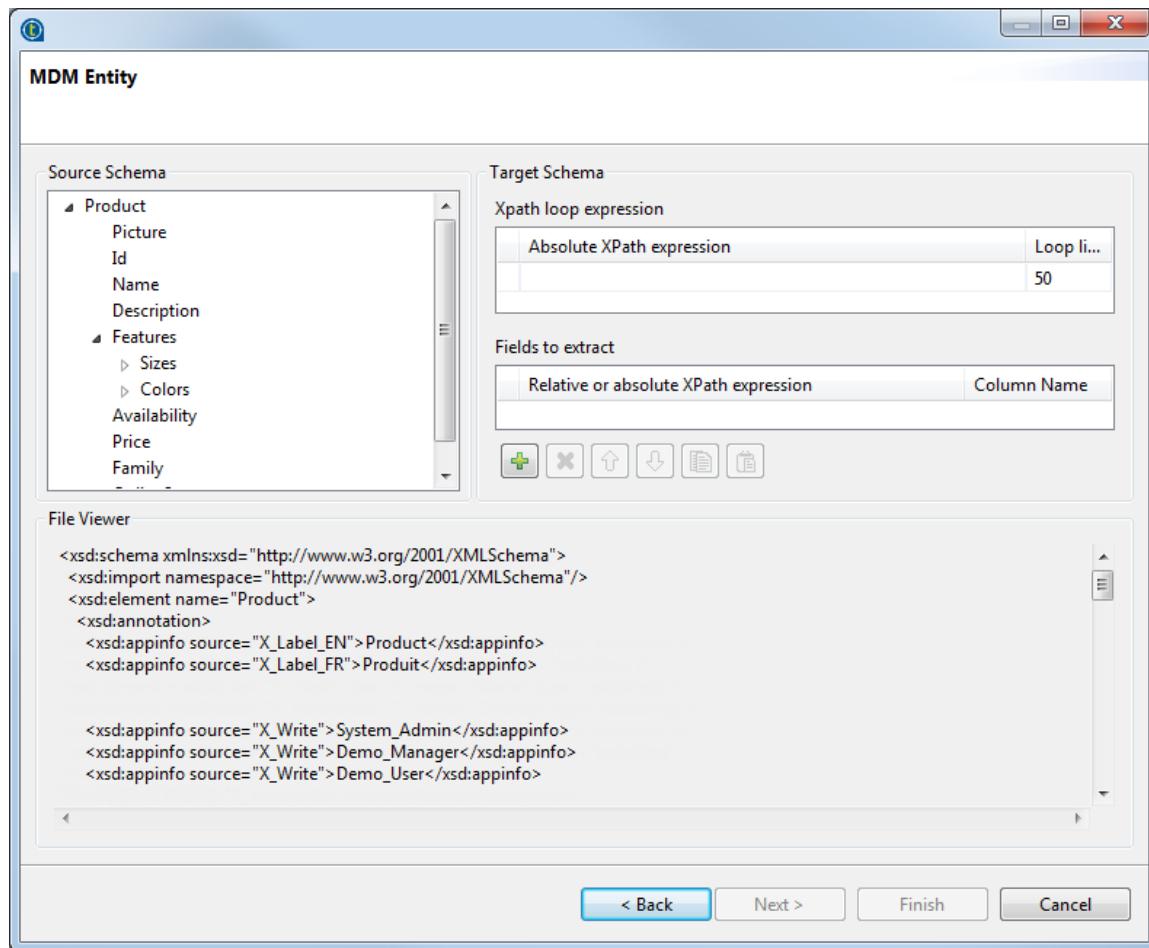
4. From the **Entities** field, select the business entity (XML schema) from which you want to retrieve values.

The name is displayed automatically in the **Name** field.



You are free to enter any text in this field, although you would likely put the name of the entity from which you are retrieving the schema.

5. Click **Next** to proceed to the next step.



The schema of the entity you selected is automatically displayed in the **Source Schema** panel.

Here, you can set the parameters to be taken into account for the XML schema definition.

The schema dialog box is divided into four different panels as the following:

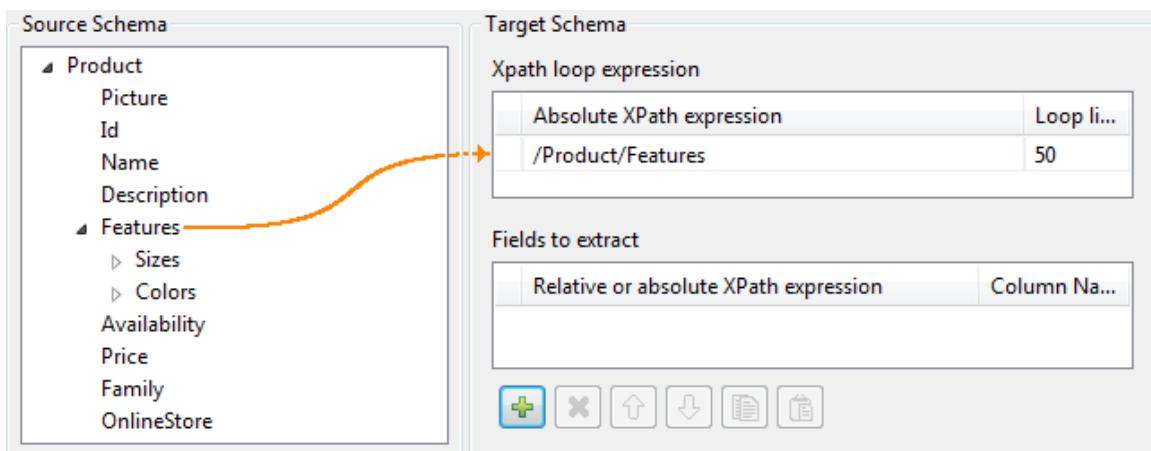
Panel	Description
Source Schema	Tree view of the uploaded entity.
Target schema	Extraction and iteration information.
Preview	Target schema preview.
File viewer	Raw data viewer.

6. In the **Xpath loop expression** area, enter the absolute XPath expression leading to the XML structure node on which to apply the iteration. Or, drop the node from the source schema to the target schema Xpath field. This link is orange in color.



The **Xpath loop expression** field is compulsory.

7. If required, define a **Loop limit** to restrict the iteration to a number of nodes.

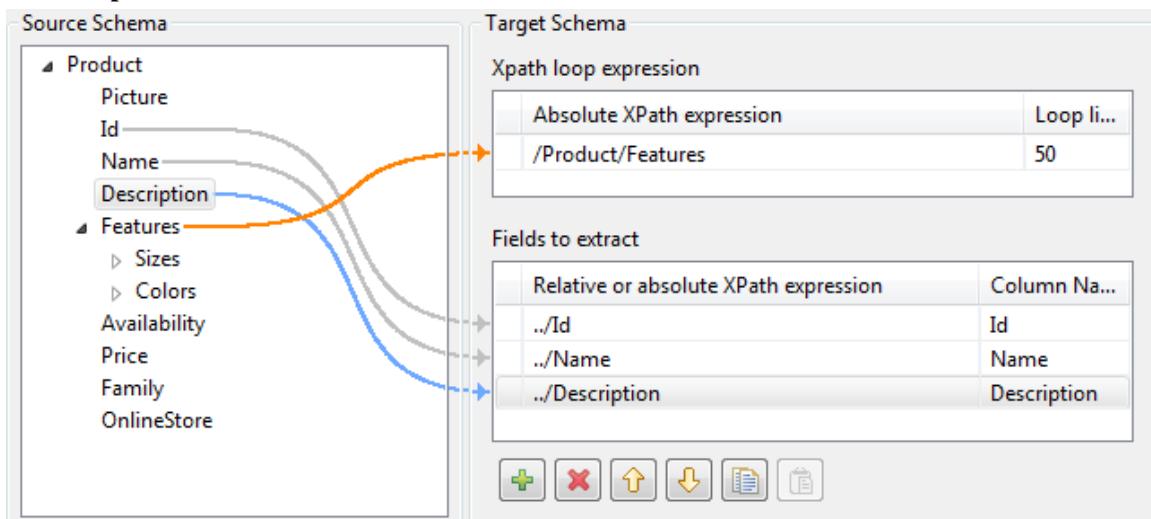


In the capture above, we use *Features* as the element to loop on because it is repeated within the *Product* entity as follows:

```
<Product>
<Id>1</Id>
<Name>Cup</Name>
<Description/>
<Features>
 <Feature>Color red</Feature>
 <Feature>Size maxi</Feature>
<Features>
...
</Product>
<Product>
<Id>2</Id>
<Name>Cup</Name>
<Description/>
<Features>
 <Feature>Color blue</Feature>
 <Feature>Thermos</Feature>
<Features>
...
</Product>
```

By doing so, the **tMDMInput** component that uses this MDM connection will create a new row for every item with different feature.

- To define the fields to extract, drop the relevant node from the source schema to the **Relative or absolute XPath expression** field.

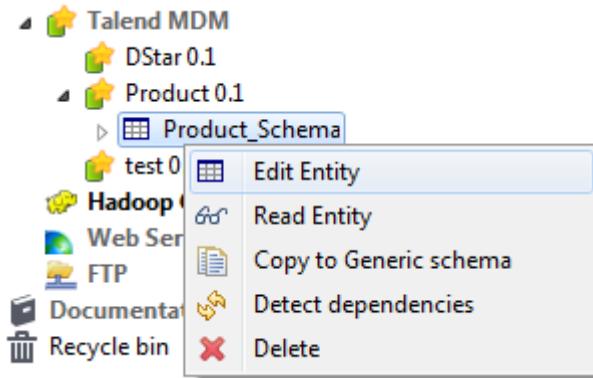




Use the [+] button to add rows to the table and select as many fields to extract as necessary. Press the **Ctrl** or the **Shift** keys for multiple selection of grouped or separate nodes and drop them to the table.

9. If required, enter a name to each of the retrieved columns in the **Column name** field.
- You can prioritize the order of the fields to extract by selecting the field and using the up and down arrows. The link of the selected field is blue, and all other links are grey.
10. Click **Finish** to validate your modifications and close the dialog box.

The newly created schema is listed under the corresponding MDM connection in the **Repository** tree view.



To modify the created schema, complete the following:

1. In the **Repository** tree view, expand **Metadata** and **Talend MDM** and then browse to the schema you want to modify.
2. Right-click the schema name and select **Edit Entity** from the contextual menu.

A dialog box is displayed.

Column	Key	Type	N.	Date Pattern (Ctrl...)	Length	Precision	Default	Comment
Id	<input checked="" type="checkbox"/>	String	<input checked="" type="checkbox"/>		0			
Name	<input type="checkbox"/>	String	<input checked="" type="checkbox"/>		0			
Description	<input type="checkbox"/>	String	<input checked="" type="checkbox"/>		0			

3. Modify the schema as needed.

You can change the name of the schema according to your needs, you can also customize the schema structure in the schema panel. The tool bar allows you to add, remove or move columns in your schema.

Make sure the data type in the **Type** column is correctly defined.

For more information regarding Java data types, including date pattern, see [Java API Specification](#).

Below are the commonly used **Talend** data types:

- Object: a generic **Talend** data type that allows processing data without regard to its content, for example, a data file not otherwise supported can be processed with a **tFileInputRaw** component by specifying that it has a data type of Object.
  - List: a space-separated list of primitive type elements in an XML Schema definition, defined using the xsd:list element.
  - Document: a data type that allows processing an entire XML document without regarding to its content.
4. Click **Finish** to close the dialog box.

The MDM input connection (**tMDMInput**) is now ready to be dropped in any of your Jobs.

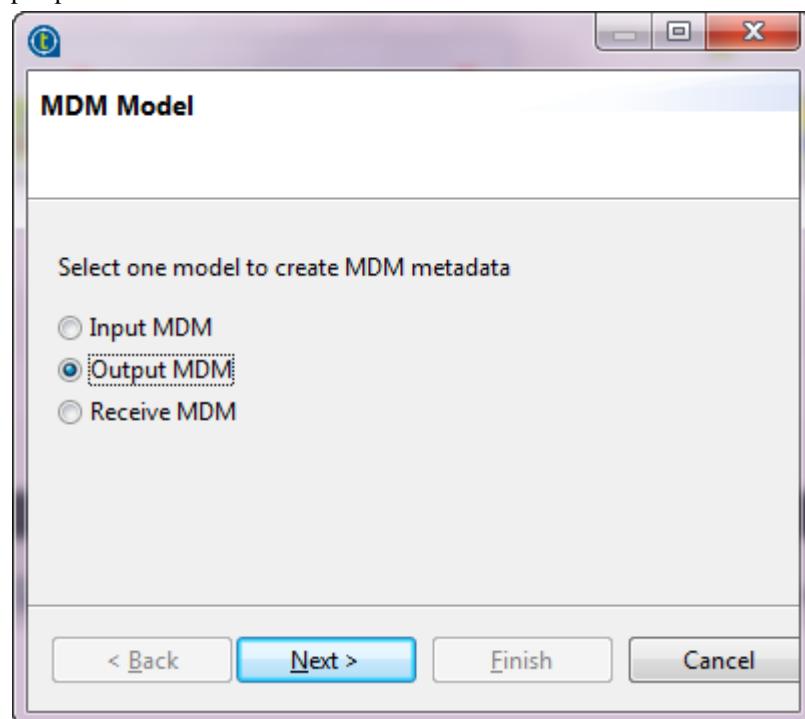
## 7.18.2.2. Defining output MDM schema

This section describes how to define and download an output MDM XML schema. To define and download an input MDM XML schema, see [Setting up the connection](#).

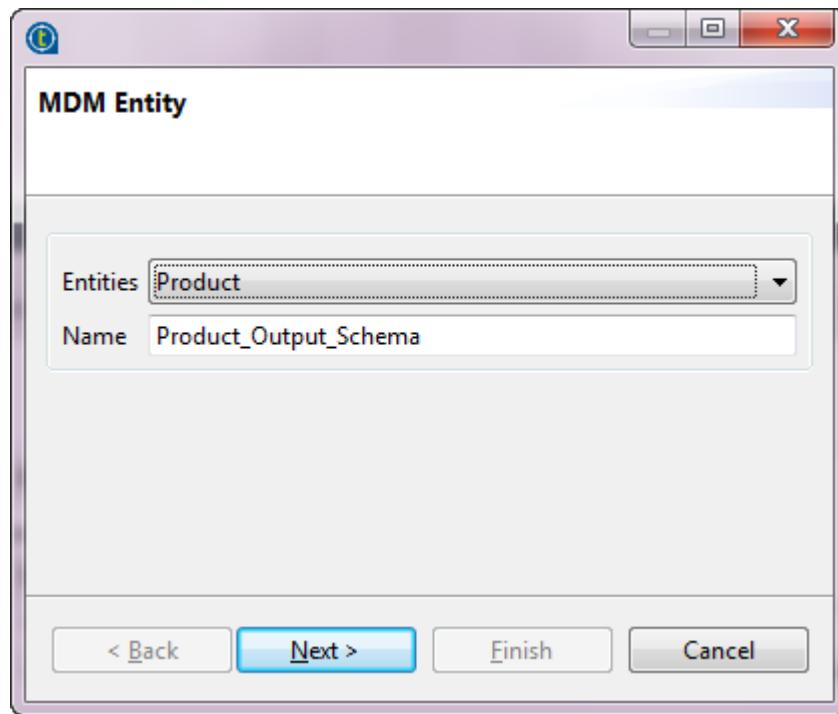
To set the values to be written in one or more entities linked to a specific MDM connection, complete the following:

1. In the **Repository** tree view, expand **Metadata** and right-click the MDM connection for which you want to write the entity values.
2. Select **Retrieve Entity** from the contextual menu.

A dialog box pops up.



3. Select the **Output MDM** option in order to define an output XML schema and then click **Next** to proceed to the following step.



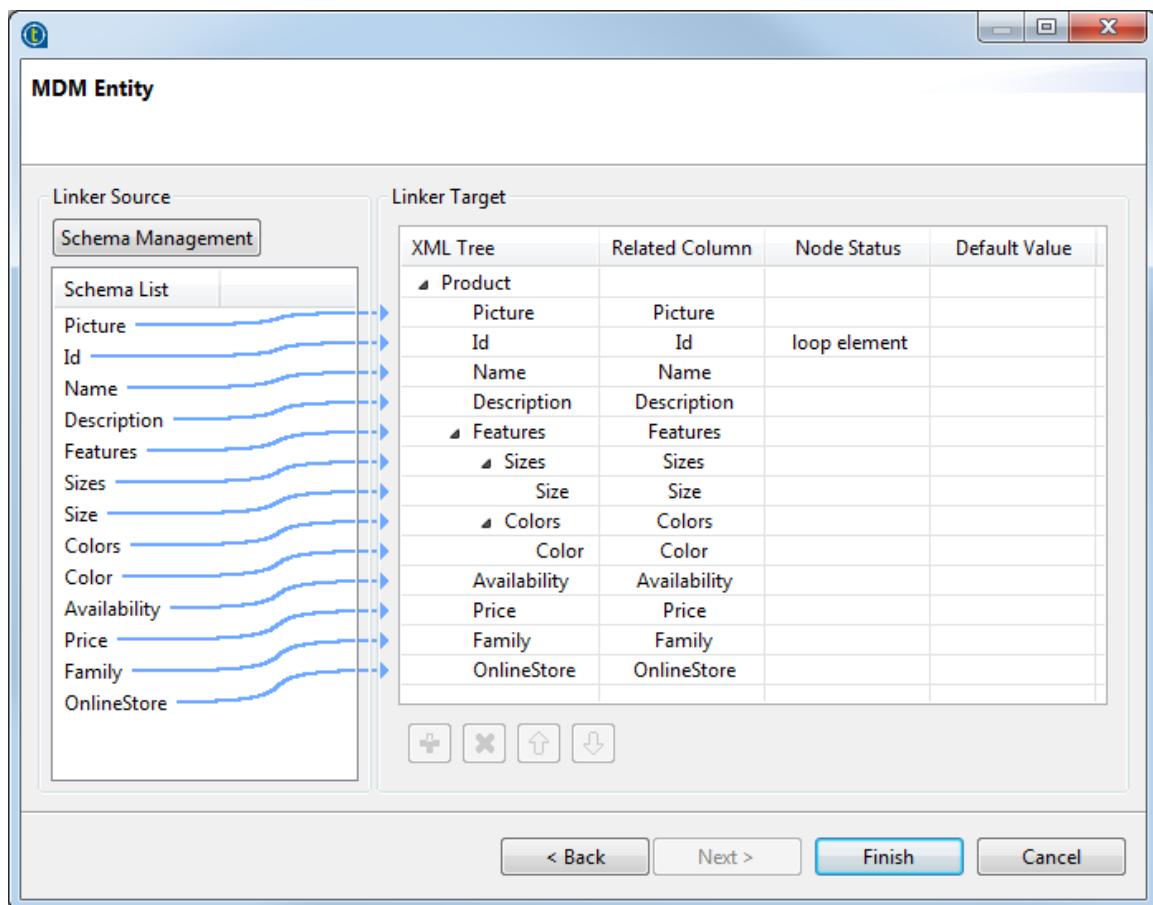
4. From the **Entities** field, select the business entity (XML schema) in which you want to write values.

The name is displayed automatically in the **Name** field.



You are free to enter any text in this field, although you would likely put the name of the entity from which you are retrieving the schema.

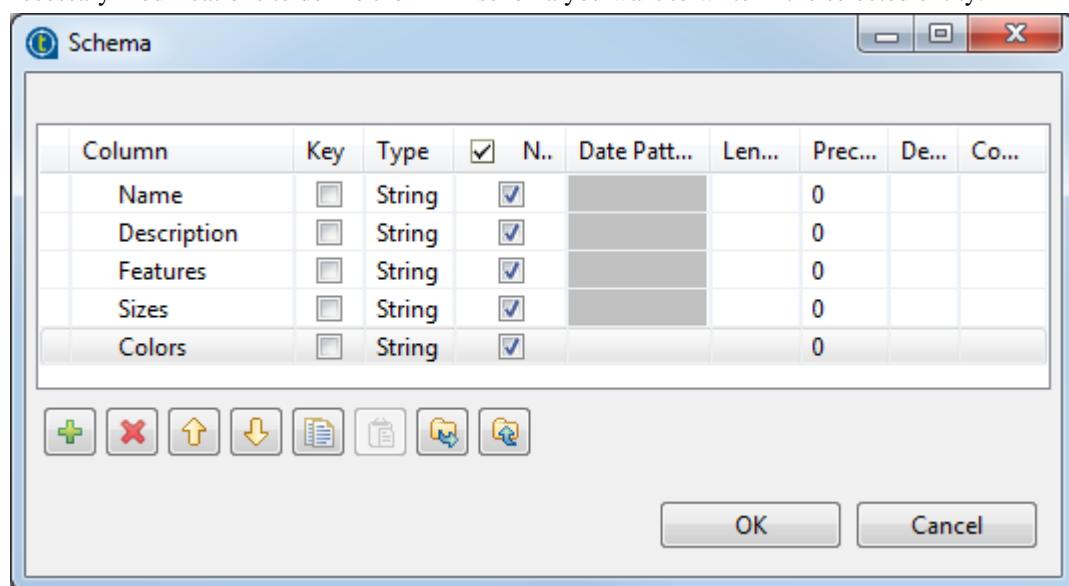
5. Click **Next** to proceed to the next step.



Identical schema of the entity you selected is automatically created in the **Linker Target** panel, and columns are automatically mapped from the source to the target panels. The wizard automatically defines the item Id as the looping element. You can always select to loop on another element.

Here, you can set the parameters to be taken into account for the XML schema definition.

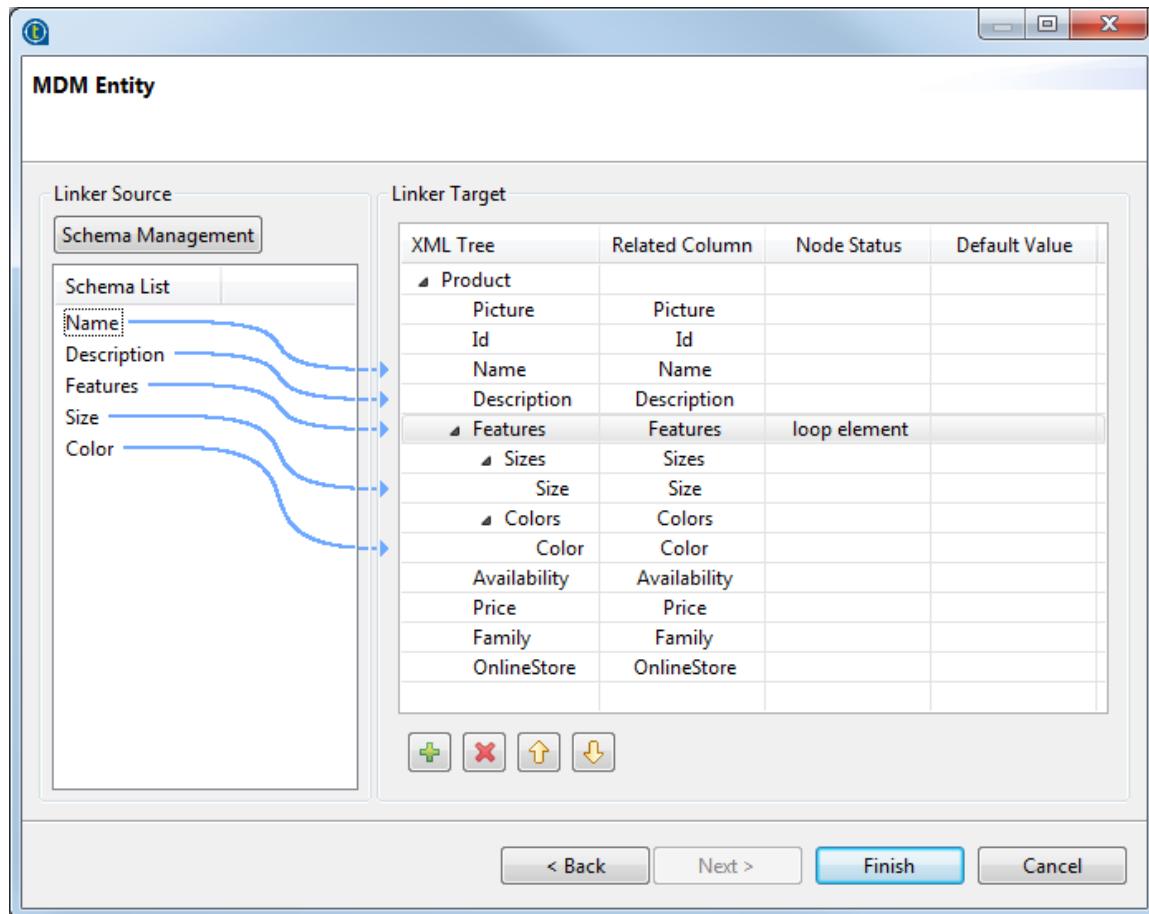
6. Click **Schema Management** to display a dialog box.
7. Do necessary modifications to define the XML schema you want to write in the selected entity.



Your **Linker Source** schema must corresponds to the **Linker Target** schema, that is to say define the elements in which you want to write values.

- Click **OK** to close the dialog box.

The defined schema is displayed under **Schema list**.



- In the **Linker Target** panel, right-click the element you want to define as a loop element and select **Set as loop element**. This will restrict the iteration to one or more nodes.

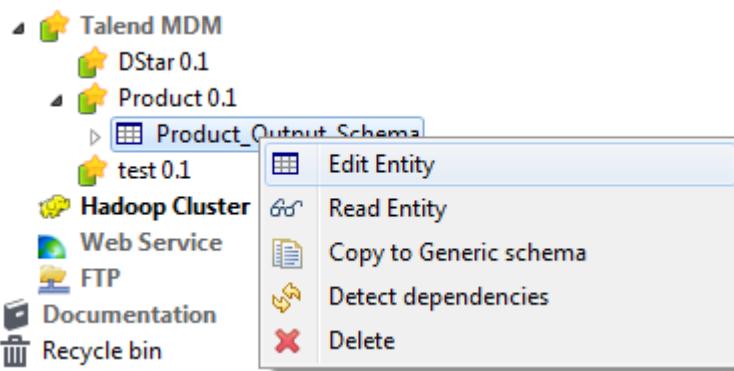
By doing so, the **tMDMOutput** component that uses this MDM connection will create a new row for every item with different feature.



You can prioritize the order of the fields to write by selecting the field and using the up and down arrows.

- Click **Finish** to validate your modifications and close the dialog box.

The newly created schema is listed under the corresponding MDM connection in the **Repository** tree view.



To modify the created schema, complete the following:

1. In the **Repository** tree view, expand **Metadata** and **Talend MDM** and then browse to the schema you want to modify.
2. Right-click the schema name and select **Edit Entity** from the contextual menu.

A dialog box is displayed.

Name	<input type="text" value="Product_Output_Schema"/>																																																					
Comment	<input type="text"/>																																																					
Schema																																																						
Click Guess button to update the schema below according to your settings																																																						
<input type="button" value="Guess"/>																																																						
Description of the Schema																																																						
<table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th>Column</th> <th>Key</th> <th>Type</th> <th><input checked="" type="checkbox"/> N..</th> <th>Date Pattern (Ctrl...)</th> <th>Length</th> <th>Precision</th> <th>Default</th> <th>Comment</th> </tr> </thead> <tbody> <tr> <td>Name</td> <td><input checked="" type="checkbox"/></td> <td>String</td> <td><input checked="" type="checkbox"/></td> <td></td> <td>0</td> <td></td> <td></td> <td></td> </tr> <tr> <td>Description</td> <td><input type="checkbox"/></td> <td>String</td> <td><input checked="" type="checkbox"/></td> <td></td> <td>0</td> <td></td> <td></td> <td></td> </tr> <tr> <td>Features</td> <td><input type="checkbox"/></td> <td>String</td> <td><input checked="" type="checkbox"/></td> <td></td> <td>0</td> <td></td> <td></td> <td></td> </tr> <tr> <td>Size</td> <td><input type="checkbox"/></td> <td>String</td> <td><input checked="" type="checkbox"/></td> <td></td> <td>0</td> <td></td> <td></td> <td></td> </tr> <tr> <td>Color</td> <td><input type="checkbox"/></td> <td>String</td> <td><input checked="" type="checkbox"/></td> <td></td> <td>0</td> <td></td> <td></td> <td></td> </tr> </tbody> </table>	Column	Key	Type	<input checked="" type="checkbox"/> N..	Date Pattern (Ctrl...)	Length	Precision	Default	Comment	Name	<input checked="" type="checkbox"/>	String	<input checked="" type="checkbox"/>		0				Description	<input type="checkbox"/>	String	<input checked="" type="checkbox"/>		0				Features	<input type="checkbox"/>	String	<input checked="" type="checkbox"/>		0				Size	<input type="checkbox"/>	String	<input checked="" type="checkbox"/>		0				Color	<input type="checkbox"/>	String	<input checked="" type="checkbox"/>		0			
Column	Key	Type	<input checked="" type="checkbox"/> N..	Date Pattern (Ctrl...)	Length	Precision	Default	Comment																																														
Name	<input checked="" type="checkbox"/>	String	<input checked="" type="checkbox"/>		0																																																	
Description	<input type="checkbox"/>	String	<input checked="" type="checkbox"/>		0																																																	
Features	<input type="checkbox"/>	String	<input checked="" type="checkbox"/>		0																																																	
Size	<input type="checkbox"/>	String	<input checked="" type="checkbox"/>		0																																																	
Color	<input type="checkbox"/>	String	<input checked="" type="checkbox"/>		0																																																	
<input type="button" value="+"/> <input type="button" value="X"/> <input type="button" value="Up"/> <input type="button" value="Down"/> <input type="button" value="Save"/> <input type="button" value="Cancel"/> <input type="button" value="Import"/>																																																						
<input type="button" value="&lt; Back"/> <input type="button" value="Next &gt;"/> <input type="button" value="Finish"/> <input type="button" value="Cancel"/>																																																						

3. Modify the schema as needed.

You can change the name of the schema according to your needs, you can also customize the schema structure in the schema panel. The tool bar allows you to add, remove or move columns in your schema.

4. Click **Finish** to close the dialog box.

The MDM output connection (**tMDMOutput**) is now ready to be dropped in any of your Jobs.

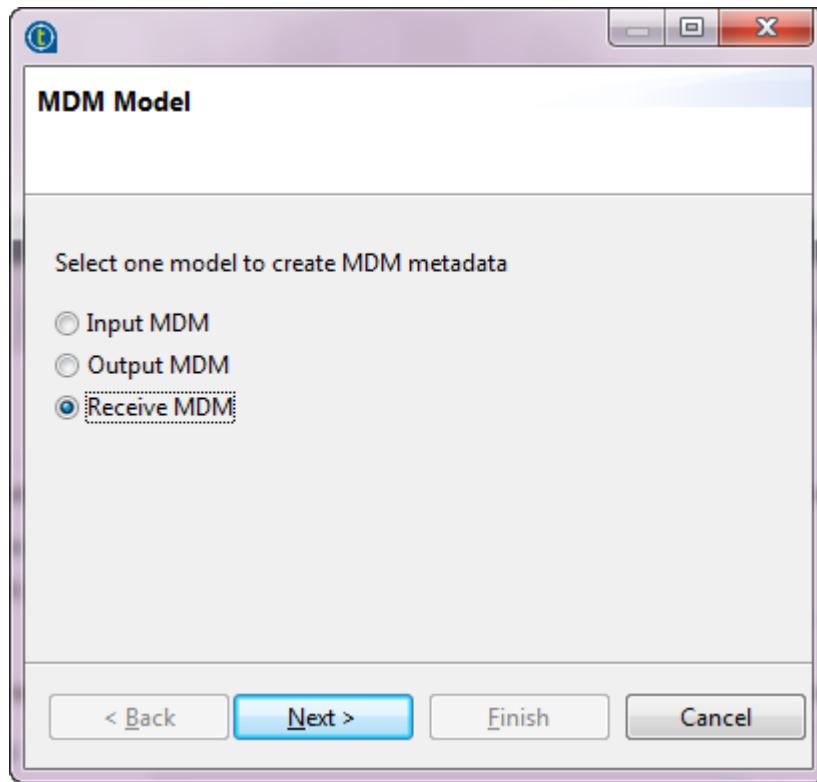
### 7.18.2.3. Defining Receive MDM schema

This section describes how to define a receive MDM XML schema based on the MDM connection.

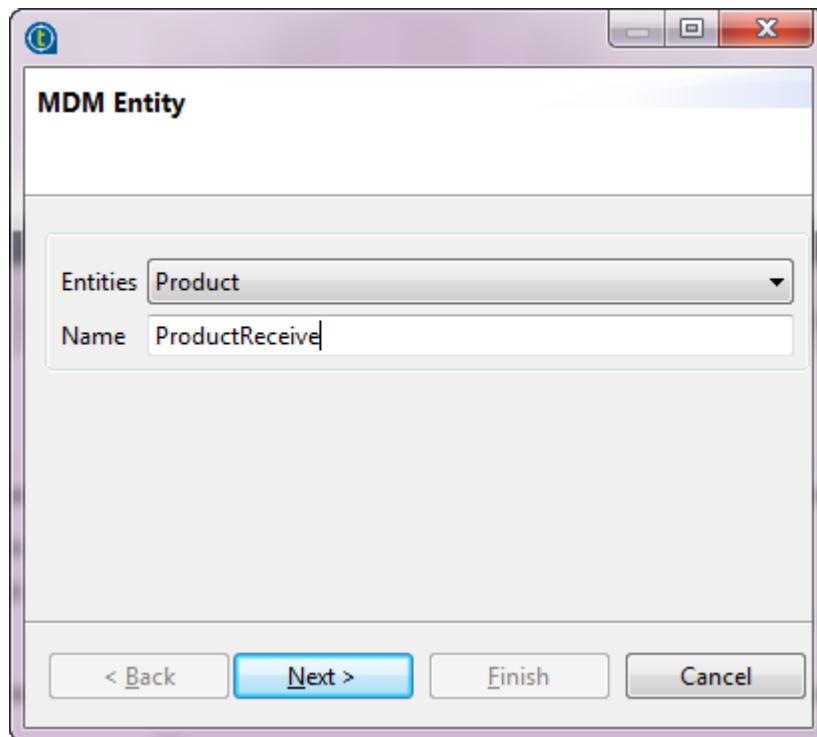
To set the XML schema you want to receive in accordance with a specific MDM connection, complete the following:

1. In the **Repository** tree view, expand **Metadata** and right-click the MDM connection for which you want to retrieve the entity values.
2. Select **Retrieve Entity** from the contextual menu.

A dialog box displays.



3. Select the **Receive MDM** option in order to define a receive XML schema and then click **Next** to proceed to the following step.



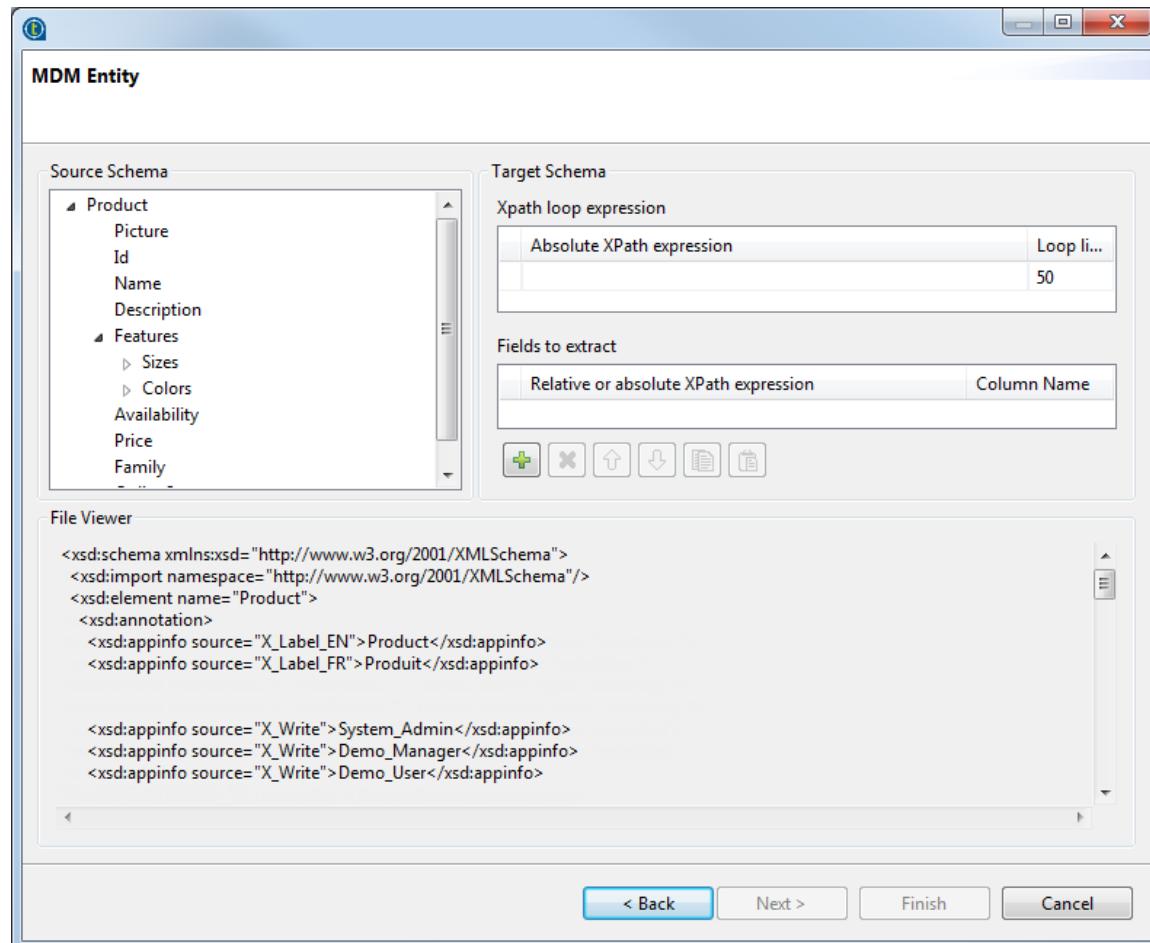
- From the **Entities** field, select the business entity (XML schema) according to which you want to receive the XML schema.

The name displays automatically in the **Name** field.



You can enter any text in this field, although you would likely put the name of the entity according to which you want to receive the XML schema.

- Click **Next** to proceed to the next step.



The schema of the entity you selected display in the **Source Schema** panel.

Here, you can set the parameters to be taken into account for the XML schema definition.

The schema dialog box is divided into four different panels as the following:

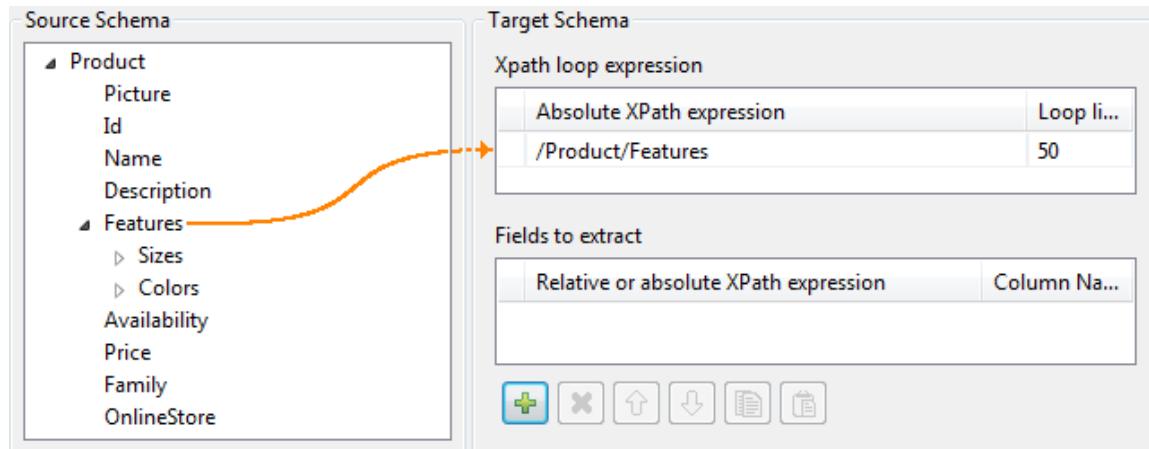
Panel	Description
Source Schema	Tree view of the uploaded entity.
Target schema	Extraction and iteration information.
Preview	Target schema preview.
File viewer	Raw data viewer.

- In the **Xpath loop expression** area, enter the absolute XPath expression leading to the XML structure node on which to apply the iteration. Or, drop the node from the source schema to the target schema Xpath field. This link is orange in color.



The **Xpath loop expression** field is compulsory.

7. If required, define a **Loop limit** to restrict the iteration to one or more nodes.

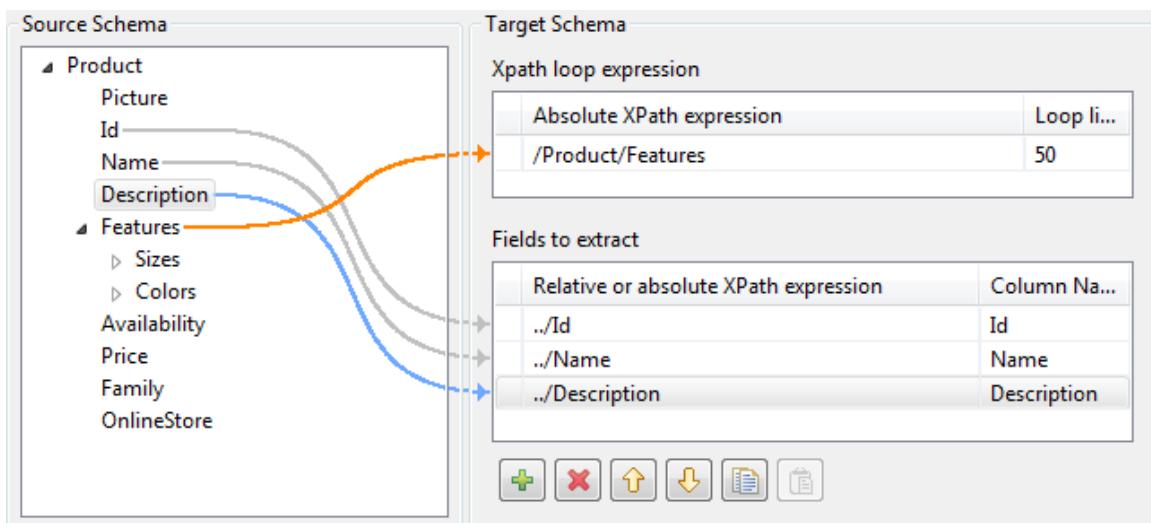


In the above capture, we use *Features* as the element to loop on because it is repeated within the *Product* entity as the following:

```
<Product>
<Id>1</Id>
<Name>Cup</Name>
<Description/>
<Features>
 <Feature>Color red</Feature>
 <Feature>Size maxi</Feature>
<Features>
 ...
</Product>
<Product>
<Id>2</Id>
<Name>Cup</Name>
<Description/>
<Features>
 <Feature>Color blue</Feature>
 <Feature>Thermos</Feature>
<Features>
 ...
</Product>
```

By doing so, the **tMDMReceive** component that uses this MDM connection will create a new row for every item with different feature.

8. To define the fields to receive, drop the relevant node from the source schema to the **Relative or absolute XPath expression** field.



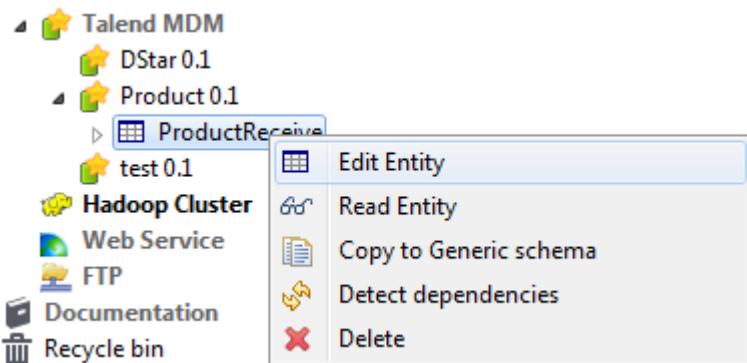
 Use the plus sign to add rows to the table and select as many fields to extract as necessary. Press the **Ctrl** or the **Shift** keys for multiple selection of grouped or separate nodes and drop them to the table.

- If required, enter a name to each of the received columns in the **Column name** field.

 You can prioritize the order of the fields you want to receive by selecting the field and using the up and down arrows. The link of the selected field is blue, and all other links are grey.

- Click **Finish** to validate your modifications and close the dialog box.

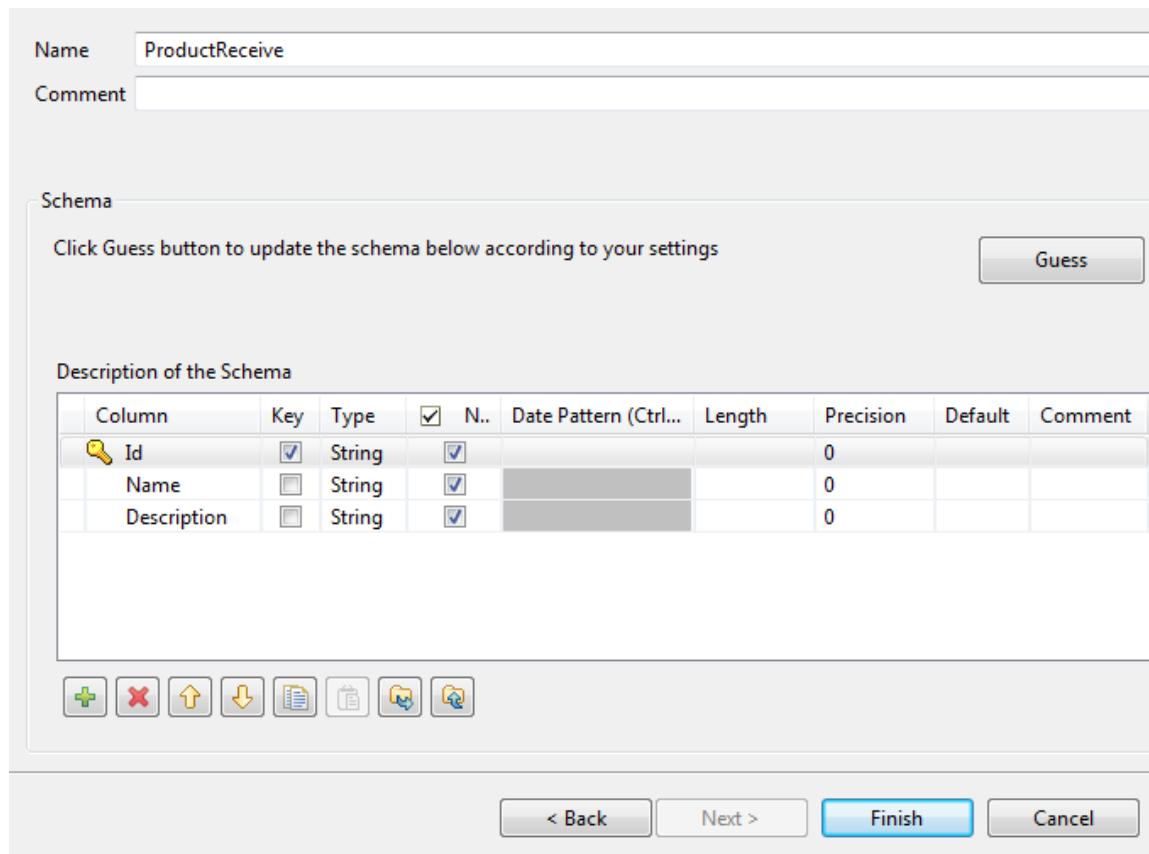
The newly created schema is listed under the corresponding MDM connection in the **Repository** tree view.



To modify the created schema, complete the following:

- In the **Repository** tree view, expand **Metadata** and **Talend MDM** and then browse to the schema you want to modify.
- Right-click the schema name and select **Edit Entity** from the contextual menu.

A dialog box displays.



3. Modify the schema as needed.

You can change the name of the schema according to your needs, you can also customize the schema structure in the schema panel. The tool bar allows you to add, remove or move columns in your schema.

Make sure the data type in the **Type** column is correctly defined.

For more information regarding Java data types, including date pattern, see [Java API Specification](#).

Below are the commonly used **Talend** data types:

- Object: a generic **Talend** data type that allows processing data without regard to its content, for example, a data file not otherwise supported can be processed with a **tFileInputRaw** component by specifying that it has a data type of Object.
- List: a space-separated list of primitive type elements in an XML Schema definition, defined using the xsd:list element.
- Document: a data type that allows processing an entire XML document without regarding to its content.

4. Click **Finish** to close the dialog box.

The MDM receive connection (**tMDMReceive**) is now ready to be dropped in any of your Jobs.

## 7.19. Centralizing Web Service metadata

If you often need to visit a Web Service from your *Talend Studio* you can save your Web Service connections in the **Repository**.

The [Web Service] schema wizard enables you to create either a simple schema (**Simple WSDL**) or an advanced schema (**Advanced WebService**), according to your needs.



In step 1, you must enter the schema metadata before choosing whether to create a simple or an advanced schema in step 2. It is therefore important to enter metadata information which will help you to differentiate between your different schema types in the future.

To create a simple schema, see [Setting up a simple schema](#).

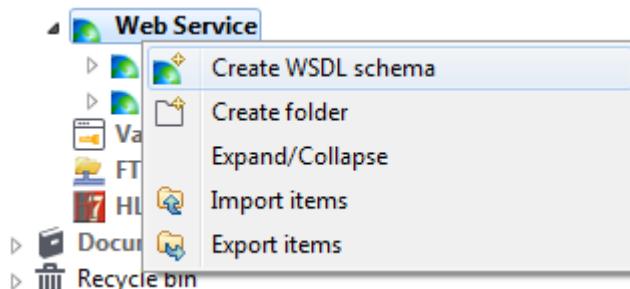
To create an advanced schema, see [Setting up an advanced schema](#).

## 7.19.1. Setting up a simple schema

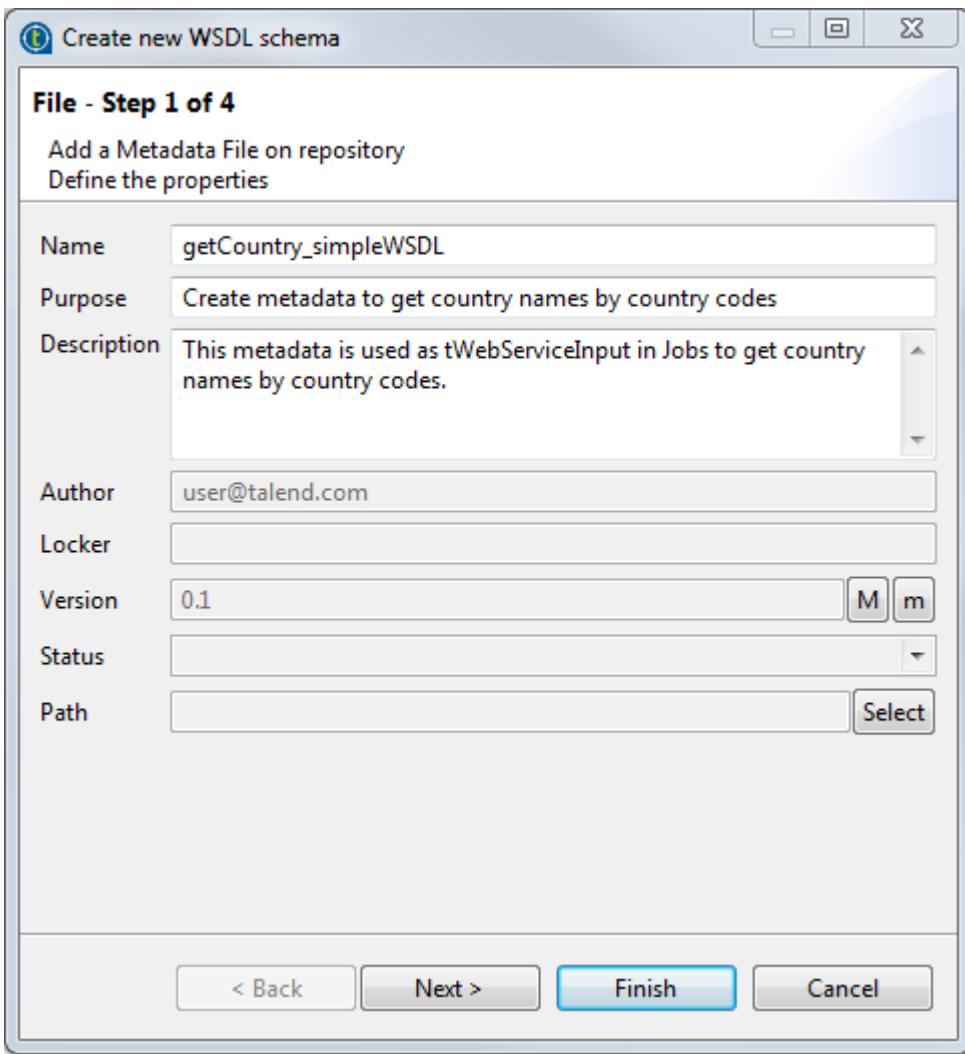
This section describes how to define a simple Web Service schema (Simple WSDL). For information about how to define an Advanced Web Service schema, see [Setting up an advanced schema](#).

### Defining general properties

1. In the **Repository**, expand the **Metadata** node.
2. Right-click **Web Service** and select **Create WSDL schema** from the context menu list.



3. Enter the generic schema information such as its **Name** and **Description**.

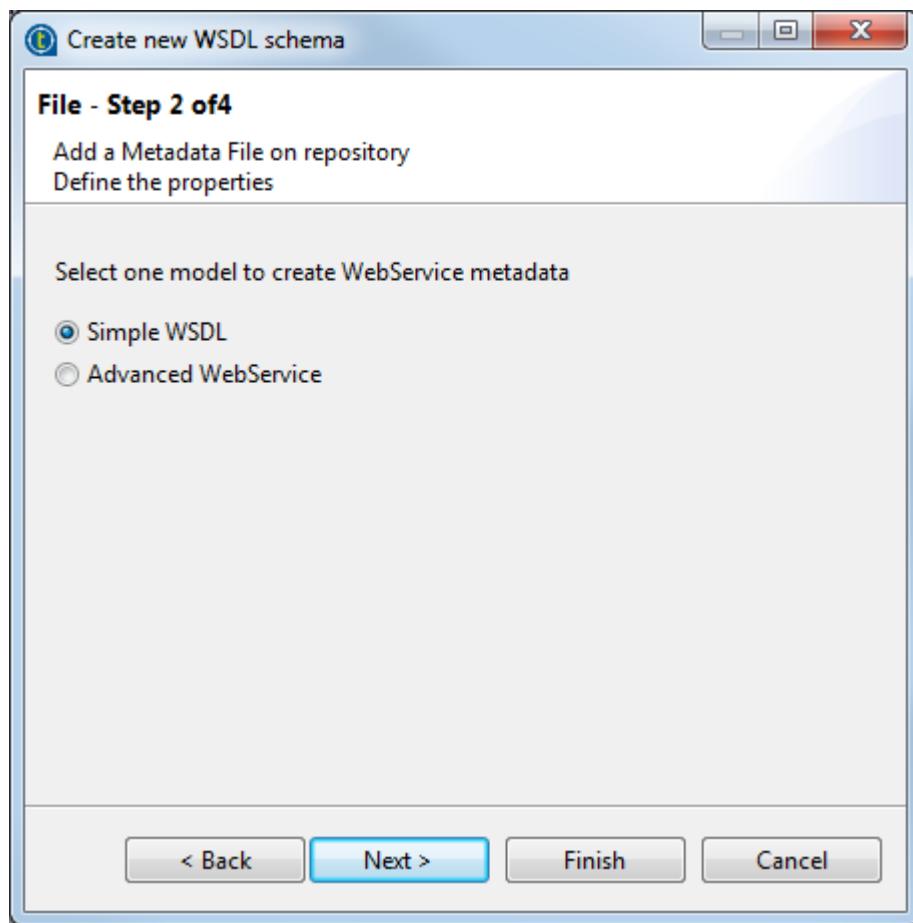


4. Click **Next** to select the schema type in step 2.

### Selecting the type of schema (Simple)

In this step, you need to indicate whether you want to create a simple or an advanced schema. In this example, a simple schema is created.

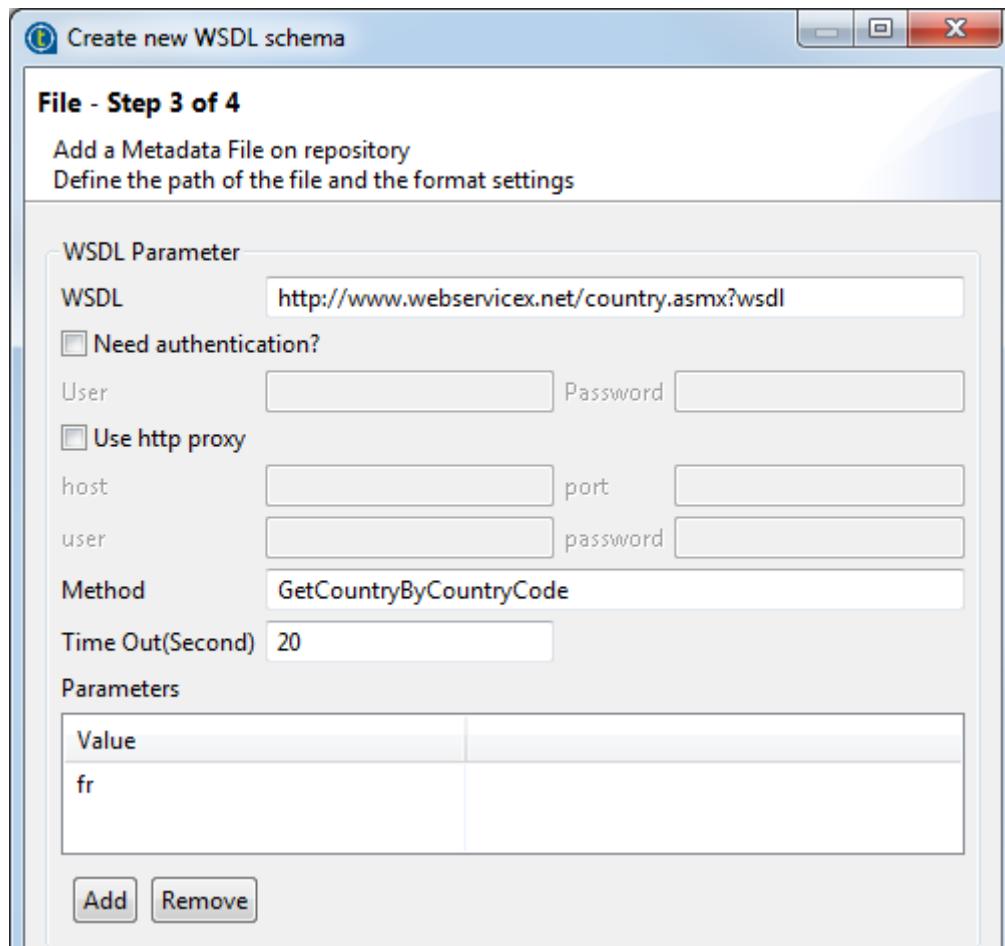
1. In the dialog box, select the **Simple WSDL** option.



2. Click **Next** to continue.

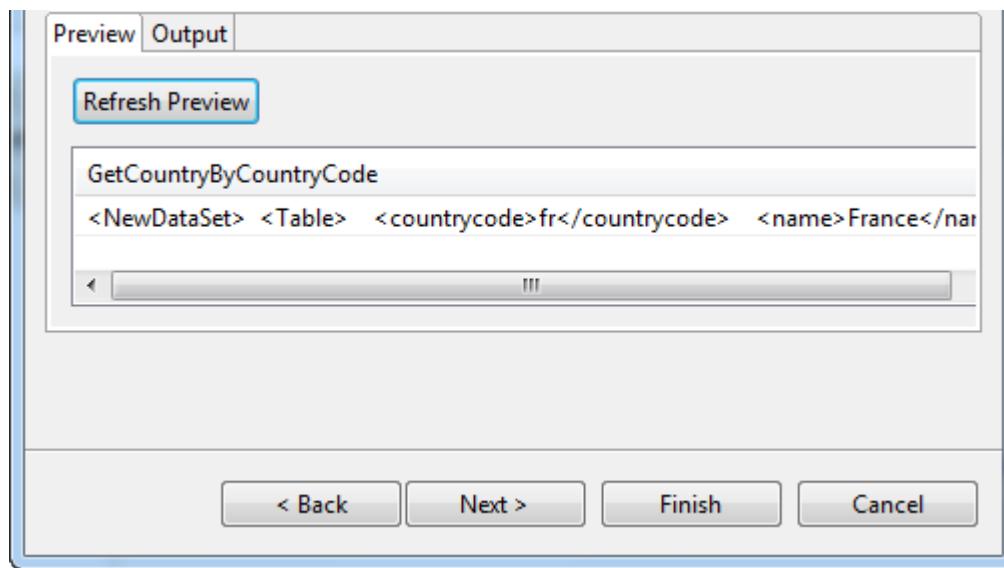
### Specifying the URI and method

This step involves the definition of the URI and other parameters required to obtain the desired values.



In the **Web Service Parameter** zone:

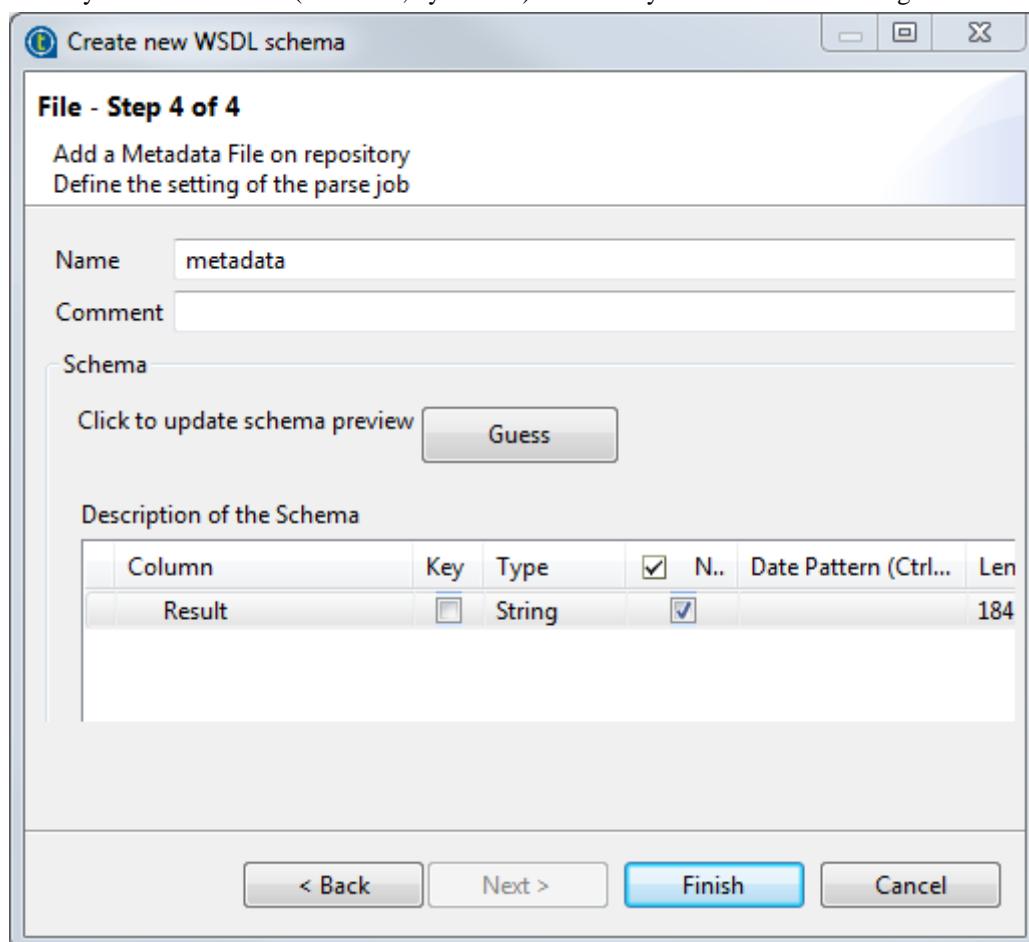
1. Enter the URI which will transmit the desired values, in the **WSDL** field, *http://www.webservicex.net/country.asmx?wsdl* in this example.
2. If necessary, select the **Need authentication?** check box and then enter your authentication information in the **User** and **Password** fields.
3. If you use an http proxy, select the **Use http proxy** check box and enter the information required in the **host**, **Port**, **user** and **password** fields.
4. Enter the **Method** name in the corresponding field, *GetCountryByCountryCode* in this example.
5. In the **Value** table, **Add** or **Remove** values as desired, using the corresponding buttons.
6. Click **Refresh Preview** to check that the parameters have been entered correctly.



In the **Preview** tab, the values to be transmitted by the Web Service method are displayed, based the parameters entered.

## Finalizing the end schema

You can modify the schema name (*metadata*, by default) and modify the schema itself using the tool bar.



1. Add or delete columns using the and buttons.

2. Modify the order of the columns using the and buttons.
3. Click **Finish**.

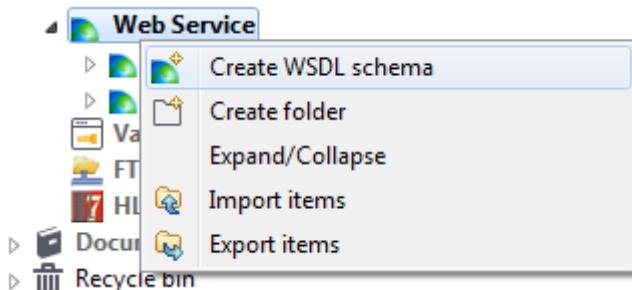
The new schema is added to the **Repository** under the **Web Service** node. You can now drop it onto the design workspace as a **tWebServiceInput** component in your Job.

## 7.19.2. Setting up an advanced schema

This section describes how to define an **Advanced WebService** schema. For information about how to define a **Simple WSDL** schema, see [Setting up a simple schema](#).

### Defining general properties

1. In the **Repository** view, expand the **metadata** node.
2. Right-click **Web Service** and select **Create WSDL schema** from the context menu list.



3. Enter the generic schema information, such as its **Name** and **Description**.

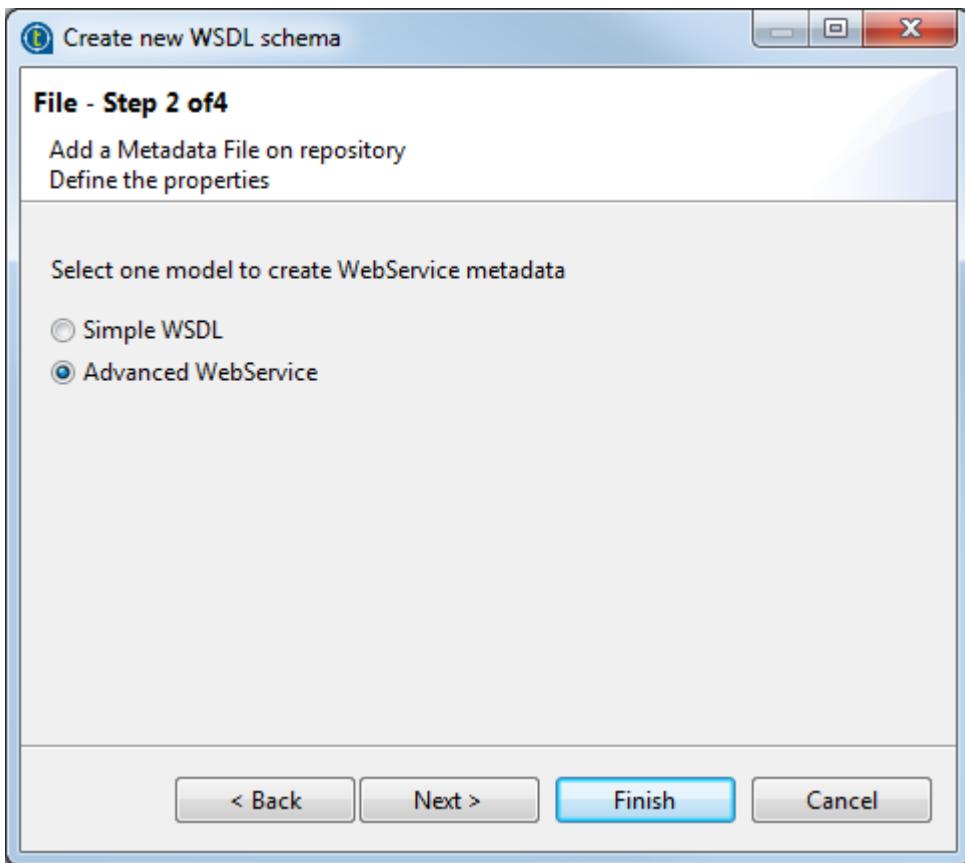


4. Click **Next** to select the schema type in step 2.

### Selecting the type of schema (Advanced)

In this step, you must indicate whether you want to create a **Simple** or an **Advanced** schema. In this example, an **Advanced** schema is created.

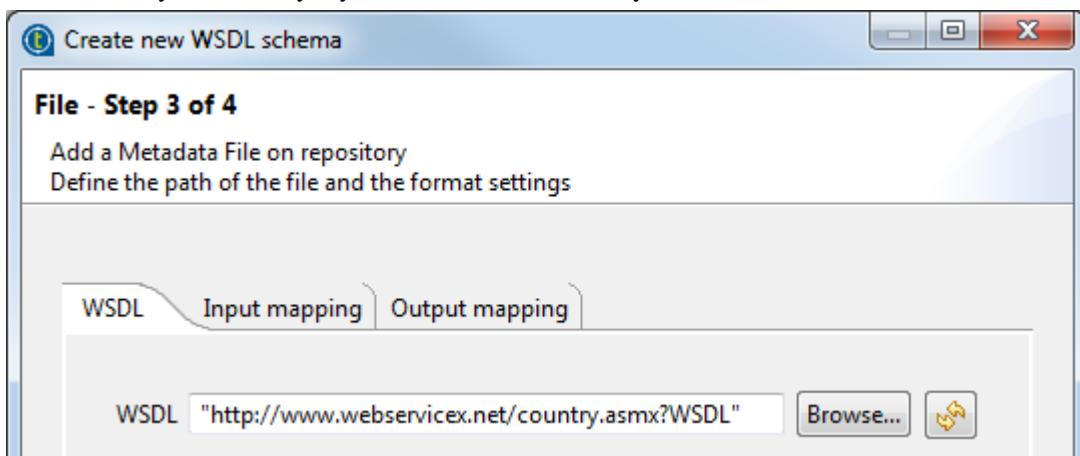
1. In the dialog box, select the **Advanced WebService** option.



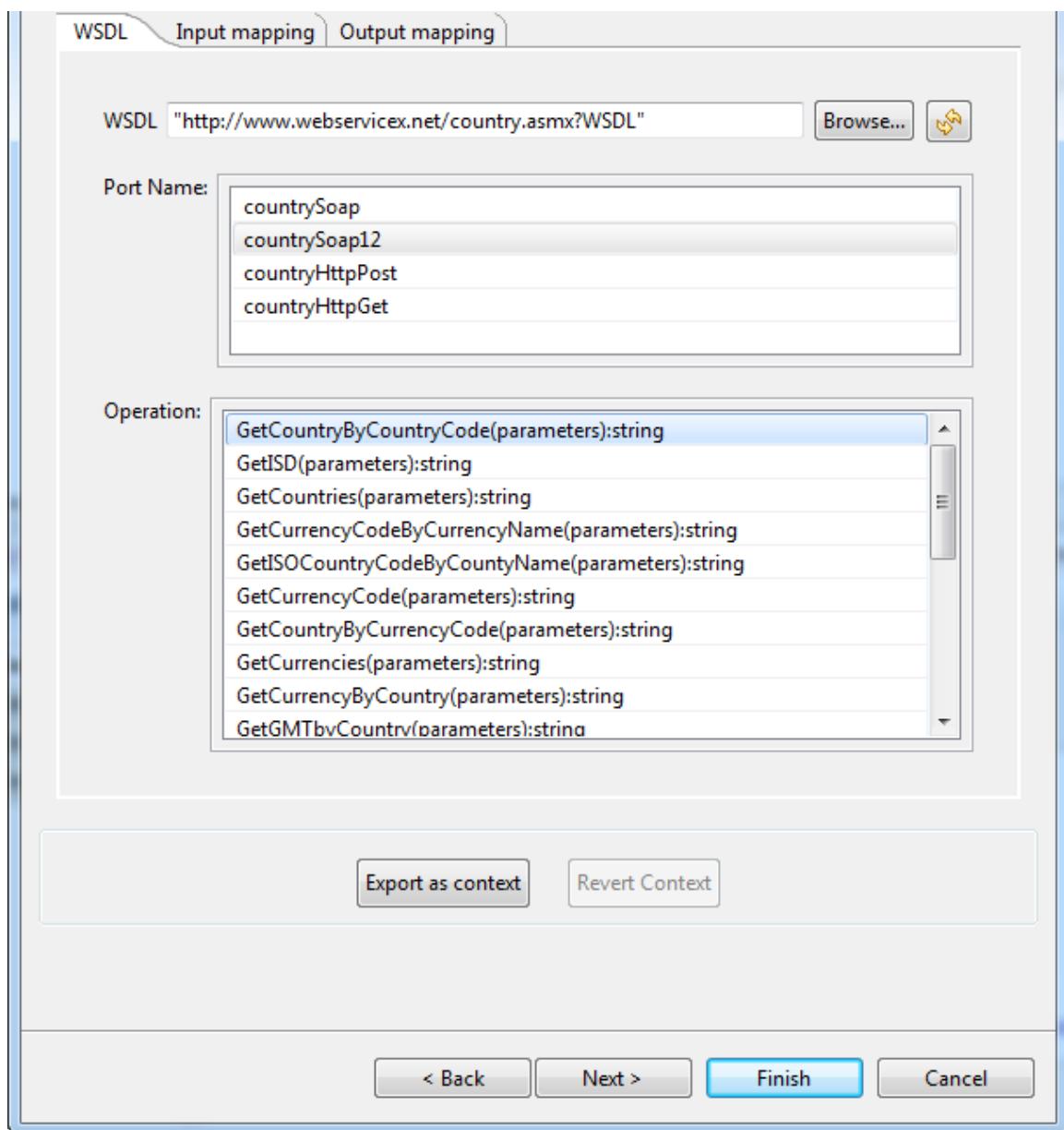
2. Click **Next** to define precise Web Service parameters.

### Defining the port name and operation

1. Type in the URI of the Web Service WSDL file manually by typing in the **WSDL** field, or click the **Browse...** button to browse your directory if your WSDL is stored locally.



2. Click the **Refresh** button to retrieve the list of port names and operations available.



3. Select the port name to be used, in the **Port Name** zone, *countrySoap12* in this example.
4. Select the operation to be carried out in the **Operation** zone.

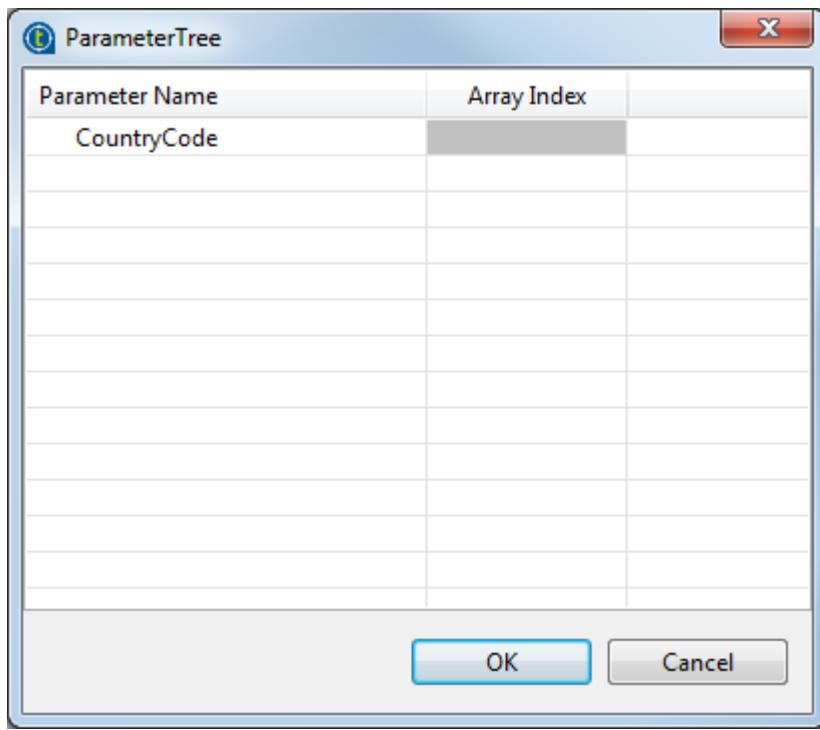
In this example, select *GetCountryByCountryCode(parameters):string* to retrieve the country name for a given country code.

Next, you need to define the input and output schemas and schema-parameter mappings in the **Input mapping** and **Output mapping** tabs.

### Defining the input schemas and mappings

To define the input schema and mappings, do the following:

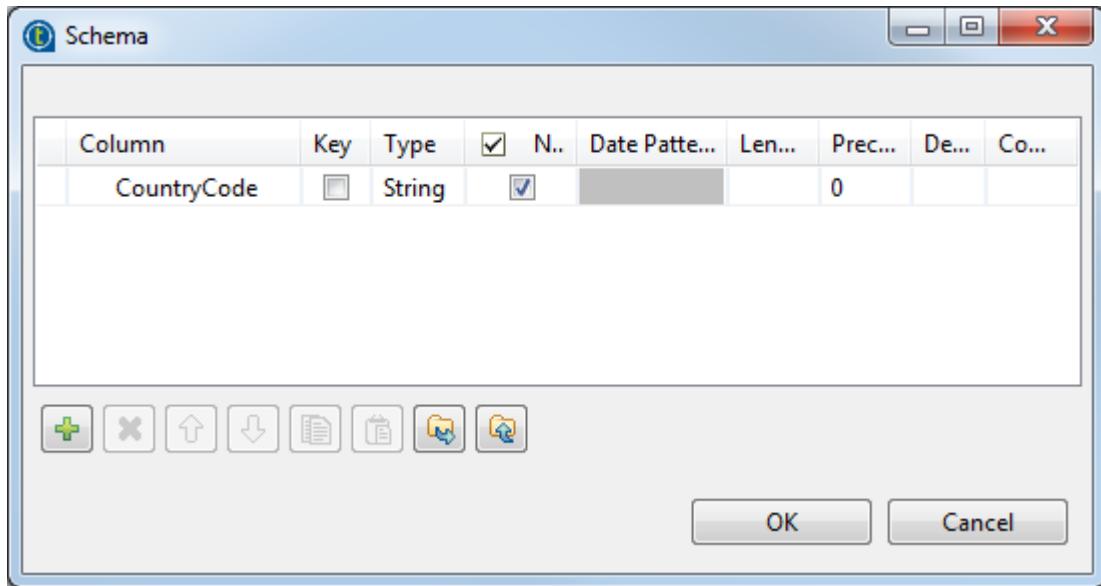
1. Click the **Input mapping** tab to define the input schema and set the parameters required to execute the operation.
2. In the table to the right, select the **parameters** row and click the **[+]** button to open the **[ParameterTree]** dialog box.



3. Select the parameter you want to use and click **OK** to close the dialog box.

A new row appears showing the parameter you added, *CountryCode* in this example.

4. In the table to the left, click the **Schema Management** button to open the [**Schema**] dialog box.



5. Define the input schema.

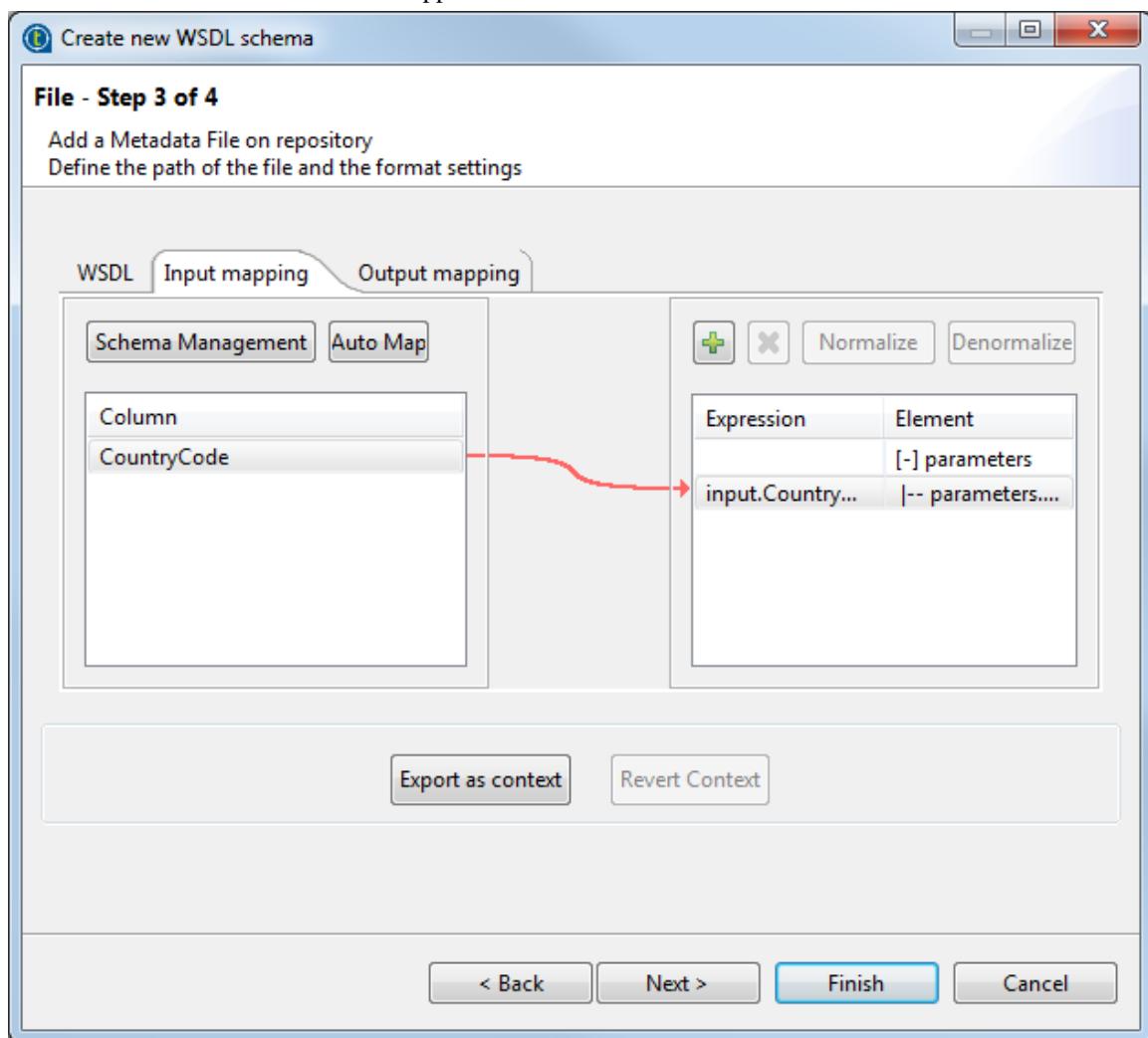
In this example, the schema has only one column: *CountryCode*.

6. Click **OK** to validate this addition and close the dialog box.

7. Create mappings between schema columns and parameters.

In this example, drop the *CountryCode* column from the left table onto the *parameters.CountryCode* row to the right.

A red line shows that the column is mapped.



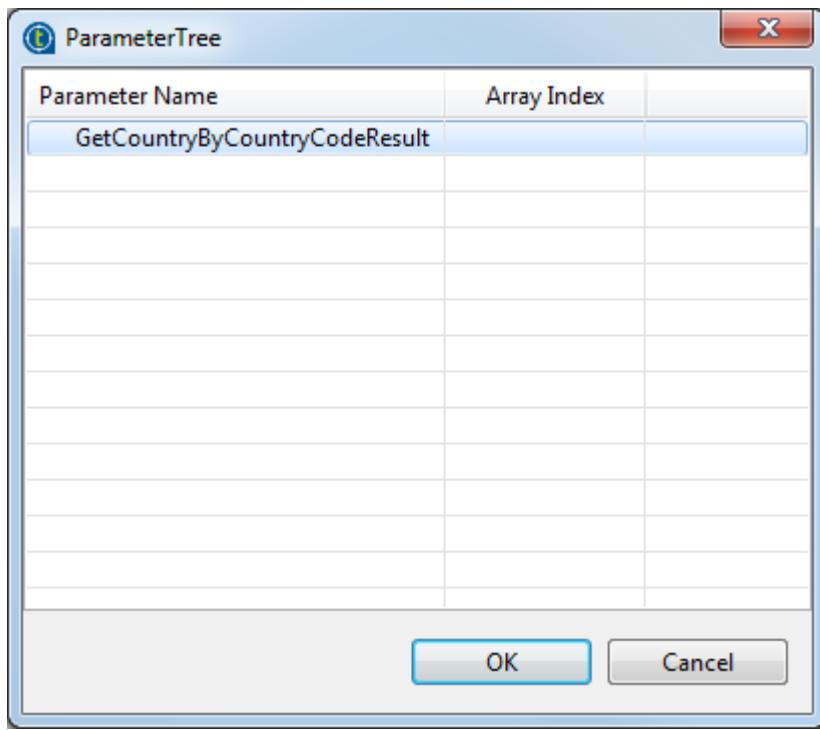
If available, use the **Auto Map** button situated to the top of the tab, to carry out the mapping automatically.

## Defining the output schemas and mappings

To define the output schema and mappings, proceed as follows:

1. Click the **Output mapping** tab to define the output schema and set its parameters.
2. In the table to the left, select the **parameter** row and click the **[+]** button to add a parameter.

The **[ParameterTree]** dialog box opens.



3. Select the parameter and click **OK** to close the dialog box.

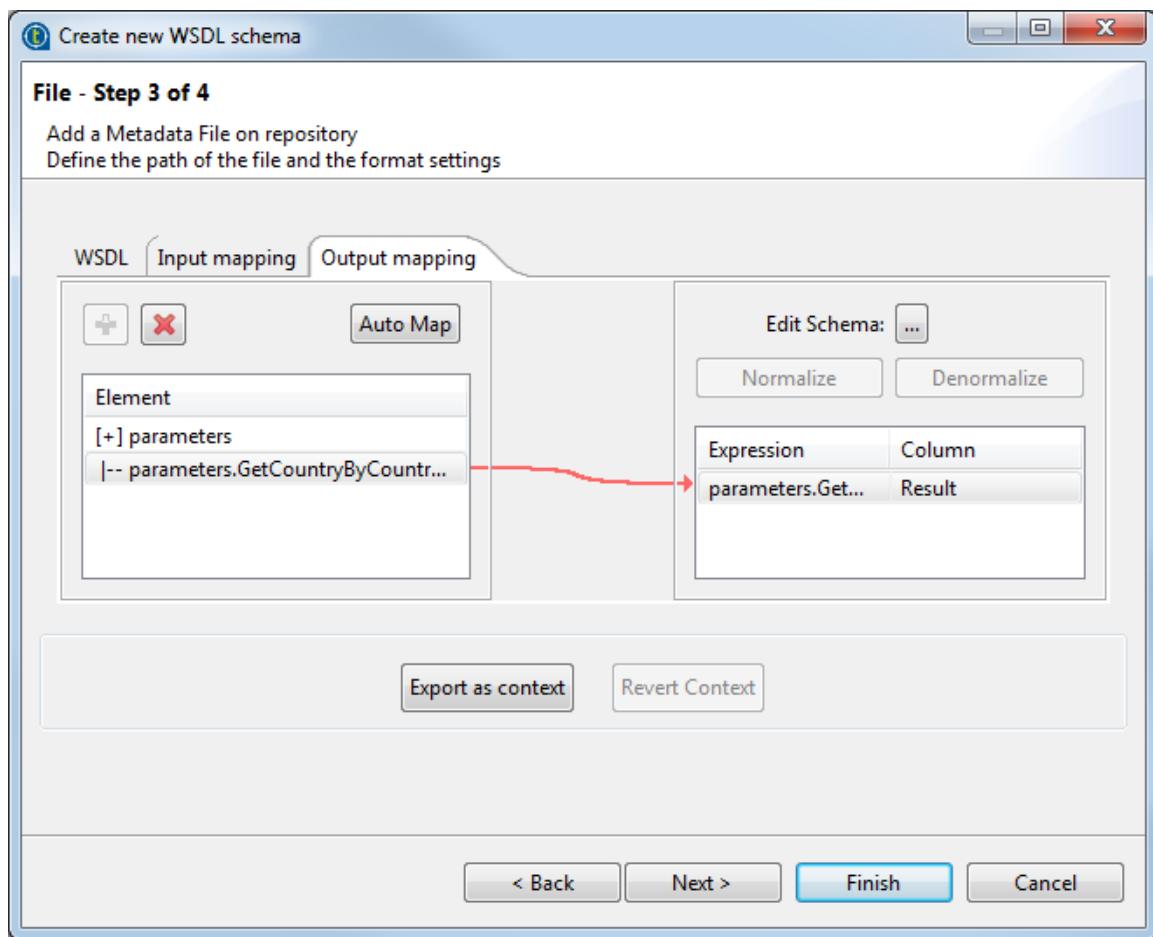
A new row appears showing the parameter you added, *GetCountryByCountryCodeResult* in this example.

4. In the table to the right, click [...] to open the [Schema] dialog box.
5. Define the output schema.

In this example, the schema has only one column: *Result*.

6. Click **OK** to validate your addition and close the dialog box.
7. Create output parameter-schema mappings.

In this example, drop the *parameters.GetCountryByCountryCodeResult* row from the table to the left onto the *Result* column to the right.



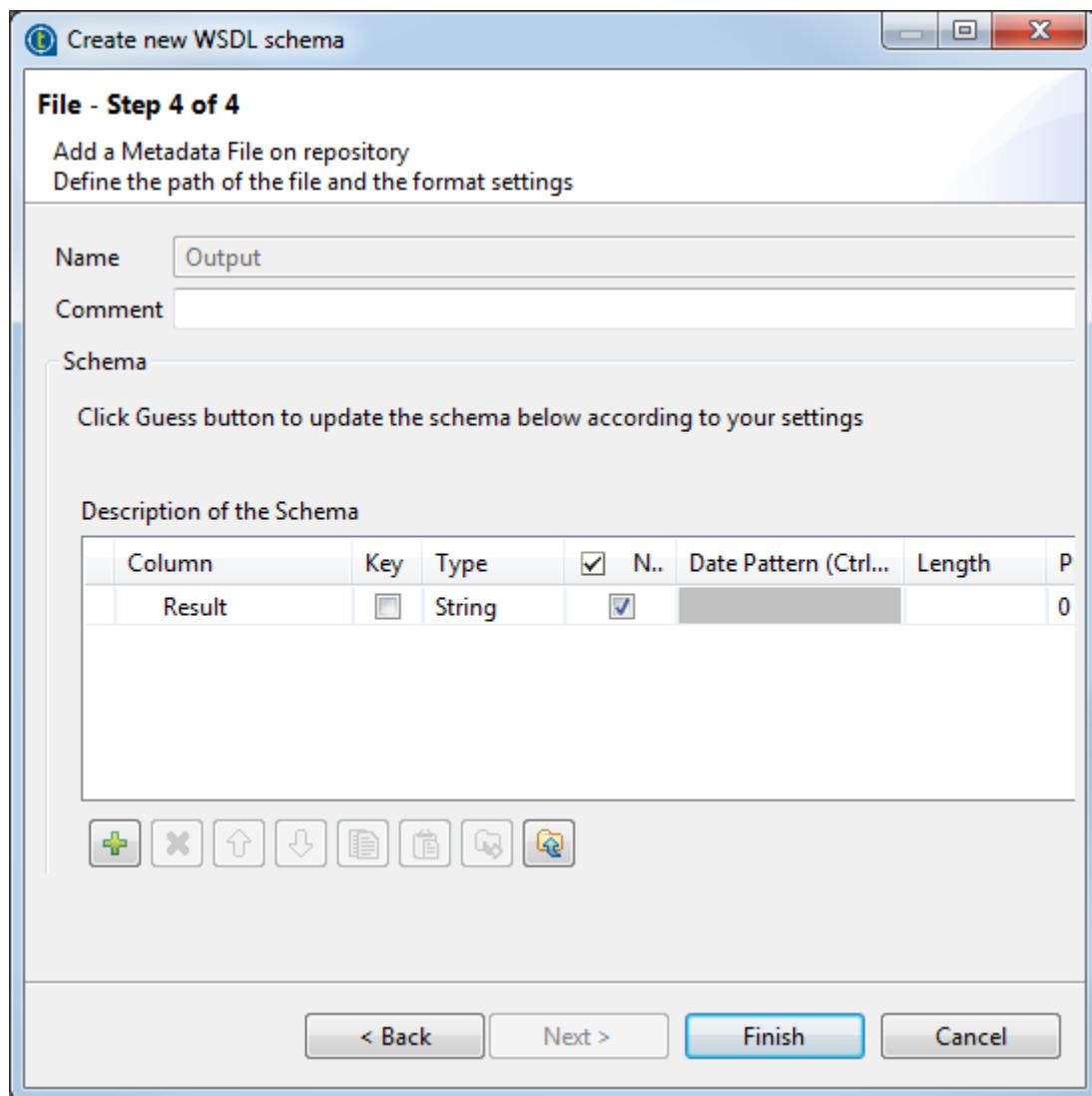
8. Click **Next** to finalize the schema.



Depending on the type of the output, you can choose to normalize or denormalize the results by clicking the **Normalize** and **Denormalize** buttons.

### Finalizing the end schema

In this step the wizard displays the output schema generated.



You can customize the metadata by changing or adding information in the **Name** and **Comment** fields and make further modifications using the toolbar, for example:

1. Add or delete columns using the and buttons.
2. Change the column order by clicking the and arrows.
3. Click **Finish** to finalize your advanced schema.

The new schema is added to the **Repository** under the corresponding Web Service node. You can now drop it onto the design workspace as a **tWebService** component in your Job.

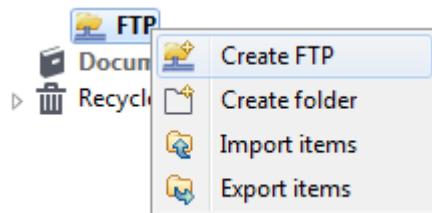
## 7.20. Centralizing an FTP connection

If you need to connect to an FTP server regularly, you can centralize the connection information under the **Metadata** node in the **Repository** view.

### Defining the general properties

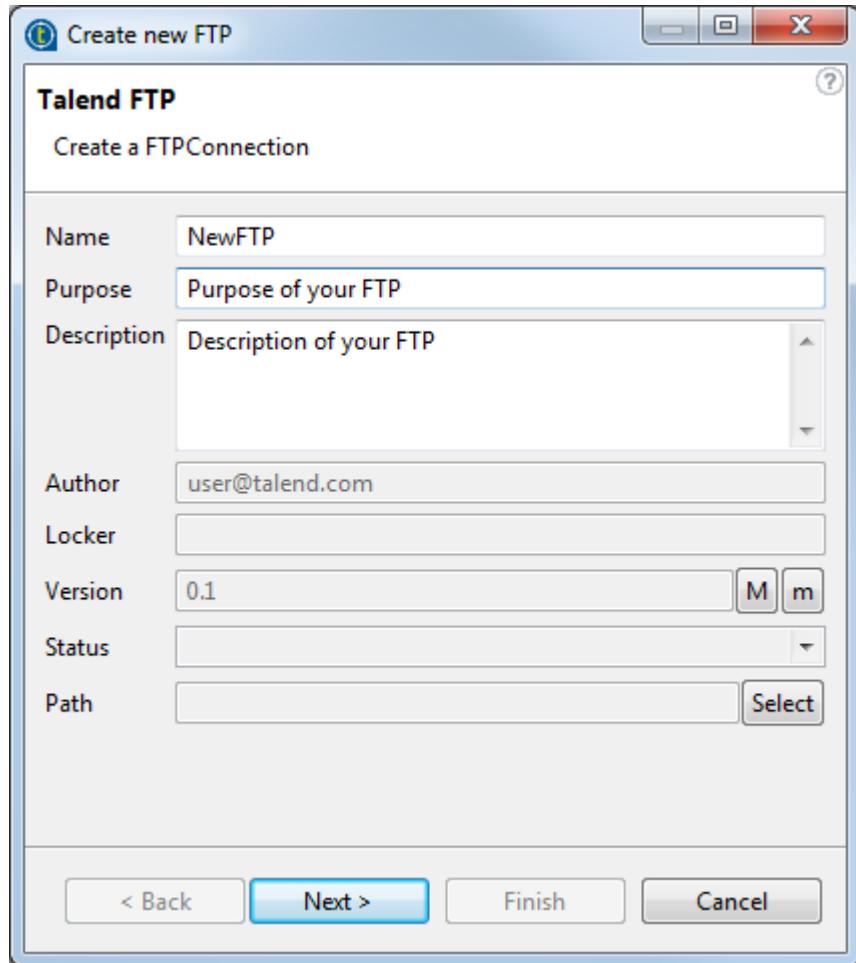
To create a connection to an FTP server, follow the steps below:

1. Expand the **Metadata** node in the **Repository** tree view.



2. Right-click **FTP** and select **Create FTP** from the context menu.

The connection wizard opens:



3. Enter the generic schema information such as its **Name** and **Description**.



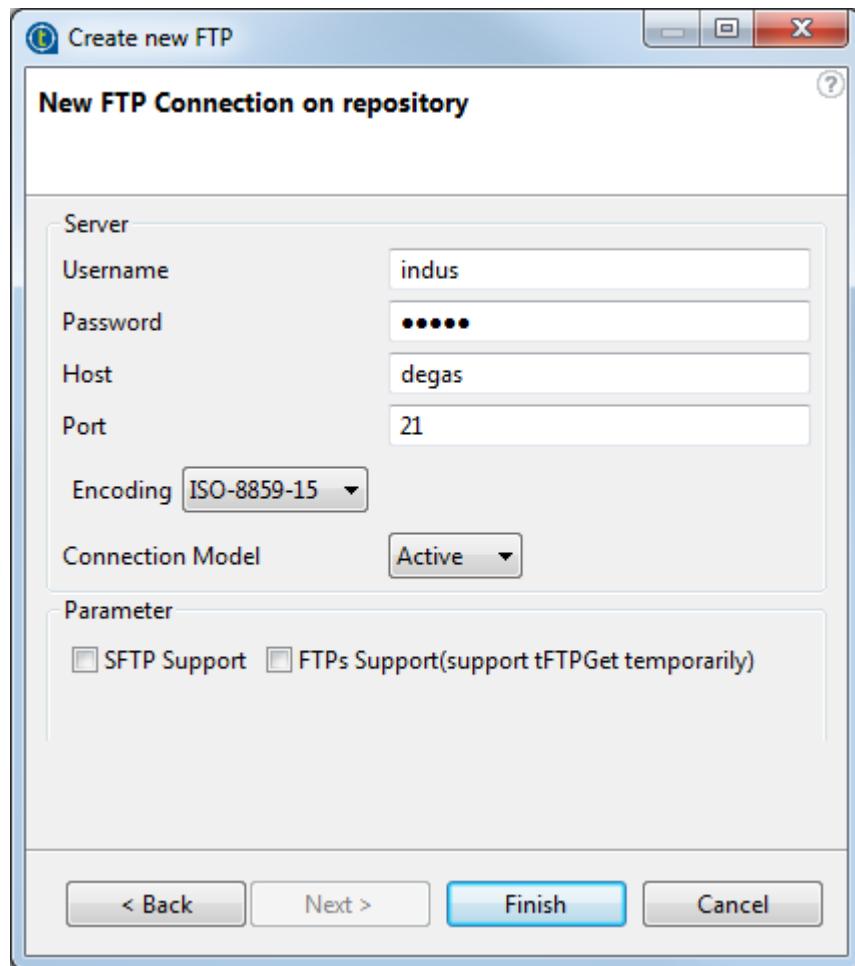
The status field is a customized field which can be defined in the [Preferences] dialog box (**Window > Preferences**). For further information about setting preferences, see [Setting Talend Studio preferences](#).

4. When you have finished, click **Next** to enter the FTP server connection information.

## Connecting to an FTP server

In this step we shall define the connection information and parameters.

1. Enter your **Username** and **Password** in the corresponding fields.



2. In the **Host** field, enter the name of your FTP server host.
3. Enter the **Port** number in the corresponding field.
4. Select the **Encoding** type from the list.
5. From the **Connection Model** list, select the connection model you want to use:
  - Select **Passive** if you want the FTP server to choose the port connection to be used for data transfer.
  - Select **Active** if you want to choose the port yourself.
6. In the **Parameter** area, select a setting for FTP server usage. For standard usage, there is no need to select an option.
  - Select the **SFTP Support** check box to use the SSH security protocol to protect server communications. An **Authentication method** appears. Select **Public key** or **Password** according to what you use.
  - Select the **FTPs Support** check box to protect server communication with the SSL security protocol.
  - Select the **Use Socks Proxy** check box if you want to use this option, then enter the proxy information (the host name, port number, username and password).
7. Click **Finish** to close the wizard.

All of the connections created appear under the FTP server connection node, in the **Repository** view.

You can drop the connection metadata from the **Repository** onto the design workspace. A dialog box opens in which you can choose the component to be used in your Job.

For further information about how to drop metadata onto the workspace, see [How to use centralized metadata in a Job](#).

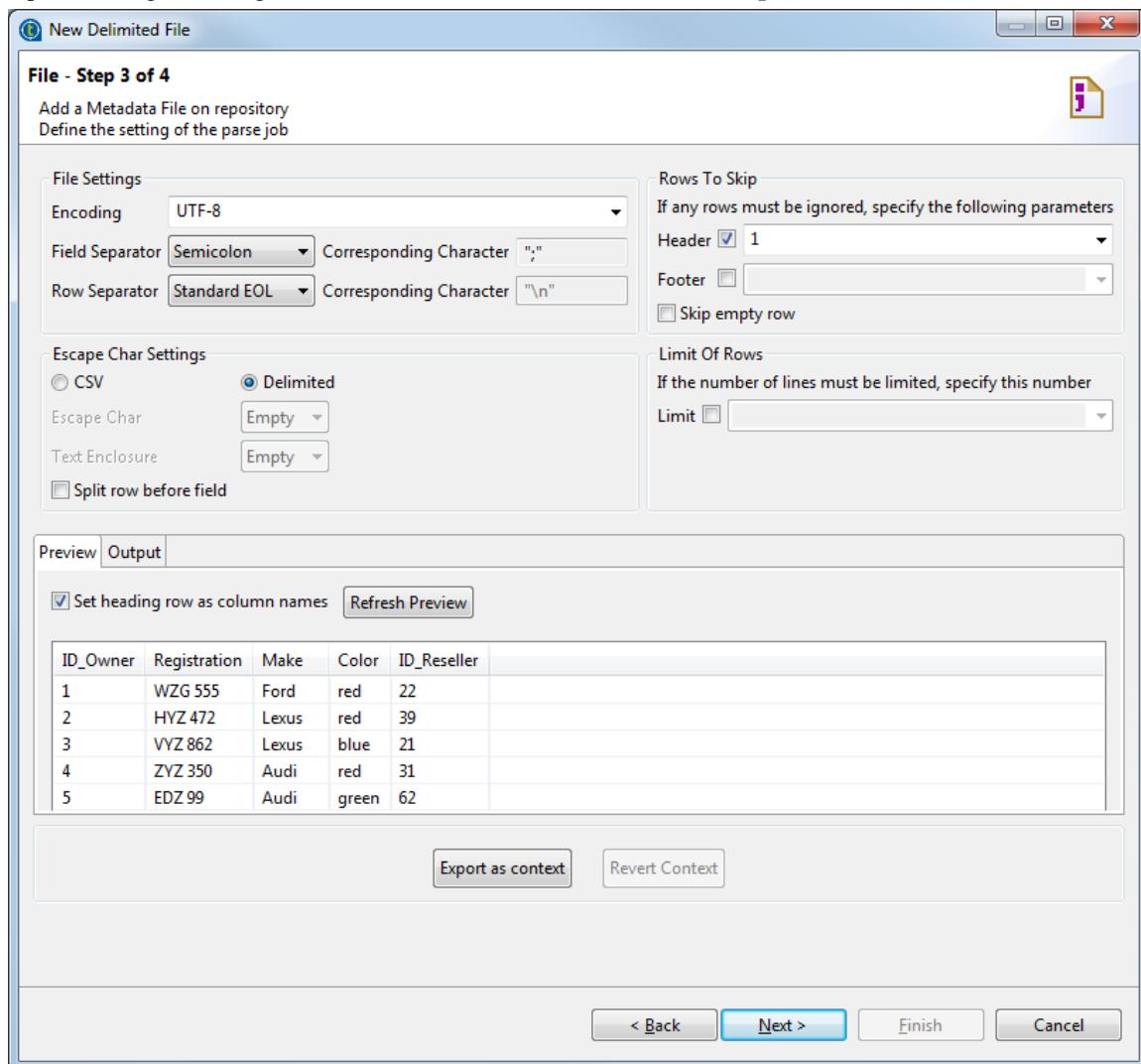
## 7.21. Exporting metadata as context and reusing context parameters to set up a connection

For every metadata connection (File, Database or Talend MDM, etc.), you can export the connection details to a new context group in the Repository for reuse in other connections or across different Jobs, or reuse variables of an existing context group to set up your metadata connection.

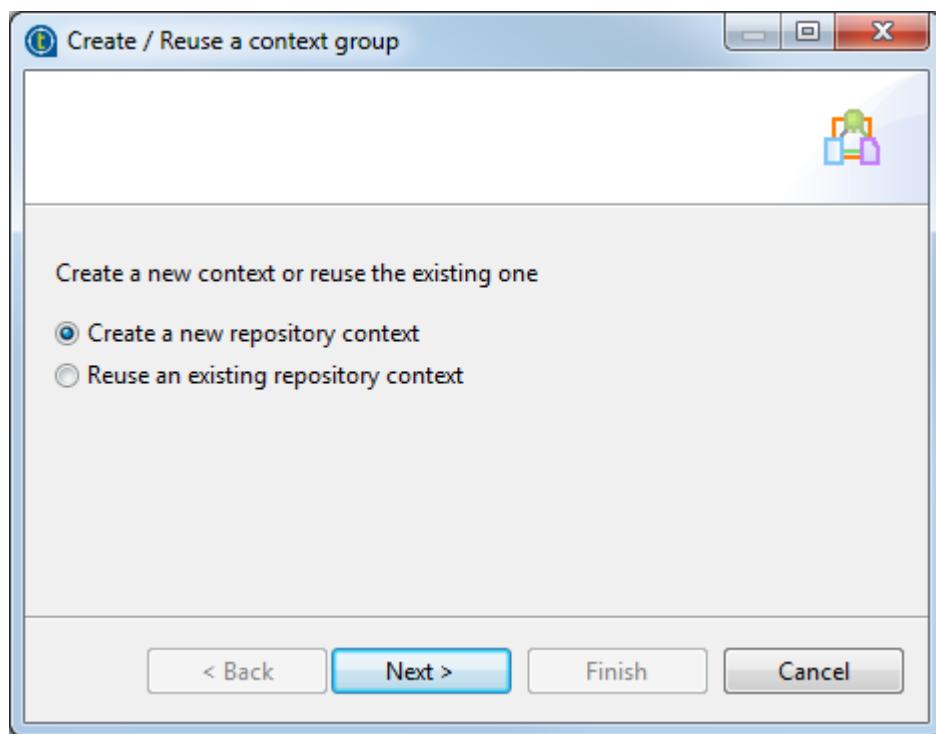
### 7.21.1. How to export connection details as context variables

To export connection details as context variables in a new context group in the Repository, follow the steps below:

- Upon creating or editing a metadata connection in the wizard, click **Export as context**.

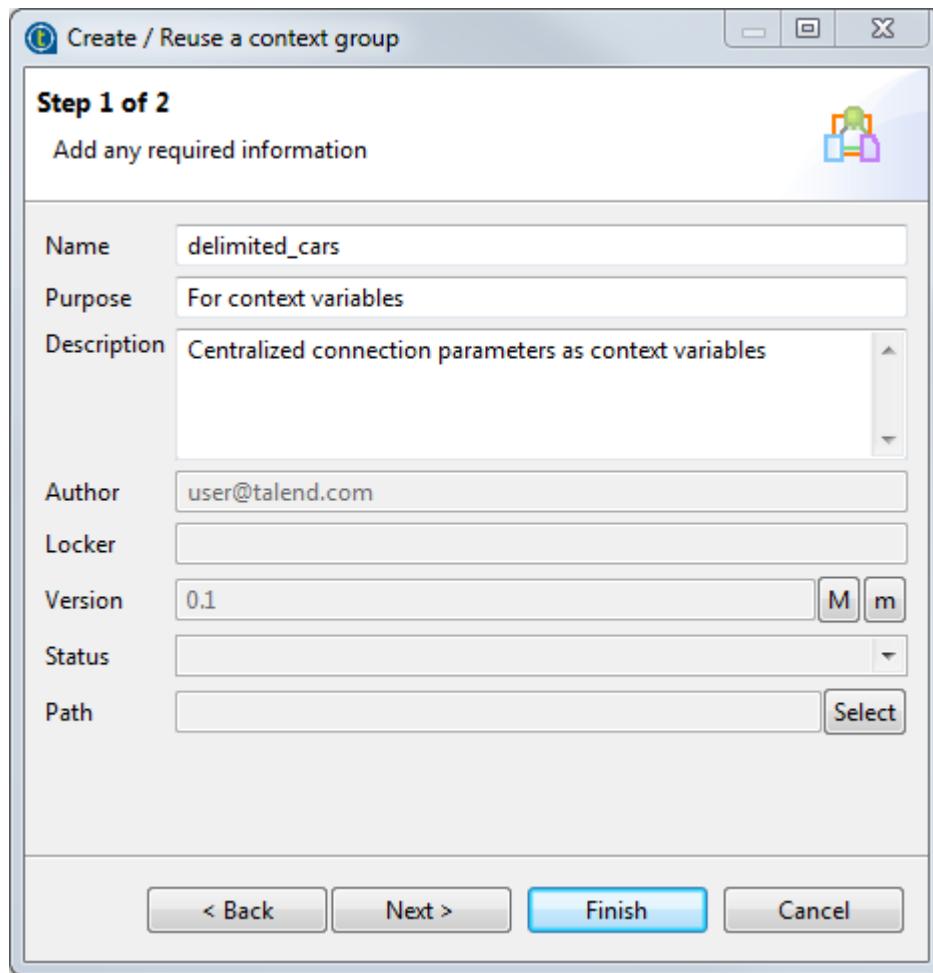


2. In the [Create / Reuse a context group] wizard that opens, select **Create a new repository context** and click **Next**.



3. Type in a name for the context group to be created, and add any general information such as a description if required.

The name of the Metadata entry is proposed by the wizard as the context group name, and the information you provide in the **Description** field will appear as a tooltip when you move your mouse over the context group in the Repository.



- Click **Next** to create and view the context group, or click **Finish** to complete context creation and return to the connection wizard directly.

In this example, click **Next**.

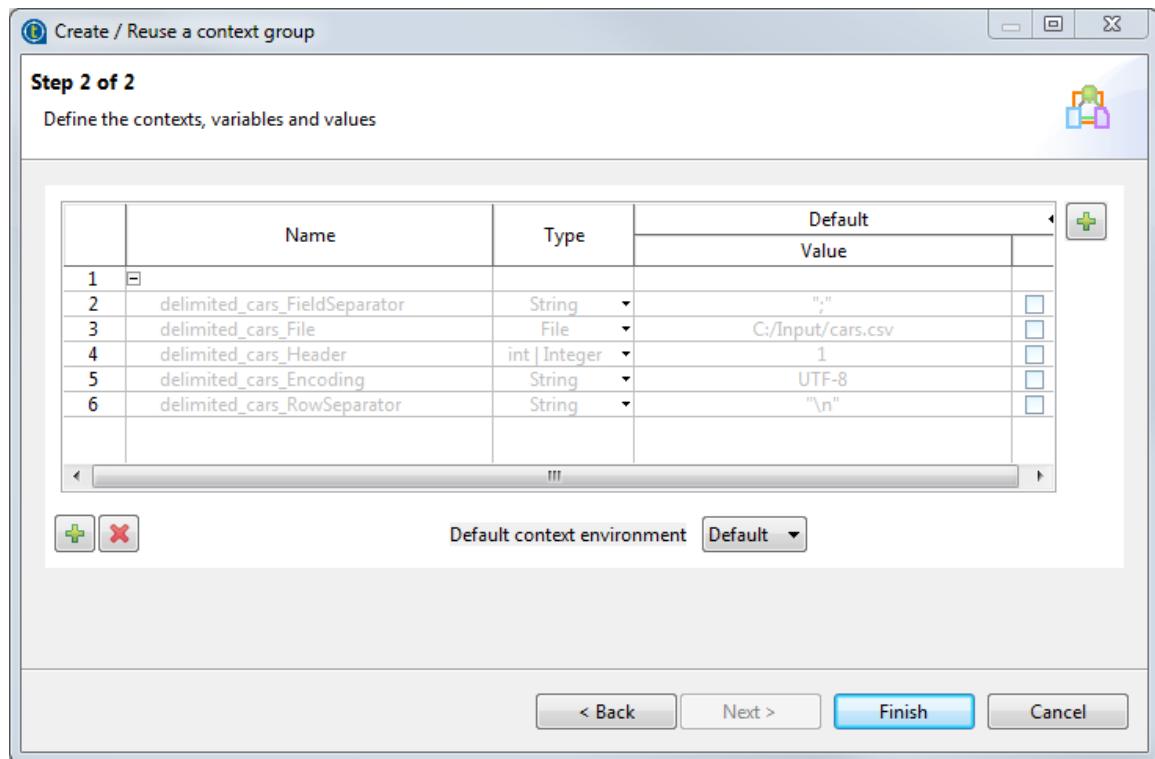
- Check the context group generation result.

To edit the context variables, go to the **Contexts** node of the Repository, right-click the newly created context group, and select **Edit context group** to open the **[Create / Edit a context group]** wizard after the connection wizard is closed.

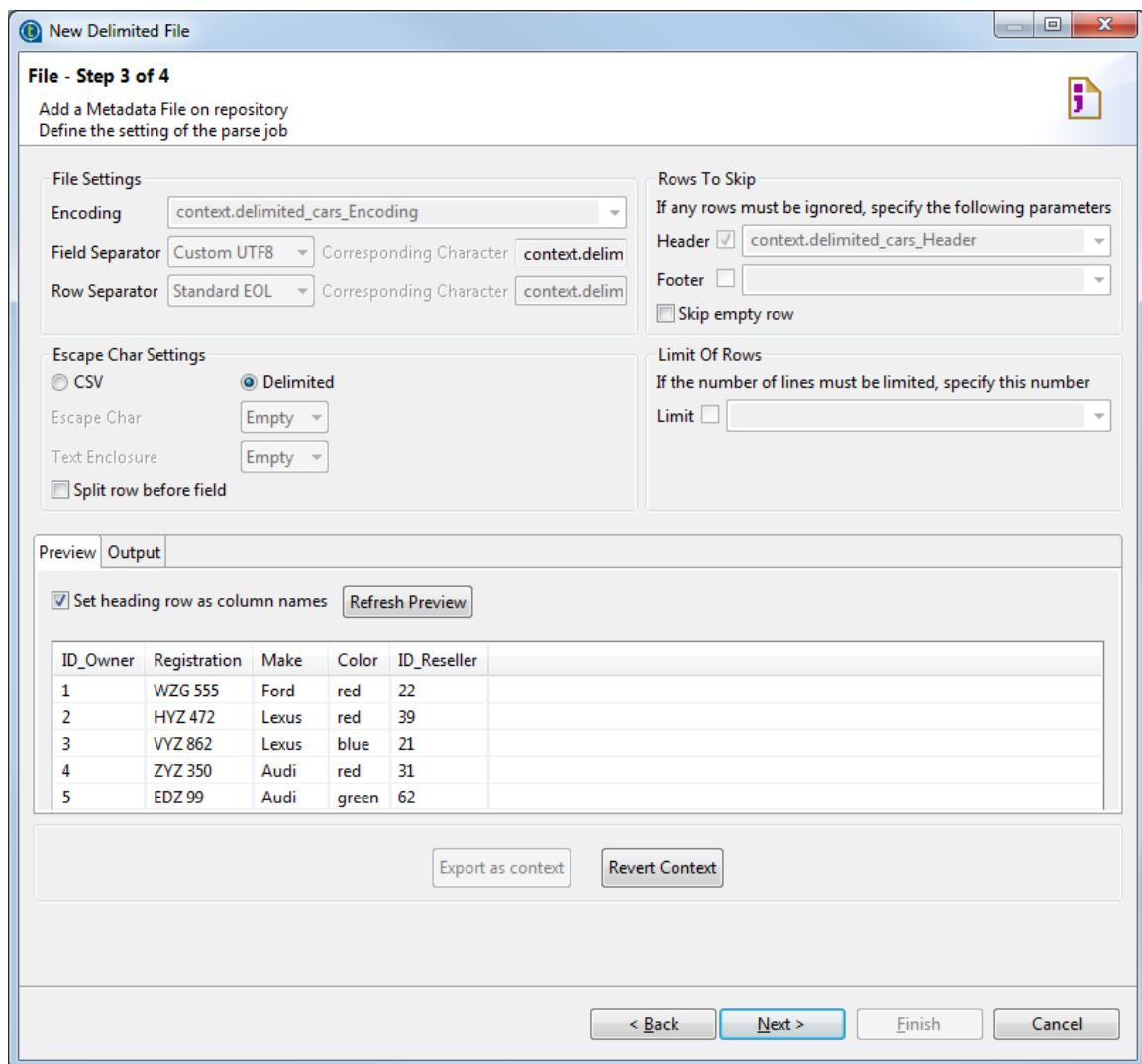
To edit the default context, or add new contexts, click the **[+]** button at the upper right corner of the wizard.

To add a new context variable, click the **[+]** button at the bottom of the wizard.

For more information on handling contexts and variables, see [Using contexts and variables](#).



6. Click **Finish** to complete context creation and return to the connection wizard.



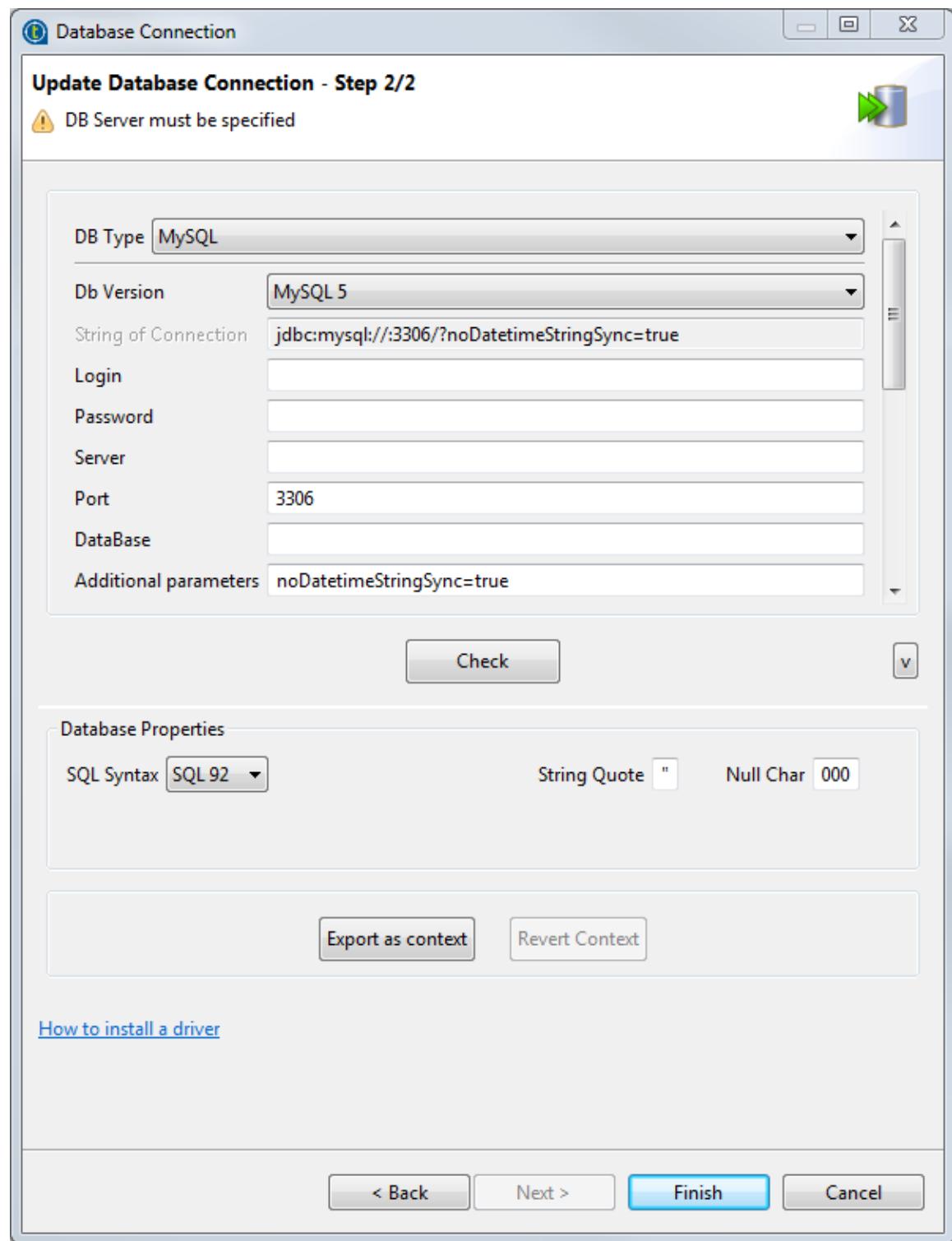
The relevant connection details fields in the wizard are set with the context variables.

To unset the connection details, click the **Revert Context** button.

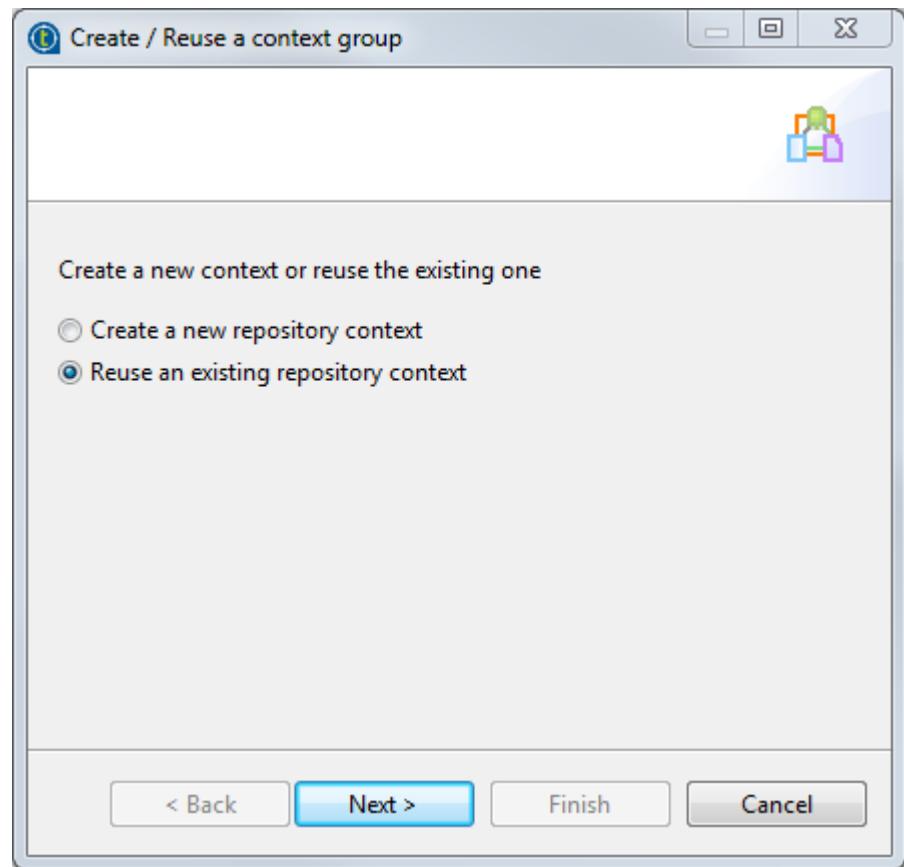
## 7.21.2. How to use variables of an existing context group to set up a connection

To use variables of an existing context group centrally stored in the Repository to set up a connection, follow the steps below:

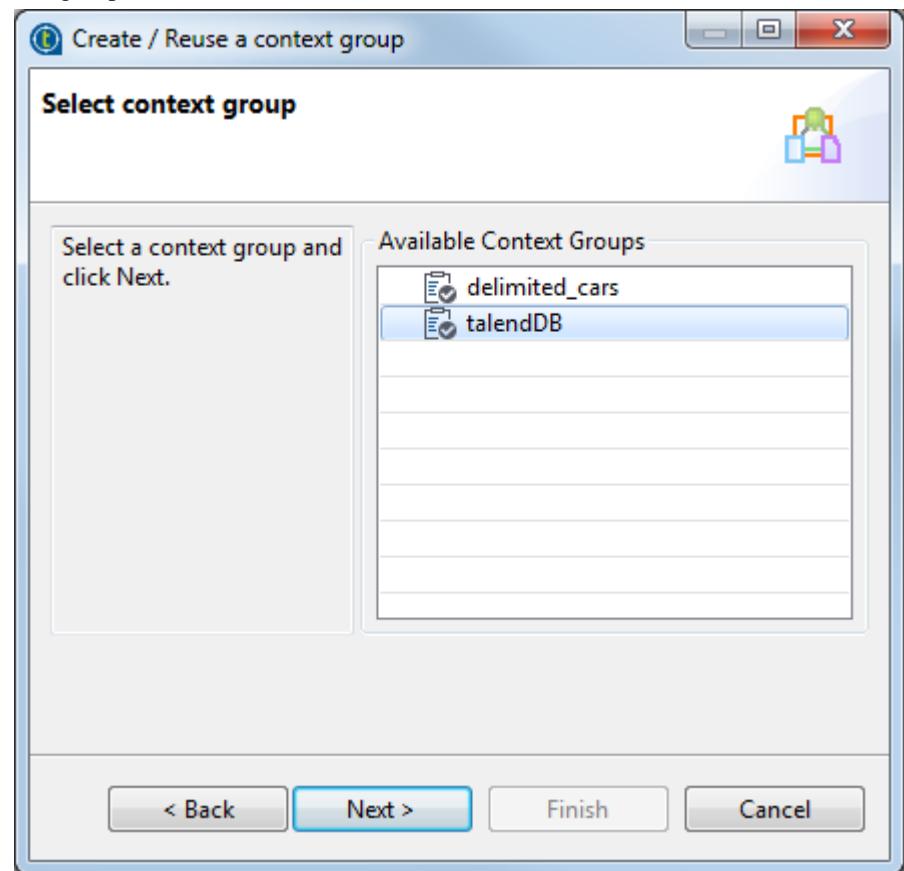
1. When creating or editing a metadata connection in the wizard, click **Export as context**.



2. In the [Create / Reuse a context group] wizard that opens, select **Reuse an existing repository context** and click **Next**.

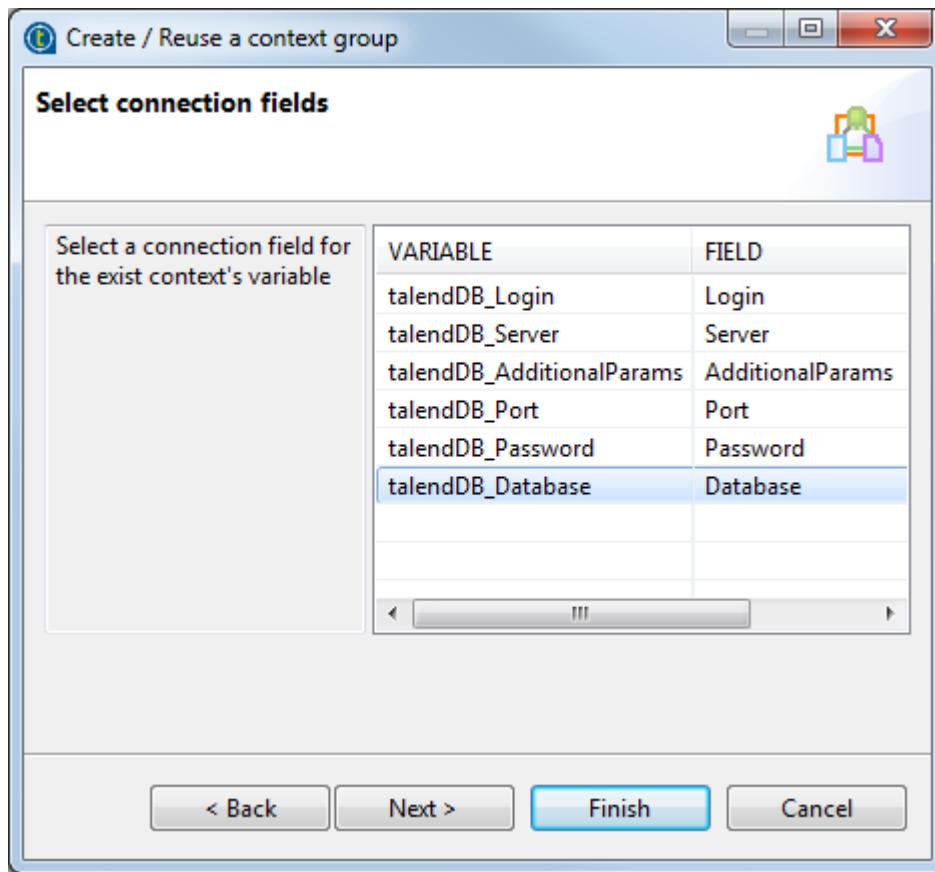


3. Select a context group from the list and click **Next**.



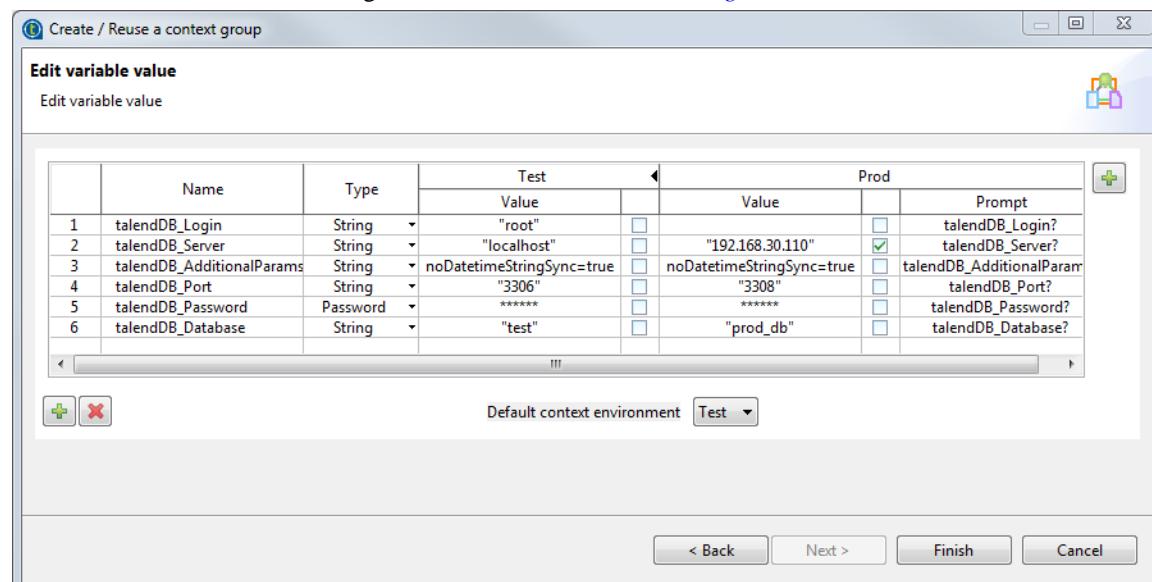
4. For each variable, select the corresponding field of the connection details, and then click **Next** to view and edit the context variables, or click **Finish** to show the connection setup result directly.

In this example, click **Next**.

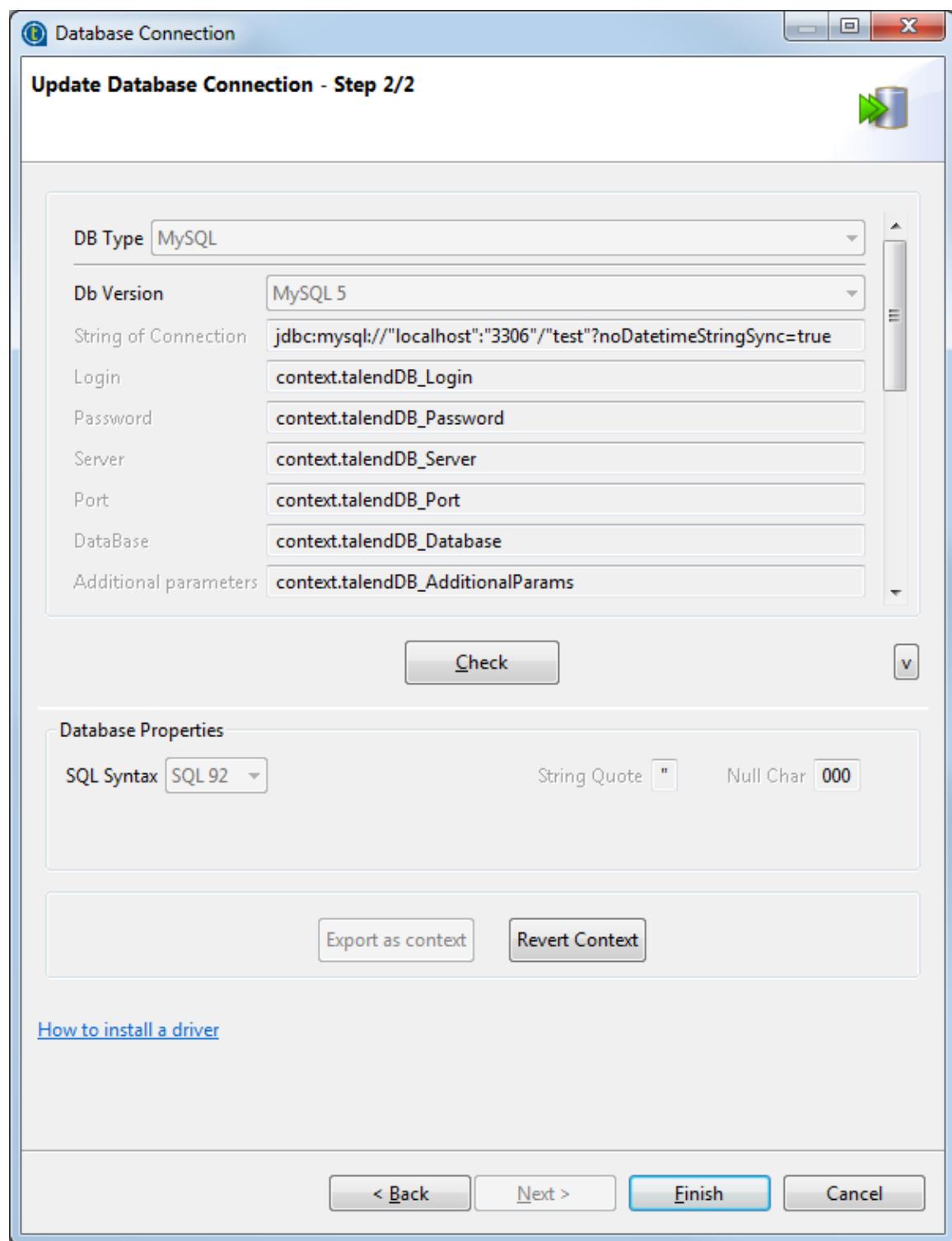


5. Edit the contexts and/or context variables if needed. If you make any changes, your centralized context group will be updated automatically.

For more information on handling contexts and variables, see [Using contexts and variables](#).



6. Click **Finish** to validate context reuse and return to the connection wizard.



The relevant connection details fields in the wizard are set with the context variables.

To unset the connection details, click the **Revert Context** button.

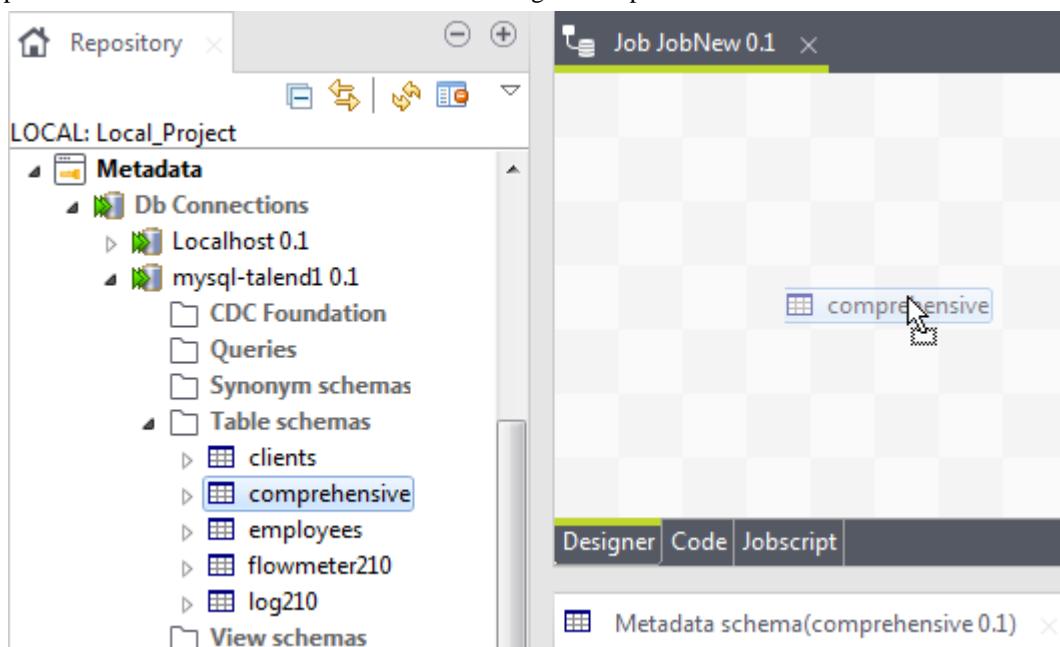
## 7.22. How to use centralized metadata in a Job

For recurrent use of files and database connections in various Jobs, we recommend you to store the connection and schema metadata in the **Repository** tree view under the **Metadata** node. Different folders under the **Metadata** node will group the established connections including those to databases, files and systems.

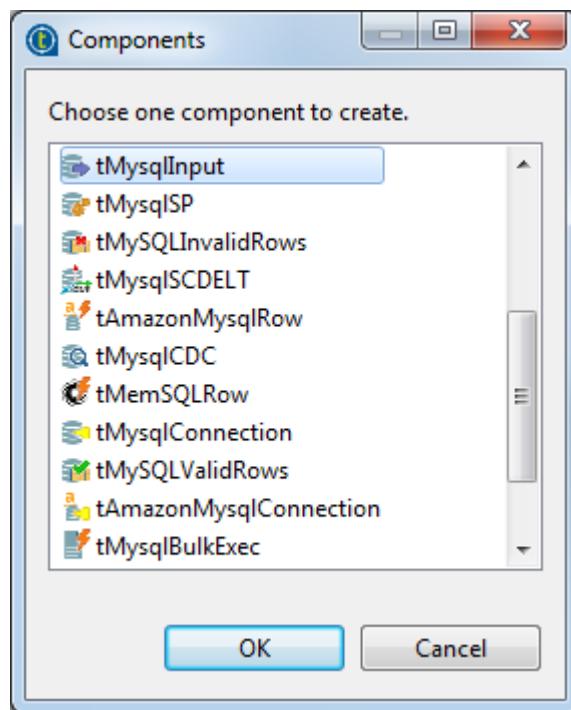
Different wizards will help you centralize connection and schema metadata in the **Repository** tree view. For more information about the **[Metadata Manager]** wizards, see [Managing Metadata](#).

Once the relevant metadata is stored under the **Metadata** node, you will be able to drop the corresponding components directly onto the design workspace.

1. In the **Repository** tree view of the **Integration** perspective, expand **Metadata** and the folder holding the connection you want to use in your Job.
2. Drop the relevant connection or schema onto the design workspace.



A dialog box prompts you to select the component you want to use among those offered.



3. Select the component and then click **OK**. The selected component displays on the design workspace.

Alternatively, according to the type of component (Input or Output) you want to use, perform one of the following operations:

- **Output:** Press **Ctrl** on your keyboard while you are dropping the component onto the design workspace to directly include it in the active Job.
- **Input:** Press **Alt** on your keyboard while you drop the component onto the design workspace to directly include it in the active Job.

If you double-click the component, the **Component** view shows the selected connection details as well as the selected schema information.



If you select the connection without selecting a schema, then the properties will be filled with the first encountered schema.





## Chapter 8. Managing routines

This chapter defines the routines, along with user scenarios, and explains how to create your own routines or how to customize the system routines. The chapter also gives an overview of the main routines and use cases of them. To have an overview of the most commonly used routines and other use cases, see [System routines](#).

## 8.1. What are routines

Routines are fairly complex Java functions, generally used to factorize code. They therefore optimize data processing and improve Job capacities.

You can also use the **Repository** tree view to store frequently used parts of code or extract parts of existing company functions, by calling them via the routines. This factorization makes it easier to resolve any problems which may arise and allows you to update the code used in multiple Jobs quickly and easily.

On top of this, certain system routines adopt the most common Java methods, using the **Talend** syntax. This allows you to escalate Java errors in the studio directly, thereby facilitating the identification and resolution of problems which may arise as your integration processes evolve with **Talend**.

There are two types of routines:

- System routines: a number of system routines are provided. They are classed according to the type of data which they process: numerical, string, date...
- User routines: these are routines which you have created or adapted from existing routines.



You do not need any knowledge of the Java language to create and use **Talend** routines.

All of the routines are stored under **Code > Routines** in the **Repository** tree view.

For further information concerning the system routines, see [\*Accessing the System Routines\*](#).

For further information about how to create user routines, see [\*How to create user routines\*](#).



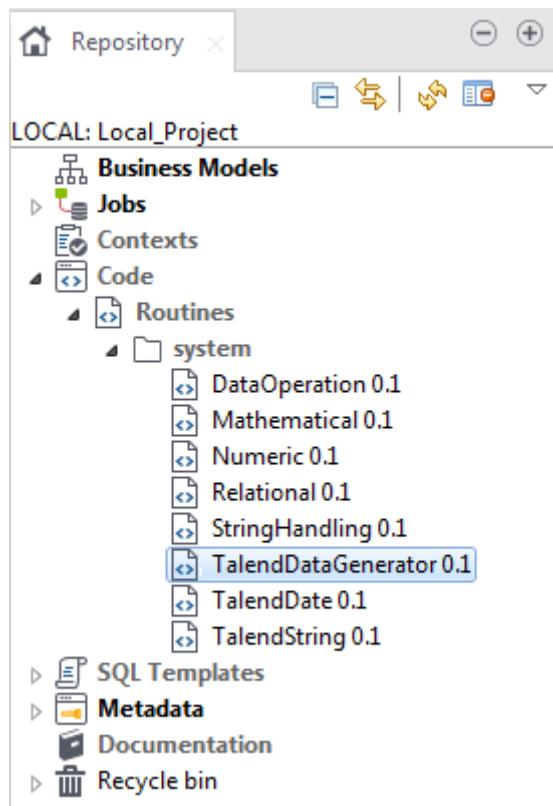
You can also set up routine dependencies on Jobs. To do so, simply right click a Job on the **Repository** tree view and select **Set up routine dependencies**. In the dialog box which opens, all routines are set by default. You can use the tool bar to remove routines if required.

## 8.2. Accessing the System Routines

To access the system routines, click **Code > Routines > system**. The routines or functions are classed according to their usage.



The **system** folder and its content are read only.



Each class or category in the system folder contains several routines or functions. Double-click the class that you want to open.

All of the routines or functions within a class are composed of some descriptive text, followed by the corresponding Java code. In the Routines view, you can use the scrollbar to browse the different routines. Or alternatively:

1. Press **Ctrl+O** in the routines view.

A dialog box displays a list of the different routines in the category.

2. Click the routine of interest.

The view jumps to the section comprising the routine's descriptive text and corresponding code.



The syntax of routine call statements is case sensitive.

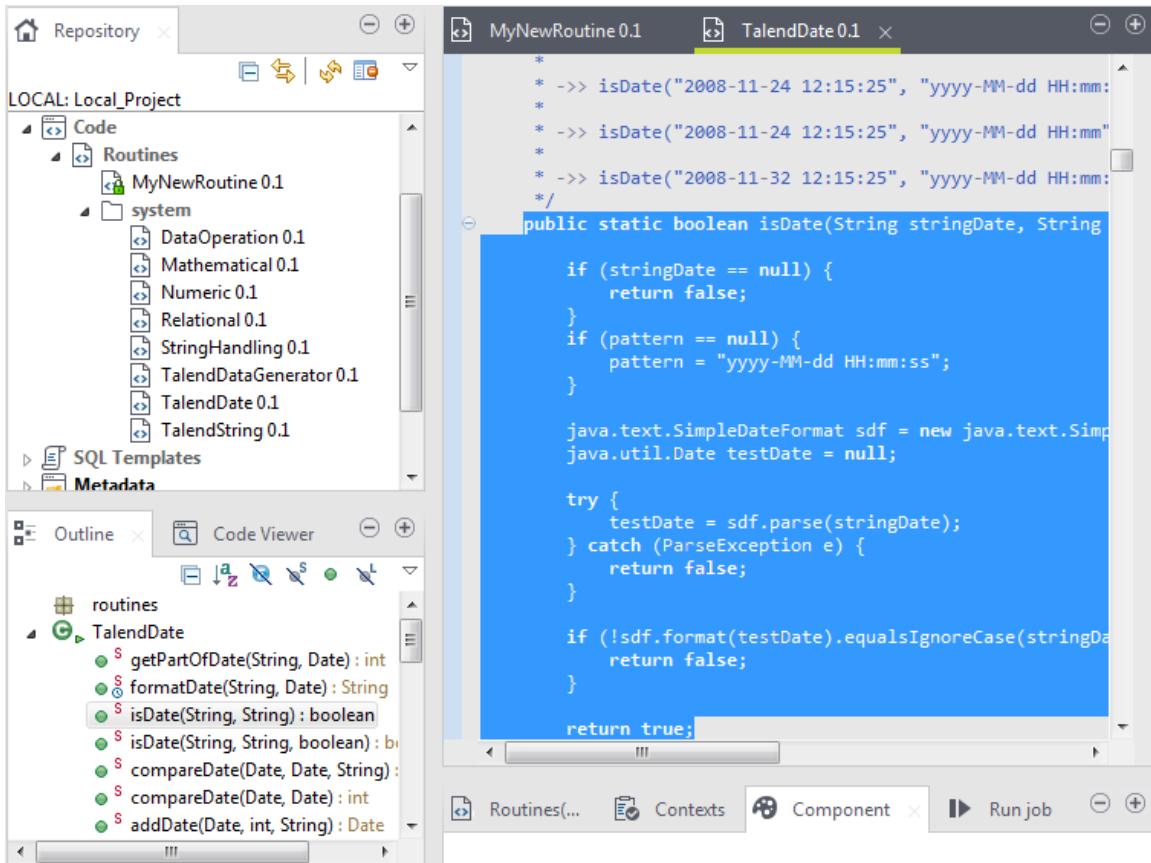
## 8.3. Customizing the system routines

If the system routines are not adapted to your specific needs, you can customize them by copying and pasting the content in a user routine, then modify the content accordingly.

To customize a system routine:

1. First of all, create a user routine by following the steps outlined in [How to create user routines](#). The routine opens in the workspace, where you shall find a basic example of a routine.
2. Then, under **Code > Routines > system**, select the class of routines which contains the routine(s) you want to customize.

3. Double-click the class which contains the relevant routine to open it in the workspace.
4. Use the **Outline** panel on the bottom left of the studio to locate the routine from which you want to copy all or part of the content.



5. In the workspace, select all or part of the code and copy it using **Ctrl+C**.
6. Click the tab to access your user routine and paste the code by pressing **Ctrl+V**.
7. Modify the code as required and press **Ctrl+S** to save it.

We advise you to use the descriptive text (in blue) to detail the input and output parameters. This will make your routines easier to maintain and reuse.

## 8.4. Managing user routines

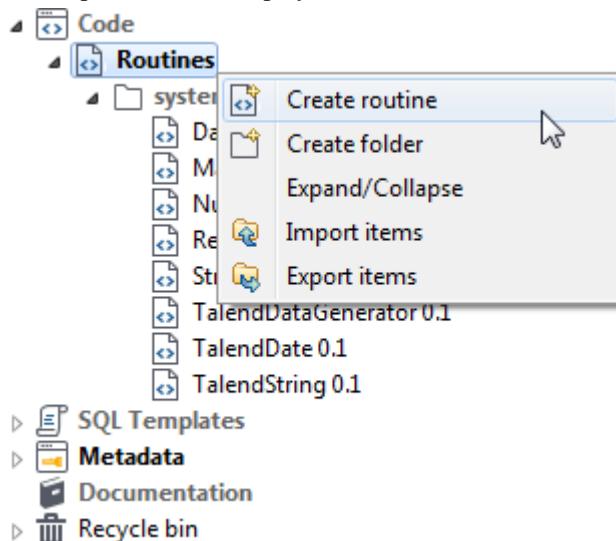
*Talend Studio* allows you to create user routines, to modify them or to modify system routines, in order to fill your specific needs.

### 8.4.1. How to create user routines

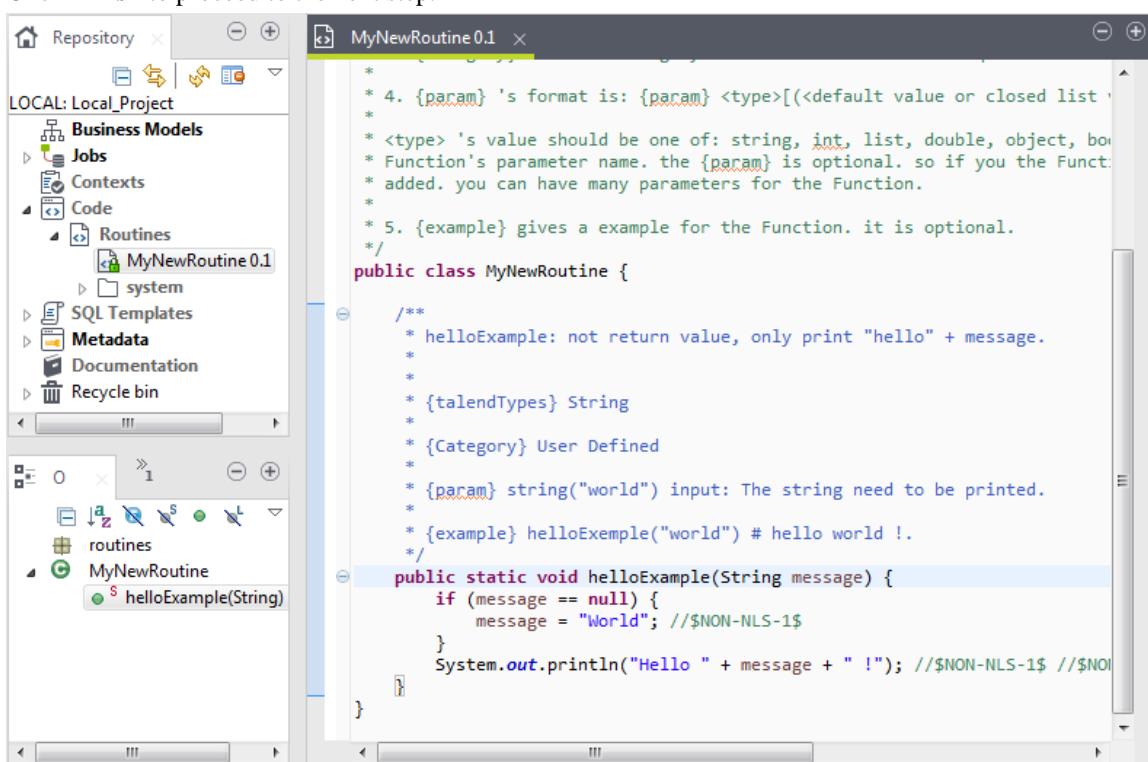
You can create your own routines according to your particular factorization needs. Like the system routines, the user routines are stored in the **Repository** tree view under **Code > Routines**. You can add folders to help organize your routines and call them easily in any of your Jobs.

To create a new user routine, complete the following:

1. In the **Repository** tree view, expand **Code** to display the **Routines** folder.



2. Right-click **Routines** and select **Create routine**.
3. The **[New routine]** dialog box opens. Enter the information required to create the routine, ie., its name, description...
4. Click **Finish** to proceed to the next step.



The newly created routine appears in the **Repository** tree view, directly below the **Routines** node. The routine editor opens to reveal a model routine which contains a simple example, by default, comprising descriptive text in blue, followed by the corresponding code.



We advise you to add a very detailed description of the routine. The description should generally include the input and output parameters you would expect to use in the routine, as well as the results returned along with an example. This information tends to be useful for collaborative work and the maintenance of the routines.

The following example of code is provided by default:

```
public static void helloExample(String message) {
 if (message == null) {
 message = "World"; //NON-NLS-1$
 }
 System.out.println("Hello " + message + " !");
```

5. Modify or replace the model with your own code and press **Ctrl+S** to save the routine. Otherwise, the routine is saved automatically when you close it.

 You can copy all or part of a system routine or class and use it in a user routine by using the **Ctrl+C** and **Ctrl+V** commands, then adapt the code according to your needs. For further information about how to customize routines, see [Customizing the system routines](#).

## 8.4.2. How to edit user routines

You can modify the user routines whenever you like.

 The **system** folder and all of the routines held within are read only.

To edit your user routines:

1. Right click the routine you want to edit and select **Edit Routine**.
2. The routine opens in the workspace, where you can modify it.
3. Once you have adapted the routine to suit your needs, press **Ctrl+S** to save it.

If you want to reuse a system routine for your own specific needs, see [Customizing the system routines](#).

## 8.4.3. How to edit user routine libraries

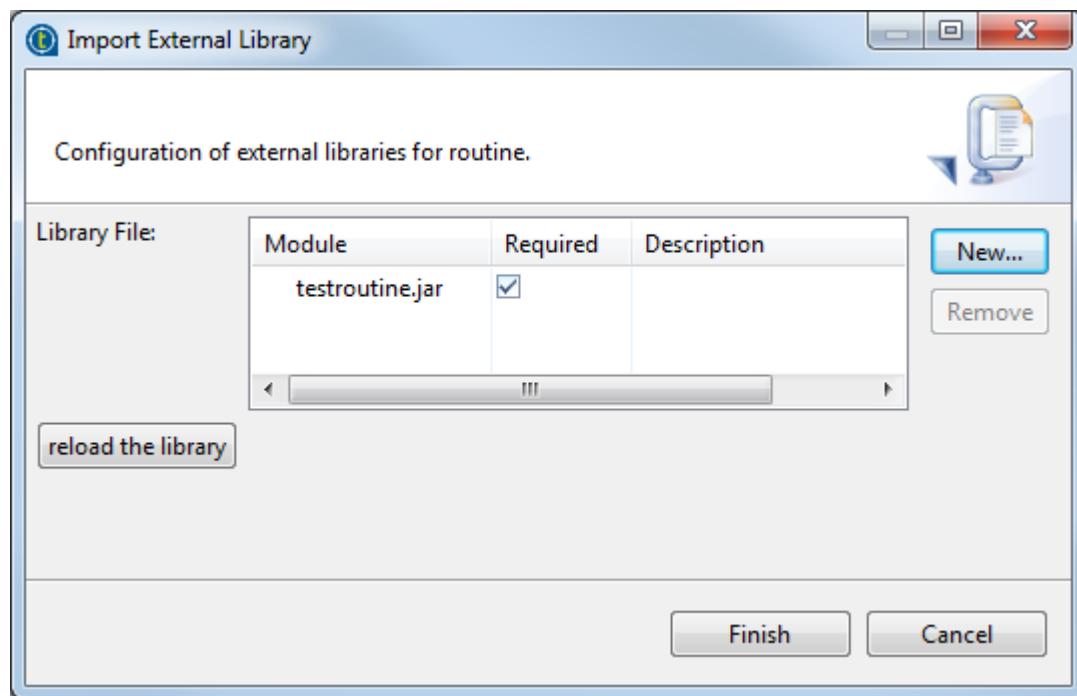
You can edit the library of any of the user routines by importing external .jar files for the selected routine. These external library files will be listed, like modules, in the **Modules** view in your current Studio. For more information on the **Modules** view, see the *Talend Installation and Upgrade Guide*.

The .jar file of the imported library will be also listed in the library file of your current Studio.

To edit a user routine library, complete the following:

1. If the library to be imported isn't available on your machine, either download and install it using the **Modules** view or download and store it in a local directory.
2. In the **Repository** tree view, expand **Code > Routines**.
3. Right-click the user routine you want to edit its library and then select **Edit Routine Library**.

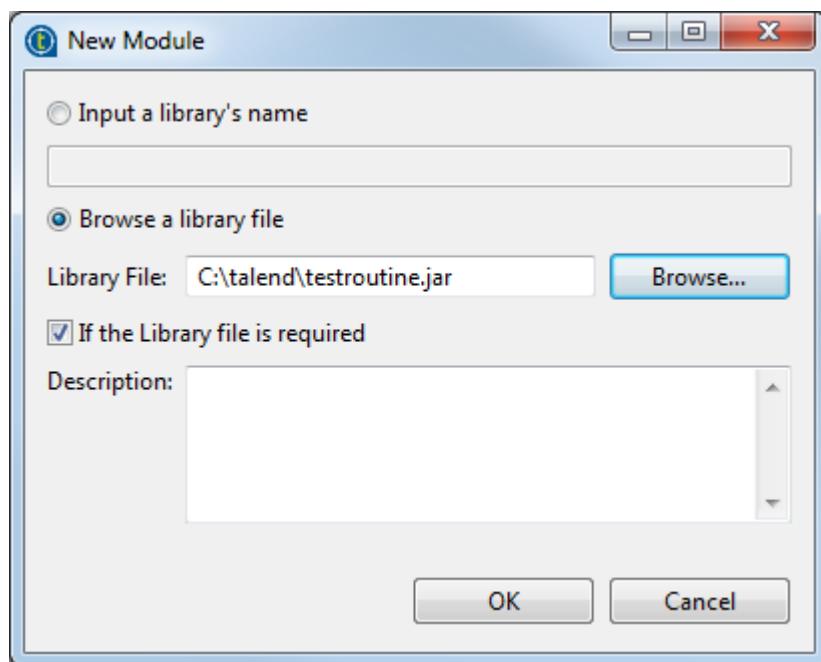
The **[Import External Library]** dialog box displays.



4. Click **New** to open the **[New Module]** dialog box where you can import the external library.



You can delete any of the already imported routine files if you select the file in the **Library File** list and click the **Remove** button.



5. Specify the library file to be imported:

- If you have installed the library using the **Modules** view, enter the full name of the library file in the **Input a library's name** field.
- If you have stored the library file in a local directory, select the **Browse a library file** option and click **browse** to set the file path in the corresponding field.

6. If required, enter a description in the **Description** field.

7. Click **OK** to confirm your changes.

The imported library file is listed in the **Library File** list in the **[Import External Library]** dialog box.

8. Click **Finish** to close the dialog box.

The library file is imported into the library folder of your current Studio and also listed in the **Module** view of the same Studio. You may need to restart your Studio to bring the external library into effect.

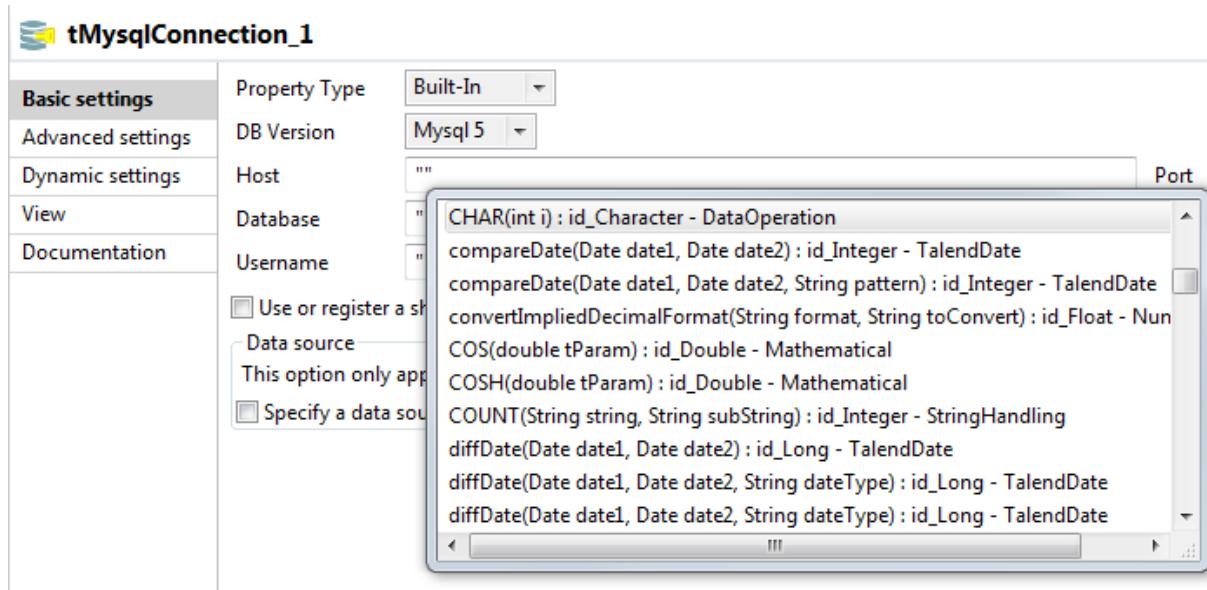
For more information about the **Modules** view, see the *Talend Installation and Upgrade Guide*.

## 8.5. Calling a routine from a Job

Pre-requisite: You must have at least one Job created, in order to run a routine. For further information regarding how to create a Job, see [Creating a Job](#).

You can call any of your user and system routines from your Job components in order to run them at the same time as your Job.

To access all the routines saved in the **Routines** folder in the **Repository** tree view, press **Ctrl+Space** in any of the fields in the **Basic settings** view of any of the **Talend** components used in your Job and select the one you want to run.



Alternatively, you can call any of these routines by indicating the relevant class name and the name of the routine, followed by the expected settings, in any of the **Basic settings** fields in the following way:

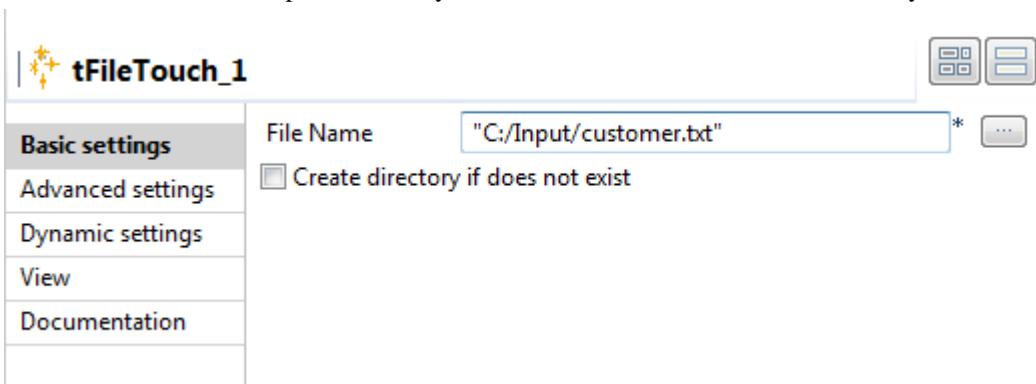
```
<ClassName>. <RoutineName>
```

## 8.6. Use case: Creating a file for the current date

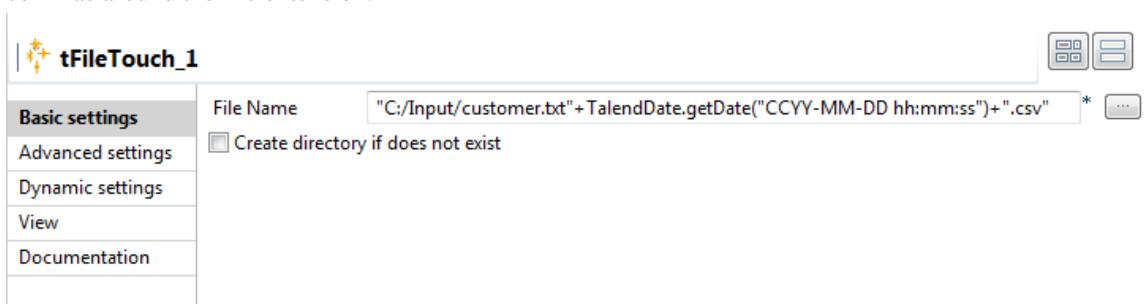
This scenario describes how to use a routine. The Job uses just one component, which calls a system routine.



1. In the **Palette**, click **File > Management**, then drop a **tFileTouch** component onto the workspace. This component allows you to create an empty file.
2. Double-click the component to open its **Basic settings** view in the **Component** tab.
3. In the **FileName** field, enter the path to access your file, or click [...] and browse the directory to locate the file.



4. Close the double inverted commas around your file extension as follows: "*D:/Input/customer".txt*".
5. Add the plus symbol (+) between the closing inverted commas and the file extension.
6. Press **Ctrl+Space** to open a list of all of the routines, and in the auto-completion list which appears, select *TalendDate.getDate* to use the **Talend** routine which allows you to obtain the current date.
7. Modify the format of the date provided by default, if required.
8. Enter the plus symbol (+) next to the *getDate* variable to complete the routine call, and place double inverted commas around the file extension.



If you are working on windows, the ":" between the hours and minutes and between the minutes and seconds must be removed.

9. Press **F6** to run the Job.

The **tFileTouch** component creates an empty file with the days date, retrieved upon execution of the *GetDate* routine called.







## Chapter 9. Using SQL templates

SQL templates are groups of pre-defined query arguments that run in the ELT mode. This chapter explains the ELT mode, defines the SQL templates and provides user scenarios to explain how to use the SQL templates or how to create your own ones.

## 9.1. What is ELT

Extract, Load and Transform (ELT) is a data manipulation process in database usage, especially in data warehousing. Different from the traditional ETL (Extract, Transform, Load) mode, in ELT, data is extracted, loaded into the database and then is transformed where it sits in the database, prior to use. This data is migrated in bulk according to the data set and the transformation process occurs after the data has been loaded into the targeted DBMS in its raw format. This way, less stress is placed on the network and larger throughput is gained.

However, the ELT mode is certainly not optimal for all situations, for example,

- As SQL is less powerful than Java, the scope of available data transformations is limited.
- ELT requires users that have high proficiency in SQL tuning and DBMS tuning.
- Using ELT with *Talend Studio*, you cannot pass or reject one single row of data as you can do in ETL. For more information about row rejection, see [Row connection](#).

Based on the advantages and disadvantages of ELT, the SQL templates are designed as the ELT facilitation requires.

## 9.2. Introducing Talend SQL templates

SQL is a standardized query language used to access and manage information in databases. Its scope includes data query and update, schema creation and modification, and data access control. *Talend Studio* provides a range of SQL templates to simplify the most common tasks. It also comprises a SQL editor which allows you to customize or design your own SQL templates to meet less common requirements.

These SQL templates are used with the components from the **Talend** ELT component family including **tSQLTemplate**, **tSQLTemplateFilterColumns**, **tSQLTemplateCommit**, **tSQLTemplateFilterRows**, **tSQLTemplateRollback**, **tSQLTemplateAggregate** and **tSQLTemplateMerge**. These components execute the selected SQL statements. Using the UNION, EXCEPT and INTERSECT operators, you can modify data directly on the DBMS without using the system memory.

Moreover, with the help of these SQL templates, you can optimize the efficiency of your database management system by storing and retrieving your data according to the structural requirements.

*Talend Studio* provides the following types of SQL templates under the **SQL templates** node in the **Repository** tree view:

- System SQL templates: They are classified according to the type of database for which they are tailored.
- User-defined SQL templates: these are templates which you have created or adapted from existing templates.

More detailed information about the SQL templates is presented in the below sections.



As most of the SQL templates are tailored for specific databases, if you change database in your system, it is inevitable to switch to or develop new templates for the new database.

## 9.3. Managing Talend SQL templates

*Talend Studio* enables you via the **SQL Templates** folder in the **Repository** tree view to use system or user-defined SQL templates in the Jobs you create in the Studio using the ELT components.

The below sections show you how to manage these two types of SQL templates.

## 9.3.1. Types of system SQL templates

This section gives detail information related to the different types of the pre-defined SQL templates.

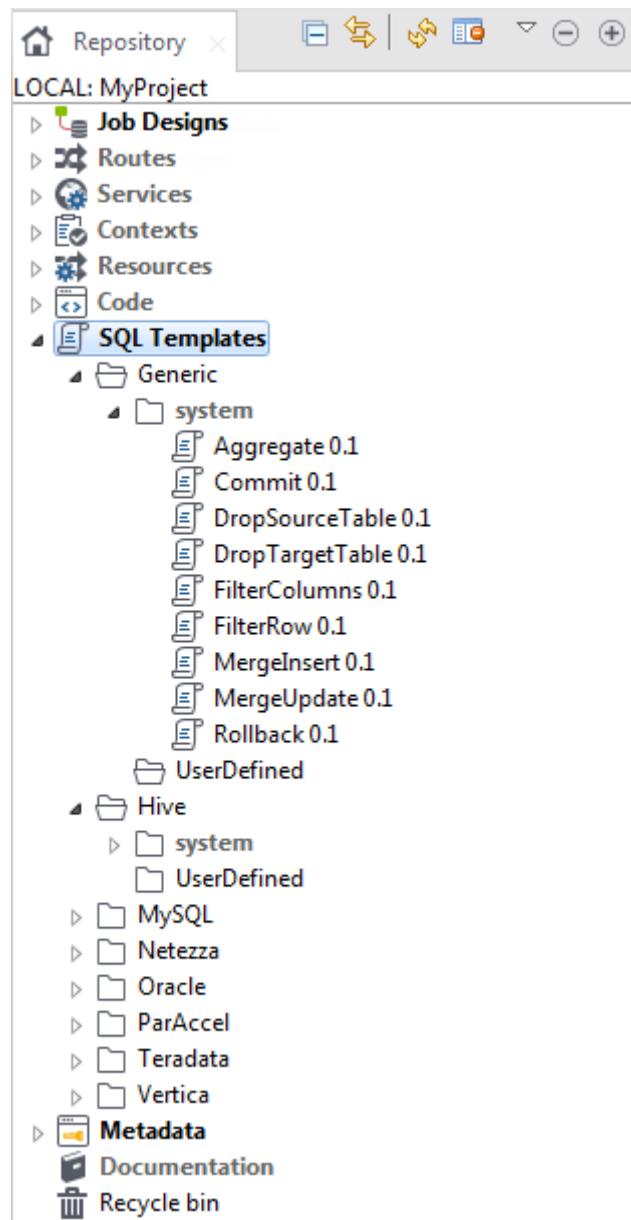
Even though the statements of each group of templates vary from database to database, according to the operations they are intended to accomplish, they are also grouped on the basis of their types in each folder.

The below table provides these types and their related information.

Name	Function	Associated components	Required component parameters
<i>Aggregate</i>	Realizes aggregation (sum, average, count, etc.) over a set of data.	<code>tSQLTemplateAggregate</code>	Database name Source table name Target table name
<i>Commit</i>	Sends a Commit instruction to RDBMS.	<code>tSQLTemplate</code> <code>tSQLTemplateAggregate</code> <code>tSQLTemplateCommit</code> <code>tSQLTemplateFilterColumns</code> <code>tSQLTemplateFilterRows</code> <code>tSQLTemplateMerge</code> <code>tSQLTemplateRollback</code>	Null
<i>Rollback</i>	Sends a Rollback instruction to RDBMS.	<code>tSQLTemplate</code> <code>tSQLTemplateAggregate</code> <code>tSQLTemplateCommit</code> <code>tSQLTemplateFilterColumns</code> <code>tSQLTemplateFilterRows</code> <code>tSQLTemplateMerge</code> <code>tSQLTemplateRollback</code>	Null
<i>DropSourceTable</i>	Removes a source table.	<code>tSQLTemplate</code> <code>tSQLTemplateAggregate</code> <code>tSQLTemplateFilterColumns</code> <code>tSQLTemplateFilterRows</code>	Table name (when use <code>tSQLTemplate</code> ) Source table name
<i>DropTargetTable</i>	Removes a target table.	<code>tSQLTemplateAggregate</code> <code>tSQLTemplateFilterColumns</code> <code>tSQLTemplateFilterRows</code>	Target table name
<i>FilterColumns</i>	Selects and extracts a set of data from given columns in RDBMS.	<code>tSQLTemplateAggregate</code> <code>tSQLTemplateFilterColumns</code> <code>tSQLTemplateFilterRows</code>	Target table name (and schema) Source table name (and schema)
<i>FilterRow</i>	Selects and extracts a set of data from given rows in RDBMS.	<code>tSQLTemplateAggregate</code> <code>tSQLTemplateFilterColumns</code> <code>tSQLTemplateFilterRows</code>	Target table name (and schema) Source table name (and schema) Conditions
<i>MergeInsert</i>	Inserts records from the source table to the target table.	<code>tSQLTemplateMerge</code> <code>tSQLTemplateCommit</code>	Target table name (and schema) Source table name (and schema) Conditions
<i>MergeUpdate</i>	Updates the target table with records from the source table.	<code>tSQLTemplateMerge</code> <code>tSQLTemplateCommit</code>	Target table name (and schema) Source table name (and schema) Conditions

## 9.3.2. How to access a system SQL template

To access a system SQL template, expand the **SQL Templates** node in the **Repository** tree view.



Each folder contains a **system** sub-folder containing pre-defined SQL statements, as well as a **UserDefined** folder in which you can store SQL statements that you have created or customized.

Each system folder contains several types of SQL templates, each designed to accomplish a dedicated task.

Apart from the **Generic** folder, the SQL templates are grouped into different folders according to the type of database for which they are to be used. The templates in the **Generic** folder are standard, for use in any database. You can use these as a basis from which you can develop more specific SQL templates than those defined in *Talend Studio*.

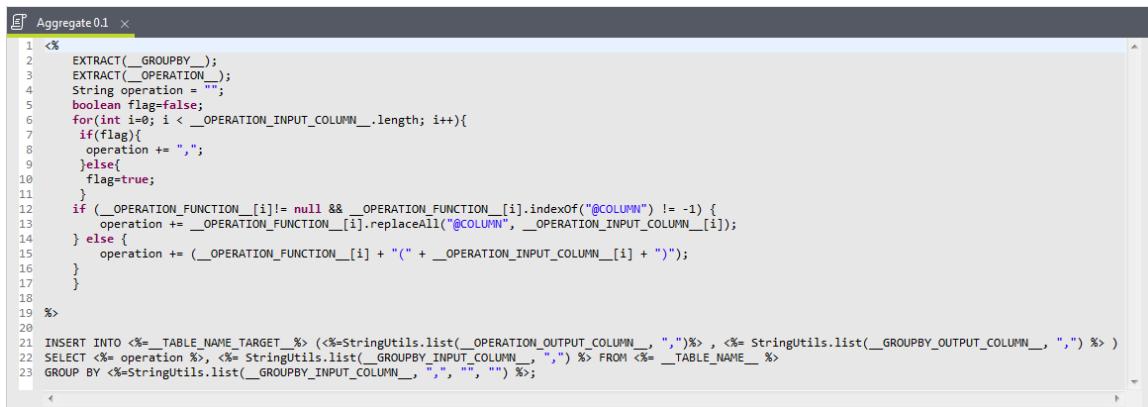


The **system** folders and their content are read only.

From the **Repository** tree view, proceed as follows to open an SQL template:

1. In the **Repository** tree view, expand **SQL Templates** and browse to the template you want to open.
2. Double-click the class that you want to open, for example, *Aggregate* in the **Generic** folder.

The *Aggregate* template view displays in the workspace.



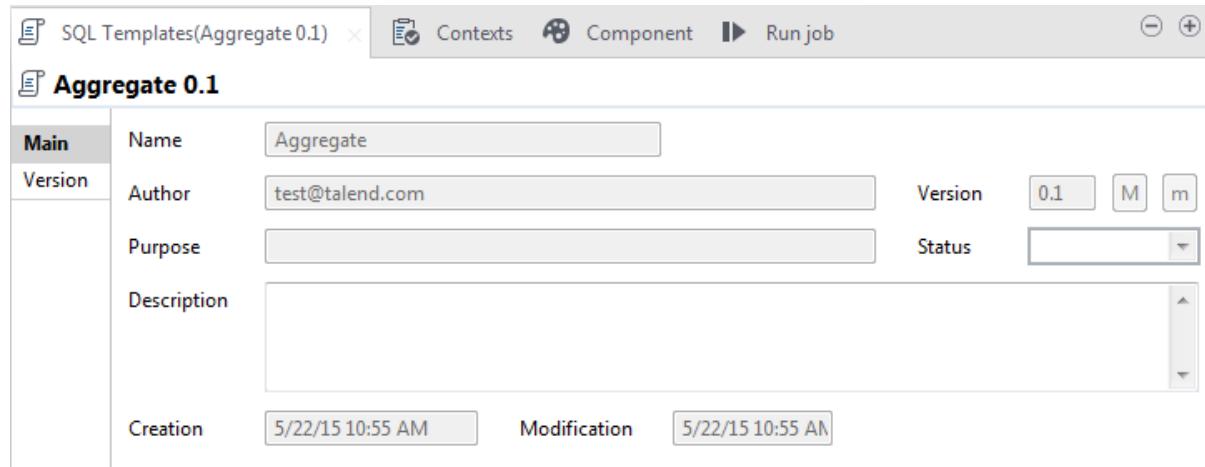
```

1 <%
2 EXTRACT(_GROUPBY_);
3 EXTRACT(_OPERATION_);
4 String operation = "";
5 boolean flag=false;
6 for(int i=0; i < _OPERATION_INPUT_COLUMN_.length; i++){
7 if(flag){
8 operation += ",";
9 }else{
10 flag=true;
11 }
12 if (_OPERATION_FUNCTION_[i]!= null && _OPERATION_FUNCTION_[i].indexOf("@COLUMN") != -1) {
13 operation += _OPERATION_FUNCTION_[i].replaceAll("@COLUMN", _OPERATION_INPUT_COLUMN_[i]);
14 } else {
15 operation += (_OPERATION_FUNCTION_[i] + "(" + _OPERATION_INPUT_COLUMN_[i] + ")");
16 }
17 }
18 %
19
20
21 INSERT INTO <%=_TABLE_NAME_TARGET%> (<%=StringUtils.list(_OPERATION_OUTPUT_COLUMN_, ",")%> , <%= StringUtils.list(_GROUPBY_OUTPUT_COLUMN_, ",") %>)
22 SELECT <%= operation %>, <%= StringUtils.list(_GROUPBY_INPUT_COLUMN_, ",") %> FROM <%= _TABLE_NAME %>
23 GROUP BY <%=StringUtils.list(_GROUPBY_INPUT_COLUMN_, ", ", ",") %>;

```

You can read the predefined *Aggregate* statements in the template view. The parameters, such as `TABLE_NAME_TARGET`, `operation`, are to be defined when you design related Jobs. Then the parameters can be easily set in the associated components, as mentioned in the previous section.

Everytime you click or open an SQL template, its corresponding property view displays at the bottom of the studio. Click the *Aggregate* template, for example, to view its properties as presented below:



For further information regarding the different types of SQL templates, see [Types of system SQL templates](#).

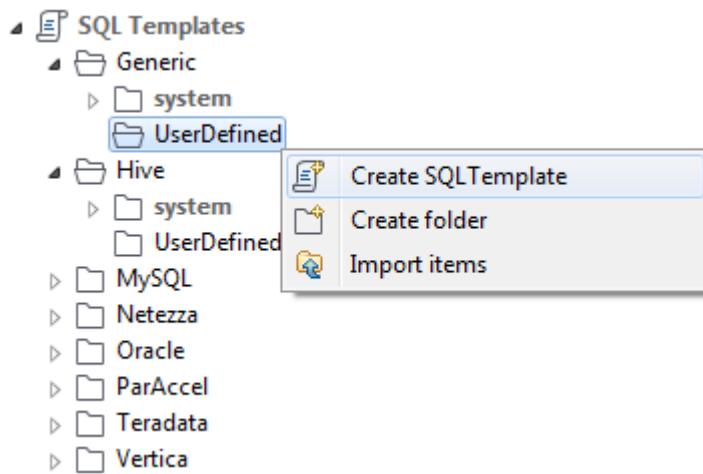
### 9.3.3. How to create user-defined SQL templates

As the transformation you need to accomplish in ELT may exceed the scope of what the given SQL templates can achieve, *Talend Studio* allows you to develop your own SQL templates according to some writing rules. These SQL templates are stored in the **UserDefined** folders grouped according to the database type in which they will be used.

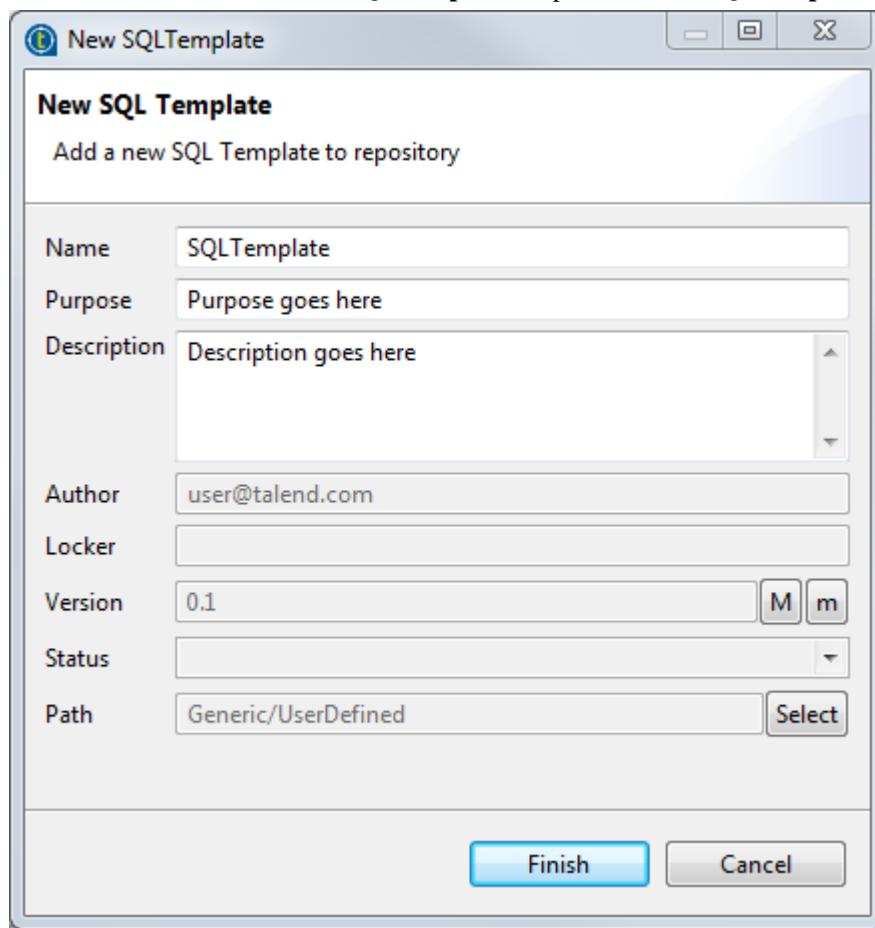
For more information on the SQL template writing rules, see [SQL template writing rules](#).

To create a user-defined SQL template:

- In the **Repository** tree view, expand **SQL Templates** and then the category you want to create the SQL template in.



- Right-click **UserDefined** and select **Create SQLTemplate** to open the [New SQLTemplate] wizard.



- Enter the information required to create the template and click **Finish** to close the wizard.

The name of the newly created template appears under **UserDefined** in the **Repository** tree view. Also, an SQL template editor opens on the design workspace, where you can enter the code for the newly created template.

For further information about how to create a user-defined SQL template and how to use it in a Job, see **tMysqlTableList** at <https://help.talend.com>.

## 9.3.4. A use case of system SQL templates

As there are many common, standardized SQL statements, *Talend Studio* allows you to benefit from various system SQL templates.

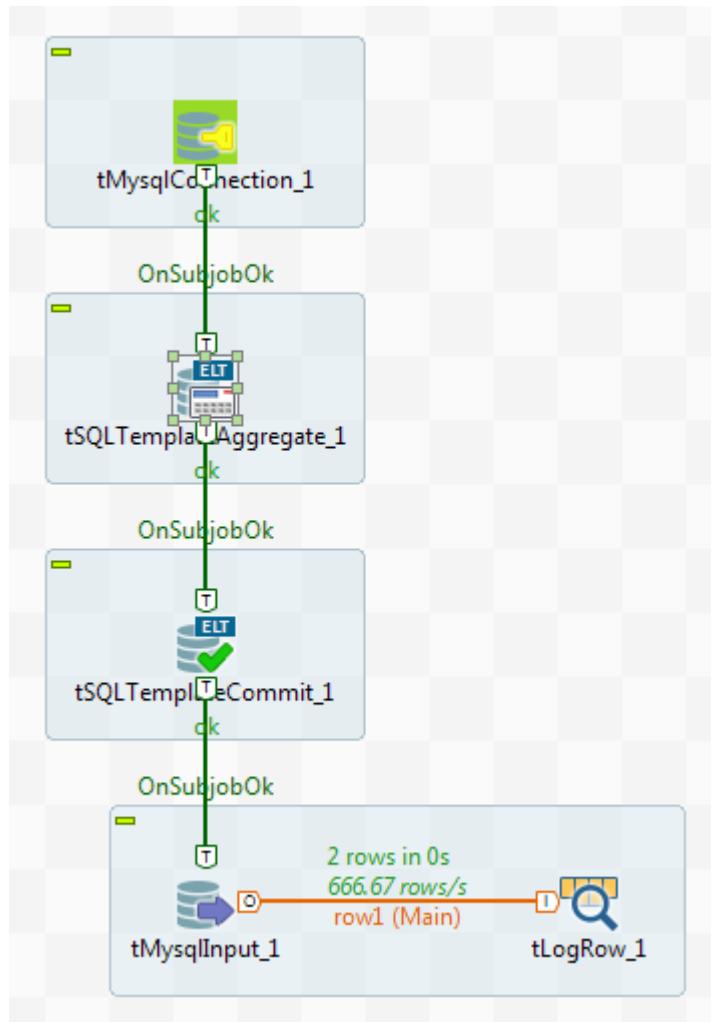
This section presents you with a use case that takes you through the steps of using MySQL system templates in a Job that:

- opens a connection to a Mysql database.
- collects data grouped by specific value(s) from a database table and writes aggregated data in a target database table.
- deletes the source table where the aggregated data comes from.
- reads the target database table and lists the Job execution result.

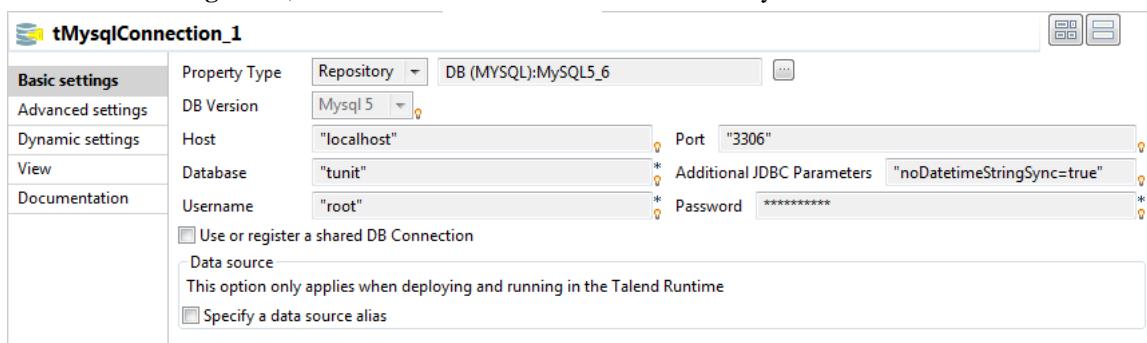
To connect to the database and aggregate the database table columns:

### Configuring a connection to a MySQL database

1. Drop the following components from the **Palette** onto the design workspace: **tMysqlConnection**, **tSQLTemplateAggregate**, **tSQLTemplateCommit**, **tMysqlInput**, and **tLogRow**.
2. Link **tMysqlConnection** to **tSQLTemplateAggregate** using a **Trigger > On Subjob Ok** connection.
3. Do the same to link **tSQLTemplateAggregate** to **tSQLTemplateCommit** and link **tSQLTemplateCommit** to **tMysqlInput**.
4. Link **tMysqlInput** to **tLogRow** using a **Row > Main** connection.



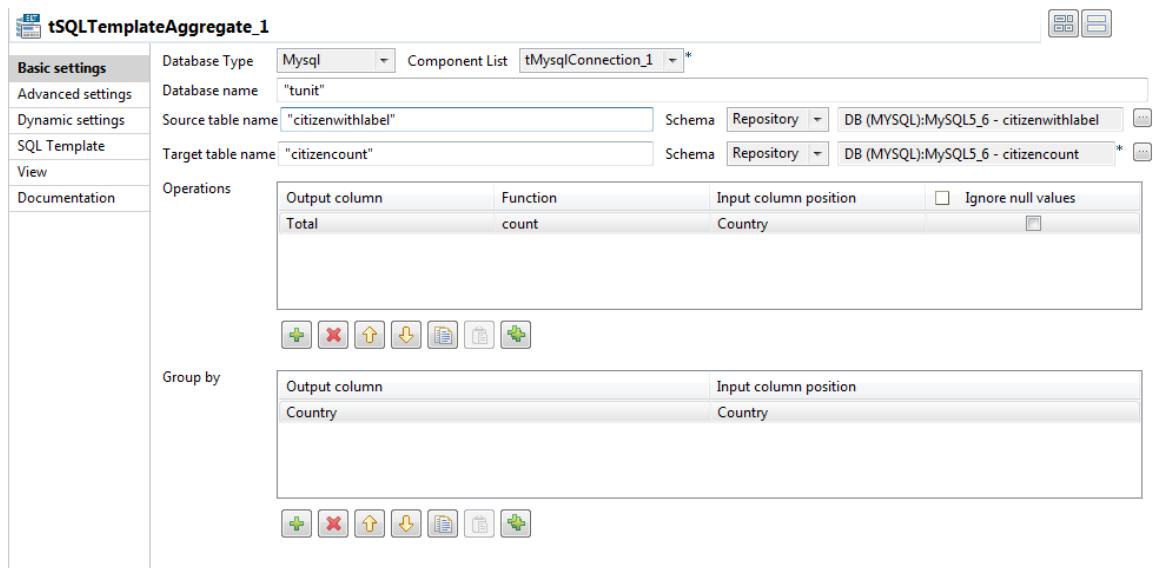
- Double-click **tMysqlConnection** to open its **Basic settings** view.
- In the **Basic settings** view, set the database connection details manually.



- Double-click **tSQLTemplateCommit** to open its **Basic settings** view.
- On the **Database Type** list, select the relevant database type, and from the **Component List**, select the relevant database connection component if more than one connection is used.

## Grouping data, writing aggregated data and dropping the source table

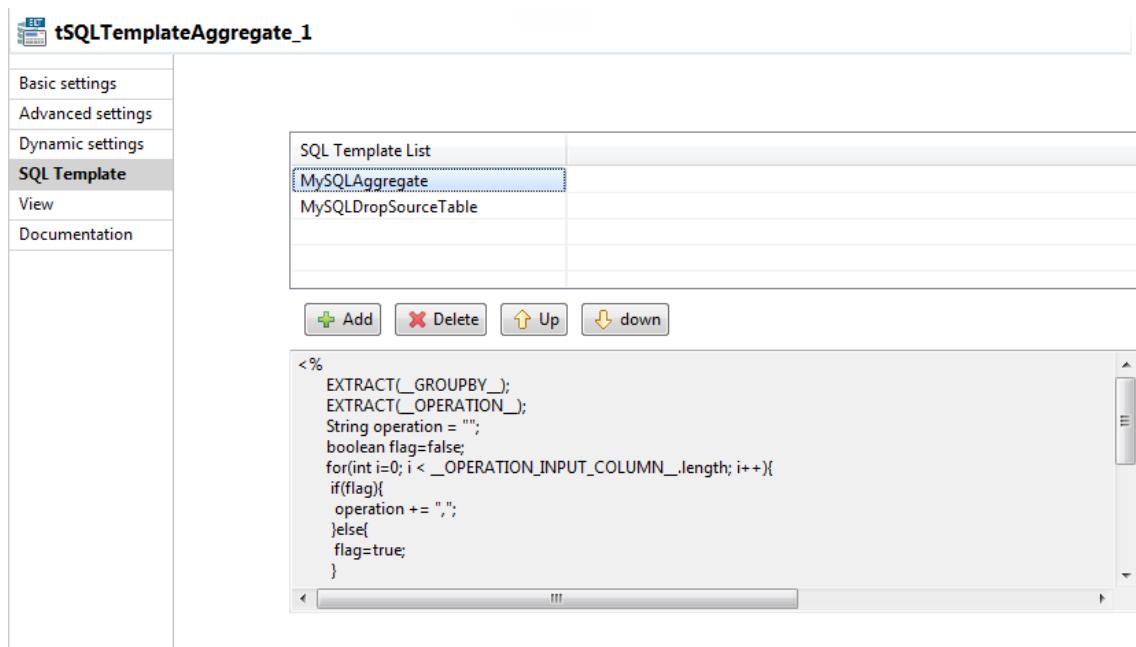
- Double-click **tSQLTemplateAggregate** to open its **Basic settings** view.



2. On the **Database Type** list, select the relevant database type, and from the **Component List**, select the relevant database connection component if more than one connection is used.
3. Enter the names for the database, source table, and target table in the corresponding fields and define the data structure in the source and target tables.

The source table schema consists of three columns: *First\_Name*, *Last\_Name* and *Country*. The target table schema consists of two columns: *country* and *total*. In this example, we want to group citizens by their nationalities and count citizen number in each country. To do that, we define the **Operations** and **Group by** parameters accordingly.

4. In the **Operations** table, click the [+] button to add one or more lines, and then click the **Output column** cell and select the output column that will hold the counted data from the drop-down list.
5. Click the **Function** cell and select the operation to be carried on from the drop-down list.
6. In the **Group by** table, click the [+] button to add one or more lines, and then click the **Output column** cell and select the output column that will hold the aggregated data from the drop-down list.
7. Click the **SQL Template** tab to open the corresponding view.



8. Click the [+] button twice under the **SQL Template List** table to add two SQL templates.
9. Click on the first SQL template row and select the **MySQLAggregate** template from the drop-down list. This template generates the code to aggregate data according to the configuration in the **Basic settings** view.
10. Do the same to select the **MySQLDropSourceTable** template for the second SQL template row. This template generates the code to delete the source table where the data to be aggregated comes from.



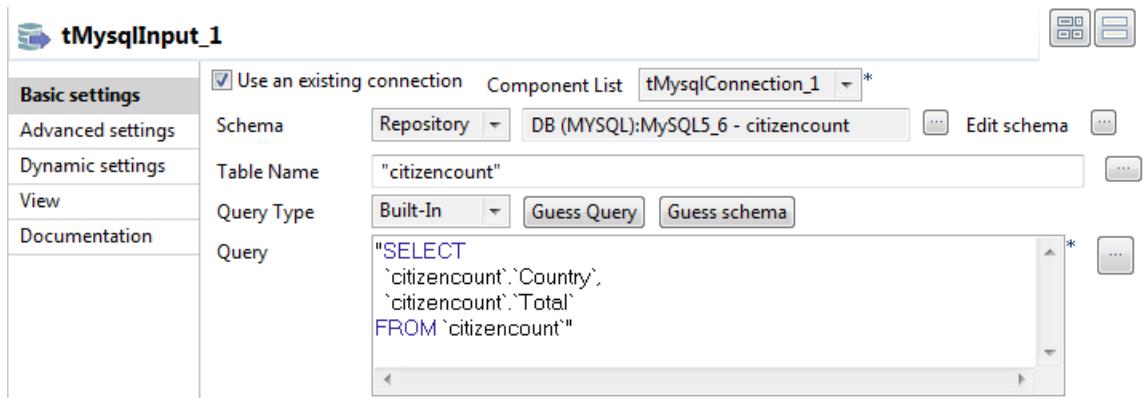
To add new SQL templates to an ELT component for execution, you can simply drop the templates of your choice either onto the component in the design workspace, or onto the component's **SQL Template List** table.



The templates set up in the **SQL Template List** table have priority over the parameters set in the **Basic settings** view and are executed in a top-down order. So in this use case, if you select **MySQLDropSourceTable** for the first template row and **MySQLAggregate** for the second template row, the source table will be deleted prior to aggregation, meaning that nothing will be aggregated.

## Reading the target database and listing the Job execution result

1. Double-click **tMysqlInput** to open its **Basic settings** view.



2. Select the **Use an existing connection** check box to use the database connection that you have defined on the **tMysqlConnection** component.
3. To define the schema, select **Repository** and then click the [...] button to choose the database table whose schema is used. In this example, the target table holding the aggregated data is selected.
4. In the **Table Name** field, type in the name of the table you want to query. In this example, the table is the one holding the aggregated data.
5. In the **Query** area, enter the query statement to select the columns to be displayed.
6. Save your Job and press **F6** to execute it.

The source table is deleted.

```
Starting job ELTYudong at 02:43 24/05/2010.
[statistics] connecting to socket on port 3918
[statistics] connected
-----+
| tLogRow_1 |
|=+---+---=|
| country | total |
|=+---+---=|
| Canada | 2030 |
| China | 2012 |
| France | 2009 |
| Japan | 1925 |
| USA | 2024 |
-----+
[statistics] disconnected
Job ELTYudong ended at 02:43 24/05/2010. [exit code=0]
```

A two-column table *citizencount* is created in the database. It groups citizens according to their nationalities and gives their total count in each country.





## Appendix A. GUI

This appendix describes the Graphical User Interface (GUI) of *Talend Studio*.

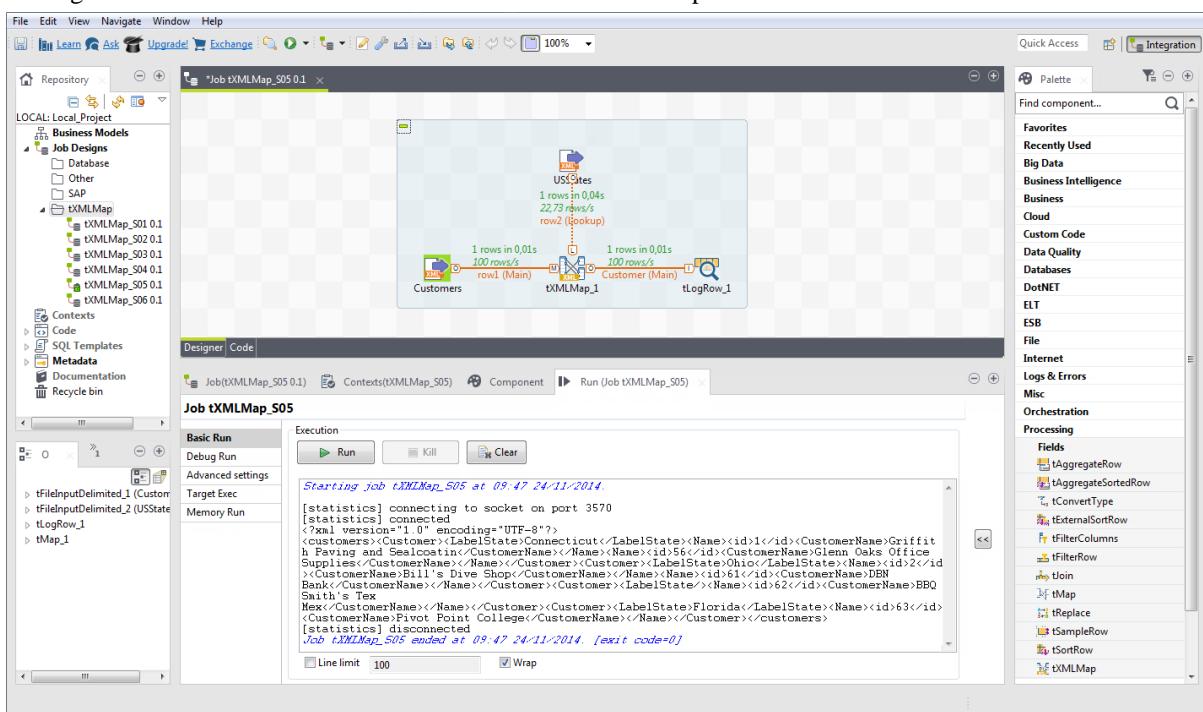
## A.1. Main window

*Talend Studio* main window is the interface from which you manage all types of data integration processes.

The *Talend Studio* multi-panel window is divided into:

- menu bar,
- toolbar,
- **Repository** tree view,
- design workspace,
- Palette,
- various configuration views in a tab system, for any of the elements in the data integration Job designed in the workspace,
- **Outline view and Code Viewer**.

The figure below illustrates *Talend Studio* main window and its panels and views.



The various panels and their respective features are detailed hereafter.



All the panels, tabs, and views described in this documentation are specific to *Talend Studio*. Some views listed in the [Show View] dialog box are Eclipse specific and are not subjects of this documentation. For information on such views, check Eclipse online documentation at <http://www.eclipse.org/documentation/>.

## A.2. Menu bar and Toolbar

At the top of the *Talend Studio* main window, various menus and a quick access toolbar gather **Talend** commonly features along with some Eclipse functions.

## A.2.1. Menu bar of *Talend Studio*

*Talend Studio's* menus include:

- some standard functions, such as **Save**, **Print**, **Exit**, which are to be used at the application level.
- some Eclipse native features to be used mainly at the design workspace level as well as specific *Talend Studio* functions.

The table below describes menus and menu items available to you on the menu bar of *Talend Studio*.



The menus on the menu bar differ slightly according to what you are working with: a Business Model or a Job.

Menu	Menu item	Description
<b>File</b>	<b>Close</b>	Closes the current open view on the Studio design workspace.
	<b>Close All</b>	Closes all open views on the Studio design workspace.
	<b>Save</b>	Saves any changes done in the current open view.
	<b>Save as</b>	Saves any changes done without changing the current open view.
	<b>Save All</b>	Saves any changes done in all open views.
	<b>Print</b>	Unavailable option.
	<b>Switch Project or Workspace</b>	Closes the current session and launches another one to enable you to open a different project in the Studio or connects to a different workspace. For more information see the Getting Started Guide.
	<b>Edit project properties</b>	Opens a dialog box where you can customize the settings of the current project. For more information, see <i>Customizing project settings</i> .
	<b>Import</b>	Opens a wizard that helps you to import different types of resources (files, items, preferences, XML catalogs, etc.) from different sources.
	<b>Export</b>	Opens a wizard that helps you to export different types of resources (files, items, preferences, breakpoints, XML catalogs, etc.) to different destinations.
	<b>Exit</b>	Closes the Studio main window.
<b>Edit</b>	<b>Undo</b>	Undoes the last action done in the Studio design workspace.
	<b>Redo</b>	Redoes the last action done in the Studio design workspace.
	<b>Cut</b>	Cuts selected object in the Studio design workspace.
	<b>Copy</b>	Copies the selected object in the Studio design workspace.
	<b>Paste</b>	Pastes the previously copied object in the Studio design workspace.
	<b>Delete</b>	Deletes the selected object in the Studio design workspace.
	<b>Select All</b>	Selects all components present in the Studio design workspace.
<b>View</b>	<b>Zoom In</b>	Obtains a larger image of the open Job.
	<b>Zoom Out</b>	Obtains a smaller image of the open Job.
	<b>Grid</b>	Displays grid in the design workspace. All items in the open Job are snapped to it.
	<b>Snap to Geometry</b>	Enables the Snap to Geometry feature.
<b>Window</b>	<b>Perspective</b>	Opens different perspectives corresponding to the different items in the list.
	<b>Show View...</b>	Opens the [Show View] dialog box which enables you to display different views on the Studio.
	<b>Maximize Active View or Editor...</b>	Maximizes the current perspective.
	<b>Preferences</b>	Opens the [Preferences] dialog box which enables you to set your preferences. For more information about preferences, see <i>Setting Talend Studio preferences</i> .
<b>Help</b>	<b>Welcome</b>	Opens a welcoming page which has links to <i>Talend Studio</i> documentation and <b>Talend</b> practical sites.

Menu	Menu item	Description
	<b>Help Contents</b>	Opens the Eclipse help system documentation.
	<b>Install Additional Packages...</b>	Opens the [Additional Talend packages] dialog box where you can select external modules required for certain components of your <i>Talend Studio</i> to work.
	<b>About Talend Studio</b>	Displays: <ul style="list-style-type: none"> <li>• the software version you are using,</li> <li>• detailed information on your software configuration that may be useful if there is a problem,</li> <li>• detailed information about plug-in(s),</li> <li>• detailed information about <i>Talend Studio</i> features.</li> </ul>
	<b>Support Logs</b>	Opens a wizard that helps you to export all logs generated in the Studio and system configuration information to an archived file.
	<b>Studio Quick Tour</b>	Opens a step-by-step presentation that introduces the Repository, the Design Workspace, the Configuration Tabs, and the Palette of <i>Talend Studio</i> .

## A.2.2. Toolbar of *Talend Studio*

The toolbar contains icons that provide you with quick access to the commonly used operations you can perform from *Talend Studio* main window.



The icons on the toolbar differ slightly according to what you are working with: a Business Model or a Job.

The table below describes the toolbar icons and their functions.

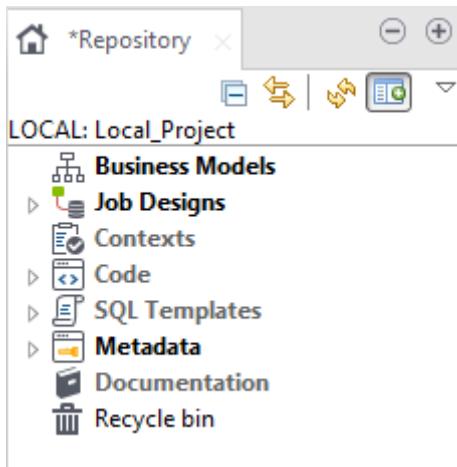
Name	Icon	Description
<b>Save</b>		Saves current job design.
<b>Save as</b>		Saves as another new Job.
<b>Export items</b>		Exports repository items to an archive file, for deploying outside <i>Talend Studio</i> . Instead if you intend to import the exported element into a newer version of <i>Talend Studio</i> or of another workstation, make sure the source files are included in the archive.
<b>Import items</b>		Imports repository items from an archive file into your current <i>Talend Studio</i> . For more information regarding the import/export items feature, see <a href="#">How to import items</a> .
<b>Find a specific job</b>		Displays the relevant dialog box that enables you to open any Job listed in the <b>Repository</b> tree view.
<b>Run job</b>		Executes the Job currently shown on the design space. For more information about job execution, see <a href="#">Handling Job execution</a> .
<b>Create</b>		Launches the relevant creation wizard. Through this menu, you can create any repository item including Business models, Job Designs, contexts, routines and metadata entries.
<b>Project settings</b>		Launches the [Project Settings] dialog box. From this dialog box, you can add a description to the current Project and customize the <b>Palette</b> display. For more information, see <a href="#">Customizing project settings</a> .
<b>Detect and update all jobs</b>		Searches for all updates available for your Jobs.
<b>Export Talend projects</b>		Launches the [Export Talend projects] wizard. For more information about project export, see <a href="#">How to export a project</a> .

## A.3. Repository tree view

The **Repository** tree view gathers all the technical items that can be used either to describe business models or to design Jobs. It gives access to any item including **Business Models**, **JobDesigns**, as well as reusable routines or documentation.

The **Repository** centralizes and stores all necessary elements for any Job design and business modeling contained in a project.

The figure below illustrates the elements stored in the **Repository**.



The **Refresh** button allows you to update the tree view with the last changes made.



The **Activate filter** button allows you to open the filter settings view so as to configure the display of the **Repository** view.

The **Repository** tree view stores all your data (Business, Jobs) and metadata (Routines, DB/File connections, any meaningful Documentation and so on).

The table below describes the nodes in the **Repository** tree view.

Node	Description
<b>Business Models</b>	Under the <b>Business Models</b> node, are grouped all business models of the project. Double-click the name of the model to open it on the design workspace. For more information, see <a href="#">Designing a Business Model</a> .
<b>Job Designs</b>	The <b>Job Designs</b> node shows the tree view of the designed Jobs for the current project. Double-click the name of a Job to open it on the design workspace. For more information, see <a href="#">Designing a Job</a> .
<b>Contexts</b>	The <b>Contexts</b> node groups files holding the contextual variables that you want to reuse in various Jobs, such as filepaths or DB connection details. For more information, see <a href="#">Using contexts and variables</a> .
<b>Code</b>	The <b>Code</b> node is a library that groups the routines available for this project and other pieces of code that could be reused in the project. Click the relevant tree entry to expand the appropriate code piece. For more information, see <a href="#">Managing routines</a> .
<b>SQL Templates</b>	The <b>SQL Templates</b> node groups all system SQL templates and gives the possibility to create user-defined SQL templates. For more information, see <a href="#">Using SQL templates</a> .
<b>Metadata</b>	The <b>Metadata</b> node bundles files holding redundant information you want to reuse in various Jobs, such as schemas and property data. For more information, see <a href="#">Managing Metadata</a> .
<b>Documentation</b>	The <b>Documentation</b> node gathers all types of documents, of any format. This could be, for example, specification documents or a description of technical format of a file. Double-click to open the document in the relevant application. For more information, see <a href="#">How to generate HTML documentation</a> .
<b>Recycle bin</b>	The <b>Recycle bin</b> groups all elements deleted from any node in the <b>Repository</b> tree view.

Node	Description
	The deleted elements are still present on your file system, in the recycle bin, until you right-click the recycle bin icon and select <b>Empty Recycle bin</b> .
	Expand the recycle bin to view any elements held within. You can action an element directly from the recycle bin, restore it or delete it forever by clicking right and selecting the desired action from the list.

## A.4. Design workspace

In the *Talend Studio's* design workspace, both Business Models and Job Designs can be laid out.

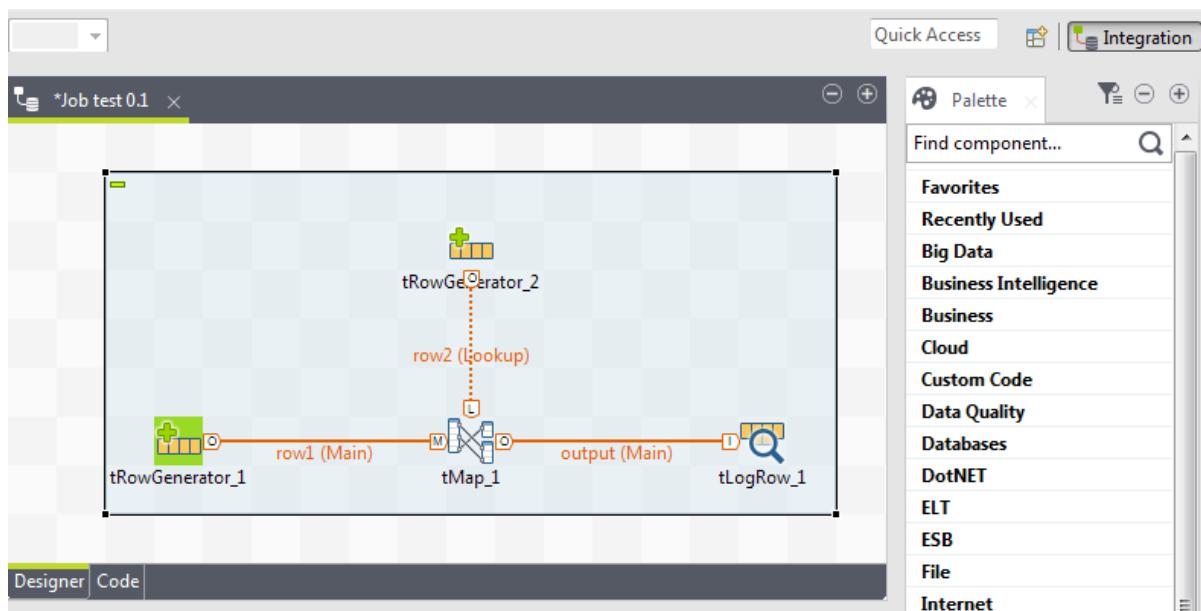
For more information, see [Opening or creating a Business Model](#) and [Creating a Job](#).

For both Business Models and Job Designs: active designs display in a easily accessible tab system above this workspace.

For Job Designs only. Under this workspace, you can access several other tabs:

- the **Designer** tab. It opens by default when creating a Job. It displays the Job in a graphical mode.
- the **Code** tab. It enables you to visualize the code and highlights the possible language errors.

Warnings are indicated in yellow whereas errors are indicated in red.



A **Palette** is docked at the top of the design workspace to help you draw the model corresponding to your workflow needs.

## A.5. Palette

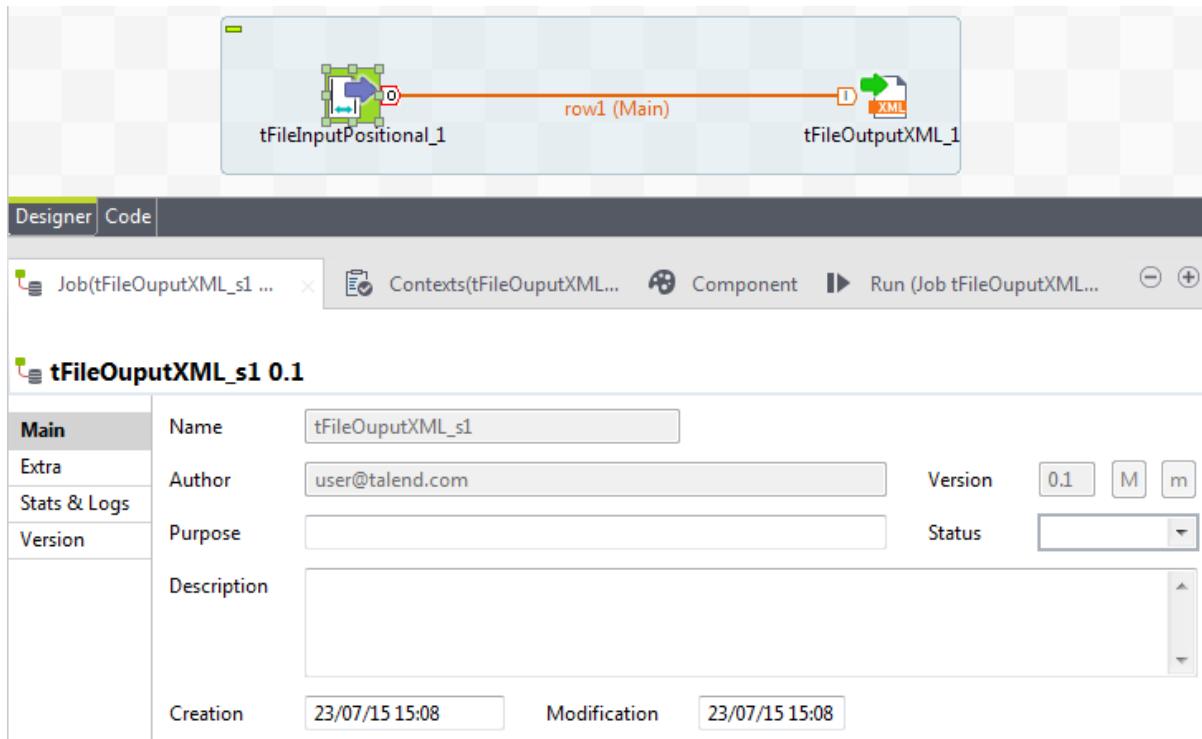
From the **Palette**, depending on whether you are designing a Job or modeling a Business Model, you can drop technical components or shapes, branches and notes to the design workspace for Job design or business modeling.

Related topics:

- [Designing a Business Model.](#)
- [Designing a Job.](#)
- [How to change the Palette layout and settings.](#)

## A.6. Configuration tabs

The configuration tabs are located in the lower half of the design workspace. Each tab opens a view that displays the properties of the selected element in the design workspace. These properties can be edited to change or set the parameters related to a particular component or to the Job as a whole.



The **Component**, **Run Jobs**, **Problems** and **Error Log** views gather all information relative to the graphical elements selected in the design workspace or the actual execution of the open Job.

The **Modules** and **Scheduler** tabs are located in the same tab system as the **Component**, **Logs** and **Run Job** tabs. Both views are independent from the active or inactive Jobs open on the design workspace.



You can show more tabs in this tab system and directly open the corresponding view if you select **Window > Show view** and then, in the open dialog box, expand any node and select the element you want to display.

The sections below describe the view of each of the configuration tabs.

View	Description
<b>Component</b>	This view details the parameters specific to each component of the <b>Palette</b> . To create a Job that will function, you are required to fill out the necessary fields of this <b>Component</b> view for each component forming your Job.  For more information about the <b>Component</b> view, see <a href="#">How to define component properties</a> .
<b>Run Job</b>	This view obviously shows the current job execution. It becomes a log console at the end of an execution.  For details about job execution, see <a href="#">Handling Job execution</a> .
<b>Error Log</b>	This view is mainly used for Job execution errors. It shows the history of warnings or errors occurring during job executions.  The log tab has also an informative function for a Java component operating progress, for example.

View	Description
	<b>Error Log</b> tab is hidden by default. As for any other view, go to <b>Window &gt; Show views</b> , then expand <b>General</b> node and select <b>Error Log</b> to display it on the tab system.
<b>Modules</b>	This view shows if a module is necessary and required for the use of a referenced component. Checking the <b>Modules</b> view helps to verify what modules you have or should have to run smoothly your Jobs.  For more information, see the <i>Talend Installation and Upgrade Guide</i> .
<b>Job view</b>	The <b>Job</b> view displays various information related to the open Job on the design workspace. This view has the following tabs:
	<p><b>Main tab</b></p> <p>This tab displays basic information about the Job opened on the design workspace, for example its name, author, version number, etc. The information is read-only. To edit it you have to close your Job, right-click its label on the <b>Repository</b> tree view and click <b>Edit properties</b> on the drop-down list.</p> <p><b>Extra tab</b></p> <p>This tab displays extra parameters including multi thread and implicit context loading features. For more information, see <a href="#">How to use the features in the Extra tab</a></p> <p><b>Stats/Log tab</b></p> <p>This tab allows you to enable/disable the statistics and logs for the whole Job.</p> <p>You can already enable these features for every single component of your Job by simply using and setting the relevant components: <b>tFlowMeterCatcher</b>, <b>tStatCatcher</b>, <b>tLogCatcher</b>.</p> <p>In addition, you can now set these features for the whole active Job (for all components of your Job) in one go, without using the Catcher components mentioned above. This way, all components get tracked and logged in the File or Database table according to your setting.</p> <p>You can also save the current setting to Project Settings by clicking the  button.</p> <p>For more details about the Stats &amp; Logs automation, see <a href="#">How to automate the use of statistics &amp; logs</a>.</p> <p><b>Version tab</b></p> <p>This tab displays the different versions of the Job opened on the design workspace and their creation and modification dates.</p>
<b>Problems</b>	This view displays the messages linked to the icons docked at a components in case of problem, for example when part of its setting is missing. Three types of icons/messages exist: <b>Error</b> , <b>Warning</b> and <b>Infos</b> .  For more information, see <a href="#">Warnings and error icons on components</a> .
<b>Job Hierarchy</b>	This view displays a tree folder showing the child Job(s) of the parent Job selected. To show this view, right-click the parent Job in the <b>Repository</b> tree view and select <b>Open Job Hierarchy</b> on the drop-down list.  You can also show this view in the <b>Window &gt; Show view...</b> combination where you can select <b>Talend &gt; Job Hierarchy</b> .  You can see <b>Job Hierarchy</b> only if you create a parent Job and one or more child Job(s) via the <b>tRunJob</b> component.
<b>Properties</b>	When inserting a shape in the design workspace, the <b>Properties</b> view offers a range of formatting tools to help you customizing your business model and improve its readability.

## A.7. Outline and code summary panel

This panel is located below the **Repository** tree view. It displays detailed information about the open Job or Business Model in the design workspace.

The Information panel is composed of two tabs, **Outline** and **Code Viewer**, which provide information regarding the displayed diagram (either Job or Business Model) and also the generated code.

For more information, see [How to display the code or the outline of your Job](#).

## A.8. Shortcuts and aliases

Below is a table gathering all keyboard shortcuts currently in use:

Shortcut	Operation	Context
F2	Shows <b>Component</b> settings view.	Global application
F4	Shows <b>Run Job</b> view.	Global application
F6	Runs current Job or shows <b>Run Job</b> view if no Job is open.	Global application
Ctrl + F2	Shows <b>Module</b> view.	Global application
Ctrl + F3	Shows <b>Problems</b> view.	Global application
Ctrl + H	Shows the <b>Designer</b> view of the current Job.	Global application
Ctrl + G	Shows the <b>Code</b> view of the current Job.	Global application
Ctrl + R	Restores the initial <b>Repository</b> view.	From <b>Repository</b> view
Ctrl + Shift + F3	Synchronizes components javajet components.	Global application
Ctrl + Shift + J	Opens a Job.	Global application (In Windows)
F7	Switches to <b>Debug</b> mode.	From <b>Run Job</b> view
F5	Refreshes the <b>Repository</b> view.	From <b>Repository</b> view
F8	Kills current Job.	From <b>Run Job</b> view
F5	Refreshes <b>Modules</b> install status.	From <b>Modules</b> view
Ctrl+L	Execute SQL queries.	<b>Talend</b> commands (in Windows)
Ctrl+Space bar	Access global and user-defined variables. It can be error messages or line number for example, depending on the component selected.	From any component field in <b>Job</b> or <b>Component</b> views





## Appendix B. Customizing *Talend Studio* and setting Studio preferences

This chapter provides information on how to customize the *Talend Studio* and set Studio preferences so that your *Talend Studio* works the way you want.

In the following sections, you will find information on:

- [\*Customizing project settings\*](#)
- [\*Customizing the workspace\*](#)
- [\*Filtering entries listed in the Repository tree view\*](#)
- [\*Setting Talend Studio preferences\*](#)

## B.1. Customizing project settings

*Talend Studio* enables you to customize the information and settings of the project in progress, including the **Palette**, Job settings and Job version management, for example.

To customize project settings:

1. Click  on the Studio tool bar, or select **File > Edit Project Properties** from the menu bar.  
The **[Project Settings]** dialog box opens.
2. In the tree diagram to the left of the dialog box, select the setting you wish to customize and then customize it, using the options that appear to the right of the box.

From the dialog box you can also export or import the full assemblage of settings that define a particular project:

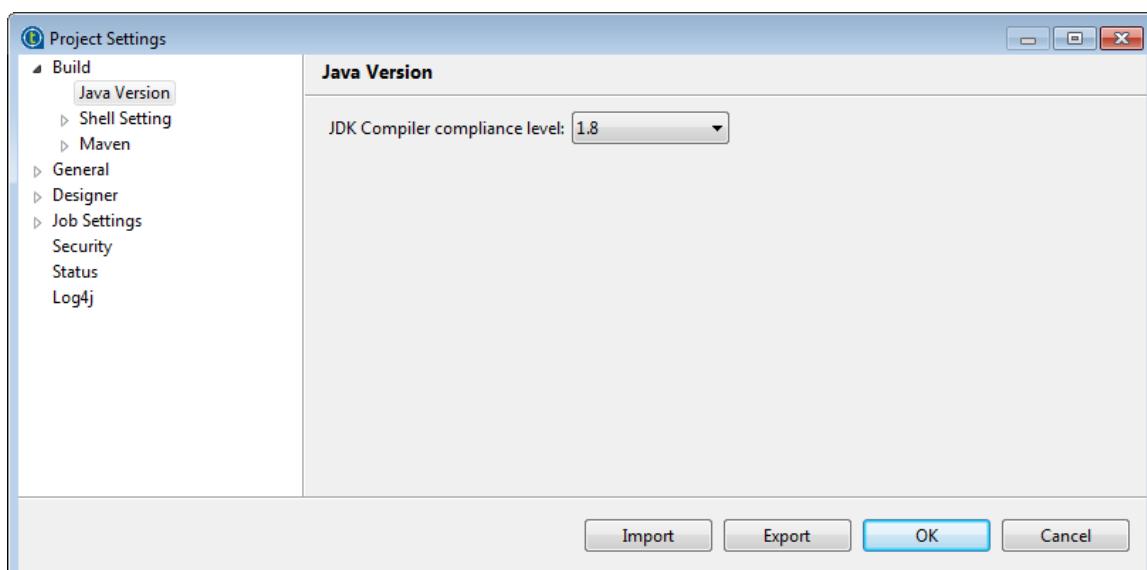
- To export the settings, click on the **Export** button. The export will generate an XML file containing all of your project settings.
- To import settings, click on the **Import** button and select the XML file containing the parameters of the project which you want to apply to the current project.

### B.1.1. Setting the compiler compliance level

The compiler compliance level corresponds to the Java version used for Job code generation.

For more information on the compiler compliance levels compatibility, see *Talend Installation and Upgrade Guide*.

1. Click  on the toolbar of the Studio main window, or click **File > Edit Project Properties** from the menu bar to open the **[Project Settings]** dialog box.
2. Expand the **Build** node and click **Java Version**.
3. From the **JDK Compiler compliance level** list, select the compiler compliance level you want to use, and then click **OK**.



## B.1.2. Customizing Maven build script templates

Your *Talend Studio* provides the following default templates for generating build scripts:

- Maven script templates for standalone Job export
- A Maven script template for OSGI bundle export of Jobs

Based on the default, global build templates, you can create folder-level build scripts. Build scripts generated based on these templates are executed when building Jobs.

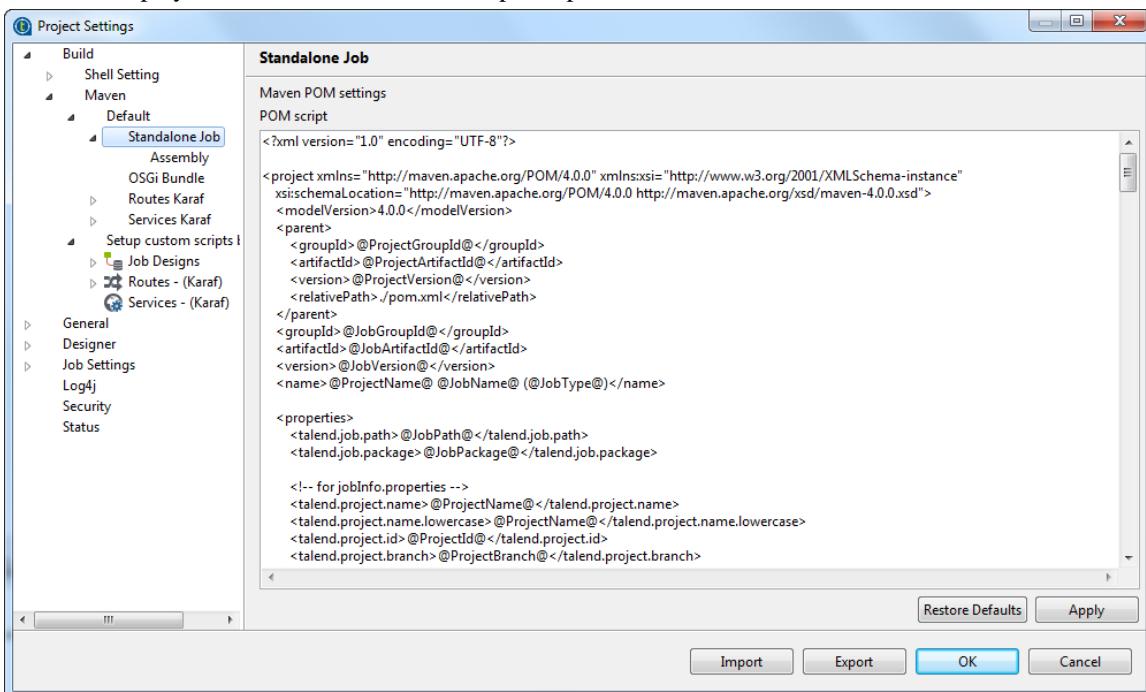
This section provides information on how to customize the build script templates. For information on how to build a Job, see [How to build Jobs](#).

### B.1.2.1. How to customize the global build script templates

In the **[Project Settings]** dialog box, you can find and customize the default, global build script templates under the **Build > Maven > Default** node. These script templates apply to all Jobs in the root folder and all sub-folders except those with their own build script templates set up.

The following example shows how to customize the global POM script template for standalone Jobs:

1. From the menu bar, click **File > Edit Project properties** to open the **[Project Settings]** dialog box.
2. Expand the **Build > Maven > Default** nodes, and then click the **Standalone Job** node to open the relevant view that displays the content of the POM script template.



Depending on the Studio product you are using, the project settings items in your Studio may differ from what is shown above.

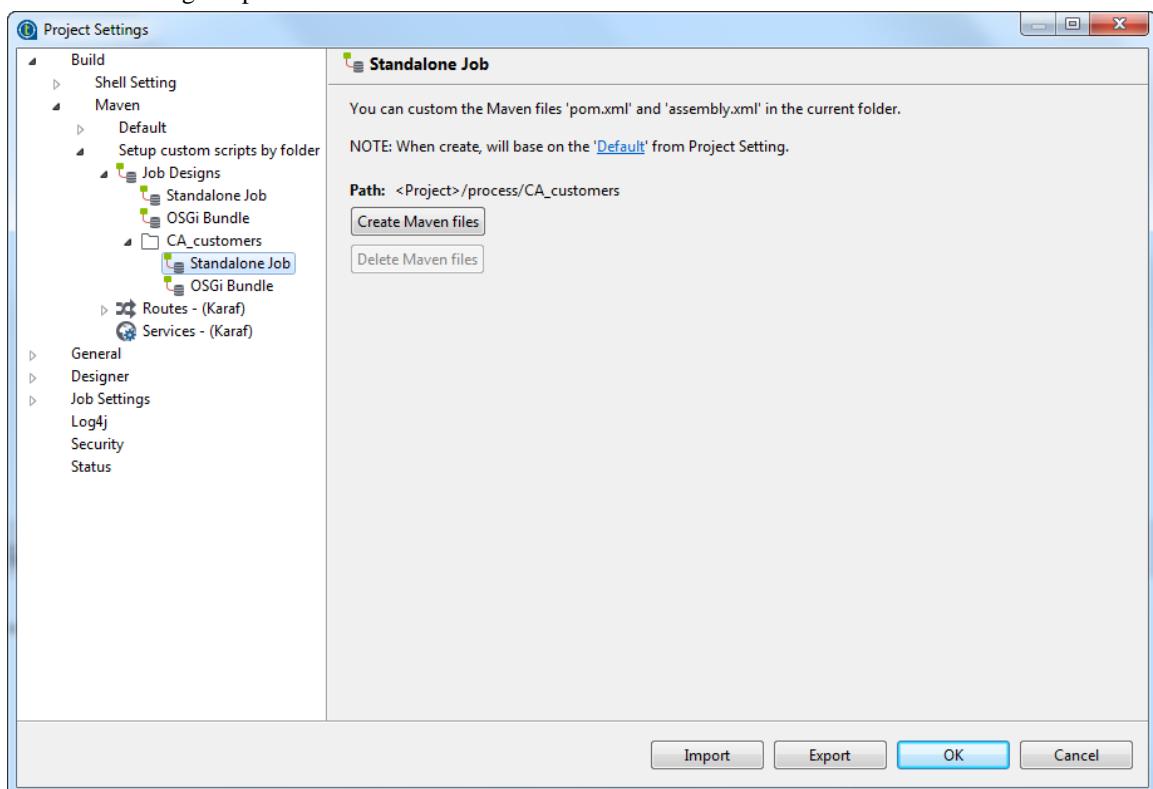
3. Modify the script code in the text panel and click **OK** to finish your customization.

## B.1.2.2. How to customize the folder-level build script templates

Based on the global build script templates, you can add and customize script templates for Jobs folder by folder under the **Build > Maven > Setup custom scripts by folder** node. The build script templates added for a folder apply to all Jobs in that folder and all its sub-folders except those with their own build script templates set up.

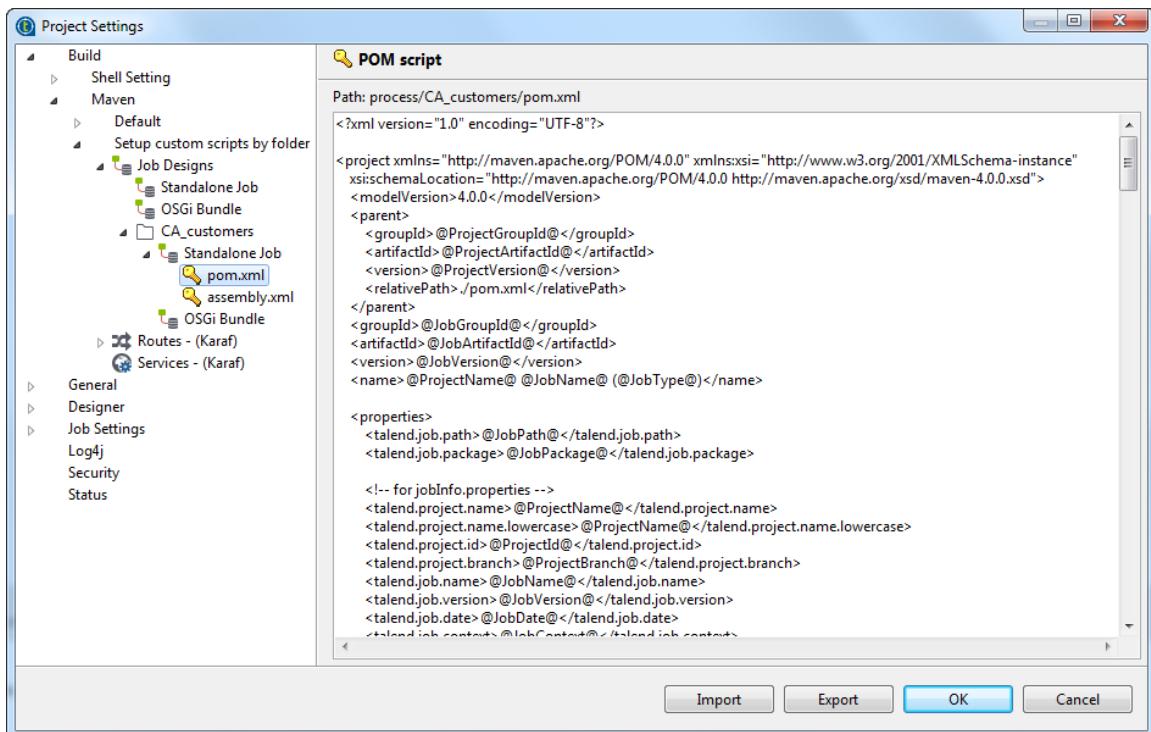
The following example shows how to add and customize the POM script template for building standalone Jobs from Jobs in the *CA\_customers* folder:

1. From the menu bar, click **File > Edit Project properties** to open the **[Project Settings]** dialog box.
2. Expand the **Build > Maven > Setup custom scripts by folder > Job Designs > CA\_customers** nodes, and then click the **Standalone Job** node to open the relevant view, from which you can add script templates or delete all existing templates.



Depending on the Studio product you are using, the project settings items in your Studio may differ from what is shown above.

3. Click the **Create Maven files** button to create script templates based on the global templates for standalone Jobs.
4. Select the script template you want to customize, *pom.xml* in this example, to display the script code in the code view. Modify the script code in the text panel and click **OK** to finish your customization.



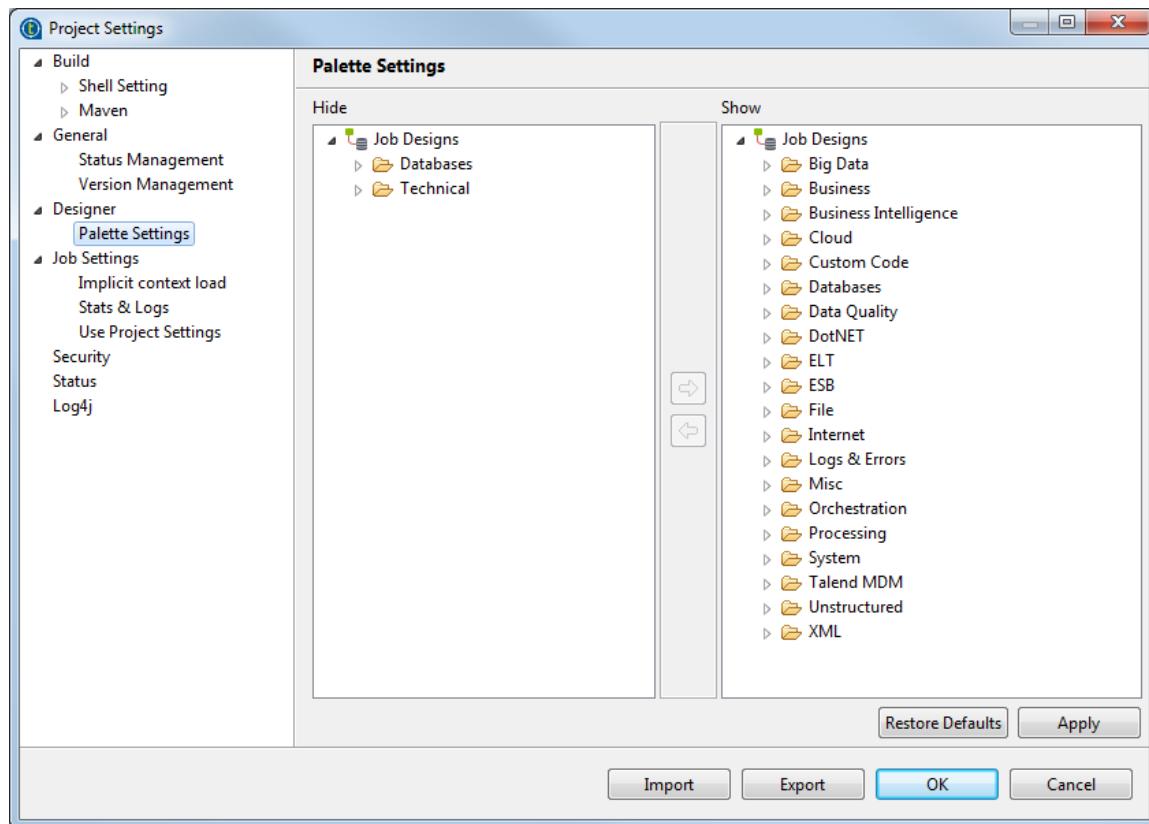
Once the build script templates are created for a folder, you can also go to the directory where the XML files are stored, `<studio_installation_directory>\workspace\<project_name>\process\CA_customers` in this example, and directly modify the XML file of the template you want to customize. Your changes will affect all Jobs in the folder and in all sub-folders except those with their own script set up.

### B.1.3. Palette Settings

You can customize the settings of the **Palette** display so that only the components used in the project are loaded. This will allow you to launch the Studio more quickly.

To customize the **Palette** display settings:

1. On the toolbar of the Studio's main window, click or click **File > Edit Project Properties** on the menu bar to open the **[Project Settings]** dialog box.



In the **General** view of the **[Project Settings]** dialog box, you can add a project description, if you did not do so when creating the project.

2. In the tree view of the **[Project Settings]** dialog box, expand **Designer** and select **Palette Settings**. The settings of the current **Palette** are displayed in the panel to the right of the dialog box.
3. Select one or several components, or even set(s) of components you want to remove from the current project's **Palette**.
4. Use the left arrow button to move the selection onto the panel on the left. This will remove the selected components from the **Palette**.
5. To re-display hidden components, select them in the panel on the left and use the right arrow button to restore them to the **Palette**.
6. Click **Apply** to validate your changes and **OK** to close the dialog box.



To get back to the **Palette** default settings, click **Restore Defaults**.

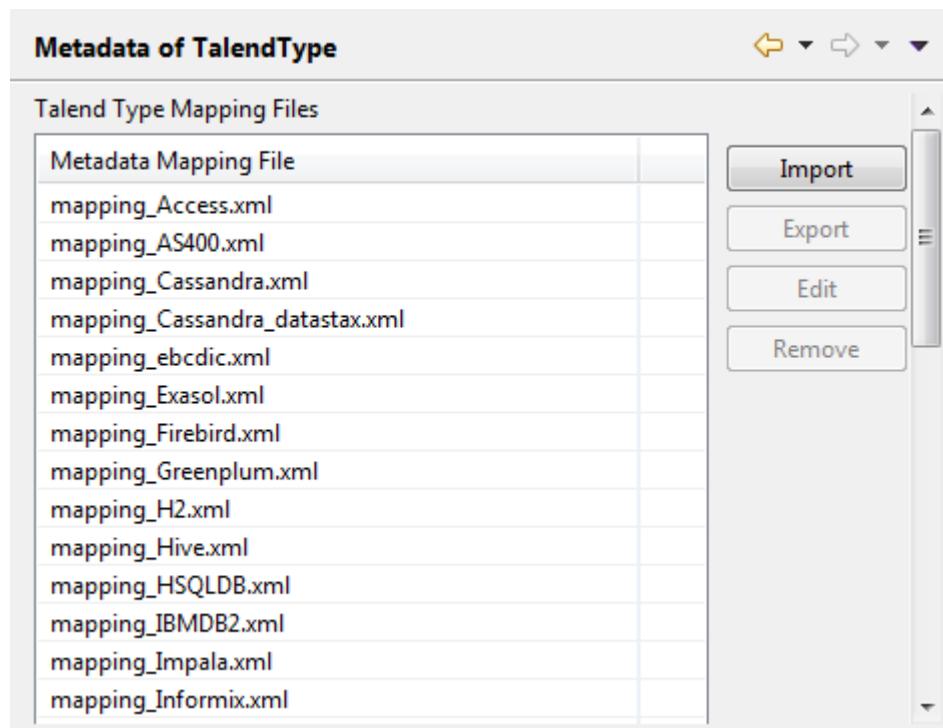
For more information on the **Palette**, see [How to change the Palette layout and settings](#).

## B.1.4. Type mapping

You can set the parameters for type conversion in *Talend Studio*, from Java towards databases and vice versa.

1. On the toolbar of the Studio main window, click or click **File > Edit Project Properties** from the menu bar to open the **[Project Settings]** dialog box.

2. In the tree view of the dialog box, expand **General** and select **Metadata of Talend Type** to open the relevant view.



The **Metadata Mapping File** area lists the XML files that hold the conversion parameters for each database type used in *Talend Studio*.

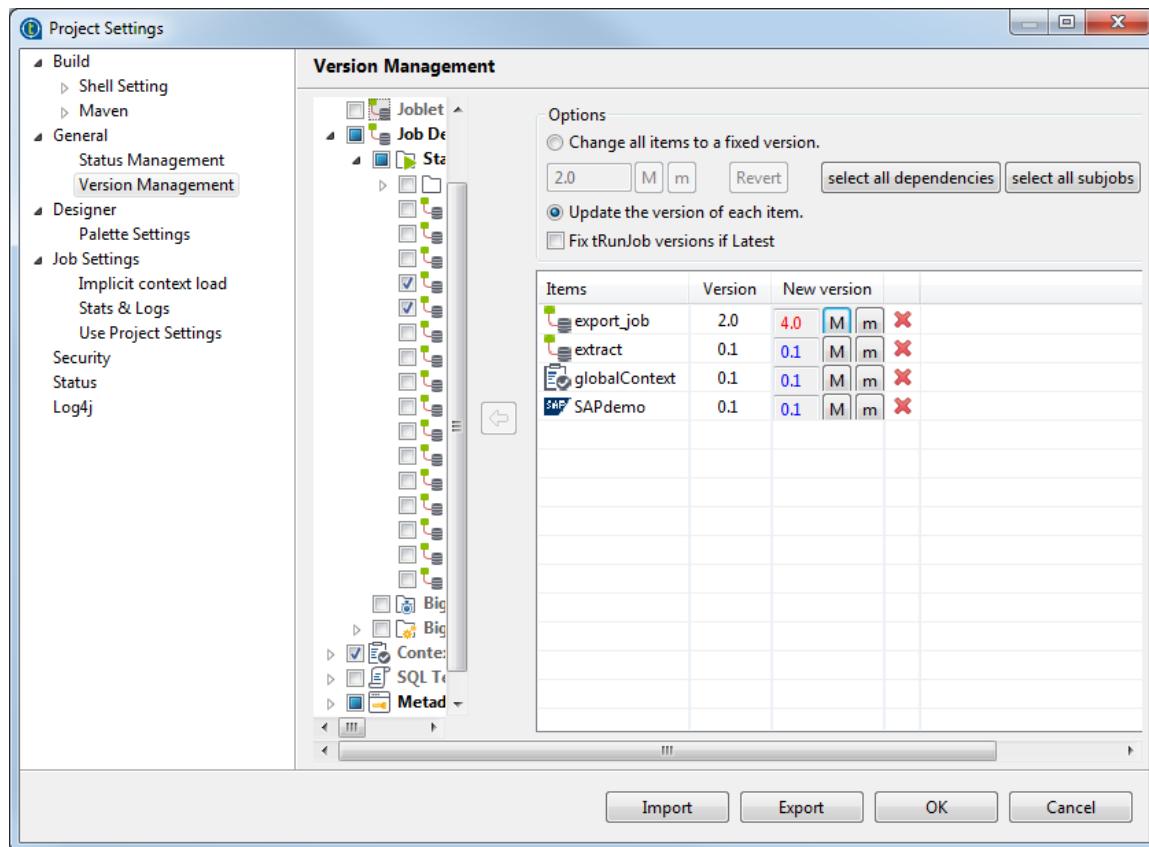
- You can import, export, or delete any of the conversion files by clicking **Import**, **Export** or **Remove** respectively.
- You can modify any of the conversion files according to your needs by clicking the **Edit** button to open the **[Edit mapping file]** dialog box and then modify the XML code directly in the open dialog box.

## B.1.5. Version management

You can also manage the version of each item in the **Repository** tree view through **General > Version Management** of the **[Project Settings]** dialog box.

To do so:

1. On the toolbar of the Studio main window, click or click **File > Edit Project Properties** from the menu bar to open the **[Project Settings]** dialog box.
2. In the tree view of the dialog box, expand **General** and select **Version Management** to open the corresponding view.



- In the **Repository** tree view, expand the node holding the items you want to manage their versions and then select the check boxes of these items.

The selected items display in the **Items** list to the right along with their current version in the **Version** column and the new version set in the **New Version** column.

- Make changes as required:
  - In the **Options** area, select the **Change all items to a fixed version** option to change the version of the selected items to the same fixed version.
  - Click **Revert** if you want to undo the changes.
  - Click **Select all dependencies** if you want to update all of the items dependent on the selected items at the same time.
  - Click **Select all subjobs** if you want to update all of the subjobs dependent on the selected items at the same time.
  - To increment each version of the items, select the **Update the version of each item** option and change them manually.
  - Select the **Fix tRunjob versions if Latest** check box if you want the father job of current version to keep using the child Job(s) of latest version as current version in the tRunjob to be versioned, regardless of how their versions will update. For example, a **tRunJob** will update from the current version *1.0* to *1.1* at both father and child levels. Once this check box is selected, the father Job *1.0* will continue to use the child Job *1.0* rather than the latest one as usual, say, version *1.1* when the update is done.



*To use this check box, the father Job must be using child Job(s) of the latest version as current version in the tRunjob to be versioned, by having selected the Latest option from the drop-down version list in the Component view of the child Job(s). For more information on tRunJob, see tRunJob at <https://help.talend.com>.*

- Click **OK** to apply your changes and close the dialog box.



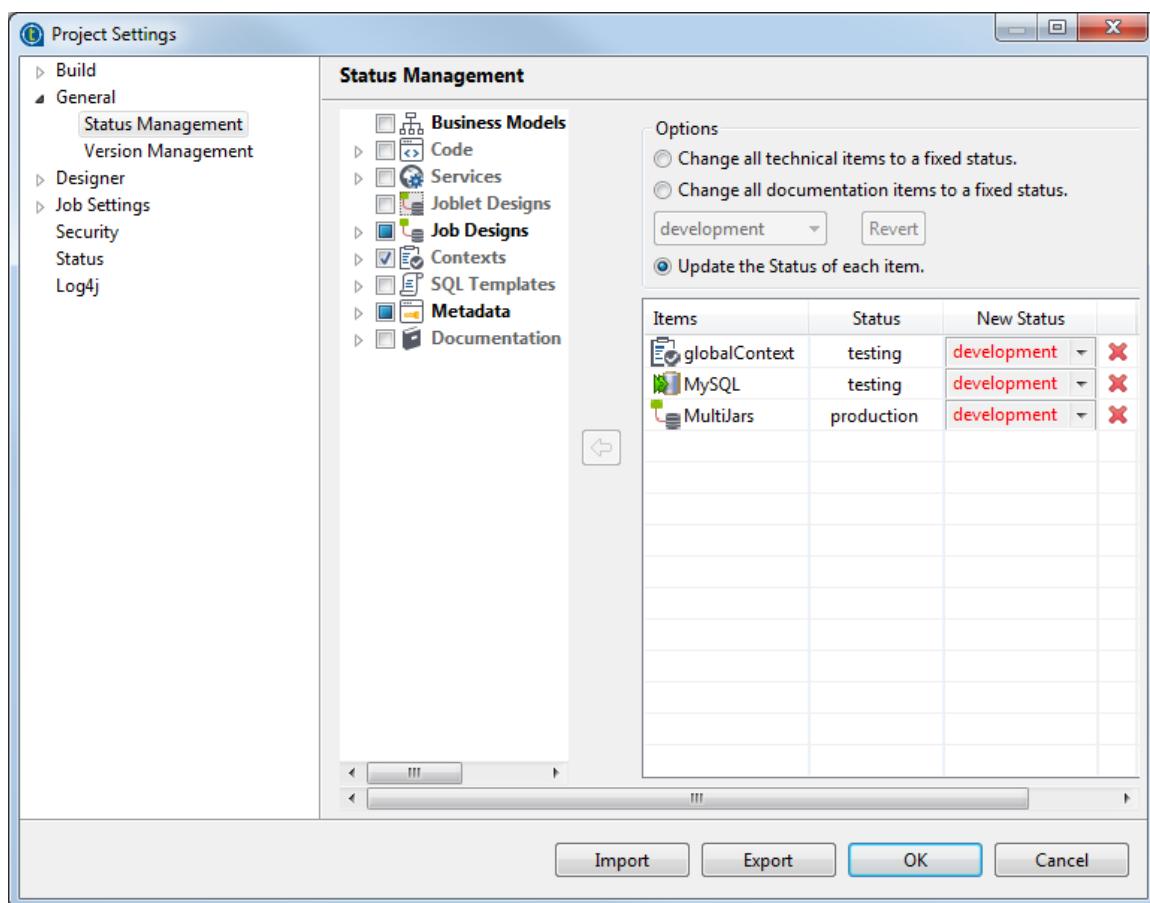
For more information on version management, see [Managing Job versions](#).

## B.1.6. Status management

You can also manage the status of each item in the **Repository** tree view through **General > Status Management** of the **[Project Settings]** dialog box.

To do so:

1. On the toolbar of the Studio main window, click or click **File > Edit Project Properties** from the menu bar to open the **[Project Settings]** dialog box.
2. In the tree view of the dialog box, expand **General** and select **Status Management** to open the corresponding view.



3. In the **Repository** tree view, expand the node holding the items you want to manage their status and then select the check boxes of these items.

The selected items display in the **Items** list to the right along with their current status in the **Status** column and the new status set in the **New Status** column.

4. In the **Options** area, select the **Change all technical items to a fixed status** check box to change the status of the selected items to the same fixed status.
5. Click **Revert** if you want to undo the changes.
6. To increment each status of the items, select the **Update the version of each item** check box and change them manually.

- Click **Apply** to apply your changes and then **OK** to close the dialog box.



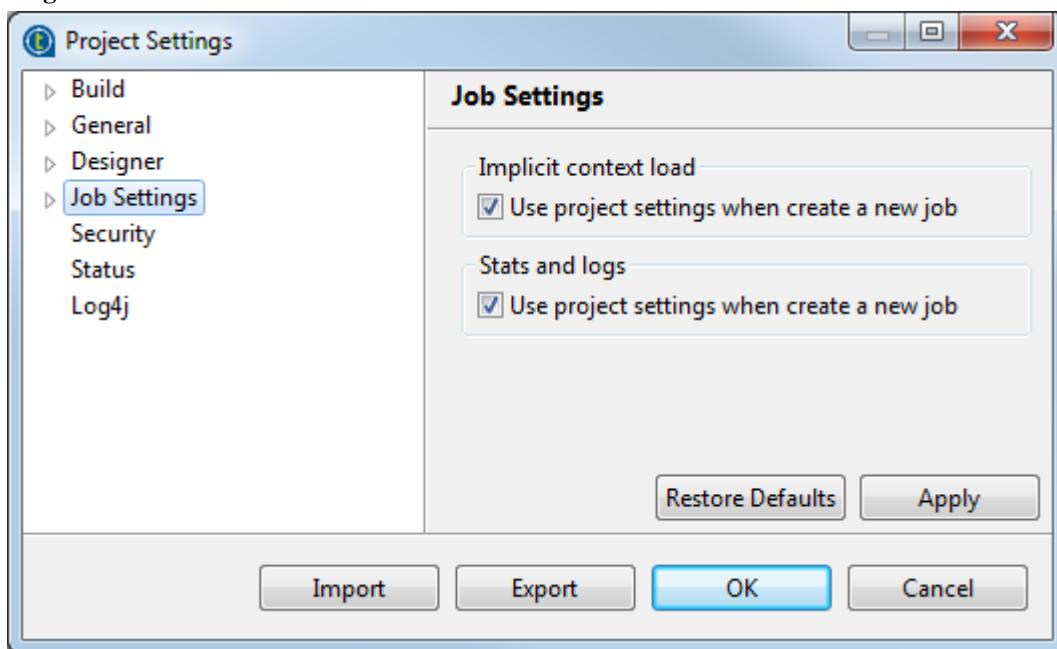
For further information about Job status, see [Status settings](#).

## B.1.7. Job Settings

You can automatically use **Implicit Context Load** and **Stats and Logs** settings you defined in the **[Project Settings]** dialog box of the actual project when you create a new Job.

To do so:

- On the toolbar of the Studio main window, click or click **File > Edit Project Properties** from the menu bar to open the **[Project Settings]** dialog box.
- In the tree view of the dialog box, click the **Job Settings** node to open the corresponding view.
- Select the **Use project settings when create a new job** check boxes of the **Implicit Context Load** and **Stats and Logs** areas.



- Click **Apply** to validate your changes and then **OK** to close the dialog box.

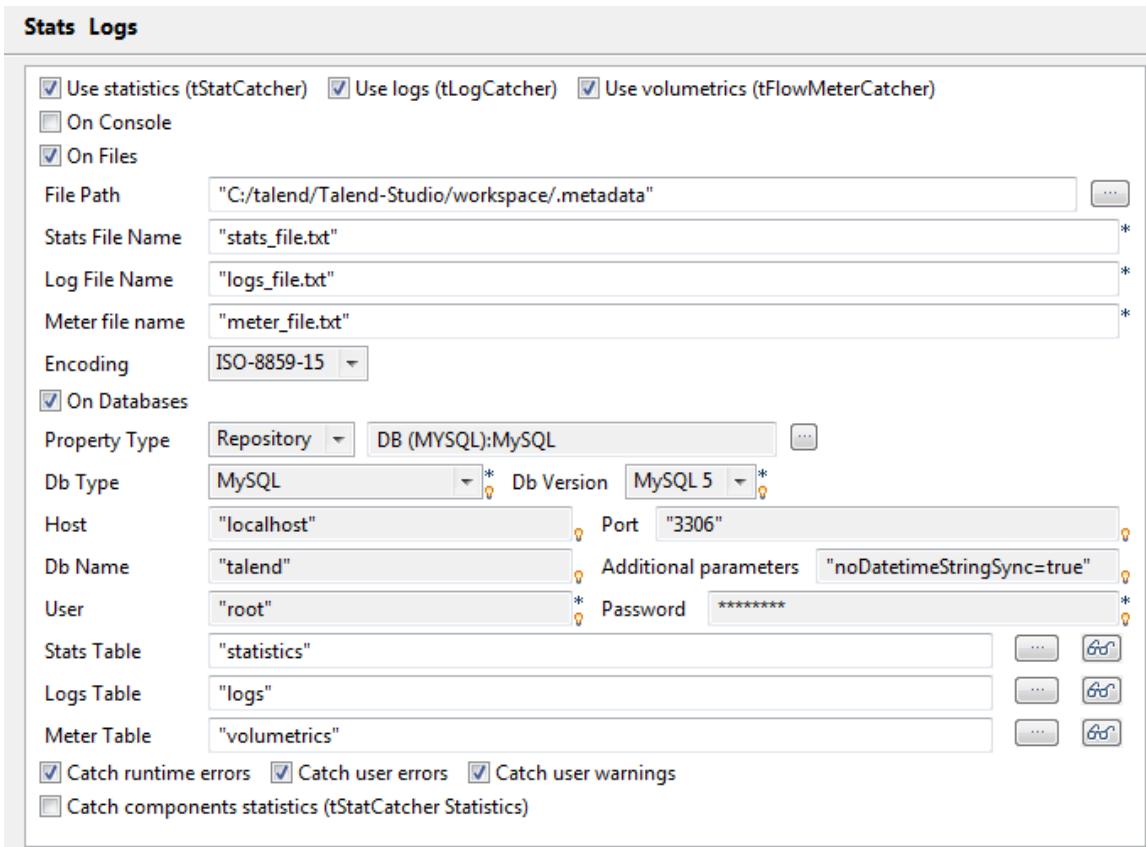
## B.1.8. Stats & Logs

When you execute a Job, you can monitor the execution through the **tStatCatcher Statistics** option or through using a log component. This will enable you to store the collected log data in .csv files or in a database.

You can then set up the path to the log file and/or database once for good in the **[Project Settings]** dialog box so that the log data get always stored in this location.

To do so:

- On the toolbar of the Studio main window, click  or click **File > Edit Project Properties** from the menu bar to open the **[Project Settings]** dialog box.
- In the tree view of the dialog box, expand the **Job Settings** node and then click **Stats & Logs** to display the corresponding view.



If you know that the preferences for Stats & Logs will not change depending upon the context of execution, then simply set permanent preferences. If you want to apply the Stats & Logs settings individually, then it is better to set these parameters directly onto the Stats & Logs view. For more information about this view, see [How to automate the use of statistics & logs](#).

- Select the **Use Statistics**, **Use Logs** and **Use Volumetrics** check boxes where relevant, to select the type of log information you want to set the path for.
- Select a format for the storage of the log data: select either the **On Files** or **On Database** check box. Or select the **On Console** check box to display the data in the console.

The relevant fields are enabled or disabled according to these settings. Fill out the **File Name** between quotes or the **DB name** where relevant according to the type of log information you selected.

You can now store the database connection information in the **Repository**. Set the **Property Type** to **Repository** and browse to retrieve the relevant connection metadata. The fields get automatically completed.



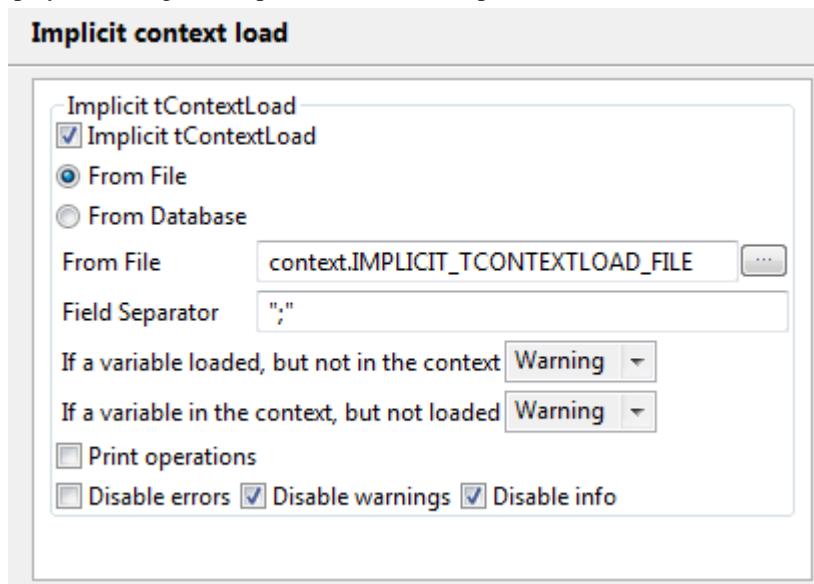
Alternatively, if you save your connection information in a Context, you can also access them through **Ctrl+Space**.

## B.1.9. Context settings

You can define default context parameters you want to use in your Jobs.

To do so:

1. On the toolbar of the Studio main window, click or click **File > Edit Project Properties** from the menu bar to open the **[Project Settings]** dialog box.
2. In the tree view of the dialog box, expand the **Job Settings** node and then select the **Implicit Context Load** check box to display the configuration parameters of the Implicit **tContextLoad** feature.



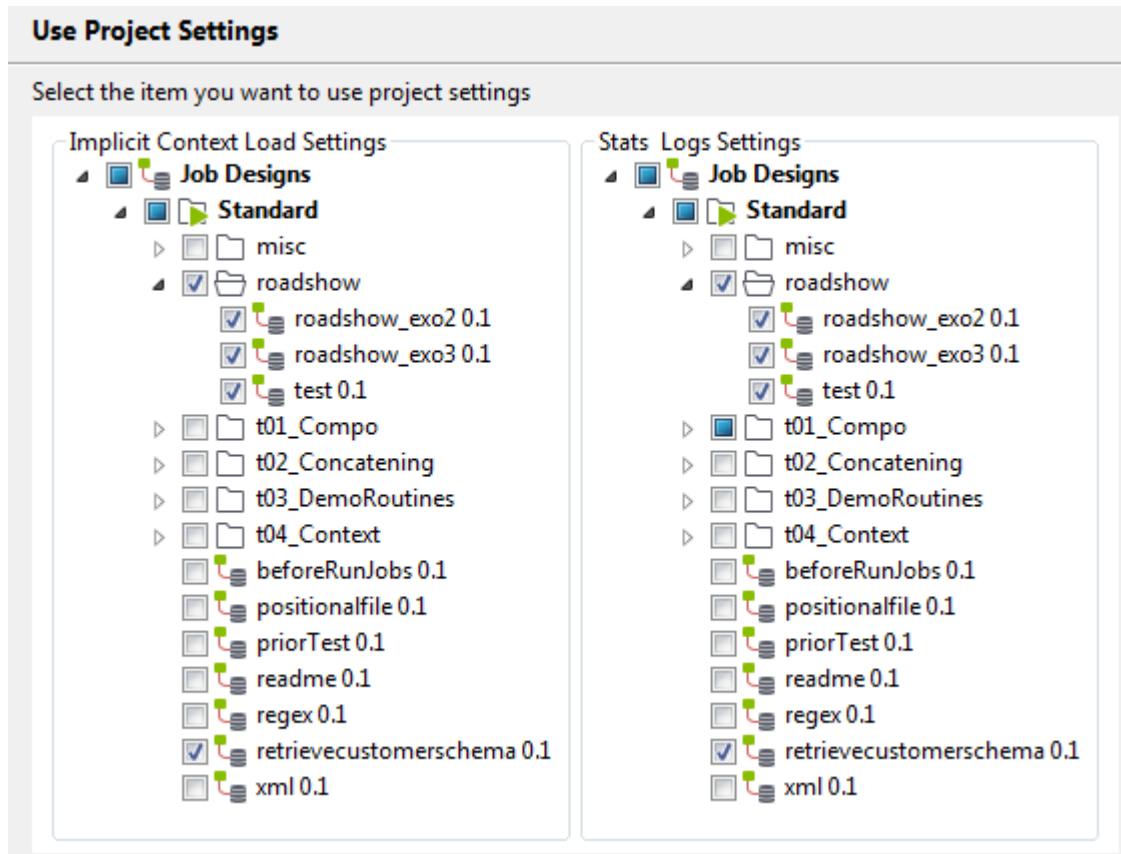
3. Select the **From File** or **From Database** check boxes according to the type of file you want to store your contexts in.
4. For files, fill in the file path in the **From File** field and the field separator in the **Field Separator** field.
5. For databases, select the **Built-in** or **Repository** mode in the **Property Type** list and fill in the next fields.
6. Fill in the **Table Name** and **Query Condition** fields.
7. Select the type of system message you want to have (warning, error, or info) in case a variable is loaded but is not in the context or vice versa.
8. Click **Apply** to validate your changes and then **OK** to close the dialog box.

## B.1.10. Applying Project Settings

From the **[Project Settings]** dialog box, you can choose to which Job in the **Repository** tree view you want to apply the **Implicit Context Load** and **Stats and Logs** settings.

To do so:

1. On the toolbar of the Studio main window, click or click **File > Edit Project Properties** from the menu bar to open the **[Project Settings]** dialog box.
2. In the tree view of the dialog box, expand the **Job Settings** node and then click **Use Project Settings** to display the use of **Implicit Context Load** and **Stats and Logs** option in the Jobs.



3. In the **Implicit Context Load Settings** area, select the check boxes corresponding to the Jobs in which you want to use the implicit context load option.
4. In the **Stats Logs Settings** area, select the check boxes corresponding to the Jobs in which you want to use the stats and logs option.
5. Click **Apply** to validate your changes and then **OK** to close the dialog box.

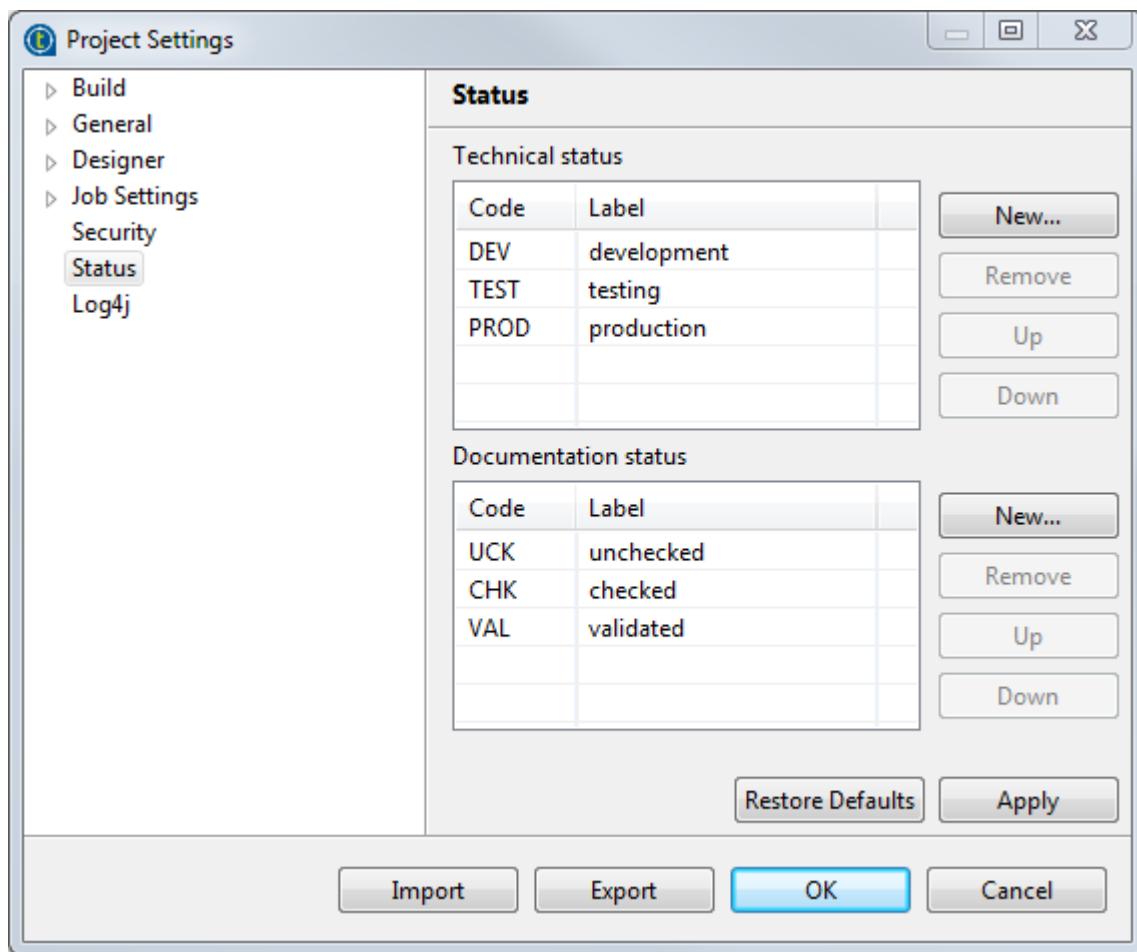
## B.1.11. Status settings

In the **[Project Settings]** dialog box, you can also define the Status.

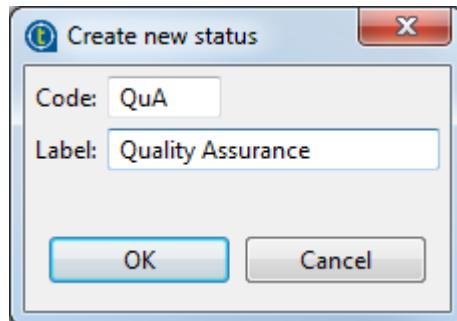
To do so:

1. On the toolbar of the Studio main window, click or click **File > Edit Project Properties** from the menu bar to open the **[Project Settings]** dialog box.
2. In the tree view of the dialog box, click the **Status** node to define the main properties of your **Repository** tree view elements.

The main properties of a repository item gathers information data such as **Name**, **Purpose**, **Description**, **Author**, **Version** and **Status** of the selected item. Most properties are free text fields, but the **Status** field is a drop-down list.



- Click the **New...** button to display a dialog box and populate the **Status** list with the most relevant values, according to your needs. Note that the **Code** cannot be more than 3-character long and the **Label** is required.



Talend makes a difference between two status types: **Technical status** and **Documentation status**.

The **Technical status** list displays classification codes for elements which are to be running on stations, such as Jobs, metadata or routines.

The **Documentation status** list helps classifying the elements of the repository which can be used to document processes(Business Models or documentation).

- Once you completed the status setting, click **OK** to save

The **Status** list will offer the status levels you defined here when defining the main properties of your Job designs and business models.

- In the **[Project Settings]** dialog box, click **Apply** to validate your changes and then **OK** to close the dialog box.

## B.1.12. Security settings

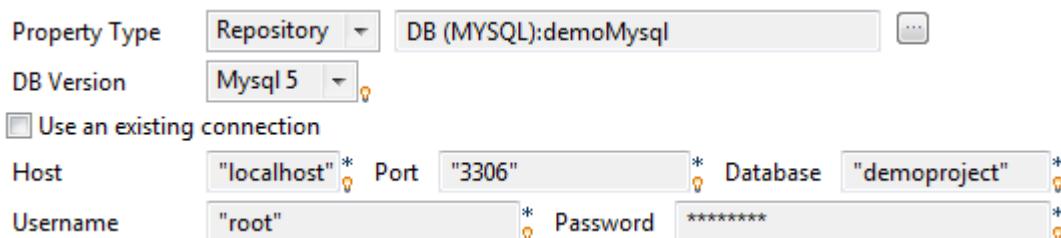
You can hide or show your passwords on your documentations, metadata, contexts, and so on when they are stored in the **Repository** tree view.

To hide your password:

1. On the toolbar of the Studio main window, click  or click **File > Edit Project Properties** from the menu bar to open the **[Project Settings]** dialog box.
2. In the tree view of the dialog box, click the **Security** node to open the corresponding view.
3. Select the **Hide passwords** check box to hide your password.



If you select the **Hide passwords** check box, your password will be hidden for all your documentations, contexts, and so on, as well as for your component properties when you select **Repository** in the **Property Type** field of the component **Basic settings** view, as in the screen capture below. However, if you select **Built-in**, the password will not be hidden.



4. In the **[Project Settings]** dialog box, click **Apply** to validate your changes and then **OK** to close the dialog box.

## B.2. Customizing the workspace

When using *Talend Studio* to design a data integration Job, you can customize the **Palette** layout and setting according to your needs. You can as well change the position of any of the panels that exist in the Studio to meet your requirements.



All the panels, tabs, and views described in this documentation are specific to *Talend Studio*. Some views listed in the **[Show View]** dialog box are Eclipse specific and are not subjects of this documentation. For information on such views, check Eclipse online documentation at <http://www.eclipse.org/documentation/>.

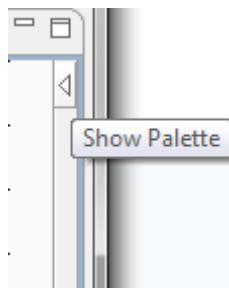
### B.2.1. How to change the Palette layout and settings

The **Palette** contains all basic technical components and shapes as well as branches for Job design and business modeling in the design workspace. These components and shapes as well as branches are grouped in families and sub-families.

*Talend Studio* enables you to change the layout and position of your **Palette** according to your requirements. the below sections explain all management options you can carry out on the **Palette**.

#### B.2.1.1. How to show, hide the Palette and change its position

By default, the **Palette** might be hidden on the right hand side of your design workspace.



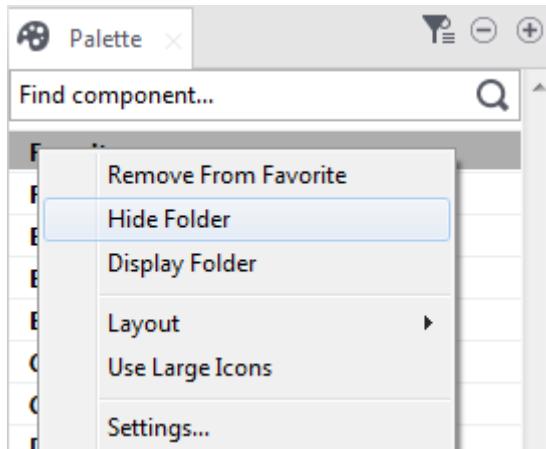
If you want the **Palette** to show permanently, click the left arrow, at the upper right corner of the design workspace, to make it visible at all times.

You can also move around the **Palette** outside the design workspace within the **Integration** perspective. To enable the standalone **Palette** view, click the **Window** menu > **Show View... > General > Palette**.

If you want to set the **Palette** apart in a panel, right-click the **Palette** head bar and select **Detached** from the contextual menu. The **Palette** opens in a separate view that you can move around wherever you like within the perspective.

### B.2.1.2. How to display/hide components families

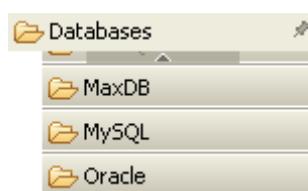
You can display/hide components families according to your needs in case of visibility problems, for example. To do so, right-click the **Palette** and select **Display folder** to display components families and **Hide folder** to display components without their families.



This display/hide option can be very useful when you are in the **Favorite** view of the **Palette**. In this view, you usually have a limited number of components that if you display without their families, you will have them in an alphabetical list and thus facilitate their usage. for more information about the **Palette** favorite, see [How to set the Palette favorite](#).

### B.2.1.3. How to maintain a component family open

If you often use one or many component families, you can add a pin on their names to stop them from collapsing when you select components from other families.



To add a pin, click the pin icon on the top right-hand corner of the family name.

### B.2.1.4. How to filter the Palette

You can select the components to be shown or hidden on your **Palette**. You can also add to the **Palette** the components that you developed yourself.

For more information about filtering the **Palette**, see [Palette Settings](#).

For more information about adding components to the **Palette**, either from **Talend Exchange** or from your own development, see [How to download/upload Talend Community components](#) and/or [How to define the user component folder \(Talend > Components\)](#).

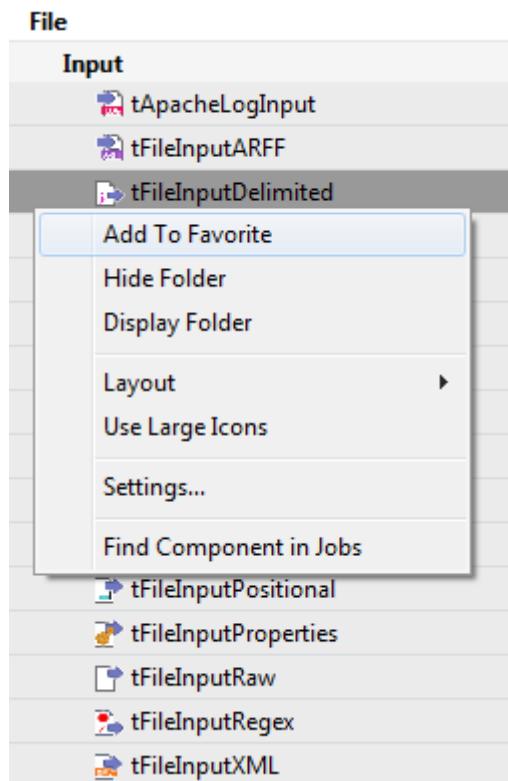
### B.2.1.5. How to set the Palette favorite

The **Palette** offers you search and favorite possibilities that by turn facilitate its usage.

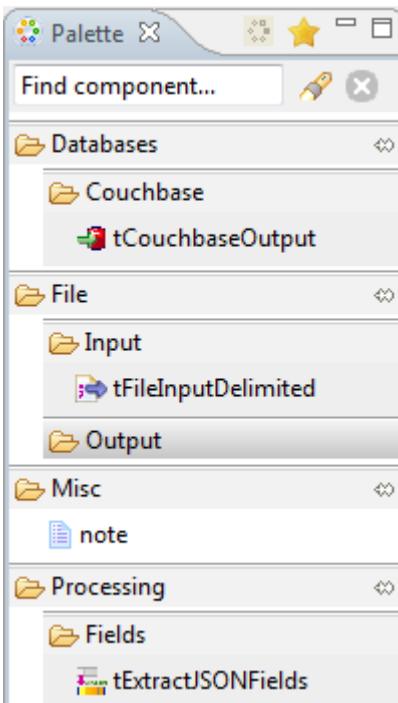
You can add/remove components to/from the **Palette** favorite view in order to have a quick access to all the components that you mostly use.

To do so:

1. From the **Palette**, right-click the component you want to add to **Palette** favorite and select **Add To Favorite**.



2. Do the same for all the components you want to add to the **Palette** favorite then click the **Favorite** button in the upper right corner of the **Palette** to display the **Palette** favorite.



Only the components added to the favorite are displayed.

To delete a component from the **Palette** favorite, right-click the component you want to remove from the favorite and select **Remove From Favorite**.

To restore the **Palette** standard view, click the **Standard** button in the upper right corner of the **Palette**.

### B.2.1.6. How to change components layout in the Palette

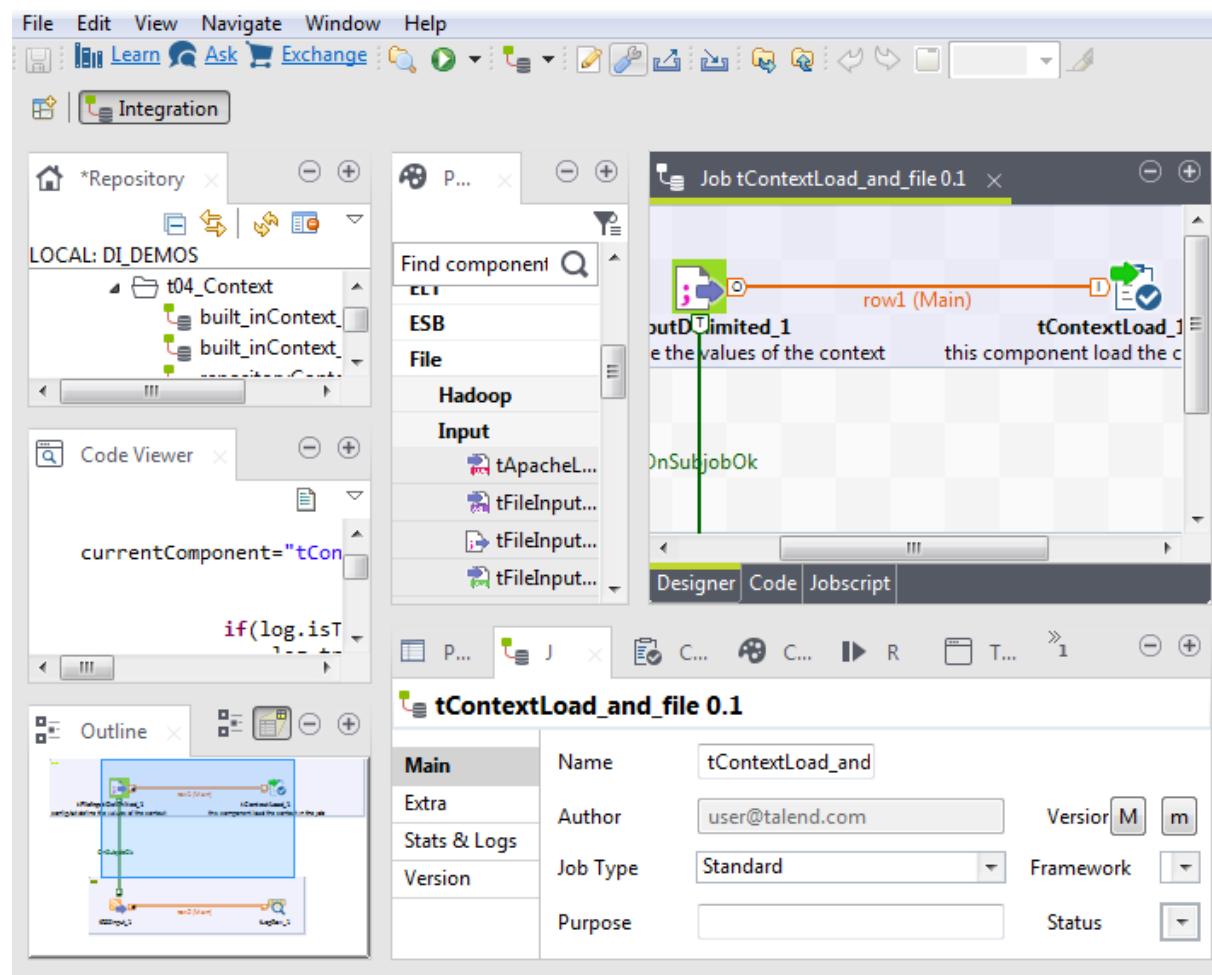
You can change the layout of the component list in the **Palette** to display them in columns or in lists, as icons only or as icons with short description.

You can also enlarge the component icons for better readability of the component list.

To do so, right-click any component family in the **Palette** and select the desired option in the contextual menu or click **Settings** to open the **[Palette Settings]** window and fine-tune the layout.

## B.2.2. How to change panels positions

All panels in the open Studio can be moved around according to your needs.



All you need to do is to click the head border of a panel or to click a tab, hold down the mouse button and drag the panel to the target destination. Release to change the panel position.

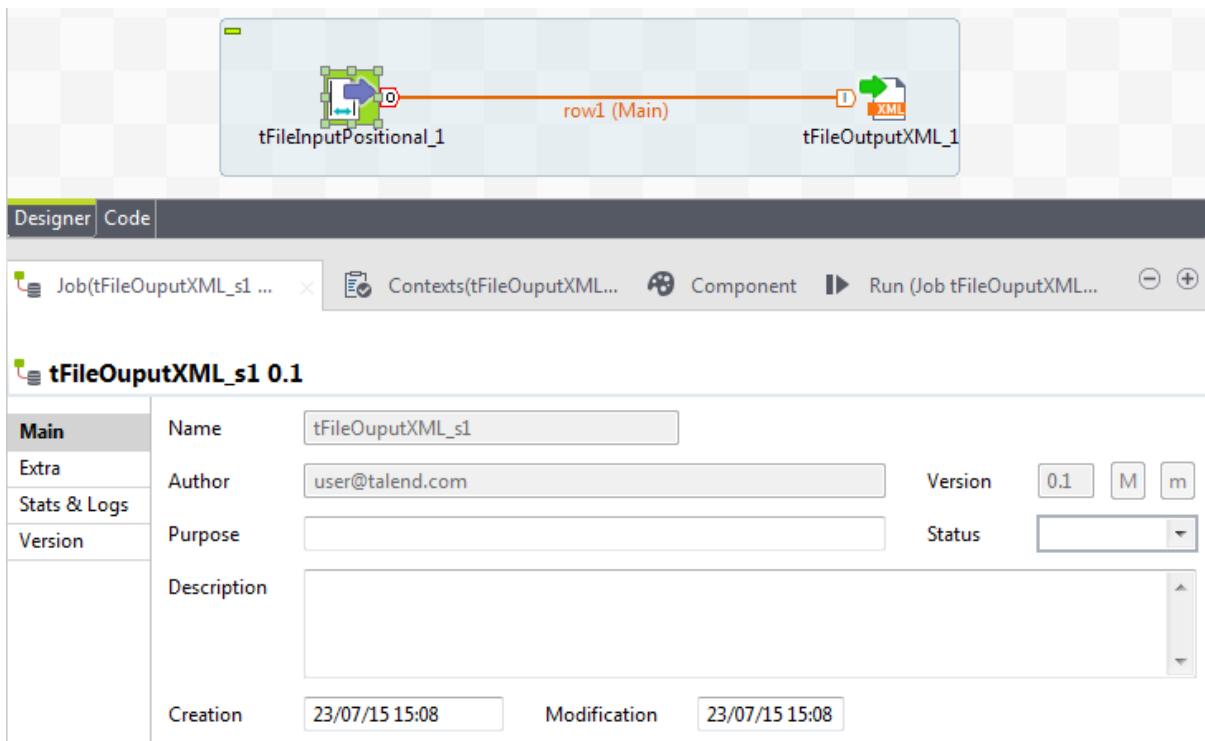
Click the minimize/maximize icons (/) to minimize the corresponding panel or maximize it. For more information on how to display or hide a panel/view, see [How to display Job configuration tabs/views](#).

Click the close icon () to close a tab/view. To reopen a view, click **Window > Show View > Talend**, then click the name of the panel you want to add to your current view or see [Shortcuts and aliases](#).

If the **Palette** does not show or if you want to set it apart in a panel, go to **Window > Show view...> General > Palette**. The **Palette** opens in a separate view that you can move around wherever you like within the perspective.

## B.2.3. How to display Job configuration tabs/views

The configuration tabs are located in the lower half of the design workspace of the **Integration** perspective. Each tab opens a view that displays detailed information about the selected element in the design workspace.



The **Component**, **Run Job**, and **Contexts** views gather all information relative to the graphical elements selected in the design workspace or the actual execution of the open Job.

 By default, when you launch *Talend Studio* for the first time, the **Problems** tab will not be displayed until the first Job is created. After that, **Problems** tab will be displayed in the tab system automatically.

The **Modules** and **Scheduler[deprecated]** tabs are located in the same tab system as the **Component**, **Logs** and **Run Job** tabs. Both views are independent from the active or inactive Jobs open on the design workspace.

Some of the configuration tabs are hidden by default such as the **Error Log**, **Navigator**, **Job Hierarchy**, **Problems**, **Modules** and **Scheduler[deprecated]** tabs. You can show hidden tabs in this tab system and directly open the corresponding view if you select **Window > Show view** and then, in the open dialog box, expand the corresponding node and select the element you want to display.

For detailed description about these tabs, see [Configuration tabs](#).

## B.3. Filtering entries listed in the Repository tree view

*Talend Studio* provides the possibility to choose what nodes, Jobs or items you want to list in the **Repository** tree view.

You can filter the **Repository** tree view by job name, Job status, the user who created the Job/items or simply by selecting/clearing the check box next to the node/ item you want to display/hide in the view. You can also set several filters simultaneously.

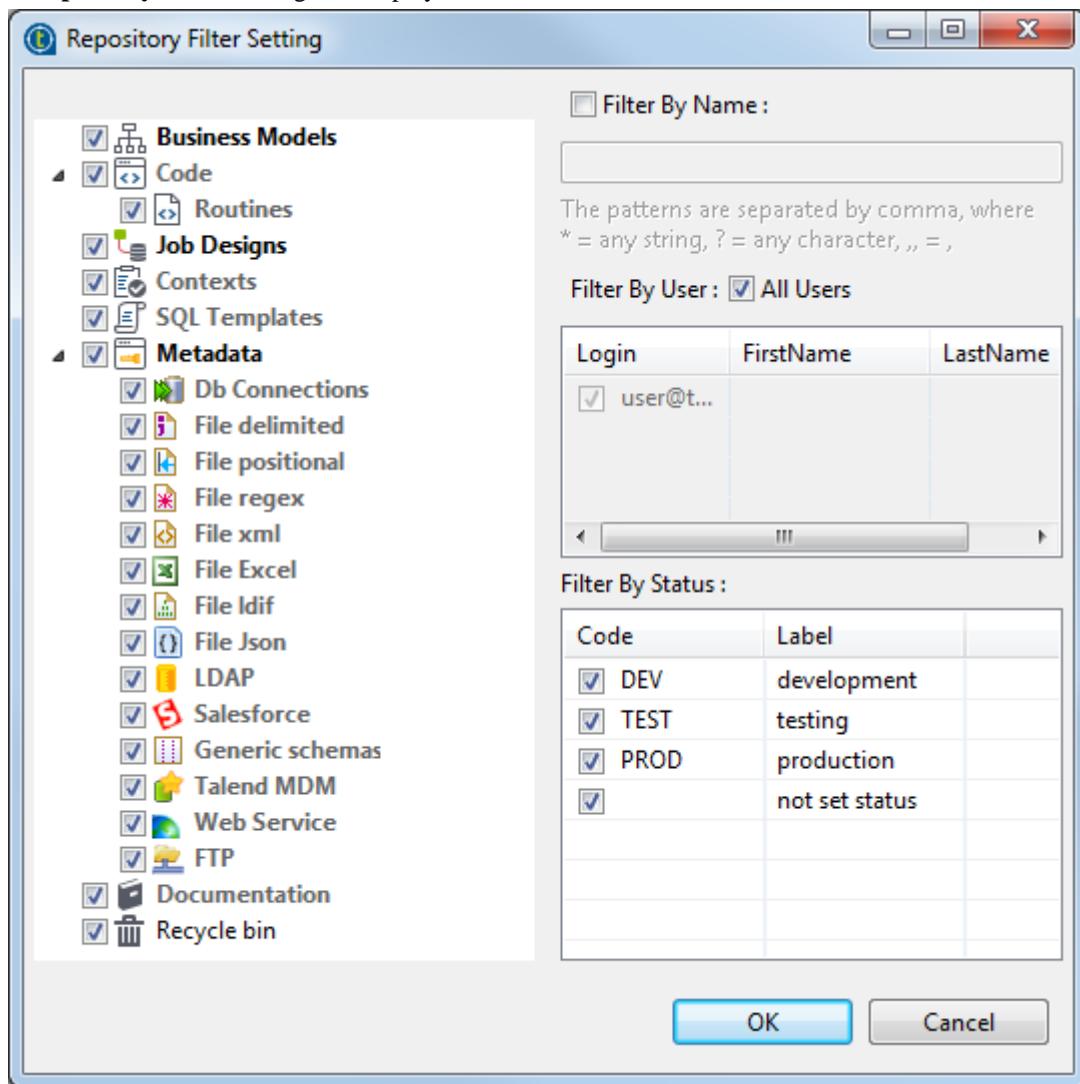
### B.3.1. How to filter by Job name

To filter Jobs listed in the **Repository** tree view by Job name, complete the following:

1.

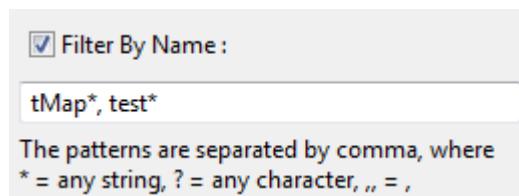
In the Studio, click the  icon in the upper right corner of the **Repository** tree view and select **Filter settings** from the contextual menu.

The **[Repository Filter]** dialog box displays.



2. Select the **Filter By Name** check box.

The corresponding field becomes available.

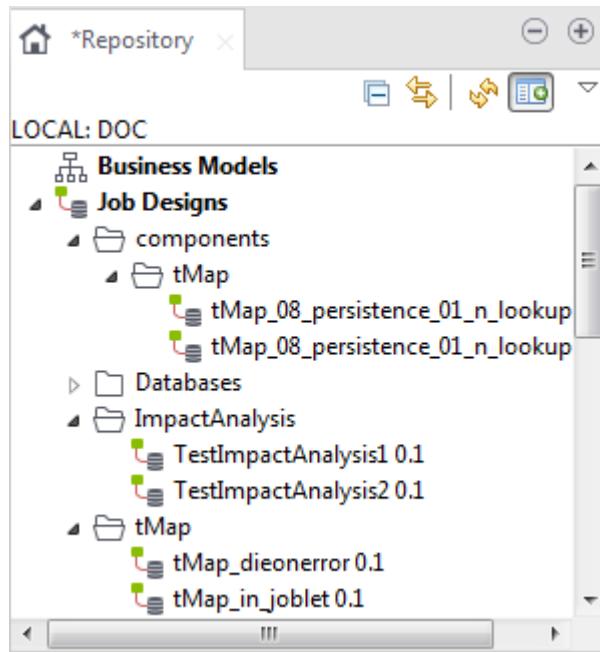


3. Follow the rules set below the field when writing the patterns you want to use to filter the Jobs.

In this example, we want to list in the tree view all Jobs that start with *tMap* or *test*.

4. In the **[Repository Filter]** dialog box, click **OK** to validate your changes and close the dialog box.

Only the Jobs that correspond to the filter you set are displayed in the tree view, those that start with *tMap* and *test* in this example



You can switch back to the by-default tree view, which lists all nodes, Jobs and items, by simply clicking the icon . This will cause the green plus sign appended on the icon to turn to a minus red sign ().

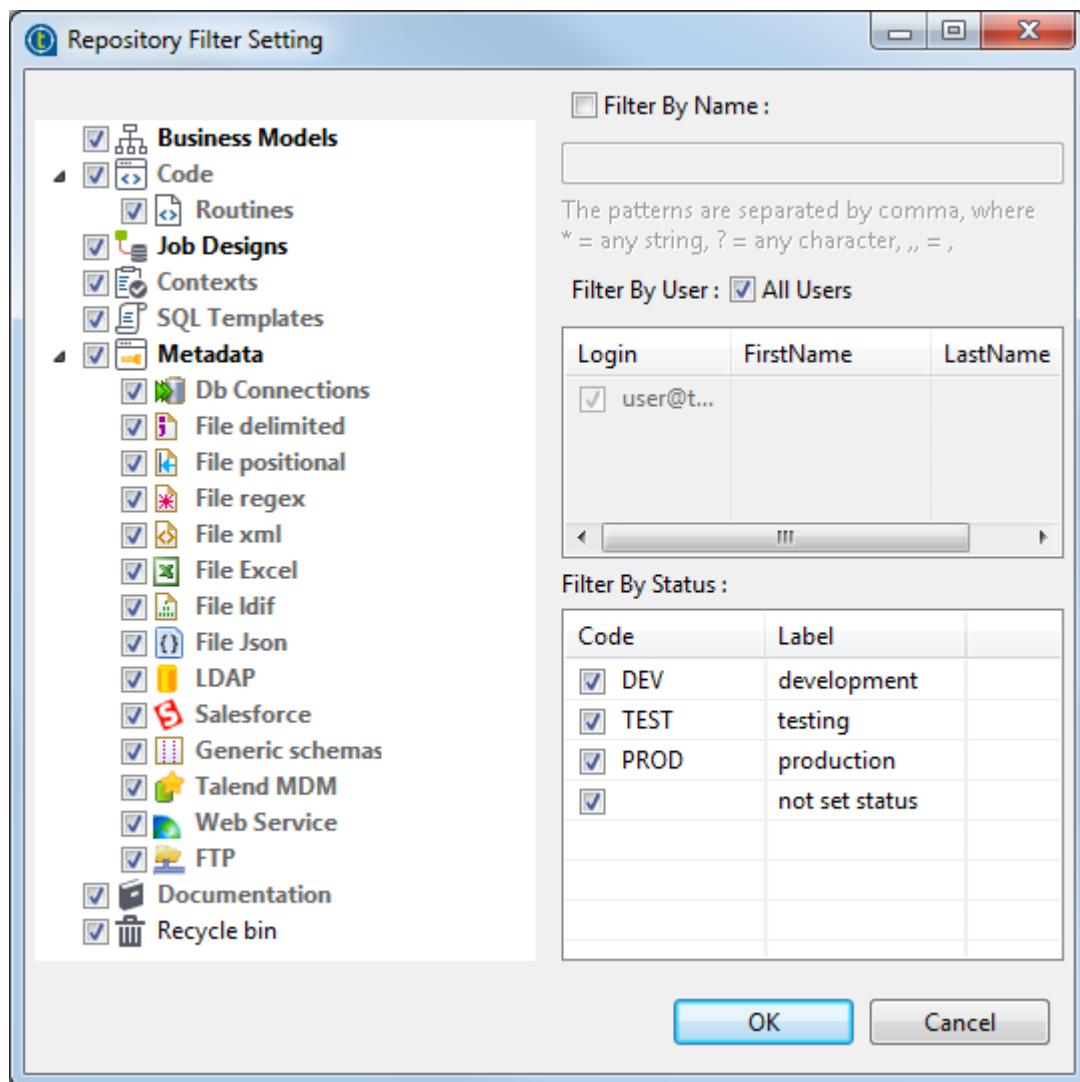
### B.3.2. How to filter by user

To filter entries in the **Repository** tree view by the user who created the Jobs/items, complete the following:

1.

In the Studio, click the icon in the upper right corner of the **Repository** tree view and select **Filter settings** from the contextual menu.

The **[Repository Filter]** dialog box displays.



- Clear the **All Users** check box.

The corresponding fields in the table that follows become available.

Filter By User : <input type="checkbox"/> All Users		
Login	FirstName	LastName
<input checked="" type="checkbox"/> user@talend.com	User	User
<input checked="" type="checkbox"/> dev1@talend.com	dev1	Talend
<input checked="" type="checkbox"/> dev2@talend.com	dev2	Talend

This table lists the authentication information of all the users who have logged in to *Talend Studio* and created a Job or an item.

- Clear the check box next to a user if you want to hide all the Jobs/items created by him/her in the **Repository** tree view.
- Click **OK** to validate your changes and close the dialog box.

All Jobs/items created by the specified user will disappear from the tree view.



You can switch back to the by-default tree view, which lists all nodes, Jobs and items, by simply clicking the icon . This will cause the green plus sign appended on the icon to turn to a minus red sign ().

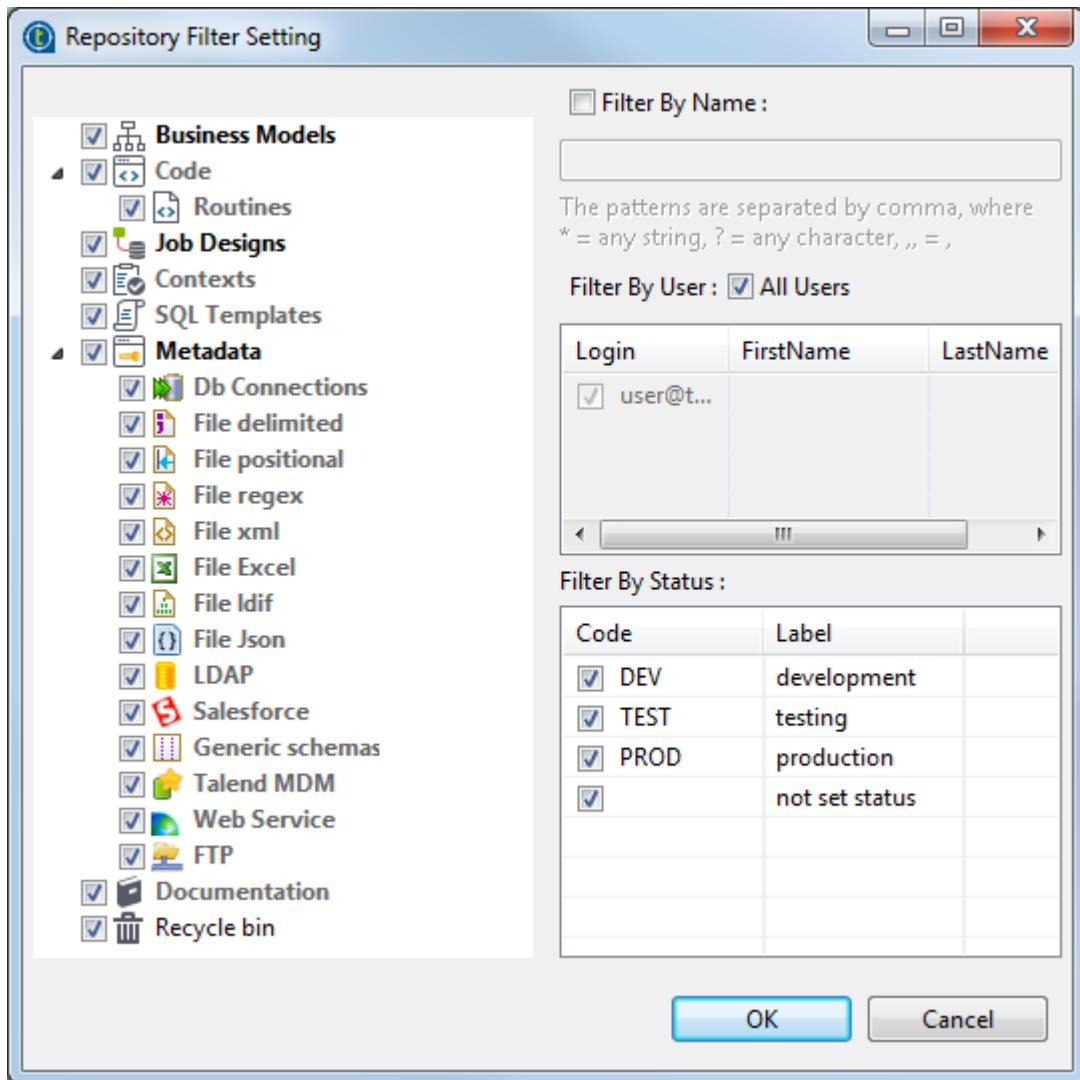
### B.3.3. How to filter by job status

To filter Jobs in the **Repository** tree view by the job status, complete the following:

1.

- In the Studio, click the icon in the upper right corner of the **Repository** tree view and select **Filter settings** from the contextual menu.

The **[Repository Filter]** dialog box displays.



- In the **Filter By Status** area, clear the check boxes next to the status type if you want to hide all the Jobs that have the selected status.
- Click **OK** to validate your changes and close the dialog box.

All Jobs that have the specified status will disappear from the tree view.



You can switch back to the by-default tree view, which lists all nodes, Jobs and items, by simply clicking the icon . This will cause the green plus sign appended on the icon to turn to a minus red sign ().

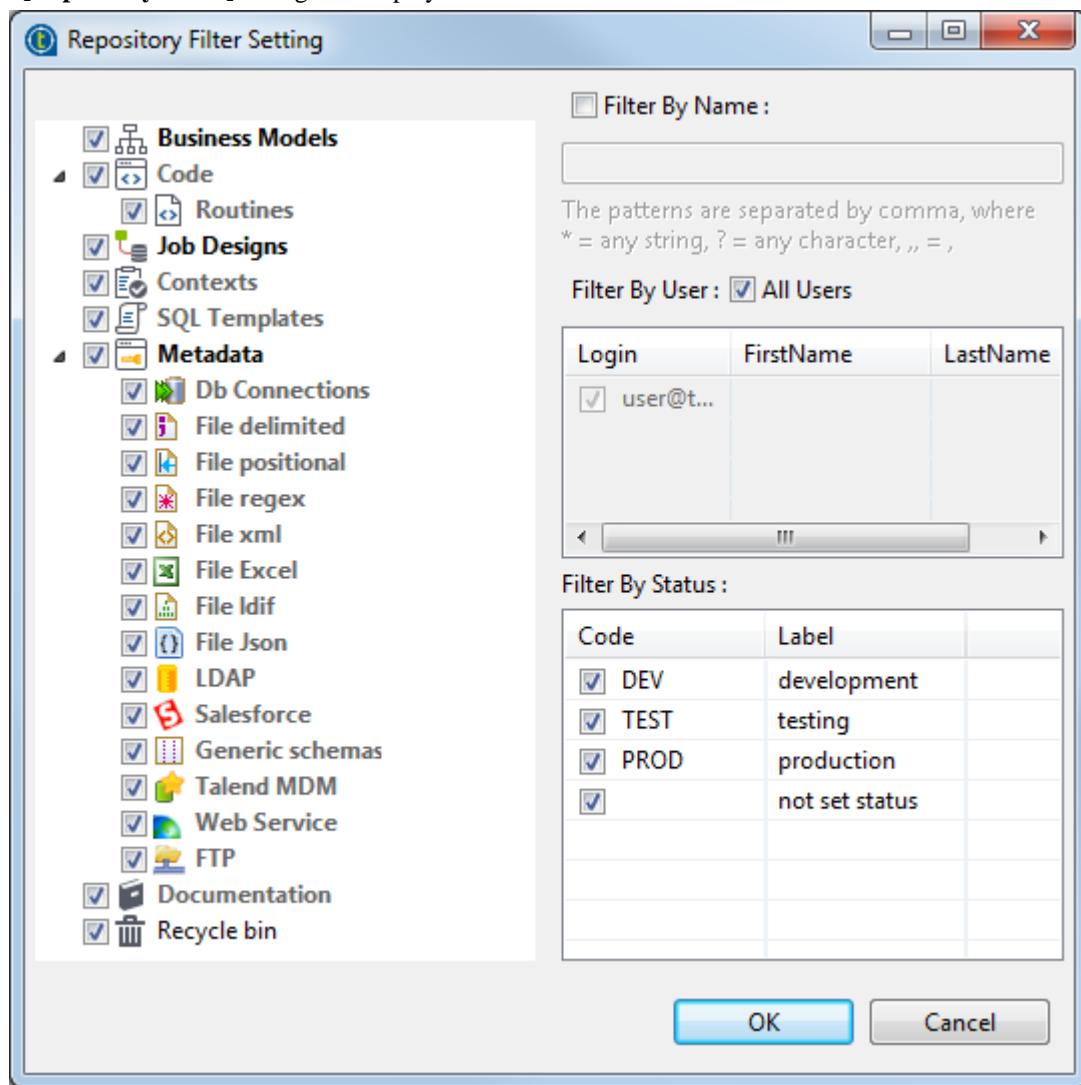
## B.3.4. How to choose what repository nodes to display

To filter repository nodes, complete the following:

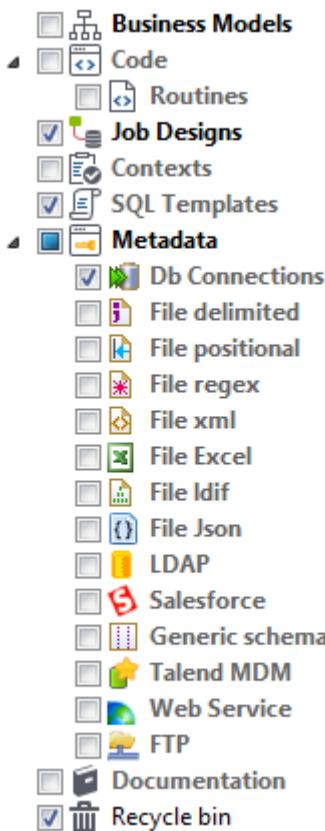
1.

In the **Integration** perspective of the Studio, click the  icon in the upper right corner of the **Repository** tree view and select **Filter settings** from the contextual menu.

The **[Repository Filter]** dialog box displays.



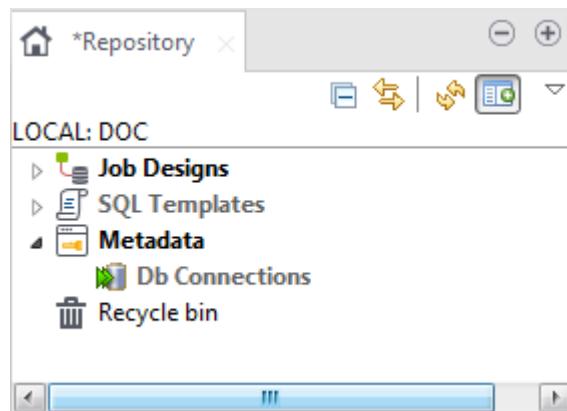
2. Select the check boxes next to the nodes you want to display in the **Repository** tree view.



Consider, for example, that you want to show in the tree view all the Jobs listed under the **Job Designs** node, three of the folders listed under the **SQL Templates** node and one of the metadata items listed under the **Metadata** node.

- Click **OK** to validate your changes and close the dialog box.

Only the nodes/folders for which you selected the corresponding check boxes are displayed in the tree view.



If you do not want to show all the Jobs listed under the **Job Designs** node, you can filter the Jobs using the **Filter By Name** check box. For more information on filtering Jobs, see [How to filter by Job name](#).

## B.4. Setting Talend Studio preferences

You can define various properties for all the perspectives of *Talend Studio* according to your needs and preferences.

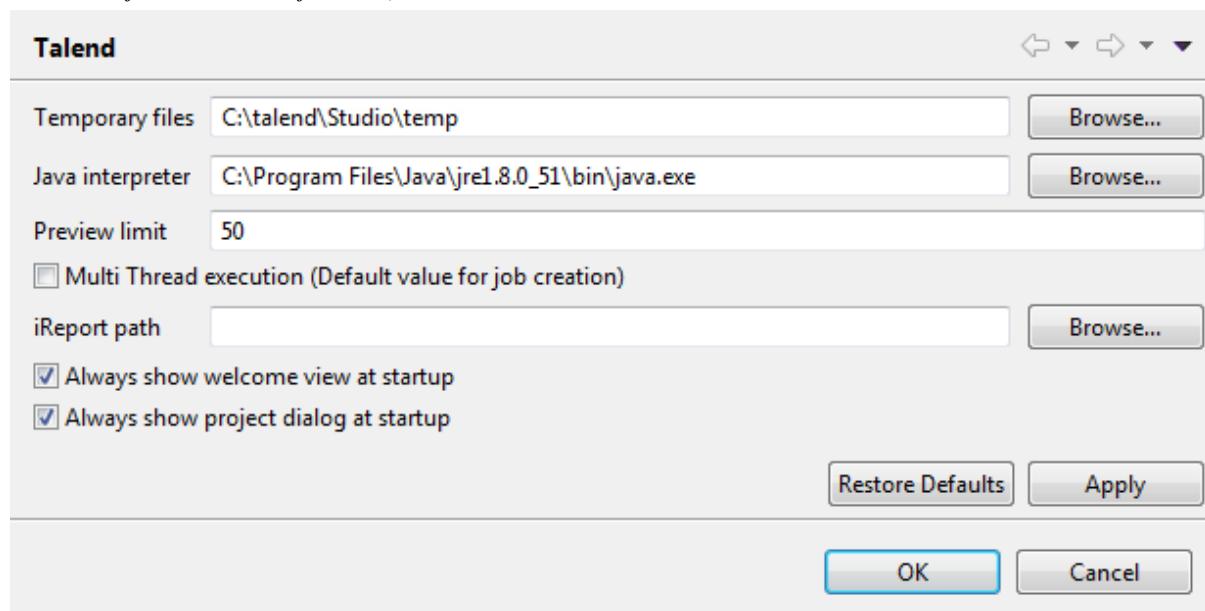
Numerous settings you define can be stored in the **Preference** and thus become your default values for all new Jobs you create.

The following sections describe specific settings that you can set as preference.

First, click the **Window** menu of *Talend Studio*, then select **Preferences**.

## B.4.1. Java Interpreter path (Talend)

The Java Interpreter path is set based on the location of the Java file on your computer (for example *C:\Program Files\Java\jre1.8.0\_51\bin\java.exe*).



To customize your Java Interpreter path:

1. If needed, click the **Talend** node in the tree view of the **[Preferences]** dialog box.
2. Enter a path in the **Java interpreter** field if the default directory does not display the right path.

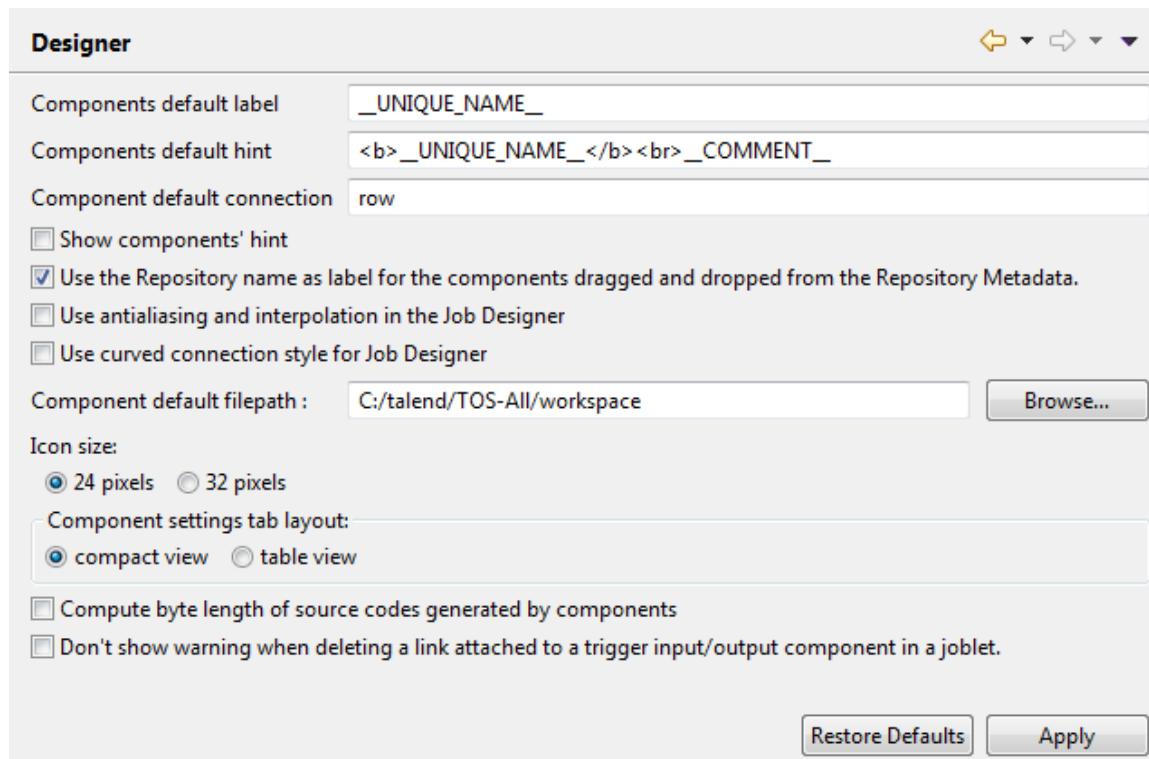
On the same view, you can also change the preview limit and the path to the temporary files or the OS language.

## B.4.2. Designer preferences (Talend > Appearance)

You can set component and Job design preferences to let your settings be permanent in the Studio.

1. From the menu bar, click **Window > Preferences** to open the **[Preferences]** dialog box.
2. Expand the **Talend > Appearance** node.
3. Click **Designer** to display the corresponding view.

On this view, you can define the way component names and hints will be displayed.



4. Select the relevant check boxes to customize your use of the *Talend Studio* design workspace.

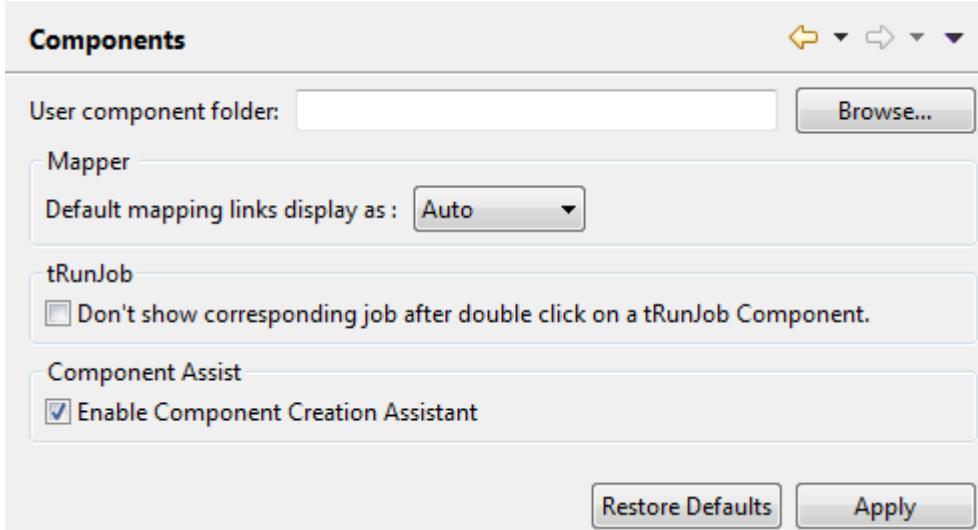
### B.4.3. How to define the user component folder (Talend > Components)

You can create and develop your own custom components for use in the **Integration** perspective of *Talend Studio*.

The following procedure applies only to the external components. For the preferences of all the components, see [How to change specific component settings \(Talend > Components\)](#).

The user component folder is the folder that contains the components you created and/or the ones you downloaded from TalendForge. To define it, proceed as follows:

1. In the tree view of the [Preferences] dialog box, expand the **Talend** node and select **Components**.



2. Enter the **User component folder** path or browse to the folder that contains the custom components to be added to the **Palette** of the Studio.

In order to be imported to the **Palette** of the Studio, the custom components have to be in separate folders located at the root of the component folder you have defined.

3. Click **Apply** and then **OK** to validate the preferences and close the dialog box.

The Studio restarts and the external components are added to the **Palette**.

This configuration is stored in the metadata of the workspace. If the workspace of *Talend Studio* changes, you have to reset this configuration again.

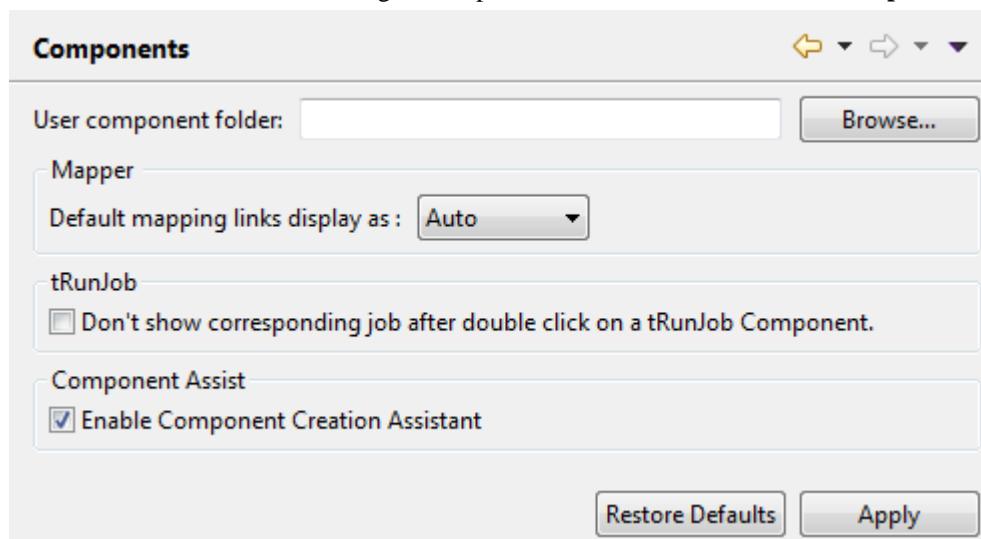
## B.4.4. How to change specific component settings (Talend > Components)

You can modify some specific component settings such as the default mapping link display.

The following procedure applies to the external components and to the components included in the Studio. For the preferences specific to the user components, see [How to define the user component folder \(Talend > Components\)](#).

To modify those specific components settings, proceed as follows:

1. In the tree view of the **[Preferences]** dialog box, expand the **Talend** node and select **Components**.



2. From the **Default mapping links display as** list, select the mapping link type you want to use in the **tMap**.
3. Under **tRunJob**, select the check box if you do not want the corresponding Job to open upon double clicking a **tRunJob** component.



You will still be able to open the corresponding Job by right clicking the **tRunJob** component and selecting **Open tRunJob Component**.

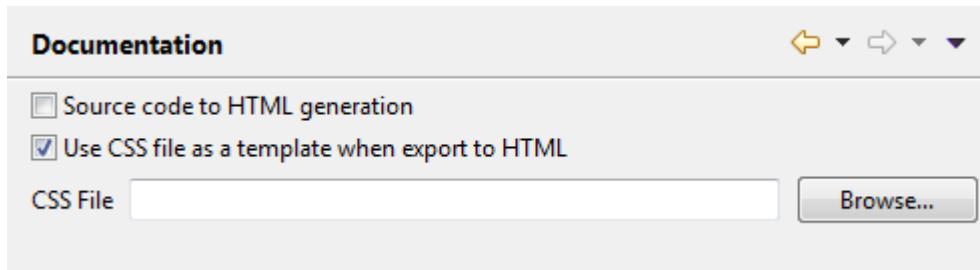
4. Under **Component Assist**, select the **Enable Component Creation Assistant** check box if you want to be able to add a component by typing its name in the design workspace. For more information, see [Adding components to the Job](#).
5. Click **Apply** and then **OK** to validate the set preferences and close the dialog box.

This configuration is stored in the metadata of the workspace. If the workspace of *Talend Studio* changes, you have to reset this configuration again.

## B.4.5. Documentation preferences (Talend > Documentation)

You can include the source code on the generated documentation.

1. From the menu bar, click **Window > Preferences** to open the **[Preferences]** dialog box.
2. Expand the **Talend** node and click **Documentation** to display the documentation preferences.



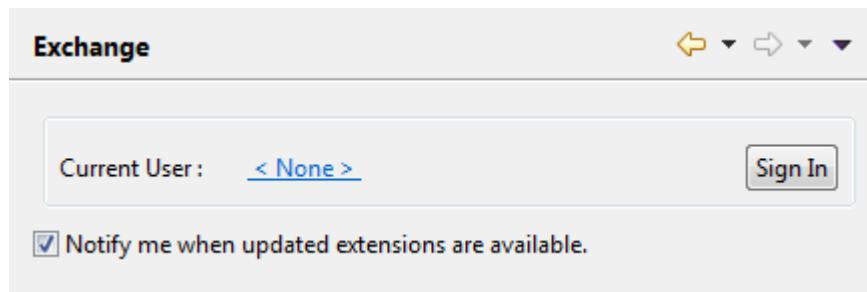
3. Customize the documentation preferences according to your needs:
  - Select the **Source code to HTML generation** check box to include the source code in the HTML documentation that you will generate.
  - Select the **Use CSS file as a template when export to HTML** check box to activate the **CSS File** field if you need to use a CSS file to customize the exported HTML files.

For more information on documentation, see [How to generate HTML documentation](#) and [Documentation tab](#).

## B.4.6. Exchange preferences (Talend > Exchange)

You can set preferences related to your connection with **Talend** Exchange, which is part of the **Talend** Community, in *Talend Studio*. To do so:

1. From the menu bar, click **Window > Preferences** to open the **[Preferences]** dialog box.
2. Expand the **Talend** node and click **Exchange** to display the **Exchange** view.



3. Set the Exchange preferences according to your needs:
  - If you are not yet connected to the **Talend** Community, click **Sign In** to go to the **Connect to TalendForge** page to sign in using your **Talend** Community credentials or create a **Talend** Community account and then sign in.

If you are already connected to the **Talend** Community, your account is displayed and the **Sign In** button becomes **Sign Out**. To get disconnected from the **Talend** Community, click **Sign Out**.

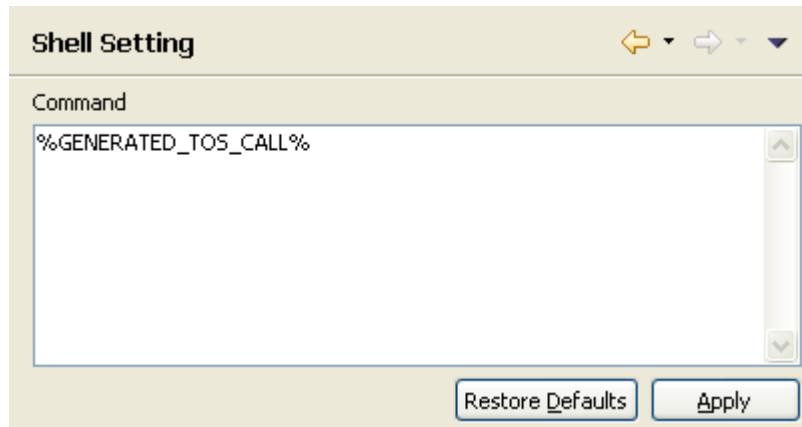
- By default, while you are connected to the **Talend** Community, whenever an update to an installed community extension is available, a dialog box appears to notify you about it. If you often check for community extension updates and you do not want that dialog box to appear again, clear the **Notify me when updated extensions are available** check box.

For more information on connecting to the **Talend** Community, see the Getting Started Guide. For more information on using community extensions in the Studio, see [How to download/upload Talend Community components](#).

## B.4.7. Adding code by default (Talend > Import/Export)

You can add pieces of code by default at the beginning and at the end of the code of your Job.

1. From the menu bar, click **Window > Preferences** to open the **[Preferences]** dialog box.
2. Expand the **Talend** and **Import/Export** nodes in succession and then click **Shell Setting** to display the relevant view.

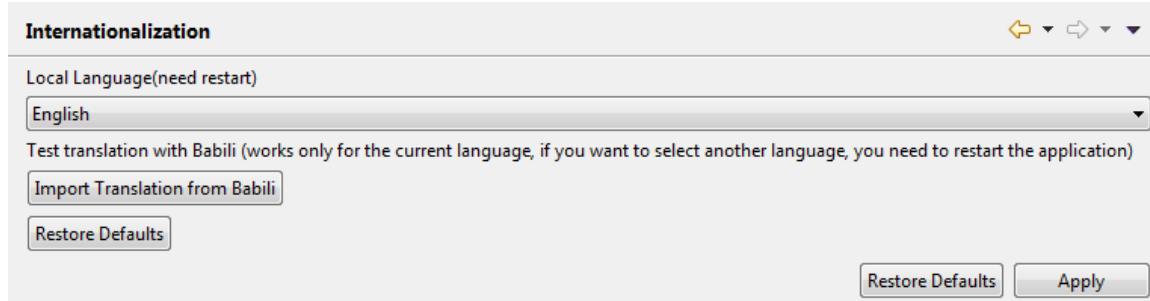


3. In the **Command** field, enter your piece/pieces of code before or after %GENERATED\_TOS\_CALL% to display it/them before or after the code of your Job.

## B.4.8. Language preferences (Talend > Internationalization)

You can set language preferences in *Talend Studio*. To do so:

1. From the menu bar, click **Window > Preferences** to open the **[Preferences]** dialog box.
2. Expand the **Talend** node and click **Internationalization** to display the relevant view.

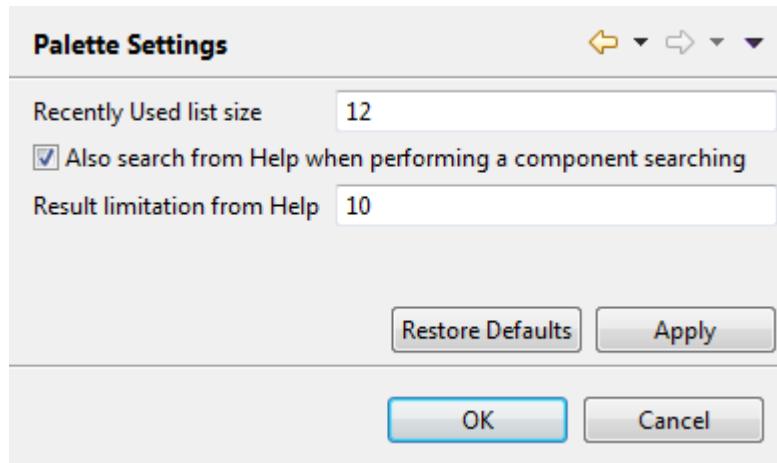


3. From the **Local Language** list, select the language you want to use for the graphical interface of *Talend Studio*.
4. Click **Apply** and then **OK** to validate your change and close the **[Preferences]** dialog box.
5. Restart the Studio to display the graphical interface in the selected language.

## B.4.9. Palette preferences (Talend > Palette Settings)

From **Palette Settings** view you can set preferences for component searching from the **Palette** and even from the component list that appears on the design workspace when adding a component without using the **Palette**.

1. From the menu bar, click **Window > Preferences** to display the **[Preferences]** dialog box.
2. Expand the **Talend** node and click **Palette Settings** to display the **Palette Settings** view.

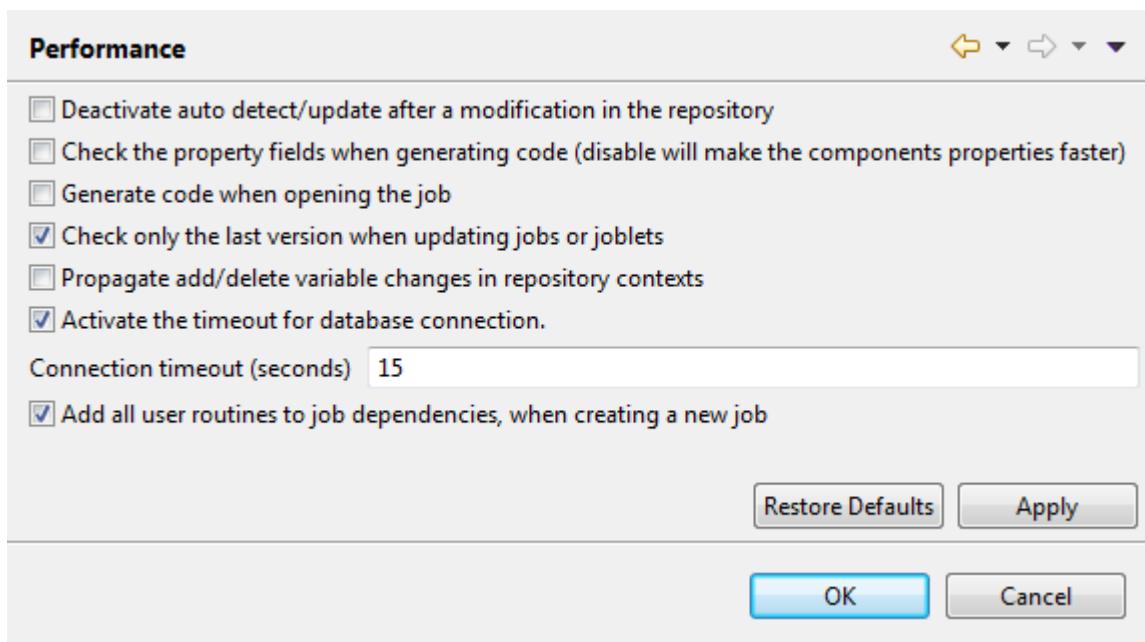


3. To limit number of components that can be displayed on the **Recently Used** list, enter your preferred number in the **Recently used list size** field.
4. To enable searching a component using a phrase that describes the function or purpose of the component as search keywords in the search field of the **Palette** or in the text field that appears on the design workspace, select the **Also search from Help when performing a component searching** check box. With this check box selected, you can find your component on the **Palette** or on the component list on the design workspace as long as you can find it from the F1 Help information by using the same descriptive phrase as keywords.
5. To change the number of the search result entries when using a descriptive phrase as search keywords, enter your preferred number in the **Result limitation from Help** field.

## B.4.10. Performance preferences (Talend > Performance)

You can set performance preferences according to your use of *Talend Studio*. To do so, proceed as follows:

1. From the menu bar, click **Window > Preferences** to open the **[Preferences]** dialog box.
2. Expand the **Talend** node and click **Performance** to display the repository refresh preference.



You can improve your performance when you deactivate automatic refresh.

3. Set the performance preferences according to your use of *Talend Studio*:

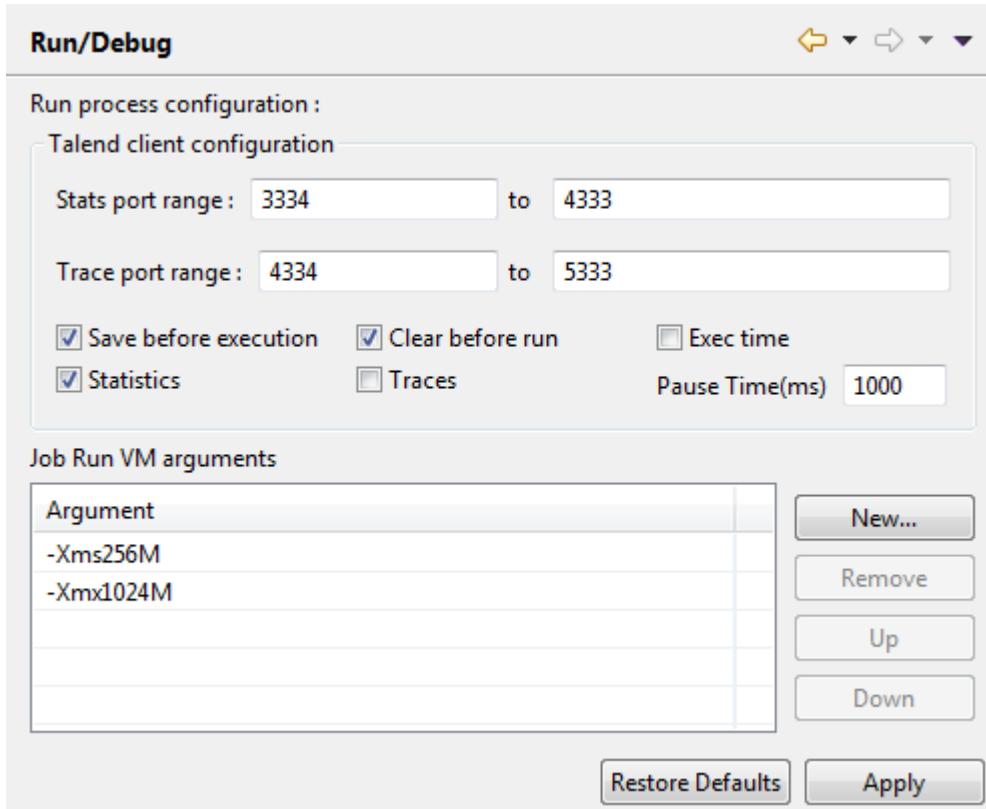
- Select the **Deactivate auto detect/update after a modification in the repository** check box to deactivate the automatic detection and update of the repository.
  - Select the **Check the property fields when generating code** check box to activate the audit of the property fields of the component. When one property filed is not correctly filled in, the component is surrounded by red on the design workspace.
-  You can optimize performance if you disable property fields verification of components, for example if you clear the **Check the property fields when generating code** check box.
- Select the **Generate code when opening the job** check box to generate code when you open a Job.
  - Select the **Check only the last version when updating jobs or joblets** check box to only check the latest version when you update a Job.
  - Select the **Propagate add/delete variable changes in repository contexts** to propagate variable changes in the Repository Contexts.
  - Select the **Activate the timeout for database connection** check box to establish database connection time out. Then set this time out in the **Connection timeout (seconds)** field.
  - Select the **Add all user routines to job dependencies, when create new job** check box to add all user routines to Job dependencies upon the creation of new Jobs.

## B.4.11. Debug and Job execution preferences (Talend > Run/Debug)

You can set your preferences for debug and job executions in *Talend Studio*. To do so:

1. From the menu bar, click **Window > Preferences** to display the **[Preferences]** dialog box.

2. Expand the **Talend** node and click **Run/Debug** to display the relevant view.



- In the **Talend client configuration** area, you can define the execution options to be used by default:

<b>Stats port range</b>	Specify a range for the ports used for generating statistics, in particular, if the ports defined by default are used by other applications.
<b>Trace port range</b>	Specify a range for the ports used for generating traces, in particular, if the ports defined by default are used by other applications.
<b>Save before run</b>	Select this check box to save your Job automatically before its execution.
<b>Clear before run</b>	Select this check box to delete the results of a previous execution before re-executing the Job.
<b>Exec time</b>	Select this check box to show Job execution duration.
<b>Statistics</b>	Select this check box to show the statistics measurement of data flow during Job execution.
<b>Traces</b>	Select this check box to show data processing during job execution.
<b>Pause time</b>	Enter the time you want to set before each data line in the traces table.

- In the **Job Run VM arguments** list, you can define the parameter of your current JVM according to your needs. The by-default parameters **-Xms256M** and **-Xmx1024M** correspond respectively to the minimal and maximal memory capacities reserved for your Job executions.

If you want to use some JVM parameters for only a specific Job execution, for example if you want to display the execution result for this specific Job in Japanese, you need open this Job's **Run** view and then in the **Run** view, configure the advanced execution settings to define the corresponding parameters.

For further information about the advanced execution settings of a specific Job, see [How to set advanced execution settings](#).

For more information about possible parameters, check the site <http://www.oracle.com/technetwork/java/javase/tech/vmoptions-jsp-140102.html>.

## B.4.12. Displaying special characters for schema columns (Talend > Specific settings)

You may need to retrieve a table schema that contains columns written with special characters like Chinese, Japanese, Korean. In this case, you need to enable *Talend Studio* to read the special characters. To do so:

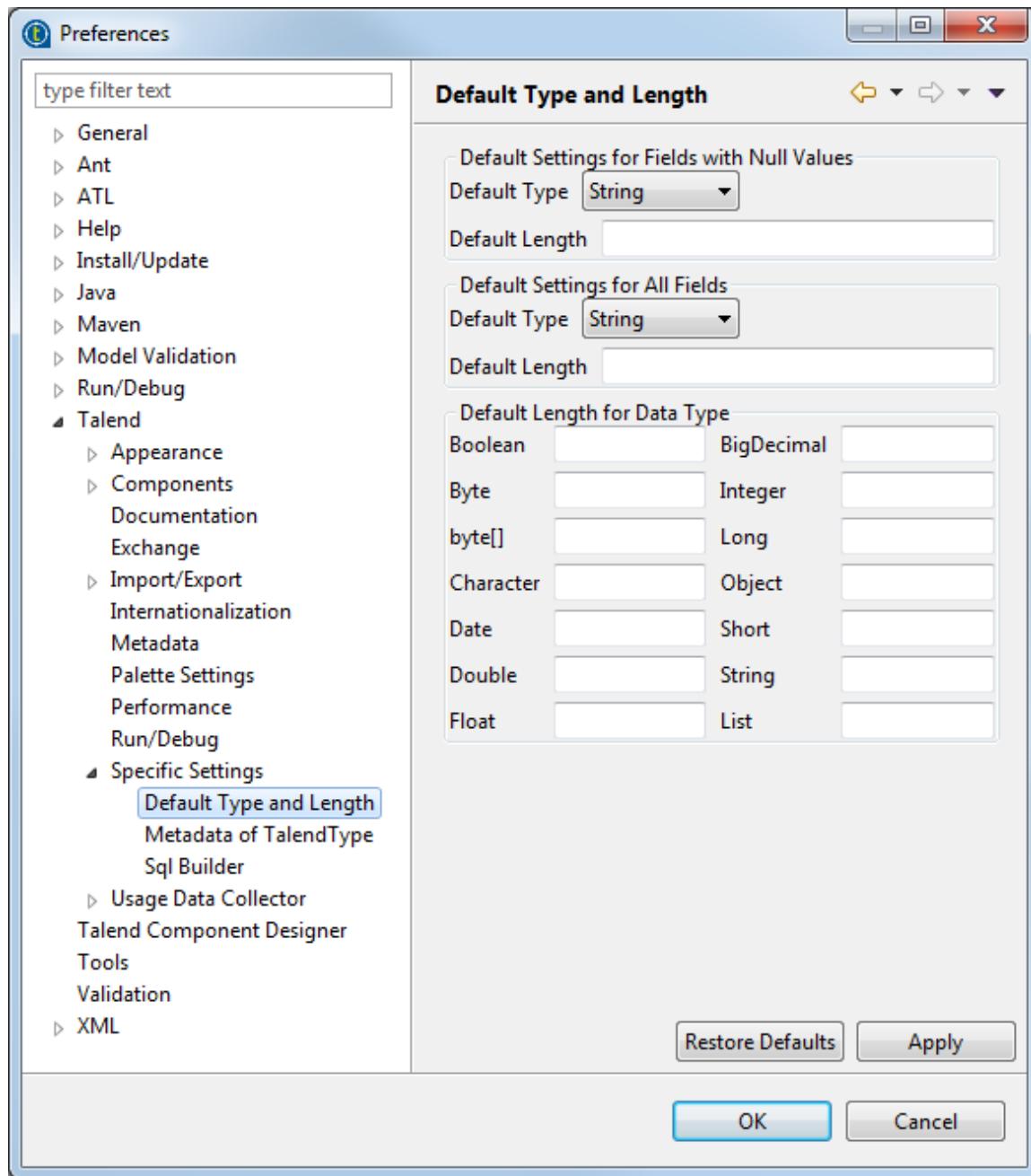
1. From the menu bar, click **Window > Preferences** to open the **[Preferences]** dialog box.
2. On the tree view of the opened dialog box, expand the **Talend** node.
3. Click the **Specific settings** node to display the corresponding view on the right of the dialog box.
4. Select the **Allow specific characters (UTF8,...) for columns of schemas** check box.



## B.4.13. Schema preferences (Talend > Specific Settings)

You can define the default data length and type of the schema fields of your components.

1. From the menu bar, click **Window > Preferences** to open the **[Preferences]** dialog box.
2. Expand the **Talend** node, and click **Specific Settings > Default Type and Length** to display the data length and type of your schema.



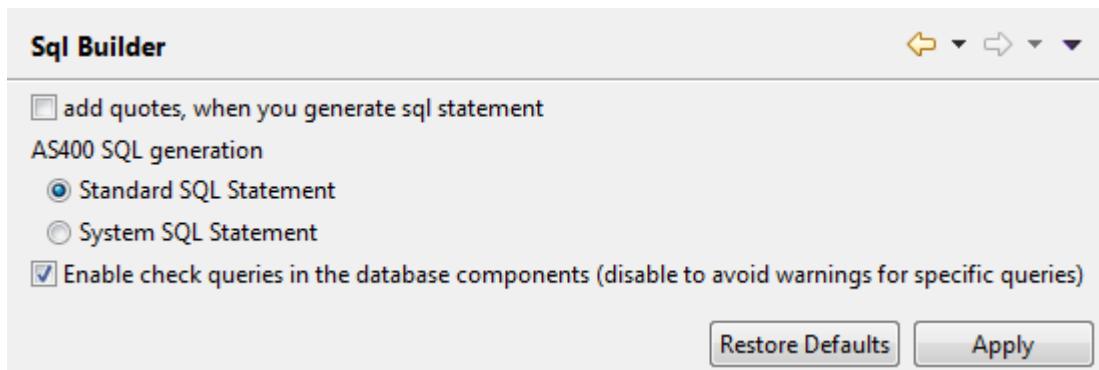
3. Set the parameters according to your needs:

- In the **Default Settings for Fields with Null Values** area, fill in the data type and the field length to apply to the null fields.
- In the **Default Settings for All Fields** area, fill in the data type and the field length to apply to all fields of the schema.
- In the **Default Length for Data Type** area, fill in the field length for each type of data.

## B.4.14. SQL Builder preferences (Talend > Specific Settings)

You can set your preferences for the SQL Builder. To do so:

- From the menu bar, click **Window > Preferences** to open the **[Preferences]** dialog box.
- Expand the **Talend** and **Specific Settings** nodes in succession and then click **Sql Builder** to display the relevant view.



- Customize the SQL Builder preferences according to your needs:
  - Select the **add quotes, when you generated sql statement** check box to precede and follow column and table names with inverted commas in your SQL queries.
  - In the **AS400 SQL generation** area, select the **Standard SQL Statement** or **System SQL Statement** check boxes to use standard or system SQL statements respectively when you use an AS/400 database.
  - Clear the **Enable check queries in the database components (disable to avoid warnings for specific queries)** check box to deactivate the verification of queries in all database components.

## B.4.15. Usage Data Collector preferences (Talend > Usage Data Collector)

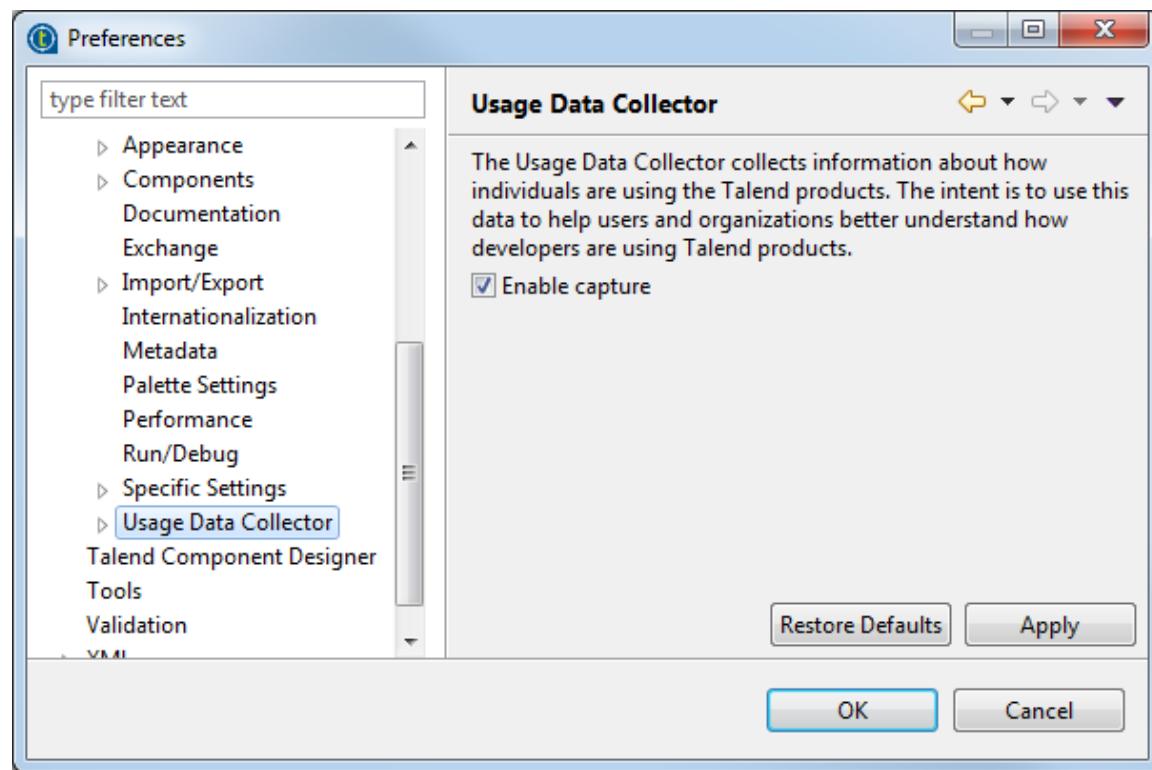
By allowing *Talend Studio* to collect your Studio usage statistics, you help users better understand **Talend** products and help **Talend** better learn how users are using the products, thus enabling **Talend** to improve product quality and performance to serve users better.

By default, *Talend Studio* automatically collects your Studio usage data and sends this data on a regular basis to servers hosted by **Talend**. You can view the usage data collection and upload information and customize the Usage Data Collector preferences according to your needs.



Be assured that only the Studio usage statistics data will be collected and none of your private information will be collected and transmitted to **Talend**.

- From the menu bar, click **Window > Preferences** to display the **[Preferences]** dialog box.
- Expand the **Talend** node and click **Usage Data Collector** to display the **Usage Data Collector** view.



3. Read the message about the Usage Data Collector, and, if you do not want the Usage Data Collector to collect and upload your Studio usage information, clear the **Enable capture** check box.
4. To have a preview of the usage data captured by the Usage Data Collector, expand the **Usage Data Collector** node and click **Preview**.

Preview	
Key	Value
tokenStudio	
stopUsageCollection	0
typeStudio	tos_di
uniqueId	cSrRusnn4QmFxwMvoHoAxHP3MR71UtrF3...
version	6.1.0.20150714_1935-SNAPSHOT
properties	
tos.count.businessModels	0
tos.count.componentsPerJob	0.2
tos.count.contextVariablesPerJob	0.0
tos.count.debugRuns	0
tos.count.generatedJobDocs	0
tos.count.jobExports	5
tos.count.jobs	18
tos.count.jobsPerProject	9.0
tos.count.localProjects	2
tos.count.metadata	0
tos.count.runs	5
tos.count.userComponents	0
tos.job.frequentComponents	
tos.user.components	<Empty>

5. To customize the usage data upload interval and view the date of the last upload, click **Uploading** under the **Usage Data Collector** node.

Uploading	
Information gathered by the Usage Data Collector is periodically uploaded to servers hosted by Talend.	
Upload Period	<input type="text" value="10"/> Days
Last Upload	<input type="text"/>
<input type="button" value="Restore Defaults"/> <input type="button" value="Apply"/>	

- By default, if enabled, the Usage Data Collector collects the product usage data and sends it to **Talend** servers every 10 days. To change the data upload interval, enter a new integer value (in days) in the **Upload Period** field.
- The read-only **Last Upload** field displays the date and time the usage data was last sent to **Talend** servers.





## Appendix C. Theory into practice: Job examples

This chapter aims at users of *Talend Studio* who seek real-life use cases to help them take full control over the product.

## C.1. tMap Job example

To illustrate the way *Talend Studio* operates, find below a real-life example scenario. In this scenario, we will load a MySQL table with a file, that gets transformed on the fly. Then in a further step, we will select the data to be loaded using a dynamic filter.

Before actually starting the Job, let's inspect the input data and the expected output data.

### C.1.1. Input data

Our input file, the data of which will be loaded into the database table, lists clients from all over the State of California.

The file structure usually called **Schema** in *Talend Studio* includes the following columns:

- First name
- Last name
- Address
- City

### C.1.2. Output data

We want to load into the database, California clients living in a couple of Counties only: Orange and Los Angeles counties.

The table structure is slightly different, therefore the data expected to be loaded into the DB table should have the following structure:

- Key (key, Type: Integer)
- Name (Type: String, max. length: 40)
- Address (Type: String, max.length: 40)
- County (Type: String, max. length:40)

In order to load this table, we will need to use the following mapping process:

The `Key` column is fed with an auto-incremented integer.

The `Name` column is filled out with a concatenation of first and last names.

The `Address` column data comes from the equivalent `Address` column of the input file, but supports a upper-case transformation before the loading.

The `County` column is fed with the name of the County where the city is located using a reference file which will help filtering Orange and Los Angeles counties' cities.

### C.1.3. Reference data

As only Orange and Los Angeles counties data should be loaded into the database, we need to map cities of California with their respective county, in order to filter only Orange and Los Angeles ones.

To do so, we will use a reference file, listing cities that are located in Orange and Los Angeles counties such as:

City	County
Agoura Hills	Los Angeles
Alhambra	Los Angeles
Aliso Viejo	Orange
Anaheim	Orange
Arcadia	Los Angeles

The reference file in this Job is named *LosAngelesandOrangeCounties.txt*.

## C.1.4. Translating the scenario into a Job

In order to implement this scenario, let's break down the Job into four steps:

1. Creation of the Job, configuration of the input file parameters, and reading of the input file,
2. Mapping of data and transformations,
3. Definition of the reference file parameters, relevant mapping using the **tMap** component, and selection of inner join mode,
4. Redirection of the output into a MySQL table.

### C.1.4.1. Step 1: Job creation, input definition, file reading

Launch *Talend Studio*, and create a local project or import the demo project if you are launching *Talend Studio* for the first time. For more information, see [Working with projects](#).

The main window of *Talend Studio* is divided into several areas:

- On the left-hand side: the **Repository** tree view that holds Jobs, Business Models, Metadata, shared Code, Documentation and so on.
- In the center: the **Editor** (main Design area)
- At the bottom: **Component** and **Job** tabs
- On the right-hand side: the **Palette** of business or technical components depending on the software tool you are using within *Talend Studio*.

To the left of the Studio, the **Repository** tree view that gives an access to:

- The **Business Modeler**: For more information, see [Modeling a Business Model](#).
- The **Job Designer**: For details about this part, see [Getting started with a basic Job](#).
- The **Metadata Manager**: For details about this part, see [Managing Metadata](#).
- Contexts and routines: For details, see [Using contexts and variables](#) and [Managing routines](#).

To create the Job, right-click **Job Designs** in the **Repository** tree view and select **Create Job**.

In the dialog box displaying then, only the first field (**Name**) is required. Type in **California1** and click **Finish**.

An empty Job then opens on the main window and the **Palette** of technical components (by default, to the right of the Studio) comes up showing a dozen of component families such as: **Databases**, **Files**, **Internet**, **Data Quality** and so on, hundreds of components are already available.

To read the file *California\_Clients*, let's use the **tFileInputDelimited** component. This component can be found in the **File/Input** group of the **Palette**. Click this component then click to the left of the design workspace to place it on the design area.

Let's define now the reading properties for this component: File path, column delimiter, encoding... To do so, let's use the **Metadata Manager**. This tool offers numerous wizards that will help us to configure parameters and allow us to store these properties for a one-click re-use in all future Jobs we may need.

As our input file is a delimited flat file, let's select **File Delimited** on the right-click list of the **Metadata** folder in the **Repository** tree view. Then select **Create file delimited**. A wizard dedicated to delimited file thus displays:

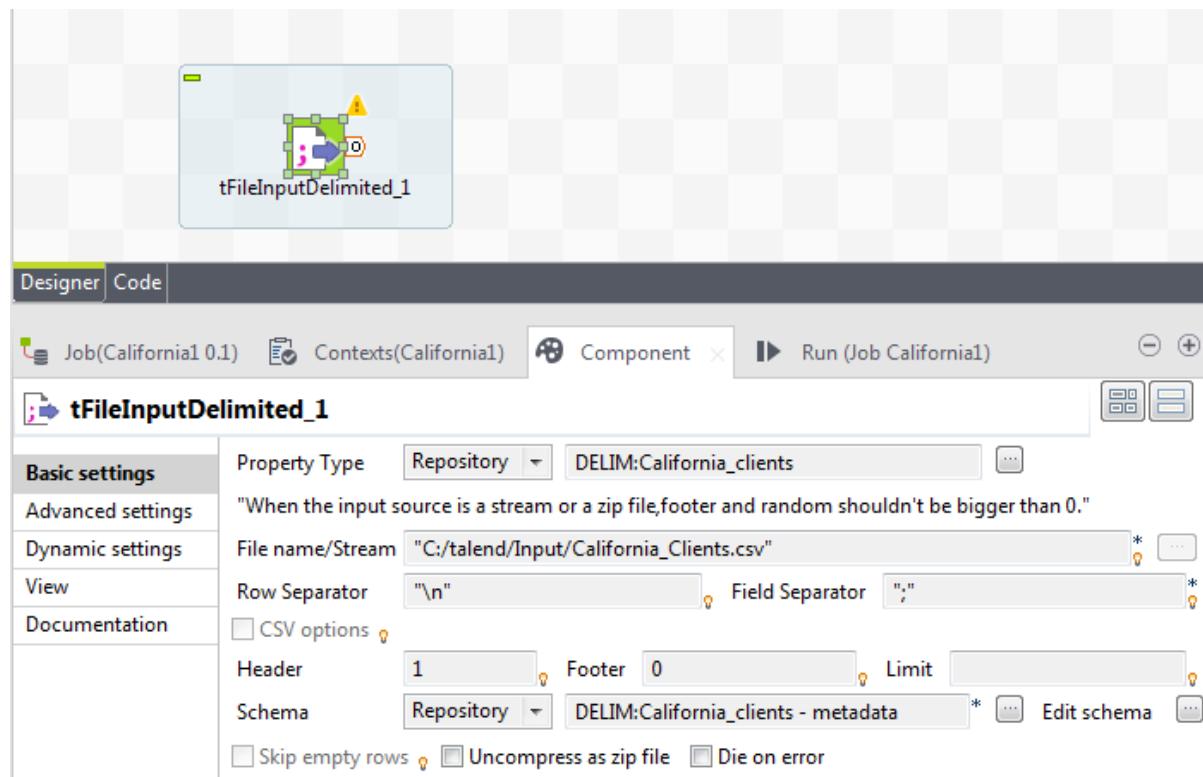
- At Step 1, only the **Name** field is required: simply type in **California\_clients** and go to the next Step.
- At Step 2, select the input file (*California\_Clients.csv*) via the **Browse...** button. Immediately an extract of the file shows on the Preview, at the bottom of the screen so that you can check its content. Click **Next**.
- At Step 3, we will define the file parameters: file encoding, line and column delimiters... As our input file is pretty standard, most default values are fine. The first line of our file is a header containing column names. To retrieve automatically these names, click **Set heading row as column names** then click **Refresh Preview**. And click **Next** to the last step.
- At Step 4, each column of the file is to be set. The wizard includes algorithms which guess types and length of the column based on the file first data rows. The suggested data description (called schema in *Talend Studio*) can be modified at any time. In this particular scenario, they can be used as is.

There you go, the **California\_clients** metadata is complete!

We can now use it in our input component. Select the **tFileInputDelimited** you had dropped on the design workspace earlier, and select the **Component** view at the bottom of the window.

Select the vertical tab **Basic settings**. In this tab, you'll find all technical properties required to let the component work. Rather than setting each one of these properties, let's use the Metadata entry we just defined.

Select **Repository** as **Property type** in the list. A new field shows: **Repository**, click "..." button and select the relevant Metadata entry on the list: **California\_clients**. You can notice now that all parameters get automatically filled out.



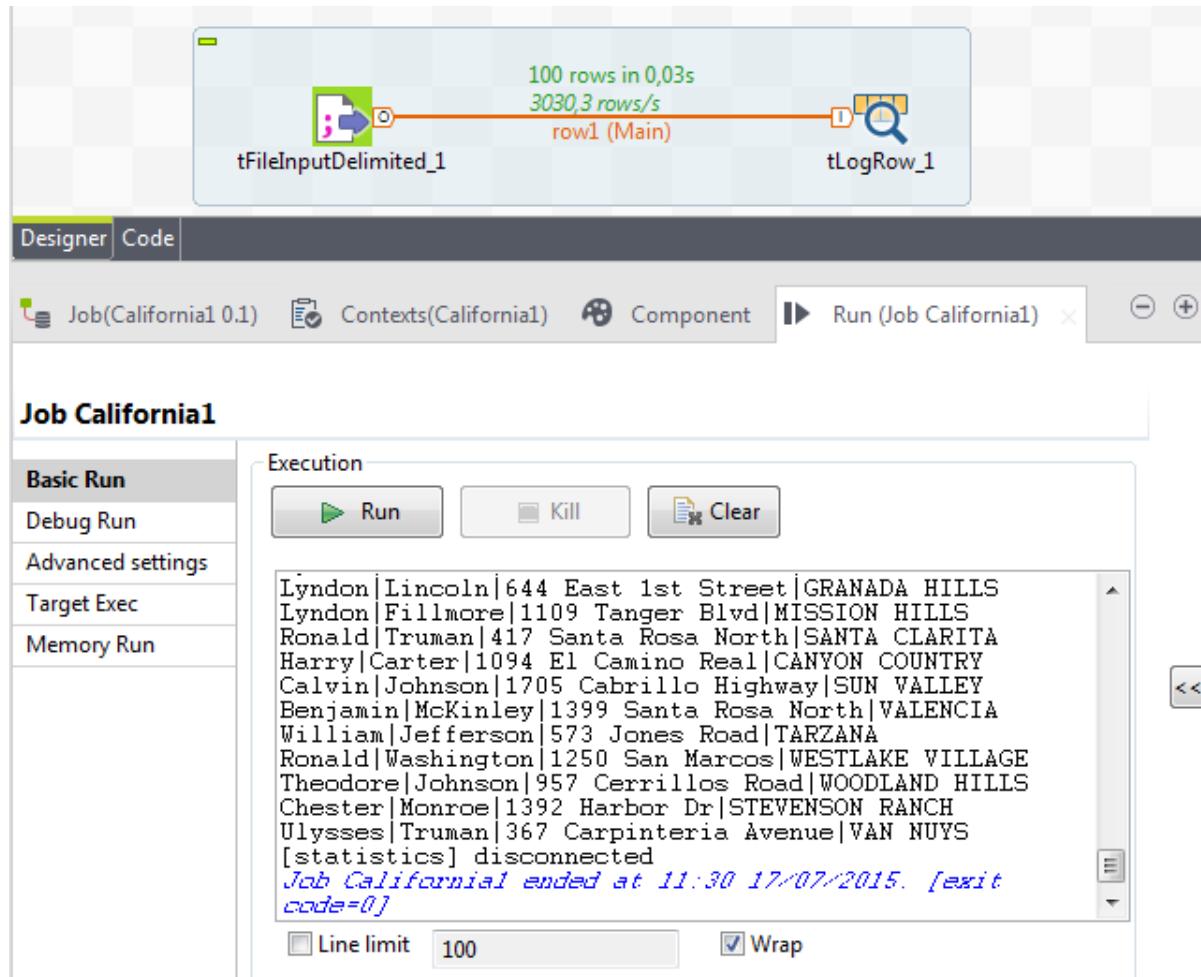
At this stage, we will terminate our flow by simply sending the data read from this input file onto the standard output (StdOut).

To do so, add a **tLogRow** component (from the **Logs & Errors** group).

To link both components, right-click the input component and select **Row/Main**. Then click the output component: **tLogRow**.

This Job is now ready to be executed. To run it, select the **Run** tab on the bottom panel.

Enable the statistics by selecting the **Statistics** check box in the **Advanced Settings** vertical tab of the **Run** view, then run the Job by clicking **Run** in the **Basic Run** tab.



The content of the input file display thus onto the console.

### C.1.4.2. Step 2: Mapping and transformations

We will now enrich our Job to include on-the-fly transformations. To implement these transformation, we need to add a **tMap** component to our Job. This component is multiple and can handle:

- multiple inputs and outputs
- search for reference (simple, cartesian product, first, last match...)
- join (inner, outer)
- transformations

- rejections
- and more...

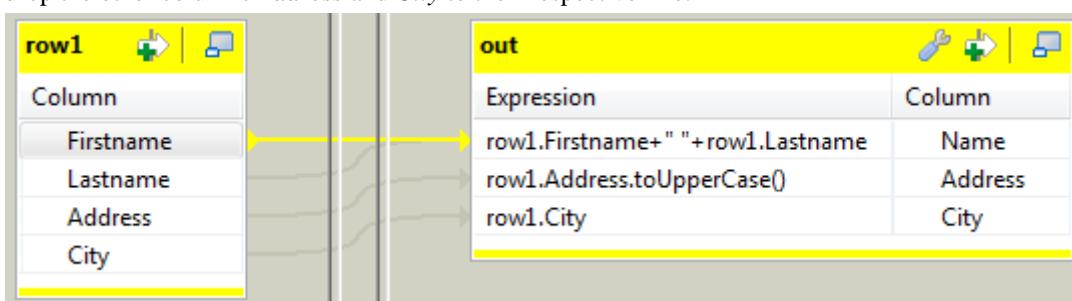
Remove the link that binds together the job's two components via a right-click the link, then **Delete** option. Then place the **tMap** of the **Processing** component group in between before linking the input component to the **tMap** as we did it previously.

Eventually to link the **tMap** to the standard output, right-click the **tMap** component, select **Row/\*New Output\* (Main)** and click the **tLogRow** component. Type in *out1* in the dialog box to implement the link. Logically, a message box shows up (for the back-propagation of schemas), ignore it by clicking on **No**.

Now, double-click the **tMap** to access its interface.

To the left, you can see the schema (description) of your input file (*row1*). To the right, your output is for the time being still empty (*out1*).

Drop the *Firstname* and *Lastname* columns to the right, onto the *Name* column as shown on the screen below. Then drop the other columns *Address* and *City* to their respective line.



Then carry out the following transformations on each column:

- Change the Expression of the *Name* column to `row1.Firstname + " " + row1.Lastname`. Concatenate the *Firstname* column with the *Lastname* column following strictly this syntax (in Java), in order for the columns to display together in one column.
- Change the Expression of the *Address* column to `row1.Address.toUpperCase()` which will thus change the address case to upper case.

Then remove the *Lastname* column from the *out1* table and increase the length of the remaining columns. To do so, go to the **Schema Editor** located at the bottom of the tMap editor and proceed as follows:

Column	Key	Type	✓	N..	Date Pattern...	Length	Precision	D...	Co...
Firstname	<input type="checkbox"/>	String	<input checked="" type="checkbox"/>			10	0		
Lastname	<input type="checkbox"/>	String	<input checked="" type="checkbox"/>			10	0		
Address	<input type="checkbox"/>	String	<input checked="" type="checkbox"/>			26	0		
City	<input type="checkbox"/>	String	<input checked="" type="checkbox"/>			22	0		

Below the table are several icons for managing the schema: a green plus sign, a red cross, arrows for moving columns, and icons for opening files and saving changes.

1. Select the column to be removed from the schema, and click the cross icon.
2. Select the column of which you need increase the length size.
3. Type in the length size you intend in the length column. In this example, change the length of every remaining column to 40.



As the first name and the last name of a client is concatenated, it is necessary to increase the length of the name column in order to match the full name size.

No transformation is made onto the *City* column. Click **OK** to validate the changes and close the Map editor interface.

If you run your Job at this stage (via the **Run** view as we did it before), you'll notice the changes that you defined are implemented.



For example, the addresses are displayed in upper case and the first names and last names are gathered together in the same column.

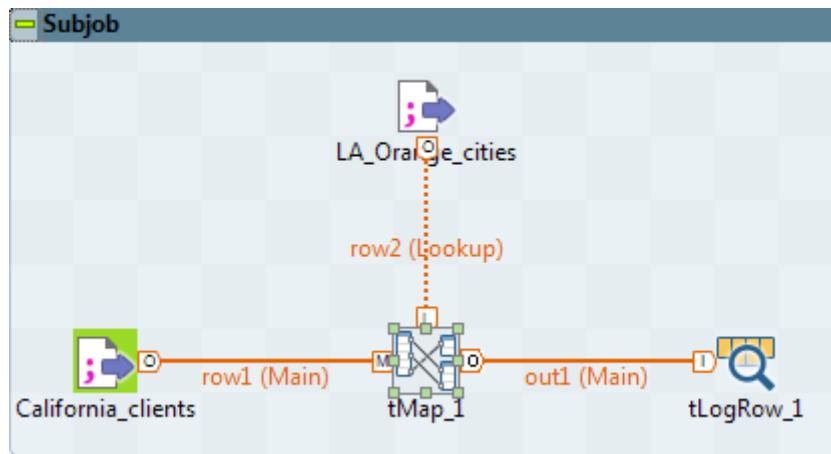
### C.1.4.3. Step 3: Reference file definition, re-mapping, inner join mode selection

Define the Metadata corresponding to the *LosAngelesandOrangeCounties.txt* file just the way we did it previously for *California\_clients* file, using the wizard.

At Step1 of the wizard, name this metadata entry: *LA\_Orange\_cities*.

Then drop this newly created metadata to the top of the design area to create automatically a reading component pointing to this metadata.

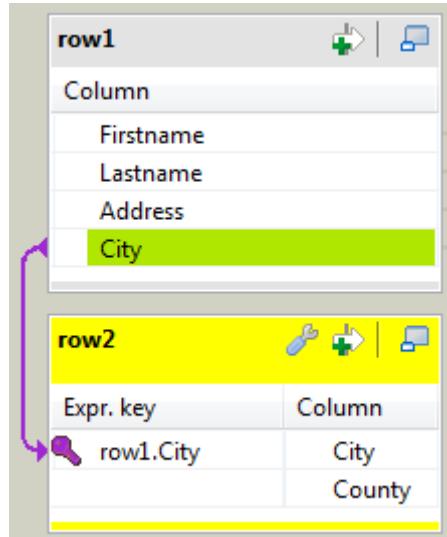
Then link this component to the **tMap** component.



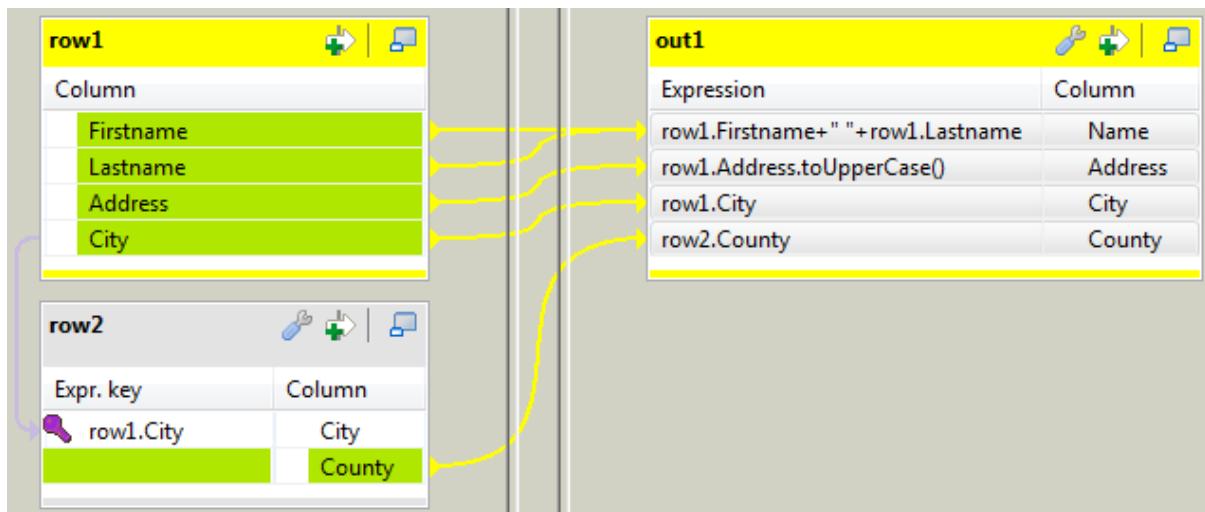
Double-click again on the **tMap** component to open its interface. Note that the reference input table (*row2*) corresponding to the LA and Orange county file, shows to the left of the window, right under your main input (*row1*).

Now let's define the join between the main flow and the reference flow. In this use case, the join is pretty basic to define as the *City* column is present in both files and the data match perfectly. But even though this was not the case, we could have carried out operations directly at this level to establish a link among the data (padding, case change...)

To implement the join, drop the *City* column from your first input table onto the *City* column of your reference table. A violet link then displays, to materialize this join.

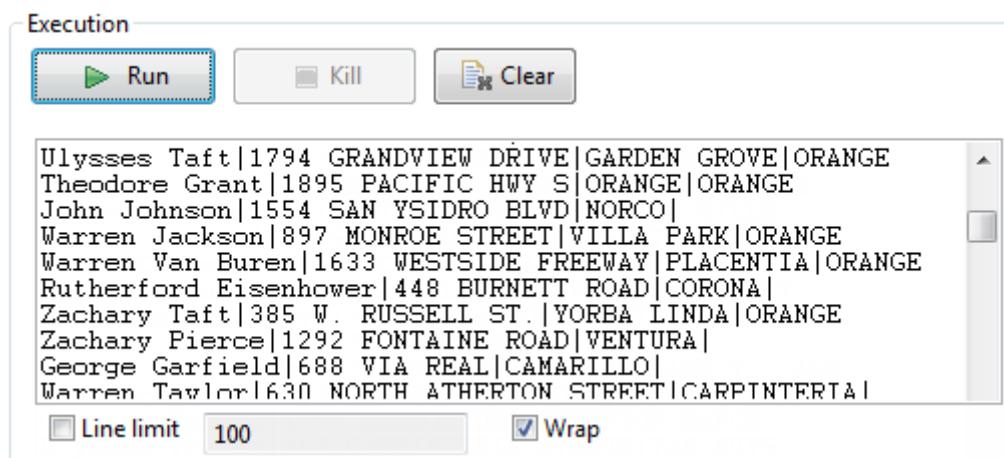


Now, we are able to use the *County* column from the reference table in the output table (out1).



Eventually, click the **OK** button to validate your changes, and run the new Job.

The following output should display on the console.



As you can notice, the last column is only filled out for *Los Angeles* and *Orange* counties' cities. For all other lines, this column is empty. The reason for this is that by default, the **tMap** implements a left outer join mode. If you want to filter your data to only display lines for which a match is found by the **tMap**, then open again the **tMap**, click the **tMap settings** button and select the **Inner Join** in the **Join Model** list on the reference table (*row2*).

#### C.1.4.4. Step 4: Output to a MySQL table

Our Job works perfectly! To finalize it, let's direct the output flow to a MySQL table.

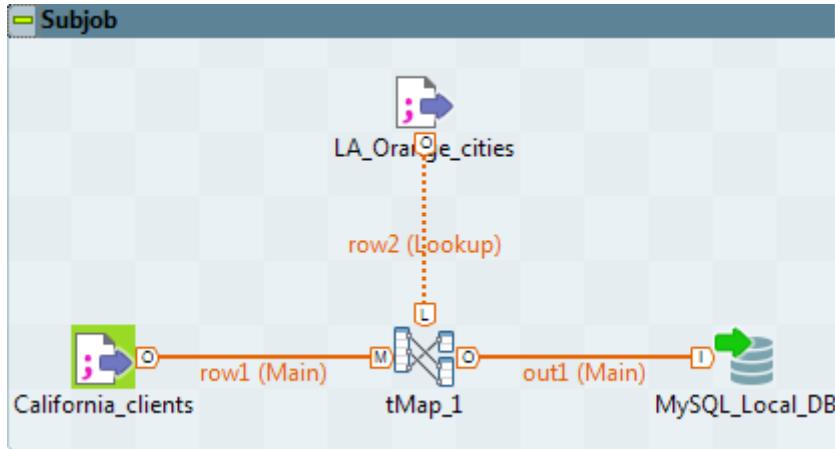
To do so, let's first create the Metadata describing the connection to the MySQL database. Double-click **Metadata/MySQL/DemoMySQL** in the referential (on the condition that you imported the Demo project properly). This opens the Metadata wizard.

On Step2 of the wizard, type in the relevant connection parameters. Check the validity of this connection by clicking on the **Check** button. Eventually, validate your changes, by clicking on **Finish**.

Drop this metadata to the right of the design workspace, while maintaining the **Ctrl** key down, in order to create automatically a **tMysqlOutput** component.

Remove the **tLogRow** component from your Job.

Reconnect the out1 output flow from the **tMap** to the new component **tMysqlOutput** (Right-click/Row/out1):



On the **Basic Settings** tab of this component:

1. Type in *LA\_Orange\_Clients* in the **Table** field, in order to name your target table which will get created on the fly.
2. Select the **Drop table if exists and create** option or on the **Action on table** field.
3. Click **Edit Schema** and click the **Reset DB type** button (DB button on the tool bar) in order to fill out automatically the DB type if need be.

Run again the Job. The target table should be automatically created and filled with data in less a second!

In this scenario, we did use only four different components out of hundreds of components available in the **Palette** and grouped according to different categories (databases, Web service, FTP and so on)!

And more components, this time created by the community, are also available on the community site ([talendforge.org](http://talendforge.org)).

## C.2. Using the output stream feature

The following use case aims to show how to use the output stream feature in a number of components in order to greatly improve the output performance.

In this scenario, a pre-defined csv file containing customer information is loaded in a database table. Then the loaded data is selected using a **tMap**, and output to a local file and to the console using the output stream feature.

### C.2.1. Input data

The input file, the data of which will be loaded into the database table, contains customer information of various aspects.

The file structure usually called **Schema** in *Talend Studio* includes the following columns:

- *id* (Type: Integer)
- *CustomerName* (Type: String)
- *CustomerAge* (Type: Integer)
- *CustomerAddress* (Type: String)

- *CustomerCity* (Type: String)
- *RegisterTime* (Type: Date)

## C.2.2. Output data

The **tMap** component is used to select *id*, *CustomerName* and *CustomerAge* columns from the input data. Then the selected data is output using the output stream feature.

Thus the expected output data should have the following structure:

- *id* (Type: Integer)
- *CustomerName* (Type: String)
- *CustomerAge* (Type: Integer)

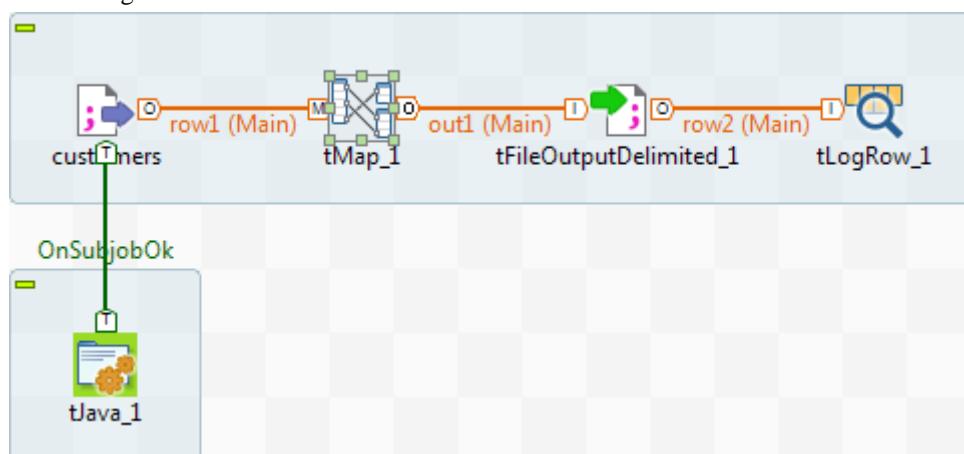
All the three columns above come from the respective columns in the input data.

## C.2.3. Translating the scenario into a Job

In order to implement this scenario, break down the Job into four steps:

1. Create the Job, define the schema for the input data, and read the input file according to the defined schema.
2. Set the command to enable the output stream feature.
3. Map the data using the **tMap** component.
4. Output the selected data stream.

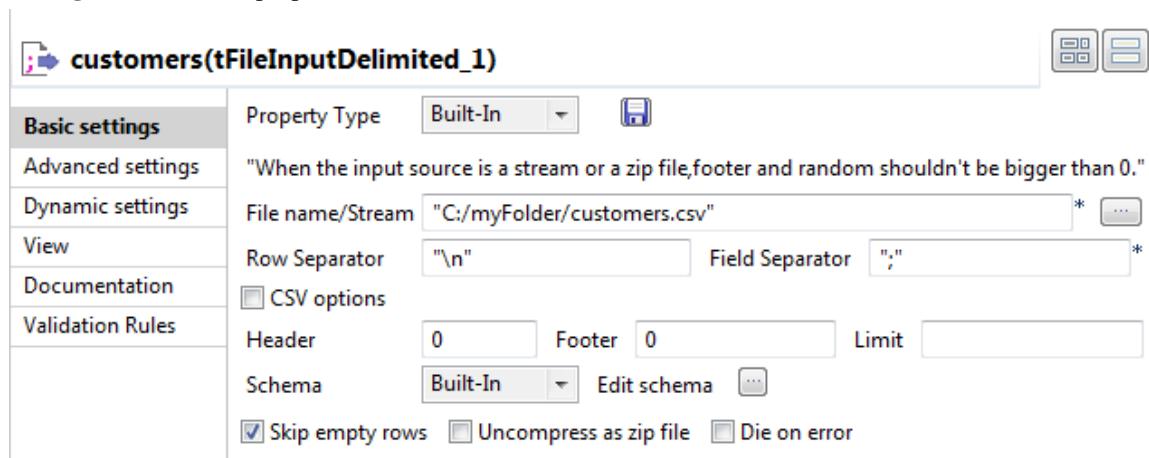
A complete Job looks as what it displays in the following image. For the detailed instruction for designing the Job, read the following sections.



### C.2.3.1. Step 1: Reading input data from a local file

We will use the **tFileInputDelimited** component to read the file *customers.csv* for the input data. This component can be found in the **File/Input** group of the **Palette**.

- Drop a **tFileInputDelimited** component onto the design workspace, and double-click the to open the **Basic settings** view to set its properties.



- Click the three-dot button next to the **File name/Stream** field to browse to the path of the input data file. You can also type in the path of the input data file manually.
- Click **Edit schema** to open a dialog box to configure the file structure of the input file.
- Click **OK** to close the dialog box.

Column	Key	Type		N..	Date Patt...	Length	Pre...	D...	Co...
id	<input type="checkbox"/>	Integer	<input checked="" type="checkbox"/>						
CustomerName	<input type="checkbox"/>	String	<input checked="" type="checkbox"/>						
CustomerAge	<input type="checkbox"/>	Integer	<input checked="" type="checkbox"/>						
CustomerAddress	<input type="checkbox"/>	String	<input checked="" type="checkbox"/>						
CustomerCity	<input type="checkbox"/>	String	<input checked="" type="checkbox"/>						
RegisterTime	<input type="checkbox"/>	Date	<input checked="" type="checkbox"/>	"dd-MM..."					

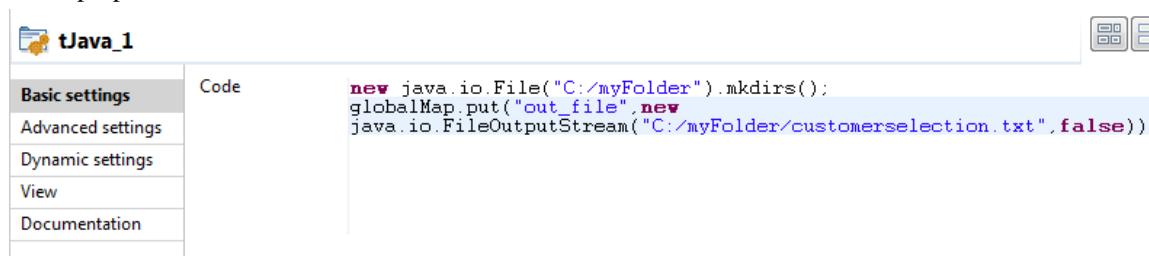
- Click **OK** to close the dialog box.

### C.2.3.2. Step2: Setting the command to enable the output stream feature

Now we will make use of **tJava** to set the command for creating an output file and a directory that contains the output file.

To do so:

- Drop a **tJava** component onto the design workspace, and double-click it to open the **Basic settings** view to set its properties.



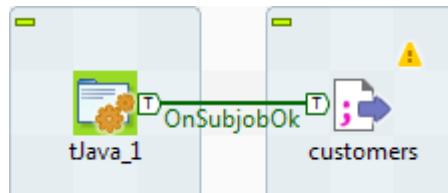
- Fill in the **Code** area with the following command:

```
new java.io.File("C:/myFolder").mkdirs();
globalMap.put("out_file",new java.io.FileOutputStream("C:/myFolder/
customerselection.txt",false));
```



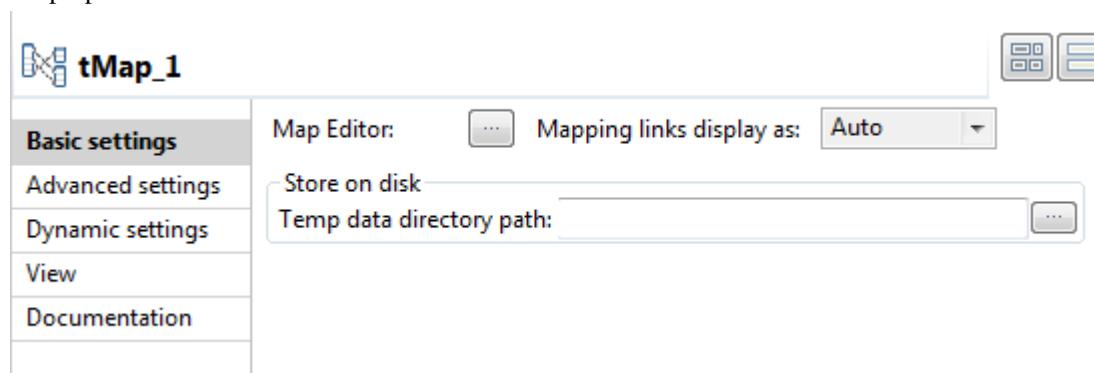
The command we typed in this step will create a new directory *C:/myFolder* for saving the output file *customerselection.txt* which is defined followingly. You can customize the command in accordance with actual practice.

- Connect **tJava** to **tFileInputDelimited** using a **Trigger > On Subjob Ok** connection. This will trigger **tJava** when subjob that starts with **tFileInputDelimited** succeeds in running.



### C.2.3.3. Step3: Mapping the data using the **tMap** component

- Drop a **tMap** component onto the design workspace, and double-click it to open the **Basic settings** view to set its properties.



- Click the three-dot button next to **Map Editor** to open a dialog box to set the mapping.
- Click the plus button on the left to add six columns for the schema of the incoming data, these columns should be the same as the following:

The screenshot shows the Schema editor interface. At the top, there's a toolbar with icons for saving, opening, and closing. Below it is a header bar with the title "row1" and a green plus sign icon. The main area is divided into two panes. The left pane is labeled "Column" and lists the following columns:

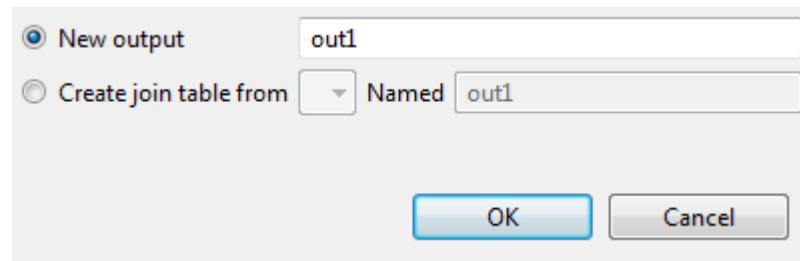
- id**
- CustomerName**
- CustomerAge**
- CustomerAddress**
- CustomerCity**
- RegisterTime**

The right pane is titled "Schema editor" and contains a table with the following schema details:

Column	Key	Type	N.	Date Pattern (Ct...)
<b>id</b>	<input type="checkbox"/>	Integer	<input checked="" type="checkbox"/>	
<b>CustomerName</b>	<input type="checkbox"/>	String	<input checked="" type="checkbox"/>	
<b>CustomerAge</b>	<input type="checkbox"/>	Integer	<input checked="" type="checkbox"/>	
<b>CustomerAddress</b>	<input type="checkbox"/>	String	<input checked="" type="checkbox"/>	
<b>CustomerCity</b>	<input type="checkbox"/>	String	<input checked="" type="checkbox"/>	
<b>RegisterTime</b>	<input type="checkbox"/>	Date	<input checked="" type="checkbox"/>	"dd-MM-yyyy"

At the bottom of the right pane are several icons for managing the schema.

- Click the plus button on the right to add a schema of the outgoing data flow.



- Select **New output** and Click **OK** to save the output schema. For the time being, the output schema is still empty.
- Click the plus button beneath the *out1* table to add three columns for the output data.

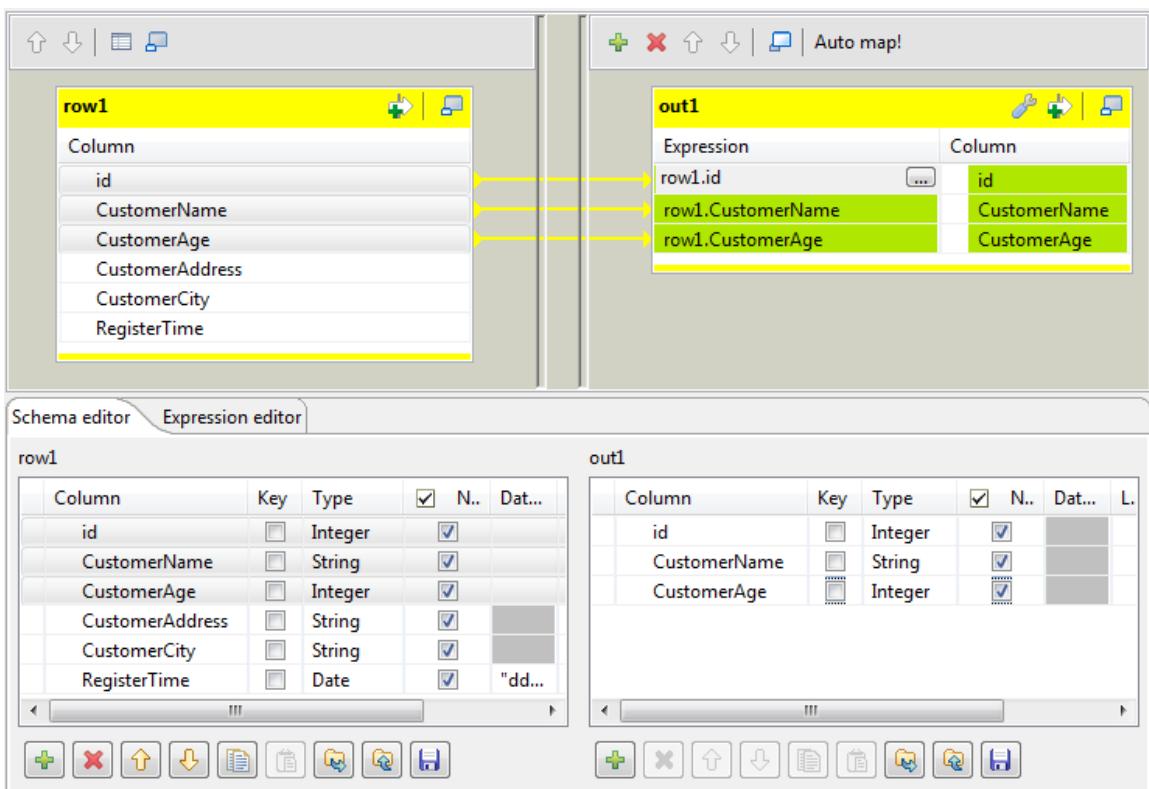
The screenshot shows the Schema editor interface for the "out1" schema. The top pane lists the columns:

- newColumn**
- newColumn1**
- newColumn2**

The bottom pane shows the schema details:

Expression	Column
	<b>newColumn</b>
	<b>newColumn1</b>
	<b>newColumn2</b>

- Drop the **id**, **CustomerName** and **CustomerAge** columns onto their respective line on the right.



- Click **OK** to save the settings.

#### C.2.3.4. Step4: Outputting the selected data stream

- Drop a **tFileOutputDelimited** component onto the design workspace, and double-click it to open the **Basic settings** view to set its component properties.
- Select the **Use Output Stream** check box to enable the **Output Stream** field and fill the **Output Stream** field with the following command:

```
(java.io.OutputStream)globalMap.get("out_file")
```



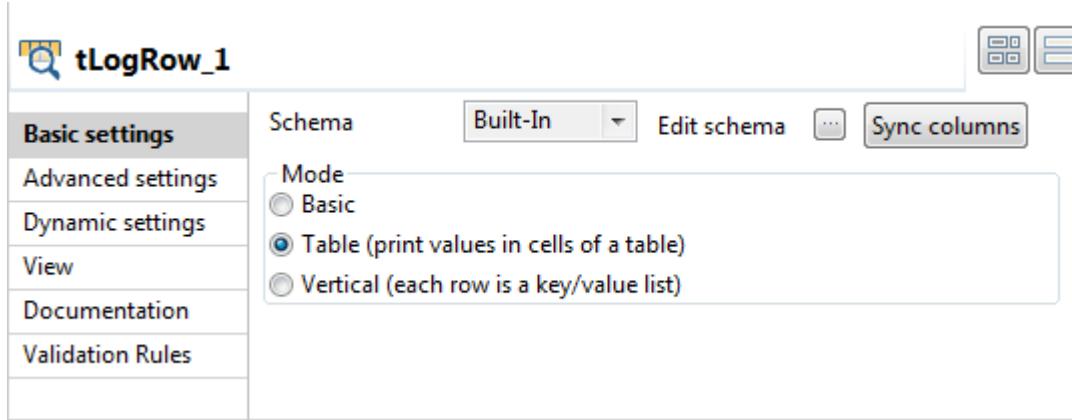
You can customize the command in the **Output Stream** field by pressing **CTRL+SPACE** to select built-in command from the list or type in the command manually in accordance with actual practice. In this scenario, the command we use in the **Output Stream** field will call the `java.io.OutputStream` class to output the filtered data stream to a local file which is defined in the **Code** area of **tJava** in this scenario.

<b>Basic settings</b>	Property Type	Built-In	
<input checked="" type="checkbox"/> Use Output Stream			
Output Stream	(java.io.OutputStream)globalMap.get("out_file")		
Row Separator	\n	Field Separator	;
<input checked="" type="checkbox"/> Include Header	<input type="checkbox"/> Compress as zip file		
Schema	Built-In	Edit schema	Sync columns

3. Connect **tFileInputDelimited** to **tMap** using a **Row > Main** connection and connect **tMap** to **tFileOutputDelimited** using a **Row > out1** connection which is defined in the **Map Editor** of **tMap**.
4. Click **Sync columns** to retrieve the schema defined in the preceding component.

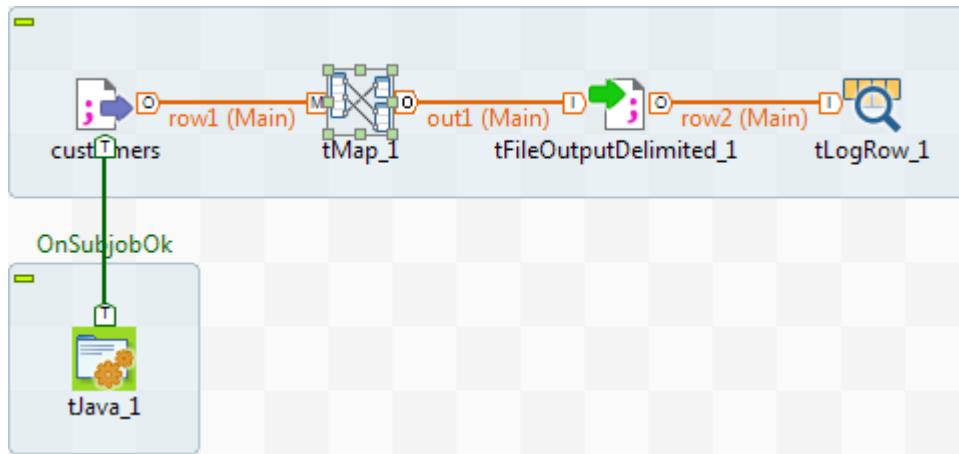
To output the selected data to the console:

1. Drop a **tLogRow** component onto the design workspace, and double-click it to open its **Basic settings** view.
2. Select the **Table** radio button in the **Mode** area.



3. Connect **tFileOutputDelimited** to **tLogRow** using a **Row > Main** connection.
4. Click **Sync columns** to retrieve the schema defined in the preceding component.

This Job is now ready to be executed.



5. Press **CTRL+S** to save your Job and press **F6** to execute it.

The content of the selected data is displayed on the console.

```

Starting job OutputStream at 17:31 19/10/2011.

[statistics] connecting to socket on port 4059
[statistics] connected
+-----+
| tLogRow_1 |
+-----+
| id | CustomerName | CustomerAge |
+-----+
| 10001 | Warren | 67 |
| 10002 | Woodrow | 68 |
| 10003 | Grover | 77 |
| 10004 | Abraham | 74 |
| 10005 | Chester | 78 |
| 10006 | Calvin | 63 |
| 10007 | Zachary | 53 |
| 10008 | Chester | 36 |
| 10009 | Chester | 60 |
| 10010 | Woodrow | 57 |
+-----+
[statistics] disconnected
Job OutputStream ended at 17:31 19/10/2011. [exit code=0]

```

The selected data is also output to the specified local file *customerselection.txt*.

customerselection.txt		
1	id;CustomerName;CustomerAge	
2	10001;Warren;67	
3	10002;Woodrow;68	
4	10003;Grover;77	
5	10004;Abraham;74	
6	10005;Chester;78	
7	10006;Calvin;63	
8	10007;Zachary;53	
9	10008;Chester;36	
10	10009;Chester;60	
11	10010;Woodrow;57	

For an example of Job using this feature, see Scenario: *Utilizing Output Stream in saving filtered data to a local file* of **tFileOutputDelimited** at <https://help.talend.com>.

For the principle of the **Use Output Stream** feature, see [How to use the Use Output Stream feature](#).

## C.3. Using the Implicit Context Load feature

Job parameterization based on context variables enables you to orchestrate and execute your Jobs in different contexts or environments. You can define the values of your context variables when creating them, or load your context parameters dynamically, either explicitly or implicitly, when your Jobs are executed.

The use case below describes how to use the Implicit Context Load feature of your *Talend Studio* to load context parameters dynamically at the time of Job execution. For how to load context parameters explicitly at the time of Job execution, see **tContextLoad** at <https://help.talend.com>. For more information on using contexts and variables, see [Using contexts and variables](#).

The Job in this use case is composed of only two components. It will read employees data stored in two MySQL databases, one for testing and the other for production purposes. The connection parameters for accessing these two databases are stored in another MySQL database. When executed, the Job loads these connection parameters dynamically to connect to the two databases.

### C.3.1. Preparing context parameter sources

Create two tables named *db\_testing* and *db\_production* respectively in a MySQL database named *db\_connections*, to hold the connection parameters for accessing the above mentioned databases, *testing* and *production*. Each table should contain only two columns: *key* and *value*, both of type VARCHAR. Below is an example of the content of the database tables:

*db\_testing*:

key	value
host	localhost
port	3306
username	root
password	talend
database	testing

*db\_production*:

key	value
host	localhost
port	3306
username	root
password	talend
database	production

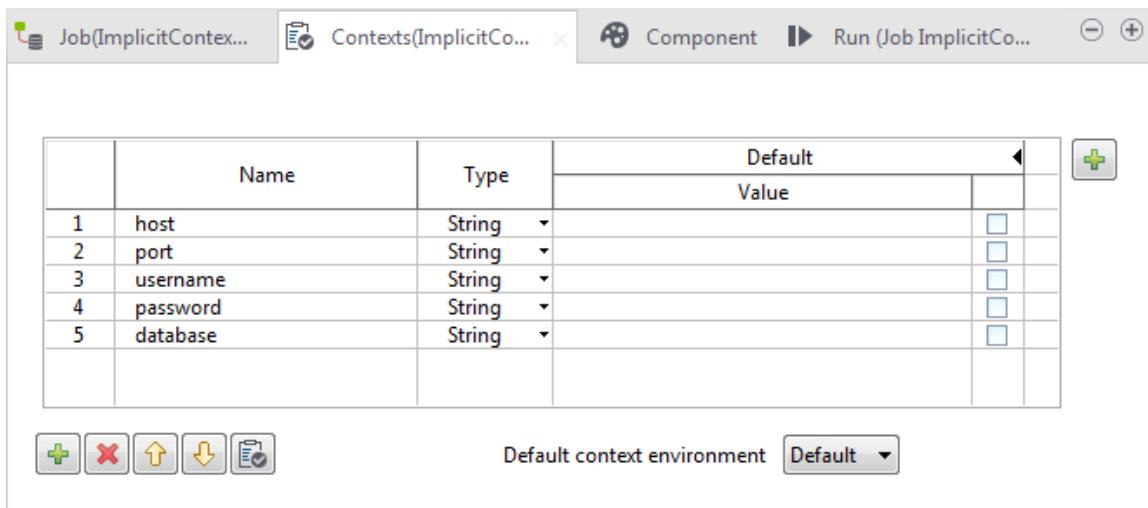
You can create these database tables using another **Talend** Job that contains **tFixedFlowInput** and **tMysqlOutput** components.

### C.3.2. Creating the Job and defining context variables

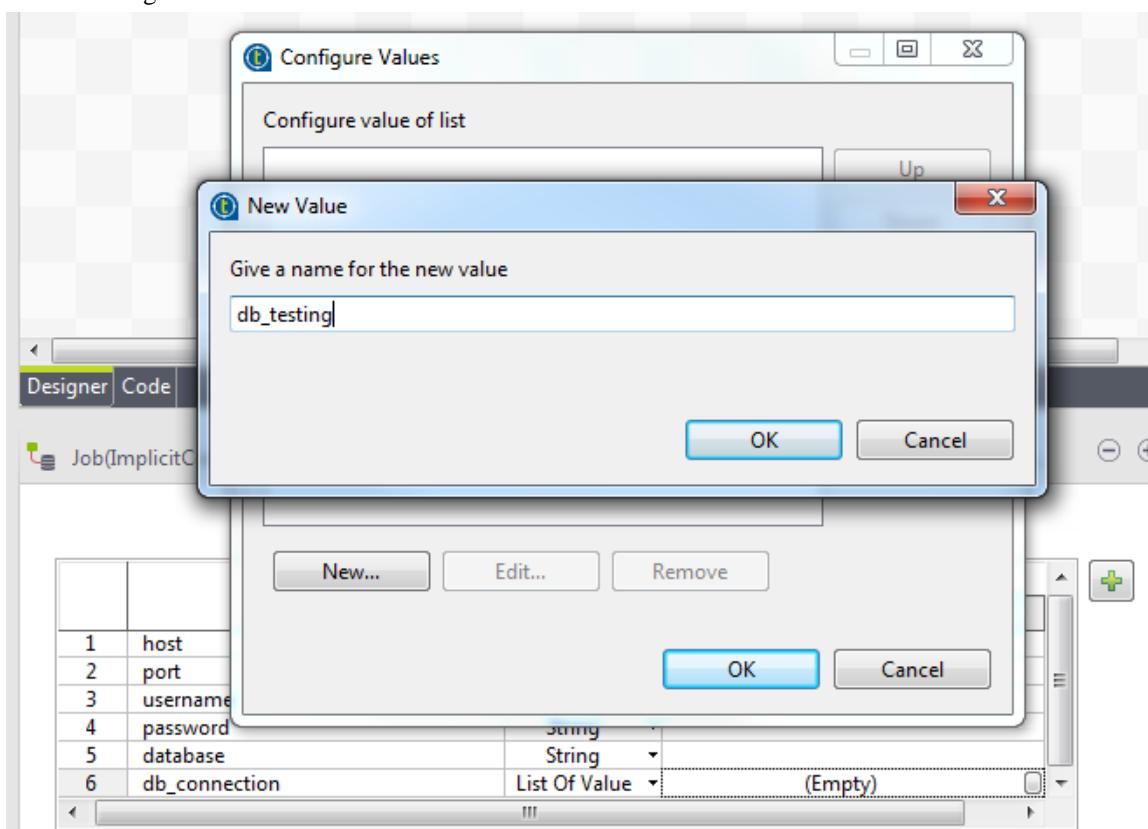
1. Create a Job and add a **tMysqlInput** component and a **tLogRow** component onto the design workspace, and link them using a **Row > Main** connection.



2. Select the **Contexts** view of the Job, and click the **[+]** button at the bottom of the view to add five rows in the table to define the following variables, all of type **String**, without defining their values, which will be loaded dynamically at Job execution: *host*, *port*, *username*, *password*, and *database*.



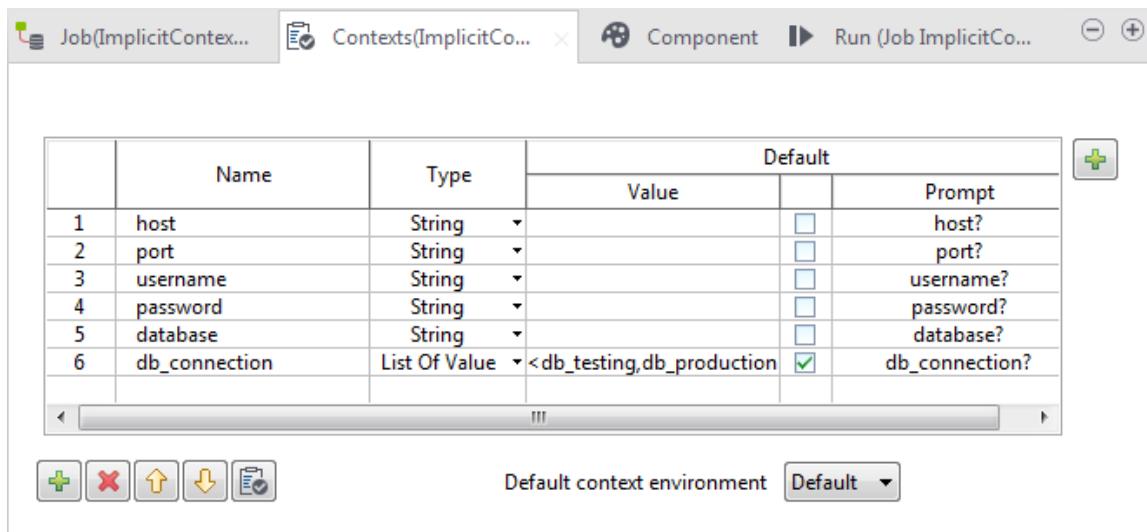
3. Now create another variable named *db\_connection* of type **List Of Value**.
4. Click in the **Value** field of the newly created variable and click the button that appears to open the **Configure Values** dialog box, and click **New...** to open the **New Value** dialog box. Enter the name of one of the database tables holding the database connection details and click **OK**.



5. Click **New...** again to define the other table holding the database connection details. When done, click **OK** to close the **Configure Values** dialog box.

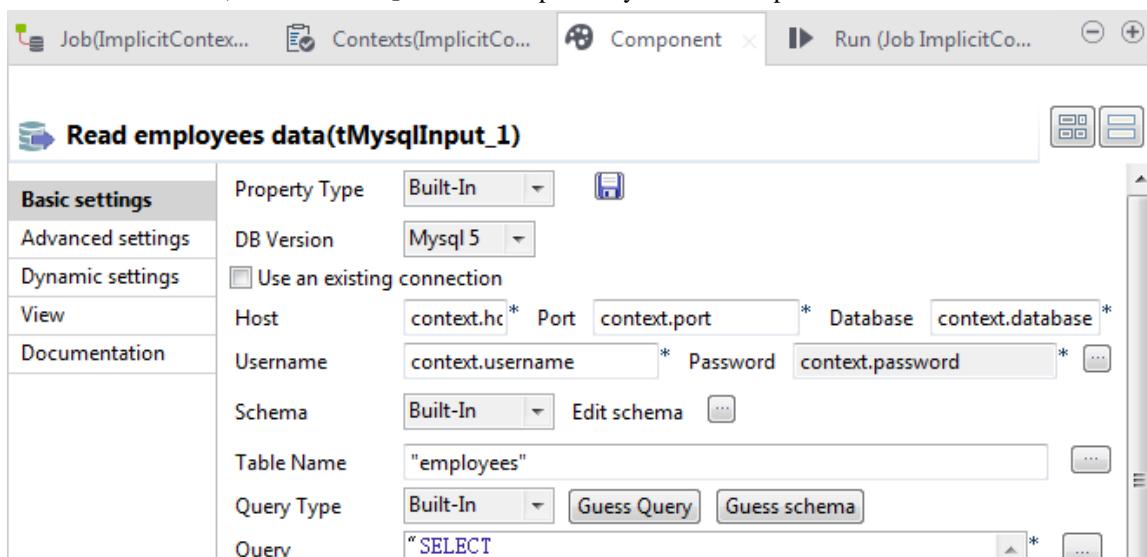
Now the variable *be\_connection* has a list of values *db\_testing* and *db\_production*, which are the database tables to load the connection parameters from.

6. Select the **Prompt** check box next to the **Value** field of the *db\_connection* variable to show the **Prompt** fields and enter the prompt message to be displayed at the execution time.



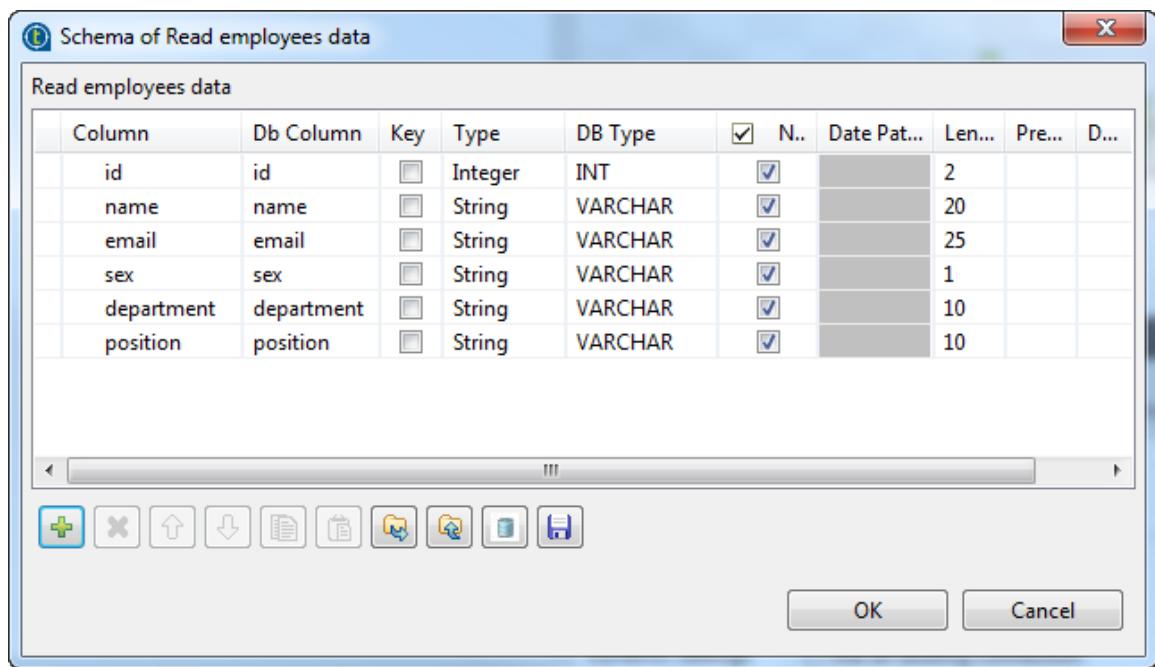
### C.3.3. Configuring the components

1. Then double-click to open the **tMysqlInput** component **Basic settings** view.
2. Fill the **Host**, **Port**, **Database**, **Username**, **Password**, and **Table Name** fields with the relevant variables defined in the **Contexts** tab view: context.host, context.port, context.database, context.username, and context.password respectively in this example.



3. Fill the **Table Name** field with *employees*, which is the name of the table holding employees information in both databases in our example.
4. Then fill in the **Schema** information. If you stored the schema in the **Repository**, then you can retrieve it by selecting **Repository** and the relevant entry in the list.

In this example, the schema of both the database tables to read is made of six columns: *id* (INT, 2 characters long), *name* (VARCHAR, 20 characters long), *email* (VARCHAR, 25 characters long), *sex* (VARCHAR, 1 characters long), *department* (VARCHAR, 10 characters long), and *position* (VARCHAR, 10 characters long).



5. Click **Guess Query** to retrieve all the table columns, which will be displayed on the **Run** tab, through the **tLogRow** component.
6. In the **Basic settings** view of the **tLogRow** component, select the **Table** option to display data records in the form of a table.

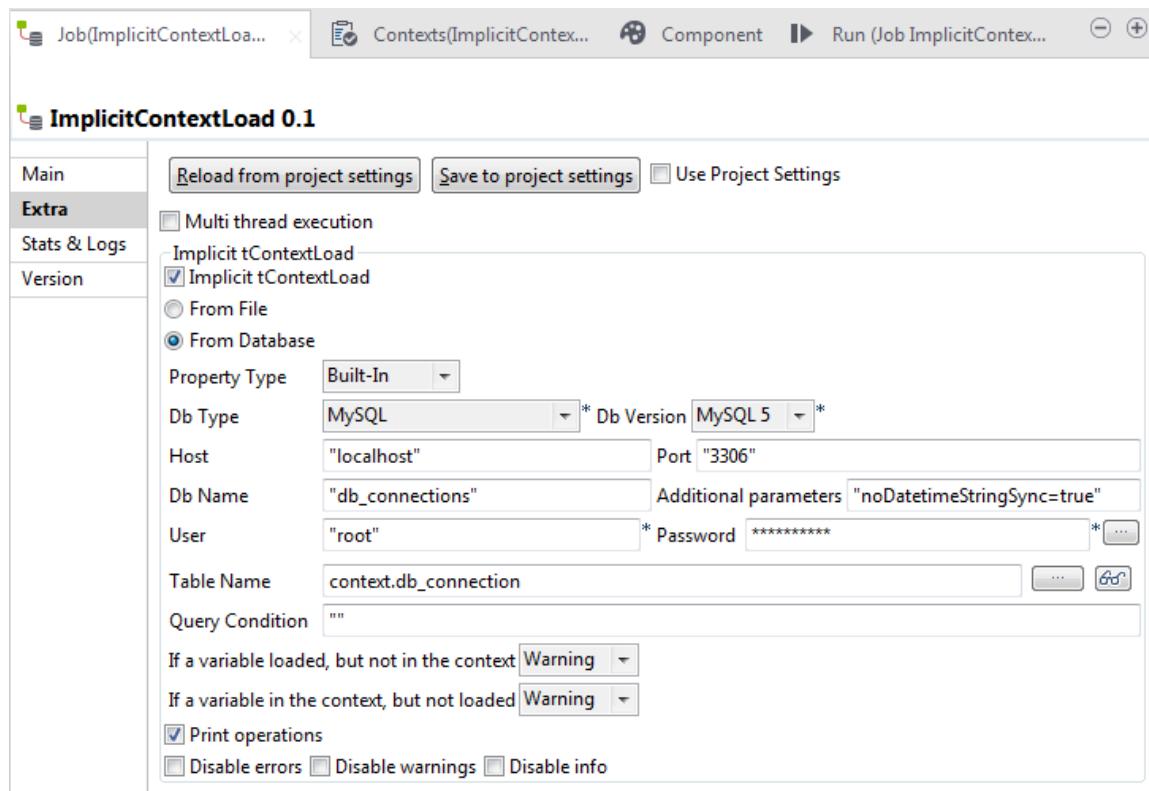
### C.3.4. Configuring the Implicit Context Load feature

You can configure the Implicit Context Load feature either in Project Settings so that it can be used across Jobs within the project, or in the Job view for a particular Job.

The following example shows how to configure the Implicit Context Load feature in the **Job** view for this particular Job. If you want to configure the feature to be reused across different Jobs, select **File > Edit Project properties** from the menu bar to open the **Project Settings** dialog box, go to **Job Settings > Implicit context load**, select the **Implicit tContextLoad** check box, and set the parameters following steps 2 through 6 below. Then in the **Job** view, select the **Use Project Settings** check box to apply the settings to the Job.

1. From the **Job** view, select the **Extra** vertical tab, and select the **Implicit tContextLoad** check box to enable context loading without using the **tContextLoad** component explicitly in the Job.
2. Select the source to load context parameters from. A context source can be a two-column flat file or a two-column database table. In this use case the database connection details are stored in database tables, so select the **From Database** option.
3. Define the database connection details just like defining the basic settings of a database input component.

In this example, all the connection parameters are used just for this particular Job, so select **Built-In** from the **Property Type** list and fill in the connection details manually.

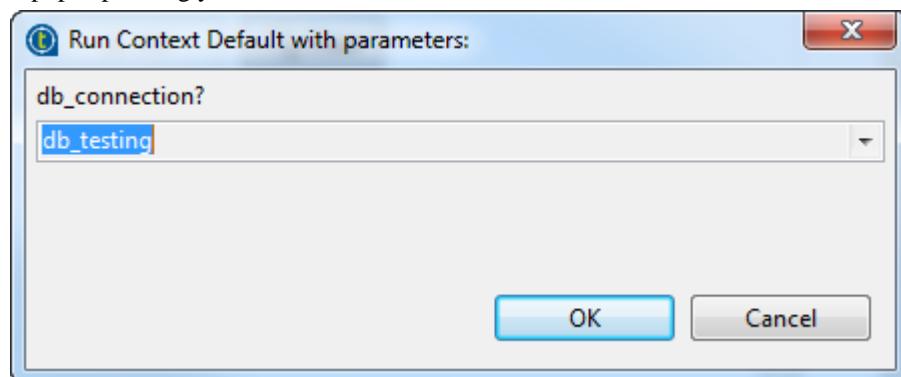


4. Fill the **Table Name** field with the context variable named *db\_connection* defined in the **Contexts** view of the Job so that we will be able to choose the database table to load context parameters from dynamically at Job execution.
5. As we will fetch all the connection details from the database tables unconditionally, leave the **Query Condition** field blank.
6. Select the **Print operations** check box to list the context parameters loaded at Job execution.

## C.3.5. Executing the Job

1. Press **Ctrl+S** to save the Job, and press **F6** to run the Job.

A dialog box pops up asking you to select a database. Select a database and click **OK**.



The loaded context parameters and the content of the *employees* table of the selected database are displayed on the **Run** console.

Execution

```
Starting job ImplicitContextLoad at 17:18 25/08/2015.

[statistics] connecting to socket on port 3969
[statistics] connected
Implicit_Context_Context set key "host" with value "localhost"
Implicit_Context_Context set key "port" with value "3306"
Implicit_Context_Context set key "username" with value "root"
Implicit_Context_Context set key "password" with value "talend"
Implicit_Context_Context set key "database" with value "testing"
Warning: Parameter "db_connection" has not been set by Implicit_Context_Context

+-----+
| Display employees data |
+-----+
| id | name | email | sex | department | position |
+-----+
| 1 | Elisa | elisa@company.com | F | R&D | Manager |
| 2 | Nicolas | nicolas@company.com | M | R&D | Developer |
| 3 | Cedric | cedric@company.com | M | null | null |
| 4 | Rabbit | rabbit@comaphny.com | M | null | null |
| 5 | Mike | mike@company.com | M | null | null |
| 6 | Sabrina | sabrina@company.com | F | Community | Developer |
| 7 | Stephane | stephane@company.com | M | Sales | Manager |
| 8 | Jim | jim@company.com | M | Sales | Pre-sales |
| 9 | John | john@company.com | M | null | null |
+-----+

[statistics] disconnected
Job ImplicitContextLoad ended at 17:18 25/08/2015. [exit code=0]
```

Line limit 100  Wrap

- Now press **F6** to launch the Job again and select the other database when prompted.

The loaded context parameters and the content of the *employees* table of the other database are displayed on the **Run** console.

Execution

```
Starting job ImplicitContextLoad at 17:20 25/08/2015.

[statistics] connecting to socket on port 3862
[statistics] connected
Implicit_Context_Context set key "host" with value "localhost"
Implicit_Context_Context set key "port" with value "3306"
Implicit_Context_Context set key "username" with value "root"
Implicit_Context_Context set key "password" with value "talend"
Implicit_Context_Context set key "database" with value "production"
Warning: Parameter "db_connection" has not been set by Implicit_Context_Context

+-----+
| Display employees data |
+-----+
| id | name | email | sex | department | position |
+-----+
| 1 | Herbert Pierce | hp@talend.com | M | R&D | Manager |
| 2 | John Hoover | jh@talend.com | M | Finance | Manager |
| 3 | Benjamin Harrison | bh@talend.com | M | HR | Manager |
| 4 | George Harrison | gh@talend.com | M | Sales | Manager |
| 5 | Hellen Monroe | hm@talend.com | F | R&D | Developer |
| 6 | Anne Harrison | ah@talend.com | F | Sales | Pre-sales |
| 7 | Thomas Nixon | tn@talend.com | M | R&D | Developer |
| 8 | James Lincoln | jl@talend.com | M | R&D | Developer |
| 9 | Rutherford Fillmore | rf@talend.com | M | Finance | Accountant |
| 10 | Maria Pierce | mp@talend.com | F | Finance | Accountant |
+-----+

[statistics] disconnected
Job ImplicitContextLoad ended at 17:20 25/08/2015. [exit code=0]
```

Line limit 100  Wrap

Related topics:

- [How to define options on the Job view](#)
- [Job Settings](#)
- [Context settings](#)
- [Applying Project Settings](#)

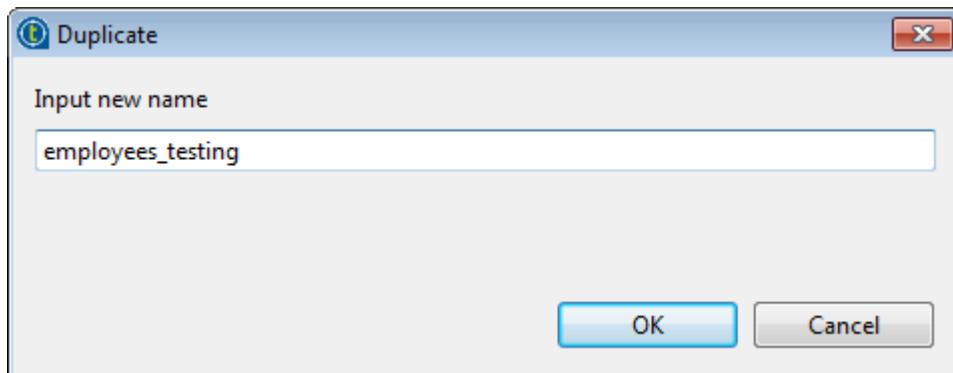
## C.4. Using the Multi-thread Execution feature to run Jobs in parallel

Based on the previous use case [Using the Implicit Context Load feature](#), this use case give an example of how to use the Multi-thread Execution feature to run two Jobs in parallel to display the employees information in both the testing and production environments at the same time. When handling large data volumes, this feature can significantly optimize the Job execution performance of the *Talend Studio*.

For more information on the multi-thread execution feature, see [How to execute multiple Subjobs in parallel](#).

### Preparing Jobs to read employees data in different contexts

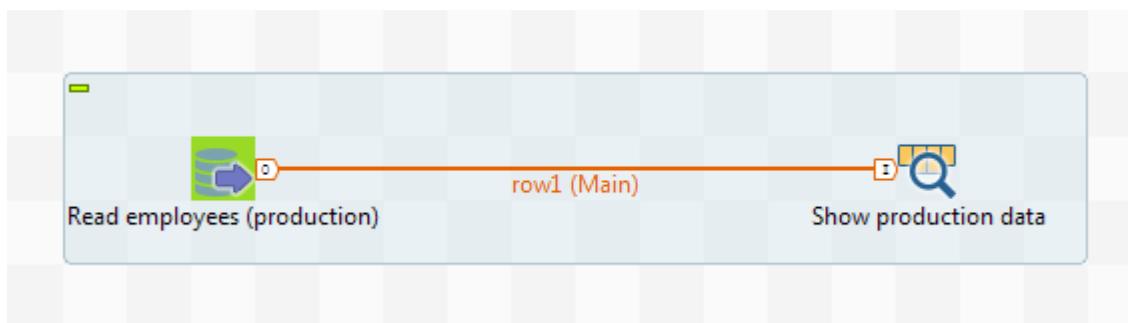
1. In the **Repository** tree view, right-click the Job created in the use case [Using the Implicit Context Load feature](#) and select **Duplicate** from the context menu. Then, in the **[Duplicate]** dialog box enter a new name for the Job, *employees\_testing* in this example, and click **OK**.



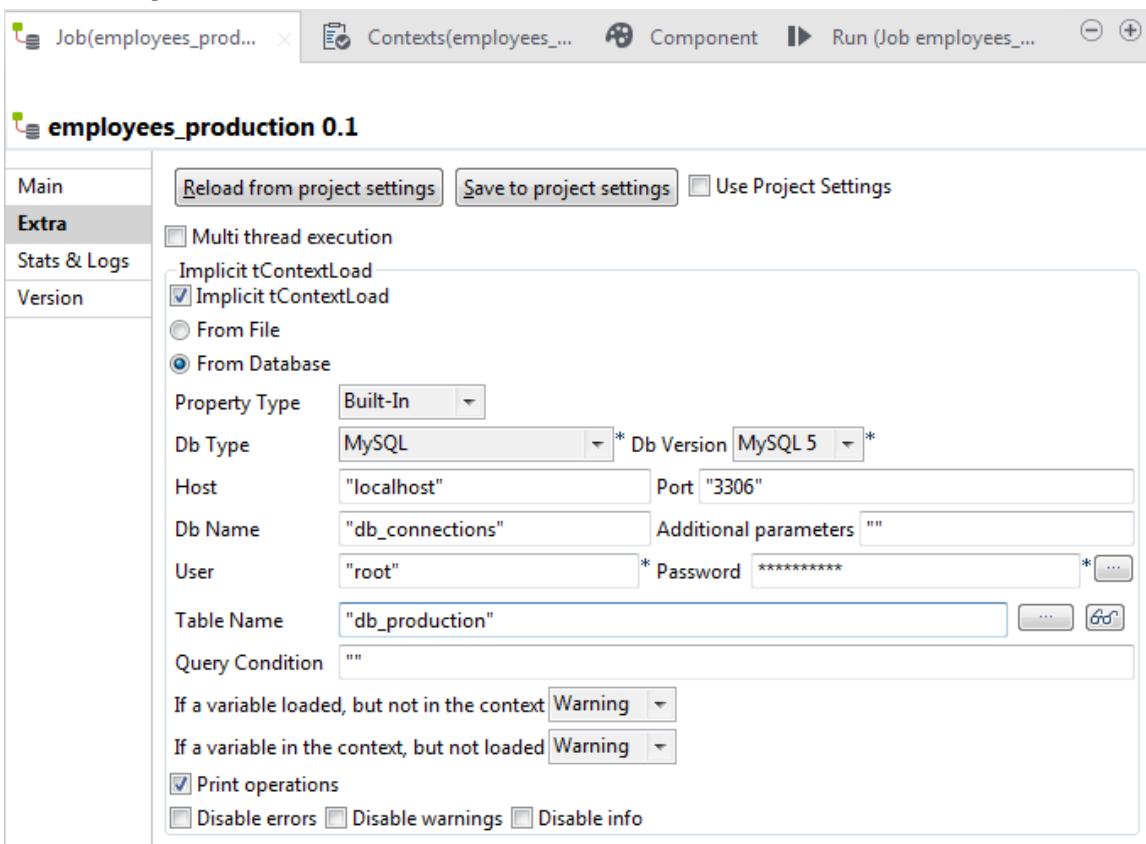
2. Open the new Job, and label the components to better reflect their roles.



3. Create another Job named *employees\_production* by repeating the steps above.

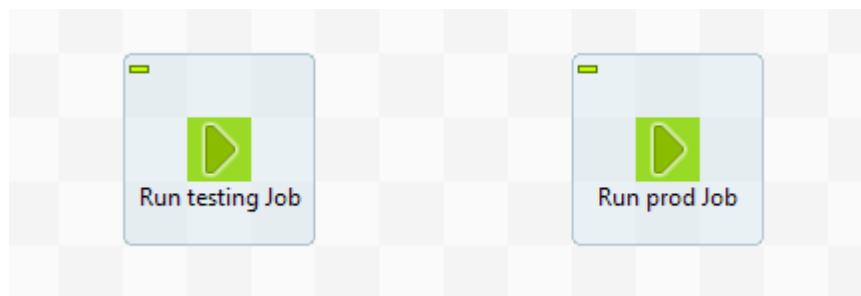


4. In the **Contexts** view of both Jobs, remove the **db\_connection** variable.
5. On **Extra** tab of the **Job** view of the Job *employees\_testing*, fill the **Table Name** field of database settings with *db\_testing*; on the **Extra** tab of the **Job** view of the Job *employees\_production*, fill the **Table Name** field with *db\_production*.



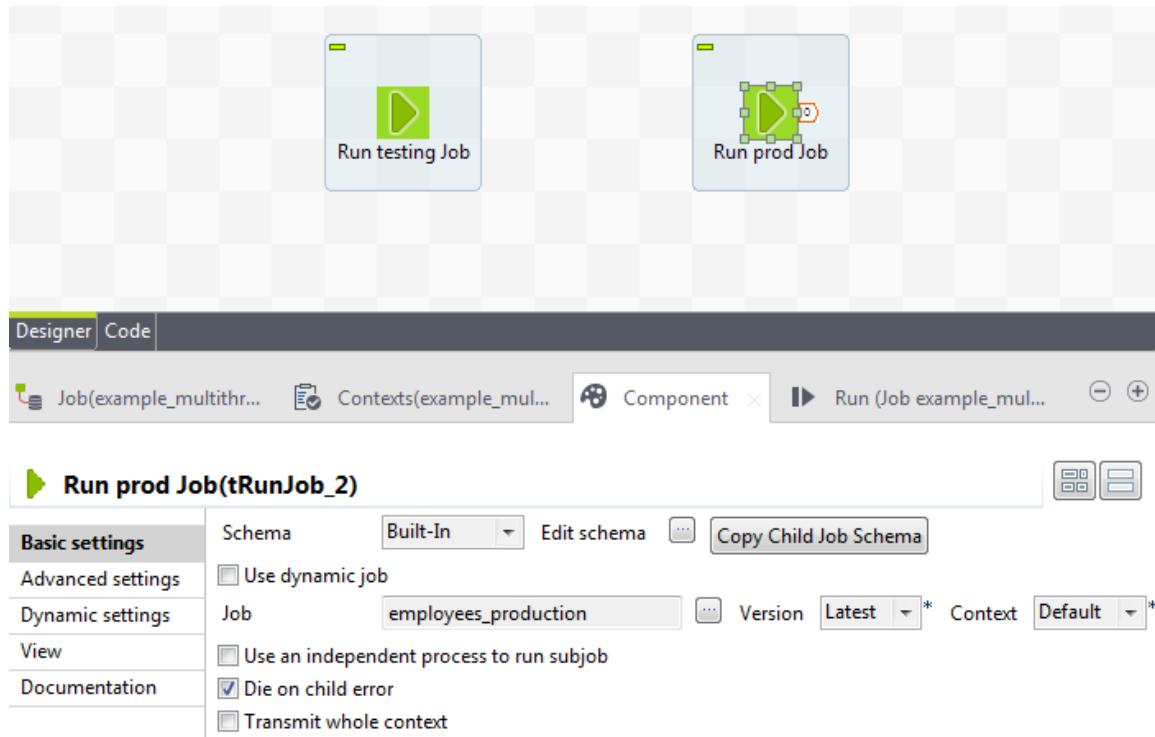
## Set up a parent Job to run the Jobs in parallel

1. Create a new Job and add two **tRunJob** components on the design workspace, and label the components to better reflect their roles.



2. In the **Component** view of the first **tRunJob** component, click the [...] button next to the **Job** field and specify the Job it will run, *employees\_testing* in this example.

Configure the other **tRunJob** component to run the other Job, *employees\_production*.



3. On the **Extra** tab of the **Job** view, select the **Multi thread execution** check box to activate the Multi-thread Execution feature.

## Executing the Jobs

1. Save each Job by pressing **Ctrl+S**.
2. In the parent Job, press **F6** or click **Run** on the **Run** view to start execution of the child Jobs.

The child Jobs are executed in parallel, reading employees data from both databases and displaying the data on the console.

Execution

IMPLICIT_CONTEXT_CONTEXT					
Show testing data					
=	=	=	=	=	=
id	name	email	sex	department	position
=	=	=	=	=	=
1	Elisa	elisa@company.com	F	R&D	Manager
2	Nicolas	nicolas@company.com	M	R&D	Developer
3	Cedric	cedric@company.com	M	null	null
4	Rabbit	rabbit@comapny.com	M	null	null
5	Mike	mike@company.com	M	null	null
6	Sabrina	sabrina@company.com	F	Community	Developer
7	Stephanie	stephane@company.com	M	Sales	Manager
8	Jim	jim@company.com	M	Sales	Pre-sales
9	John	john@company.com	M	null	null

Show production data					
=	=	=	=	=	=
id	name	email	sex	department	position
=	=	=	=	=	=
1	Herbert Pierce	hp@talend.com	M	R&D	Manager
2	John Hoover	jh@talend.com	M	Finance	Manager
3	Benjamin Harrison	bh@talend.com	M	HR	Manager
4	George Harrison	gh@talend.com	M	Sales	Manager
5	Hellen Monroe	hm@talend.com	F	R&D	Developer
6	Anne Harrison	ah@talend.com	F	Sales	Pre-sales
7	Thomas Nixon	tn@talend.com	M	R&D	Developer
8	James Lincoln	jl@talend.com	M	R&D	Developer
9	Rutherford Fillmore	rf@talend.com	M	Finance	Accountant
10	Maria Pierce	mp@talend.com	F	Finance	Accountant

Line limit 100  Wrap





## Appendix D. System routines

This appendix gives you an overview of the most commonly used routines, along with use cases. In this Appendix, routines follow the order in which they display in the **Repository**. They are grouped according to their types. Each type is detailed in a different section.

For more information on how to define routines, to access to system routines or to manage system or user routines, see [Managing routines](#).

Before starting any data integration processes, you need to be familiar with *Talend Studio* Graphical User Interface (GUI). For more information, see [GUI](#).

## D.1. Numeric Routines

Numeric routines allow you to return whole or decimal numbers in order to use them as settings in one or more Job components. To add numeric IDs, for instance.

To access these routines, double click on the **Numeric** category, in the **system** folder. The Numeric category contains several routines, notably sequence, random and decimal (convertImpliedDecimalFormat):

Routine	Description	Syntax
<i>sequence</i>	Returns an incremental numeric ID.	<code>Numeric.sequence("Parameter name", start value, increment value)</code>
<i>resetSequence</i>	Creates a sequence if it doesn't exist and attributes a new start value.	<code>Numeric.resetSequence (Sequence Identifier, start value)</code>
<i>removeSequence</i>	Removes a sequence.	<code>Numeric.RemoveSequence (Sequence Identifier)</code>
<i>random</i>	Returns a random whole number between the maximum and minimum values.	<code>Numeric.random(minimum start value, maximum end value)</code>
<i>convertImplied DecimalFormat</i>	Returns a decimal with the help of an implicit decimal model.	<code>Numeric.convertImpliedDecimalFormat ("Target Format", value to be converted)</code>

The three routines *sequence*, *resetSequence*, and *removeSequence* are closely related.

- The *sequence* routine is used to create a sequence identifier, named *s1* by default, in the Job. This sequence identifier is global in the Job.
- The *resetSequence* routine can be used to initialize the value of the sequence identifier created by *sequence* routine.
- The *removeSequence* routine is used to remove the sequence identifier from the global variable list in the Job.

### D.1.1. How to create a Sequence

The **sequence** routine allows you to create automatically incremented IDs, using a **tJava** component:

```
System.out.println(Numeric.sequence("s1",1,1));
System.out.println(Numeric.sequence("s1",1,1));
```

The routine generates and increments the ID automatically:

```
[statistics] connecting to socket on port 3360
[statistics] connected
1
2
```

### D.1.2. How to convert an Implied Decimal

It is easy to use the **convertImpliedDecimalFormat** routine, along with a **tJava** component, for example:

```
System.out.println(Numeric.convertImpliedDecimalFormat("9V99", "123"));
```

The routine automatically converts the value entered as a parameter according to the format of the implied decimal provided:

```
1.23
[statistics] disconnected
Job test_routine ended at 14:12 04/02/2010. [exit code=0]
```

## D.2. Relational Routines

Relational routines allow you to check affirmations based on booleans.

To access these routines, double-click **Relational** under the **system** folder. The **Relational** class contains several routines, notably:

Routine	Description	Syntax
<i>ISNULL</i>	Checks if the variable provided is a null value.  It returns <i>true</i> if the value is NULL and <i>false</i> if the value is not NULL.	<code>Relational.ISNULL(variable)</code>
<i>NOT</i>	Returns the complement of the logical value of an expression.	<code>Relational.NOT(expression)</code>
<i>isNull</i>	Checks if the variable provided is a null value.  It returns <i>1</i> if the value is NULL and <i>0</i> if the value is not NULL.	<code>Relational.isNull(variable)</code>

To check a Relational Routine, you can use the **ISNULL** routine, along with a **tJava** component, for example:

```
String str = null;
System.out.println(Relational.ISNULL(str));
```

In this example, the test result is displayed in the **Run** view:

```
Starting job test_routine at 14:14 04/02/2010.
[statistics] connecting to socket on port 3375
[statistics] connected
true
[statistics] disconnected
Job test_routine ended at 14:14 04/02/2010. [exit code=0]
```

## D.3. StringHandling Routines

The **StringHandling** routines allow you to carry out various kinds of operations and tests on alphanumeric expressions, based on Java methods.

To access these routines, double-click **StringHandling** under the **system** folder. The **StringHandling** class includes the following routines:

Routine	Description	Syntax
<i>ALPHA</i>	Checks whether the expression is arranged in alphabetical order. Returns the true or false boolean accordingly.	<code>StringHandling.ALPHA("string to be checked")</code>
<i>IS_ALPHA</i>	Checks whether the expression contains alphabetical characters only, or otherwise. Returns the true or false boolean accordingly.	<code>StringHandling.IS_ALPHA("string to be checked")</code>
<i>CHANGE</i>	Replaces an element of a string with a defined replacement element and returns the new string.	<code>StringHandling.CHANGE("string to be checked", "string to be replaced", "replacement string")</code>
<i>COUNT</i>	Returns the number of times a substring occurs within a string.	<code>StringHandling.COUNT("string to be checked", "substring to be counted")</code>

Routine	Description	Syntax
<i>DOWNCASE</i>	Converts all uppercase letters in an expression into lowercase and returns the new string.	<code>StringHandling.DOWNCASE("string to be converted")</code>
<i>UPCASE</i>	Converts all lowercase letters in an expression into uppercase and returns the new string.	<code>StringHandling.UPCASE("string to be converted")</code>
<i>DQUOTE</i>	Encloses an expression in double quotation marks.	<code>StringHandling.DQUOTE("string to be enclosed in double quotation marks")</code>
<i>EREPLACE</i>	Substitutes all substrings that match the given regular expression in the given old string with the given replacement and returns a new string.	<code>StringHandling.EREPLACE(oldStr, regex, replacement)</code>
<i>INDEX</i>	Returns the position of the first character in a specified substring, within a whole string. If the substring specified does not exist in the whole string, the value - 1 is returned.	<code>StringHandling.INDEX("string to be checked", "substring specified")</code>
<i>LEFT</i>	Specifies a substring which corresponds to the first n characters in a string.	<code>StringHandling.LEFT("string to be checked", number of characters)</code>
<i>RIGHT</i>	Specifies a substring which corresponds to the last n characters in a string.	<code>StringHandling.RIGHT("string to be checked", number of characters)</code>
<i>LEN</i>	Calculates the length of a string.	<code>StringHandling.LEN("string to check")</code>
<i>SPACE</i>	Generates a string consisting of a specified number of blank spaces.	<code>StringHandling.SPACE(number of blank spaces to be generated)</code>
<i>SQUOTE</i>	Encloses an expression in single quotation marks.	<code>StringHandling.SQUOTE("string to be enclosed in single quotation marks")</code>
<i>STR</i>	Generates a particular character a the number of times specified.	<code>StringHandling.STR('character to be generated', number of times)</code>
<i>TRIM</i>	Deletes the spaces and tabs before the first non-blank character in a string and after the last non-blank character, then returns the new string.	<code>StringHandling.TRIM("string to be checked")</code>
<i>BTRIM</i>	Deletes all the spaces and tabs after the last non-blank character in a string and returns the new string.	<code>StringHandling.BTRIM("string to be checked")</code>
<i>FTRIM</i>	Deletes all the spaces and tabs preceding the first non-blank character in a string.	<code>StringHandling.FTRIM("string to be checked")</code>
<i>SUBSTR</i>	Returns a portion of a string. It counts all characters, including blanks, starting at the beginning of the string.	<p><code>StringHandling.SUBSTR(string, start, length)</code></p> <ul style="list-style-type: none"> <li>• <b>string:</b> the character string you want to search.</li> <li>• <b>start:</b> the position in the string where you want to start counting.</li> <li>• <b>length:</b> the number of characters you want to return.</li> </ul>
<i>LTRIM</i>	Removes blanks or characters from the beginning of a string.	<p><code>StringHandling.LTRIM(string[, trim_set])</code></p> <ul style="list-style-type: none"> <li>• <b>string:</b> the string you want to change.</li> <li>• <b>trim_set:</b> the characters you want to remove from the beginning of the string. <i>LTRIM</i> will compare the <i>trim_set</i> to the string character-by-character, starting with the left side of the string, and remove characters until it fails to find a matching character in the <i>trim_set</i>. If this parameter</li> </ul>

Routine	Description	Syntax
		is not specified, <i>LTRIM</i> will remove any blanks from the beginning of the string.
<i>RTRIM</i>	Removes blanks or characters from the end of a string.	StringHandling.RTRIM(string[, trim_set]) <ul style="list-style-type: none"> <li>• <b>string</b>: the string you want to change.</li> <li>• <b>trim_set</b>: the characters you want to remove from the ending of the string. <i>RTRIM</i> will compare the <i>trim_set</i> to the string character-by-character, starting with the right side of the string, and remove characters until it fails to find a matching character in the <i>trim_set</i>. If this parameter is not specified, <i>RTRIM</i> will remove any blanks from the ending of the string.</li> </ul>
<i>LPAD</i>	Converts a string to a specified length by adding blanks or characters to the beginning of the string.	StringHandling.LPAD(first_string, length[, second_string]) <ul style="list-style-type: none"> <li>• <b>first_string</b>: the string you want to change.</li> <li>• <b>length</b>: the length you want the string to be after being padded.</li> <li>• <b>second_string</b>: the characters you want to append to the left side of the <i>first_string</i>.</li> </ul>
<i>RPAD</i>	Converts a string to a specified length by adding blanks or characters to the end of the string.	StringHandling.RPAD(first_string, length[, second_string]) <ul style="list-style-type: none"> <li>• <b>first_string</b>: the string you want to change.</li> <li>• <b>length</b>: the length you want the string to be after being padded.</li> <li>• <b>second_string</b>: the characters you want to append to the right side of the <i>first_string</i>.</li> </ul>
<i>INSTR</i>	<p>Returns the position of a character set in a string, counting from left to right and starting from 1.</p> <p>Note that it returns 0 if the search is unsuccessful and <i>NULL</i> if the search value is <i>NULL</i>.</p>	StringHandling.INSTR(string, search_value, start, occurrence) <ul style="list-style-type: none"> <li>• <b>string</b>: the string you want to search.</li> <li>• <b>search_value</b>: the set of characters you want to search for.</li> <li>• <b>start</b>: the position in the string where you want to start the search. The default is 1, meaning it starts the search from the first character in the string.</li> <li>• <b>occurrence</b>: the occurrence you want to search for.</li> </ul> <p>For example, StringHandling.INSTR("Talend Technology", "e", 3, 2), it will start the search from the third character l and return 7, the position of the second character e.</p>
<i>TO_CHAR</i>	Converts numeric values to text strings.	StringHandling.TO_CHAR(numeric_value)

### D.3.1. How to store a string in alphabetical order

It is easy to use the ALPHA routine along with a **tJava** component, to check whether a string is in alphabetical order:

```
System.out.println(StringHandling.ALPHA("abcdefg"));
```

The check returns a boolean value.

```
Starting job test_routine at 14:29 04/02/2010.

[statistics] connecting to socket on port 3469

[statistics] connected

true
```

## D.3.2. How to check whether a string is alphabetical

It is easy to use the **IS\_ALPHA** routine along with a **tJava** component, to check whether the string is alphabetical:

```
System.out.printIn(StringHandling.IS_ALPHA("ab33cd"));
```

The check returns a boolean value.

```
Starting job routine1 at 11:45 23/02/2010.
[statistics] connecting to socket on port 3892
[statistics] connected
false
[statistics] disconnected
Job routine1 ended at 11:45 23/02/2010. [exit code=0]
```

## D.3.3. How to replace an element in a string

It is easy to use the **CHANGE** routine along with a **tJava** component, to replace one element in a string with another:

```
System.out.printIn(StringHandling.CHANGE("hello world!", "world", "guy"));
```

The routine replaces the old element with the new element specified.

```
|hello guy!
```

## D.3.4. How to check the position of a specific character or substring, within a string

The **INDEX** routine is easy to use along with a **tJava** component, to check whether a string contains a specified character or substring:

```
System.out.printIn(StringHandling.INDEX("hello world!", "hello"));
System.out.printIn(StringHandling.INDEX("hello world!", "world"));
System.out.printIn(StringHandling.INDEX("hello world", "!"));
System.out.printIn(StringHandling.INDEX("hello world", "?"));
```

The routine returns a whole number which indicates the position of the first character specified, or indeed the first character of the substring specified. Otherwise, -1 is returned if no occurrences are found.

```
Starting job routine1 at 15:47 24/02/2010.
[statistics] connecting to socket on port 4027
[statistics] connected
0
6
11
-1
[statistics] disconnected
Job routine1 ended at 15:47 24/02/2010. [exit code=0]
```

## D.3.5. How to calculate the length of a string

The **LEN** routine is easy to use, along with a **tJava** component, to check the length of a string:

```
System.out.printIn(StringHandling.LEN("hello world!"));
```

The check returns a whole number which indicates the length of the chain, including spaces and blank characters.

|12

## D.3.6. How to delete blank characters

The **FTRIM** routine is easy to use, along with a **tJava** component, to delete blank characters from the start of a chain:

```
System.out.println(StringHandling.FTRIM("Hello world "));
```

The routine returns the string with the blank characters removed from the beginning.

```
Starting job routine1 at 16:14 24/02/2010.
[statistics] connecting to socket on port 3790
[statistics] connected
Hello world!
[statistics] disconnected
Job routine1 ended at 16:14 24/02/2010. [exit code=0]
```

## D.4. TalendDataGenerator Routines

The **TalendDataGenerator** routines are functions which allow you to generate sets of test data. They are based on fictitious lists of first names, second names, addresses, towns and States provided by **Talend**. These routines are generally used when developing Jobs, using a **tRowGenerator**, for example, to avoid using production or company data.

To access the routines, double click on **TalendDataGenerator** under the **system** folder:

Routine	Description	Syntax
<code>getFirstName</code>	returns a first name taken randomly from a fictitious list.	<code>TalendDataGenerator.getFirstName()</code>
<code>getLastName</code>	returns a random surname from a fictitious list.	<code>TalendDataGenerator.getLastName()</code>
<code>getUsStreet</code>	returns an address taken randomly from a list of common American street names.	<code>TalendDataGenerator.getUsStreet()</code>
<code>getUsCity</code>	returns the name of a town taken randomly from a list of American towns.	<code>TalendDataGenerator.getUsCity()</code>
<code>getUsState</code>	returns the name of a State taken randomly from a list of American States.	<code>TalendDataGenerator.getUsState()</code>
<code>getUsStateId</code>	returns an ID randomly taken from a list of IDs attributed to American States.	<code>TalendDataGenerator.getUsStateId()</code>



No entry parameter is required as **Talend** provides the list of fictitious data.

You can customize the fictitious data by modifying the **TalendGeneratorRoutines**. For further information on how to customize routines, see [Customizing the system routines](#).

## D.4.1. How to generate fictitious data

It is easy to use the different functions to generate data randomly. Using a **tJava** component, you can, for example, create a list of fictitious client data using functions such as **getFirstName**, **getLastName**, **getUSCity**:

```
System.out.println(TalendDataGenerator.getFirstname());
```

```
System.out.printIn(TalendDateGenerator.getLastname());
System.out.printIn(TalendDateGenerator.getUsCity());
System.out.printIn(TalendDateGenerator.getUsState());
System.out.printIn(TalendDateGenerator.getUsStateId());
System.out.printIn(TalendDateGenerator.getUsStreet());
```

The set of data taken randomly from the list of fictitious data is displayed in the **Run** view:

```
Starting job test_routine at 14:44 04/02/2010.
[statistics] connecting to socket on port 3907
[statistics] connected
Jimmy
Arthur
Des Moines
Wyoming
UT
Milpas Street
[statistics] disconnected
Job test_routine ended at 14:44 04/02/2010. [exit code=0]
```

## D.5. TalendDate Routines

The TalendDate routines allow you to carry out different kinds of operations and checks concerning the format of Date expressions.

To access these routines, double-click **TalendDate** under the **system** folder:

Routine	Description	Syntax
<i>addDate</i>	Adds n days, n months, n hours, n minutes or n seconds to a Java date and returns the new date.  The Date format is: "yyyy", "MM", "dd", "HH", "mm", "ss" or "SSS".	TalendDate.addDate("String date initiale", "format Date - eg.: yyyy/MM/dd", whole n,"format of the part of the date to which n is to be added - eg.:yyyy").
<i>compareDate</i>	Compares all or part of two dates according to the format specified. Returns 0 if the dates are identical, -1 if the first date is earlier and 1 if the second date is earlier.	TalendDate.compareDate(Date date1, Date date2, "format to be compared - eg.: yyyy-MM-dd")
<i>diffDate</i>	Returns the difference between two dates in terms of days, months or years according to the comparison parameter specified.	TalendDate.diffDate(Date1(), Date2(), "format of the part of the date to be compared - eg.:yyyy")
<i>diffDateFloor</i>	Returns the difference between two dates by floor in terms of years, months, days, hours, minutes, seconds or milliseconds according to the comparison parameter specified.	TalendDate.diffDateFloor(Date1(), Date2(), "format of the part of the date to be compared - eg.:MM")
<i>formatDate</i>	Returns a date string which corresponds to the format specified.	TalendDate.formatDate("date format - eg.: yyyy-MM-dd HH:mm:ss", Date() to be formatted)
<i>formatDateLocale</i>	Changes a date into a date/hour string according to the format used in the target country.	TalendDate.formatDateLocale ("format target", java.util.Date date, "language or country code")
<i>getCurrentDate</i>	Returns the current date. No entry parameter is required.	TalendDate.getCurrentDate()
<i>getDate</i>	Returns the current date and hour in the format specified (optional). This string can contain fixed character strings or variables linked to the date. By default, the string is returned in the format, DD/MM/CCYY.	TalendDate.getDate ("Format of the string - ex: CCYY-MM-DD")

Routine	Description	Syntax
<code>getFirstDayOfMonth</code>	Changes the date of an event to the first day of the current month and returns the new date.	<code>TalendDate.getFirstDayMonth(Date)</code>
<code>getLastDayOfMonth</code>	Changes the date of an event to the last day of the current month and returns the new date.	<code>TalendDate.getLastDayMonth(Date)</code>
<code>getPartOfDay</code>	Returns part of a date according to the format specified. This string can contain fixed character strings or variables linked to the date.	<code>TalendDate.getPartOfDay("String indicating the part of the date to be retrieved, "String in the format of the date to be parsed")</code>
<code>getRandomDate</code>	Returns a random date, in the ISO format.	<code>TalendDate.getRandomDate("format date of the character string", String minDate, String maxDate)</code>
<code>isDate</code>	Checks whether the date string corresponds to the format specified. Returns the boolean value true or false according to the outcome.	<code>TalendDate.isDate(Date() to be checked, "format of the date to be checked - eg.: yyyy-MM-dd HH:mm:ss")</code>
<code>parseDate</code>	Changes a string into a Date. Returns a date in the standard format.	<code>TalendDate.parseDate("format date of the string to be parsed", "string in the format of the date to be parsed")</code>
<code>parseDateLocale</code>	Parses a .string according to a specified format and extracts the date. Returns the date according to the local format specified.	<code>TalendDate.parseDateLocale("date format of the string to be parsed", "String in the format of the date to be parsed", "code corresponding to the country or language")</code>
<code> setDate</code>	Modifies part of a date according to the part and value of the date specified and the format specified.	<code>TalendDate.setDate(Date, whole n, "format of the part of the date to be modified - eg.:yyyy")</code>
<code>TO_CHAR</code>	Converts a date to a character string.	<p><code>TalendDate.TO_CHAR(date[,format])</code></p> <ul style="list-style-type: none"> <li>• <b>date:</b> the date value you want to convert to a character string.</li> <li>• <b>format:</b> the string which defines the format of the return value.</li> </ul>
<code>TO_DATE</code>	Converts a character string to a Date/Time datatype.	<p><code>TalendDate.TO_DATE(string[, format])</code></p> <ul style="list-style-type: none"> <li>• <b>string:</b> the string you want to convert to a Date/Time datatype.</li> <li>• <b>format:</b> the format string that matches the part of the <i>string</i> argument. If not specified, the <i>string</i> value must be in the date format MM/dd/yyyy HH:mm:ss.SSS.</li> </ul> <p>For example, <code>TalendDate.TO_DATE("04/24/2017 13:55:42.123")</code> will return <i>Mon Apr 24 13:55:42 CST 2017</i>.</p>
<code>ADD_TO_DATE</code>	Adds a specified amount to one part of a datetime value, and returns a date in the same format as the date you pass to the function.	<p><code>TalendDate.ADD_TO_DATE(date, format, amount)</code></p> <ul style="list-style-type: none"> <li>• <b>date:</b> the date value you want to change.</li> <li>• <b>format:</b> the format string specifying the portion of the date value you want to change. <ul style="list-style-type: none"> <li>• Valid format strings for year: <i>Y</i>, <i>YY</i>, <i>YYY</i>, and <i>YYYY</i>.</li> <li>• Valid format strings for month: <i>MONTH</i>, <i>MM</i>, and <i>MON</i>.</li> <li>• Valid format strings for day: <i>D</i>, <i>DD</i>, <i>DDD</i>, <i>DAY</i>, and <i>DY</i>.</li> <li>• Valid format strings for hour: <i>HH</i>, <i>HH12</i>, and <i>HH24</i>.</li> <li>• Valid format string for minute: <i>MI</i>.</li> <li>• Valid format string for second : <i>SS</i>.</li> <li>• Valid format string for millisecond: <i>MS</i>.</li> </ul> </li> </ul>

Routine	Description	Syntax
		<ul style="list-style-type: none"> <li>• <b>amount:</b> the integer value specifying the amount of years, months, days, hours, and so on by which you want to change the date value.</li> </ul> <p>For example,</p> <p>if <code>TalendDate.getCurrentDate()</code> returns <i>Mon Apr 24 14:26:03 CST 2017</i>,</p> <p><code>TalendDate.ADD_TO_DATE(TalendDate.getCurrentDate(), "YY", 1)</code> will return <i>Tue Apr 24 14:26:03 CST 2018</i>.</p>

## D.5.1. How to format a Date

The **formatDate** routine is easy to use, along with a **tJava** component:

```
System.out.println(TalendDate.format("dd-MM-yyyy", new Date()));
```

The current date is initialized according to the pattern specified by the `new Date()` Java function and is displayed in the **Run** view:

```
Starting job routine1 at 17:28 25/02/2010.
2010-02-25 17:28:07
Job routine1 ended at 17:28 25/02/2010. [exit code=0]
```

## D.5.2. How to check a Date

It is easy to use the **isDate** routine, along with a **tJava** component to check if a date expression is in the format specified:

```
System.out.println(TalendDate.isDate("2010-02-09 00:00:00", "yyyy-MM-dd
HH:mm:ss"));
```

A boolean is returned in the **Run** view:

```
Starting job routine1 at 17:36 25/02/2010.
true
Job routine1 ended at 17:36 25/02/2010. [exit code=0]
```

## D.5.3. How to compare Dates

It is easy to use the **compareDate** routine, along with a **tJava** component to compare two dates, for example to check if the current date is identical to, earlier than or later than a specific date, according to the format specified.

```
System.out.println(TalendDate.compareDate(new Date(),
TalendDate.parseDate("yyyy-MM-dd", "2010/11/24", "YYYY-MM-dd")));
```

In this example the current date is initialized by the Java function `new Date()` and the value -1 is displayed in the **Run** view to indicate that the current date is earlier than the second date.

```
Starting job routine1 at 18:09 25/02/2010.
-1
Job routine1 ended at 18:09 25/02/2010. [exit code=0]
```

## D.5.4. How to configure a Date

It is easy to use the **setDate** routine, along with a **tJava** component to change the year of the current date, for example:

```
System.out.println(TalendDate.formatDate("YYYY/MM/dd HH:mm:ss", new Date()));
System.out.println(TalendDate.setDate(new Date(), 2011, "yyyy"));
```

The current date, followed by the new date are displayed in the **Run** view:

```
Starting job routine1 at 18:03 26/02/2010.
2010/02/26 18:03:14
Sat Feb 26 18:03:14 CET 2011
Job routine1 ended at 18:03 26/02/2010. [exit code=0]
```

## D.5.5. How to parse a Date

It is easy to use the **parseDate** routine, along with a **tJava** component to change a date string from one format into another Date format, for example:

```
System.out.println(TalendDate.parsedate("YYYY/MM/dd HH:mm:ss",
"1979-10-20 19:00:59"));
```

The string is changed and returned in the Date format:

```
Starting job routine1 at 11:58 01/03/2010.
Sat Oct 20 19:00:59 CET 1979
Job routine1 ended at 11:58 01/03/2010. [exit code=0]
```

## D.5.6. How to retrieve part of a Date

It is easy to use the **getPartOfDate** routine, along with a **tJava** component to retrieve part of a date, for example:

```
Date D=TalendDate.parsedate("dd-MM-yyyy HH:mm:ss", "13-10-2010 12:23:45");

System.out.println(D.toString());
System.out.println(TalendDate.getPartofDate("DAY_OF_MONTH", D));
System.out.println(TalendDate.getPartOfDate("MONTH", D));
System.out.println(TalendDate.getPartOfDate("YEAR", D));
System.out.println(TalendDate.getPartOfDate("DAY_OF_YEAR", D));
System.out.println(TalendDate.getPartOfDate("DAY_OF_WEEK", D));
```

In this example, the day of month (DAY\_OF\_MONTH), the month (MONTH), the year (YEAR), the day number of the year (DAY\_OF\_YEAR) and the day number of the week (DAY\_OF\_WEEK) are returned in the **Run** view. All the returned data are numeric data types.

```
Starting job routine at 10:52 17/12/2010.
[statistics] connecting to socket on port 3565
[statistics] connected
Wed Oct 13 12:23:45 CEST 2010
13
9
2010
286
4
[statistics] disconnected
Job routine ended at 10:52 17/12/2010. [exit code=0]
```



In the **Run** view, the date string referring to the months (MONTH) starts with 0 and ends with 11: 0 corresponds to January, 11 corresponds to December.

## D.5.7. How to format the Current Date

It is easy to use the **getDate** routine, along with a **tJava** component, to retrieve and format the current date according to a specified format, for example:

```
System.out.printIn(TalendDate.getDate(CCYY-MM-DD));
```

The current date is returned in the specified format (optional):

*Starting job routine1 at 10:58 02/03/2010.*

*2010-03-02*

*Job routine1 ended at 10:58 02/03/2010. [exit code=0]*

## D.6. TalendString Routines

The **TalendString** routines allow you to carry out various operations on alphanumerical expressions.

To access these routines, double click on **TalendString** under the **system** folder. The **TalendString** class contains the following routines:

Routine	Description	Syntax
<i>replaceSpecialCharForXML</i>	returns a string from which the special characters (eg.: <, >, &...) have been replaced by equivalent XML characters.	TalendString.replaceSpecialCharForXML ("string containing the special characters - eg.: Thelma & Louise")
<i>checkCDATAForXML</i>	identifies characters starting with <![CDATA[ and ending with ]]> as pertaining to XML and returns them without modification. Transforms the strings not identified as XML in a form which is compatible with XML and returns them.	TalendString.checkCDATAForXML("string to be parsed")
<i>talendTrim</i>	parses the entry string and removes the filler characters from the start and end of the string according to the alignment value specified: -1 for the filler characters at the end of the string, 1 for those at the start of the string and 0 for both. Returns the trimmed string.	TalendString.talendTrim("string to be parsed", "filler character to be removed", character position)
<i>removeAccents</i>	removes accents from a string and returns the string without the accents.	TalendString.removeAccents("String")
<i>getAsciiRandomString</i>	generates a random string with a specific number of characters.	TalendString.getAsciiRandomString (whole number indicating the length of the string)

### D.6.1. How to format an XML string

It is easy to run the **replaceSpecialCharForXML** routine along with a **tJava** component, to format a string for XML:

```
System.out.printIn(TalendString.replaceSpecialCharForXML("Thelma & Louise"));
```

In this example, the "&" character is replaced in order to make the string XML compatible:

```
Starting job routine1 at 15:48 02/03/2010.
Thelma & Louise
Job routine1 ended at 15:48 02/03/2010. [exit code=0]
```

## D.6.2. How to trim a string

It is easy to use the **talendTrim** routine, along with a **tJava** component to remove the string padding characters from the start and end of the string:

```
System.out.printIn(TalendString.talendTrim("**talend open studio****",
'*', -1));
System.out.printIn(TalendString.talendTrim("**talend open studio****",
'*', 1));
System.out.printIn(TalendString.talendTrim("**talend open studio****",
'*', 0));
```

The star characters are removed from the start, then the end of the string and then finally from both ends:

```
Starting job routine1 at 14:19 02/03/2010.
**talend open studio
talend open studio****
talend open studio
Job routine1 ended at 14:19 02/03/2010. [exit code=0]
```

## D.6.3. How to remove accents from a string

It is easy to use the **removeAccents** routine, along with a **tJava** component, to replace the accented characters, for example:

```
System.out.printIn(TalendString.removeAccents("sâcrebleü! "));
```

The accented characters are replaced with non-accented characters:

```
Starting job routine1 at 16:02 02/03/2010.
sacrebleu!
Job routine1 ended at 16:02 02/03/2010. [exit code=0]
```

# D.7. TalendStringUtil Routines

The **TalendStringUtil** class contains only one routine **DECODE** that allows you to search a value in a port. To access the routine, double-click **TalendStringUtil** under the **system** folder.

Routine	Description	Syntax
<b>DECODE</b>	Searches a port for a value you specify. If the function finds the value, it returns a result value, which you define. You can build an unlimited number of searches within a DECODE function.	<pre>TalendStringUtil.DECODE(value, defaultValue, search1, result1[, search2, result2]...)</pre> <ul style="list-style-type: none"> <li>• <b>value:</b> the value you want to search.</li> </ul>

Routine	Description	Syntax
		<ul style="list-style-type: none"><li>• <b>defaultValue</b>: the value you want to return if the search does not find a matching value. The default value can be set to <i>null</i>.</li><li>• <b>search</b>: the value for which you want to search. The search value must have the same datatype as the <i>value</i> argument.</li><li>• <b>result</b>: the value you want to return if the search finds a matching value.</li></ul>

Below is an example of how to use the **DECODE** routine with a **tJava** component. You need to add a **tJava** component to a new Job, then enter the following code, which will search the value for *10*, in the **Code** field on the **Basic settings** view of the **tJava** component.

```
TalendStringUtil<Integer, String> example = new TalendStringUtil<Integer, String>();
System.out.println(example.DECODE(10, "error", 5, "five", 10, "ten", 15, "fifteen", 20,
"twenty"));
```

Note that you need to create a new object of the **TalendStringUtil** type, and better to use generic type to constrain the input data, then use the object to call the **DECODE** routine.

Press **F6** to run the Job. It will return *ten*, which is the result of the value *10*.



## Appendix E. SQL template writing rules

This chapter describes the rules applied for the creation of SQL templates. It aims to help users of SQL templates in *Talend Studio* to understand and develop the SQL templates for more customized usage.

These rules provide details that you have to respect when writing the template statement, a comment line or the different relevant syntaxes.

These rules helps to use the SQL code in specific use cases, such as to access the various parameters defined in components.

## E.1. SQL statements

An SQL statement can be any valid SQL statement that the related JDBC is able to execute. The SQL template code is a group of SQL statements. The basic rules to write an SQL statement in the SQL template editor are:

- An SQL statement must end with ;.
- An SQL statement can span lines. In this case, no line should be ended with ; except the last one.

## E.2. Comment lines

A comment line starts with # or --. Any line that starts with # or -- will be ignored in code generating.



There is no exception to the lines in the middle part of a SQL statement or within the <%...%> syntax.

## E.3. The <%...%> syntax

This syntax can span lines. The following list points out what you can do with this syntax and what you should pay attention to.

- You can define new variables, use Java logical code like if, for and while, and also get parameter values.

For example, if you want to get the *FILE\_Name* parameter, use the code as follows:

```
<%
String filename = __FILE_NAME__;
%>
```

- This syntax cannot be used within an SQL statement. In other words, it should be used between two separated SQL statements.

For example, the syntax in the following code is valid.

```
#sql sentence
DROP TABLE temp_0;
<%
#loop
for(int i=1; i<10; i++){
%>
#sql sentence
DROP TABLE temp_<%=i %>;
<%
}
%>
#sql sentence
DROP TABLE temp_10;
```

In this example, the syntax is used between two separated SQL templates: `DROP TABLE temp_0;` and `DROP TABLE temp_<%=i %>;`.

The SQL statements are intended to remove several tables beginning from *temp\_0*. The code between <% and %> generate a sequence of number in loop to identify tables to be removed and close the loop after the number generation.

- Within this syntax, the <%=...%> or </.../> syntax should not be used.

<%=...%> and </.../> are also syntax intended for the SQL templates. The below sections describe related information.



Parameters that the SQL templates can access with this syntax are simple. They are often used for connection purpose and can be easily defined in components, such as *TABLE\_NAME*, *DB\_VERSION*, *SCHEMA\_TYPE*, etc.

## E.4. The <%=...%> syntax

This syntax cannot span lines and is used for SQL statement. The following list points out what you can do with this syntax and what you should pay attention to.

- This syntax can be used to generate any variable value, and also the value of any existing parameter.
- No space char is allowed after <%=.
- Inside this syntax, the <%...%> or </.../> syntax should not be used.

The statement written in the below example is a valid one.

```
#sql sentence
DROP TABLE temp_<%=__TABLE_NAME__ %>;
```

The code is used to remove the table defined through an associated component.

For more information about what components are associated with the SQL templates, see [Designing a Job](#).

For more information on the <%...%> syntax, see the previous section.

For more information on the </.../> syntax, see the following section.



Parameters that the SQL templates can access with this syntax are simple. They are often used for connection purpose and can be easily defined in components, such as *TABLE\_NAME*, *DB\_VERSION*, *SCHEMA\_TYPE*, etc.

## E.5. The </.../> syntax

This syntax cannot span lines. The following list points out what you can do with this syntax and what you should pay attention to.

- It can be used to generate the value of any existing parameter. The generated value should not be enclosed by quotation marks.
- No space char is allowed after </ or before />.
- Inside this syntax, the <%...%> or <%=...%> syntax should not be used.

The statement written in the below example is a valid one.

```
#sql sentence
DROP TABLE temp_</TABLE_NAME/>;
```

The statement identifies the *TABLE\_NAME* parameter and then removes the corresponding table.

For more information on the <%...%> and <%=...%> syntaxes, see the previous sections.

The following sections present more specific code used to access more complicated parameters.



Parameters that the SQL templates can access with this syntax are simple. They are often used for connection purpose and can be easily defined in components, such as *TABLE\_NAME*, *DB\_VERSION*, *SCHEMA\_TYPE*, etc.

## E.6. Code to access the component schema elements

Component schema elements are presented on a schema column name list (delimited by a dot "."). These elements are created and defined in components by users.

The below code composes an example to access some elements included in a component schema. In the following example, the *ELT\_METADATA\_SCHEMA* variable name is used to get the component schema.

```
<%
String query = "select ";
SCHEMA(__ELT_METADATA_SCHEMA__);
for (int i=0; i < __ELT_METADATA_SCHEMA__.length ; i++) {
query += (__ELT_METADATA_SCHEMA__[i].name + ",");
}
query += " from " + __TABLE_NAME__;
%>
<%=query %>;
```

In this example, and according to what you want to do, the *\_\_ELT\_METADATA\_SCHEMA\_\_[i].name* code can be replaced by *\_\_ELT\_METADATA\_SCHEMA\_\_[i].dbType*, *\_\_ELT\_METADATA\_SCHEMA\_\_[i].isKey*, *\_\_ELT\_METADATA\_SCHEMA\_\_[i].length* or *\_\_ELT\_METADATA\_SCHEMA\_\_[i].nullable* to access the other fields of the schema column.

The extract statement is *SCHEMA(\_\_ELT\_METADATA\_SCHEMA\_\_)*. In this statement, *ELT\_METADATA\_SCHEMA* is the variable name representing the schema parameter to be extracted. The variable name used in the code is just an example. You can change it to another variable name to represent the schema parameter you already defined.



*Make sure that the name you give to the schema parameter does not conflict with any name of other parameters.*

For more information on component schema, see [Basic Settings tab](#).

## E.7. Code to access the component matrix properties

The component matrix properties are created and changed by users according to various data transformation purposes. These properties are defined by tabular parameters, for example, the *operation* parameters or *groupby* parameters that users can define through the **tSQLTemplateAggregate** component.

To access these tabular parameters that are naturally more flexible and complicated, two approaches are available:

- The *</ . . . />* approach:

*</ . . . />* is one of the syntax used by the SQL templates. This approach often needs hard coding for every parameter to be extracted.

For example, a new parameter is created by user and is given the name *NEW\_PROPERTY*. If you want to access it by using *</NEW\_PROPERTY/>*, the below code is needed.

```
| else if (paramName.equals("NEW_PROPERTY")) {
```

```

List<Map<String, String>> newPropertyValue = (List<Map<String, String>>)
ElementParameterParser.getObjectValue(node, "__NEW_PROPERTY__");
for (int ii = 0; ii <newPropertyValue.size(); ii++) {
Map<String, String> newPropertyMap =newPropertyValue.get(ii);
realValue += ...;//append generated codes
.....
}
}

```

- The `EXTRACT(__GROUPBY__)` ; approach:

The below code shows the second way to access the tabular parameter (`GROUPBY`).

```

<%
String query = "insert into " + __TABLE_NAME__ + "(id, name, date_birth) select sum(id), name, date_birth from cust_teradata
group by";
EXTRACT(__GROUPBY__);
for (int i=0; i < __GROUPBY_LENGTH__ ; i++) {
query += (__GROUPBY_INPUT_COLUMN__[i] + " ");
}
%>
<%=query %>;

```

When coding the statements, respect the rules as follows:

- The extract statement must use `EXTRACT(__GROUPBY__)`; Upcase should be used and no space char is allowed. This statement should be used between `<%` and `%>`.
- Use `__GROUPBY_LENGTH__`, in which the parameter name is followed by `_LENGTH`, to get the line number of the tabular `GROUPBY` parameters you define in the **Groupby** area on a **Component** view. It can be used between `<%` and `%>` or `<%=` and `%>`.
- Use code like `__GROUPBY_INPUT_COLUMN__[i]` to extract the parameter values. This can be used between `<%` and `%>` or between `<%=` and `%>`.
- In order to access the parameter correctly, do not use the identical name prefix for several parameters. For example in the component, avoid to define two parameters with the names `PARAMETER_NAME` and `PARAMETER_NAME_2`, as the same prefix in the names causes erroneous code generation.

