



# Talend Open Studio for Data Integration User Guide

7.3.1

Last updated: 2020-02-23

# Contents

<b>Copyleft.....</b>	<b>4</b>
<b>What is Talend Studio?.....</b>	<b>5</b>
<b>Projects and Business Models.....</b>	<b>6</b>
Working with projects.....	6
Working with business models.....	13
<b>Data Integration.....</b>	<b>25</b>
Designing Jobs.....	25
Managing Jobs.....	96
Mapping data flows.....	132
<b>Centralizing metadata for Data Integration.....</b>	<b>189</b>
Objectives.....	189
Centralizing database metadata.....	189
Centralizing JDBC metadata.....	198
Centralizing SAS metadata.....	203
Centralizing File Delimited metadata.....	206
Centralizing File Positional metadata.....	213
Centralizing File Regex metadata.....	219
Centralizing XML file metadata.....	223
Centralizing File Excel metadata.....	243
Centralizing File LDIF metadata.....	249
Centralizing JSON file metadata.....	254
Centralizing LDAP connection metadata.....	270
Centralizing Azure Storage metadata.....	275
Centralizing Google Drive metadata.....	279
Centralizing Marketo metadata.....	281
Centralizing Salesforce metadata.....	285
Centralizing Snowflake metadata.....	289
Setting up a generic schema.....	293
Centralizing MDM metadata.....	299
Centralizing Web Service metadata.....	317
Centralizing an FTP connection.....	331
Working with Hierarchical Mapper.....	334
Exporting metadata as context and reusing context parameters to set up a connection.....	334
Using centralized metadata in a Job.....	344
<b>Using routines.....</b>	<b>346</b>
Managing routines.....	346
System routines.....	355

<b>Appendices.....</b>	<b>370</b>
Customizing Talend Studio and setting Studio preferences.....	370
Using SQL templates.....	421
SQL template writing rules.....	431

# Copyleft

Adapted for 7.3.1. Supersedes previous releases.

The content of this document is correct at the time of publication.

However, more recent updates may be available in the online version that can be found on [Talend Help Center](#).

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, Paraccel 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.

## What is Talend Studio?

Talend provides you with a range of open source and subscription Studios you can use to create your projects and manage data of any type or volume.

Using the graphical User Interface and hundreds of pre-built components and connectors, you can design your Jobs with a drag-and-drop interface and native code generation.

# Projects and Business Models

## Working with projects

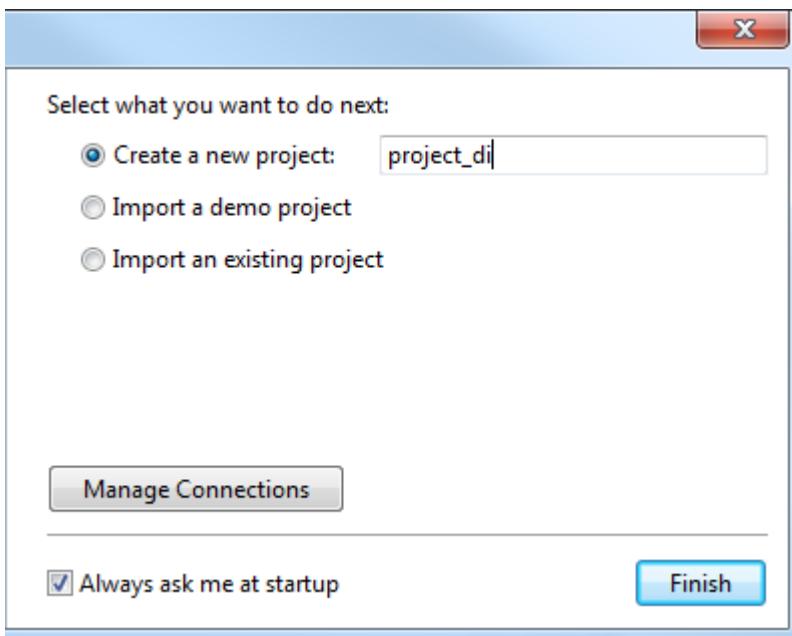
### Creating a project

A project is the highest physical structure for storing all different types of items. Once you launch your Talend Studio and before you start a Business Model, a data integration Job, a Route, or any other tasks, you need first create or import a project.

#### Creating a project at initial Studio launch

##### Procedure

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



**Note:** Bear in mind:

- A project name is case insensitive
- A project name must start with an English letter and can contain only letters, numbers, the hyphen (-), and the underscore (\_)
- The hyphen (-) character is deemed as the underscore (\_)

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

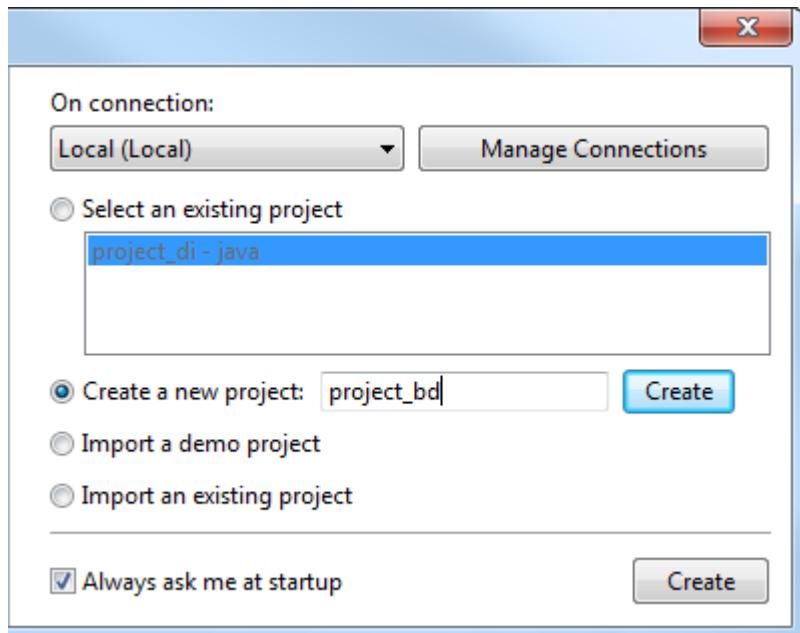
#### Creating a new project after initial Studio launch

##### About this task

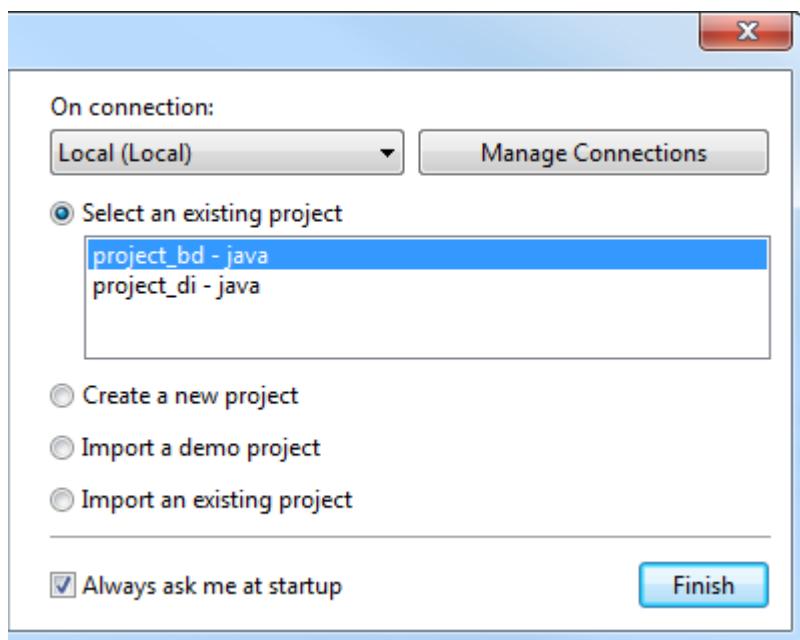
To create a new local project after the initial startup of the Studio, do the following:

##### Procedure

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



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



- 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**.

## Importing a demo project

Talend provides you with different demo projects you can import into your Talend Studio. Available demos depend on the product you are using and may include ready to use Jobs which help you understand the functionalities of different Talend components.

You can import the demo project either from the login window of your studio as a separate project, or from the **Integration** perspective into your current project.

## Importing a demo project as a separate project

### Procedure

1. Launch your Talend Studio and from the login window select **Import a demo project** and then click **Select**.
2. In the open dialog box, select the demo project you want to import and click **Finish**.

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

3. In the dialog box that opens, type in a name for the demo project you want to import and click **Finish**.  
A bar is displayed to show the progress of the operation.
4. On the login window, select from the project list the demo project you imported and then click **Finish** to open the demo project in the studio.  
All the samples of the demo project are imported into the studio under different folders in the repository tree view including the input files and metadata connection necessary to run the demo samples.

## Importing a demo project into your current project

### Procedure

1. Launch your studio and in the **Integration** perspective, click the  icon on the toolbar.
2. In the open dialog box, select the demo project to import and click **Finish**.  
A bar is displayed to show the progress of the operation and then a confirmation message opens.
3. Click **OK**.

## Importing projects

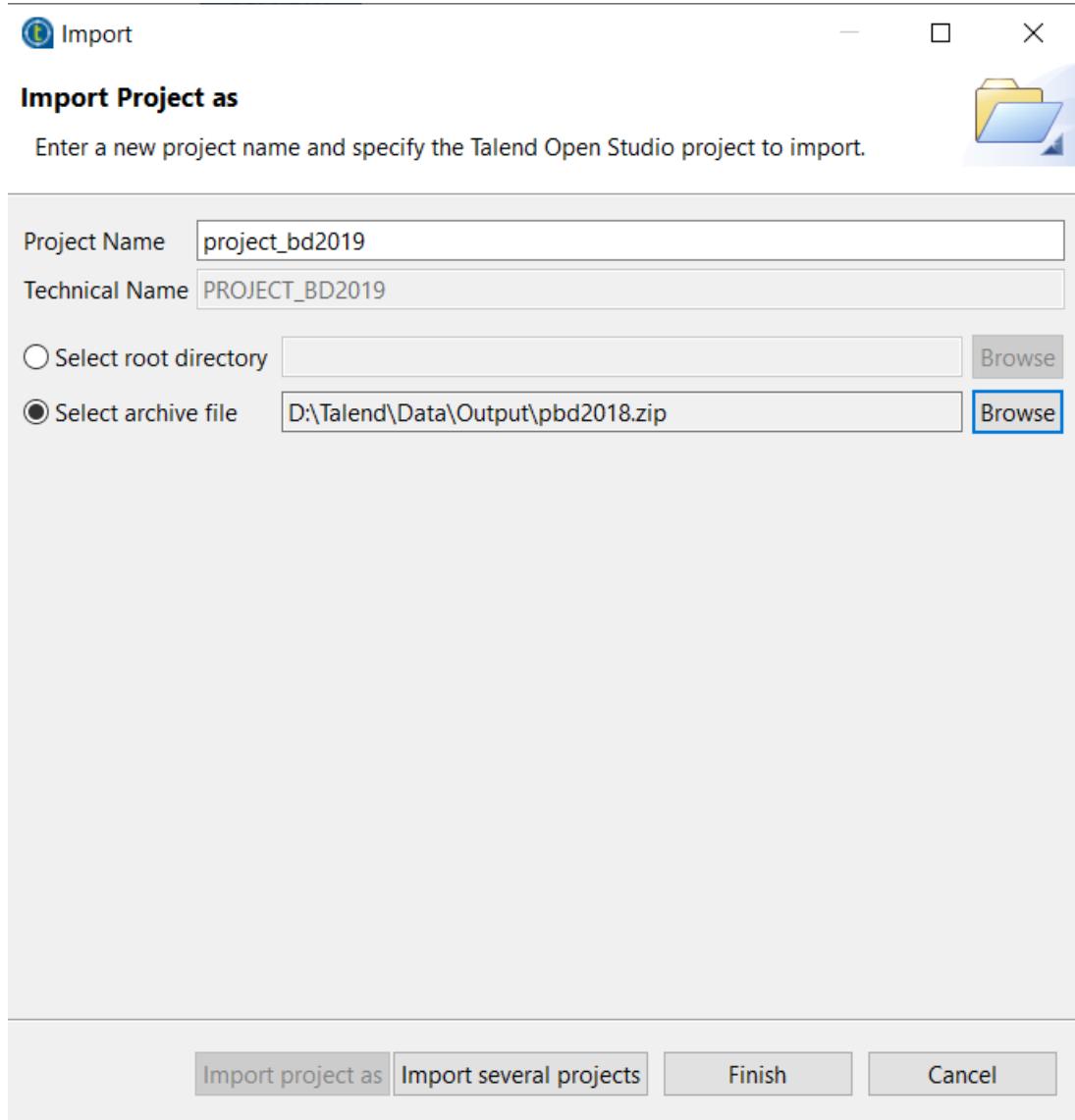
In Talend Studio, you can import one or more projects you already created with previous releases of the Studio.

### About this task

#### Importing a single project

### Procedure

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



2. Enter a name for your new project in the **Project Name** field.

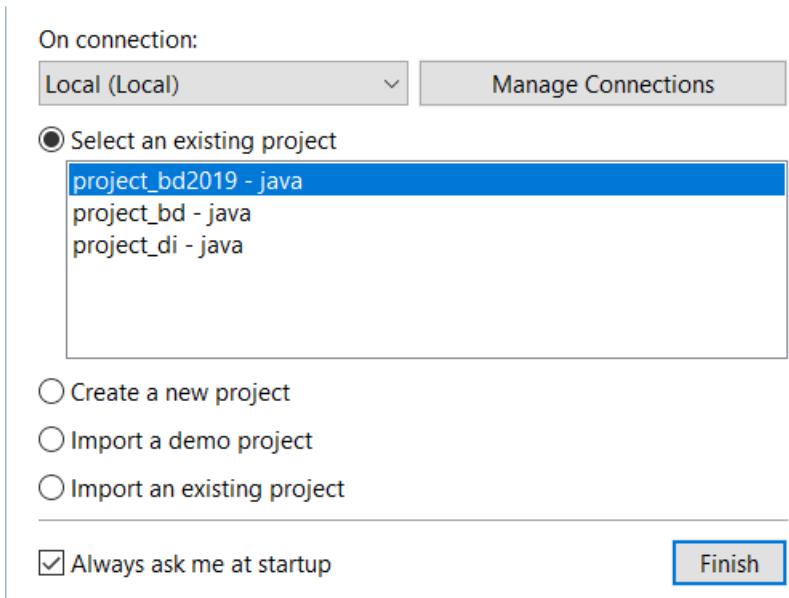
**Warning:** Make sure the project name you entered is unique, and bear in mind:

- A project name is case insensitive
- A project name must start with an English letter and can contain only letters, numbers, the hyphen (-), and the underscore (\_)
- The hyphen (-) character is deemed as the underscore (\_)

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.

## Results

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



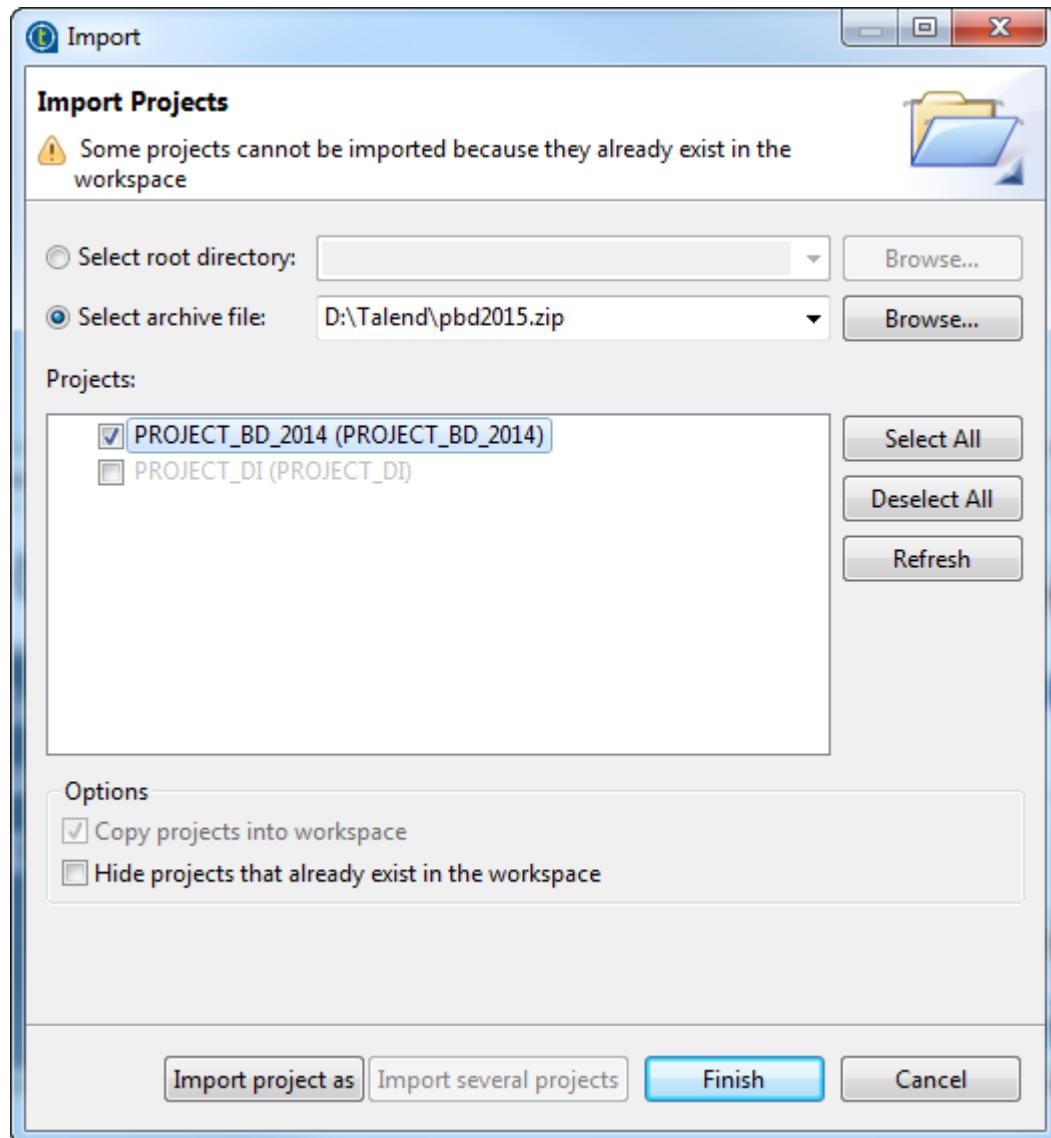
You can now select the imported project you want to open in Talend Studio and click **Finish** to launch the Studio.

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

## Importing multiple projects

### Procedure

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.

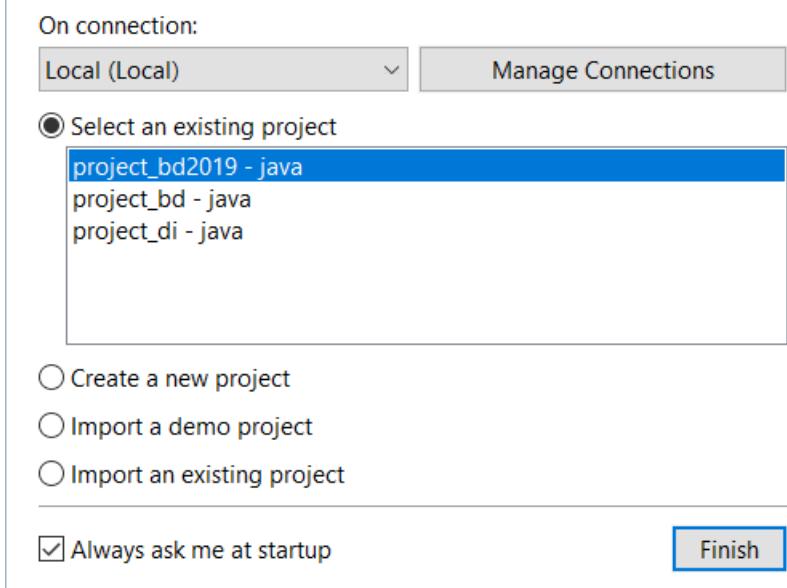
This option is available only when you import several projects from a root directory.

**Note:** 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.

## Results

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.

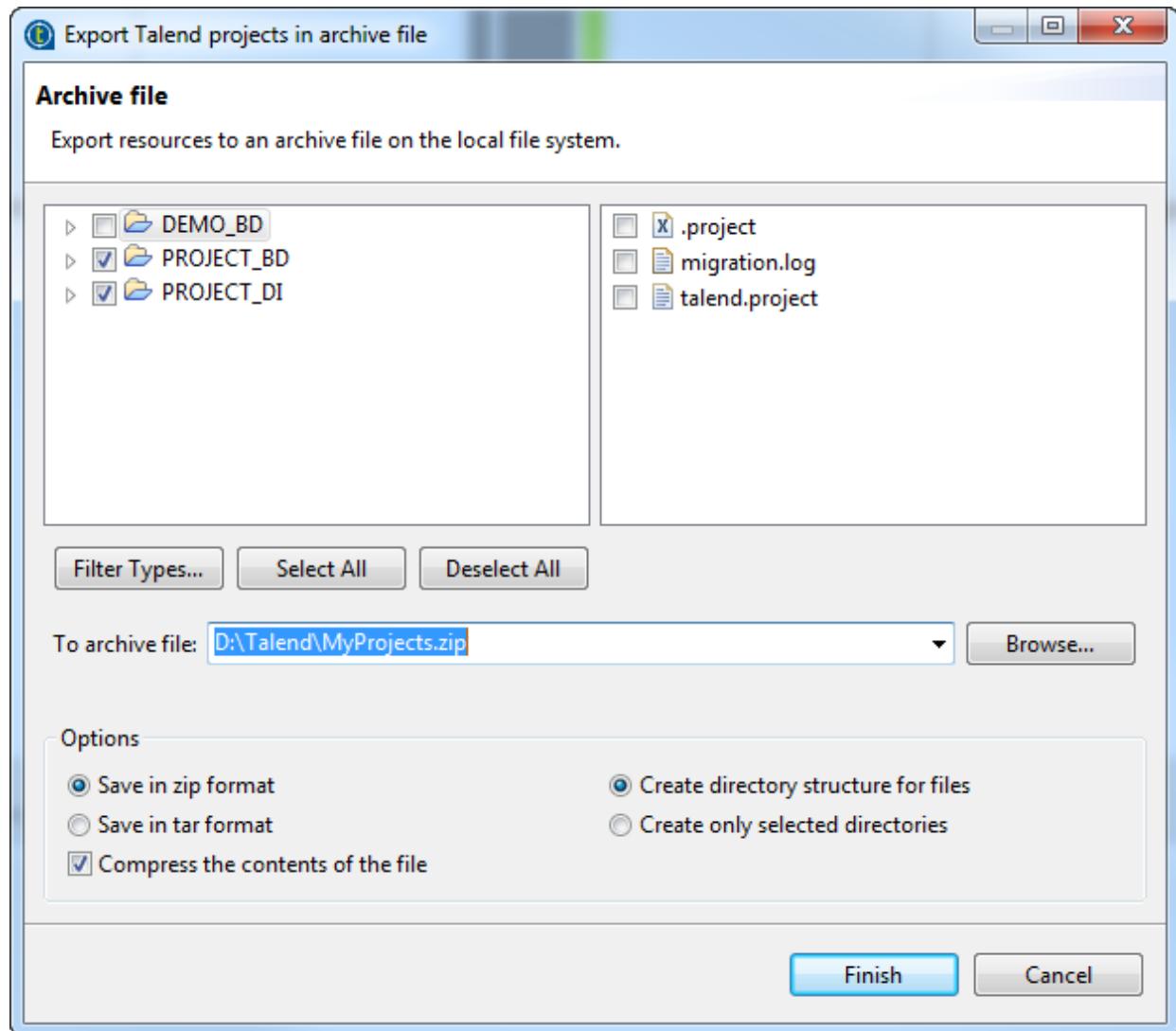
**Note:** A generation initialization window might come up when launching the application. Wait until the initialization is complete.

## Exporting a project

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

### Procedure

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.

## Results

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

# Working with business models

## 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. 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.

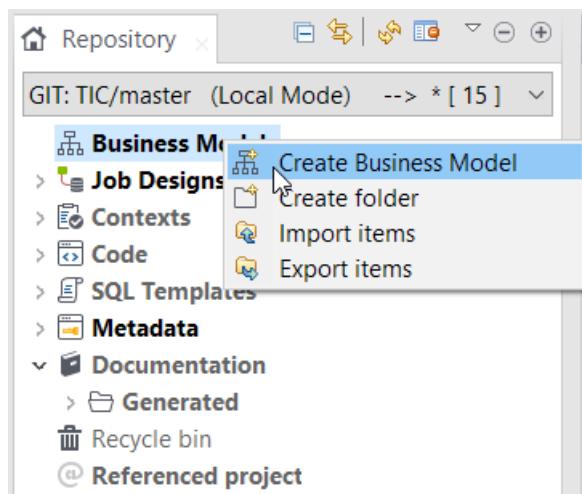
In the design workspace of the **Integration** perspective of Talend Studio, you can use multiple tools in order to:

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

## Creating a Business Model

### Procedure

1. In the **Repository** tree view of the **Integration** perspective, 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.

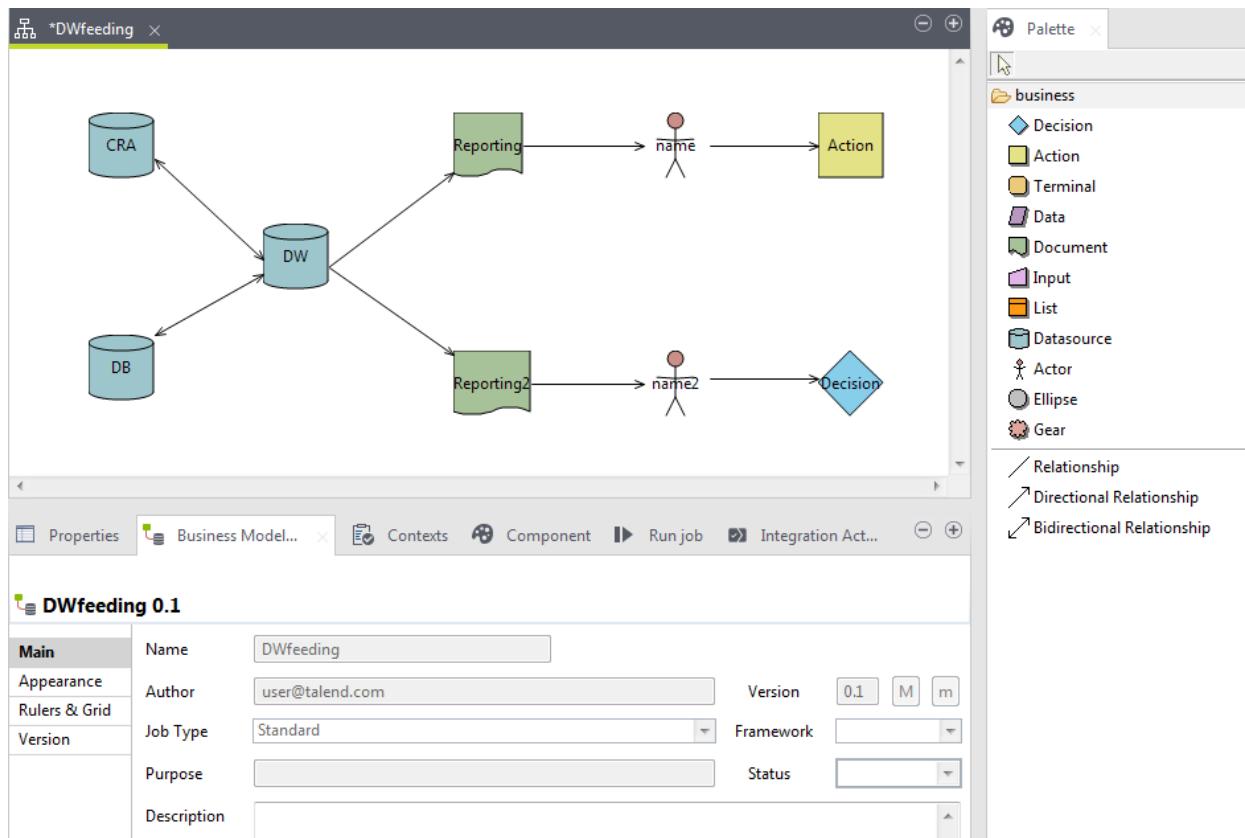
2. Enter the Business Model properties according to the following table:

Field	Description
<b>Name</b>	Name of the new Business Model. A message comes up if you enter prohibited characters.
<b>Purpose</b>	Business Model purpose or any useful information regarding the Business Model use.
<b>Description</b>	Business Model description.
<b>Author</b>	Read-only field that shows by default the current user login.

Field	Description
<b>Locker</b>	Read-only field that shows by default the login of the user who owns the lock on the current Business Model. 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.
<b>Version</b>	Read-only field. You can manually increment the version using the <b>M</b> and <b>m</b> buttons.
<b>Status</b>	List to select from the status of the Business Model you are creating.
<b>Path</b>	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:

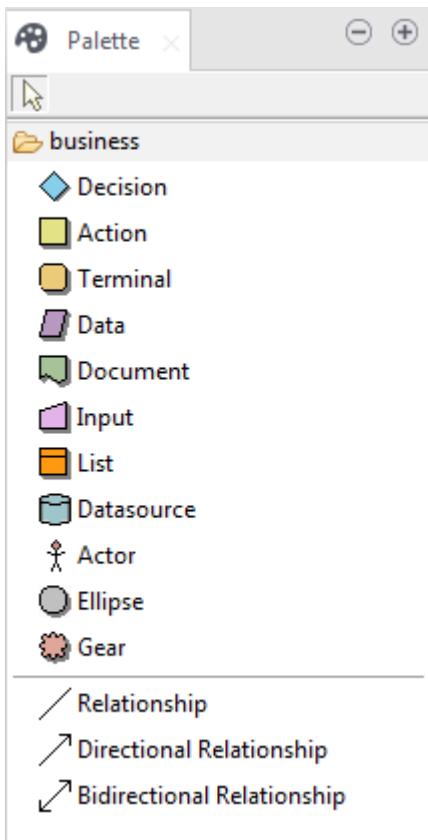
- the **Integration** perspective's design workspace
- a **Palette** of shapes and lines specific to the business modeling
- the **Business Model** panel showing specific information about all or part of the model.

## 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**.

### 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:

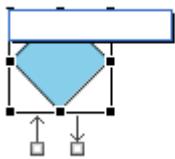


For instance, if your business process includes a decision step, select the diamond shape in the **Palette** to add this decision step to your model.

**Note:** 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](#) on page 17.

The available shapes include:

Callout	Details
Decision	The diamond shape generally represents an <b>if</b> 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 <b>Talend</b> Job for example.

## Connecting shapes

### About this task

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:

- 
- ↗ Relationship
  - ↗ Directional Relationship
  - ↙ Bidirectional Relationship

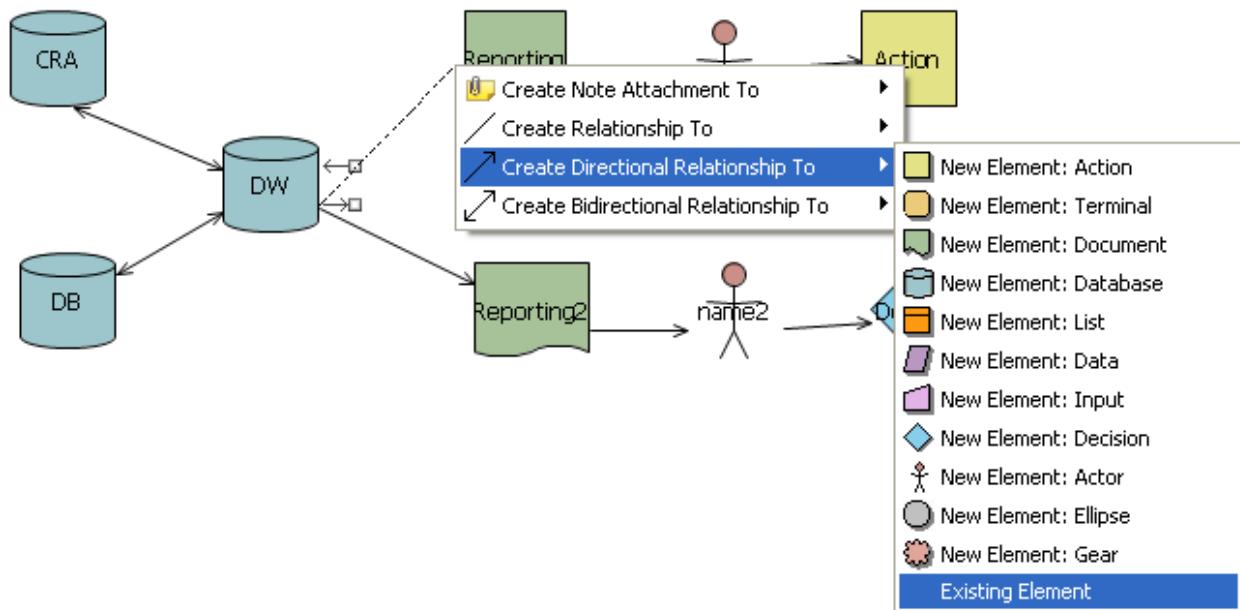
Either select the relevant **Relationship** tool in the **Palette**. Then, in the design workspace, pull a link from one shape to the other to draw a connection between them.

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

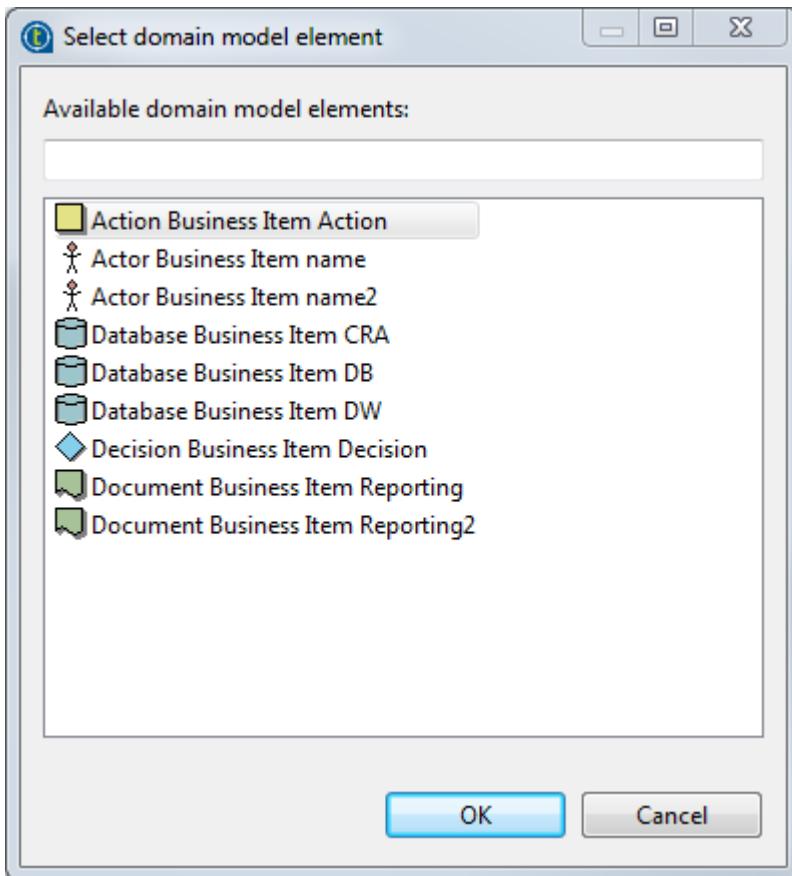
### Procedure

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.

### Results



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](#) on page 21.

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.

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

Related topic: [Commenting and arrange a model](#) on page 19.

### Commenting 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 Shift 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.

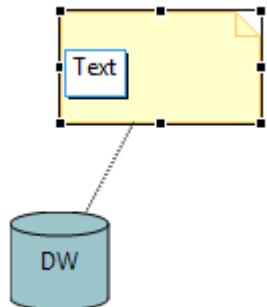
### Adding a note or free text

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.

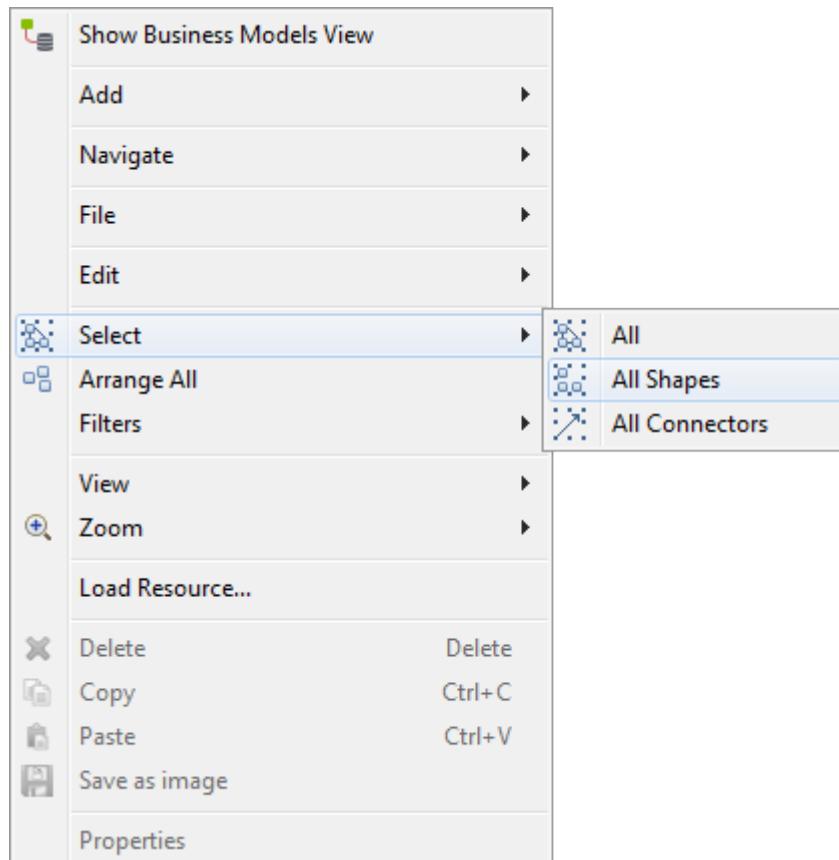


If you want to link your notes and specific shapes of your model, click the down arrow next to the **Note** tool on the **Palette** and select **Note attachment**. Pull the black arrow towards an empty area of the design workspace, and release. The popup menu offers you to attach a new Note to the selected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

### Arranging 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.

## **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. For more information about the **Main** tab, see [Displaying Job configuration tabs/views](#) on page 396.

### **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:

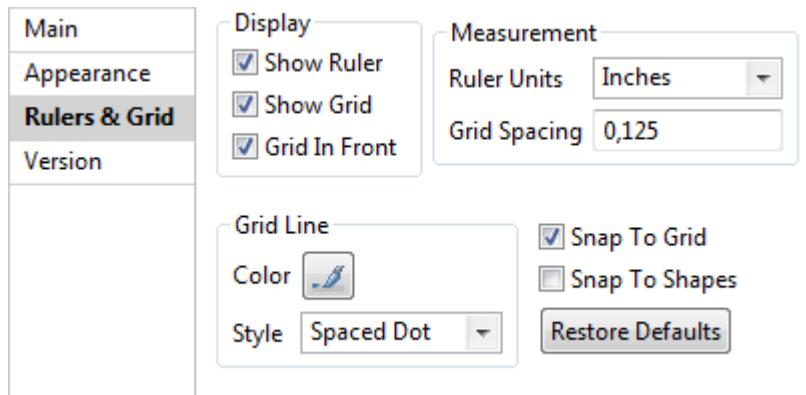
- fill the shape with selected color.
- color the shape border
- insert text above the shape
- insert gradient colors to the shape
- insert shadow to the shape

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

### **Rulers and Grid tab**

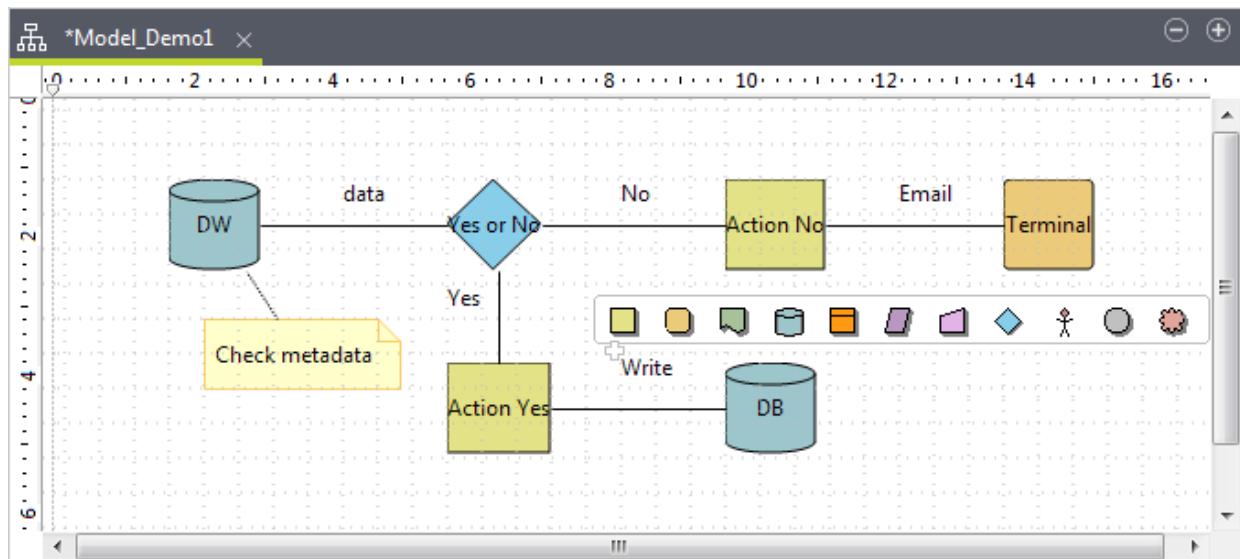
#### **Procedure**

1. To display the **Rulers & Grid** tab, click  on the **Palette**, then click any empty area of the design workspace to deselect any current selection.
2. Click the **Rulers & Grid** tab to access the ruler and grid setting view.
3. 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.
4. In the **Measurement** area, select the ruling unit among **Centimeters**, **Inches** or **Pixels**.



5. 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.
6. 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.



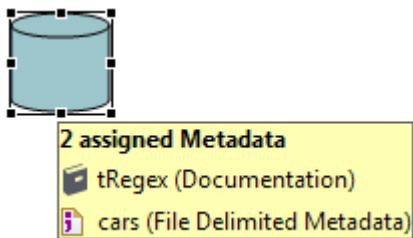
### Assignment tab

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.

The screenshot shows the Business Model view in a software application. The title bar includes tabs for 'Proper...', 'Business Models' (which is selected), 'Contexts', 'Comp...', 'Run job', and 'Integr...'. Below the title bar, the window title is 'DWfeeding 0.1'. On the left, there's a vertical toolbar with icons for Appearance and Assignment. The main area has a table titled 'Assignment' with columns 'Type', 'Name', and 'Comment'. Two items are listed: 'Documentation' (tRegex) and 'File Delimited Metadata' (cars). A scroll bar is visible at the bottom right of the table area.

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

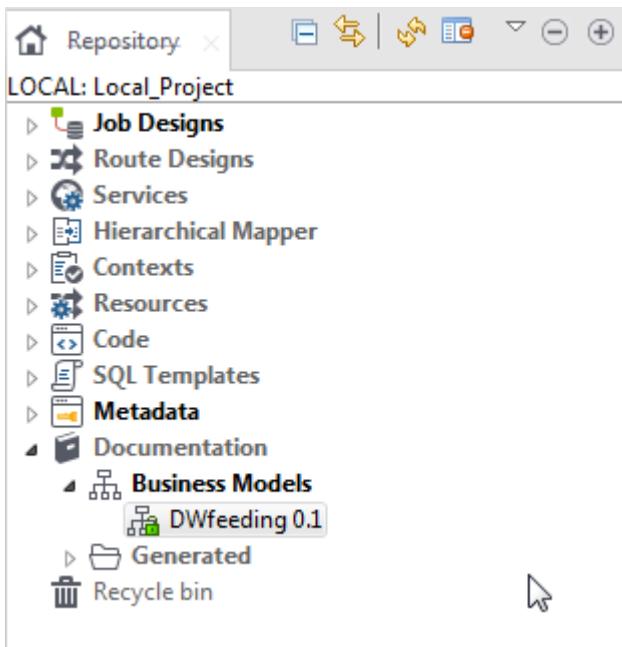


You can modify some information or attach a comment. Also, if you update data from the **Repository** tree view, assignment information gets automatically updated.

For further information about how to assign elements to a Business Model, see [Assigning repository elements to a Business Model](#) on page 23.

## 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 <b>Code</b> folder of the <b>Repository</b> tree view.

For more information about the **Repository** elements, see [What is a Job design?](#) on page 25.

# Data Integration

## Designing Jobs

### 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).

#### Warning:

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

## 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.

### Creating a Job

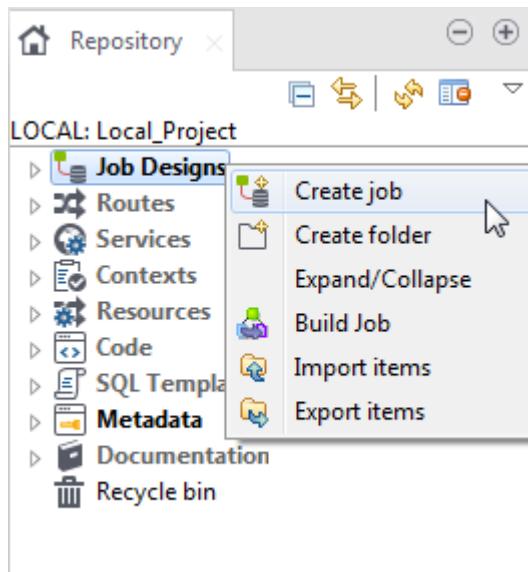
Talend Studio enables you to create a Job by dropping different technical components from the **Palette** onto the design workspace and then connecting these components together.

### About this task

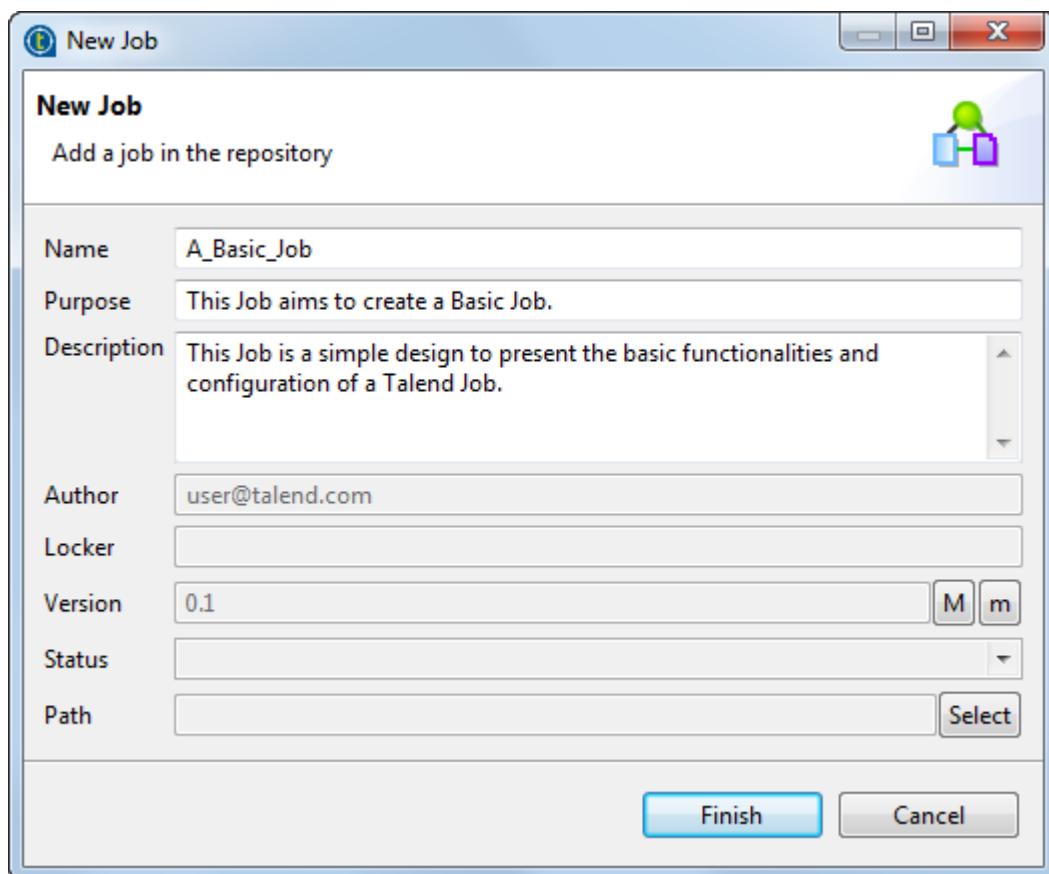
To create the example Job described in this section, proceed as follows:

### Procedure

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.



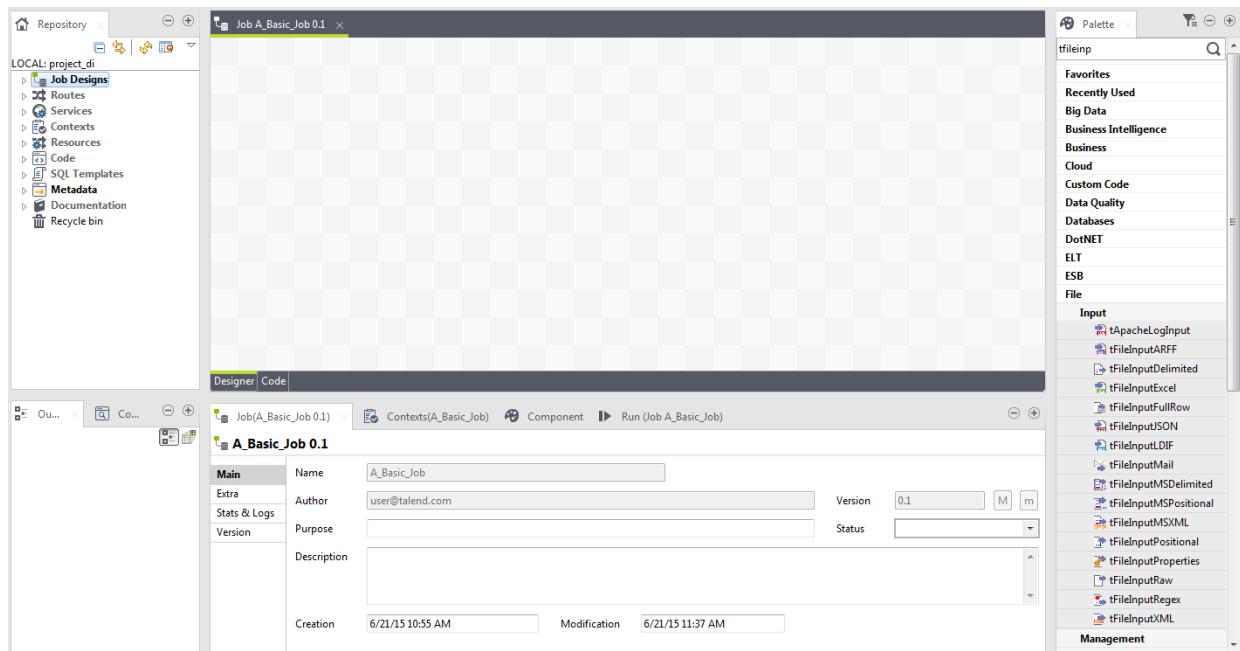
- Fill the Job properties as shown in the previous screenshot.

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
<b>Description</b>	Job description containing any information that helps you describe what the Job does and how it does it.
<b>Author</b>	a read-only field that shows by default the current user login.
<b>Locker</b>	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.
<b>Version</b>	a read-only field. You can manually increment the version using the <b>M</b> and <b>m</b> buttons. For more information, see <a href="#">Managing Job versions</a> on page 112.
<b>Status</b>	a list to select from the status of the Job you are creating.
<b>Path</b>	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.



## Results

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.

## 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 [Centralizing database metadata](#) on page 189 .

### Dropping the first component from the Palette

#### About this task

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:

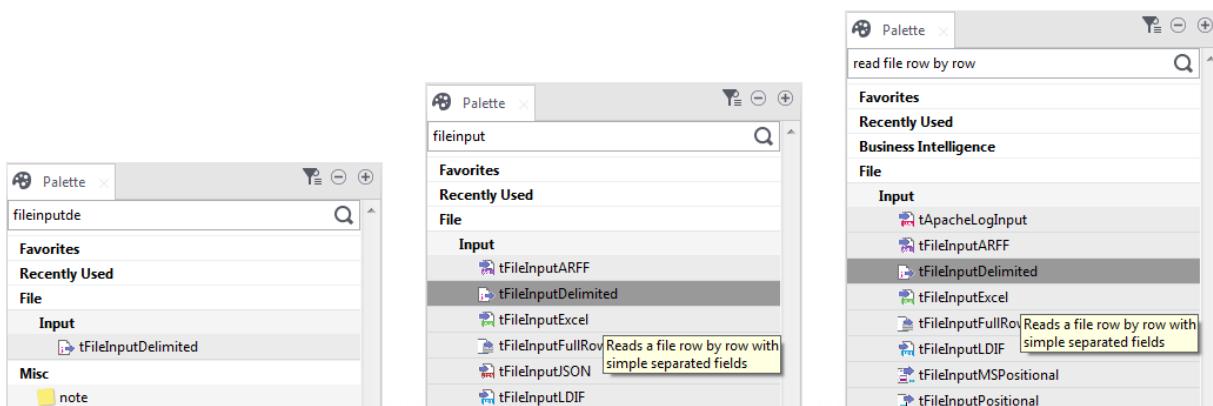
#### Procedure

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, `fileinputde`, `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 [Changing the Palette layout and settings](#) on page 391.

#### Note:

- 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 \(Talend> Palette Settings\)](#) on page 412.
- Components may not always have a prefix letter in the name. Therefore, as a best practice, you may want to simply specify the main body when searching for a component by its name.

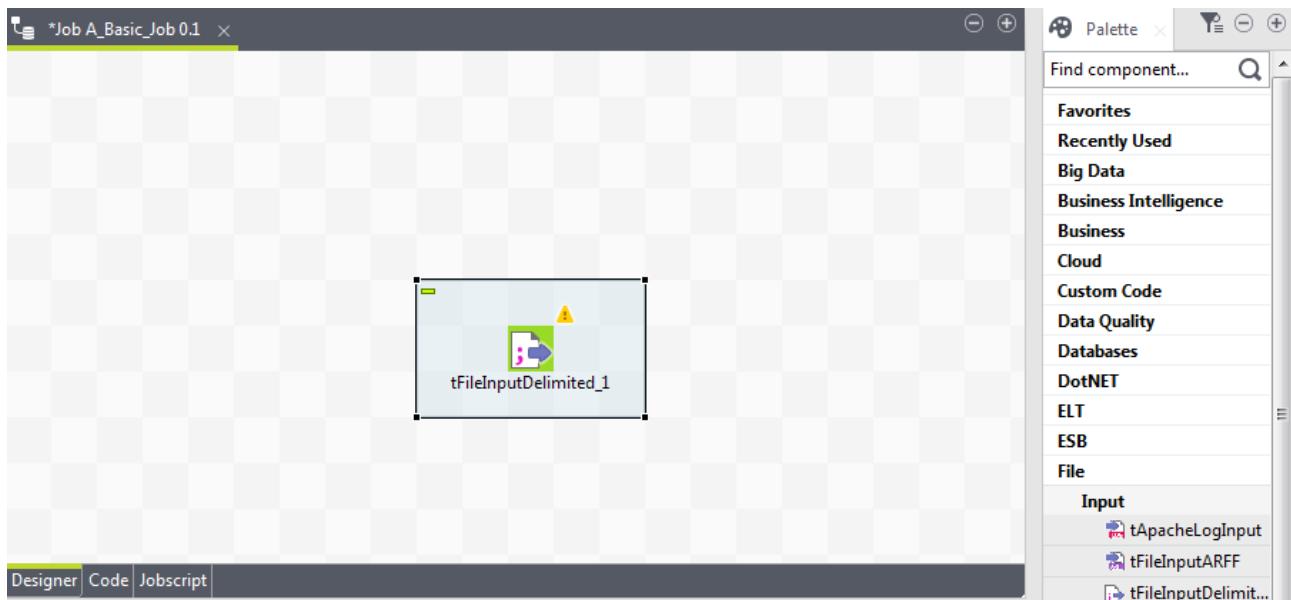


2. Select the component you want to use and click on the design workspace where you want to drop the component.

## Results

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.



### Adding the second component by typing on the design workspace

#### About this task

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 [Using centralized metadata in a Job](#) on page 344.

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

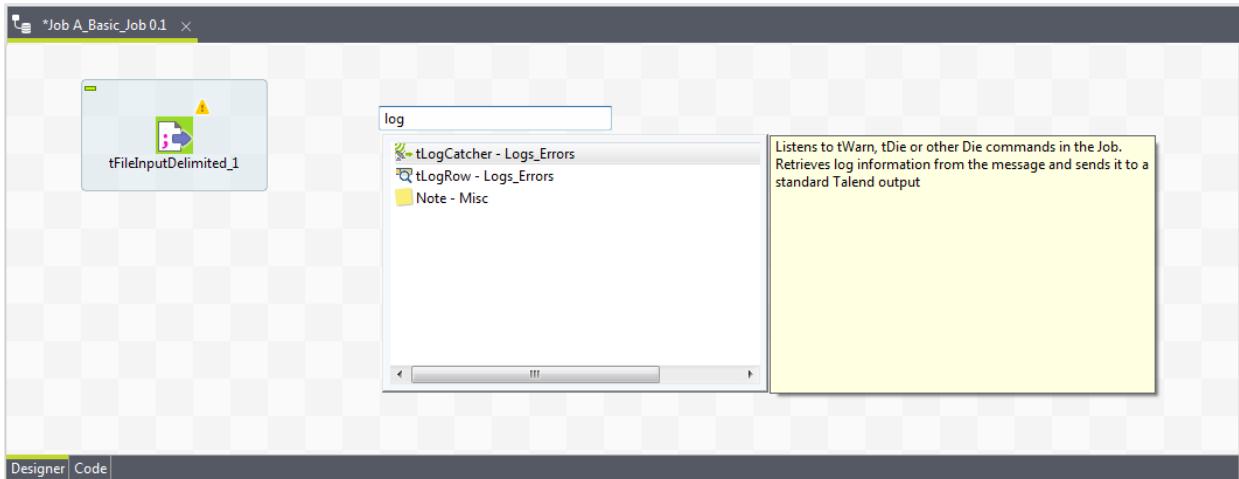
#### Procedure

- 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 log.

#### Note:

- 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 \(Talend> Palette Settings\)](#) on page 412.
- Components may not always have a prefix letter in the name. Therefore, as a best practice, you may want to simply specify the main body when searching for a component by its name.

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



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

### Adding an output component by dragging from an input one

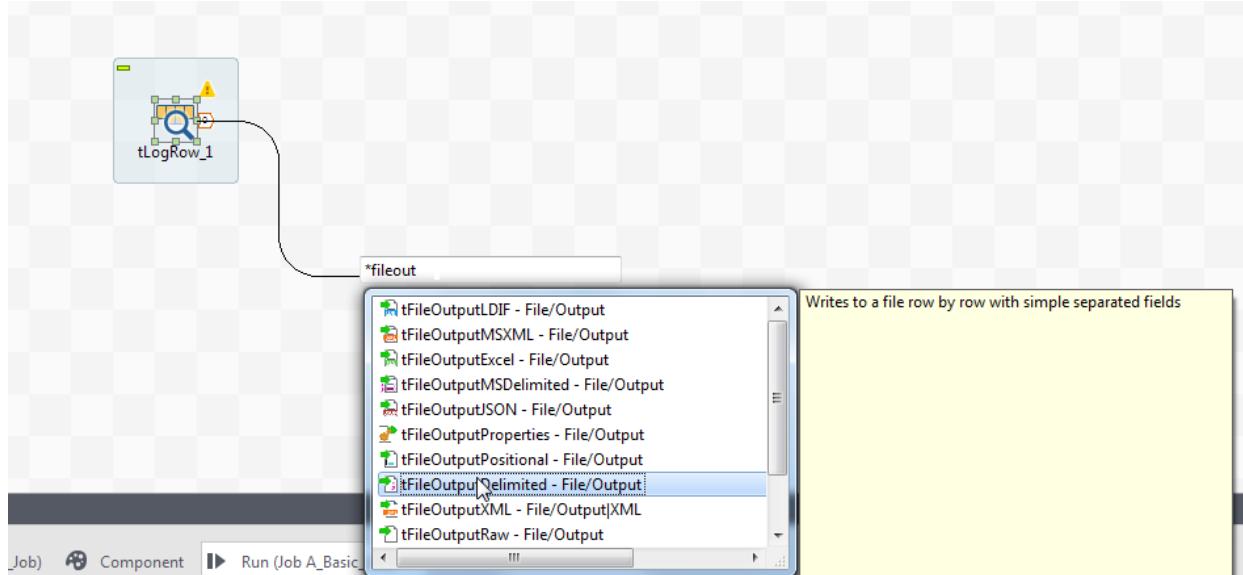
#### About this task

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.

#### Procedure

- Click the **tLogRow** component to show the **o** icon docked to it.
- 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.

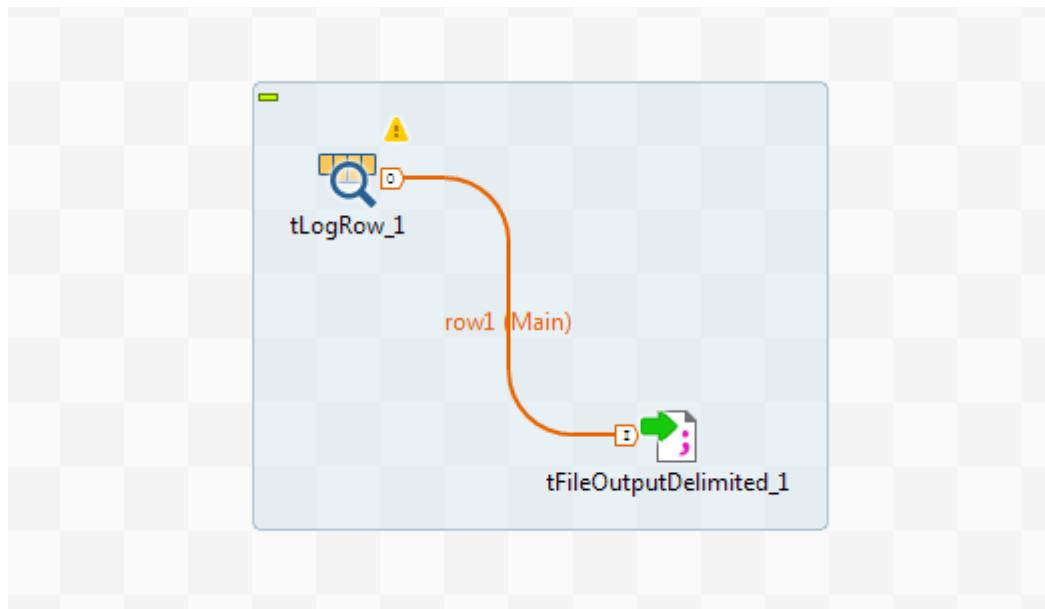


- To narrow the search, type in the text field the name of the component you want to add or part of it, or a phrase describing the component's functionality if you don't know its name, and then double-click the component of interest, **tFileOutputDelimited** in this example, on the component list to add it onto the design workspace.

**Note:**

- 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)** on page 412.
- Components may not always have a prefix letter in the name. Therefore, as a best practice, you may want to simply specify the main body when searching for a component by its name.

The new component is automatically connected with the input component **tLogRow**, using a **RowMain** connection.



### 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.

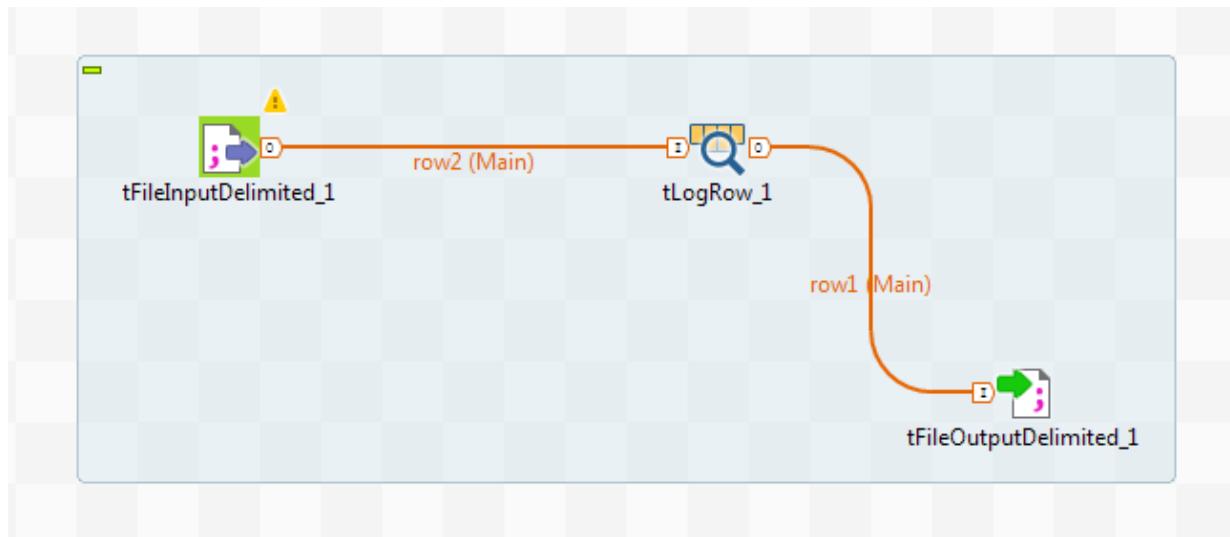
In this example, as the **tLogRow** and **tFileOutputDelimited** components are already connected, you only need to connect the **tFileInputDelimited** to the **tLogRow** component.

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

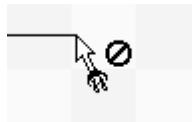
#### Right-click and click again

#### Procedure

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, **RowMain** 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.

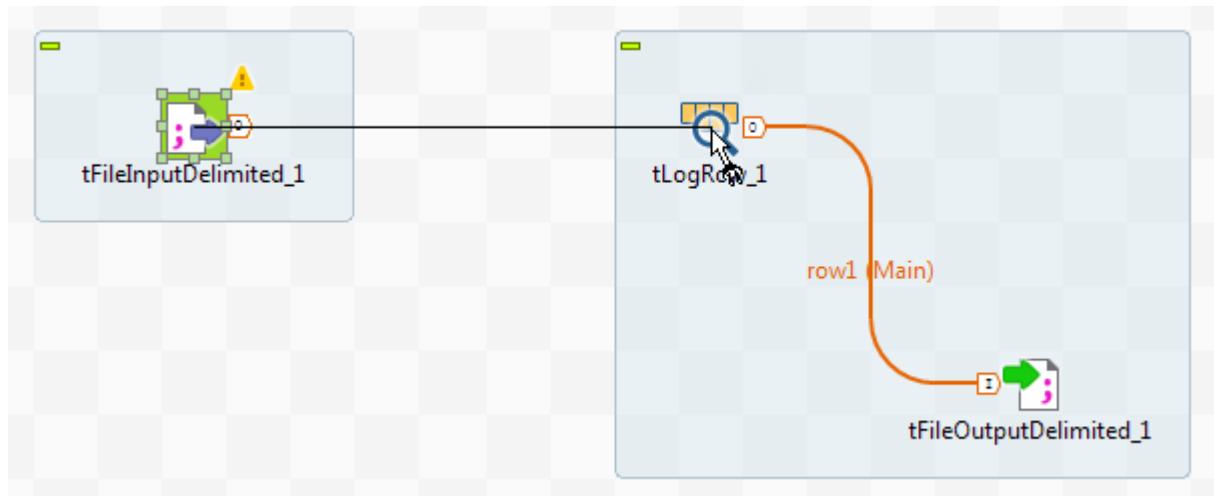


According to the nature and the role of the components you want to link together, several types of link are available. Only the authorized connections are listed in the contextual menu.

### Drag and drop

#### Procedure

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.

## Configuring the components

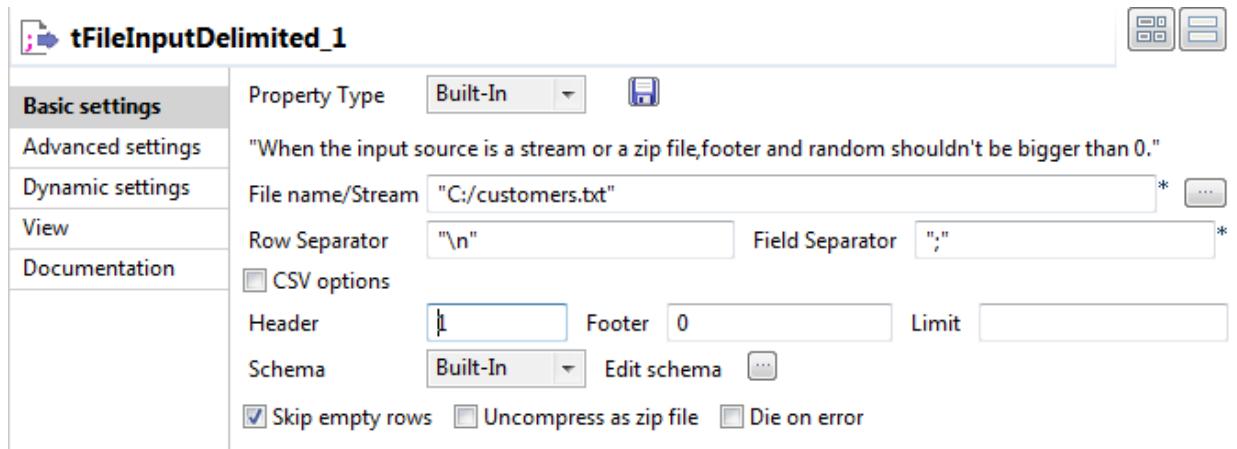
Now that the components are linked, their properties should be defined.

For more advanced details regarding the components properties, see [Defining component properties](#) on page 39.

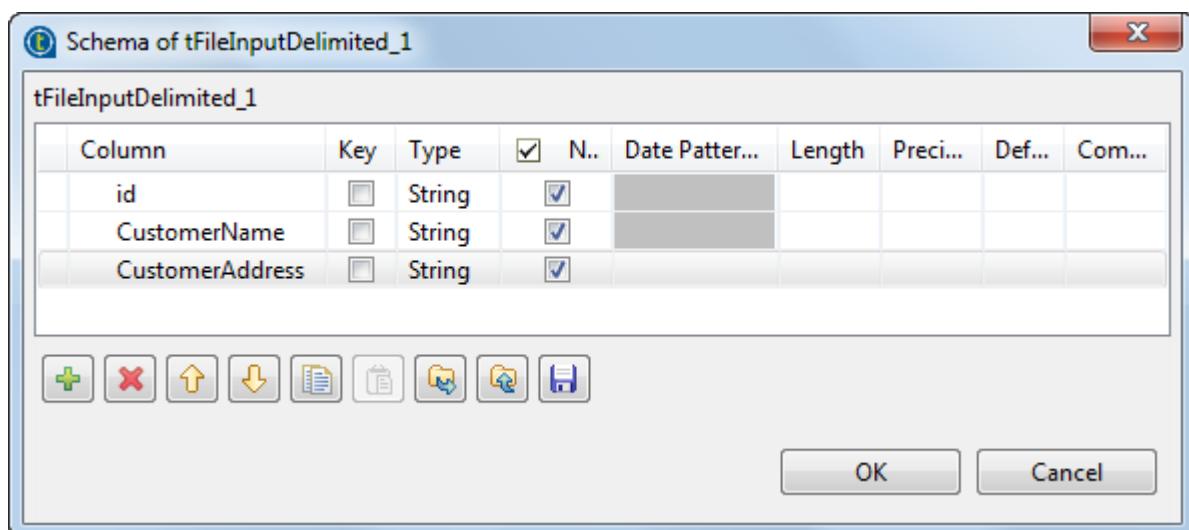
### Configuring the tFileInputDelimited component

#### Procedure

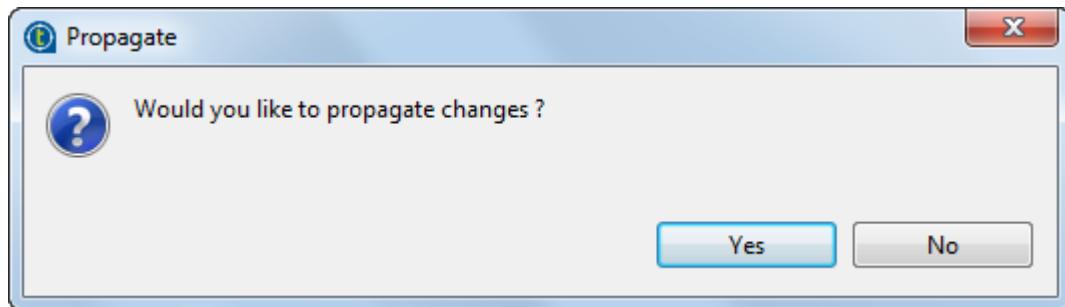
- Double-click the **tFileInputDelimited** component to open its **Basic settings** view.



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



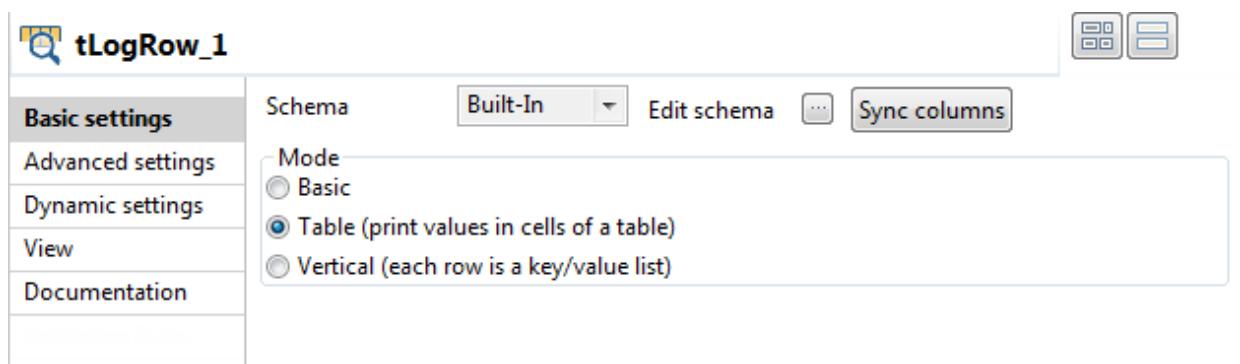
- In the pop-up that opens, click **OK** accept the propagation of the changes.  
This allows you to copy the schema you created to the next component, **tLogRow** in this example.



## Configuring the tLogRow component

### Procedure

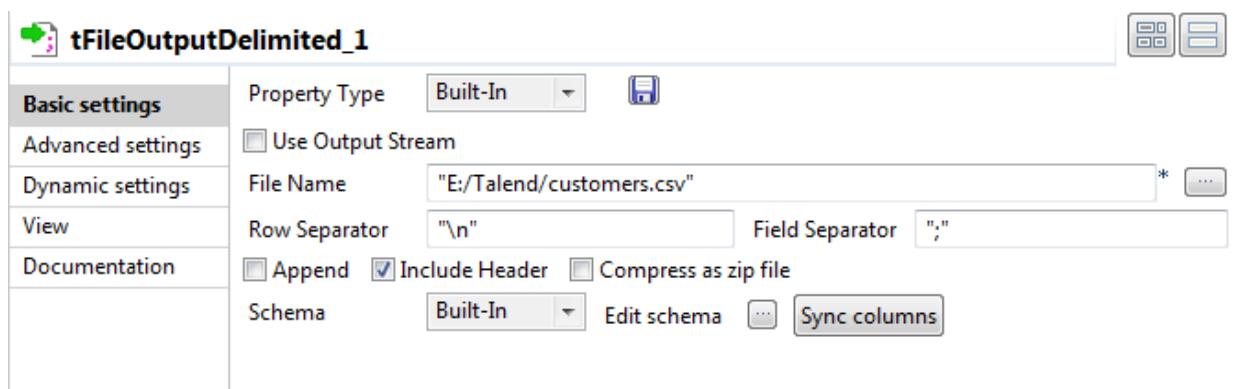
1. Double-click the **tLogRow** component to open its **Basic settings** view.
2. In the **Mode** area, select **Table (print values in cells of a table)**.



## Configuring the tFileOutputDelimited component

### Procedure

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.

## Executing the Job

### About this task

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

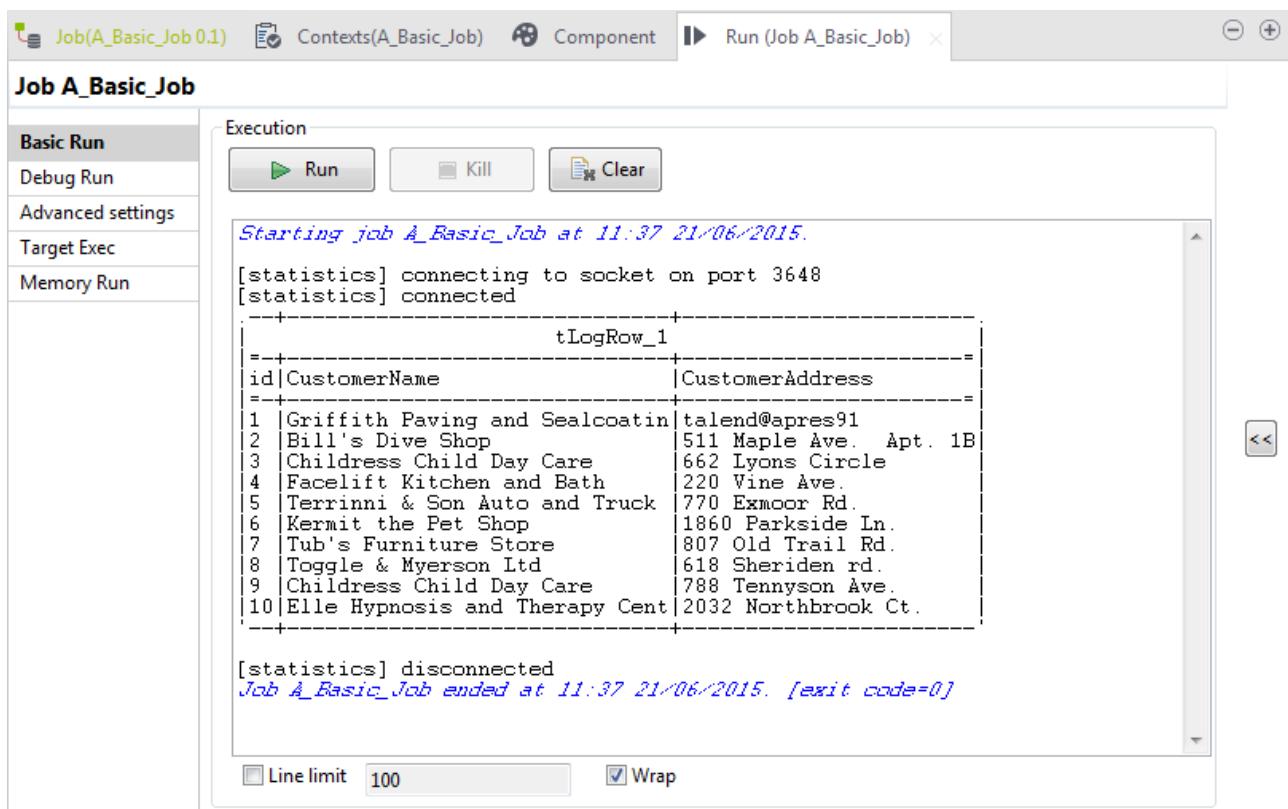
To do so, proceed as follows:

### Procedure

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

### Results

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



The screenshot shows the Talend Data Integration interface with the 'Run' tab selected for 'Job A\_Basic\_Job'. The console output window displays the following log:

```

Starting job A_Basic_Job at 11:37 21/06/2015.
[statistics] connecting to socket on port 3648
[statistics] connected
---+
tLogRow_1
---+
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 |Terrinini & 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]

```

At the bottom of the console window, there are checkboxes for 'Line limit' (unchecked) and 'Wrap' (checked).

## Working with components

The sections below give detailed information about various subjects related to handling components in Jobs , including:

- [Adding a component between two connected components](#) on page 36
- [Defining component properties](#) on page 39
- [Finding Jobs containing a specific component](#) on page 48
- [Setting default values in the schema of a component in a Job](#) on page 50

## Adding a component between two connected components

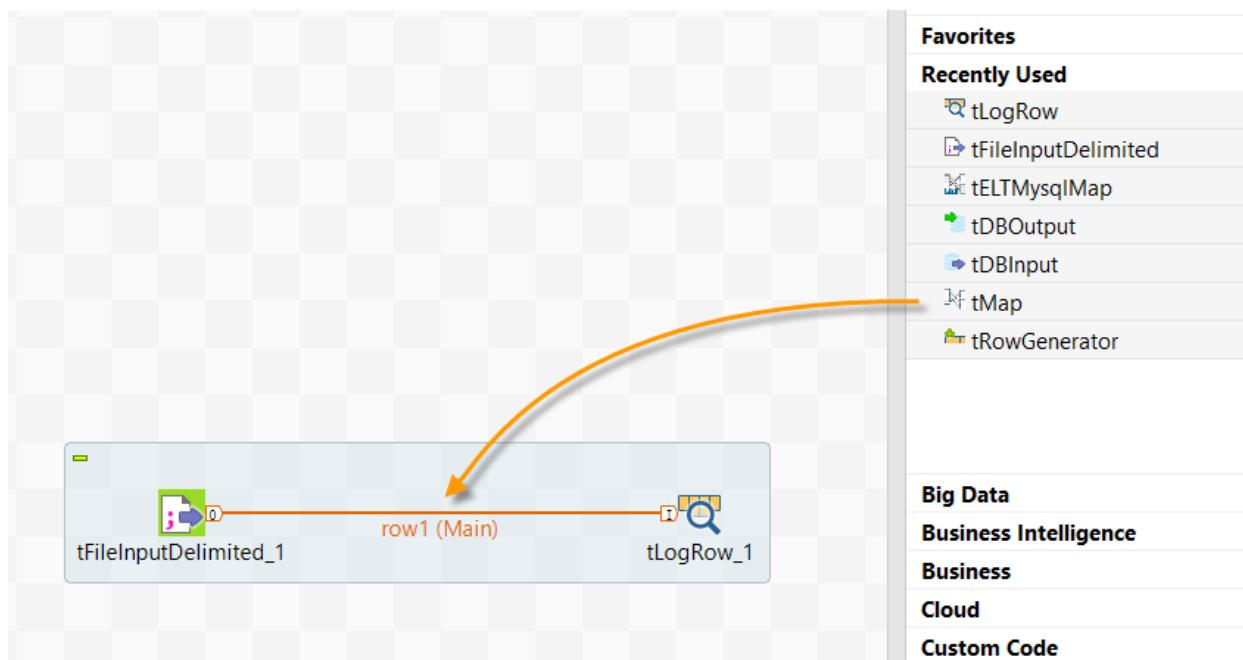
When designing a Job, you can insert a component between two components linked by a **Row** connection, provided that the new component can serve as a middle component between the two.

The examples below show different options for you to insert a **tMap** between a **tFileInputDelimited** and **LogRow** linked by a **Row > Main** connection.

### Dropping the component from the Palette onto the connection

#### Procedure

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

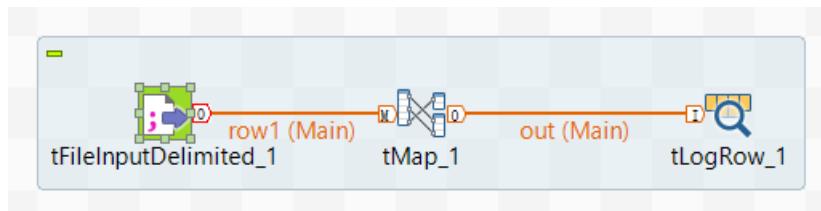


If you are prompted to give a name to the output connection from the newly added component, which is true in the case of a **tMap**, type in a name and click **OK** to close the dialog box.

#### Note:

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

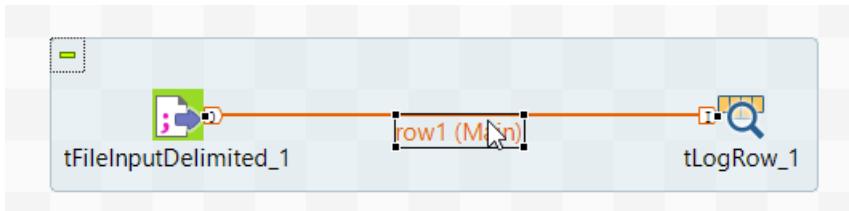
The component is inserted in the middle of the connection, which is now divided into two connections.



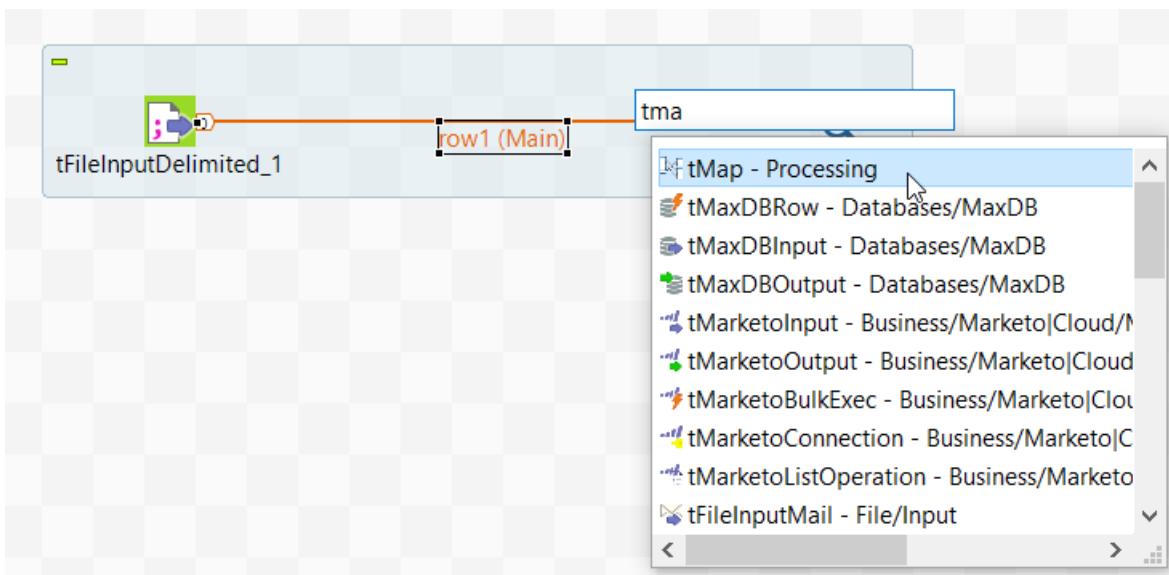
## Adding the component by typing on the connection

### Procedure

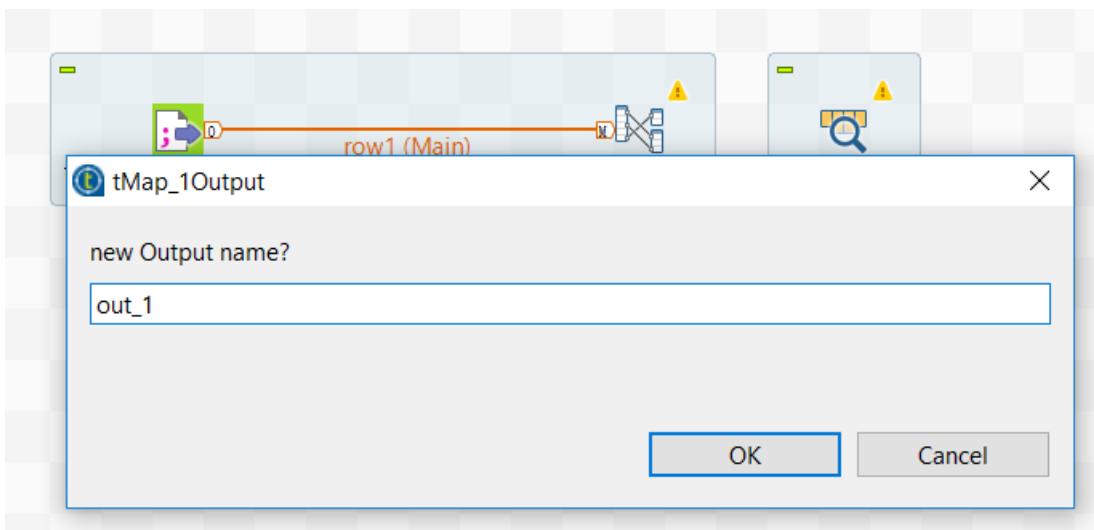
- Click on the connection that links the two existing components to select it.



- 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.



- 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.



#### Note:

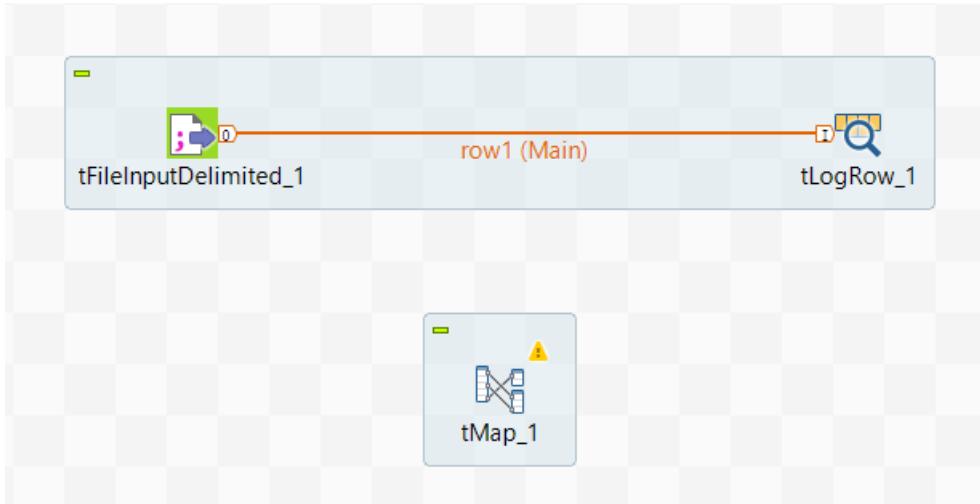
You may be asked to retrieve the schema of the target component. In that case, click **OK** to accept or click **No** to deny.

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

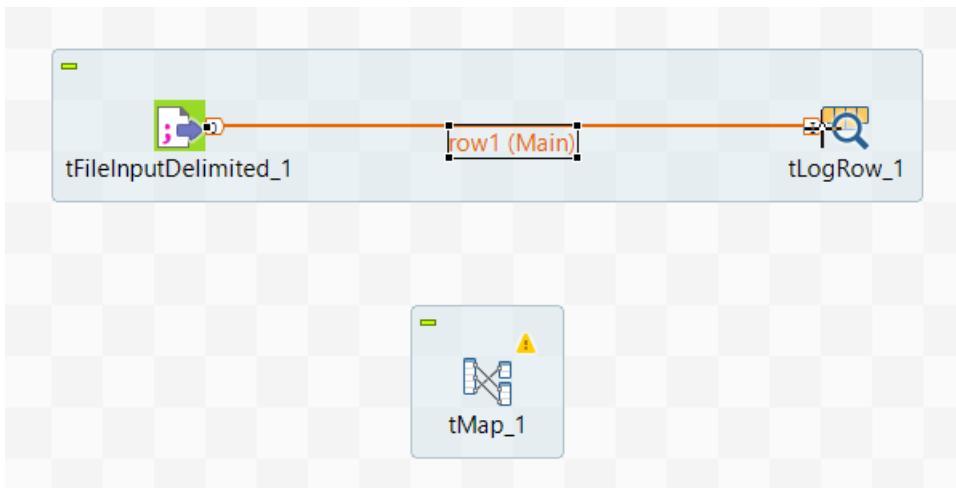
### Adding the component to the design workspace and moving the existing connection

#### Procedure

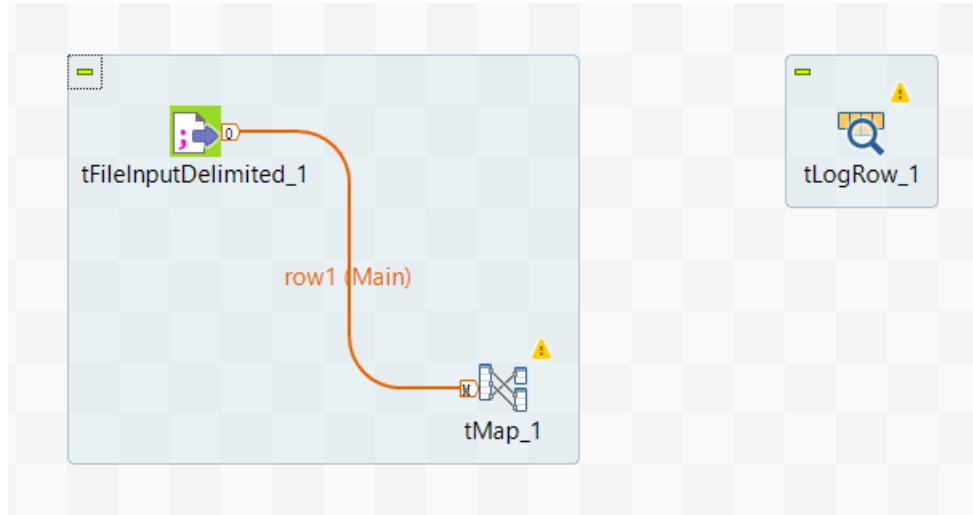
1. Add the new component, **tMap** in this example, onto the design workspace by either dropping it from the **Palette** or clicking in the design workspace and typing the component name.



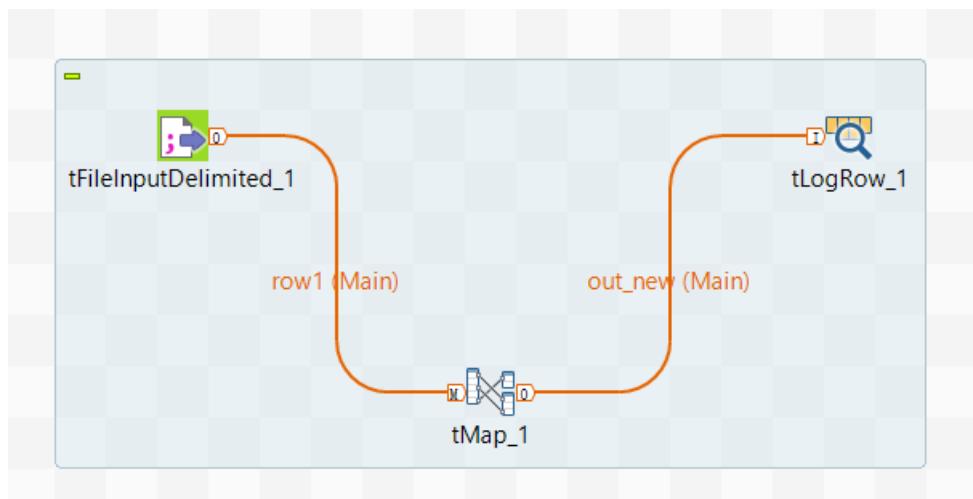
2. Select the connection and move your mouse pointer towards the end of the connection until the mouse pointer becomes a + symbol.



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



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



## Defining component properties

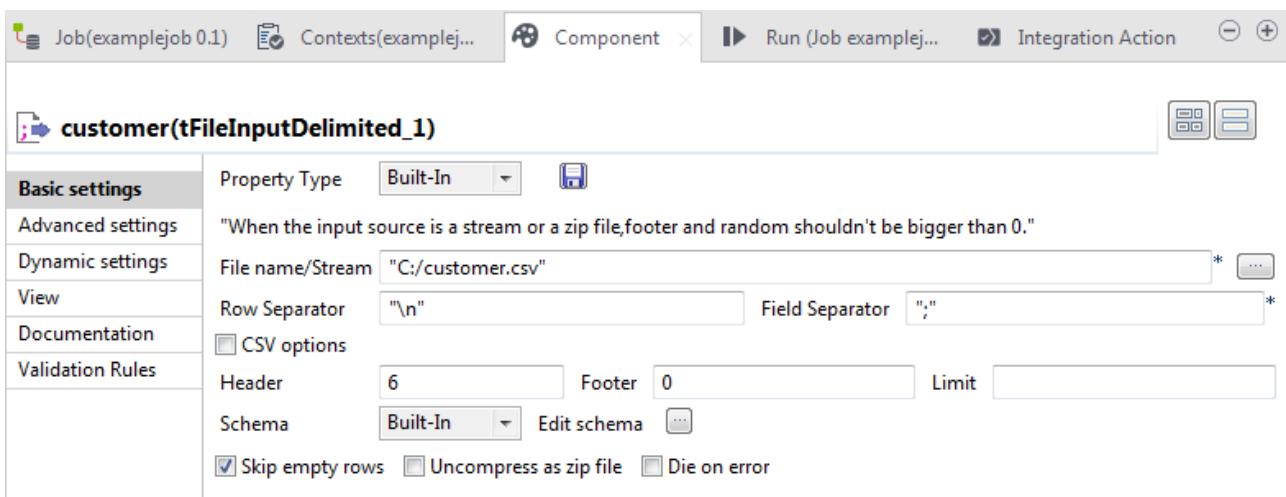
The properties information for each component in a Job allows to set the actual technical implementation of the active Job.

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.

### Basic Settings tab

#### About this task

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



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

#### Note:

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 in a Job, 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: [Centralizing database metadata](#) on page 189.

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:

#### Procedure

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 [Centralizing database metadata](#) on page 189.
3. The metadata displays under the **Metadata** node of the **Repository**.

#### Results

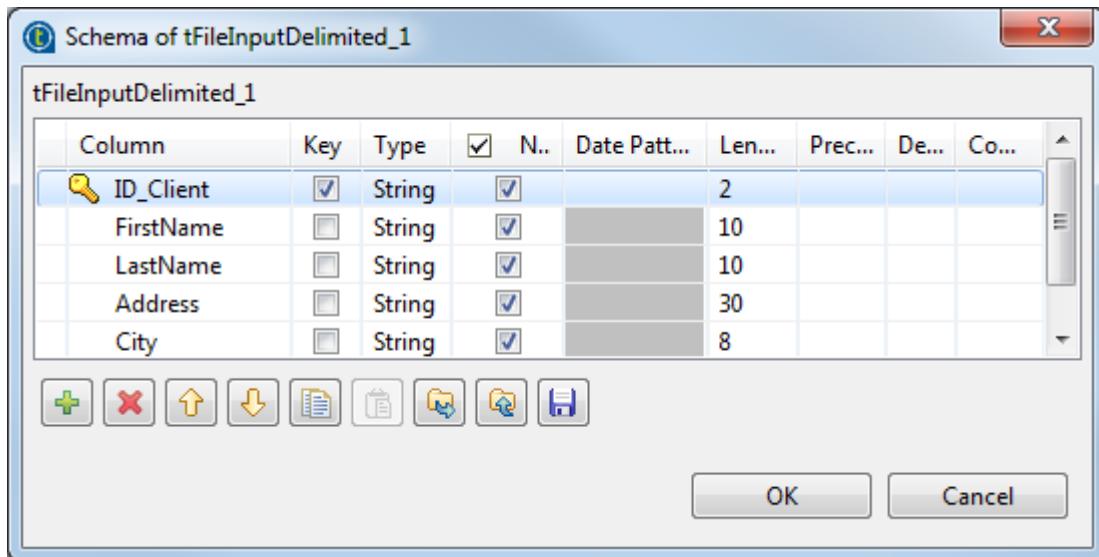
For all components that handle a data flow (most components), you can define a **Talend** schema in order to describe and possibly select the data to be processed. Like the **Properties** data, this schema is either **Built-in** or stored remotely in the **Repository** in a metadata file that you created.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

#### Setting a built-in schema in a Job

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

## Procedure

1. Select **Built-in** in the **Property Type** list of the **Basic settings** view.
2. 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.

### Warning:

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.

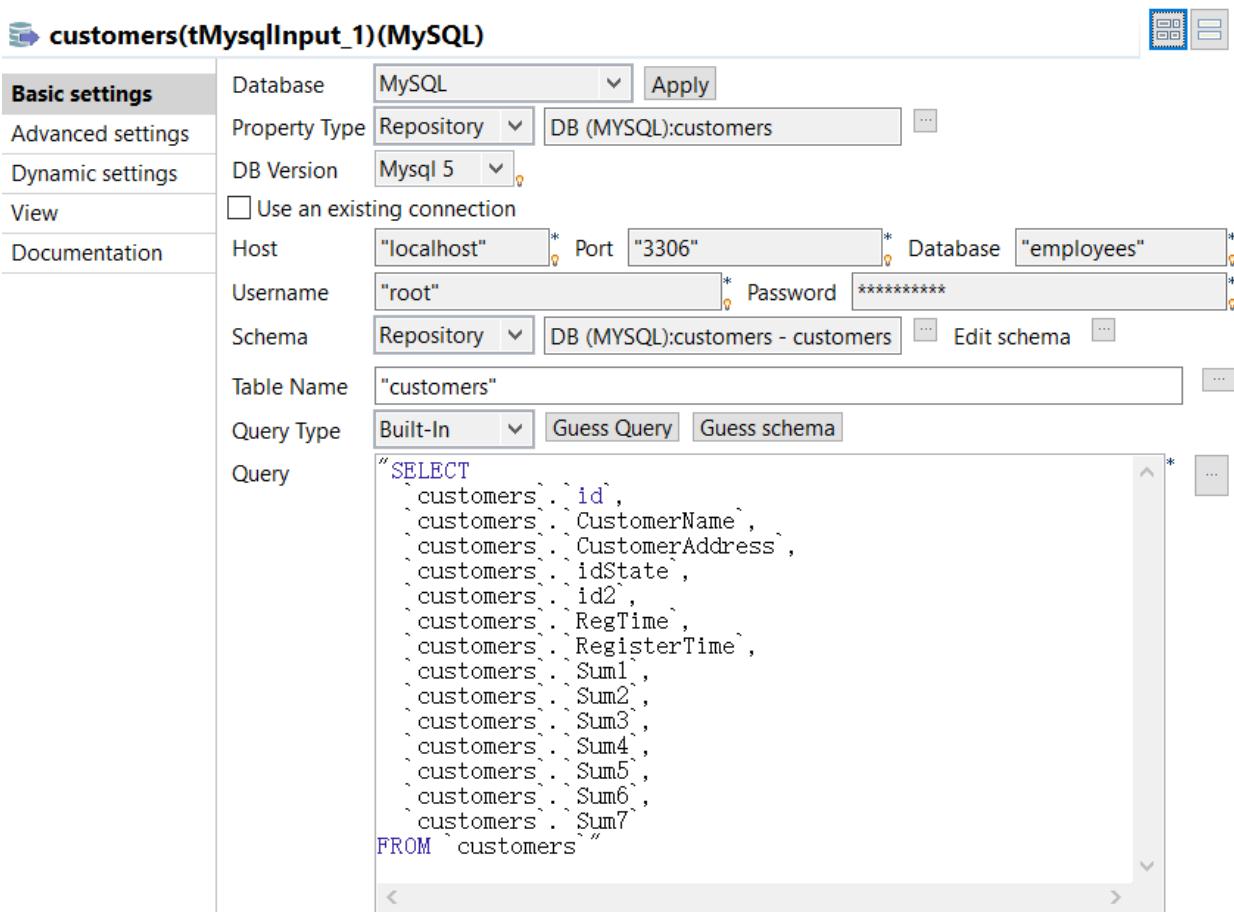
## Setting a repository schema in a Job

If you often use certain database connections or specific files when creating your 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.

## Procedure

1. 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.
2. Click **Edit Schema** to check that the data is appropriate.



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.

**Note:**

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.

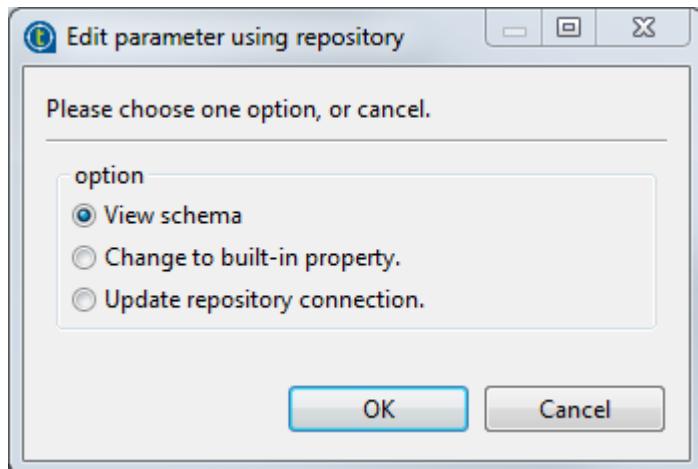
### Using a repository schema partially in a Job

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.

### Procedure

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.



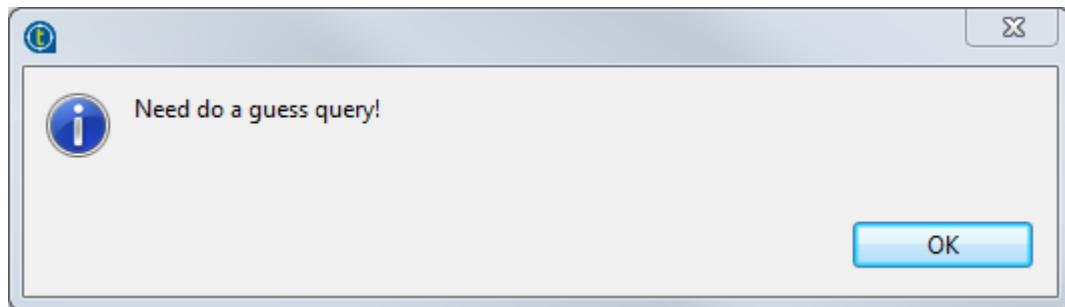
2. Click **OK**. The **Schema** dialog box pops up, which displays all columns in the schema. The **Used Column** check box before each column name indicates whether the column is used.
3. Select the columns you want to use.

Used Column	Column	Db Column	Key	Type	DB Type	Nullability	Date Pattern	Length	Precision
<input checked="" type="checkbox"/>	id	id	<input checked="" type="checkbox"/>	Int...	INT	<input checked="" type="checkbox"/>		10	0
<input checked="" type="checkbox"/>	CustomerName	CustomerName	<input type="checkbox"/>	Stri...	VARC...	<input checked="" type="checkbox"/>		30	0
<input checked="" type="checkbox"/>	CustomerAddress	CustomerAddress	<input type="checkbox"/>	Stri...	VARC...	<input checked="" type="checkbox"/>		30	0
<input checked="" type="checkbox"/>	idState	idState	<input type="checkbox"/>	Int...	INT	<input checked="" type="checkbox"/>		10	0
<input checked="" type="checkbox"/>	RegisterTime	RegisterTime	<input type="checkbox"/>	Date	DATET...	<input checked="" type="checkbox"/>	"dd-M...	19	0
<input checked="" type="checkbox"/>	Sum1	Sum1	<input type="checkbox"/>	Stri...	VARC...	<input checked="" type="checkbox"/>		5	0
<input checked="" type="checkbox"/>	Sum2	Sum2	<input type="checkbox"/>	Float	FLOAT	<input checked="" type="checkbox"/>		10	5
<input checked="" type="checkbox"/>	Sum3	Sum3	<input type="checkbox"/>	Int...	INT	<input checked="" type="checkbox"/>		10	0
<input type="checkbox"/>	Sum4	Sum4	<input type="checkbox"/>	Int...	INT	<input checked="" type="checkbox"/>		10	0
<input type="checkbox"/>	Sum5	Sum5	<input type="checkbox"/>	Float	FLOAT	<input checked="" type="checkbox"/>		10	5
<input type="checkbox"/>	Sum6	Sum6	<input type="checkbox"/>	Stri...	VARC...	<input checked="" type="checkbox"/>		11	0
<input type="checkbox"/>	Sum7	Sum7	<input type="checkbox"/>	Float	FLOAT	<input checked="" type="checkbox"/>		11	2

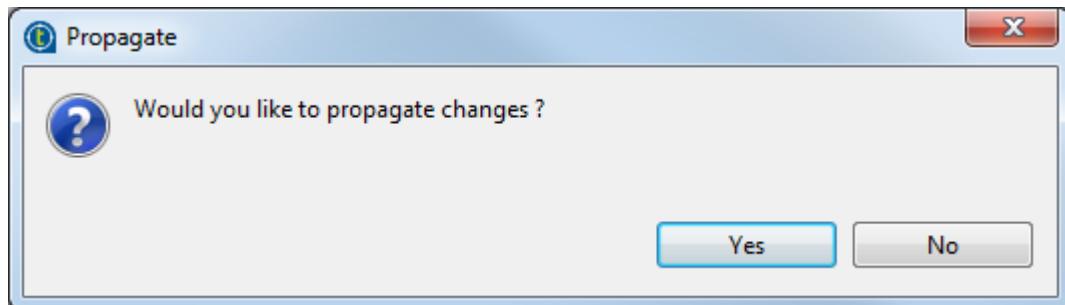
4. Click **OK**. A message box appears, which prompts you to do a guess query.

**Note:**

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.

customers(tMysqlInput_1)(MySQL)	
<b>Basic settings</b>	Database: MySQL Property Type: Repository (DB (MYSQL):customers) DB Version: Mysql 5 <input type="checkbox"/> Use an existing connection Host: "localhost" * Port: "3306" * Database: "employees" * Username: "root" * Password: ***** Schema: Repository (DB (MYSQL):customers - customers) Edit schema Table Name: "customers" Query Type: Built-In (Guess Query) Guess schema <pre>"SELECT     `customers`.`id`,     `customers`.`CustomerName`,     `customers`.`CustomerAddress`,     `customers`.`idState`,     `customers`.`id2`,     `customers`.`RegTime`,     `customers`.`RegisterTime`,     `customers`.`Sum1`,     `customers`.`Sum2`,     `customers`.`Sum3`,     `customers`.`Sum4`,     `customers`.`Sum5`,     `customers`.`Sum6`,     `customers`.`Sum7` FROM `customers`"</pre>
Advanced settings	
Dynamic settings	
View	
Documentation	

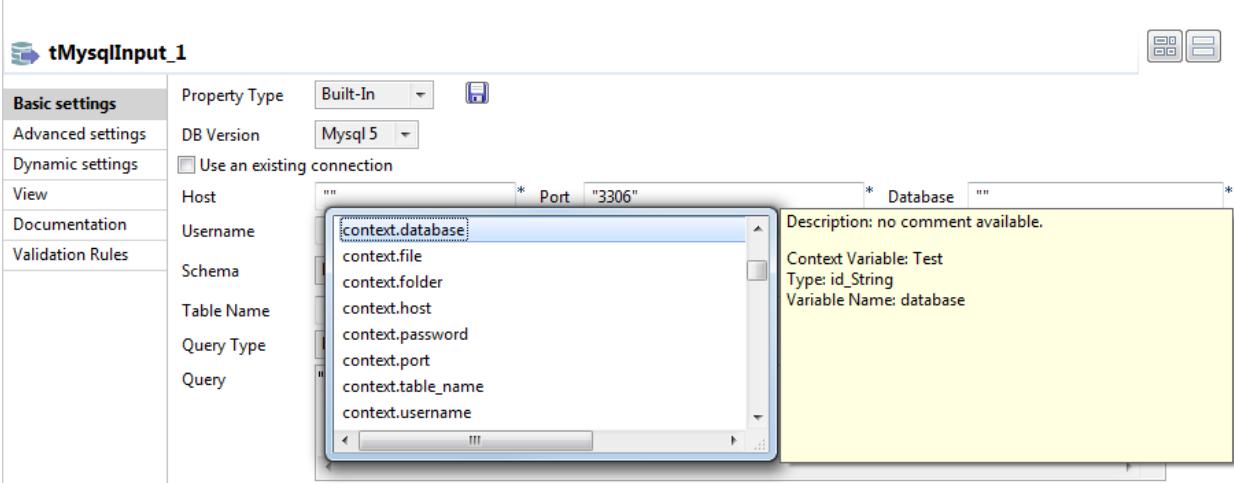
## Setting a field dynamically (Ctrl+Space bar)

### About this task

On any field of your 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.

### Procedure

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.

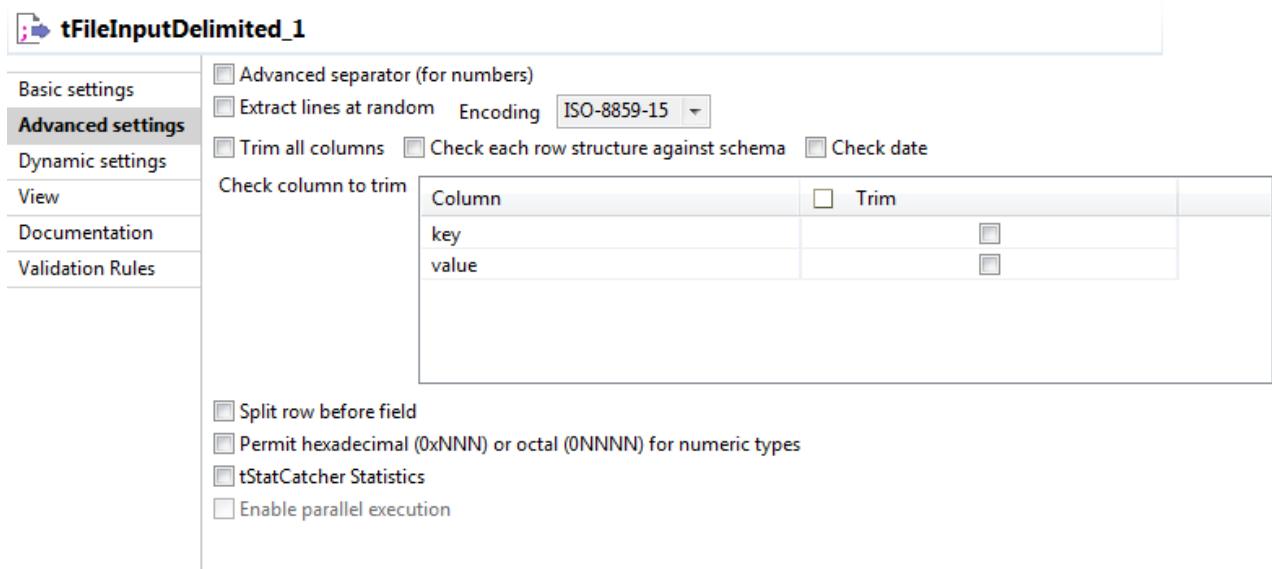


This can be any parameter including: error messages, number of lines processed, or else... The list varies according to the component in selection or the context you are working in.

Related topic: [Using contexts and variables](#) on page 66.

### Advanced settings tab

Some components, especially **File** and **Databases** components in Jobs, 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.

### Measuring 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 [Automating the use of statistics & logs](#) on page 94.

### Dynamic settings tab of components in a Job

#### About this task

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.

Name	Code
Print operations	context.verbose

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

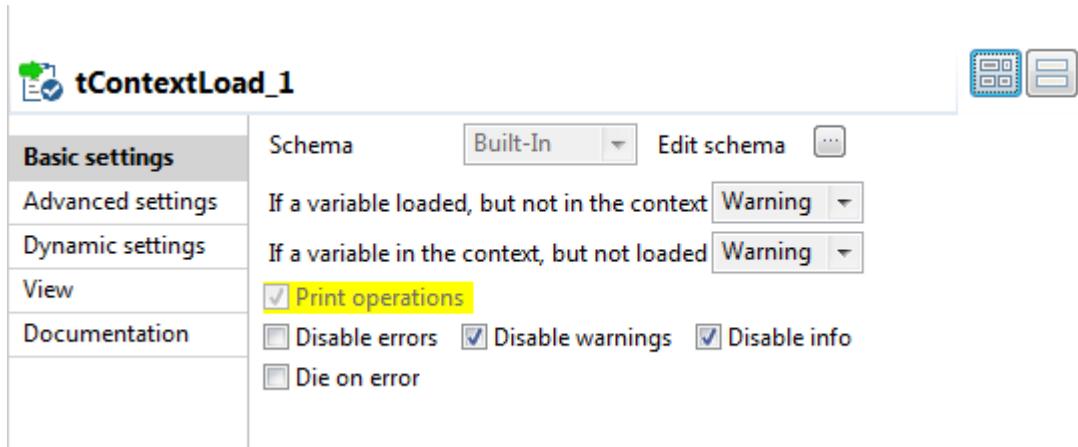
#### Procedure

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`.

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

## Results

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



**Note:** 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 [Defining context variables in the Contexts view on page 67](#).

For use cases showing how to define a dynamic parameter, see [Defining Context Groups](#).

## 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: <b>YourLabelOrHint</b>
- Italic: <i>YourLabelOrHint</i>
- Return carriage: YourLabelOrHint <br> ContdOnNextLine
- Color: <Font color='#RGBcolor'> YourLabelOrHint </Font>

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

## 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.

### Finding Jobs containing a specific component

#### About this task

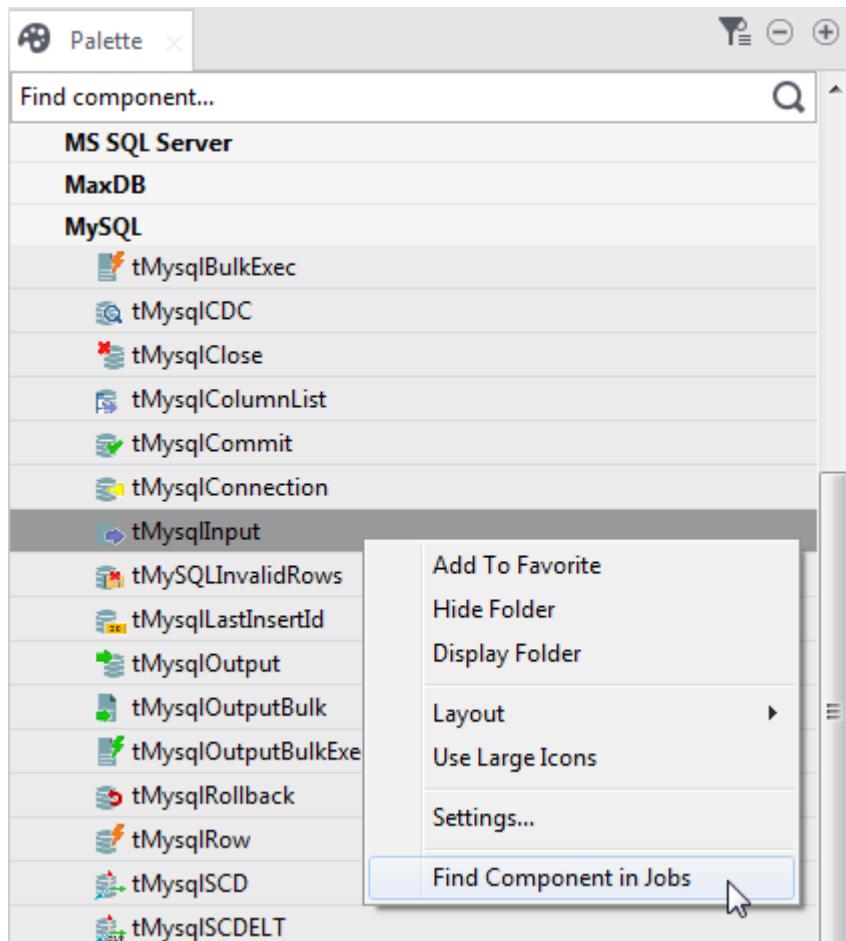
##### Note:

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.

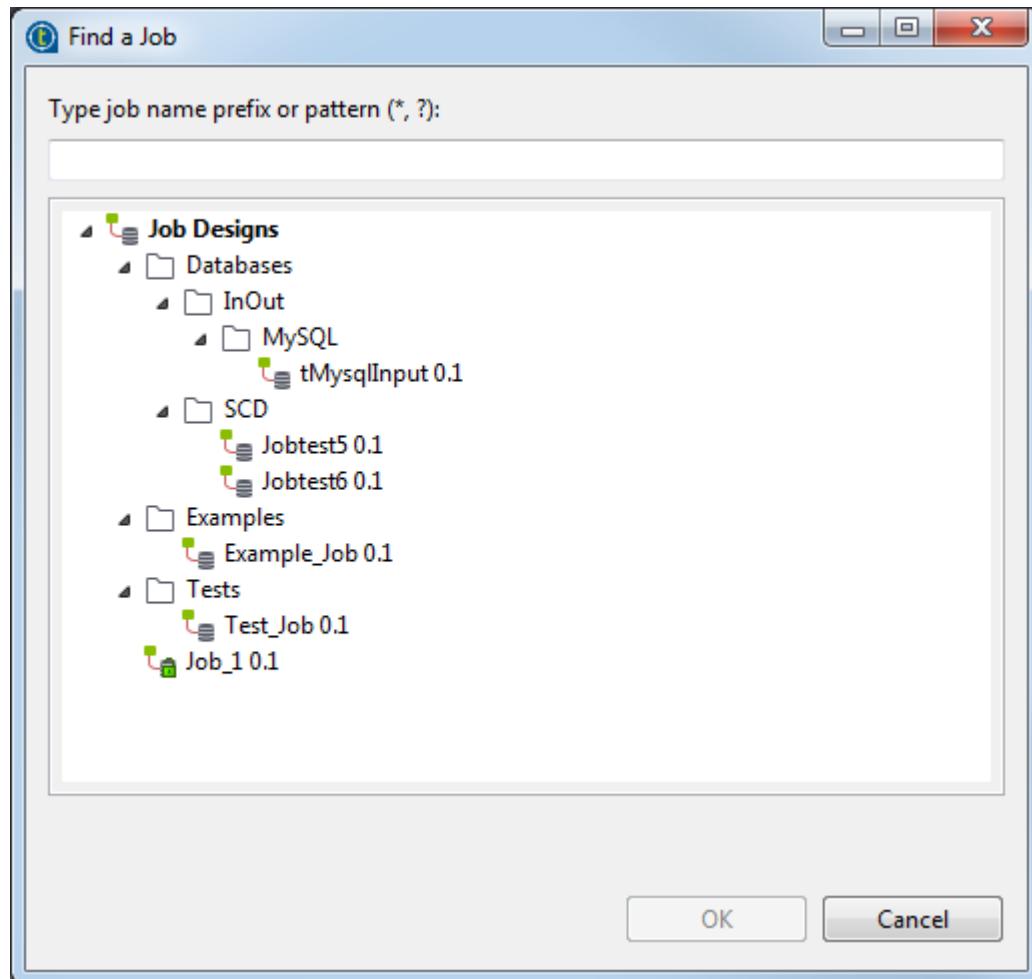
From the **Palette**, you can search for all the Jobs that use the selected component. To do so:

#### Procedure

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.

### Setting default values in the schema of a component in a Job

#### About this task

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

#### Note:

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

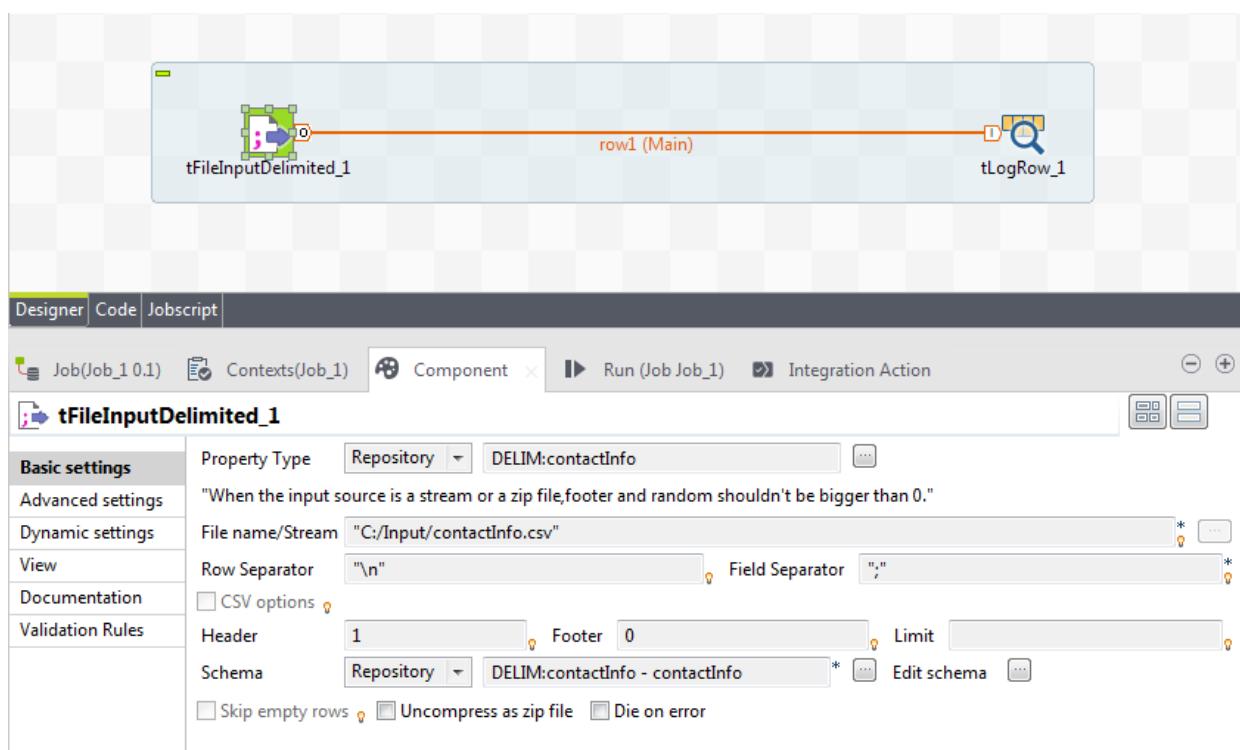
In the following example, the `company` and `city` fields of some records of the source CSV file are left blank, as shown below. The input component reads data from the source file and completes the missing information using the default values set in the schema, `Talend` and `Paris` respectively.

```
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:

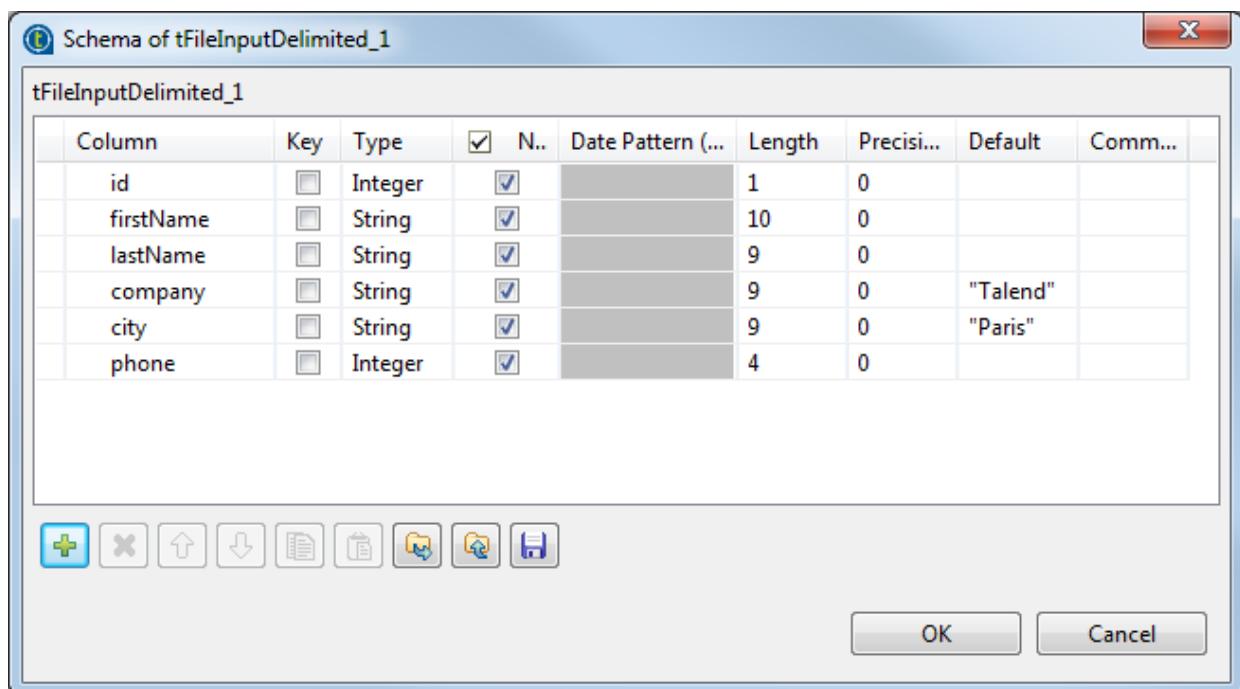
## Procedure

- 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 [Centralizing database metadata](#) on page 189.

- 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.
- 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.

```

Starting job Job_1 at 15:57 09/08/2015.

[statistics] connecting to socket on port 3454
[statistics] connected
+-----+-----+-----+-----+
|          tLogRow_1                         |
+-----+-----+-----+-----+
| id | firstName | lastName | company | city   | phone  |
+-----+-----+-----+-----+
| 1  | Michael   | Jackson  | IBM     | Roma   | 2323  |
| 2  | Elisa     | Black    | Microsoft | London | 4499  |
| 3  | Michael   | Dujardin | Talend  | Paris   | 8872  |
| 4  | Marie     | Dolvina  | Talend  | Paris   | 6655  |
| 5  | Jean      | Perfide  | Talend  | Paris   | 3344  |
| 6  | Emilie    | Taldor   | Oracle   | Madrid  | 2266  |
| 7  | Anne-Laure | Paldufier | Apple   | Paris   | 4422  |
+-----+-----+-----+-----+
[statistics] disconnected
Job Job_1 ended at 15:57 19/08/2015. [exit code=0]

```

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

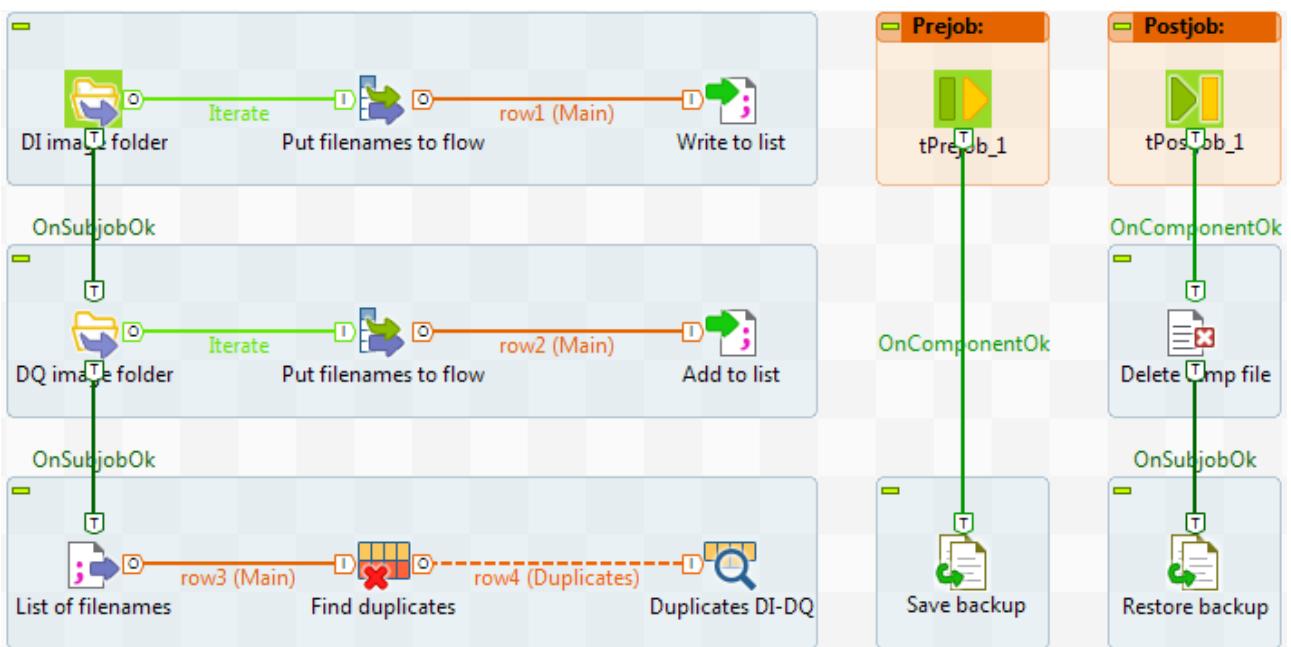
### Using the **tPrejob** and **tPostjob** components

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.

**Note:**

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 [Orchestration \(Integration\)](#)

### Downloading/uploading 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.

**Warning:** Make sure that the `-Dtalend.disable.internet` parameter is not present in the Studio .ini file or is set to false.

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.

**Note:**

- 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 \(Talend > Exchange\)](#) on page 410.
- 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.

**Installing community components from Talend Exchange****About this task**

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

**Procedure**

- 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>

- 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.
- Click the **view/download** link for the component of interest to display the component download page.

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

Version 1.1  
2011-05-17

**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.

- 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 [Reviewing and rate a community component](#) on page 56.

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

- 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

## Reinstalling or update community components

### About this task

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.

#### Note:

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 \(Talend > Exchange\)](#) on page 410.

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

### Procedure

- 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.
- 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**.

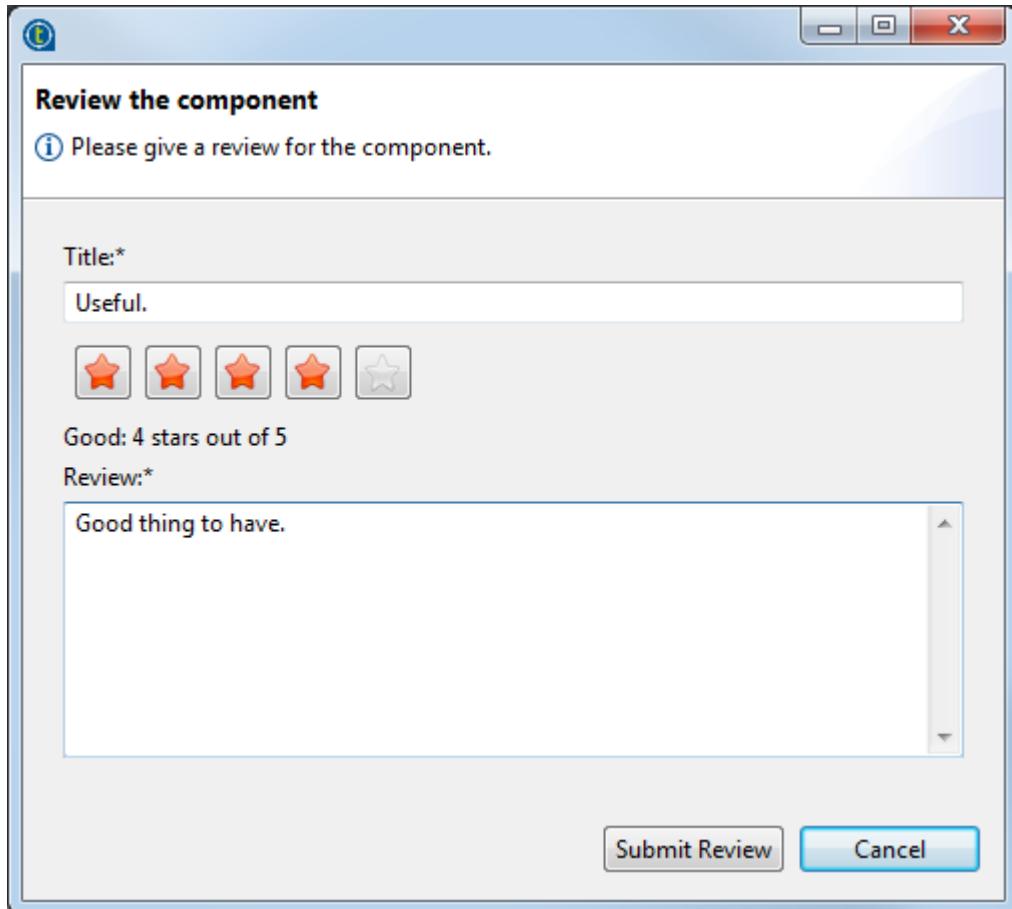
## Reviewing and rate a community component

### About this task

To review and rate a community component:

## Procedure

- 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.
- 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.

## Uploading a component you created to Talend Exchange

### About this task

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 \(Talend > Components\)](#) on page 408.

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

## Procedure

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

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

2. 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: [text input]
- Versions and newer: 5.0
- All versions except: [text input]
- Only these versions: [text input]

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**.

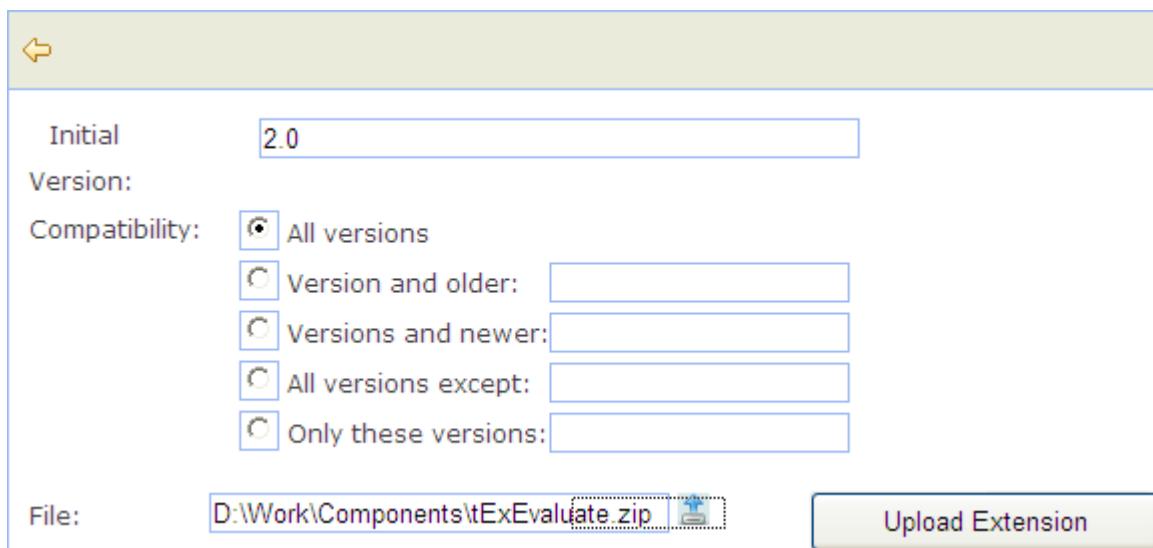
Available Extensions				Add New Extension
Downloaded Extensions	Extension Name	Version	Upload Date	Operation
My Extensions				
	tExEvaluate	1.0	2011-11-18	

### Managing 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.



Initial Version: 2.0

Version:

Compatibility:

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

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

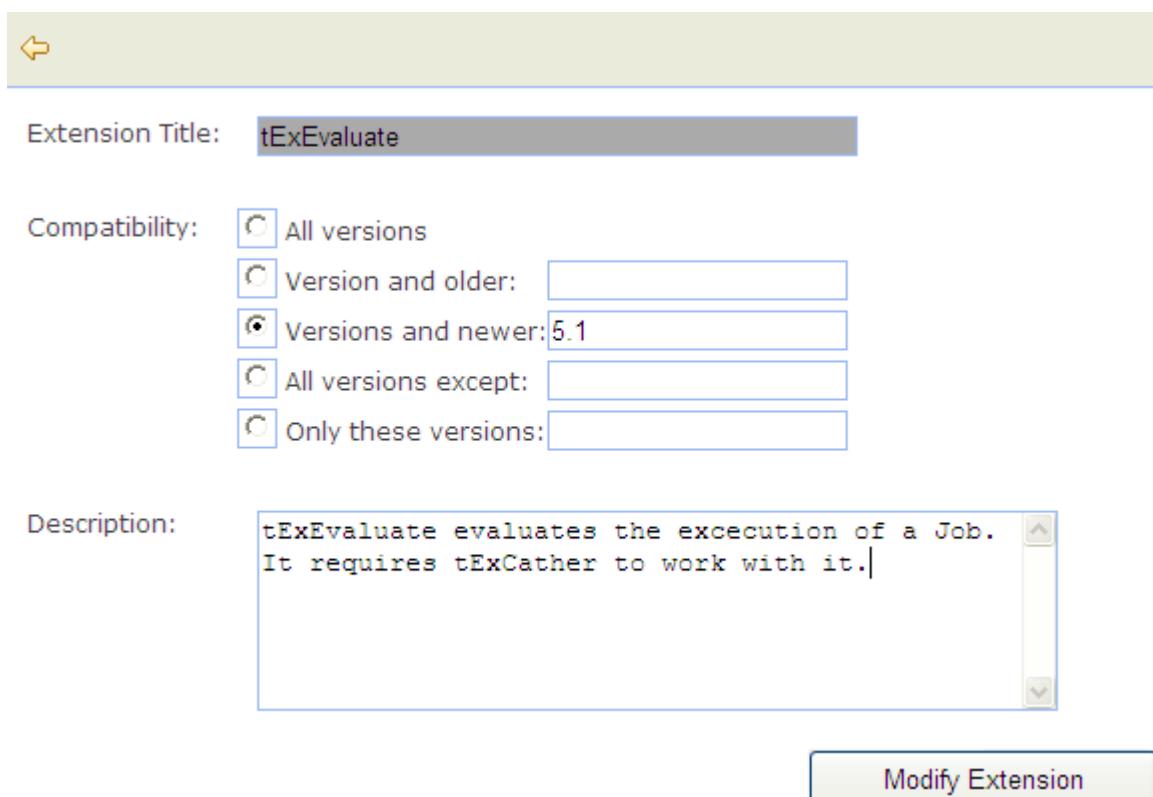
**Upload Extension**

- 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.



Extension Title: tExEvaluate

Compatibility:

- All versions
- Version and older: [ ]
- Versions and newer: 5.1
- All versions except: [ ]
- Only these versions: [ ]

Description:

```
tExEvaluate evaluates the execution of a Job.  
It requires tExCather to work with it.
```

**Modify Extension**

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.

## Using connections in a Job

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.

### 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.

#### 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.

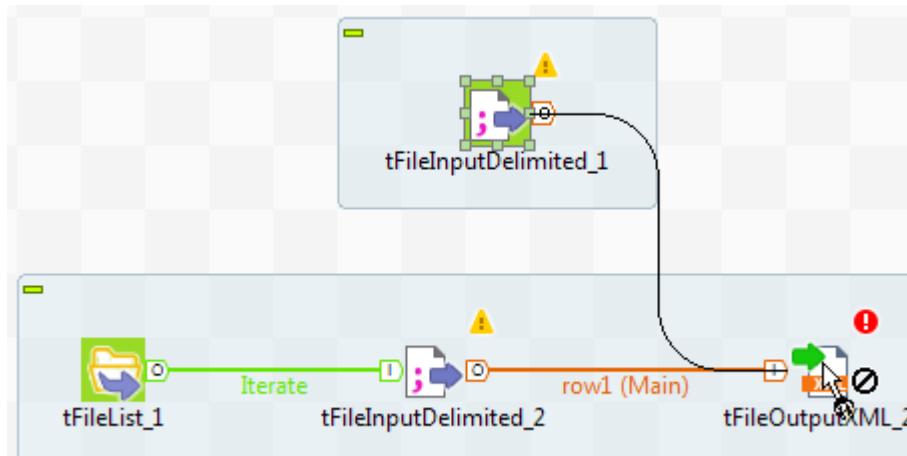
#### Main

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.

#### Note:

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**.



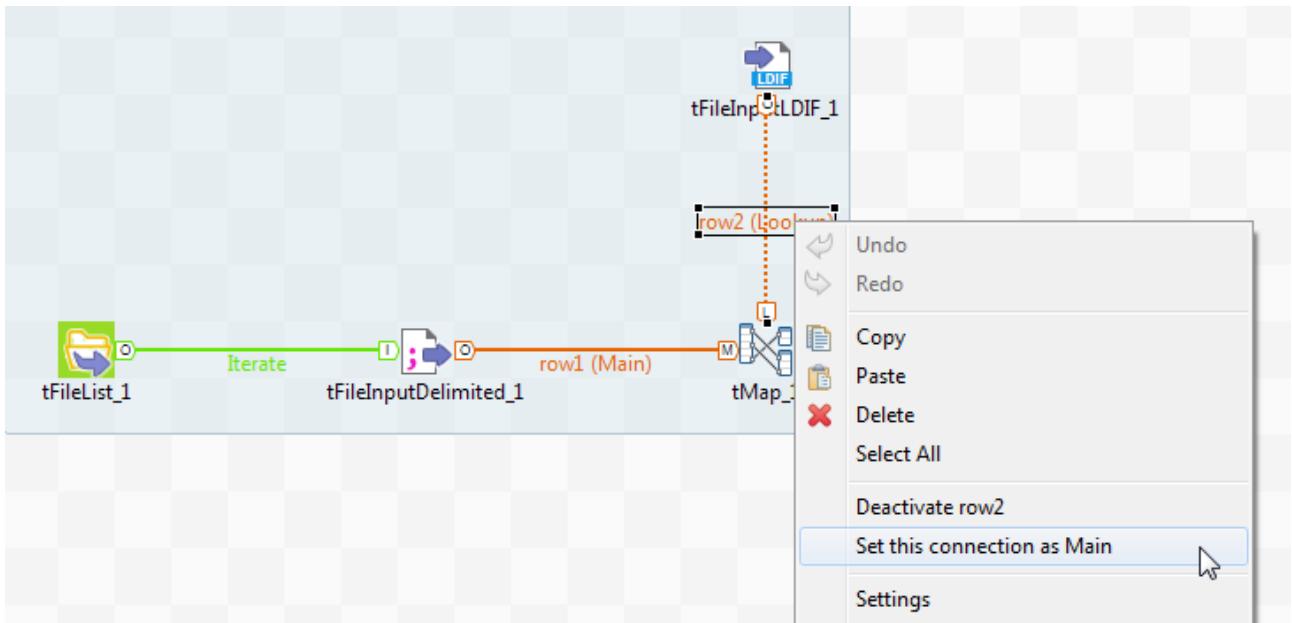
To connect two components using a Main connection, right-click the input component and select **Row > Main** on the connection list.

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](#) on page 62.

### 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 conversely, 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](#) on page 62.

### 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](#) on page 155.

## 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.

### Note:

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](#) on page 62.

## 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 [Map editor interfaces](#) on page 132.

## 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**.

## 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 **tFileList** 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](#) on page 65.

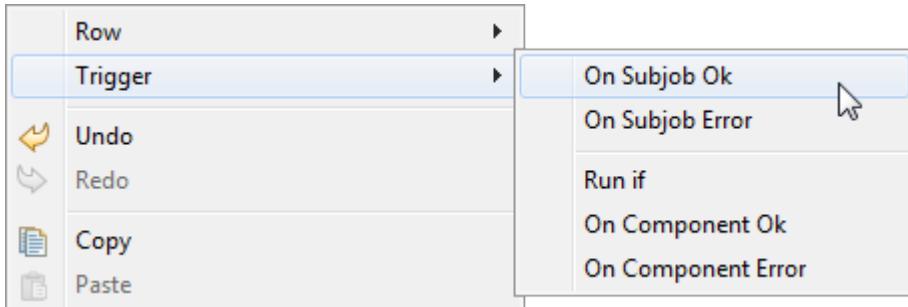
**Note:** The name of the **Iterate** connection is read-only unlike other types of connections.

**Warning:** Note that `globalMap` is thread unsafe. Be cautious when using `globalMap.put("key", "value")` and `globalMap.get("key")` to create your own global variables and then retrieve their values in your Jobs, especially after an **Iterate** connection with the parallel execution option enabled.

## Trigger connections

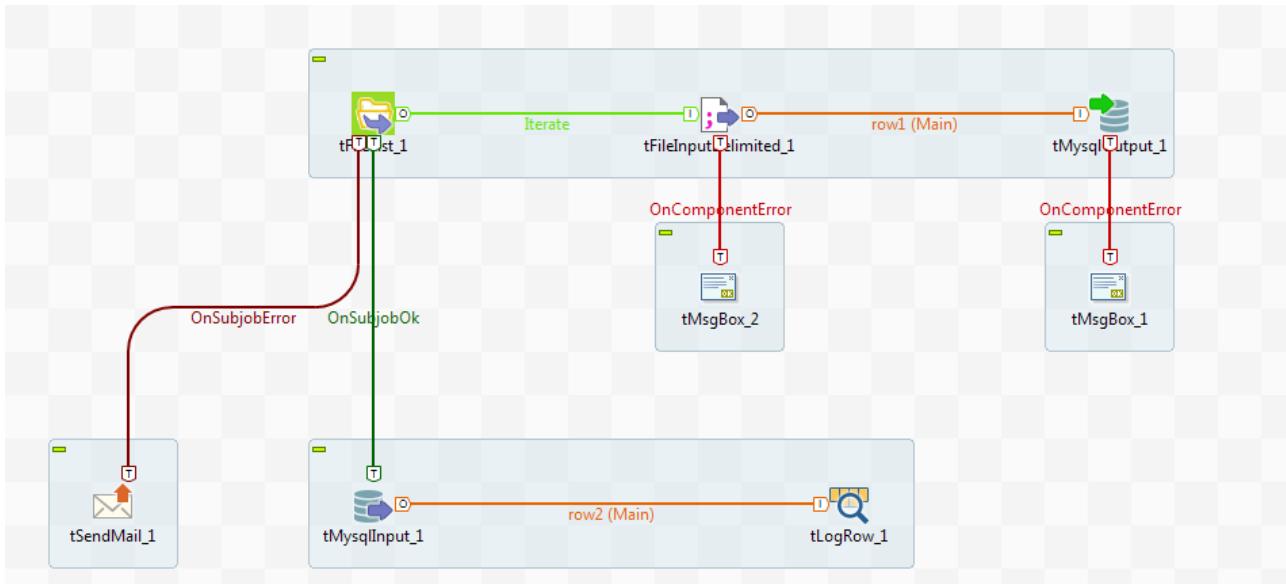
Trigger connections define the processing sequence, so no data is handled through these connections.

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.

**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 subJob 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](#) on page 65.

For how to set a trigger condition, see [Trigger connection settings](#) on page 65.

## 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**.

### Warning:

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.

## Defining 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.

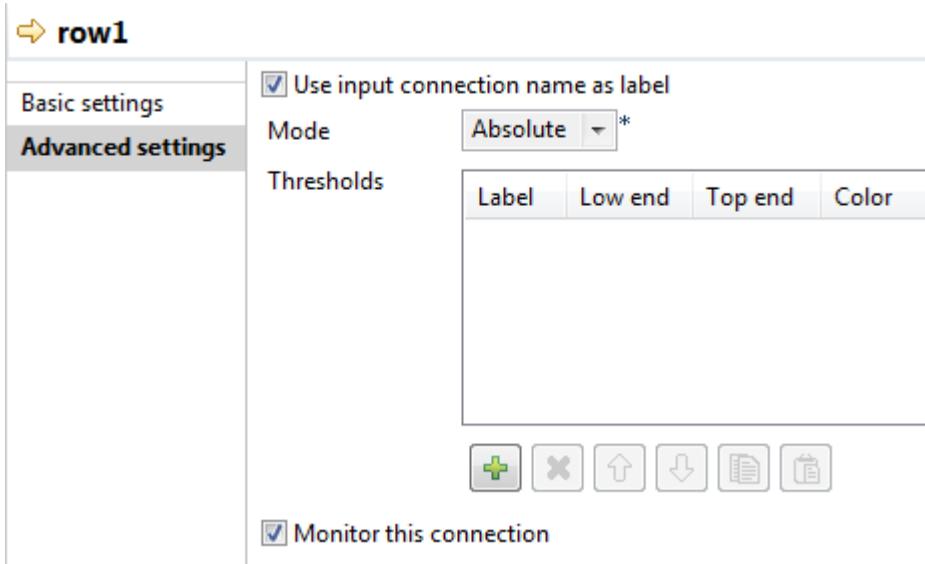
### Row connection settings

#### About this task

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 [Setting a built-in schema in a Job](#) on page 40.

Column	Key	Type	N.	Date Patter...	Length	Preci...	Def...	Com...
<input checked="" type="checkbox"/> 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.



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

### Procedure

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

### Iterate connection settings

When you configure an Iterate connection, you are actually enabling parallel iterations. For further information, see [Launching parallel iterations to read data on page 132](#).

### Trigger connection settings

[Run if connection settings](#)

### About this task

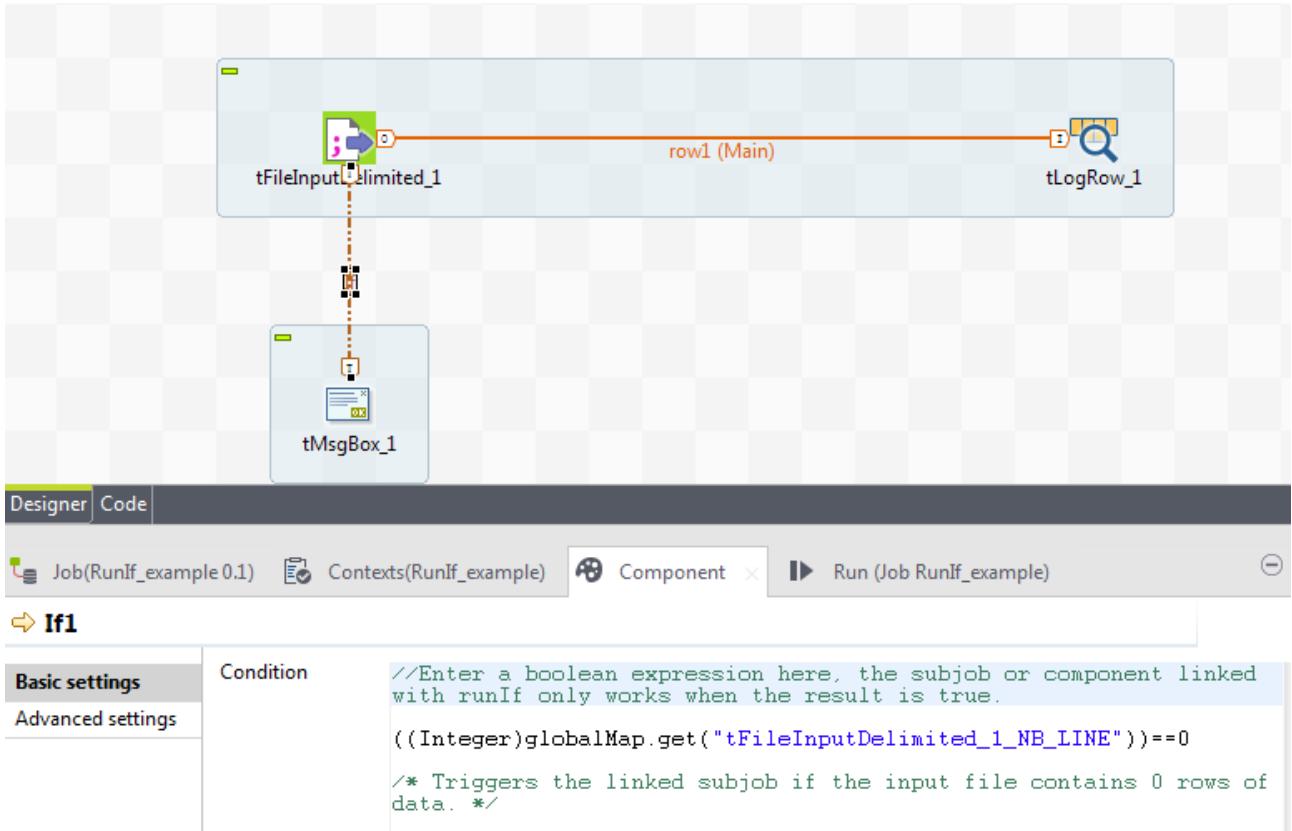
In the **Basic settings** view of a **Run if** connection, you can set the condition to the subJob in Java.

You can use variables in your condition. Pressing **Ctrl+Space** allows you to access all global and context variables. For more information, see [Using variables in a Job](#) on page 80.

#### Warning:

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.



## Procedure

1. 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.

## 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.

**Warning:** Note that `globalMap` is thread unsafe. Be cautious when using `globalMap.put("key", "value")` and `globalMap.get("key")` to create your own global variables and then retrieve their values in your Jobs, especially after an **Iterate** connection with the parallel execution option enabled.

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.

For a Job, 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 [Context](#).

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

### Defining context variables for a Job

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

- Using the **Contexts** view of the Job.
- Using the **F5** key from the **Component** view of a component.

### Defining context variables in the Contexts view

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

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

	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"/>

Default context environment **Test**

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.

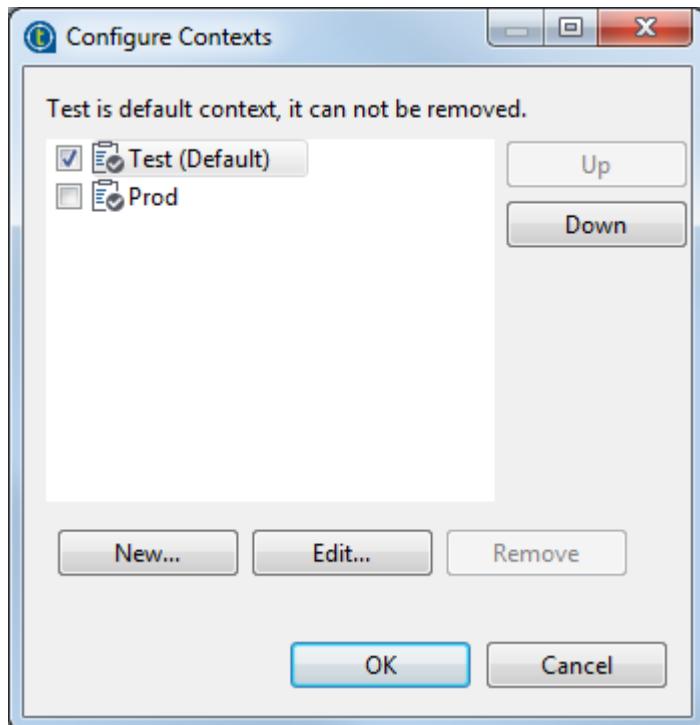
## Defining contexts

### Procedure

1. Open the Job in the design workspace and go to the **Contexts** view.  
If the **Contexts** view is not displayed, select **Window > Show view > TalendContexts** to open the **Contexts** view in the **Integration** perspective.
2. Click the **[+]** button at the top right corner.  
The **Configure Contexts** dialog box opens and a context named **Default** is created by default.
3. Select the default context and click **Edit** to rename it, **Prod** in this example. Click **OK**.
4. In the open dialog box, click **New...** and enter **Test** in the **New Context** dialog box, click **OK**.
5. 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 in the **Contexts** tab view.  
If needed, move a context up or down by selecting it and clicking the **Up** or **Down** button.

### Example

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



- Click **OK** to validate your context definition and close the dialog box.

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

- Repeat the above steps to create as many new contexts as needed.

If you do not want to define the values of each new context from scratch, you can create the first context and define all its values, as when you create a new one all the parameters of the context selected as default are copied to the new context. You can then modify the values of the new context as needed.

## Defining variables

### Procedure

- Click the **[+]** button at the bottom of the **Contexts** view to add lines in the table.

	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?

- Click in the **Name** field and enter the name of the variable you are creating.  
Name the first variable **host** for this example.
- From the **Type** list, select the type of the variable.
- If needed, click in the **Comment** field and enter a comment to describe the variable.
- 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.	

**Warning:** 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 above steps to define all the variables for the different contexts.

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

## Results

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 [Adding a built-in context variable to the Repository](#) on page 77.

Related topics:

- [Defining variables from the Component view](#) on page 71
- [Centralizing context variables in the Repository](#) on page 72
- [Using variables in a Job](#) on page 80
- [Running a Job in a selected context](#) on page 81

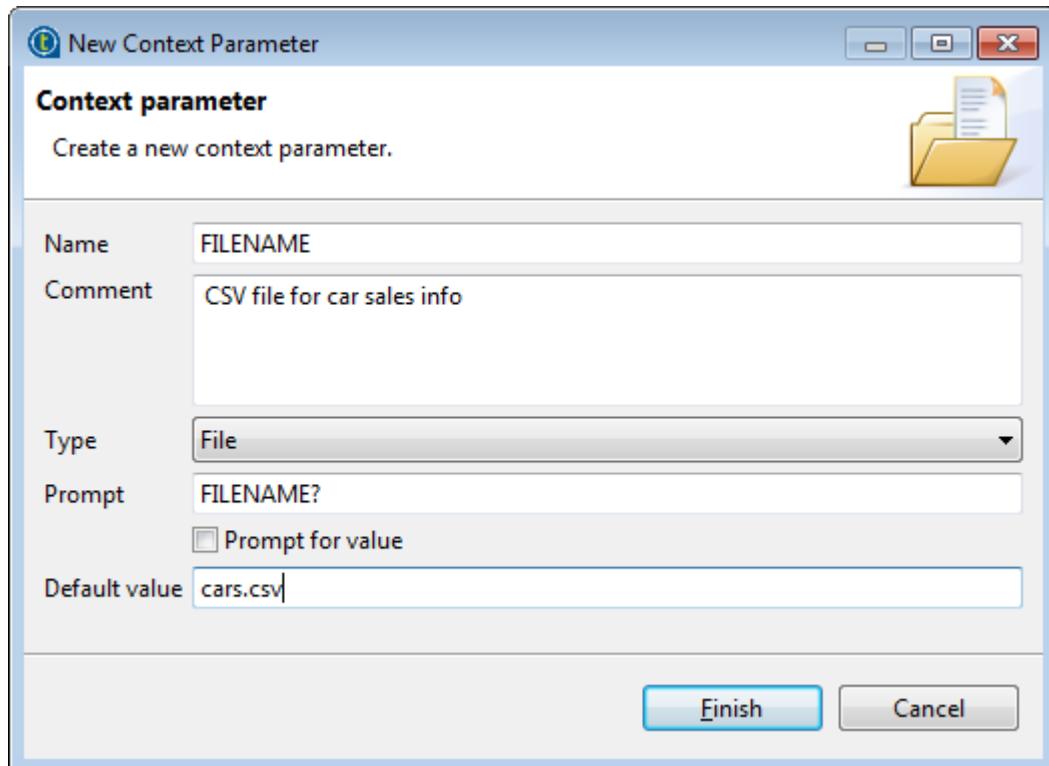
## Defining variables from the Component view

### About this task

The quickest way to create a single context variable is to use the **F5** key from the **Component** view. The following example demonstrates how to create a context variable while configuring a file path for a component in a Job.

### Procedure

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**.

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

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.

### Results

The newly created variables are listed in the **Contexts** view. 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](#) on page 68.

## Centralizing 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 [Creating a context group and define context variables in it](#) on page 72 for details.
- Adding a built-in context variable to an existing or new context group in the Repository. See [Adding a built-in context variable to the Repository](#) on page 77 for details.
- Saving a context from metadata. See [Creating a context from a Metadata](#) on page 78 for more information.

### Creating a context group and define context variables in it

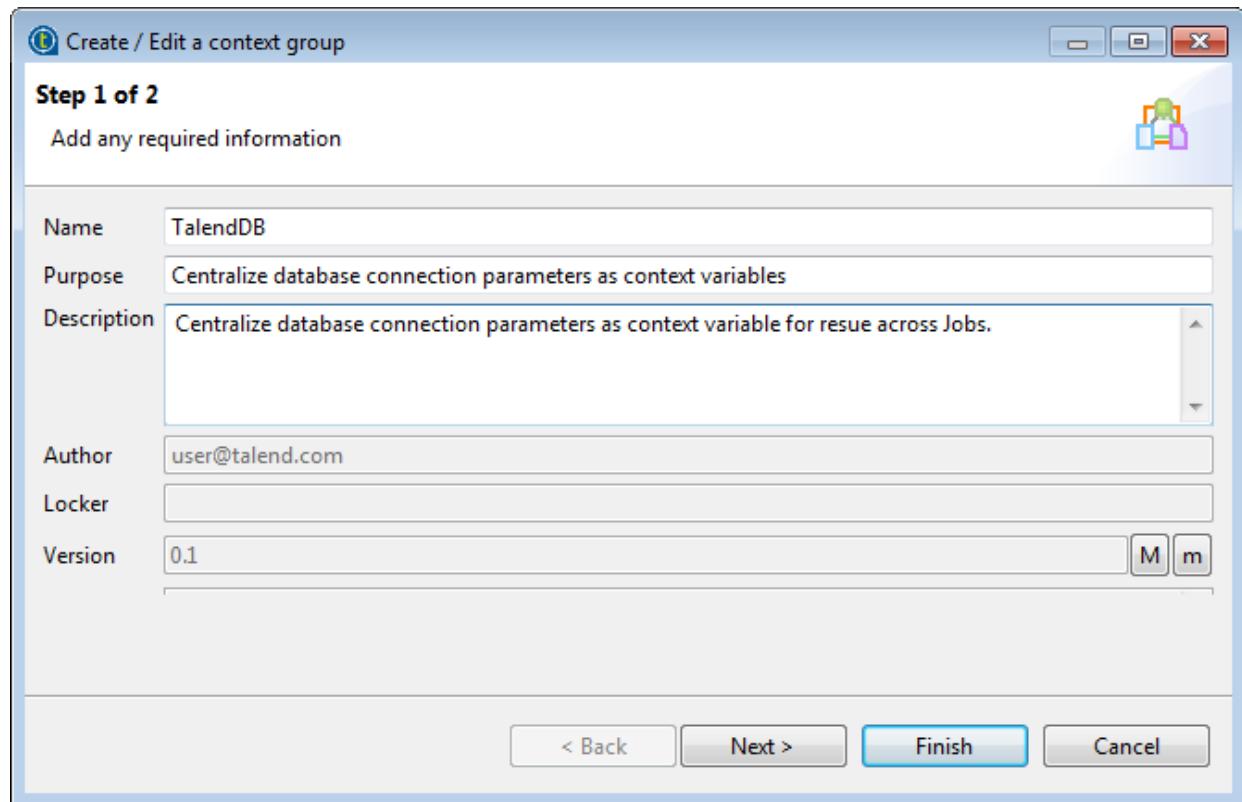
The following example will demonstrate how to use the **Create / Edit a context group** wizard to create a context group named `TalendDB` that contains two contexts named `Prod` and `Test` and define a set of variables - `host`, `port`, `database`, `username`, `password`, and `table_name` - under the two contexts in the Repository, for reuse in database handling Jobs.

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.

#### Creating the context group and contexts

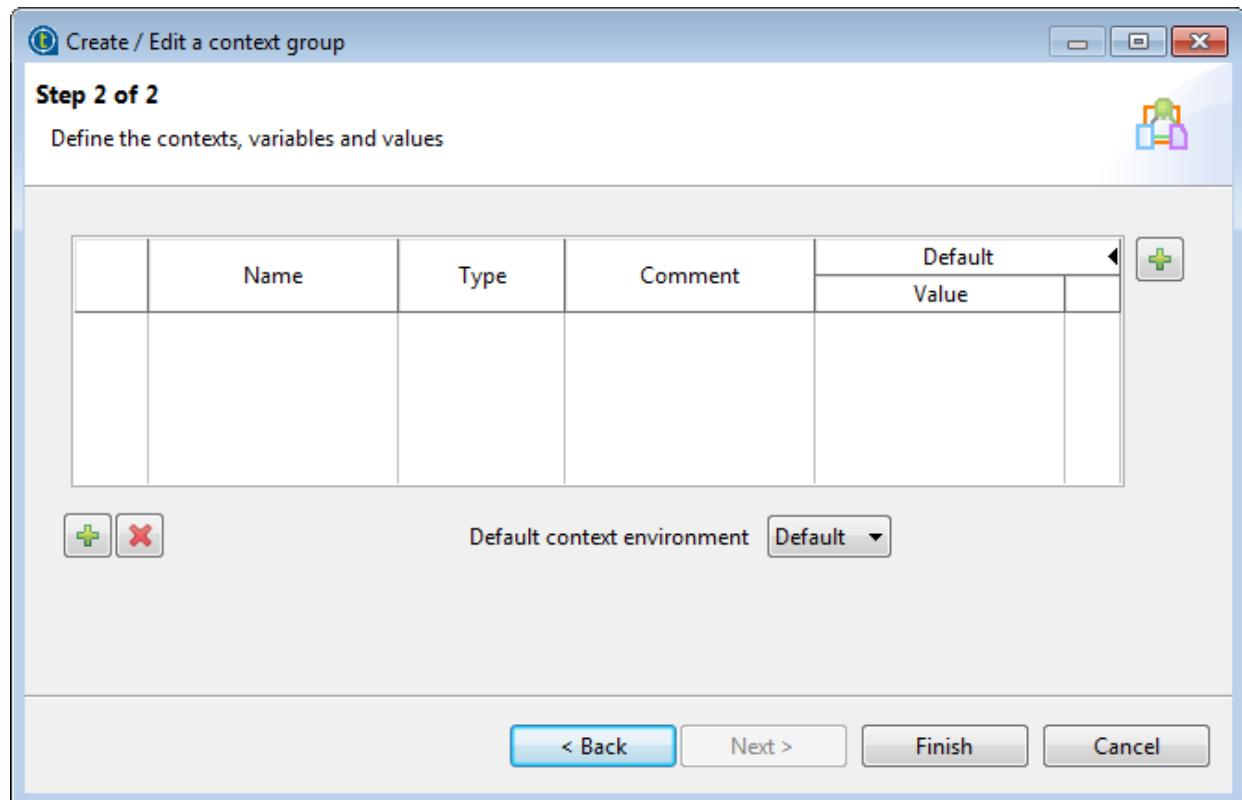
##### Procedure

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.

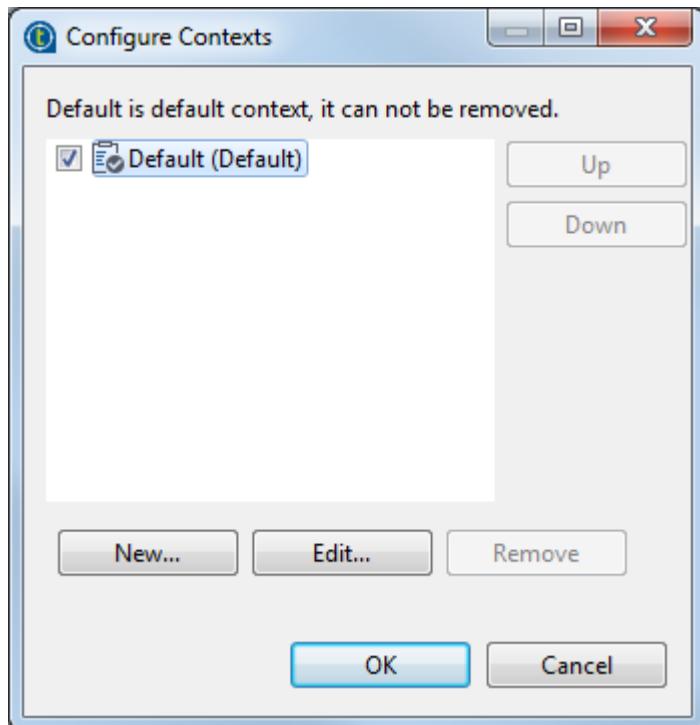


3. Click **Next** to go to Step 2 of 2, which allows you to define the various contexts and variables that you need.

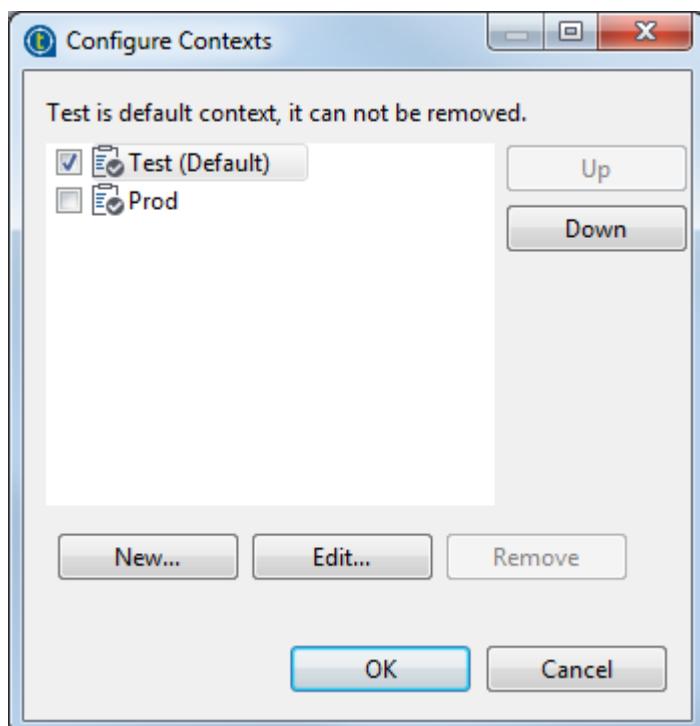
A context named Default has been created and set as the default one by the system.



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

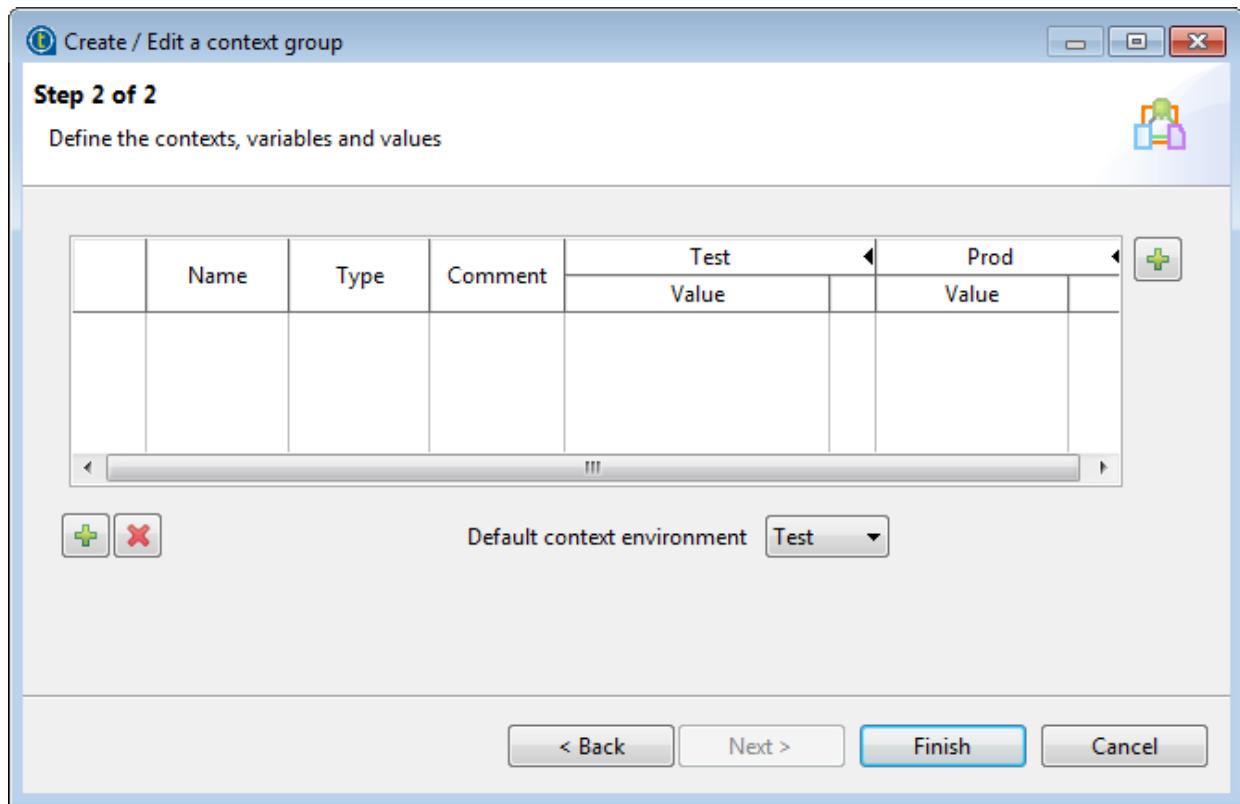


5. Select the context Default, click the **Edit...** button and enter Prod in the **Rename Context** dialog box that opens to rename the context Default to Prod.  
Then click **OK** to close the dialog box.
6. Click the **New...** button and enter Test in the **New Context** dialog box. Then click **OK** 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.



8. 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.



## Defining context variables

### Procedure

- 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.
- 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.
- If needed, click in the **Comment** field and enter a comment to describe the variable.
- Click in **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.	

Type	Value field	Default value
Directory	Editable text field, with a button to open the <b>Browse for Folder</b> dialog box for folder selection.	
List of Value	Editable text field, with a button to open the <b>Configure Values</b> dialog box for list creation and configuration.	(Empty)
Password	Editable text field; text entered appears encrypted.	

**Warning:** 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	
host	String		"localhost"	<input type="checkbox"/>	host?	"192.168.30.110"	<input type="checkbox"/>
port	String		"3306"	<input type="checkbox"/>	port?	"3308"	<input type="checkbox"/>
database	String		"test"	<input type="checkbox"/>	database?	"prod_db"	<input type="checkbox"/>
username	String		"root"	<input type="checkbox"/>	username?	"prod_user"	<input type="checkbox"/>
password	Password		*****	<input type="checkbox"/>	password?	*****	<input type="checkbox"/>
table_name	String		"testtable"	<input checked="" type="checkbox"/>	table_name?	"addresses"	<input type="checkbox"/>

Default context environment **Test**

< Back Next > Finish Cancel

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. Through preference configuration, you can enable or disable propagation of variable changes to your Jobs. For more information, see [Performance preferences \(Talend > Performance\)](#) on page 413.

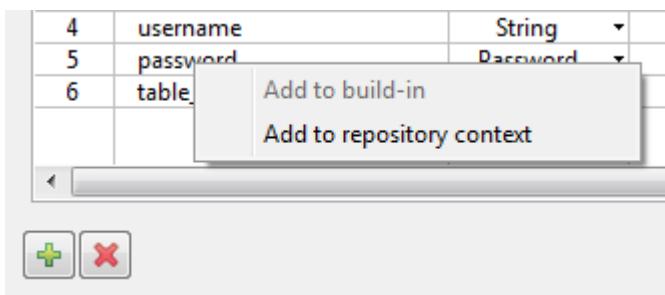
## Adding a built-in context variable to the Repository

### About this task

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.

### Procedure

- 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.

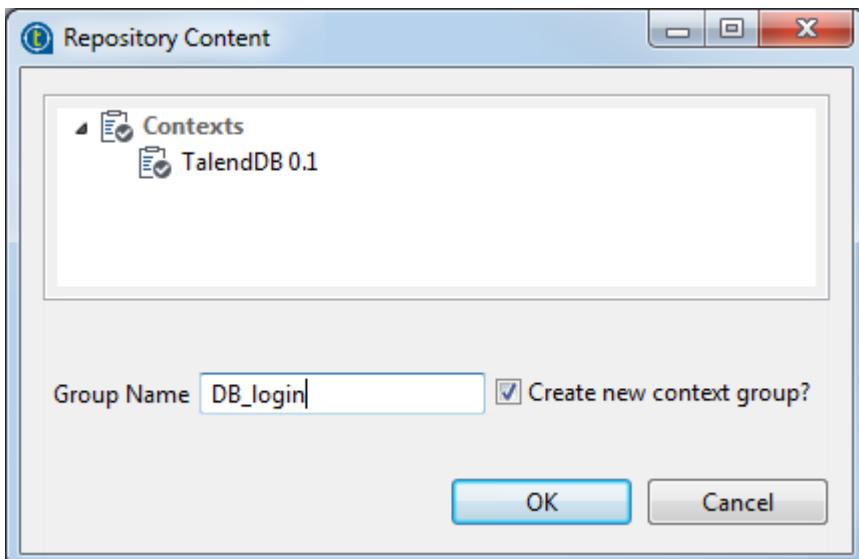


- In the dialog box, do either of the following:

- 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**.

**Warning:** 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 context variable `password` defined in a Job 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"	<input checked="" type="checkbox"/>	host?
2	port	String		"3306"	<input type="checkbox"/>	port?	"3308"	<input type="checkbox"/>	port?
3	database	String		"test"	<input type="checkbox"/>	database?	"prod_db"	<input type="checkbox"/>	database?
4	username	String		"root"	<input type="checkbox"/>	username?	"prod_user"	<input type="checkbox"/>	username?
5	DB.login (from repository)			*****	<input type="checkbox"/>	password?	*****	<input type="checkbox"/>	password?
6	password	Password		"testtable"	<input checked="" type="checkbox"/>	table_name?	"addresses"	<input type="checkbox"/>	table_name?
7	table_name	String							

## Creating 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](#) on page 334.

## Applying 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 [Dropping a context group onto a Job](#) on page 78 for details.
- Use the button. This way, the variables of a context group can be applied separately. See [Applying context variables to a Job using the context button](#) on page 79 for details.

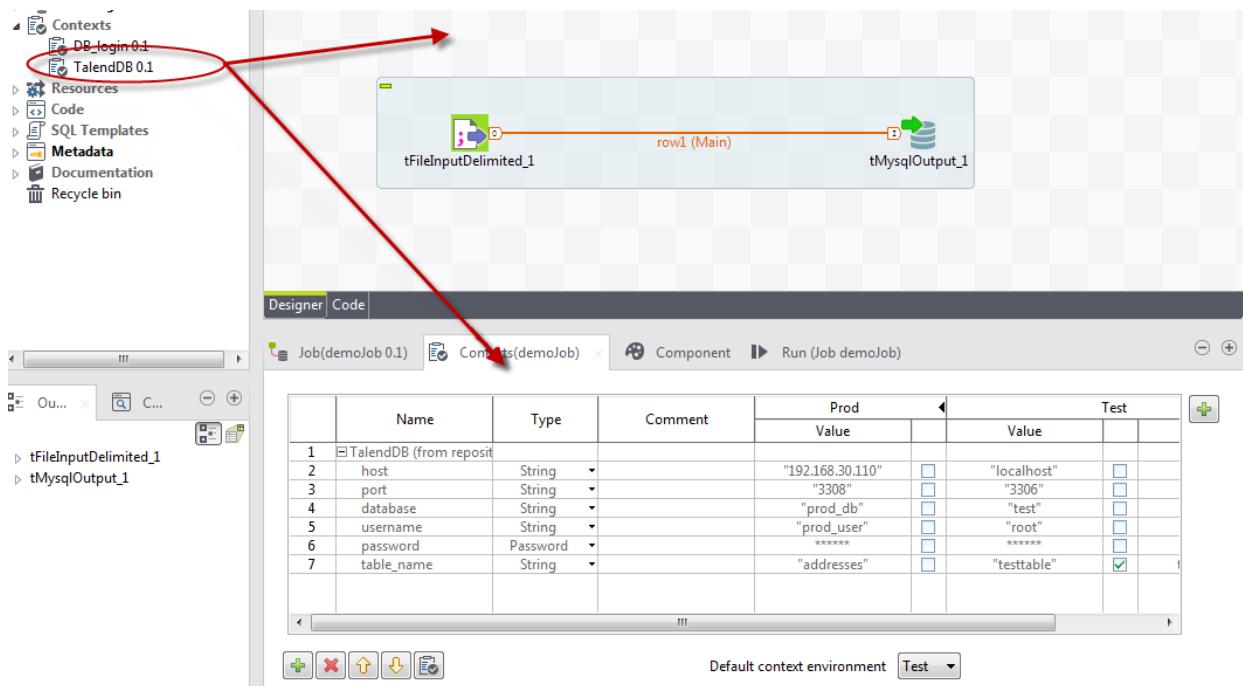
### Dropping a context group onto a Job

#### About this task

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

#### Procedure

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 design 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.

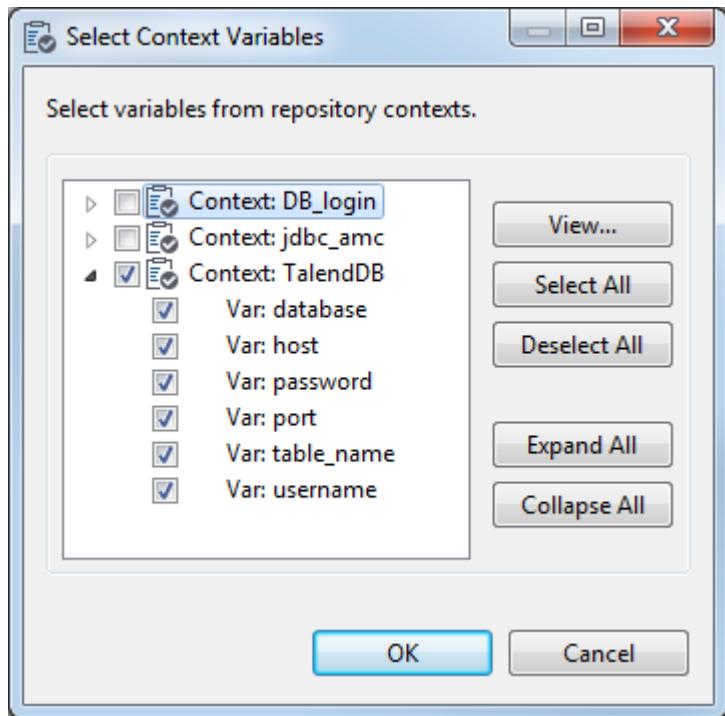
### Applying context variables to a Job using the context button

#### About this task

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

#### Procedure

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



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

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

- 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.

## Using variables in a Job

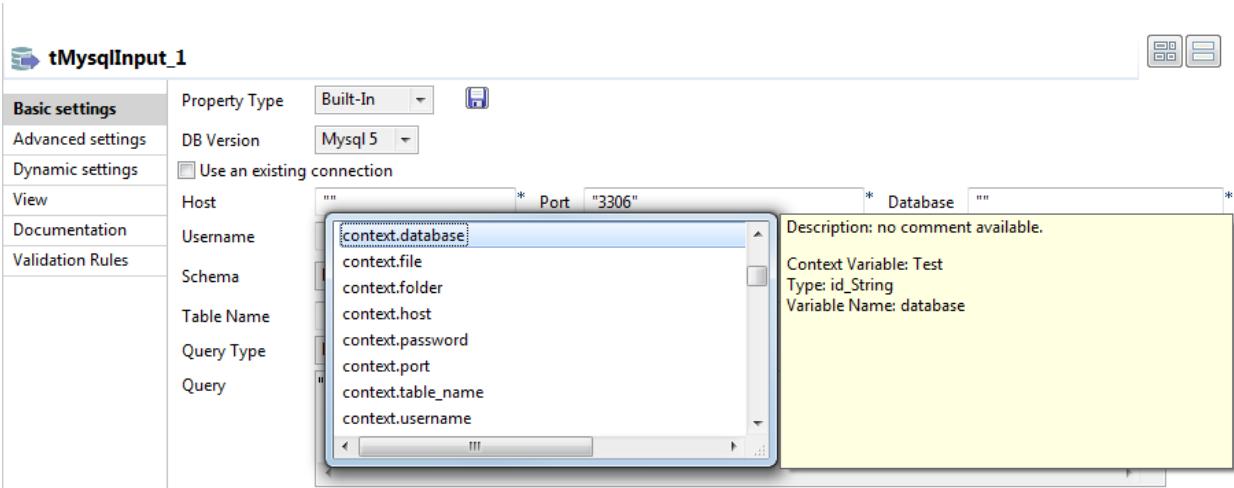
### About this task

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.

### Procedure

- 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).



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

### Running a Job in a selected context

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

#### Procedure

- Click the **Run** tab.
- In the **Context** area, select the relevant context among the various ones you created.

Name	Value
host	"localhost"
port	"3306"
database	"test"
username	"root"
password	****
table_name	"testtable"

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.

## 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.

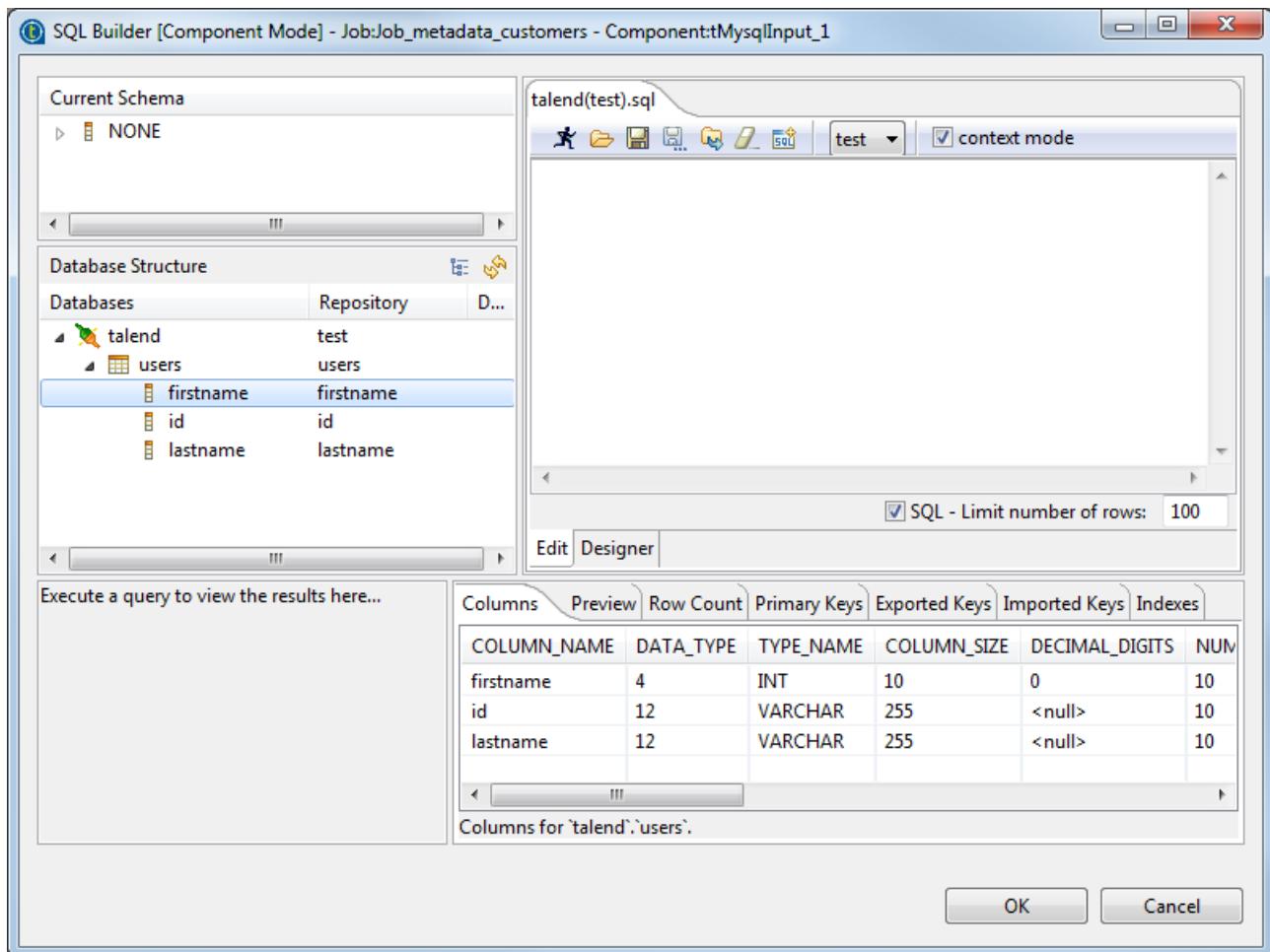
## Creating queries using the SQLBuilder

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.

## Comparing 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.

### Note:

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.

Databases	Repository	Diff
talend	ClientDB	
client	client	
CLIENT_ACCOUNT	CLIENT_ACCOUNT	
CLIENT_BIRTHDAY	CLIENT_BIRTHDAY	
ID_CLIENT	ID_CLIENT	
CLIENT_NAME	CLIENT_NAME	Red highlight
client_contract	client-contract	Red highlight
CONTRACT_TYPE	CONTRACT_TYPE	
CONTRACT_VALUE	CONTRACT_VALUE	
ID_CLIENT	ID_CLIENT	
ID_CONTRACT	ID_CONTRACT	Blue highlight
FOO		Red highlight
axetable1	axetable1	Red highlight
ID_MONTH	ID_MONTH	
ID_TYPE	ID_TYPE	
MONTH	MONTH	
ID_USER		Blue highlight
+ sales		

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**.

## Creating a query

### About this task

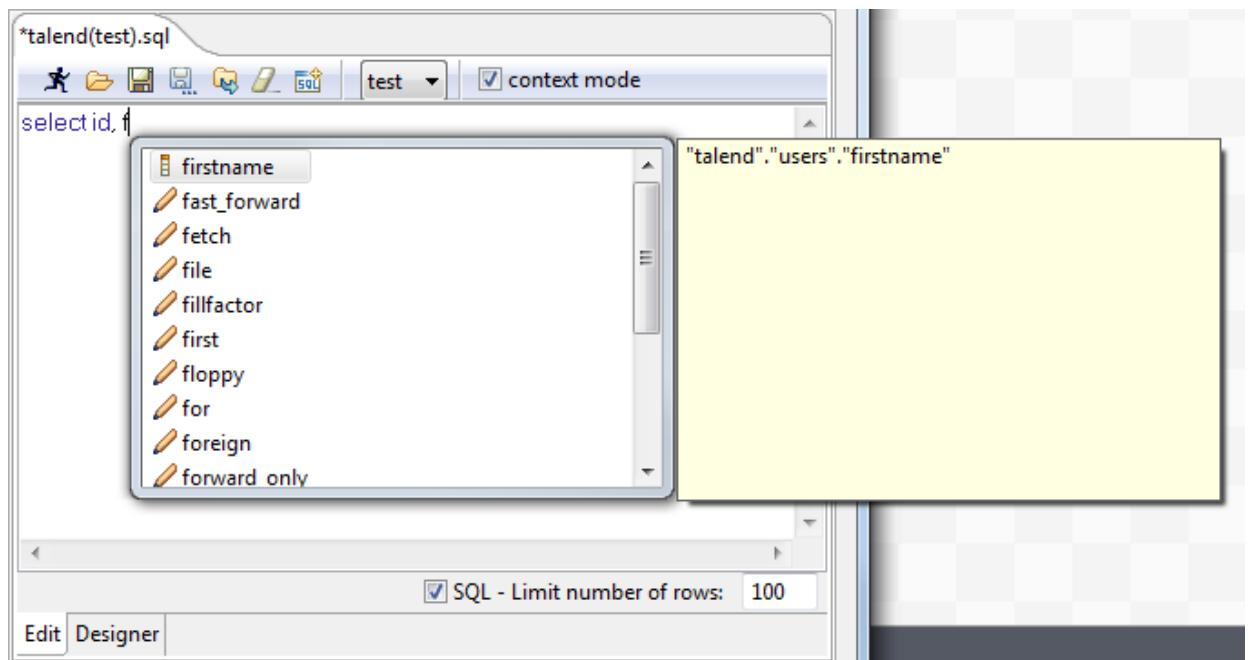
The **SQL Builder** editor is a multiple-tab editor that allows you to write or graphically design as many queries as you want.

To create a new query, complete the following:

### Procedure

1. Right-click the table or on the table column and select **Generate Select Statement** on the pop-up list.

- Click the empty tab showing by default and type in your SQL query or press **Ctrl+Space** to access the autocomplete list. The tooltip bubble shows the whole path to the table or table section you want to search in.



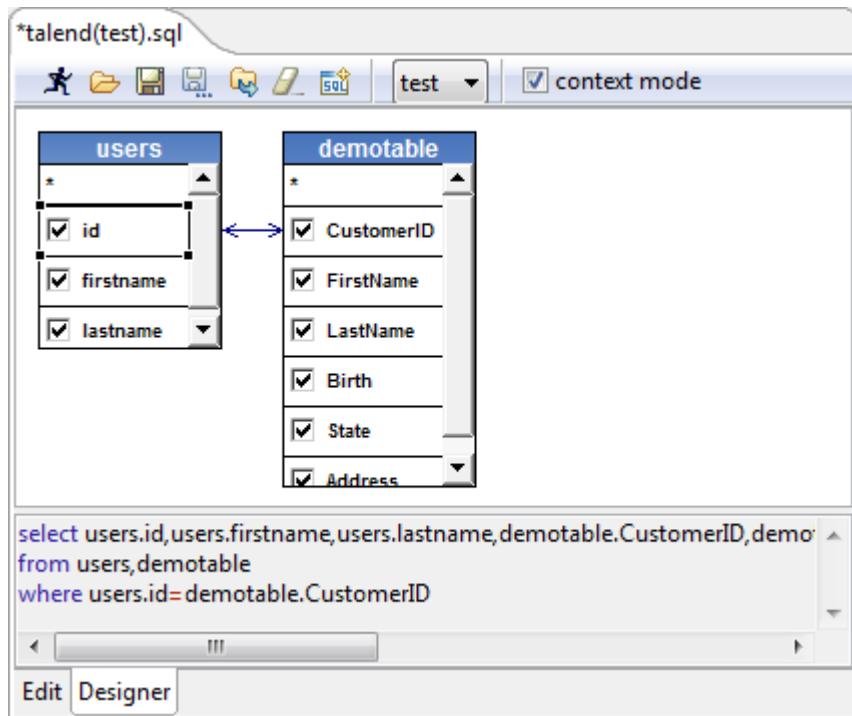
Alternatively, the graphical query **Designer** allows you to handle tables easily and have real-time generation of the corresponding query in the **Edit** tab.

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

**Note:**

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

- 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.
  - 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.

**Note:**

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.

#### Storing a query in the repository

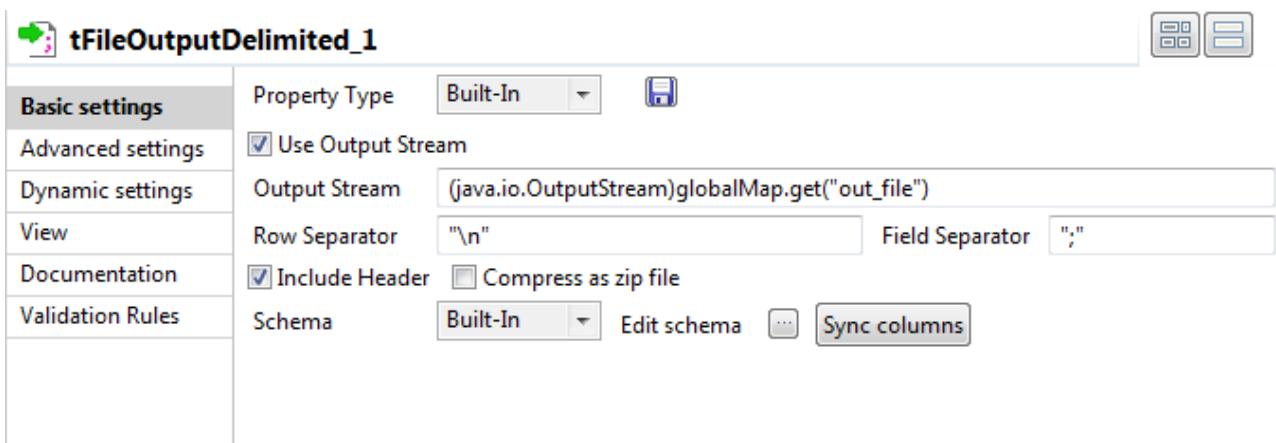
To be able to retrieve and reuse queries, we recommend you to store them in the repository.

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.

#### Using 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.

**Note:**

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 [Data Integration Job Examples](#).

## Handling Jobs: miscellaneous subjects

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

- [Using folders](#) on page 86
- [Sharing a database connection](#) on page 87
- [Adding notes to a Job design](#) on page 90
- [Displaying the code or the outline of your Job](#) on page 91
- [Managing the subJob display](#) on page 92
- [Defining options on the Job view](#) on page 94

### Using folders

#### About this task

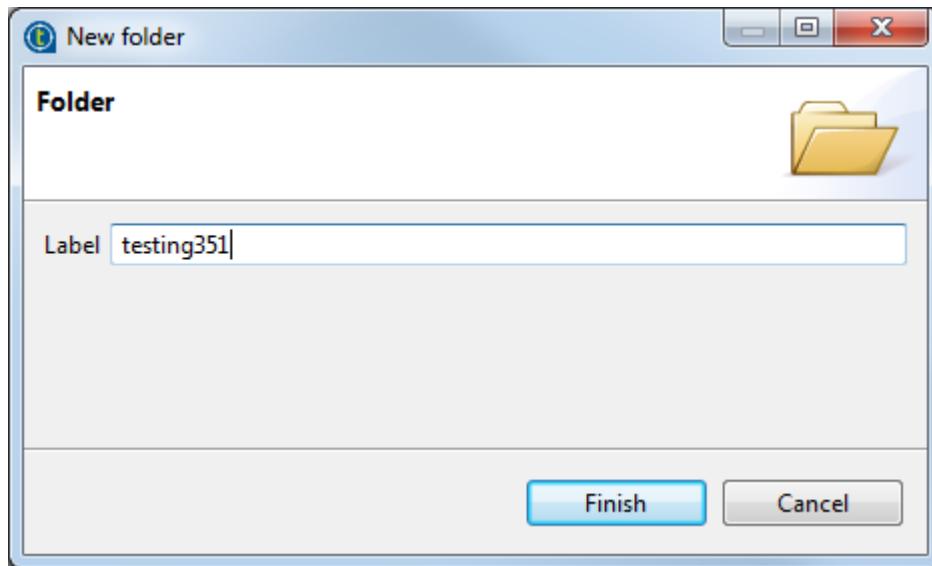
You can organize your Jobs into folders.

To create a folder, proceed as follows:

#### Procedure

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.

## Results

### Note:

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

## Sharing a database connection

### About this task

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.

**Warning:** 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.

The procedure below assumes that you have two related Jobs (a parent Job and a child Job) that both need to connect to your remote MySQL database.

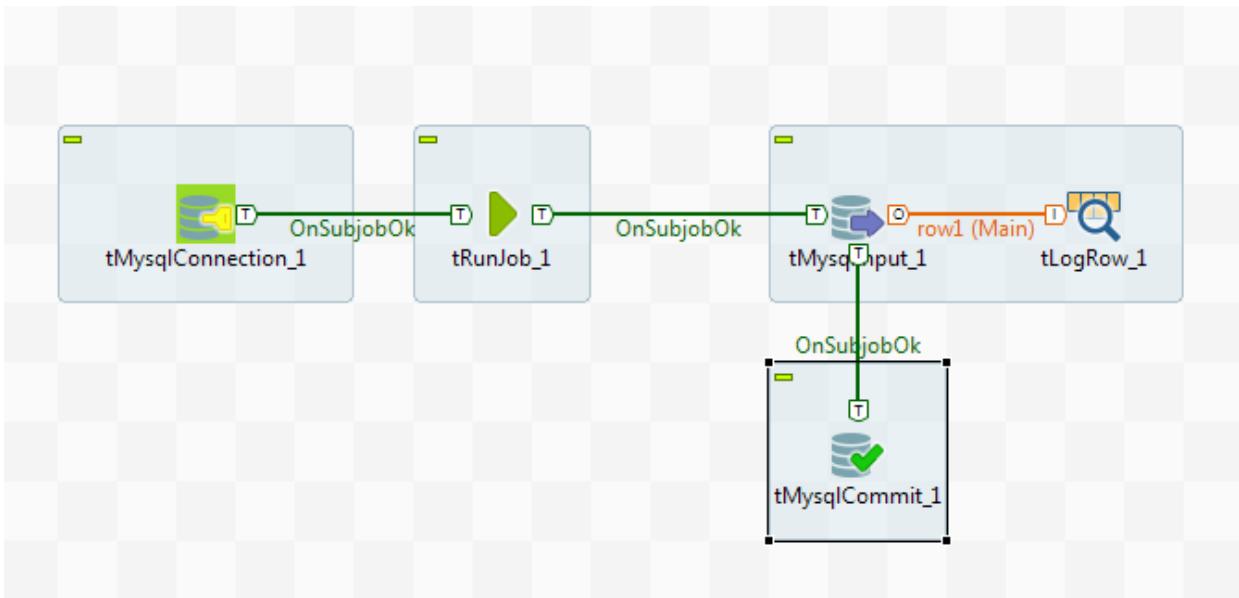
For a complete use case, see [MySQL](#).

To use a shared database connection in the two Jobs, to the following:

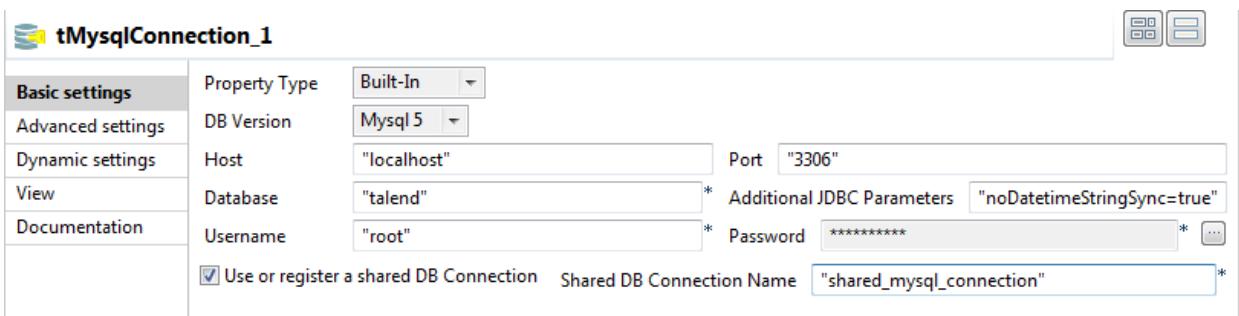
### Procedure

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.

**Note:**

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.

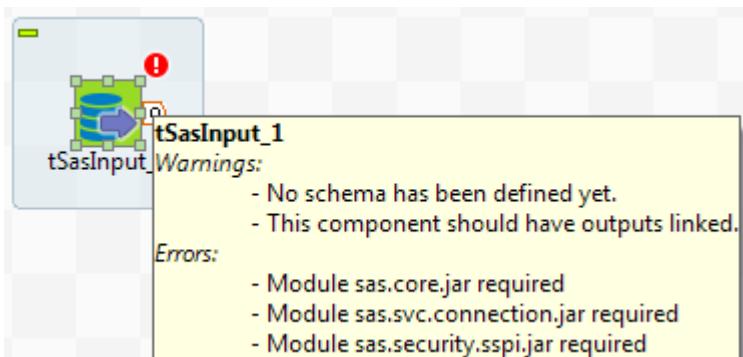
### Handling error icons on components or Jobs

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.

### Warnings and error icons on components

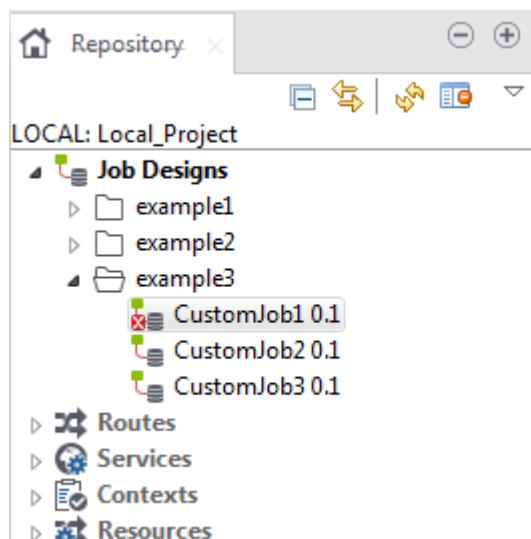
When a component is not properly defined or if the link to the next component does not exist yet, a red checked circle or a warning sign is docked at the component icon.

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.



### 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.



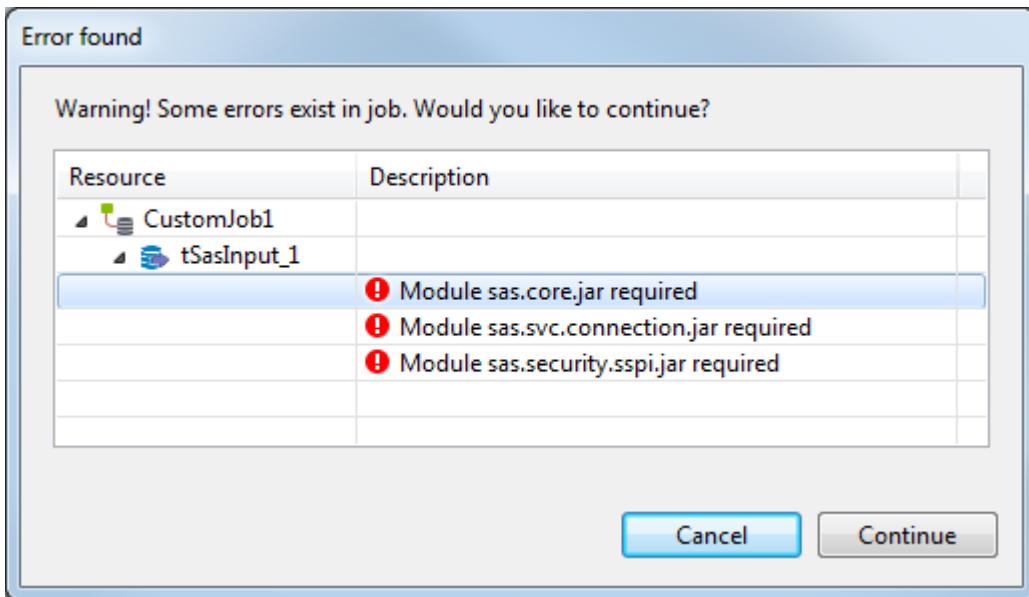
The error icon displays as well on the tab next to the Job name when you open the Job on the design workspace.

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.



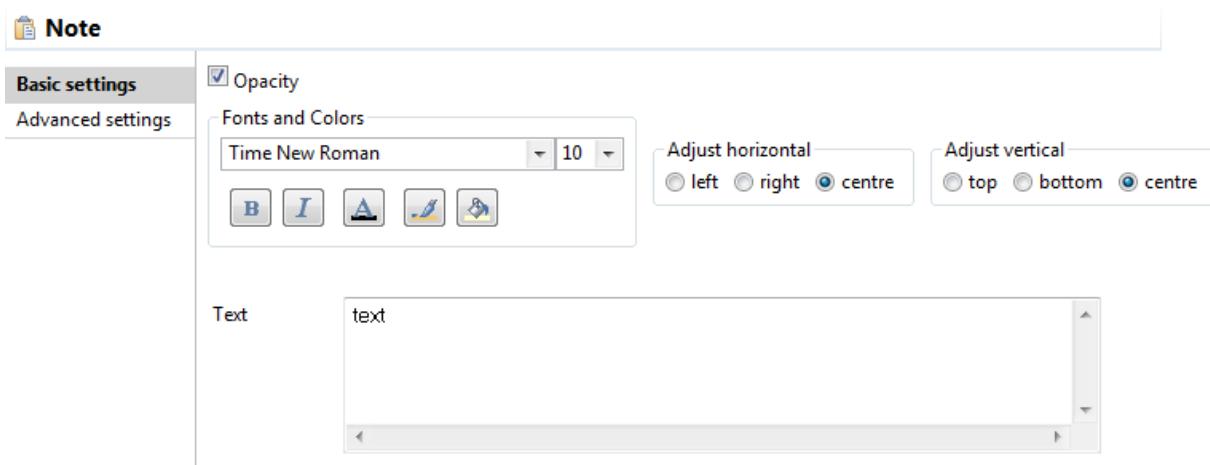
Click **Cancel** to stop your Job execution or click **Continue** to continue it.

### Adding 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.

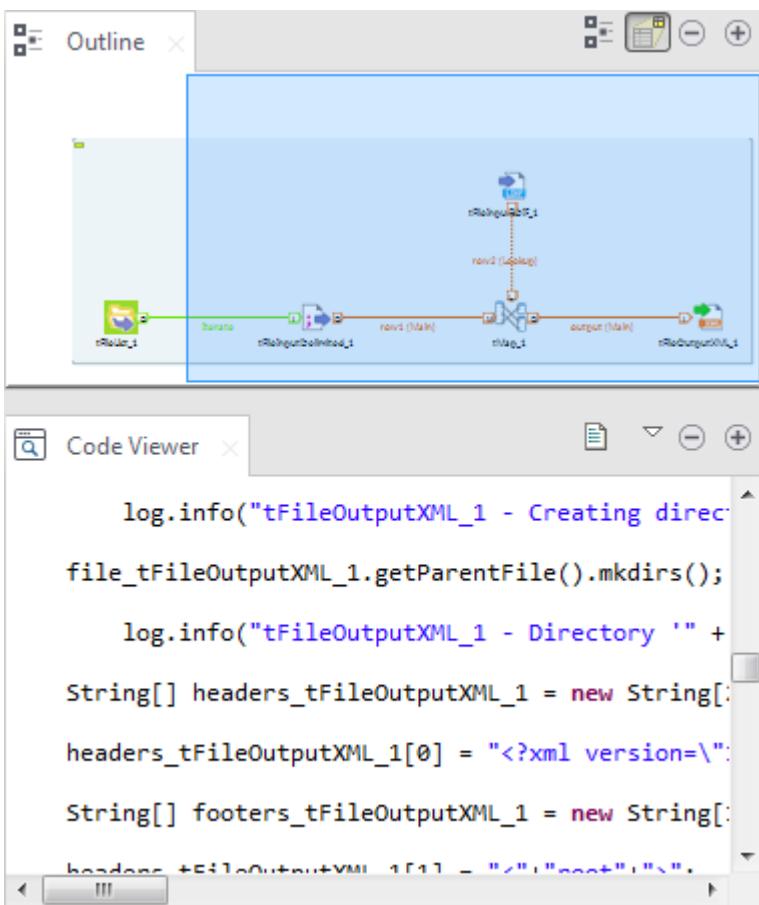
## Displaying the code or the outline of your Job

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).

### 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.

## 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.

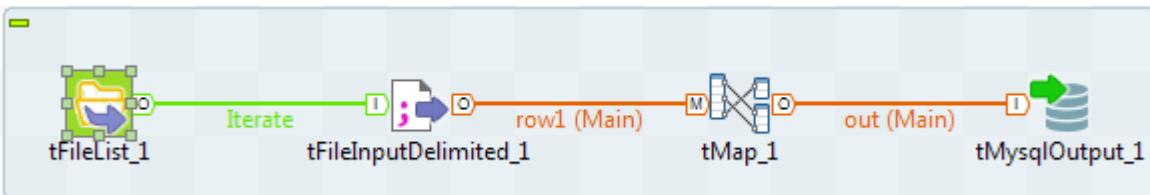
### Note:

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.

## Managing 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.



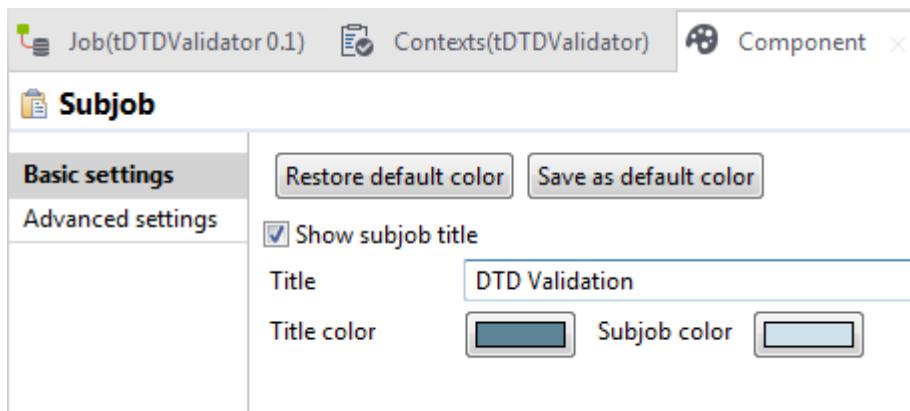
This blue highlight helps you easily distinguish one subJob from another.

**Note:** 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 [Using the tPrejob and tPostjob components on page 52](#).

## Formatting subJobs

### About this task

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.

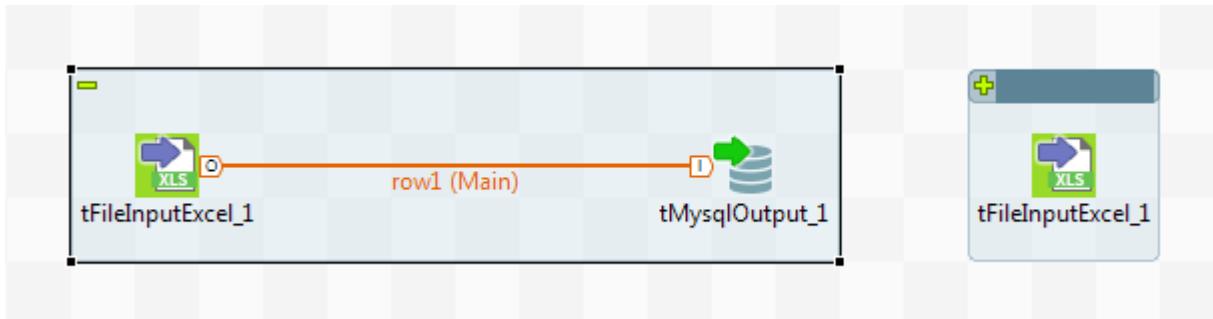
To modify the title color and the subJob color:

## Procedure

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.

## Collapsing 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.



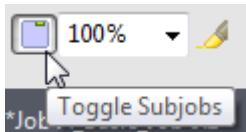
Click the minus sign (-) to collapse the subJob. When reduced, only the first component of the subJob is displayed.

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

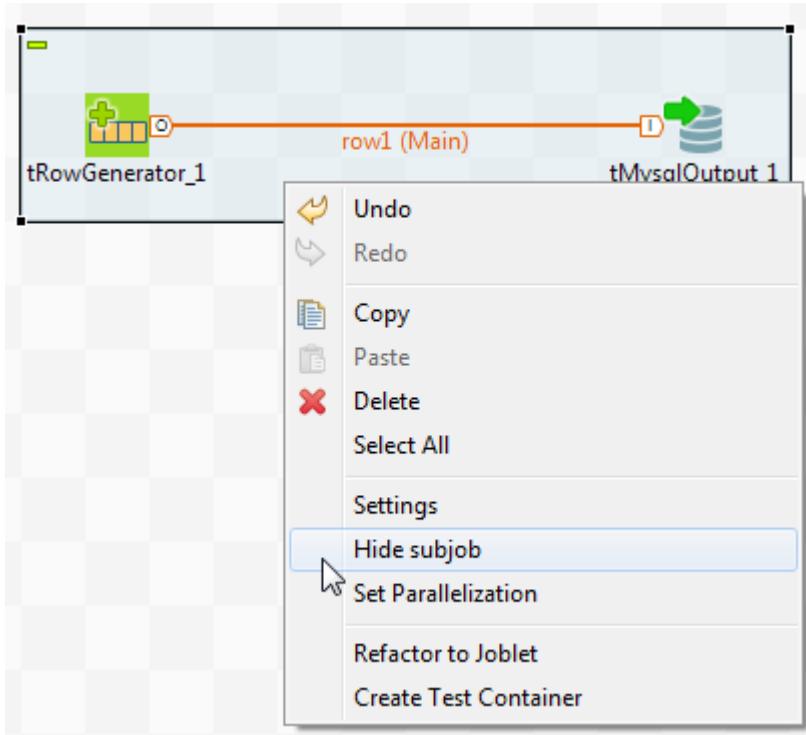
## Removing the subJob background color

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.



## Defining 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 [Automating the use of statistics & logs](#) on page 94.

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 [Using the features in the Extra tab](#) on page 95.

### Automating the use of statistics & logs

#### About this task

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.

#### Note:

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

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

#### Procedure

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

Main      Reload from project settings      Save to project settings       Use Project Settings

Use statistics (tStatCatcher)     Use logs (tLogCatcher)     Use volumetrics (tFlowMeterCatcher)

On Console     On Files     On Databases

Property Type: Repository    DB (JDBC):jdbc\_amc    ...

Db Type: General JDBC    \*

JDBC URL: context.jdbc\_amc\_JdbcUrl

Driver jar: Jar Name  
context.jdbc\_amc\_DriverJAR

User: context.jdbc\_amc\_Login    \*    Password: context.jdbc\_amc\_Password    \*

Class name: context.jdbc\_amc\_ClassName    ...   

Logs Table: "log"    ...   

Meter Table: "flow"    ...   

Catch runtime errors     Catch user errors     Catch user warnings

Catch components statistics (tStatCatcher Statistics)

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.

## Results

### Note:

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.

Job\_metadata\_customers 0.1

Main      Reload from project settings      Save to project settings       Use Project Settings

Use statistics (tStatCatcher)     Use logs (tLogCatcher)     Use volumetrics (tFlowMeterCatcher)

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.

## Using 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 Data Integration Job Examples.

- 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](#) on page 387.

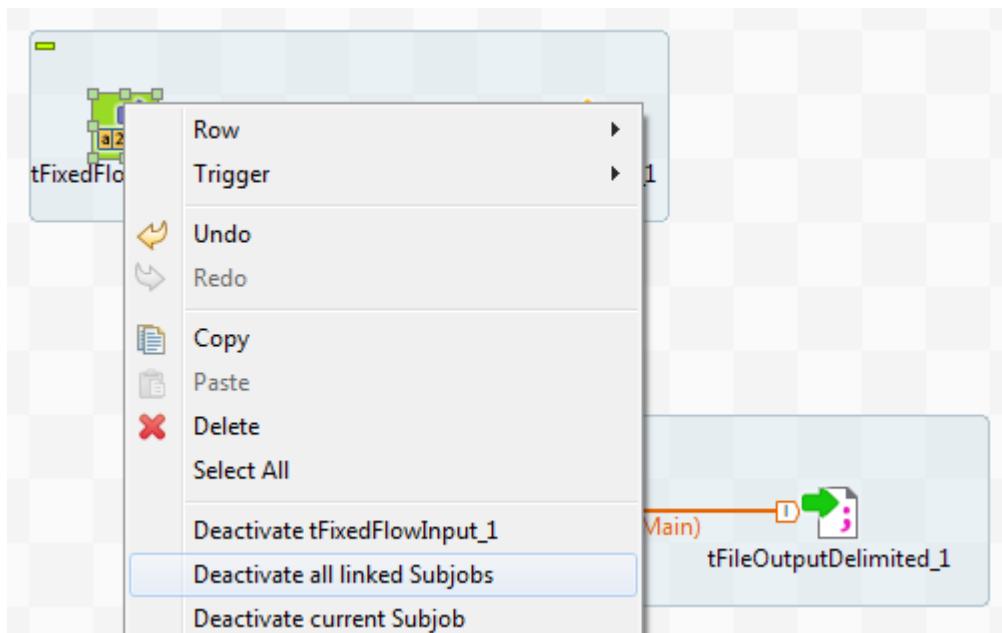
- Click **Reload from project settings** to update the context parameters list with the latest context parameters from the project settings.

## Managing Jobs

### 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.

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.



## Activate or deactivate a component

### About this task

To activate or deactivate a component, proceed as follows:

### Procedure

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.

## Activate or deactivate a subJob

### About this task

To activate or deactivate a subJob, proceed as follows:

### Procedure

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.

## Activate or deactivate all linked subJobs

### About this task

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

### Procedure

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.

## 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.

### Importing items

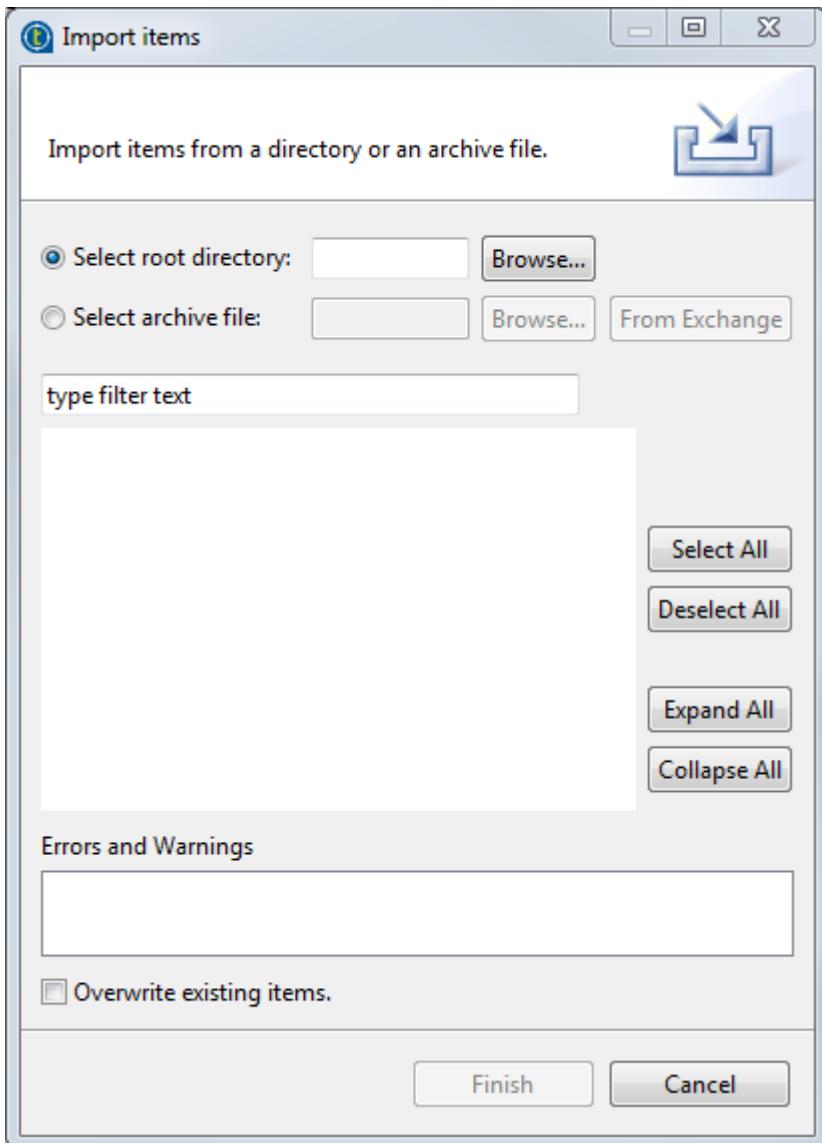
You can import items from previous versions of Talend Studio or from a different project of your current version.

The items you can possibly import are multiple:

- Business Models

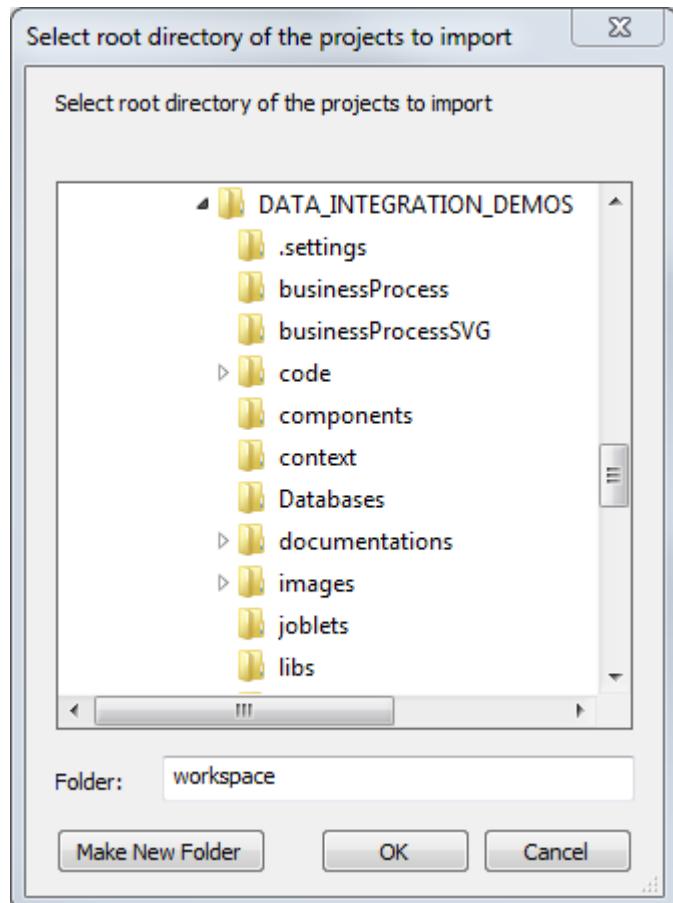
- Jobs Designs
- Routines
- Documentation
- Metadata

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**.

**Note:** You cannot overwrite the existing items, if:

- the item is identical but the path is different, or
- the name is identical but the item is different.

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**.

**Note:** You cannot overwrite the existing items, if:

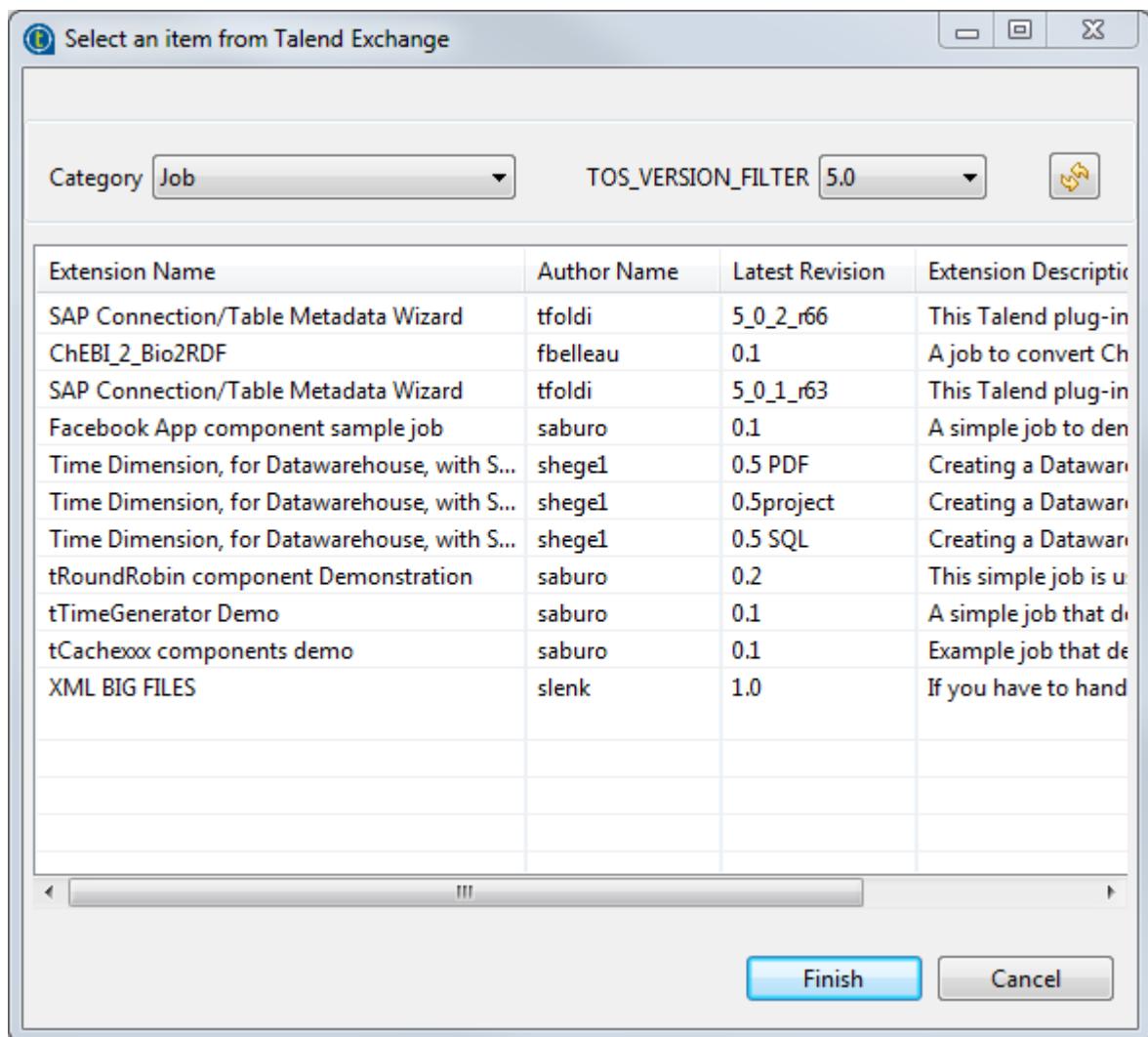
- the item is identical but the path is different, or
- the name is identical but the item is different.

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**.

**Note:** You cannot overwrite the existing items, if:

- the item is identical but the path is different, or
- the name is identical but the item is different.

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.

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

## Building Jobs

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

### About this task

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.

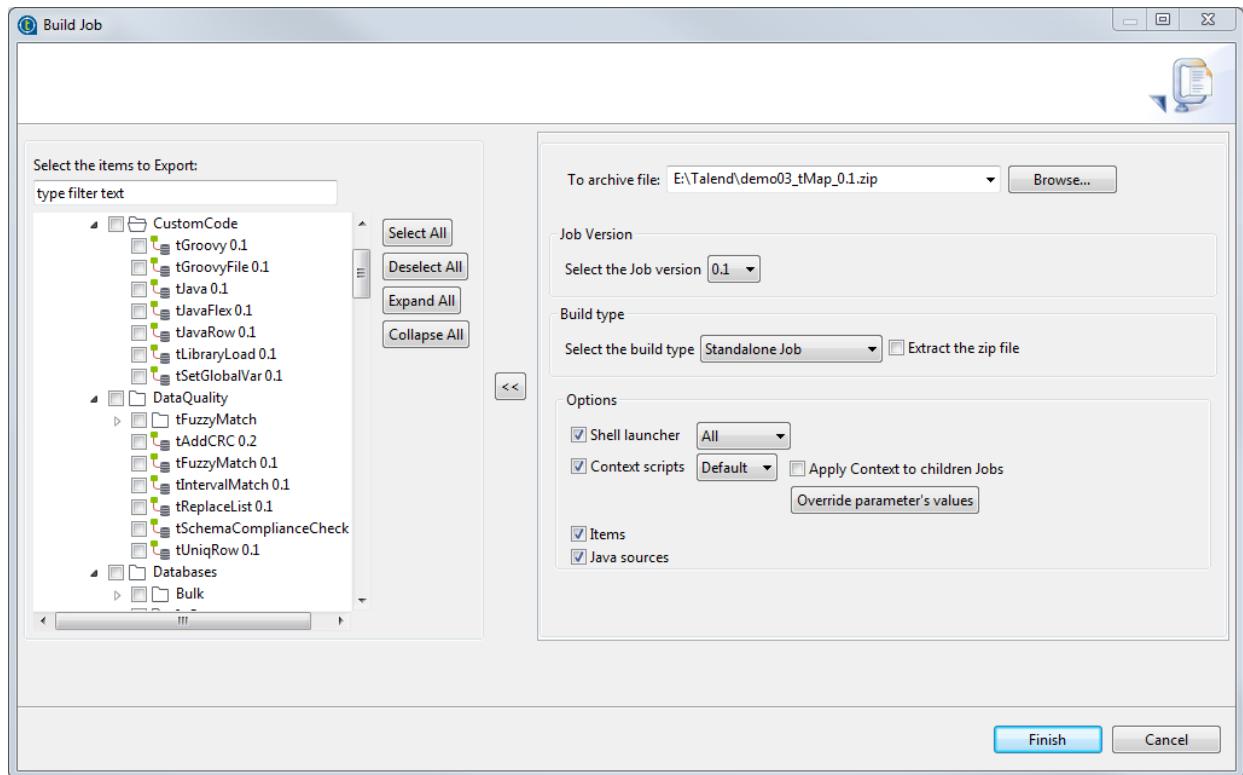
**Note:** Your Talend Studio provides a set of default build script templates. You can customize those templates to meet your actual needs. For more information, see [Customizing shell command templates](#) on page 371 and [Customizing Maven build script templates](#) on page 372.

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 > 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.

### Procedure

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.

**Note:** 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:
  - **Standalone Job**
  - **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 .bat and/or .sh files necessary to launch the built Job. <ul style="list-style-type: none"> <li>• <b>All</b>: exports the .bat and .sh files.</li> <li>• <b>Unix</b> exports the .sh file.</li> <li>• <b>Windows</b> exports the .bat file.</li> </ul>

Option	Description
<b>Context scripts</b>	Select this check box to export ALL context parameters files and not just those you select in the corresponding list.  <b>Note:</b> To export only one context, select the context that fits your needs from the <b>Context scripts</b> list, including the .bat or .sh files holding the appropriate context parameters. Then you can, if you wish, edit the .bat and .sh 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 .item and .properties files, Java and <b>Talend</b> sources.  <b>Note:</b> If you select the <b>Items</b> or <b>Source files</b> check box, you can reuse the built Job in a Talend Studio installed on another machine. These source files are only used in Talend Studio.
<b>Java sources</b>	Select this check box to export the .java 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.

## Results

A zipped file for the Jobs is created in the defined place.

**Note:** 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 on page 348](#). For more information about classes and inner classes, see relevant Java manuals.

## Building 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.

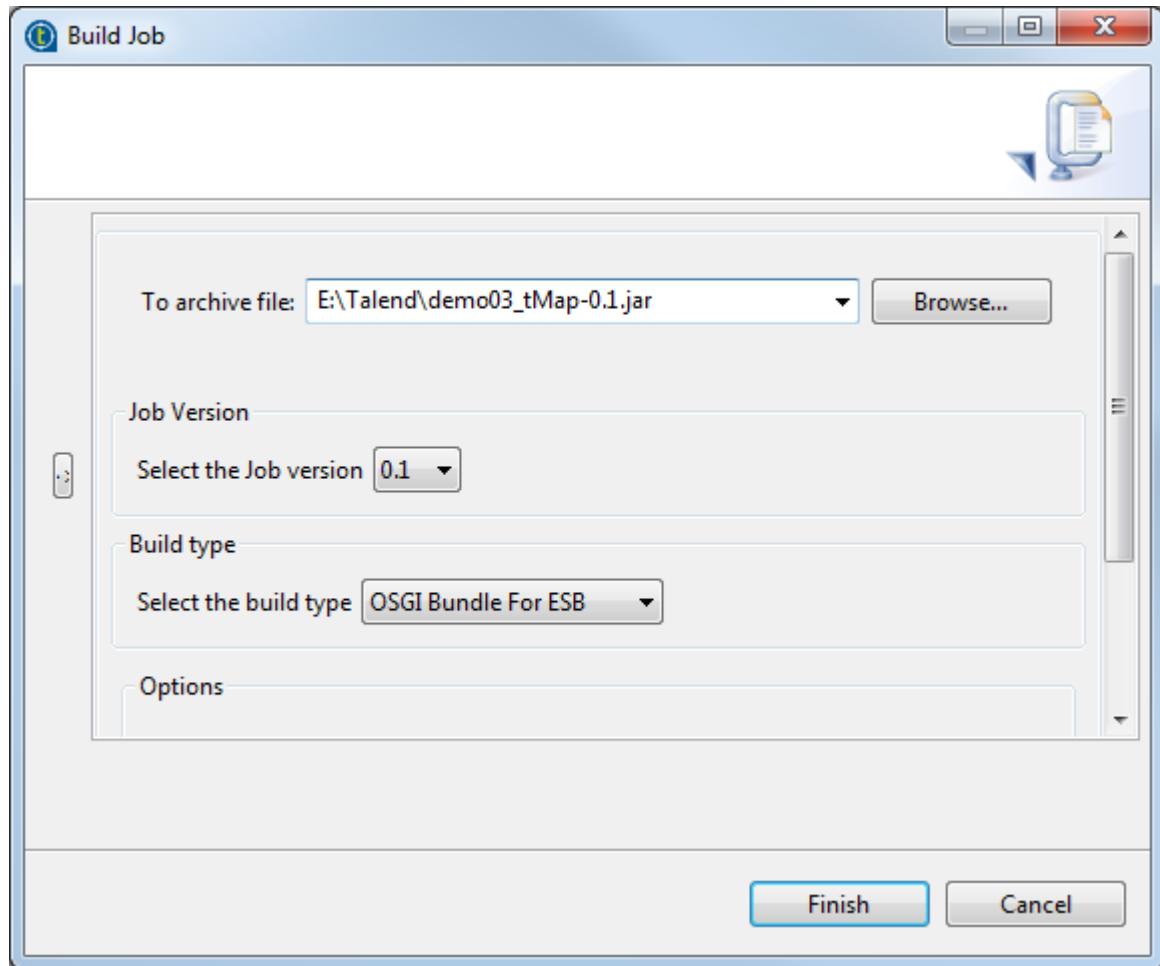
**Note:**

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.

## Building a Job as an OSGI Bundle For ESB

### About this task

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**.



### Procedure

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.

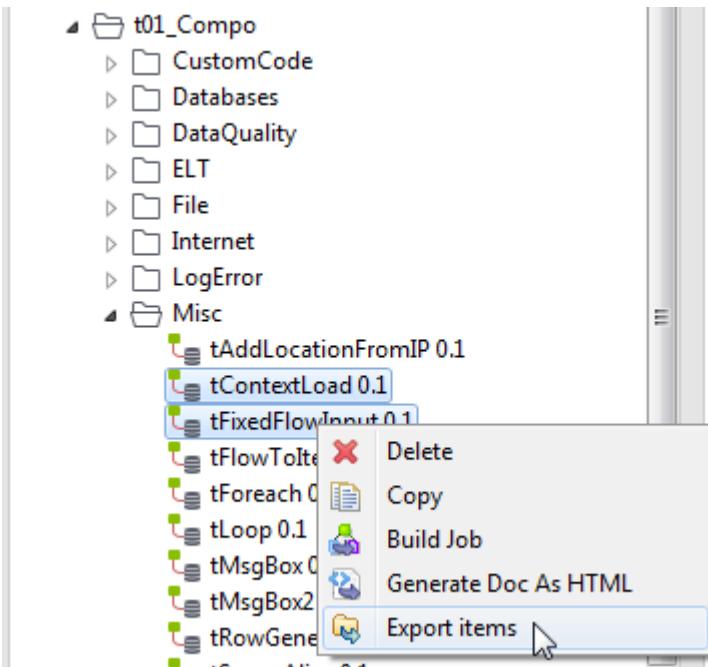
### Exporting items

### About this task

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:

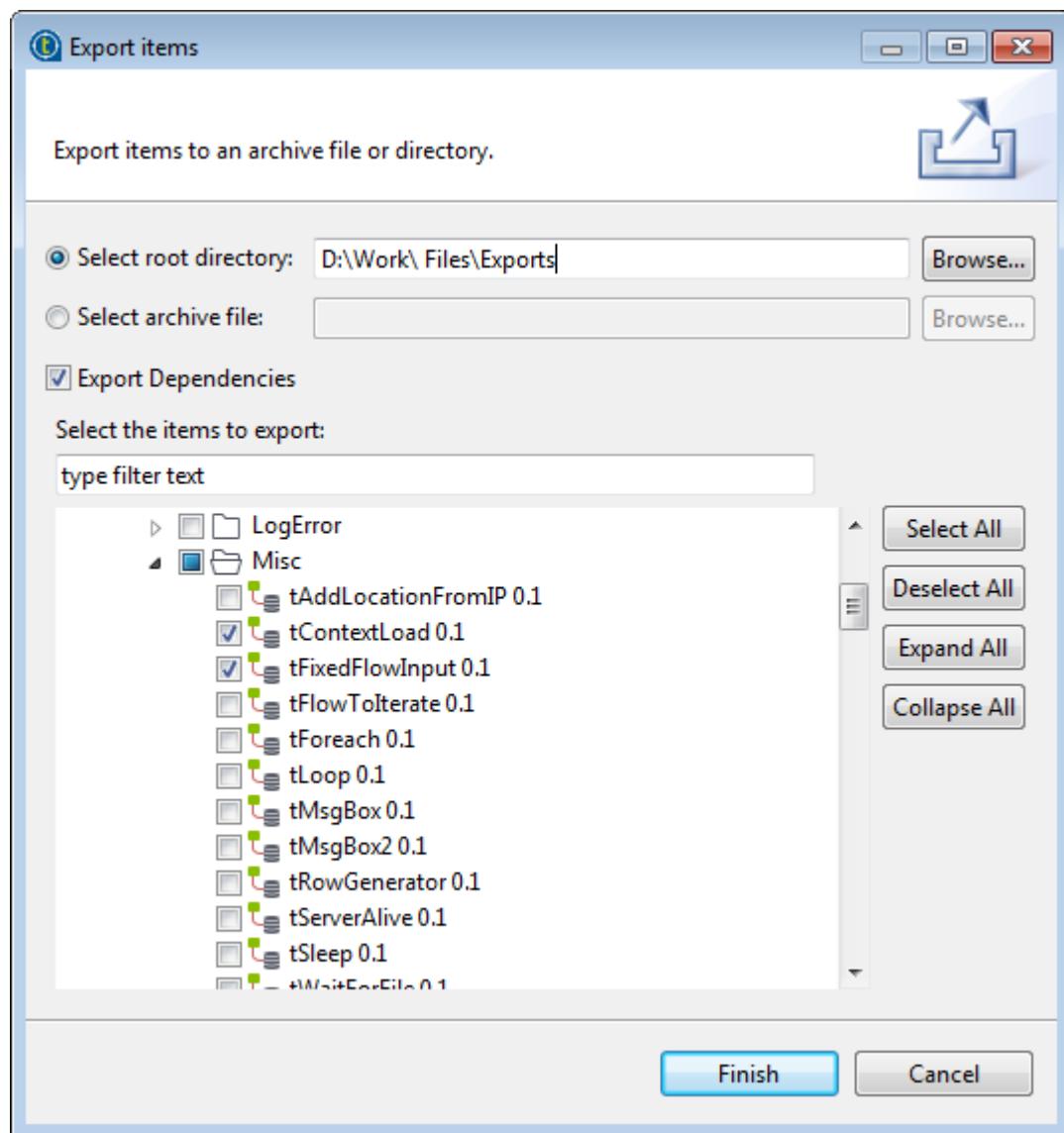
## Procedure

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.



**Warning:** 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.

- 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.

**Note:** 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](#) on page 346.

- Click **Finish** to close the dialog box and export the items.

## Changing context parameters in Jobs

### About this task

As explained in [Building Jobs](#) on page 101, you can edit the context parameters:

### Procedure

- 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 value1 for parameter key1	--context_param key1=value1
To change value1 and value2 for respective parameters key1 and key2	--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"

## 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.

### Handling 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 **Update Detection** dialog box is displayed to let you update the impacted Jobs when:

- you modify a centralized repository entry that is used in any Jobs and click **Yes** in the **Modification** dialog box that is display automatically.
- you select **Detect Dependencies** from the right-click menu of a modified repository entry that is used in any Jobs, or click the  icon on the toolbar after modifying a centralized repository entry that is used in any Jobs.

For more information on updating impacted Jobs, see [Updating impacted Jobs automatically](#) on page 108 and [Updating impacted Jobs manually](#) on page 109.

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.

### Modifying a repository item

#### About this task

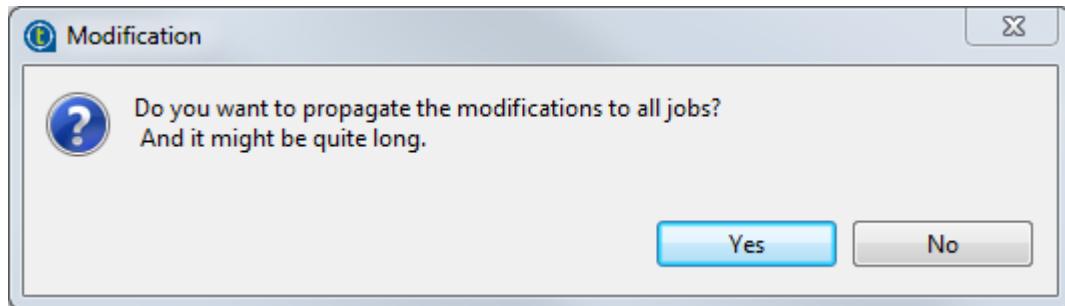
To update the parameters of a repository item, complete the following:

#### Procedure

- Expand the **Metadata**, or **Contexts** node in the **Repository** tree view and browse to the relevant entry that you need to update.
- 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.



3. 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 [Updating impacted Jobs automatically](#) on page 108.

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 one by one. For more information on another way of propagating changes, see [Updating impacted Jobs manually](#) on page 109.

### [Updating impacted Jobs automatically](#)

#### About this task

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.

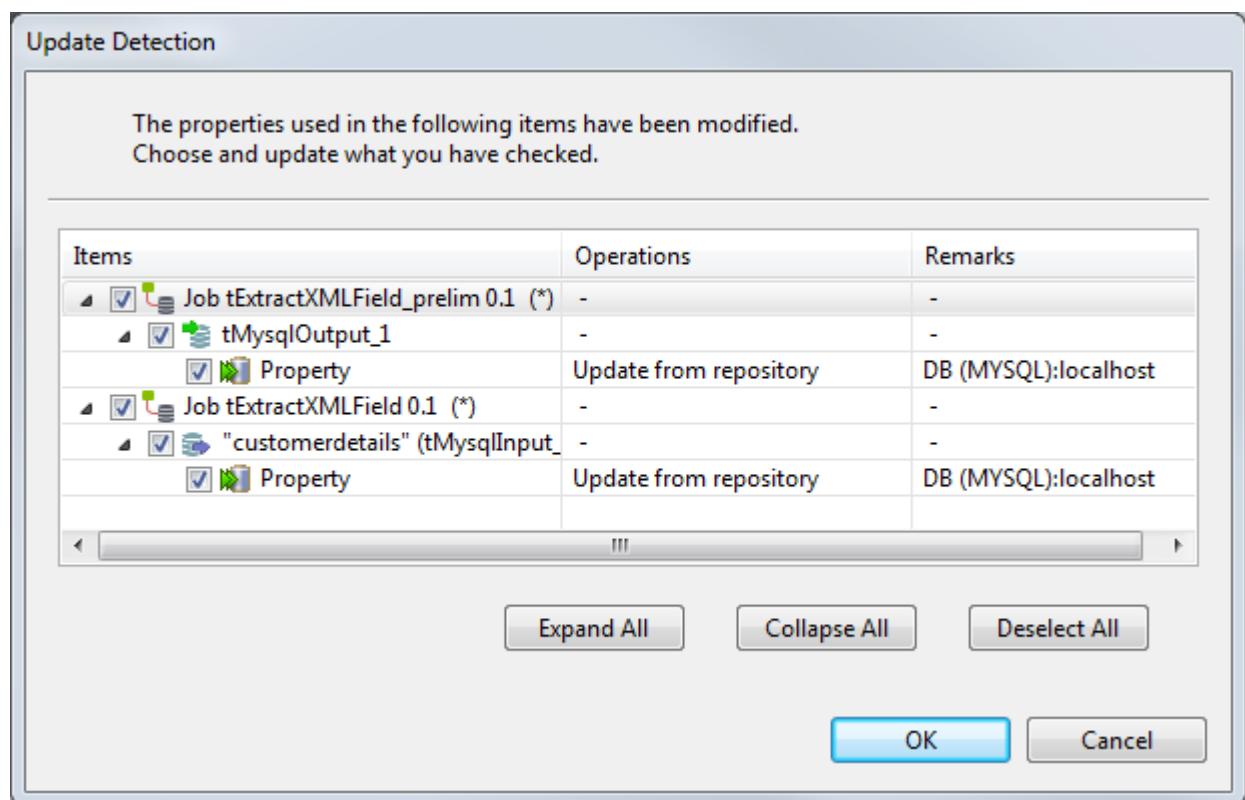
To update impacted Jobs, complete the following:

#### Procedure

1. 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.



2. Select the check boxes corresponding to the Jobs you want to update and clear those corresponding to the Jobs you do not want to update.

You can update them any time later through the **Detect Dependencies** menu or the  icon on the toolbar. For more information, see [Updating impacted Jobs manually](#) on page 109.

3. Click **OK** to close the dialog box and update all selected Jobs.

### [Updating impacted Jobs manually](#)

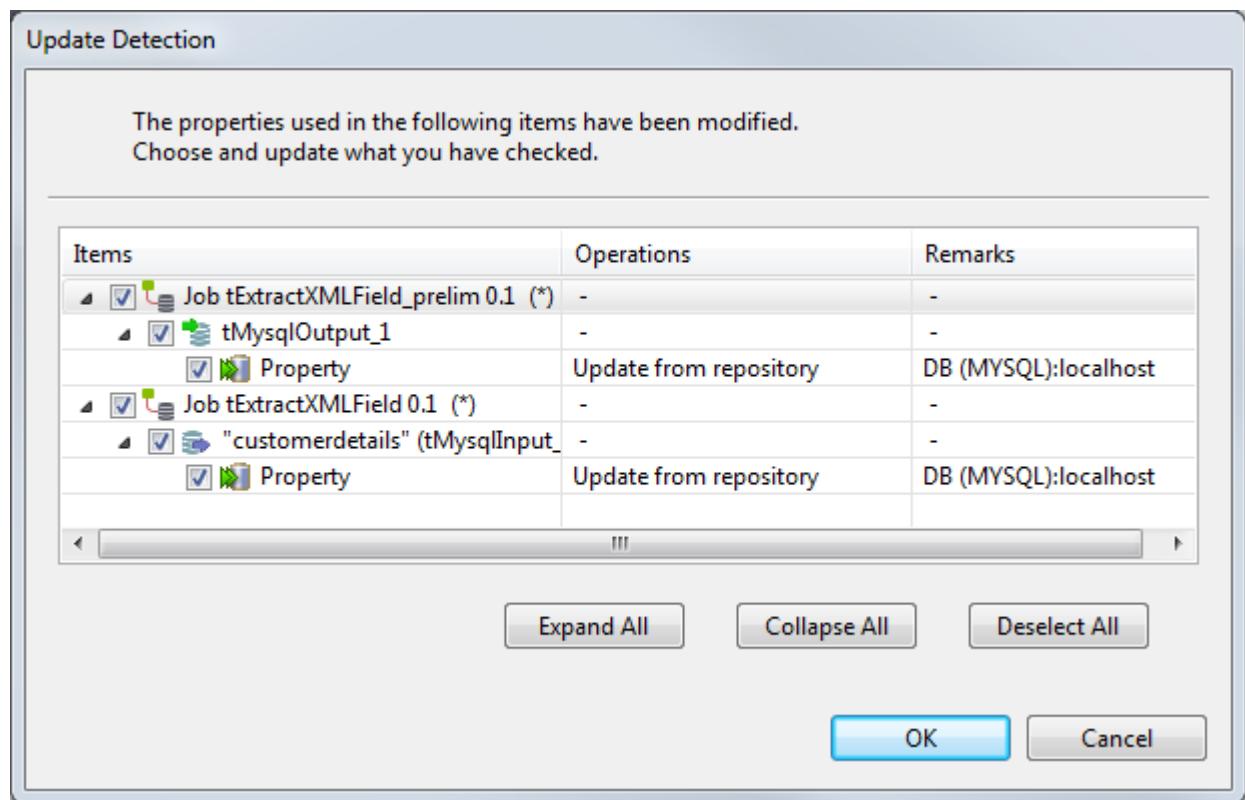
#### About this task

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:

#### Procedure

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.



**Note:** You can also display this dialog box by clicking the  icon on the toolbar.

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.

## Results

**Note:** 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.

## Searching a Job in the repository

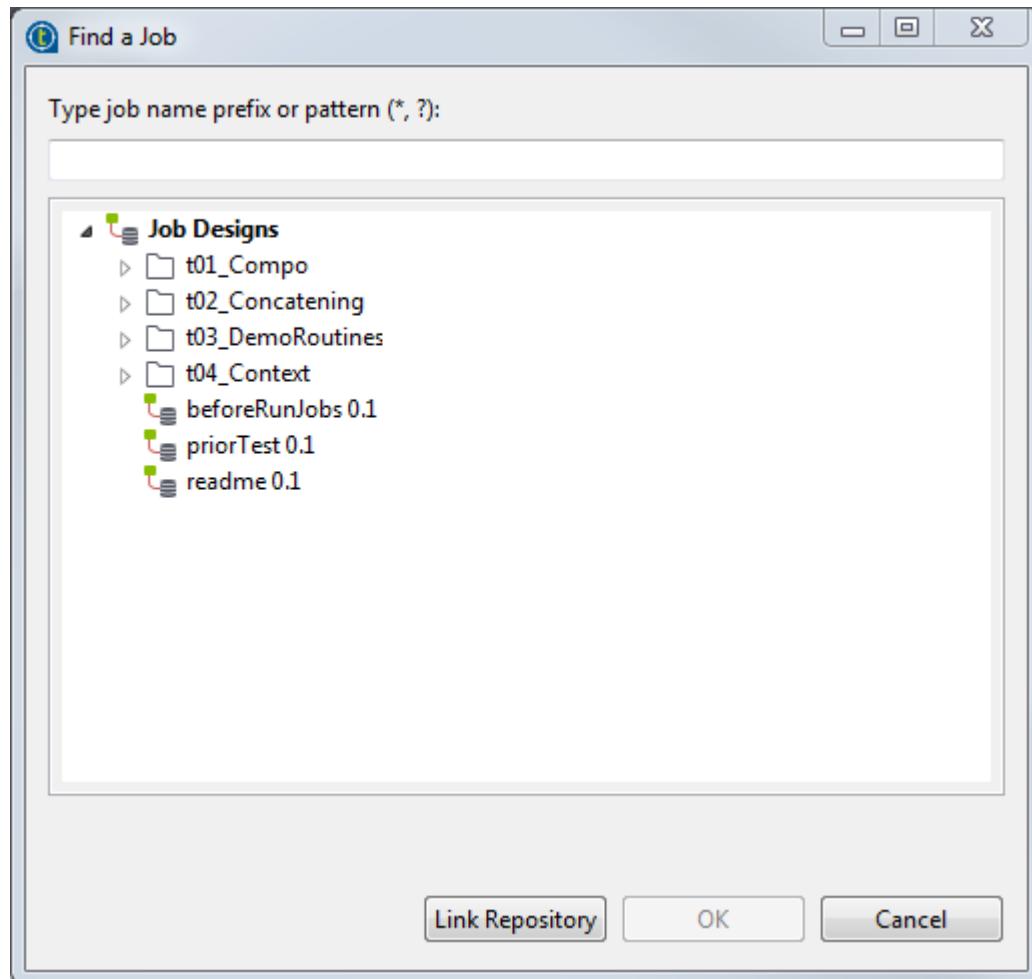
### About this task

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:

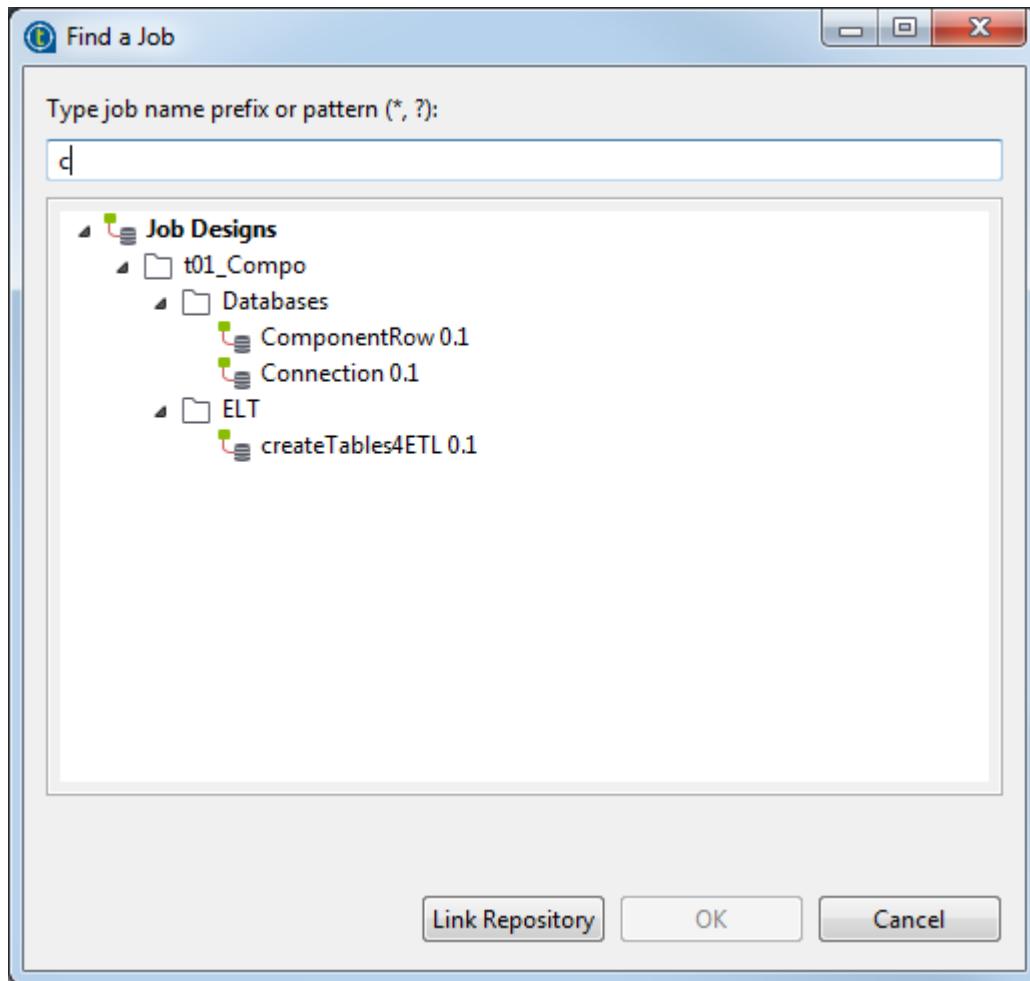
### Procedure

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.

## 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.

The following sections describe how to manage the version of a 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](#) on page 382.

### Updating the version of an inactive Job

#### Procedure

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, and click the **M** button next to the **Version** field to increment the major version or the **m** button to increment the minor version.
  - select **Open another version** from the drop-down list, then in the dialog box select the **Create new version and open** check box and click the **M** button to increment the major version or the **m** button to increment the minor version.
3. Click **Finish** to validate the modification.

## Results

You have created a new version for the Job.

**Note:** 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.

## Updating the version of an active Job

You can also save your currently active Job and increment its version at the same time.

### Procedure

- Click **File > Save As....**
- In the **Save As** dialog box, set a new version and click **Finish**.

**Note:** 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.

## Working on different versions of a Job

You can access a list of the different versions of a Job and perform certain operations.

### Procedure

- In the **Repository** tree view, select the Job you want to consult the versions of.
- On the configuration tabs panel, click the **Job** tab and then click **Version** to display the version list of the selected Job.
- Right-click the Job version you want to work on.
- Select an option:

Select	To...
<b>Edit Job</b>	open the last version of the Job.  <b>Note:</b> 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.

Select	To...
<b>Edit properties</b>	edit Job properties. <b>Note:</b> The Job should not be open on the design workspace, otherwise it will be in read-only mode. <b>Note:</b> 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.

## Removing a version of a Job

If you are sure that a version of a Job is no longer useful, you can remove it by deleting its resource files.

### Warning:

- A Job removed this way will not go to the Recycle bin and therefore cannot be restored.
- Mis-deletion of a resource file may cause damage to the integrity of the corresponding Job and thus cause it to stop functioning.

## Procedure

1. If you want to remove the latest version of a Job and if it is currently open, close it.
2. Select **Window > Show view...** from the menu, then in the **Show View** dialog box, select **General > Navigator** and click **OK** to open the **Navigator** view in the configuration tabs area. Skip this step if the **Navigator** view is already displayed.
3. In the **Navigator** view, expand to the node named after your project. This node is in all capitals, **MY\_PROJECT** for example.
4. Go to the **process** folder to show the resource files of your Job. If your Job is in a sub folder, go to that sub folder to show the corresponding resource files.
5. Select the three resource files corresponding to your Job name and the version you want to delete, right-click the selection and click **Delete** on the context menu, and then click **OK** in the **Delete Resources** dialog box.

## Example

To delete the 0.1 version of a Job named `my_job`, delete these files:

- `my_job_0.1.item`
- `my_job_0.1.properties`
- `my_job_0.1.screenshot`

## 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.

## Generating HTML documentation

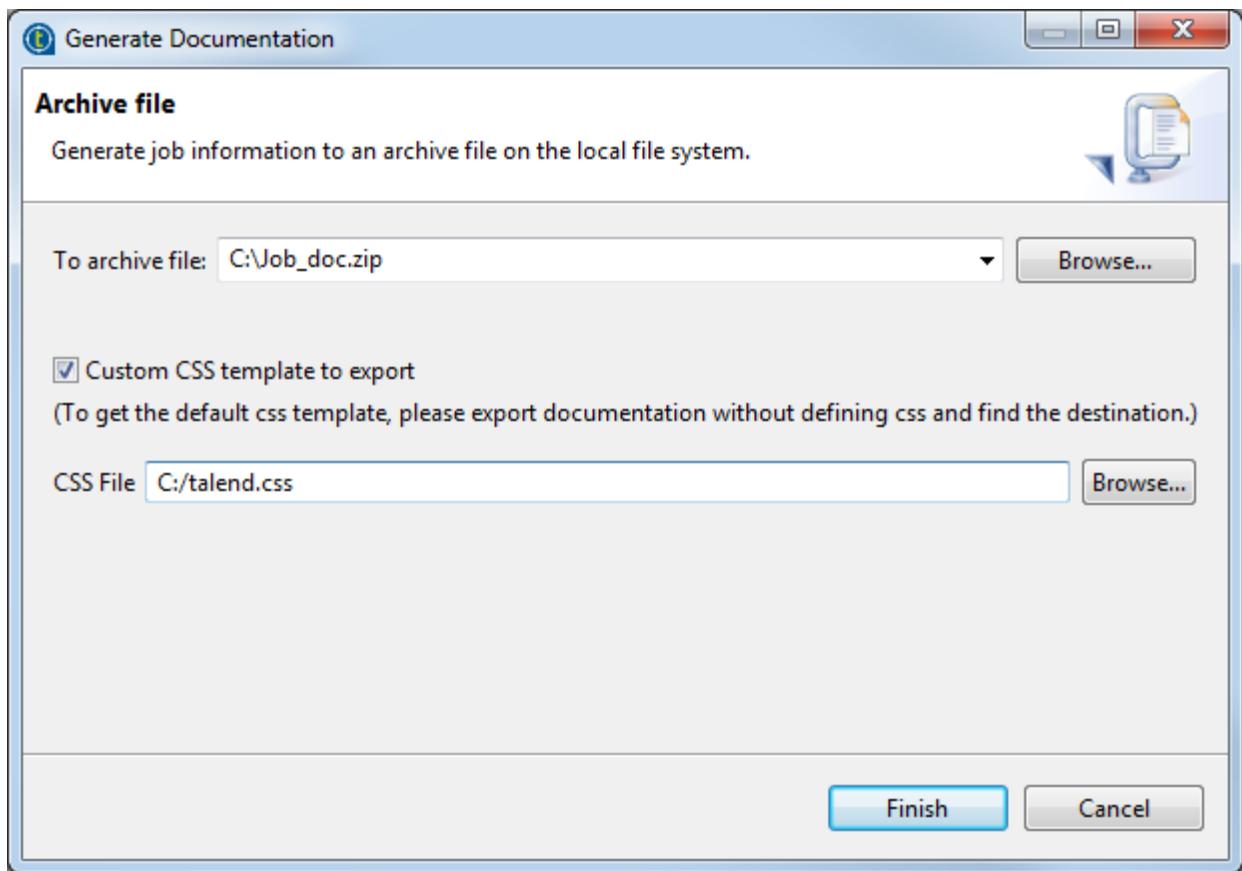
Talend Studio allows you to generate detailed documentation in HTML of the Job(s) you select in the **Repository** tree view of your Studio in the **Integration** perspective. This auto-documentation offers the following:

- 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:

### Procedure

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.

### Results

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.

## Updating the documentation on the spot

You can choose to manually update your documentation on the spot.

### Procedure

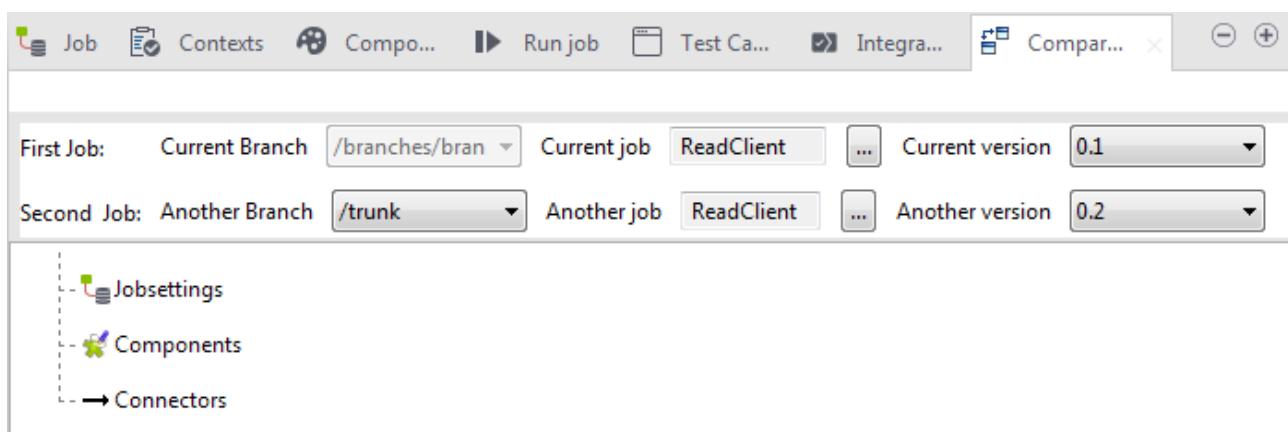
- To update a single document, right-click the relevant documentation entry and select **Update documentation**.

## Comparing Jobs

Talend Studio provides a **Compare Job** option that enables you to compare Jobs on the same or different branches in order to list the differences in the items used in the two Jobs. Using this option, you can:

- compare different versions of the same Job,
- compare the same Job in different releases of the Studio, in order to see if any modifications were done on the Job in the previous/current release, for example,
- compare Jobs that have been designed using the same template, but different parameters, to check the differences among these Jobs.

Differences between the compared Jobs are displayed in the **Compare Result** view. The result detail are grouped under the three categories: **Jobsettings**, **Components** and **Connectors**.



The table below gives the description of the comparison results under each of the above categories.

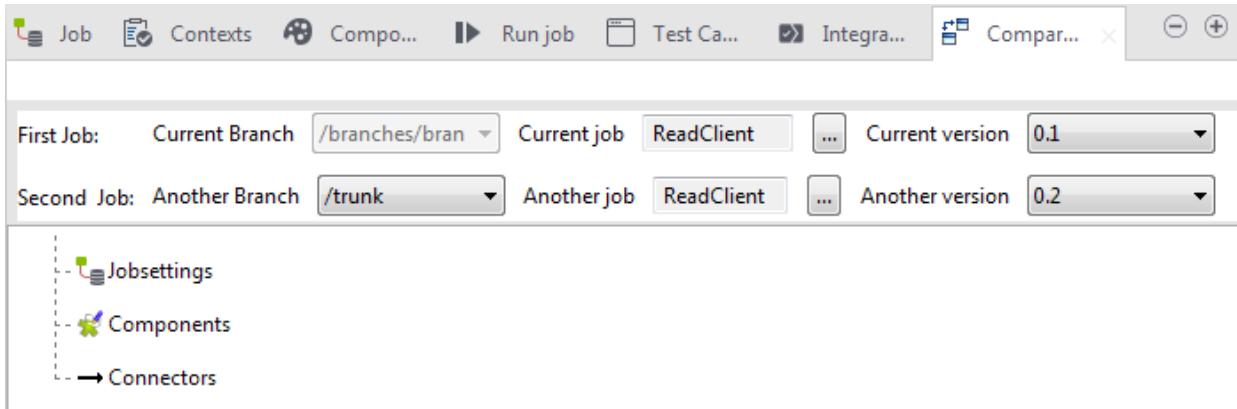
Category	Description
<b>Jobsettings</b>	lists all differences related to the settings of the compared Job.
<b>Components</b>	lists the differences in the components and component parameter used in the two Jobs. A minus sign appended on top of a component listed in the <b>Compare Result</b> view indicates that this component is missing in the design of one of the two compared Jobs. A plus sign appended on top of a component listed in the view indicates that this component is added in one of the two compared Jobs. All differences in the component parameters will be listed in tables that display under the corresponding component.
<b>Connectors</b>	lists differences in all the links used to connect components in the two Jobs.

The procedure to compare two Jobs or two different versions of the same Job are the same.

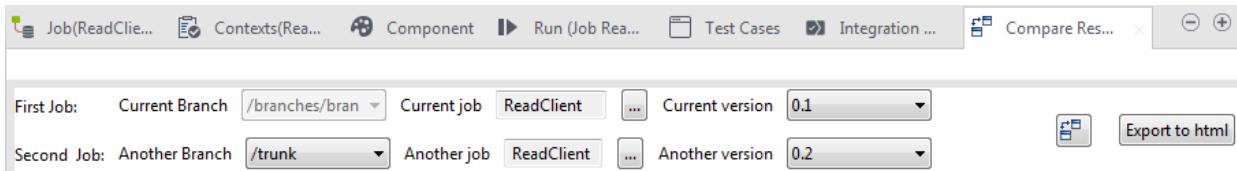
To compare two different versions of the same Job, complete the following:

## Procedure

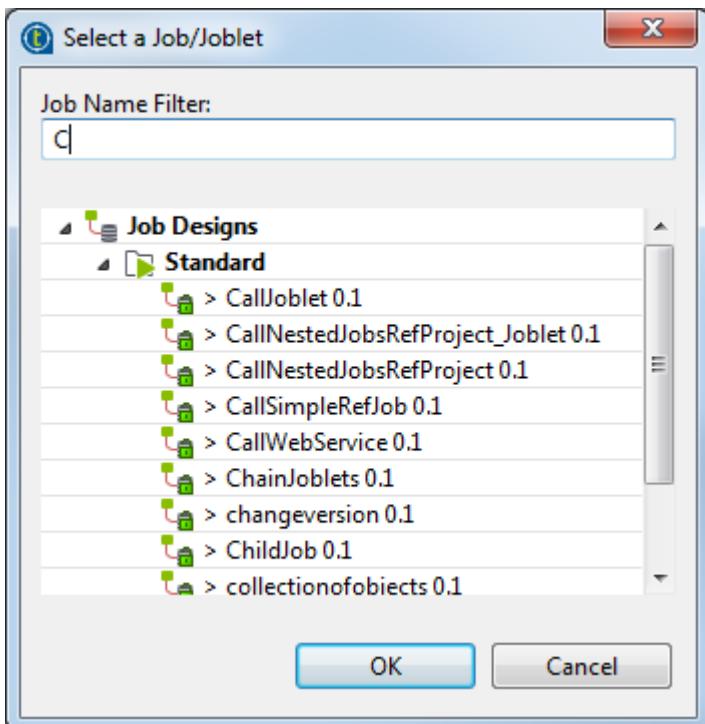
- In the **Repository** tree view, right-click the Job version you want to compare with another version of the same Job and then select **Compare Job** from the contextual menu.



The **Compare Result** view displays in the Studio workspace. The selected Job name and version show, by default, in the corresponding fields.



- If the other version of the Job with which you want to compare the current version is on another branch, select the branch from the **Another Branch** list.
- Click the three-dot button next to the **Another job** field to open the Select a Job/Joblet dialog box.

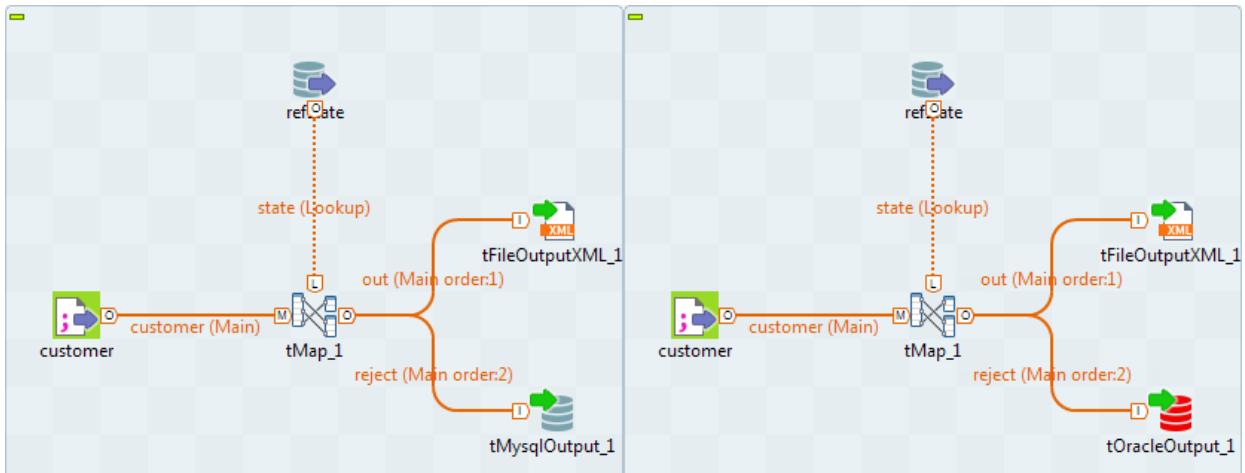


- In the **Name Filter** field, type in the name of the Job or Joblet you want to use for this comparison. The dialog box returns you this Job or Joblet you are searching for.
- Select the returned Job or Joblet from the list in the dialog box and click **OK**.
- From the **Current version** and **Another version** lists select the Job versions you want to compare.

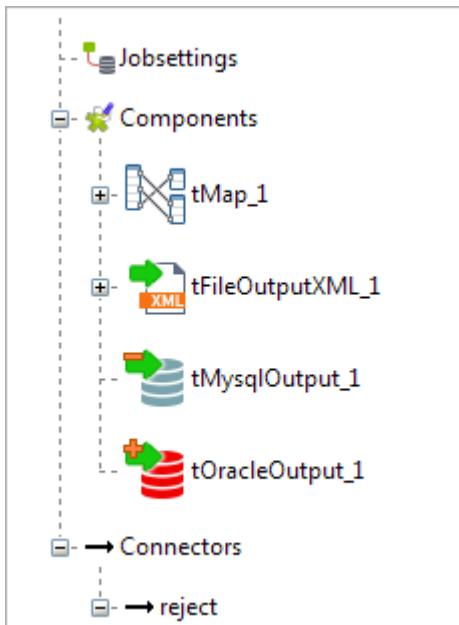
7.

Click the  button to launch the compare operation.

The two indicated versions of the Job display in the design workspace.

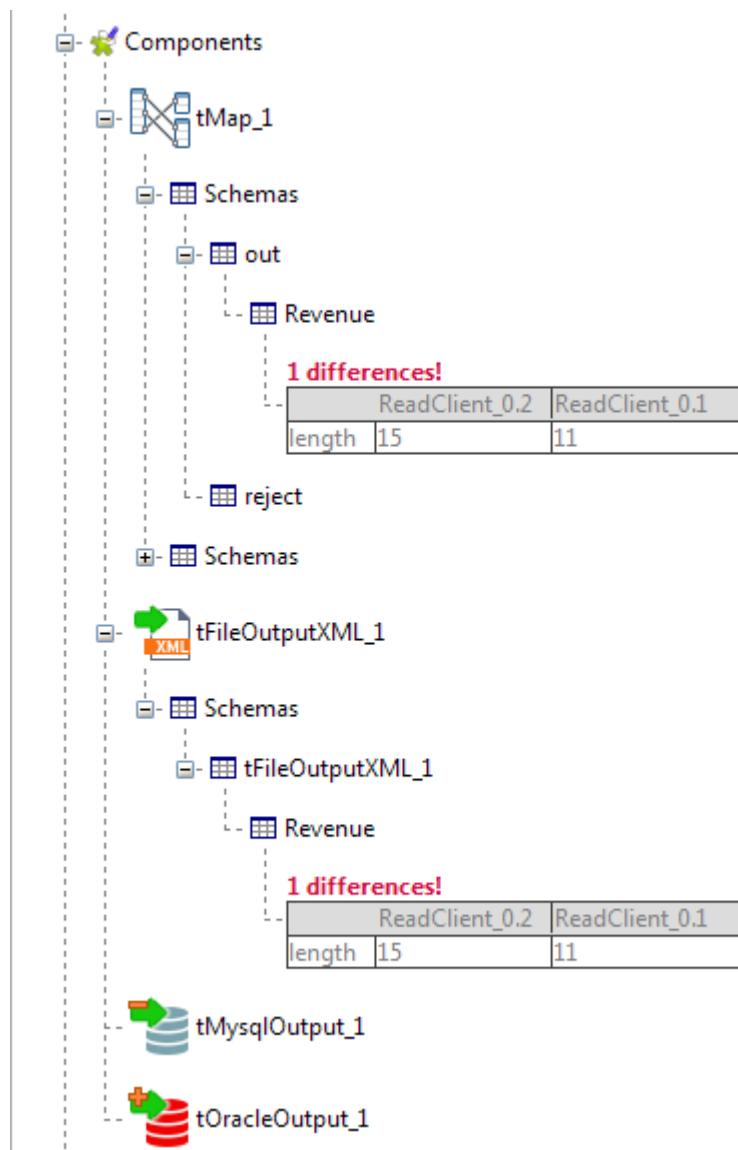


The differences between the two versions are listed in the **Compare Result** view



## Results

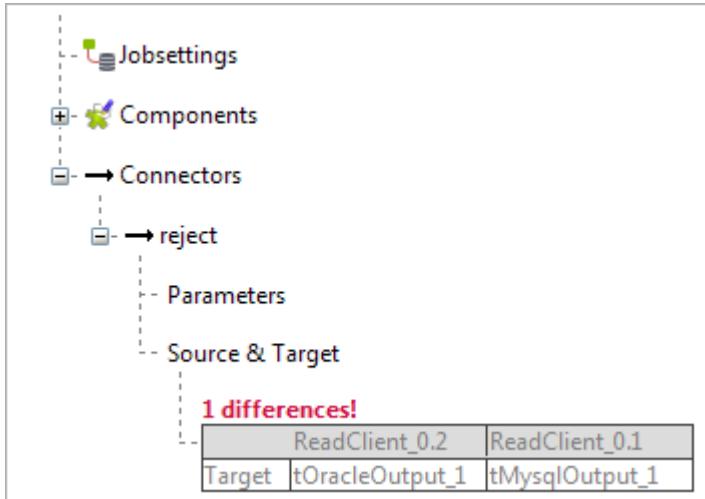
In this example, differences between the two Job versions are related to components and links (connectors). The figure below shows the differences in the components used in the two versions.



For example, there is one difference in the output schemas used in the **tMap** and **tFileOutputXML** components: the length of the Revenue column is 15 in the second version of the Job while the length is 11 in the first version of the same Job. The minus sign appended on top of **tMysqlOutput** indicates that this component is missing in the design of one of the two compared Jobs. The plus sign appended on top of **tOracleOutput** indicates that this component is added in one of the two compared Jobs.

**Note:** If you click any of the components listed in the **Compare Result** view, the component will be automatically selected, and thus identified, in the open Job in the design workspace.

The figure below shows the differences in the links used to link the components in the two versions of the same Job.



In this example, there is one difference related to the **reject** link used in the two versions: the target of this link in the first version is a **tMysqlOutput** component, while it is a **tOracleOutput** component in the second version.

**Note:** You can export the Job compare results to an html file by clicking **Export to html**. Then browse to the directory you want to save the file in and enter a file name. You have the option of using a default CSS template or a customized one. The destination folder will contain the html file, a css file, an xml file and a pictures folder. For related topic, see [Exporting the results of impact analysis/data lineage to HTML](#).

## Running Jobs

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:

- [Running a Job in normal mode](#) on page 120
- [Running a Job in Java Debug mode](#) on page 121
- [Running a Job in Traces Debug mode](#) on page 122
- [Setting advanced execution settings](#) on page 124
- [Showing JVM resource usage during Job execution](#) on page 127
- [Deploying a Job on SpagoBI server \(deprecated\)](#) on page 128

### Running a Job in normal mode

**Note:**

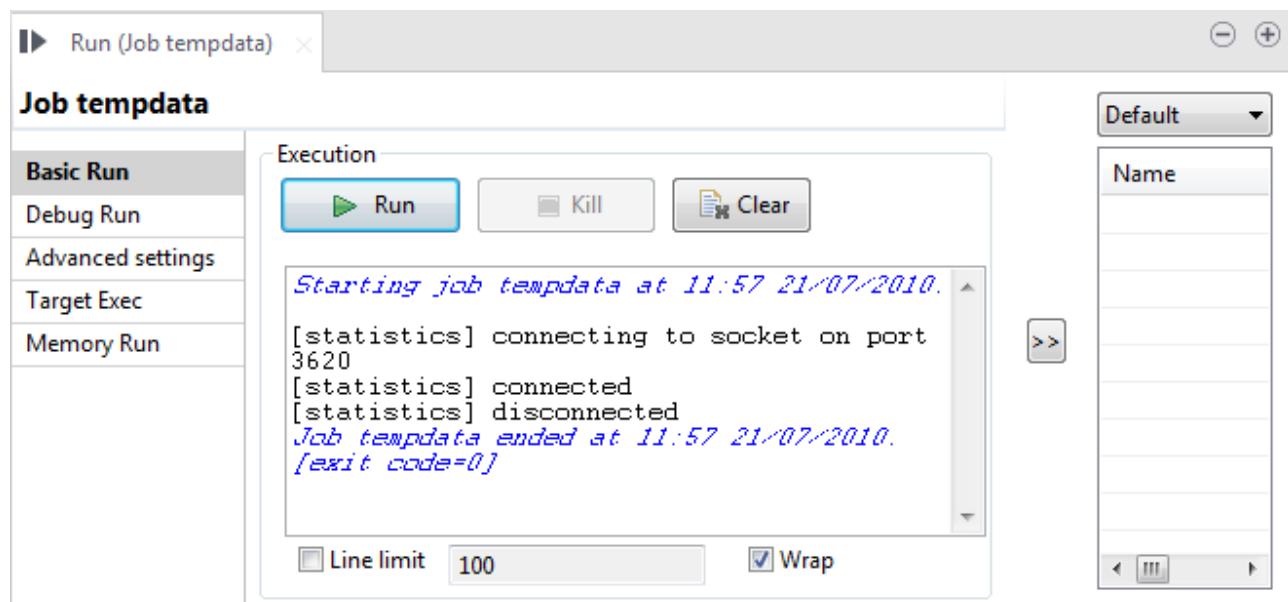
Make sure you saved your **Job** before running it in order for all properties to be taken into account.

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](#) on page 66.

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.
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.

## [Running a Job in Java Debug mode](#)

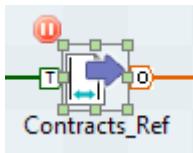
### [About this task](#)

To follow step by step the execution of a Job to identify possible bugs, you can run it in Debug mode.

Before running your Job in Debug mode, you can 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 access the Debug mode:

### Procedure

1. Click the **Run** view to access it.
2. Click the **Debug Run** tab to access the debug execution modes.  
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.

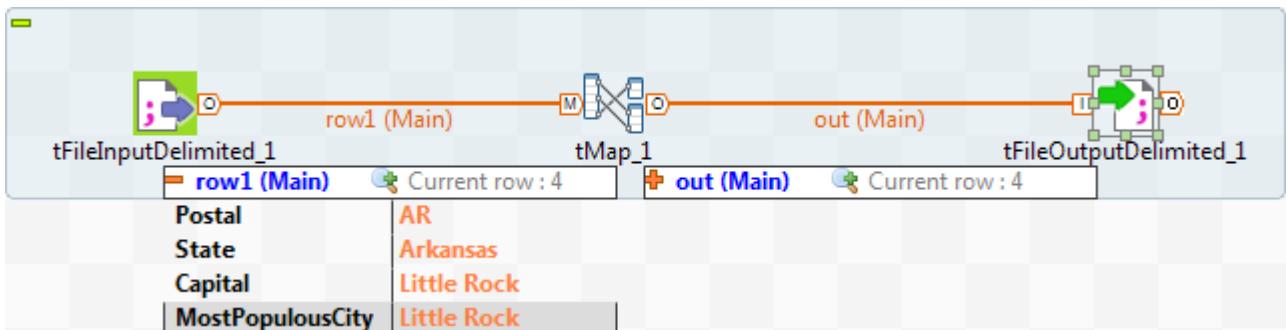
### Results

You can then run the Job step by step and, if you have added breakpoints, check each breakpoint component for the expected behavior and variable values.

### Running 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.

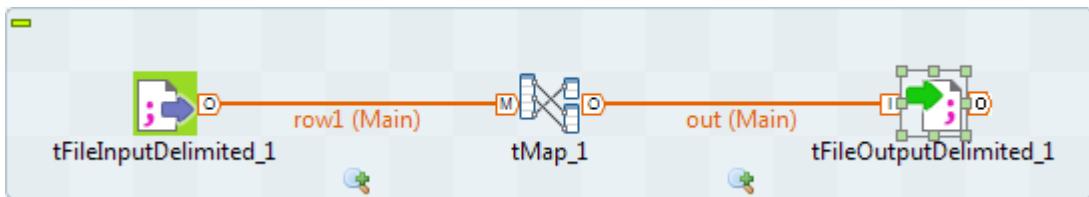
The **Traces** function displays the content of processed rows in a table.

#### Note:

Exception is made for external components which cannot offer this feature if their design does not include it.

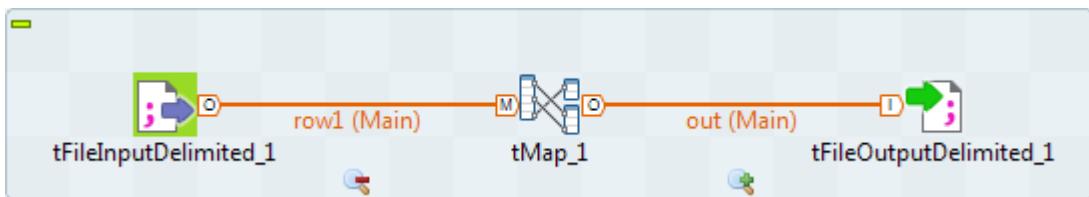
You can activate or deactivate **Traces** or decide what processed columns to display in the traces table that displays on the design workspace when launching the current Job.

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.

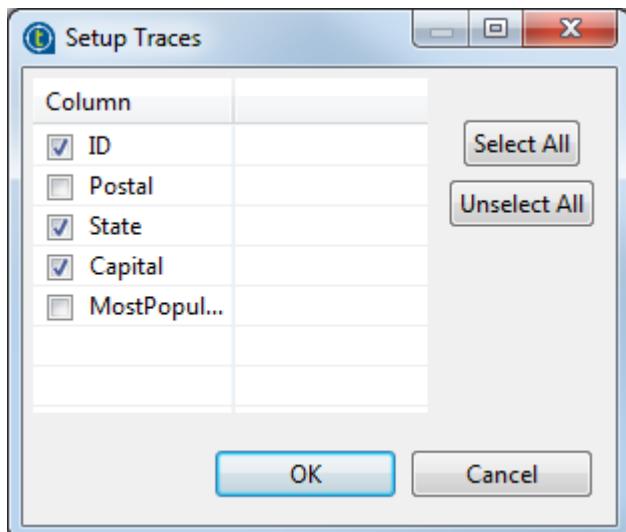
To deactivate the **Traces** on one of the flows in your Job:



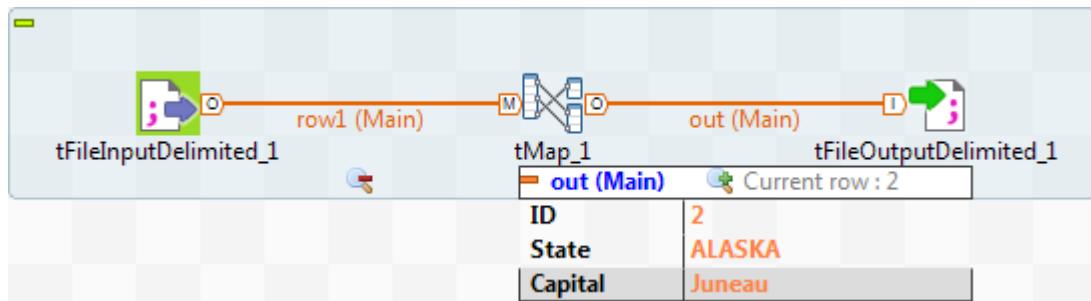
1. Right-click the **Traces** icon under the relevant flow.
2. Select **Disable Traces** from the list. A red minus sign replaces the green plus sign on the icon to indicate that the **Traces** mode has been deactivated for this flow.

To choose which columns of the processed data to display in the traces table, do the following:

1. Right-click the **Traces** icon for the relevant flow, then select **Setup Traces** from the list. The **Setup Traces** dialog box appears.



2. In the dialog box, clear the check boxes corresponding to the columns you do not want to display in the Traces table.
3. Click **OK** to close the dialog box.



Monitoring data processing starts when you execute the Job and stops at the end of the execution.

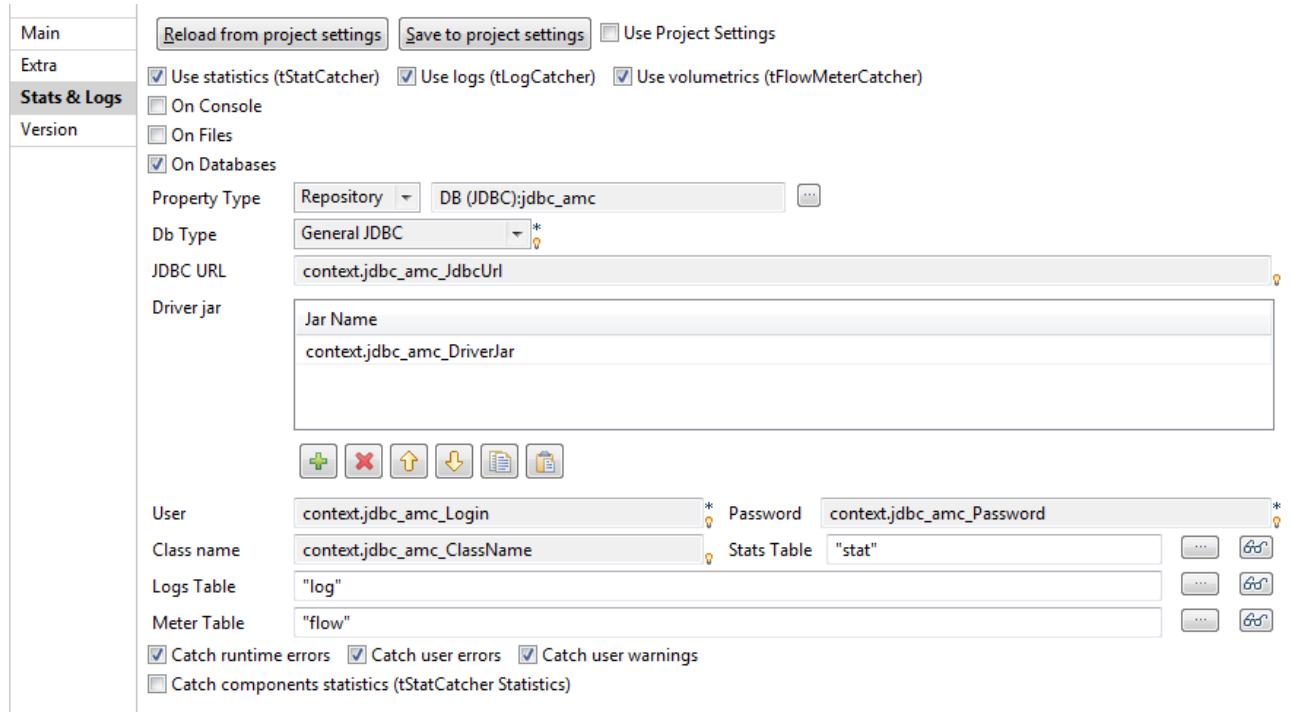
To remove the displayed monitoring information, click the **Clear** button in the **Debug Run** tab.

### Setting 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.

### Displaying 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.

**Note:** Exception is made for external components which cannot offer this feature if their design does not include it.

## Procedure

- 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.

**Note:** The statistics thread slows down Job execution as the Job must send these stats data to the design workspace in order to be displayed.

- 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.

## Displaying the execution time and other options

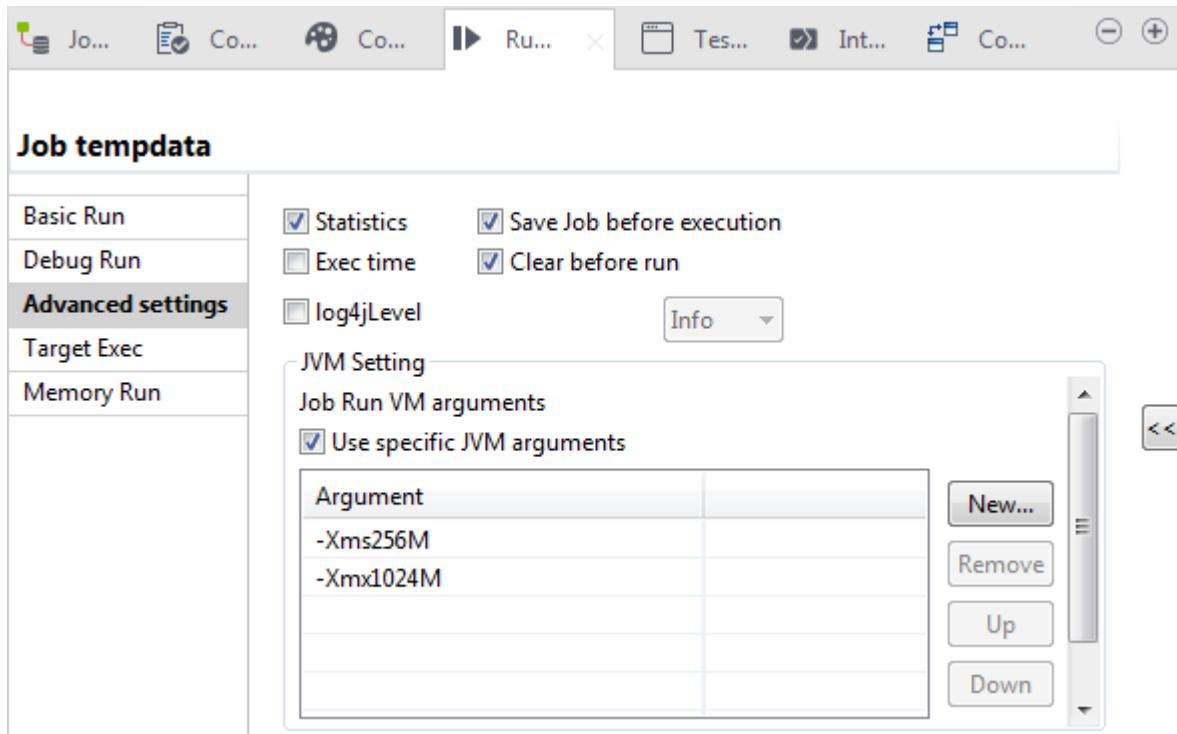
### Procedure

- To display the total execution time, 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.
- To clear the design workspace before each Job execution, select the check box **Clear before Run**.
- To save your Job before the execution starts, select the relevant option check box.

## Displaying special characters in the console

### About this task

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:



## Procedure

- 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.

### Specifying the limits of VM memory for a Job

In Talend Studio, you can define the parameters of your current JVM before executing your Job according to your needs.

The default parameters `-Xms256M` and `-Xmx1024M` correspond respectively to the minimal and maximal memory sizes reserved for your Job executions. Edit these parameters to meet your specific needs.

To specify these parameters globally in Talend Studio, see [Debug and Job execution preferences \(Talend > Run/Debug\) on page 415](#).

### Procedure

1. In the **Run** view, in the **Advanced settings** tab, select the **Use specific JVM arguments** check box.
2. Click the **New** button and then, in the **Set the VM Argument** dialog box that opens, enter the argument to use.  
For example, to successfully execute a Job that handles an Excel file containing a million records, you may want to specify `-Xmx8192M` to increase the maximum VM memory size to 8 GB.
3. Click **OK** to add the argument.

### Customizing log4j output level at runtime

#### About this task

When activated in components, the Apache logging utility log4j outputs component-related logging information at runtime. By default, all logging messages of or higher than the level defined in the log4j configuration will be output to the defined target.

You can change the logging output level for an execution of your Job. To do so, take the following steps:

### Procedure

1. In the **Run** view, click the **Advanced settings** tab.
2. Select the **log4jLevel** check box, and select the desired output level from the drop-down list.  
This check box is displayed only when log4j is activated in components.  
For more information on the logging output levels, see Apache documentation at <http://logging.apache.org/log4j/1.2/apidocs/org/apache/log4j/Level.html>.
3. Run your Job.

### Results

All the logging messages of and higher than the level you set are output to the defined target.

For information on how to activate log4j in components and how to configure the logging behaviors globally, see [Configuring Log4j](#) on page 388.

For more information regarding the components with which you can use the log4j feature, see *List of components that support the Log4j feature* on Talend Help Center (<https://help.talend.com>)..

## Showing JVM resource usage during Job execution

### About this task

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:

### Procedure

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.

## Deploying a Job on SpagoBI server (deprecated)

This feature is deprecated from Talend 7.1 onwards.

From Talend Studio interface, you can deploy your Jobs easily on a SpagoBI server in order to execute them from your SpagoBI administrator.

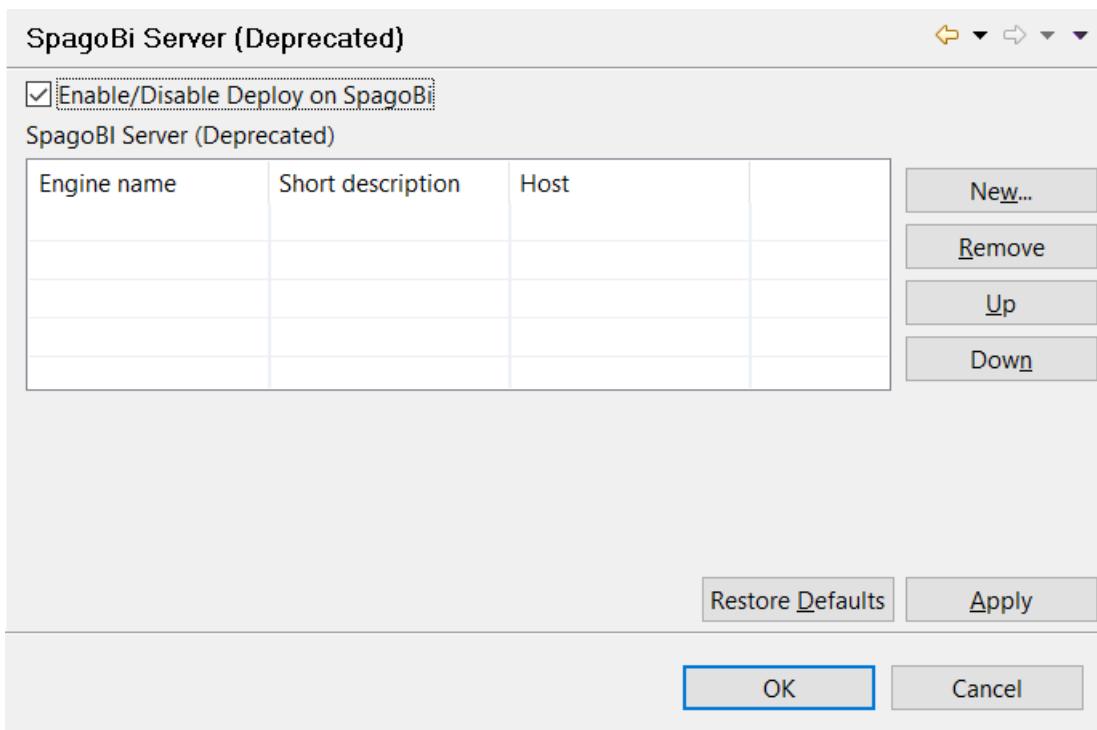
### Creating a SpagoBI server entry

#### About this task

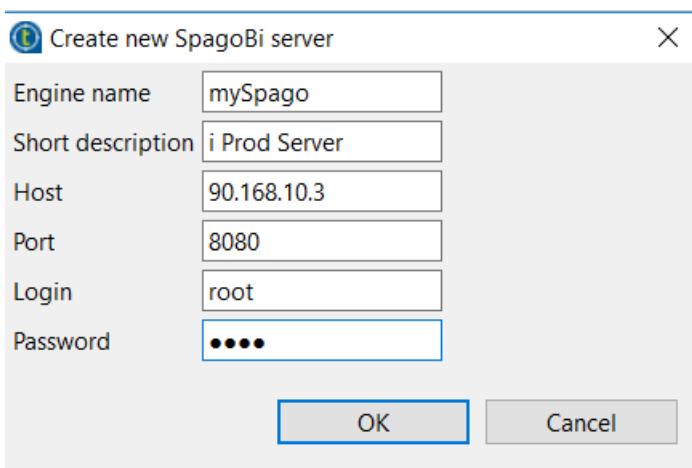
Beforehand, you need to set up your single or multiple SpagoBI server details in Talend Studio.

#### Procedure

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 (Deprecated)** to display the relevant view.



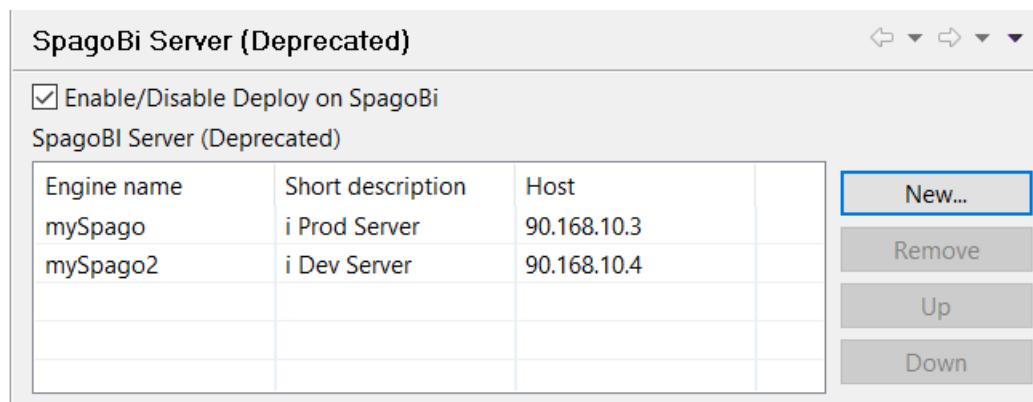
3. Select the **Enable/Disable Deploy on SpagoBI** check box to activate the deployment operation.
4. Click **New** to open the **Create new SpagoBI server** dialog box and add a new server to the list.



5. Enter your SpagoBI server details, as described below:

Field	Description
Engine Name	Internal engine name used in Talend Studio. 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.

7. Click **Apply** and then **OK** to close the **Preferences** dialog box.

### Editing or remove a SpagoBI server entry

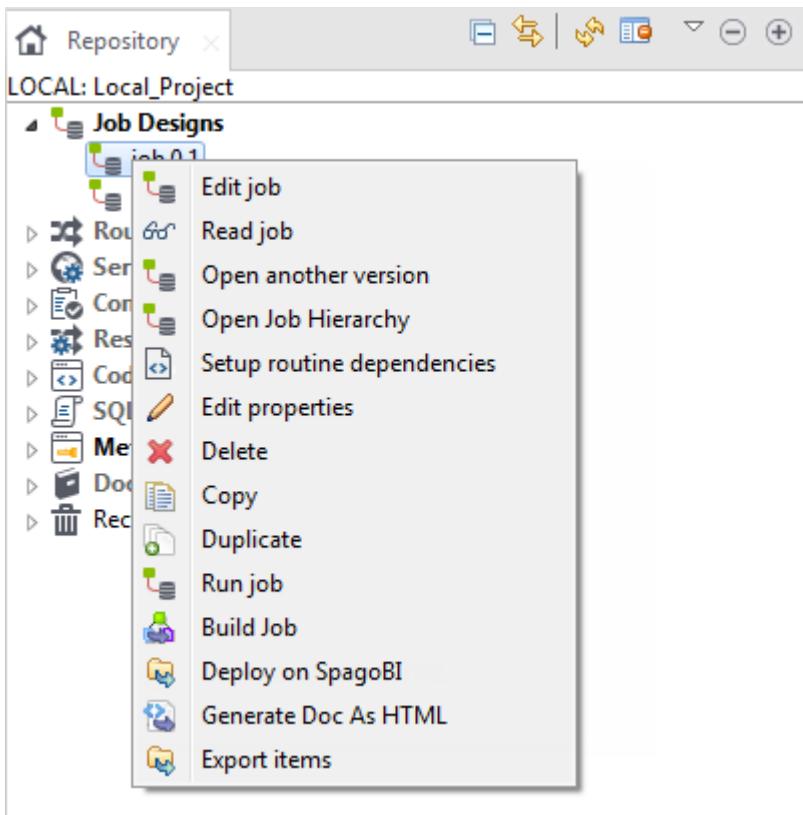
#### Procedure

1. Select the relevant entry in the table, click the **Remove** button next to the table to first delete the outdated entry.
2. Then if required, simply create a new entry including the updated details.

### Deploying your Jobs on a SpagoBI server

#### About this task

Follow the steps below to deploy your Job(s) onto a SpagoBI server.



## Procedure

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** to open the **Deploy on SpagoBI** dialog box.
3. Select the relevant **SpagoBI server** on the drop-down list.
4. The **Label**, **Name** and **Description** fields come from the Job main properties.
5. Select the relevant context in the list.
6. Click **Finish** once you have completed the setting operation.

## Results

The Jobs are now deployed onto the relevant SpagoBI server. Open your SpagoBI administrator to execute your Jobs.

## 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:

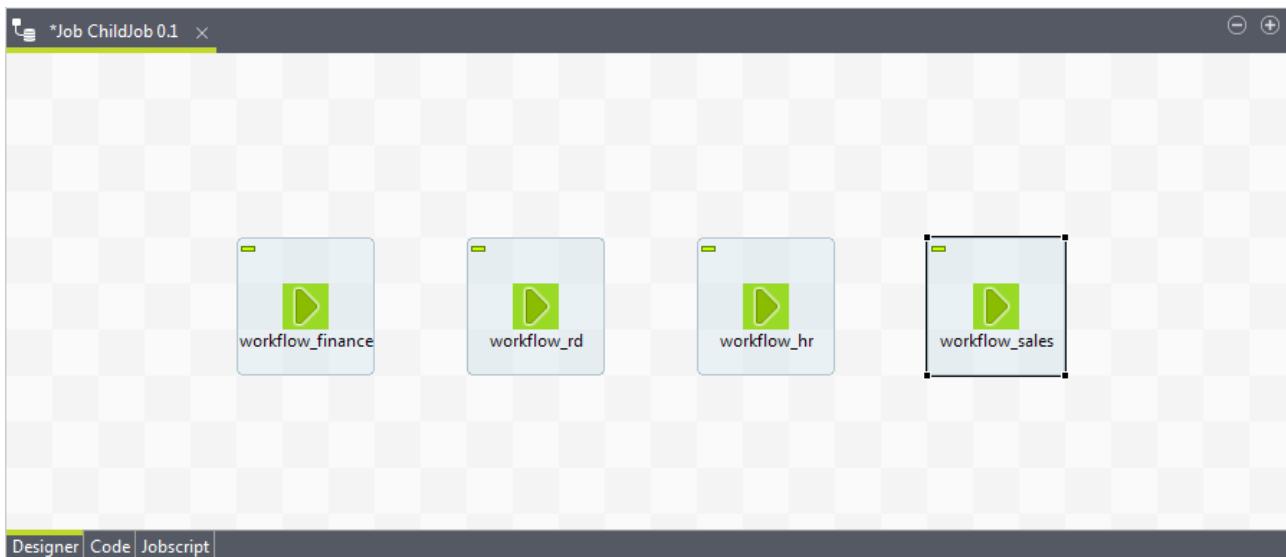
1. Parallel executions of multiple subJobs. For further information, see [Executing multiple subJobs in parallel](#) on page 131.
2. Parallel iterations for reading data. For further information, see [Launching parallel iterations to read data](#) on page 132.

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 [What is a Job design?](#) on page 25.

## Executing 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.

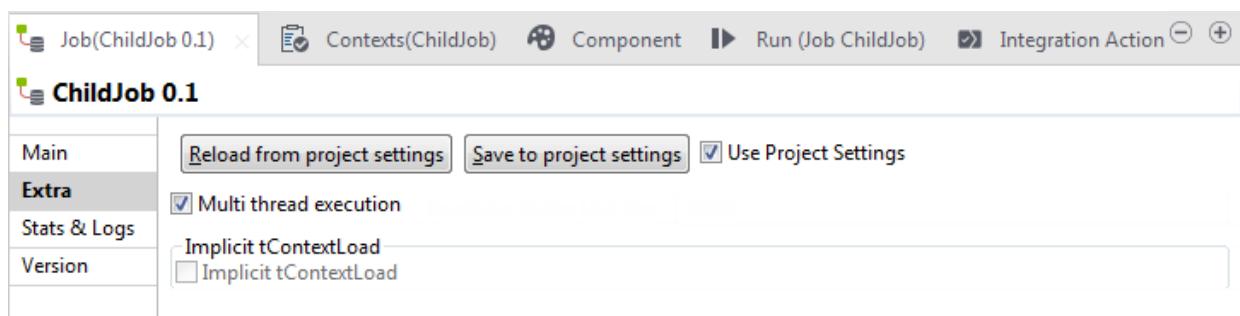


The **tRunJob** component is used in this example to call each subJob they represent.

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

### Procedure

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.

This feature is optimal when the number of threads (in general a subJob counts 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.

## Launching 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.

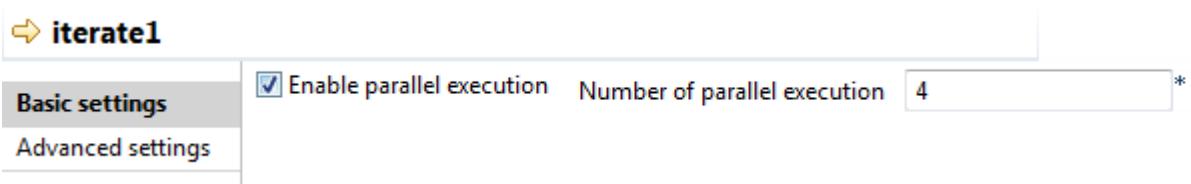
**Warning:** Note that `globalMap` is thread unsafe. Be cautious when using `globalMap.put("key", "value")` and `globalMap.get("key")` to create your own global variables and then retrieve their values in your Jobs, especially after an **Iterate** connection with the parallel execution option enabled.

### About this task

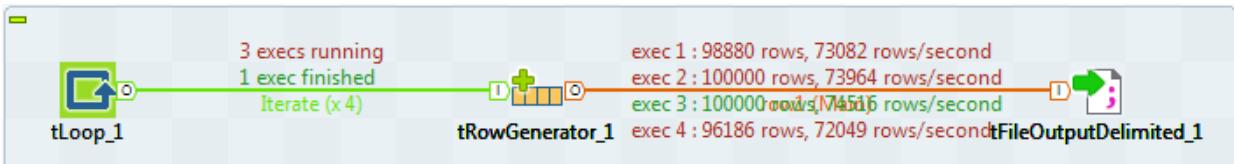
You need to proceed as follows to set the parallel iterations:

#### Procedure

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.

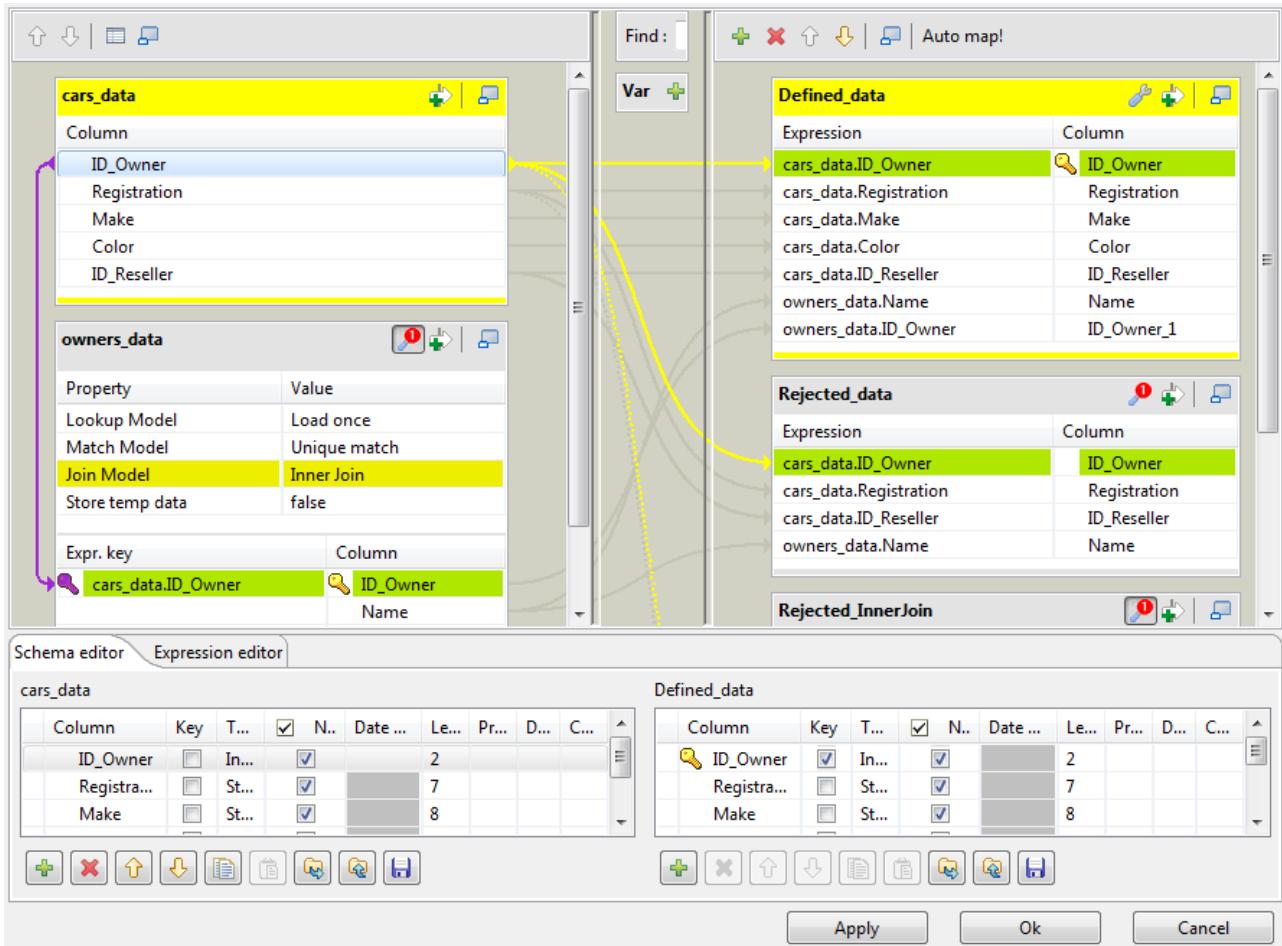
## Mapping data flows

### Map editor interfaces

The most common way to handle multiple input and output flows including transformations and data re-routing is to use dedicated mapping components.

Mapping components are advanced components which require more detailed explanation than other Talend 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](#) on page 169.

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 allows 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).

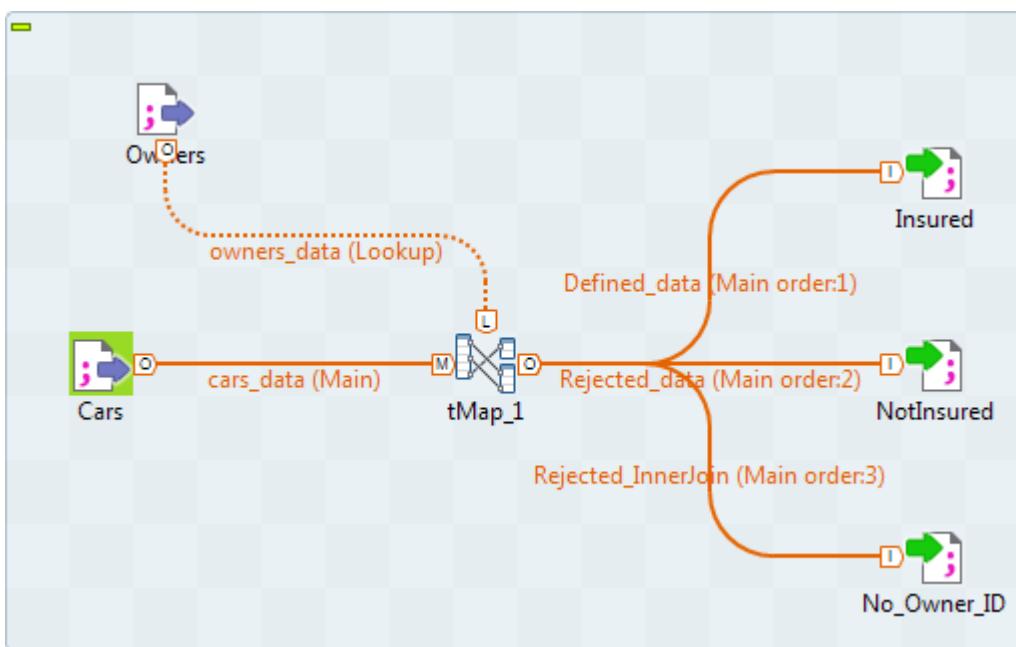
The following sections present separately different mapping components of which each is able to map flows of a specific nature.

## 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.



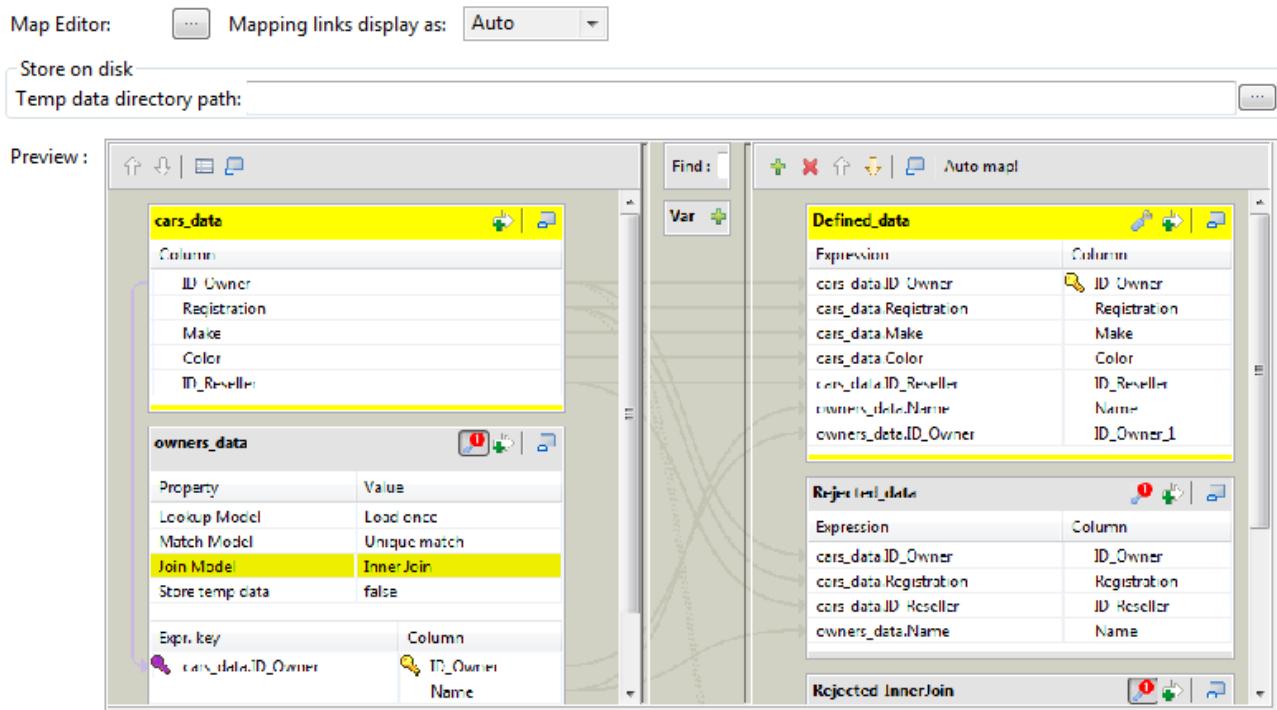
**tMap** uses incoming connections to pre-fill input schemas with data in the **Map Editor**. Therefore, you cannot create new input schemas directly in the **Map Editor**. Instead, you need to implement as many **Row** connections incoming to **tMap** component as required, in order to create as many 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](#) on page 60.

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.

### 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.

**Owners**

Column

- ID\_Owner
- Name
- ID\_Insurance
- Children\_Nr

Owners.Children\_Nr >= 2 && Owners.Children\_Nr <= 6

**Cars**

Property Value

- Lookup Model Load once
- Match Model All matches
- Join Model Inner Join
- Store temp data false

Expr. key	Column
Owners.ID_Owner	ID_Owner
	Registration
"BMW"	...
	Make
	Color
	ID_Reseller

**Resellers**

Property Value

- Lookup Model Load once
- Match Model All matches
- Join Model Inner Join
- Store temp data false

Expr. key	Column
Cars.ID_Reseller	ID_Reseller
	Name_Resel...
	Address_Res...
	City
	ZIP

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: [Using Explicit Join](#) on page 138.

## Filling 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 [Defining component properties](#) on page 39.

For more information about setting an input schema in the **Map Editor**, see [Setting schemas in the Map Editor](#) on page 157.

### 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.

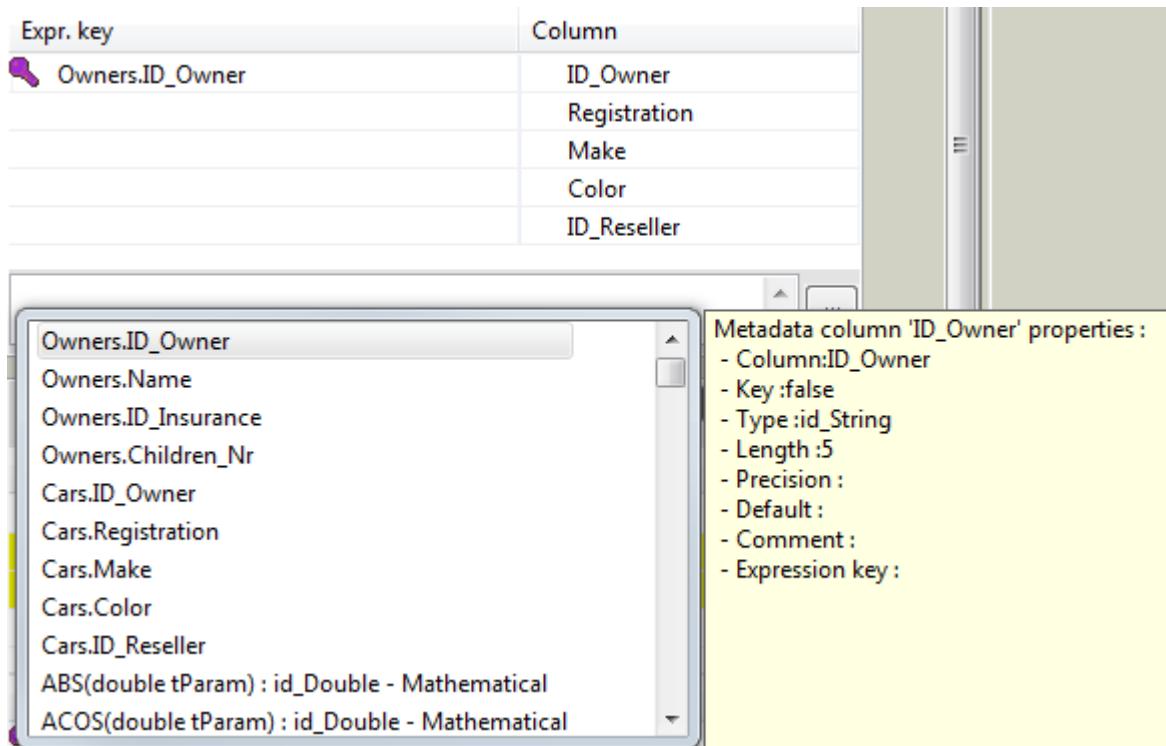
The **Lookup** connections' content fills in all other (secondary or subordinate) tables which displays below the **Main** flow table. If you have not define the schema of an input component yet, the input table displays as empty in the Input area.

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.



Docked at the **Variable** list, a metadata tip box display to provide information about the selected column.

Related topic: [Mapping variables](#) on page 142

### Using 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.

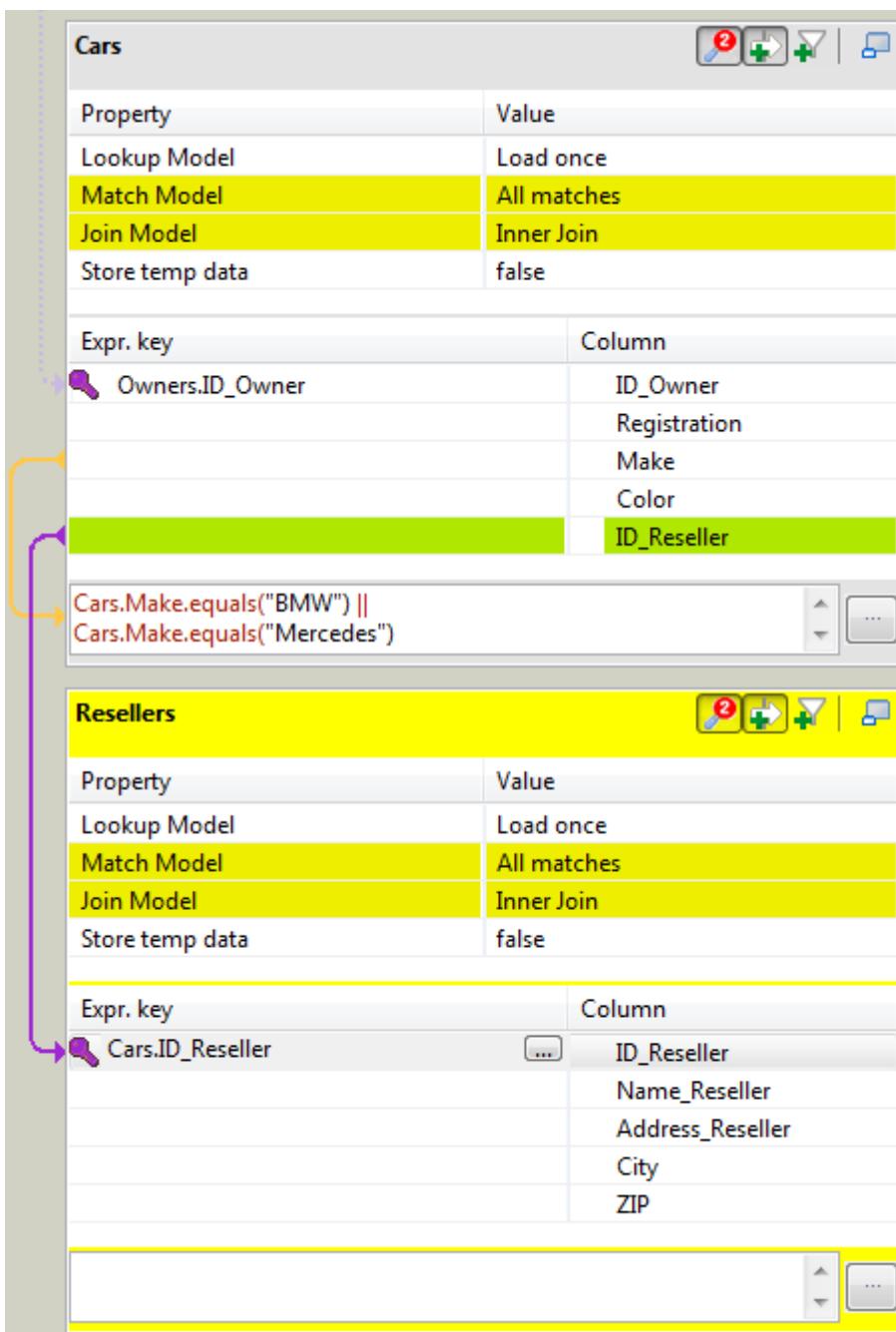
The join displays graphically as a purple link and creates automatically a key that will be used as a hash key to speed up the match search.

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.

**Note:** 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](#) on page 151.

**Note:** 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](#) on page 157
- [Using Inner Join](#) on page 141

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.

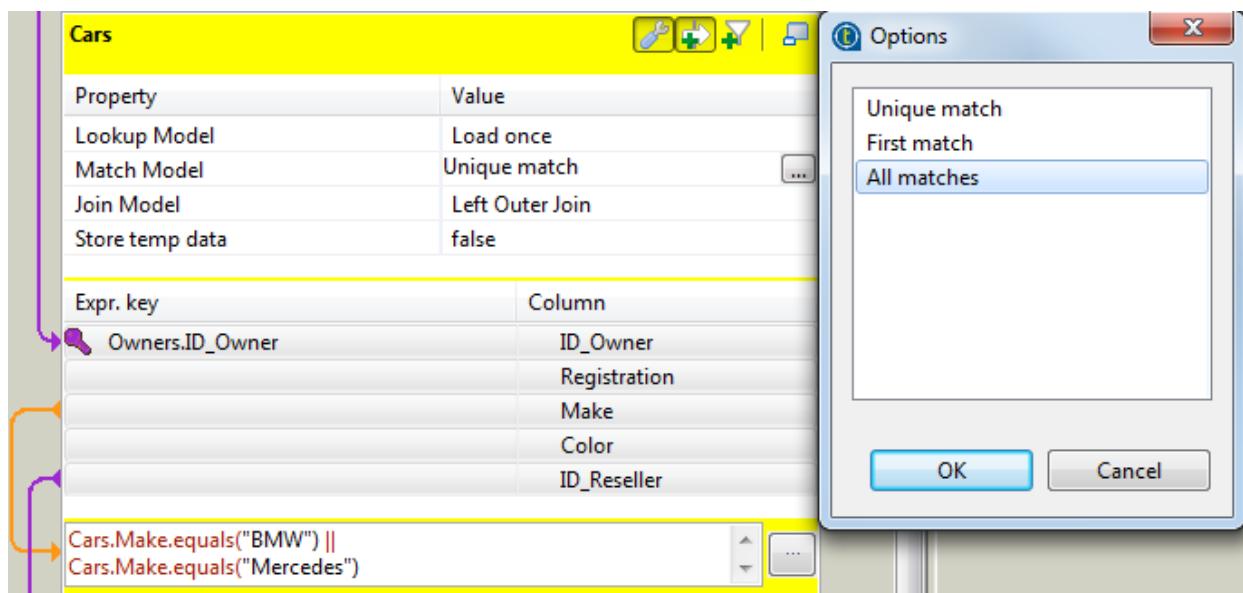
### Defining the match model for an explicit Join

#### Before you begin

To define the match model for an explicit Join:

#### Procedure

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.

## Using 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.

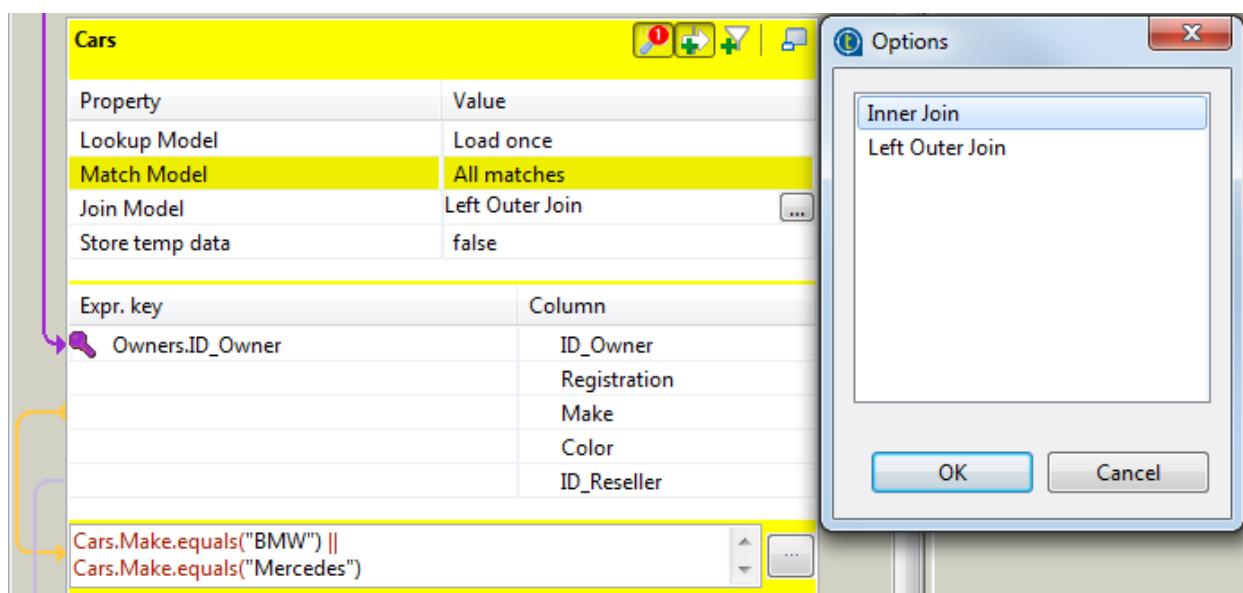
Simply drop column names from one table to a subordinate one, to create a **Join** relationship between the two tables. The Join is displayed graphically as a purple link and creates automatically a key that will be used as a hash key to speed up the match search.

## About this task

To define the type of an explicit Join:

### Procedure

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.



**Note:** 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](#) on page 154.

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](#) on page 154.
- [Filtering an input flow](#) on page 142.

## Using the All Rows option

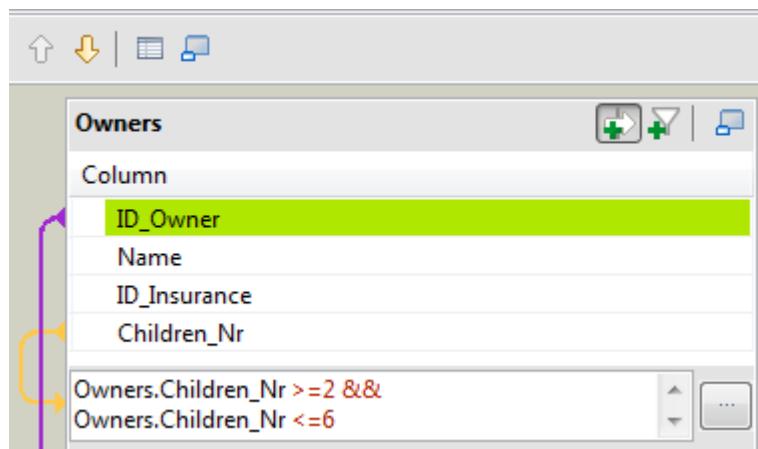
By default, without a Join set up, in each input table of the input area of the **Map Editor**, the **All rows** match model option is selected. This **All rows** option means that all the rows are loaded from the **Lookup** flow and searched against the **Main** flow.

The output corresponds to the Cartesian product of both table (or more tables if need be).

**Note:** 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 [Using Explicit Join](#) on page 138 and [Using Inner Join](#) on page 141.

## Filtering 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** keystrokes in order to reuse schema columns in the condition statement.

## Removing 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.

**Note:** If you remove Input entries from the **Map Editor** schema, this removal also occurs in your component schema definition.

## 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		
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
StringHandling.UPCASE(Cars.Make)	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.

### Accessing 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.

### Removing 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.

## 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.

### Accessing the expression editor

#### About this task

The expression editor provides visual comfort to write any function or transformation in a handy dedicated view.

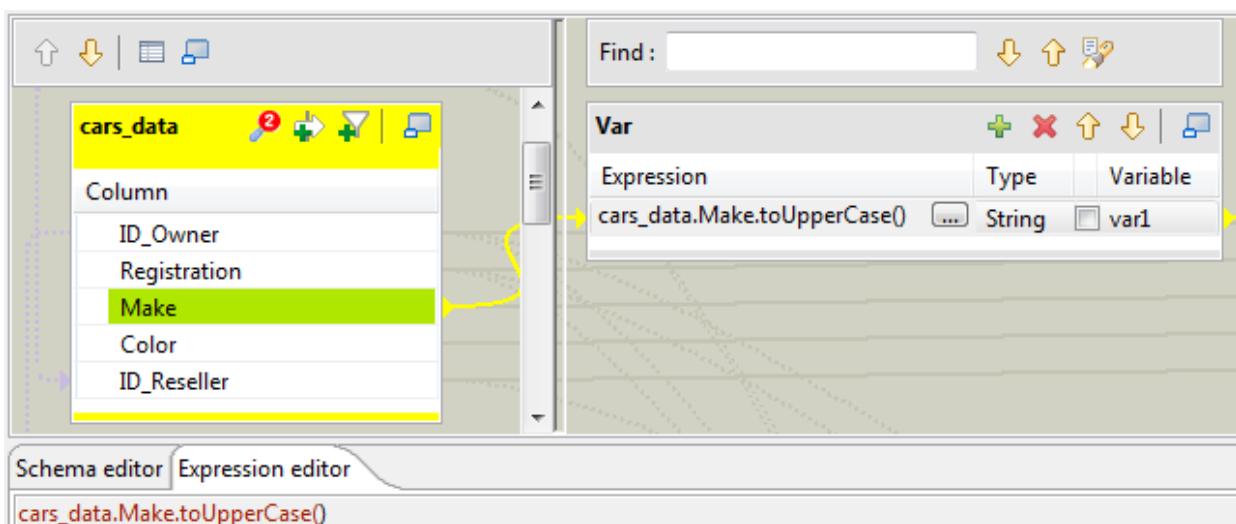
You can write the expressions necessary for the data transformation directly in the **Expression editor** view located in the lower half of the expression editor.

To open the **Expression editor** view, complete the following:

#### Procedure

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.

**Note:** 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.

#### Results

**Note:** Refer to the Java documentation for more information regarding functions and operations.

### Writing code using the Expression Builder

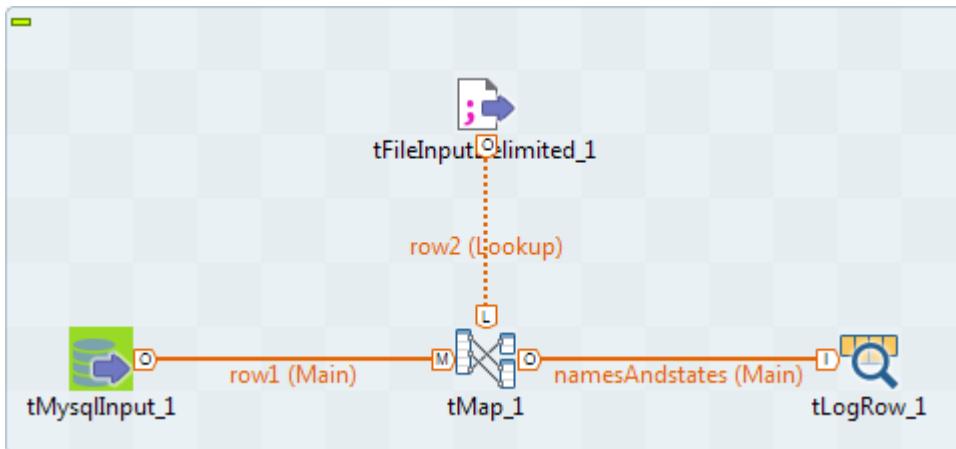
Some Jobs require pieces of code to be written in order to provide components with parameters. In the **Component** view of some components, an **Expression Builder** interface can help you write such pieces of code (in Java), known as expressions.

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

### About this task

The following example shows how to use the **Expression Builder** to edit two individual expressions.



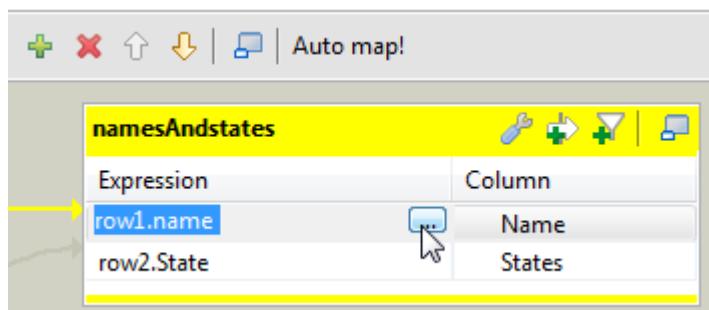
In this example, two input flows are connected to the **tMap** component.

- 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.

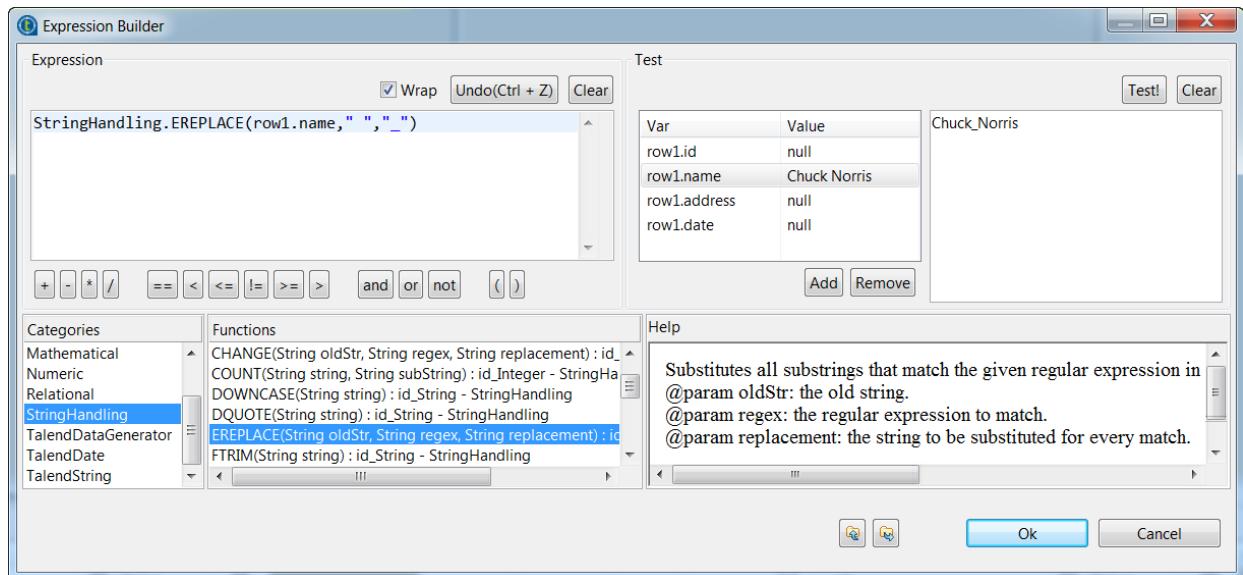
In the **tMap**, use the expression builder to: First, replace the blank char separating the first and last names with an underscore char, and second, change the states from lower case to upper case.

### Procedure

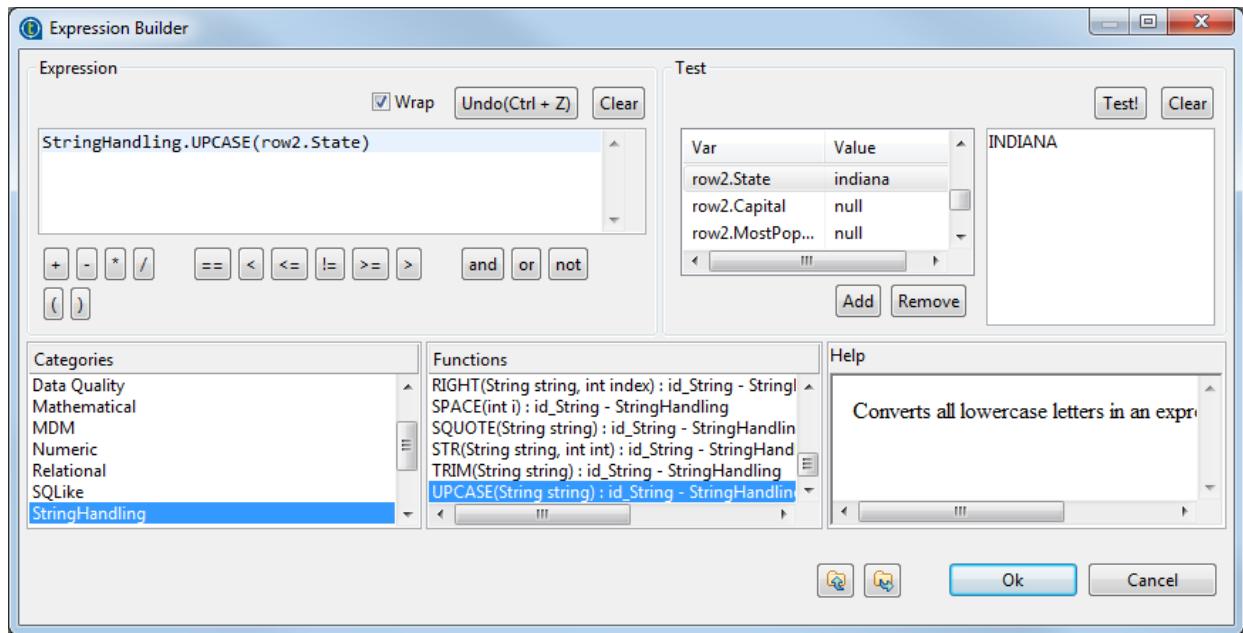
- In the **tMap**, set the relevant inner join to set the reference mapping.  
For more information regarding **tMap**, see [tMap operation](#) on page 134 and [Map editor interfaces](#) on page 132.
- From the main (`row1`) input, drop the Names column to the output area, and the State column from the lookup (`row2`) input towards the same output area.
- 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
StringHandling.EREPLACE(row1.name, " ", "_")	Name
StringHandling.UPCASE(row2.State)	States

## Results

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.*

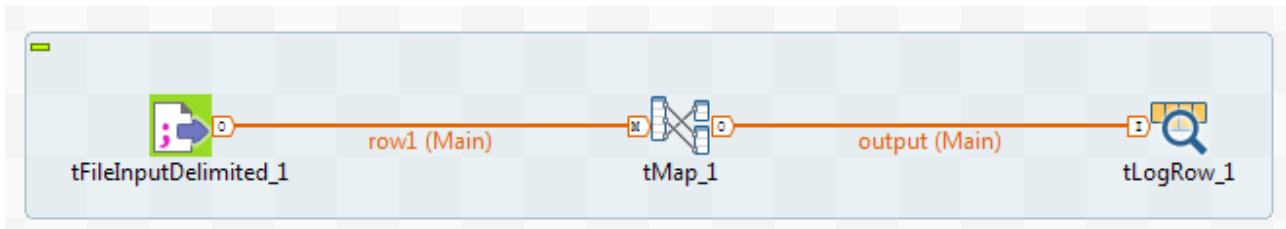
tLogRow_1	
Name	RandomStates
William_Grant	IOWA
William_Hoover	NEW YORK
Grover_Lincoln	NORTH DAKOTA
Lyndon_Jefferson	OHIO
Gerald_Hayes	WASHINGTON
Benjamin_Grant	MAINE
George_Pierce	CONNECTICUT
Jimmy_Reagan	ALASKA
Martin_Hayes	WASHINGTON
Franklin_Jefferson	IOWA
Andrew_Nixon	NEW HAMPSHIRE

## Setting expressions for multiple output columns simultaneously

### About this task

**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.

### Procedure

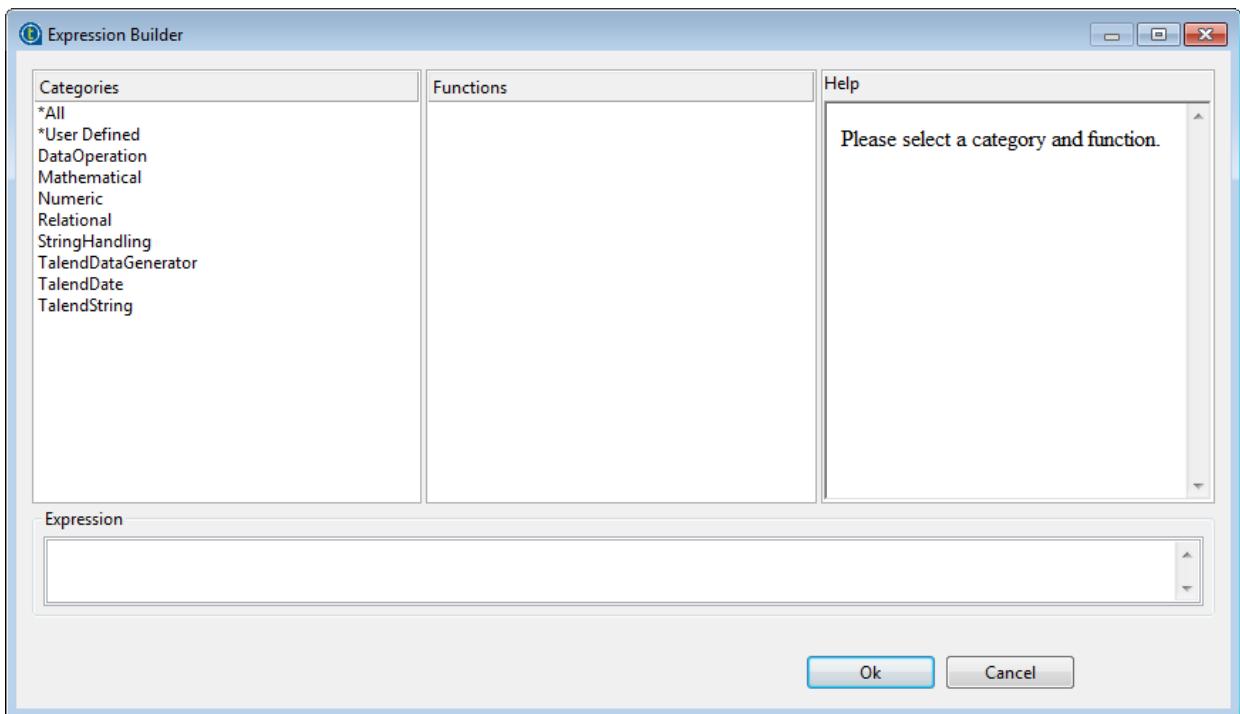
1. In the Map Editor, complete the input-output mappings.

The screenshot shows the Talend Map Editor interface. The top section displays the input mapping ('row1') and output mapping ('output'). The input mapping shows five columns: id, firstname, lastname, city, and state. The output mapping shows five columns: id, firstname, lastname, city, and state. Arrows connect each input column to its corresponding output column. The bottom section shows the schema editors for both 'row1' and 'output'. Both schema editors have five columns: id, firstname, lastname, city, and state. The 'row1' schema editor has 'Key' checked for all columns except 'id'. The 'output' schema editor also has 'Key' checked for all columns except 'id'. Both schema editors show 'String' as the type for all columns except 'id', which is 'Integer'. The 'output' schema editor also includes 'Date Patt...', 'Len...', 'Prec...', and 'De...' columns.

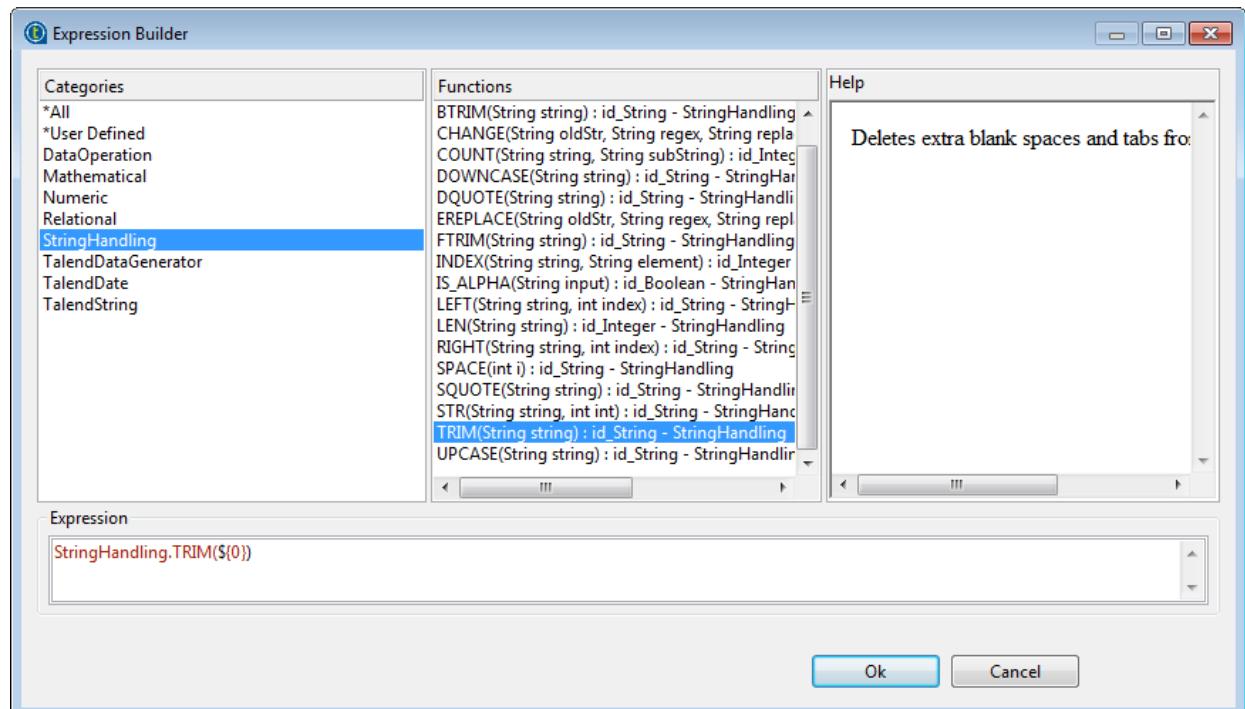
2. Select the columns of type String in the output table, namely `firstname`, `lastname`, `city`, and `state` in this example, and right-click the selection so that the **Apply Routine** button shows up.

output	
Expression	Column
row1.id	id
row1.firstname	firstname
row1.lastname	lastname
row1.city	city
row1.state	state

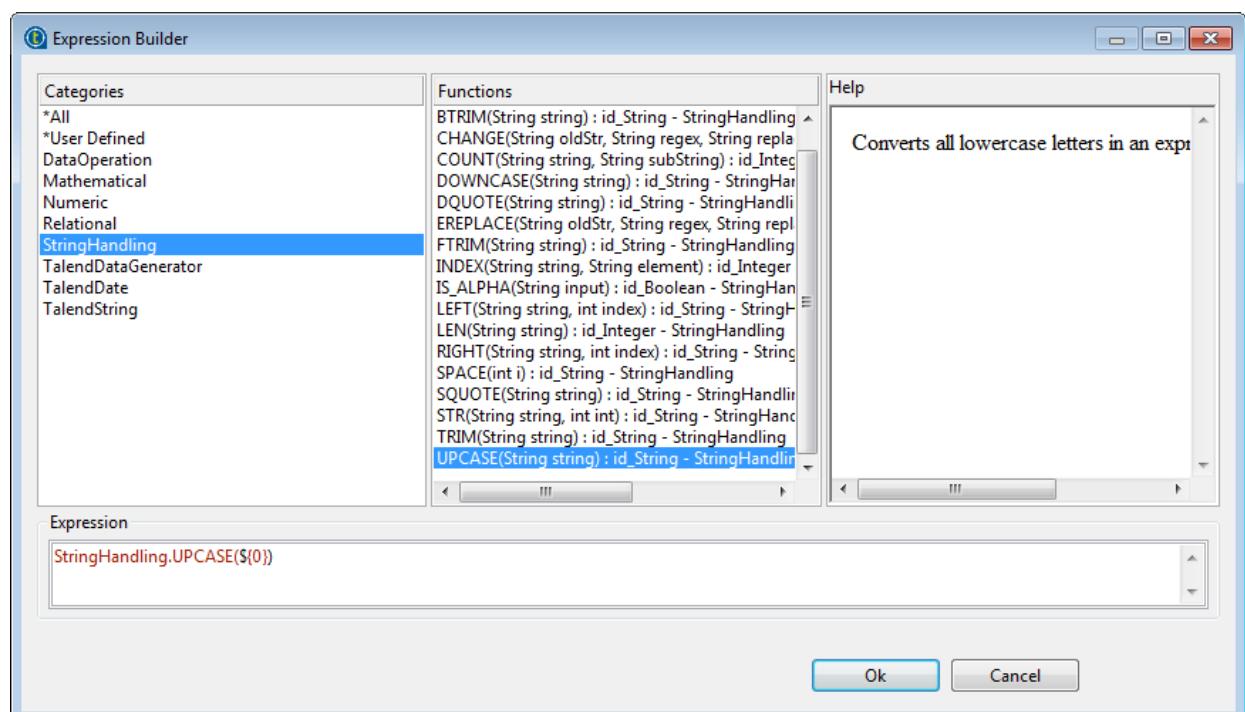
3. Click the **Apply Routine** button to open the **Expression Builder** dialog box.



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.UPPERCASE(\${0})** in the **Expression** field.



8. Click **OK** to close the **Expression Builder** dialog box.

## Results

Now the expressions on those output columns look like below:

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

Run	Kill	Clear
[statistics] connected		
<pre>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 +-----+</pre>		
[statistics] disconnected		

### Mapping the Output setting

**Tip:**

There is no order among the output flows of **tMap**. To make the output flows to be executed one by one, you can output them to temporary files or memory, and then read and insert them into files or databases using different subjobs linked by **Trigger > OnSubjobOK** connections.

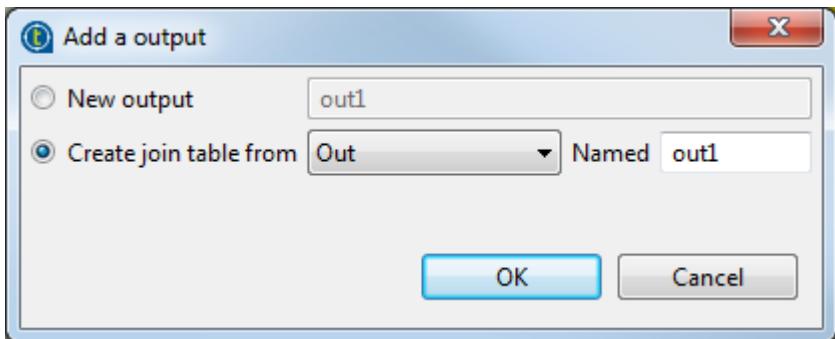
On the design workspace, the creation of a **Row** connection from the **tMap** component to the output components adds Output schema tables in the **Map Editor**.

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.

**Note:** 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...
New output	Add an independent table.
Create join table from	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.

### 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.

## Procedure

1. Click the **Expression** field of your input or output table to display the [...] button.
2. Then click this three-dot button to open the **Expression Builder**.

For more information regarding the Expression Builder, see [Writing code using the Expression Builder](#) on page 144.

## Filters

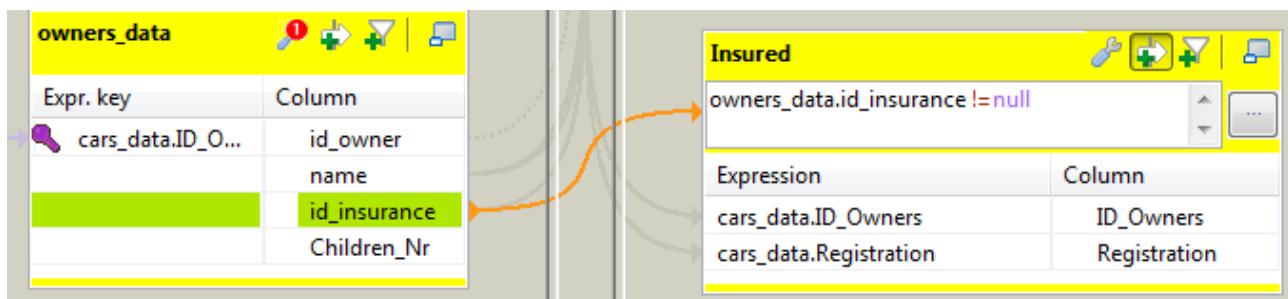
Filters allow you to make a selection among the input fields, and send only the selected fields to various outputs.

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.

## Output rejection

### About this task

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.

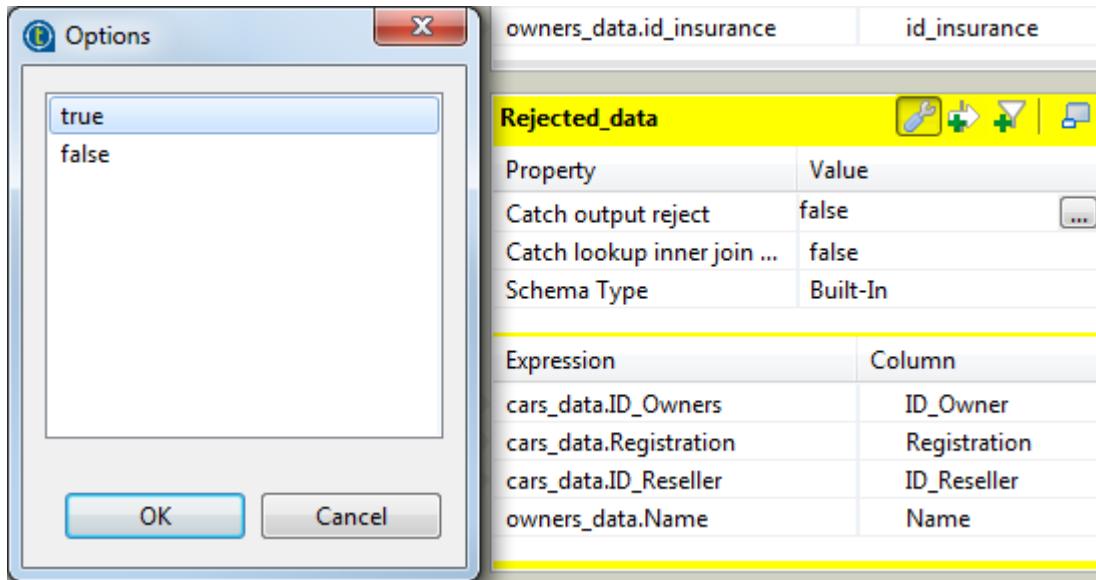
The Reject principle concatenates all non Reject tables filters and defines them as an ELSE statement.

To define an output table as the Else part of the regular tables:

## Procedure

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.



## Results

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.

## Lookup Inner Join rejection

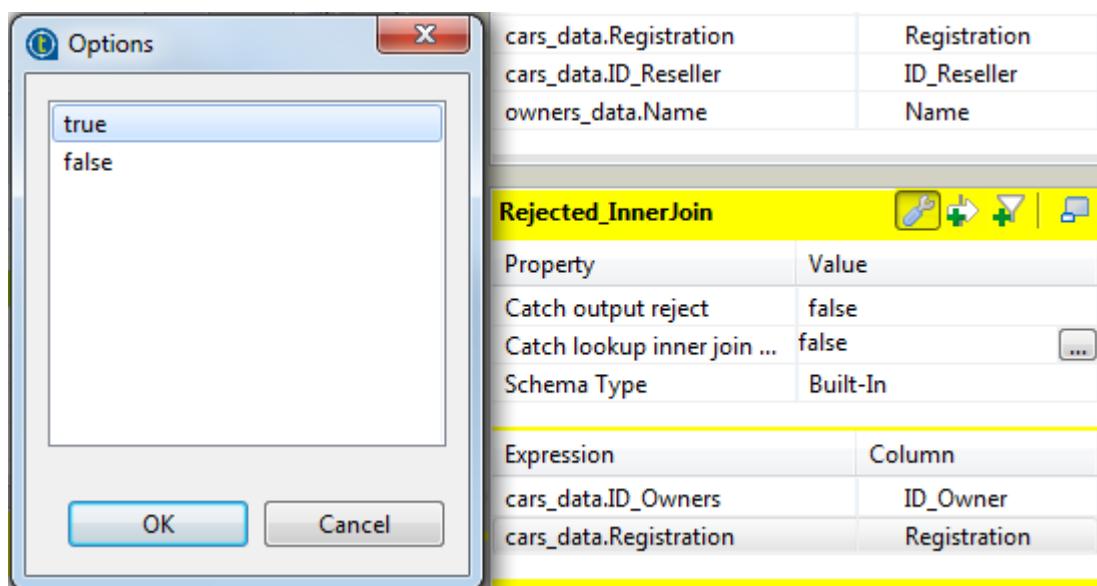
### About this task

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:

### Procedure

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.



### Removing Output entries

To remove Output entries, click the cross sign on the Schema Editor of the selected table.

### Handling errors

#### About this task

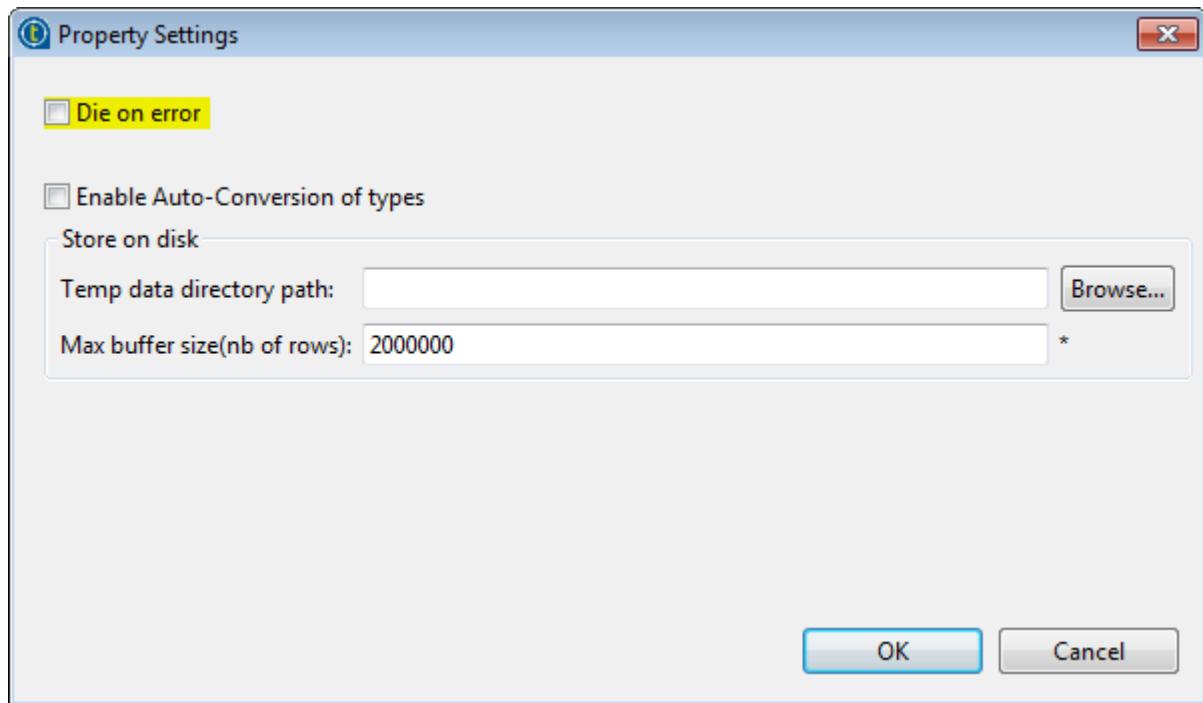
The **Die on error** option prevents error to be processed. To do so, it stops the Job execution as soon as an error is encountered. The **tMap** component provides this option to prevent processing erroneous data. The **Die on error** option is activated by default in **tMap**.

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:

#### Procedure

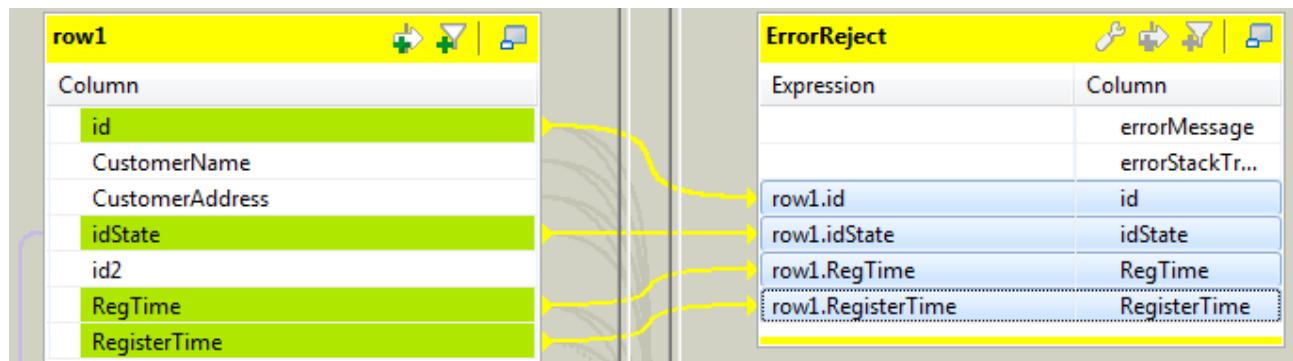
1. Double-click the **tMap** component on the design workspace to open the **Map Editor**.
2. Click the **Property Settings** button at the top of the input area to display the **Property Settings** dialog box.
3. In **Property Settings** dialog box, clear the **Die on error** check box and click **OK**.



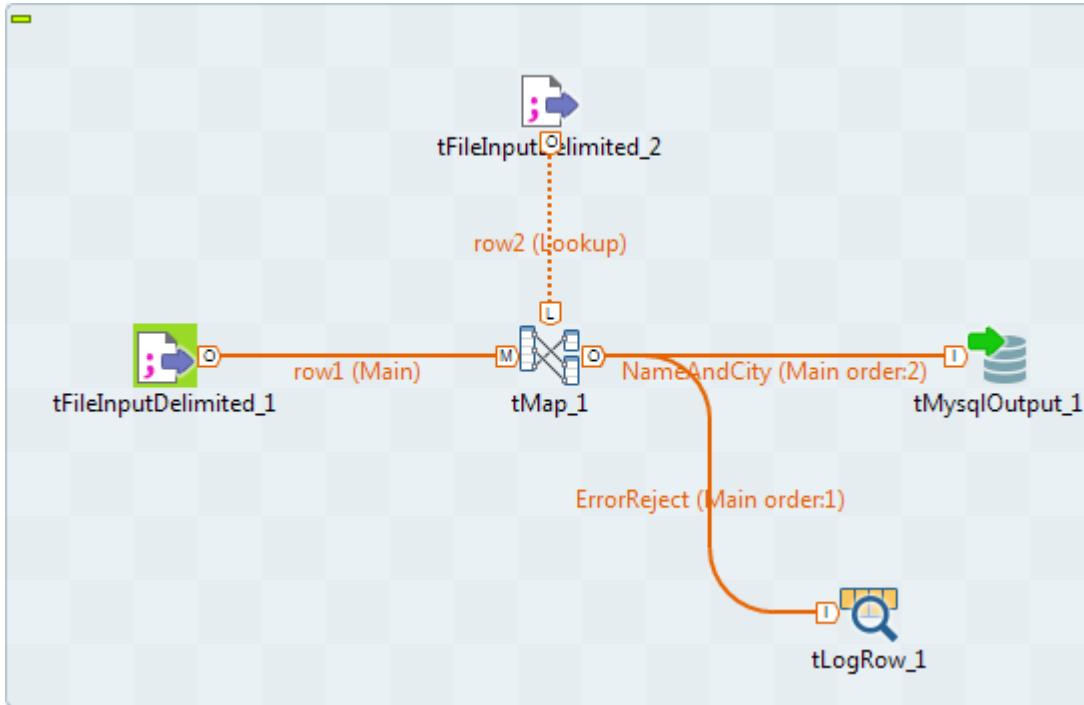
## Results

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 "|".

### 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.

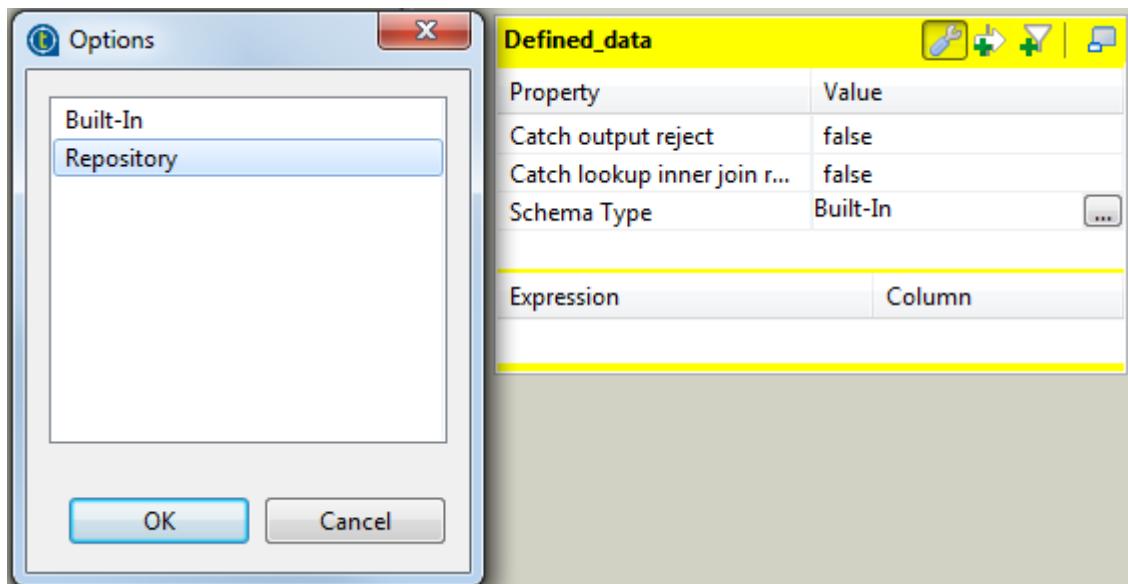
## Retrieving the schema structure from the Repository

### About this task

To retrieve the schema structure of the selected table from the Repository:

### Procedure

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**.

**Note:** 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.

Property	Value
Catch output reject	false
Catch lookup inner join reject	false
Schema Type	Repository
Schema Id	DELIM:owners - metadata

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		

**Warning:** Changing the schema type of the subordinate table across a Join from **Built-In** to **Repository** causes the Join to get lost.

**Note:** 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.

## Searching schema columns

### About this task

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.

### Procedure

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

## 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.

Column	Key	Type	N..	Date Patter...	Length	Preci...	De...	Com...
ID_Owner	<input checked="" type="checkbox"/>	Integ...	<input checked="" type="checkbox"/>		2	0		
First N	<input type="checkbox"/>	String	<input type="checkbox"/>		19	0		
ID_Insurance	<input type="checkbox"/>	String	<input checked="" type="checkbox"/>		3	0		
Children_Nr	<input type="checkbox"/>	Integ...	<input checked="" type="checkbox"/>		2	0		

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.  <b>Note:</b> 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.

**Note:** 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.

### Enabling automatic data type conversion

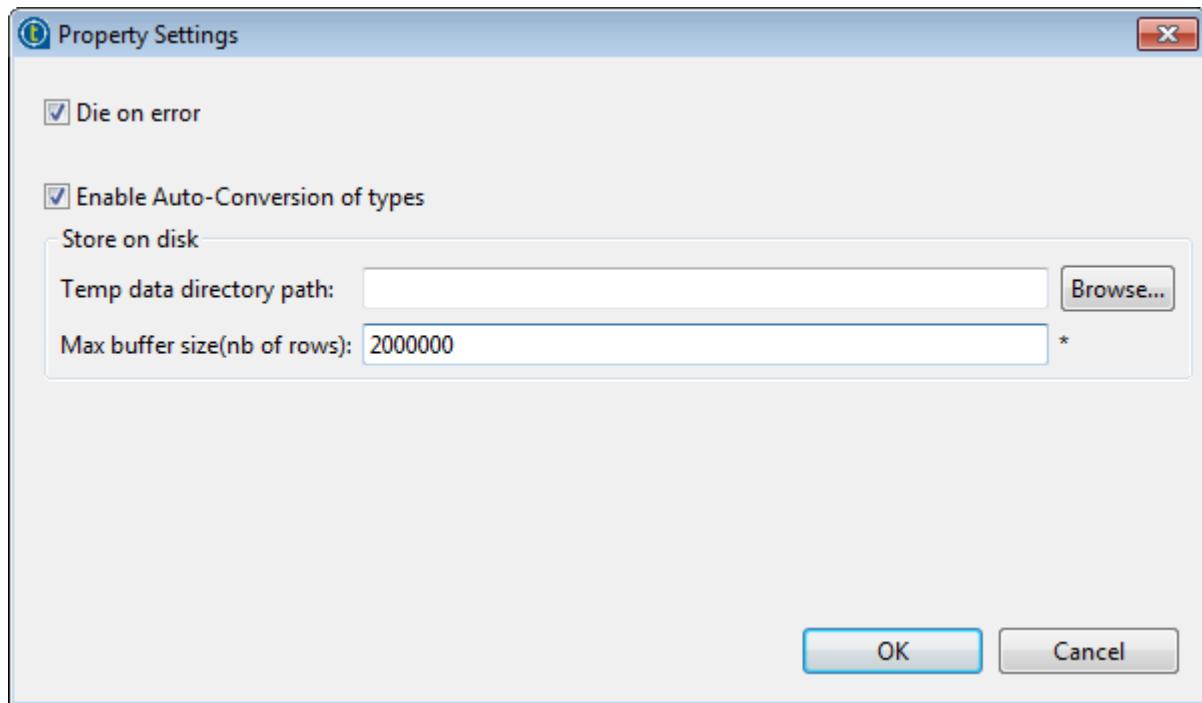
#### Before you begin

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:

#### Procedure

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**.



### What to do next

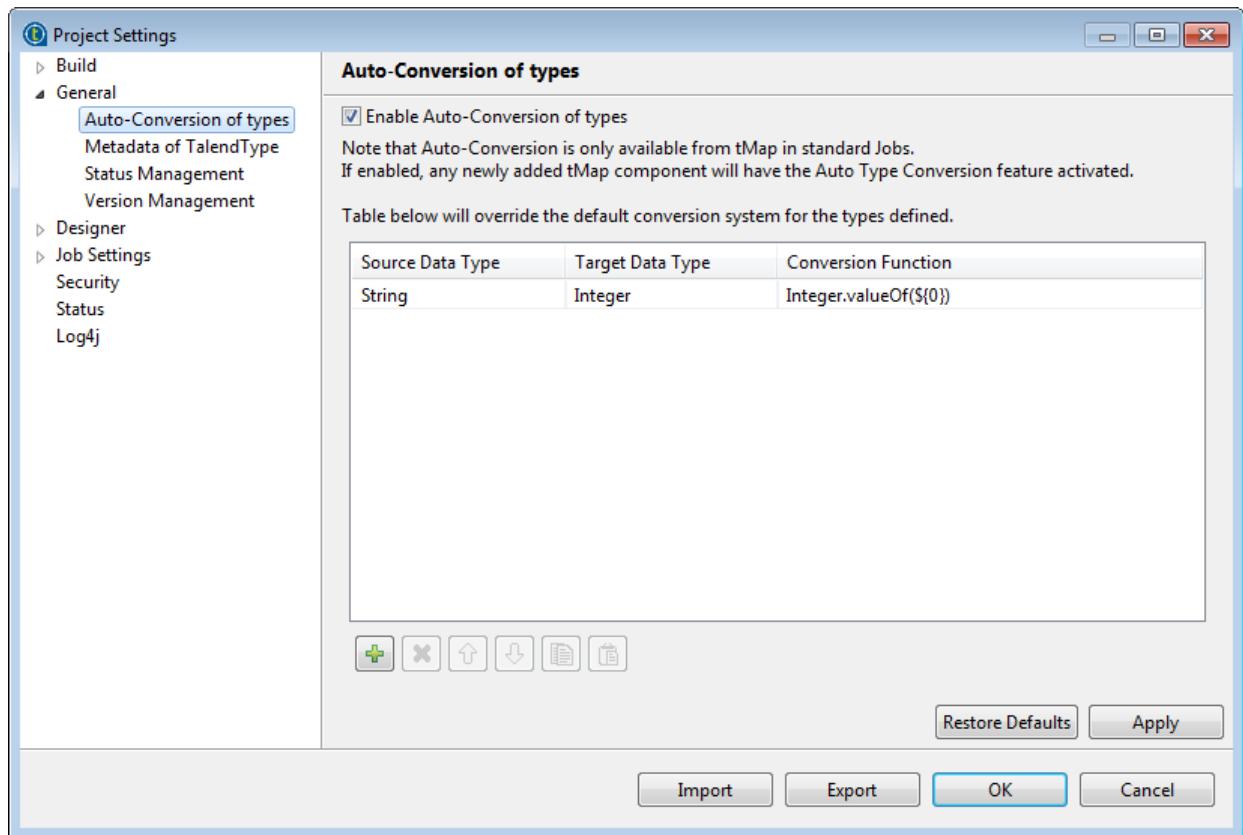
You can activate the automatic conversion option at the project level so that any **tMap** component added afterwards in the project will have this feature enabled.

### Defining rules to override the default conversion behavior

If needed, you can also define conversion rules to override the default conversion behavior of **tMap**.

### Procedure

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.

### 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.

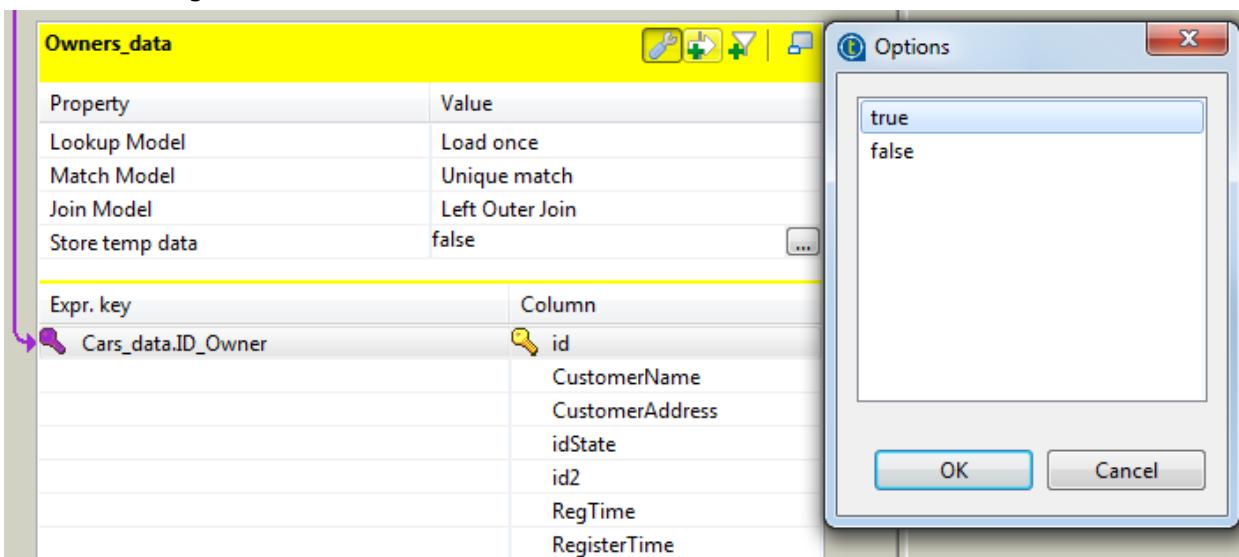
A feature has been added (in Java only for the time being) to the **tMap** component, in order to reduce the memory in use for lookup loading. In fact, rather than storing the temporary data in the system memory and thus possibly reaching the memory limitation, the **Store temp data** option allows you 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**.

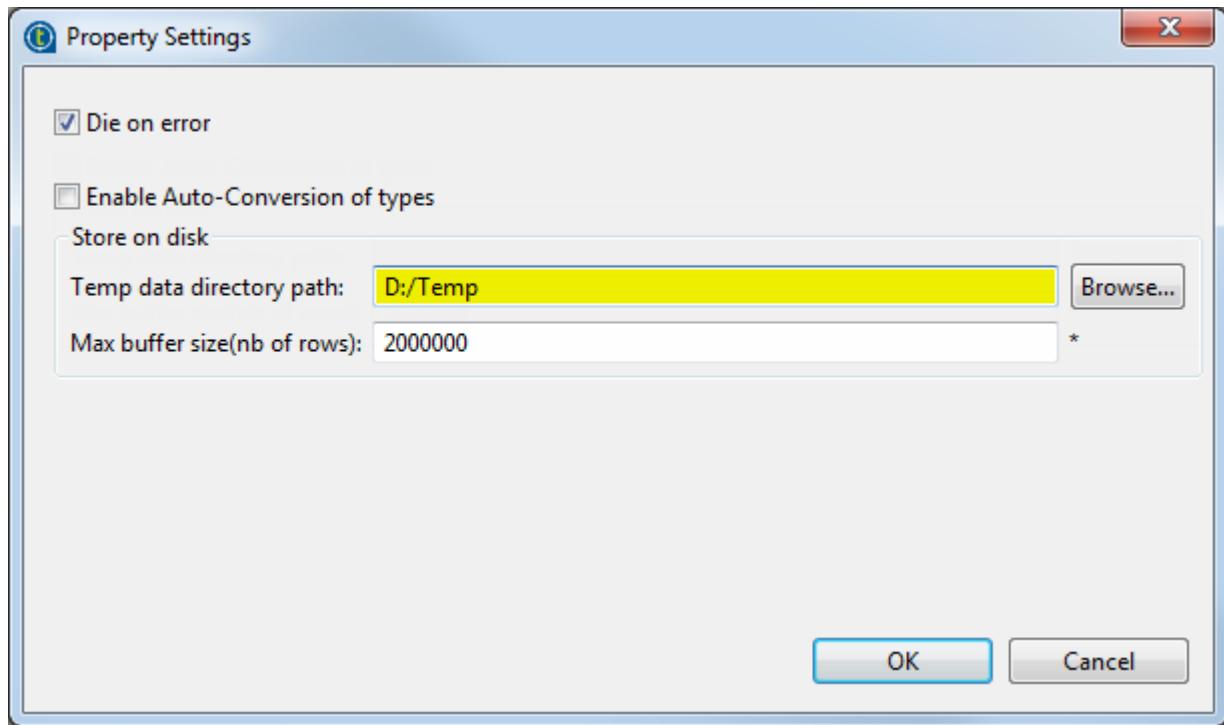
2. In input area, click the Lookup table describing the temporary data you want to be loaded onto the disk rather than in the memory.
3. Click the **tMap settings** button to display the table properties.
4. Click in the **Value** field corresponding to **Store temp data**, and then click the [...] button 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**:

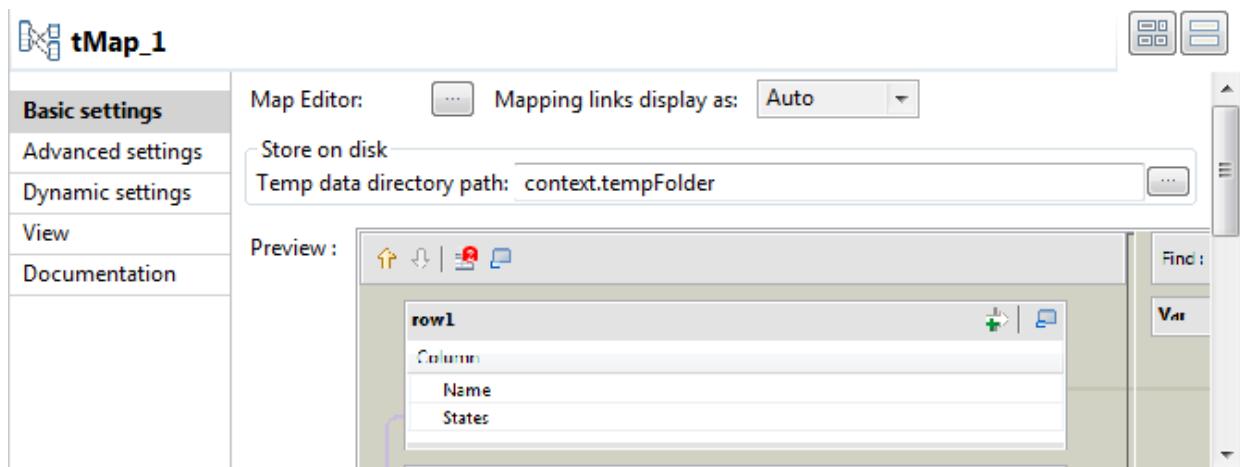
1. Click the **Property Settings** button at the top of the input area to display the **Property Settings** dialog box.
2. In **Property Settings** dialog box, fill the **Temp data directory path** field with the full path to the directory where the temporary data should be stored.
3. In the **Max buffer 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](#) on page 66.



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.

**Note:** 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.

## 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](#) on page 167.

Note that **Reload at each row** in a streaming Job is supported by the Lookup Input components only such as **tMongoDBLookupInput**.

- **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.

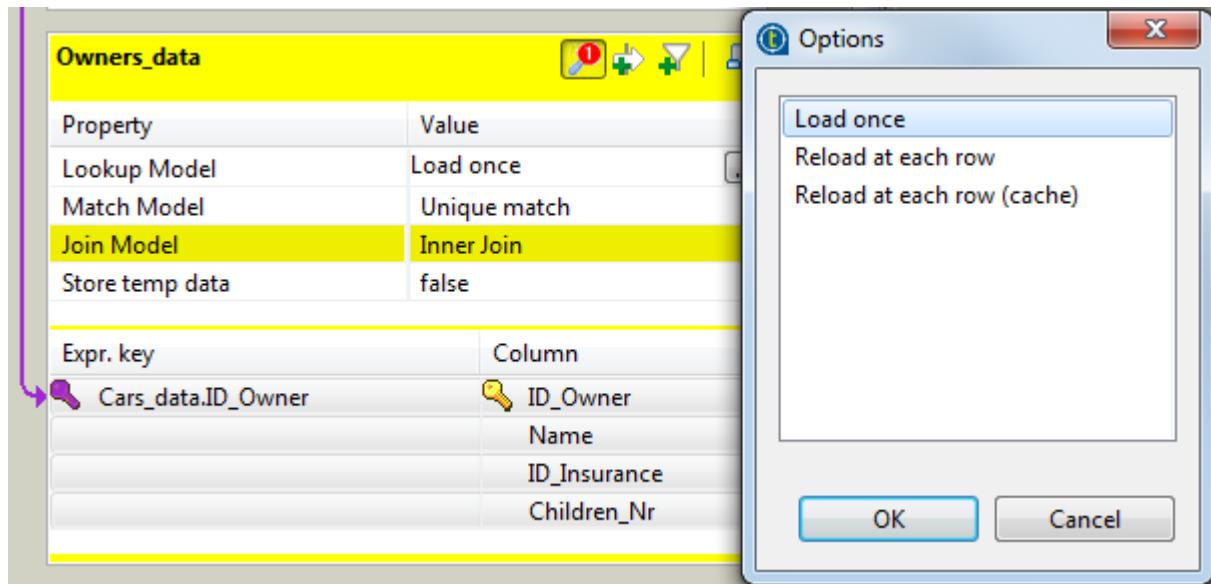
## Setting the loading mode of a lookup flow

### About this task

To set the loading mode of a lookup flow:

### Procedure

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.

## Results

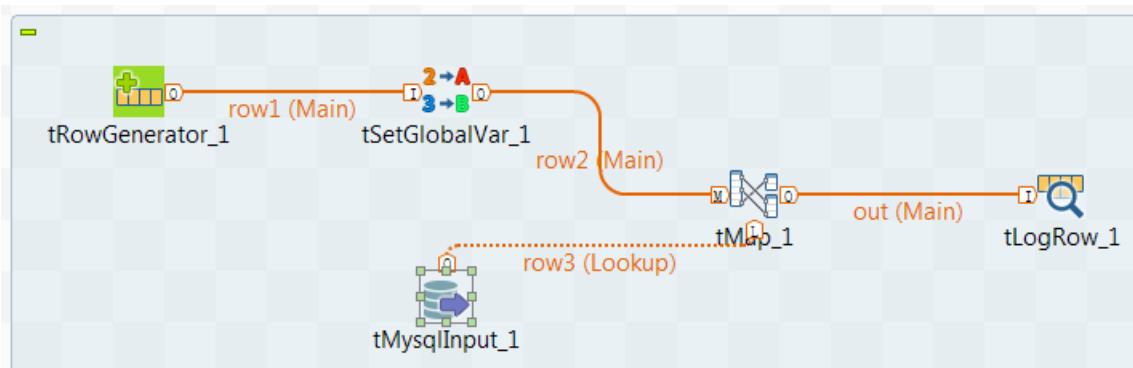
For use cases using these options, see the related documentation of the **tMap** component.

### Reloading data at each row

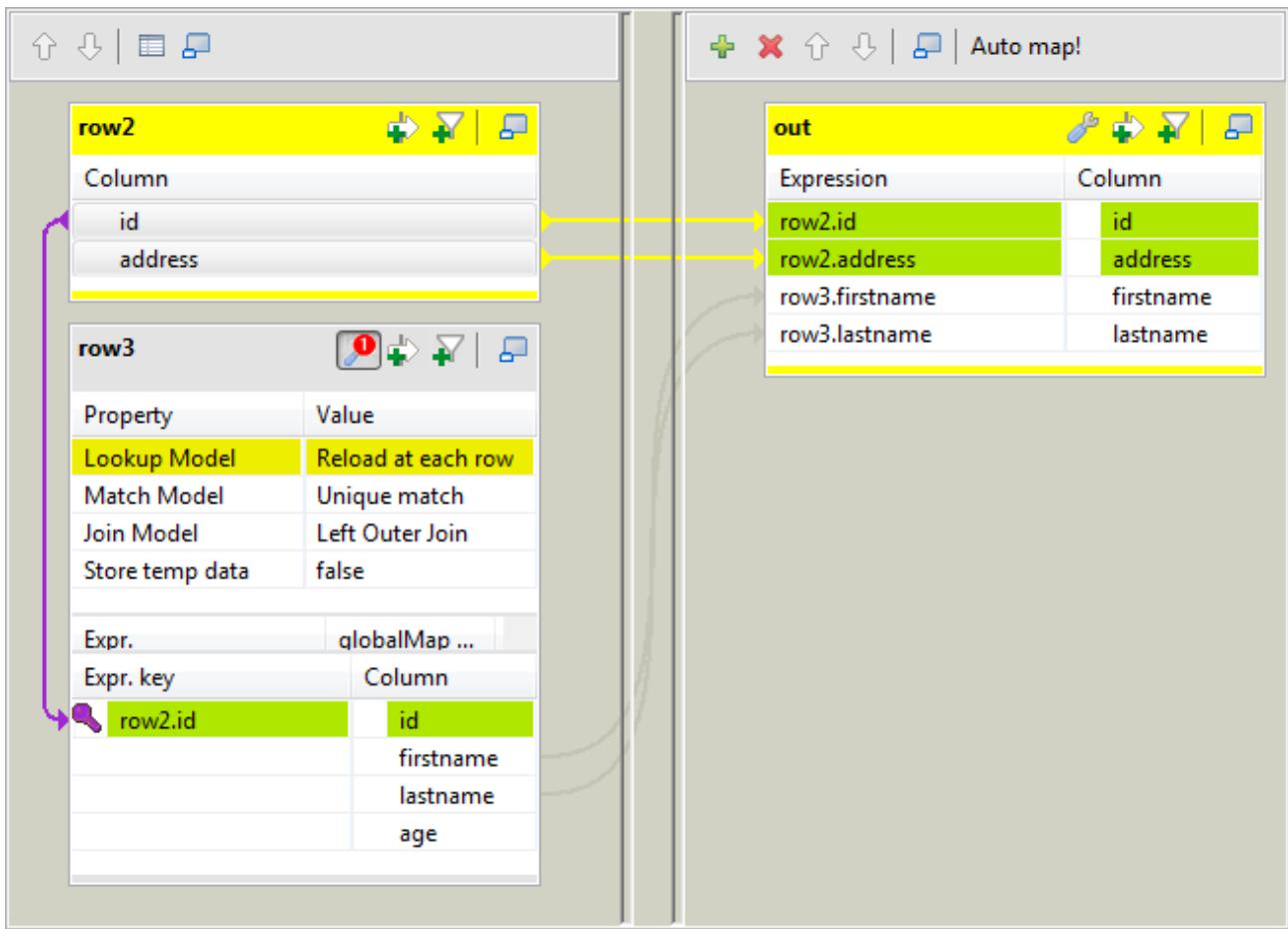
#### About this task

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:

### Procedure

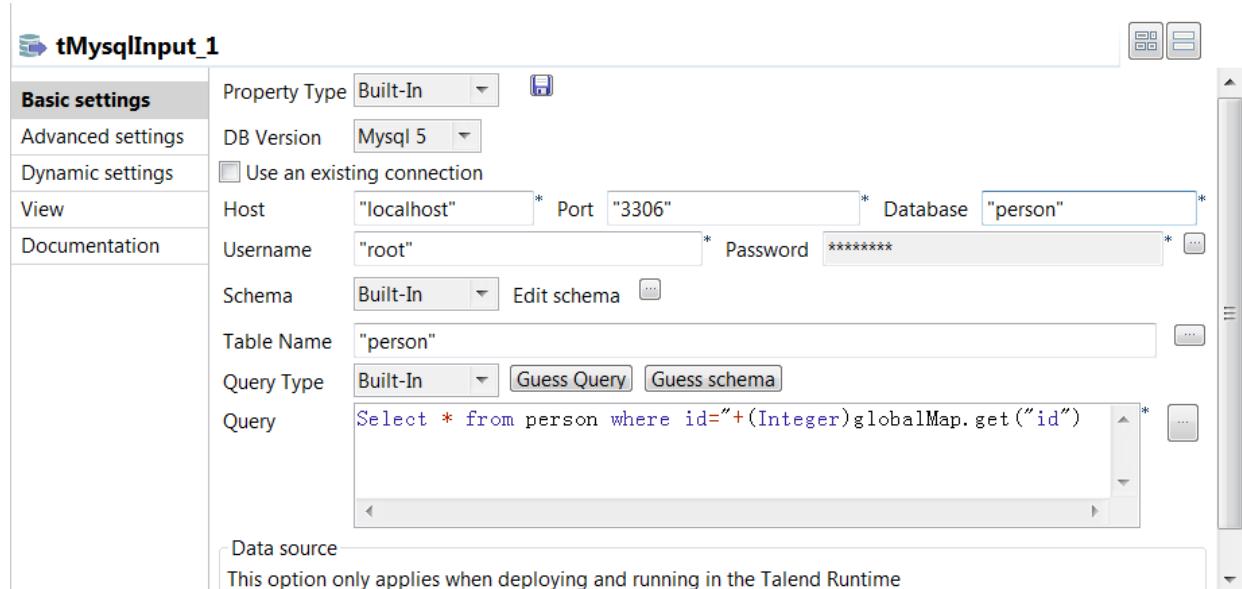
1. Double-click **tSetGlobalVar** to open its **Component** view.

The screenshot shows the **tSetGlobalVar** component view. On the left, there is a sidebar with tabs: 'Basic settings' (selected), 'Advanced settings', 'Dynamic settings', 'View', and 'Documentation'. On the right, there is a 'Variables' section with a table:

Key	Value
"id"	"row1.id"

Below the table are several icons: a green plus sign, a red minus sign, up and down arrows, a document icon, and a clipboard icon.

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.



- 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")"`

## Results

Refer to the related documentation of the components used in this example for more information.

## tXMLMap operation

**Note:** 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: [Using Explicit Join on page 138](#)),
- Automated XML tree construction on both of the input and the output sides,
- inner join and left outer join (related topic: [Using Inner Join on page 141](#))
- lookup between data sources whatever they are flat or XML data using models like **Load once** (related topic: [Handling Lookups on page 166](#)),
- 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.

**Note:** 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](#) on page 163.

### Using the document type to create the XML tree

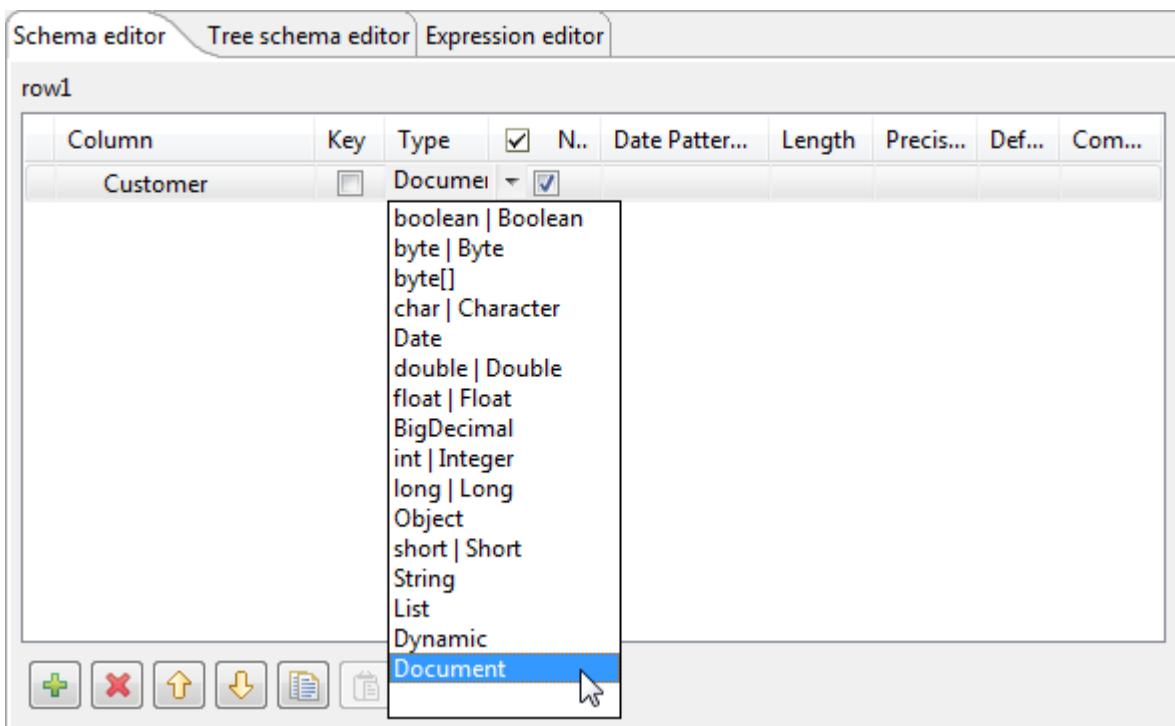
The **Document** data type fits perfectly the conception of defining XML structure as easily as possible. When you need the XML tree structure to map the input or output flow or both, use this type. Then you can import the XML tree structure from various XML sources and edit the tree directly in the mapping editor, thus saving the manual efforts.

#### Setting up the Document type

##### About this task

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](#) on page 160.

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.



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.

**Note:** 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 on page 39](#).

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:

## Procedure

- Import the custom XML tree structure from one of the following types of sources:

- XML or XSD files (related topic: [Importing the XML tree structure from XML and XSD files](#) on page 173)

**Note:** 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: [Importing the XML tree structure from the Repository](#) on page 174).

**Note:** 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 [Setting or resetting a loop element for an imported XML structure](#) on page 175
- Optional: 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</b> and <b>Create Attribute</b>	Add elements or attributes to develop an XML tree. Related topic: <a href="#">Adding a sub-element or an attribute to an XML tree structure</a> on page 176
<b>Set a namespace</b>	Add and manage given namespaces on the imported XML tree. Related topic: <a href="#">Managing a namespace</a> on page 178
<b>Delete</b>	Delete an element or an attribute. Related topic: <a href="#">Deleting an element or an attribute from the XML tree structure</a> on page 177
<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.  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.

Options	Operations
<b>As group element</b>	On the XML tree of the output side, set an element as group element. Related topic: <a href="#">Grouping the output data</a> on page 180
<b>As aggregate element</b>	On the XML tree of the output side, set an element as aggregate element. Related topic: <a href="#">Aggregating the output data</a> on page 181
<b>Add Choice</b>	<p>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.</p> <p>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.</p> <div data-bbox="747 698 1426 848" style="border: 1px solid #ccc; padding: 5px;"> <p><b>Note:</b></p> <p>The <b>tXMLMap</b> component declares automatically any Choice element set in the XSD file it imports.</p> </div>
<b>Set as Substitution</b>	<p>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.</p> <p>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.</p> <div data-bbox="747 1125 1426 1275" style="border: 1px solid #ccc; padding: 5px;"> <p><b>Note:</b></p> <p>The <b>tXMLMap</b> component declares automatically any Substitution element set in the XSD file it imports.</p> </div>

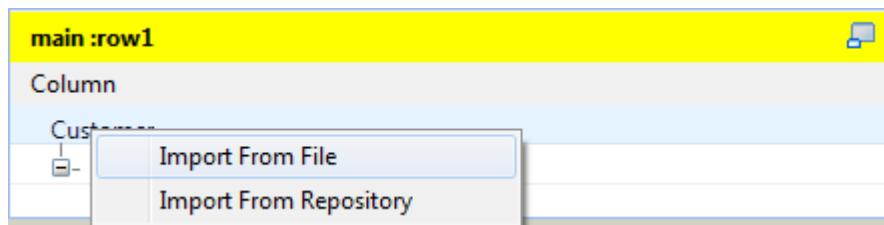
The following sections present more details about the process of creating the XML tree.

## Importing the XML tree structure from XML and XSD files

### Importing the XML tree structure from an XML file

#### Procedure

1. In the input flow table of interest, right-click the column name to open the contextual menu. In this example, it is Customer.

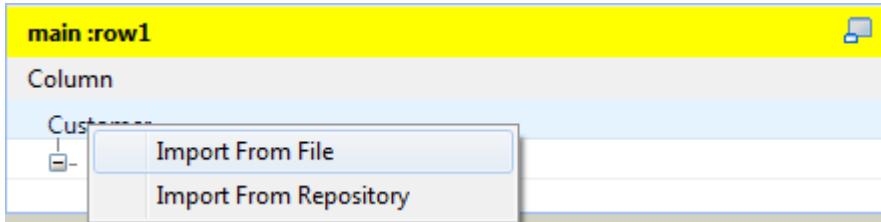


2. From this menu, select **Import From File**.
3. In the pop-up dialog box, browse to the XML file you need to use to provide the XML tree structure of interest and double-click the file.

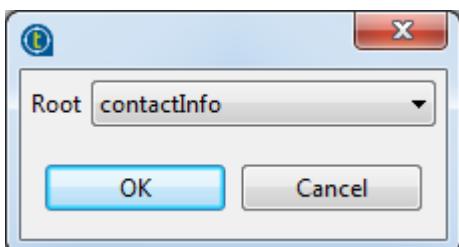
## Importing the XML tree structure from an XSD file

### Procedure

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.



**Note:** 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 use the tree schema editor. For further information about this editor, see [Editing the XML tree schema on page 187](#).

### What to do next

Then, you need to define the loop element in this XML tree structure. For further information about how to define a loop element, see [Setting or resetting a loop element for an imported XML structure on page 175](#).

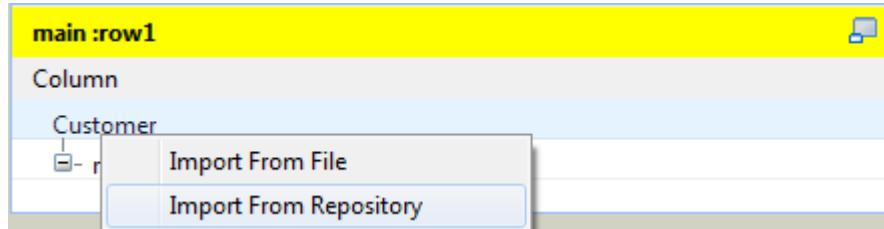
## Importing the XML tree structure from the Repository

### About this task

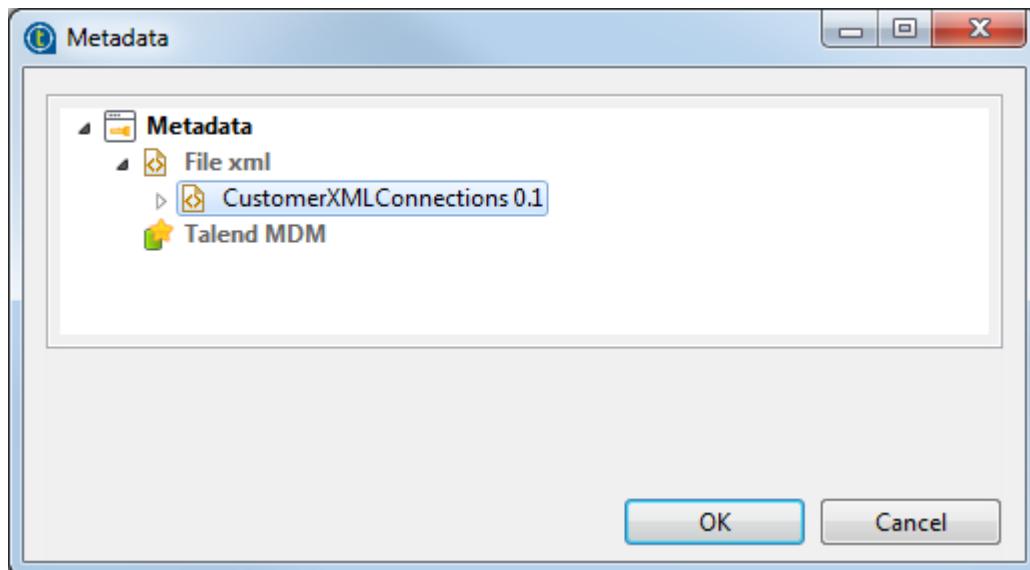
To do this, proceed as follows:

### Procedure

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.

#### Note:

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](#) on page 223.

4. Click **OK** to validate this selection.

#### Results

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.

#### [Setting or resetting a loop element for an imported XML structure](#)

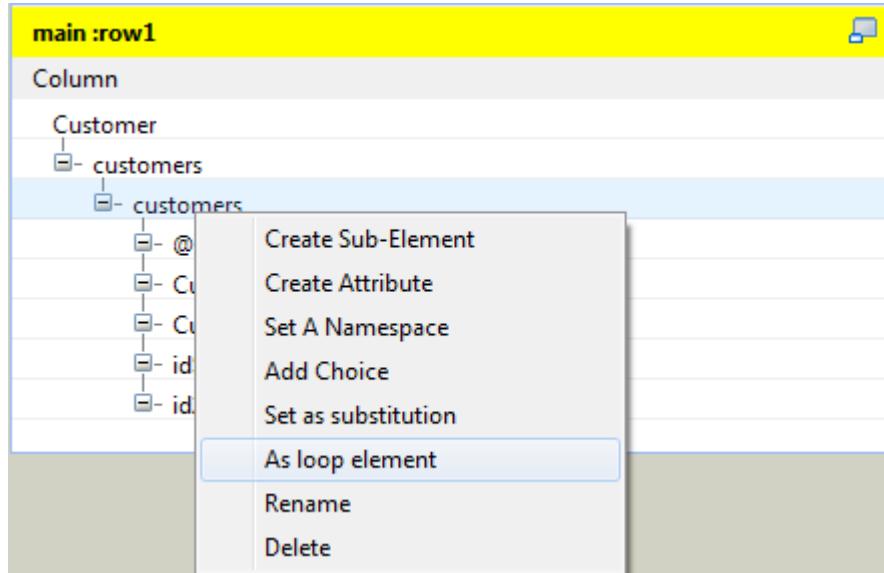
#### About this task

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.

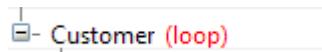
Whatever you need to set or reset a loop element, proceed as follows:

#### Procedure

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**.



## Results

### Note:

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.

## Adding a sub-element or an attribute to an XML tree structure

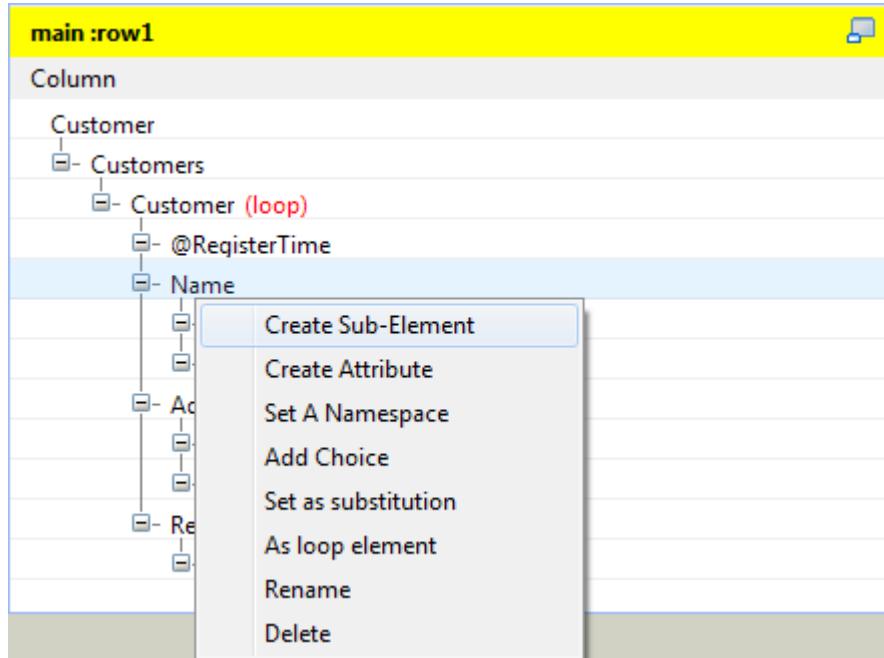
### About this task

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.

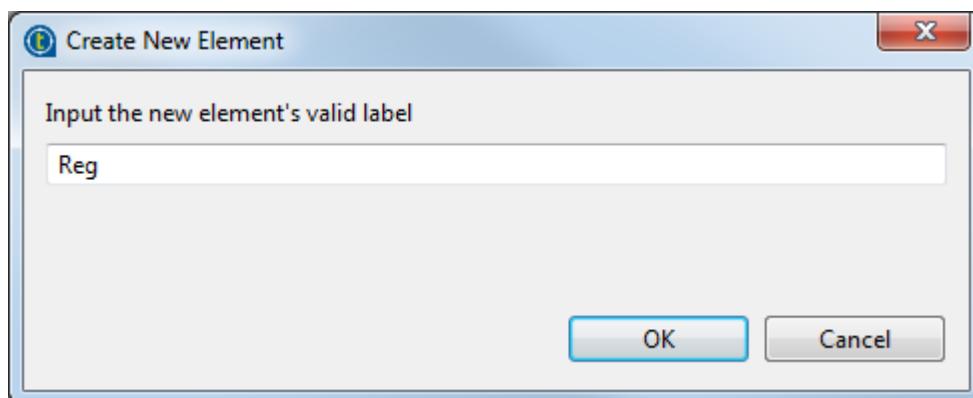
To do either of these operations, proceed as follows:

### Procedure

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.

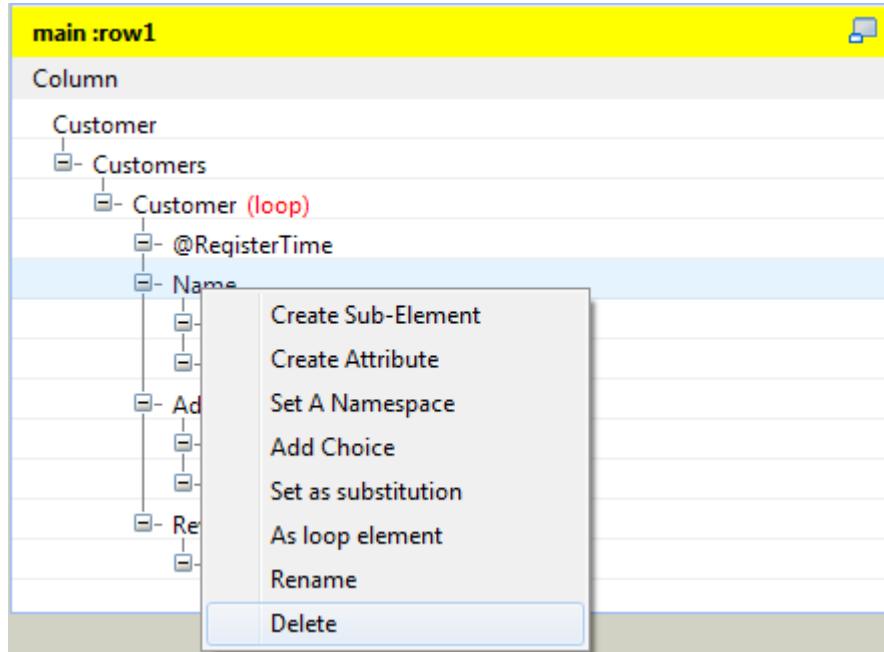
#### **Deleting an element or an attribute from the XML tree structure**

##### **About this task**

From an established XML tree, you may need to delete an element or an attribute. To do this, proceed as follows:

##### **Procedure**

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.

### Managing 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.

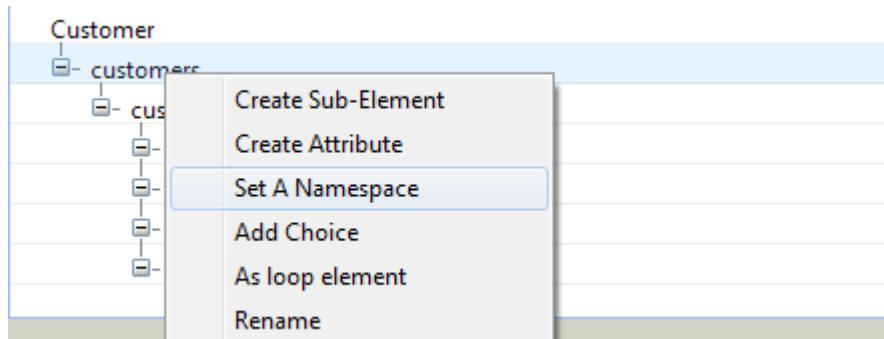
#### Defining a namespace

#### About this task

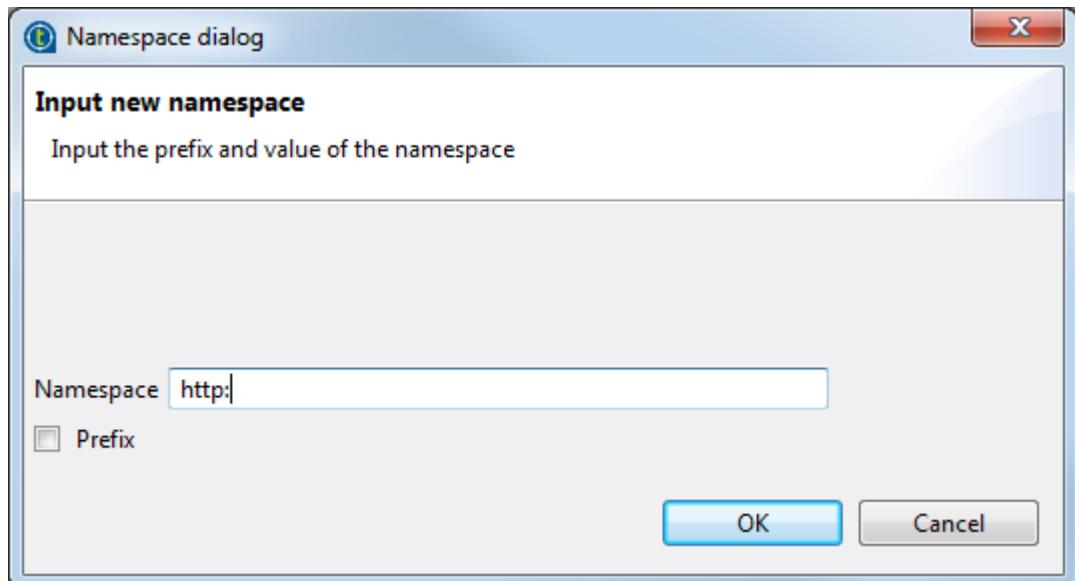
To do this, proceed as follows:

#### Procedure

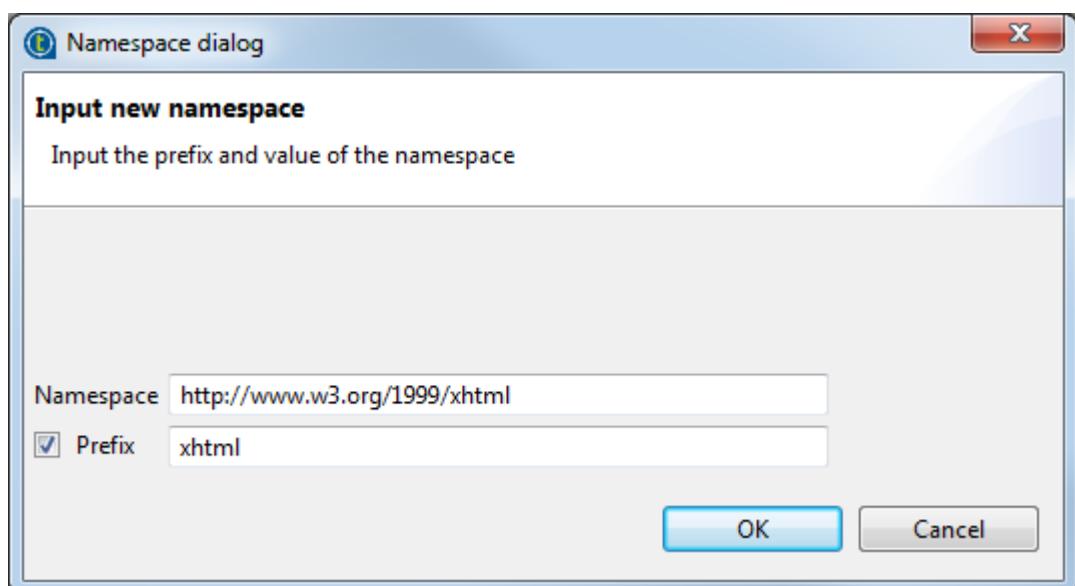
1. In the XML tree of the input or the output data flow you need to edit, right click the element for which you need to declare a namespace. For example, in a Customer XML tree of the output flow, you need to set a namespace for the root.



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.

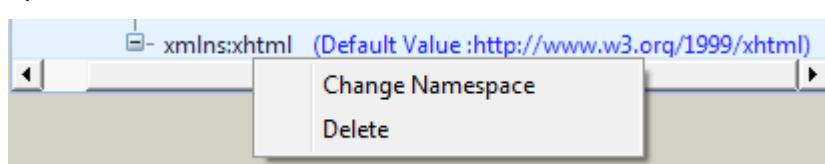
#### Modifying the default value of a namespace

##### About this task

To do this, proceed as follows:

##### Procedure

- 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.

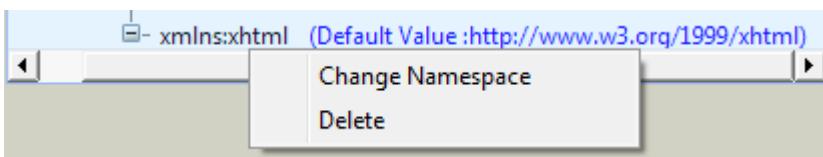
## Deleting a namespace

### About this task

To do this, proceed as follows:

### Procedure

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

### Grouping 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.

#### Note:

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](#).

**Note:** **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

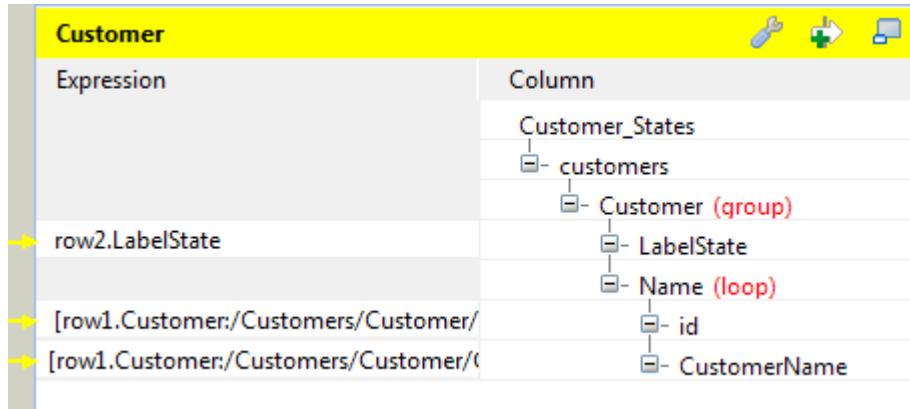
### About this task

To set a group element, proceed as follows:

## Procedure

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

### About this task

To revoke a defined group element, proceed as follows:

## Procedure

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.

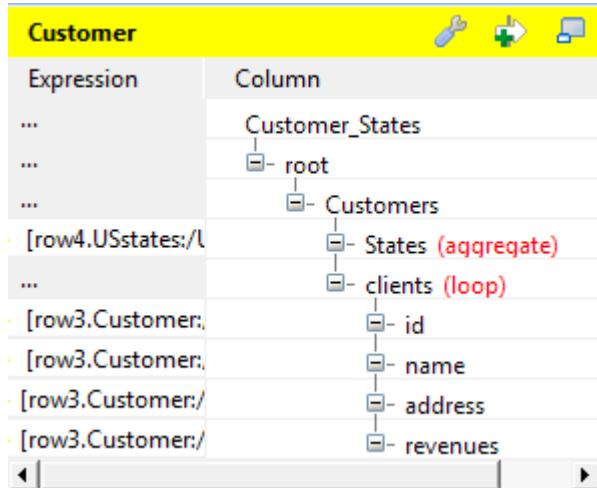
## Aggregating the output data

### About this task

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.

## Procedure

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.



- To revoke the definition of the aggregate element, simply right-click the defined aggregate element and from the contextual menu, select **Remove aggregate element**.

## Results

### Note:

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 an example about how to use the aggregate element with **tXMLMap**, see [tXMLMap](#).

**Note:** **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.

## 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.

### Outputting elements into one document

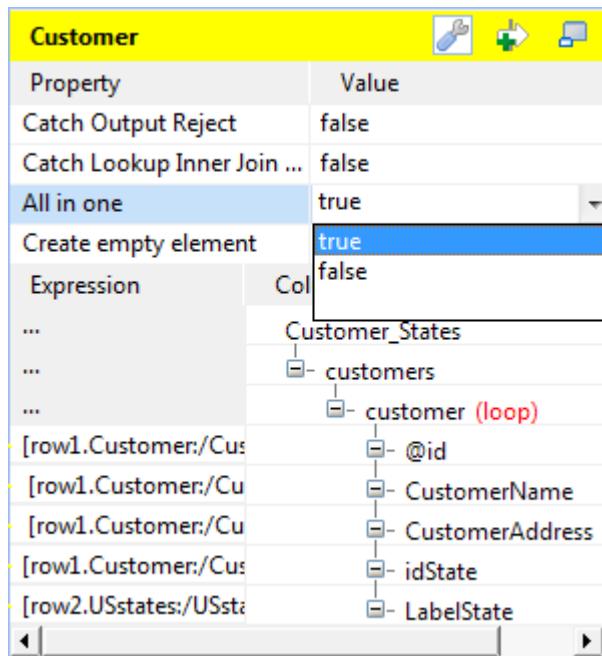
#### About this task

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:

#### Procedure

- 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 Kholwood 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</CustomerAddress><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>Connecticut</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>Florida</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>
```

**Note:** The **All in one** feature is disabled if you are using the aggregate element.

## Managing empty element in Map editor

### About this task

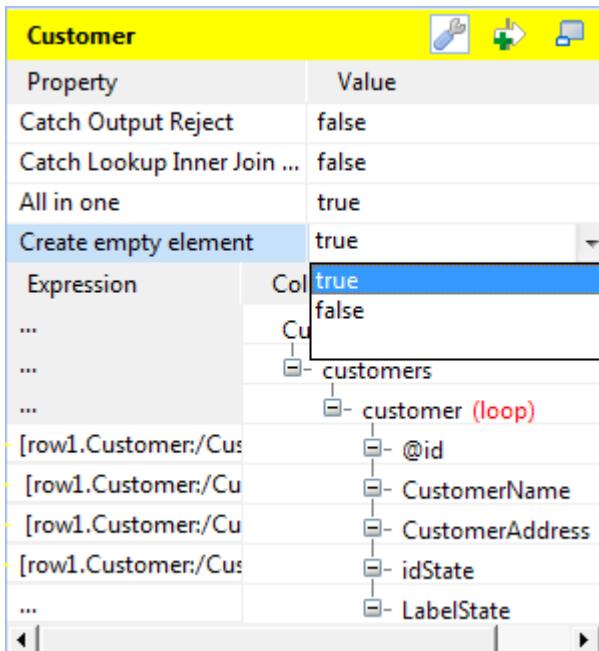
It may be necessary to create and output empty elements during the process of transforming data into XML flow, such as, when **tXMLMap** works along with **tWriteXMLField** that creates empty elements or when there is no input column associated with certain XML node in the output XML data flow.

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:

## Procedure

- Click the pincer icon to open the map setting panel.

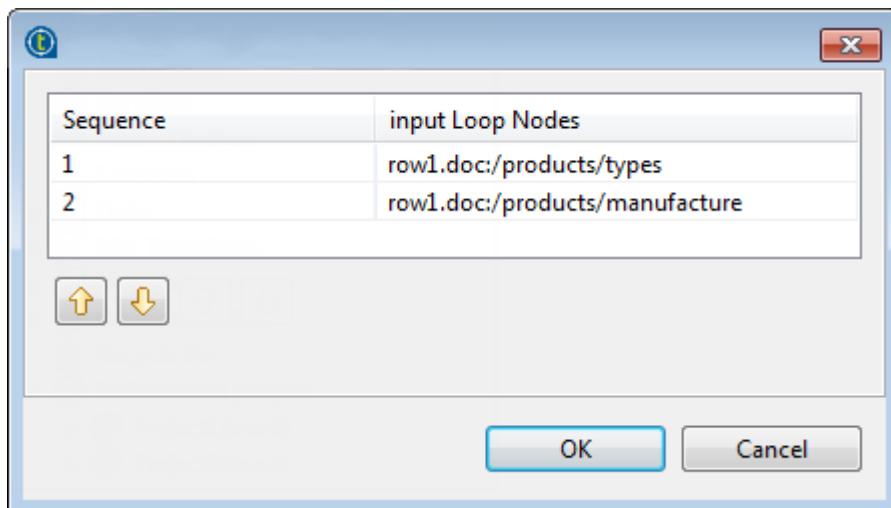


- In the panel, click the **Create empty element** field and from the drop-down list, select **true** or **false** to decide whether to output the empty element.
  - If you select **true**, the empty element is created in the output XML flow and output, for example, <customer><LabelState/></customer>.
  - If you select **false**, the empty element is not output.

## Defining the sequence of multiple input loops

### About this task

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:

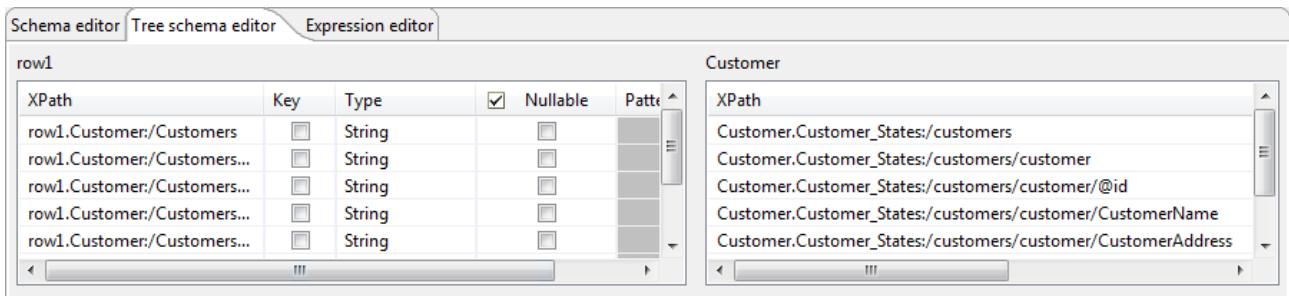
### Procedure

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.

### 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.  <b>Note:</b> 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.

#### Note:

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.

# Centralizing metadata for Data Integration

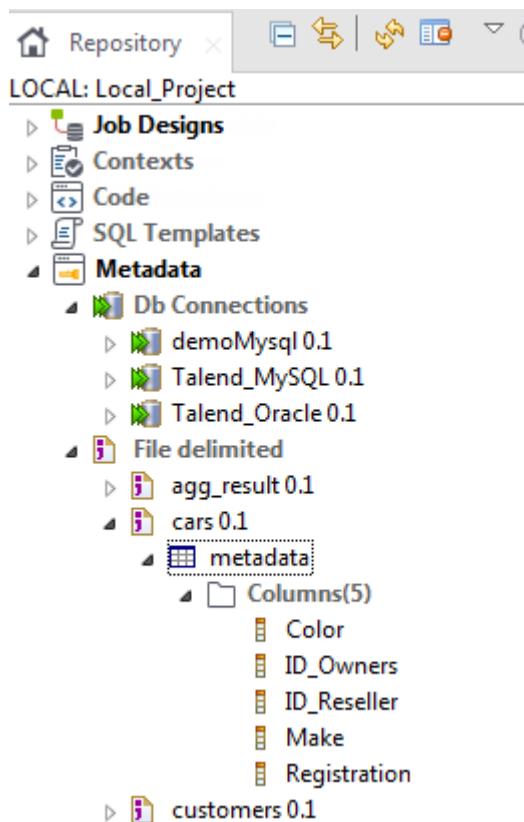
## Objectives

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.



## 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.

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.

## 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.

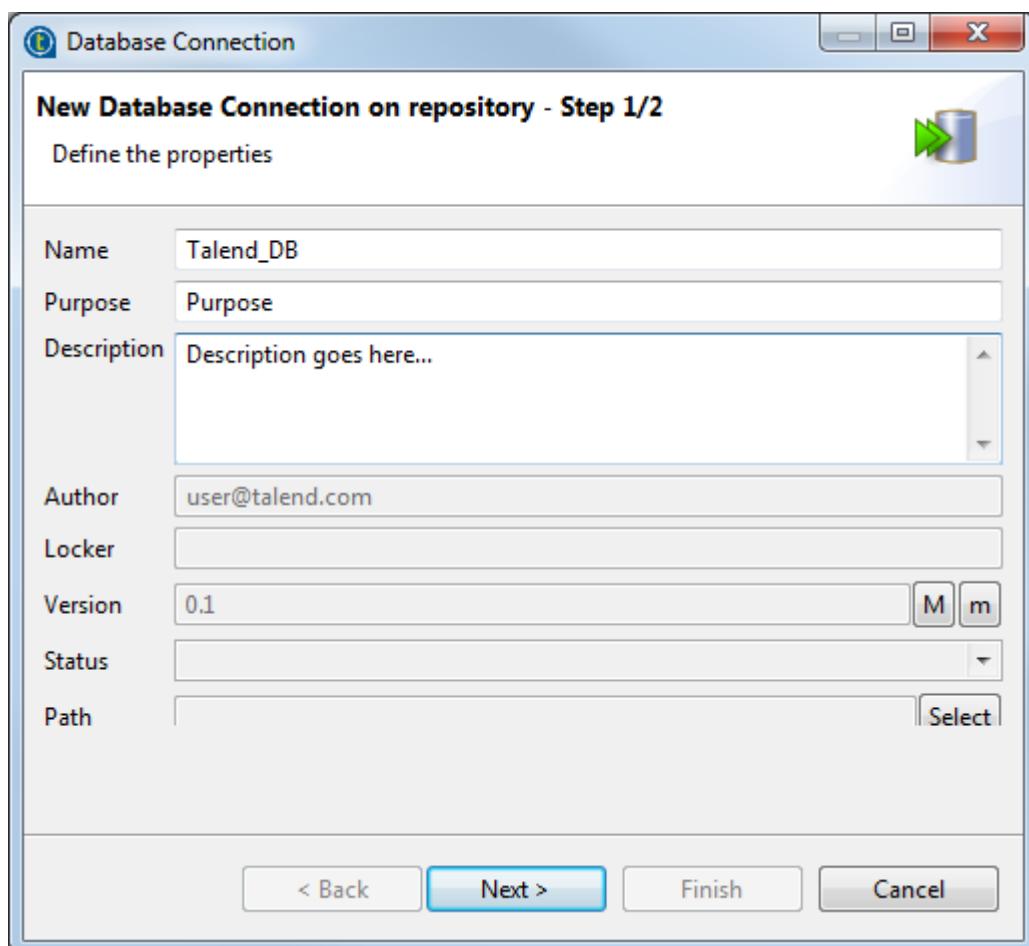
To modify an existing database connection, right-click the connection item from the **Repository** tree view, and select **Edit connection** to open the connection setup wizard.

Then define the general properties and parameters of the connection in the wizard.

### Defining general properties

#### Procedure

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.



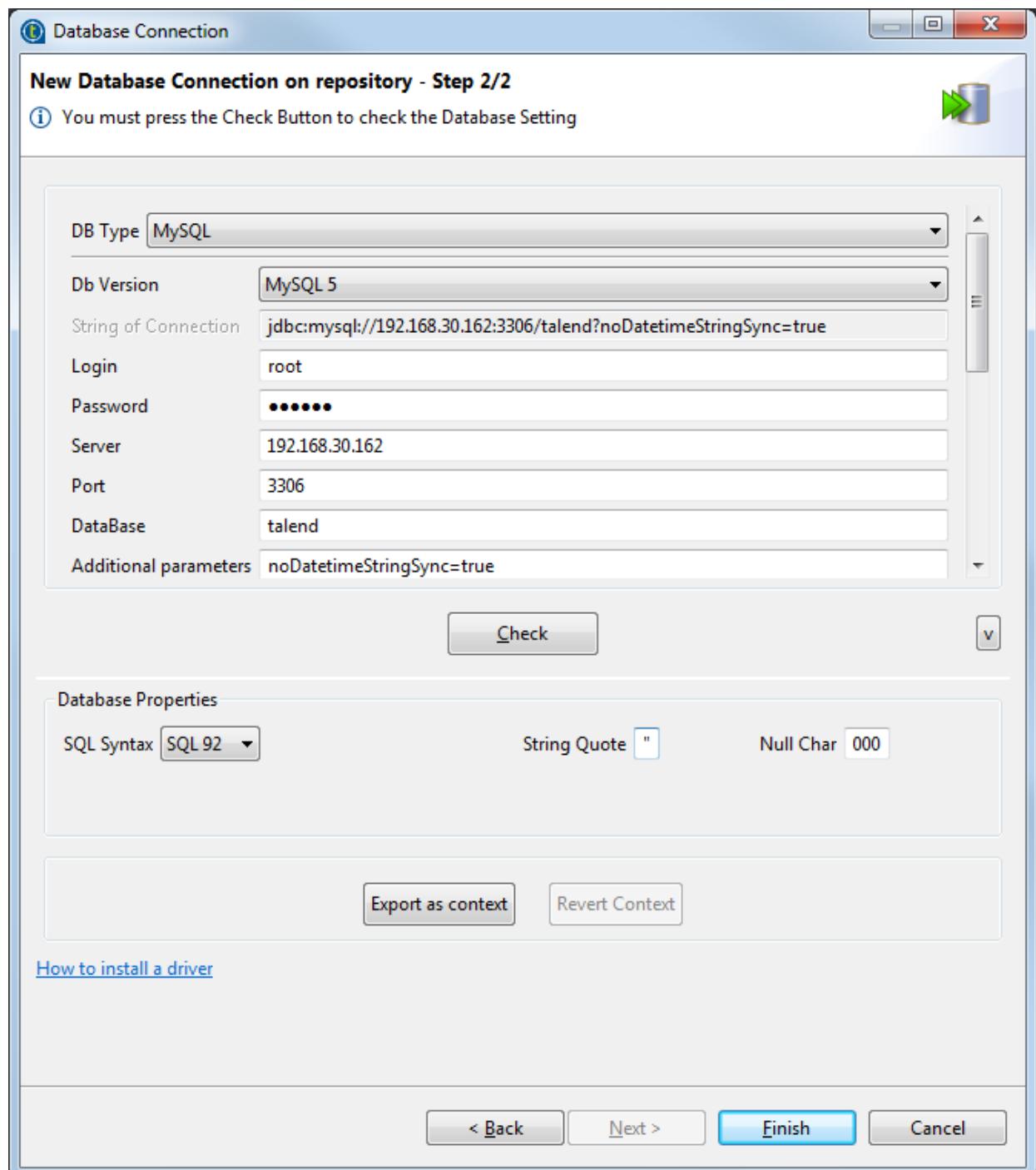
2. Fill in the optional **Purpose** and **Description** fields as required. The information you fill in the **Description** field will appear as a tooltip when you move your mouse pointer over the connection.

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](#) on page 382 and [Status management](#) on page 383 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.

## Defining connection parameters

### Procedure

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 with the database type you select.



**Note:**

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](#).

You can set up a connection to Oracle using the Wallet by selecting **Oracle Custom** from the **DB Type** drop-down list, then selecting the **Use SSL Encryption** check box and specifying the related properties, including the path to your TrustStore and KeyStore files and the password for each of them, and whether to disable the use of CBC (CipherBlock Chaining).

If you need to connect to Hive, we recommend using one of the **Talend** solutions with Big Data.

**Warning:**

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.

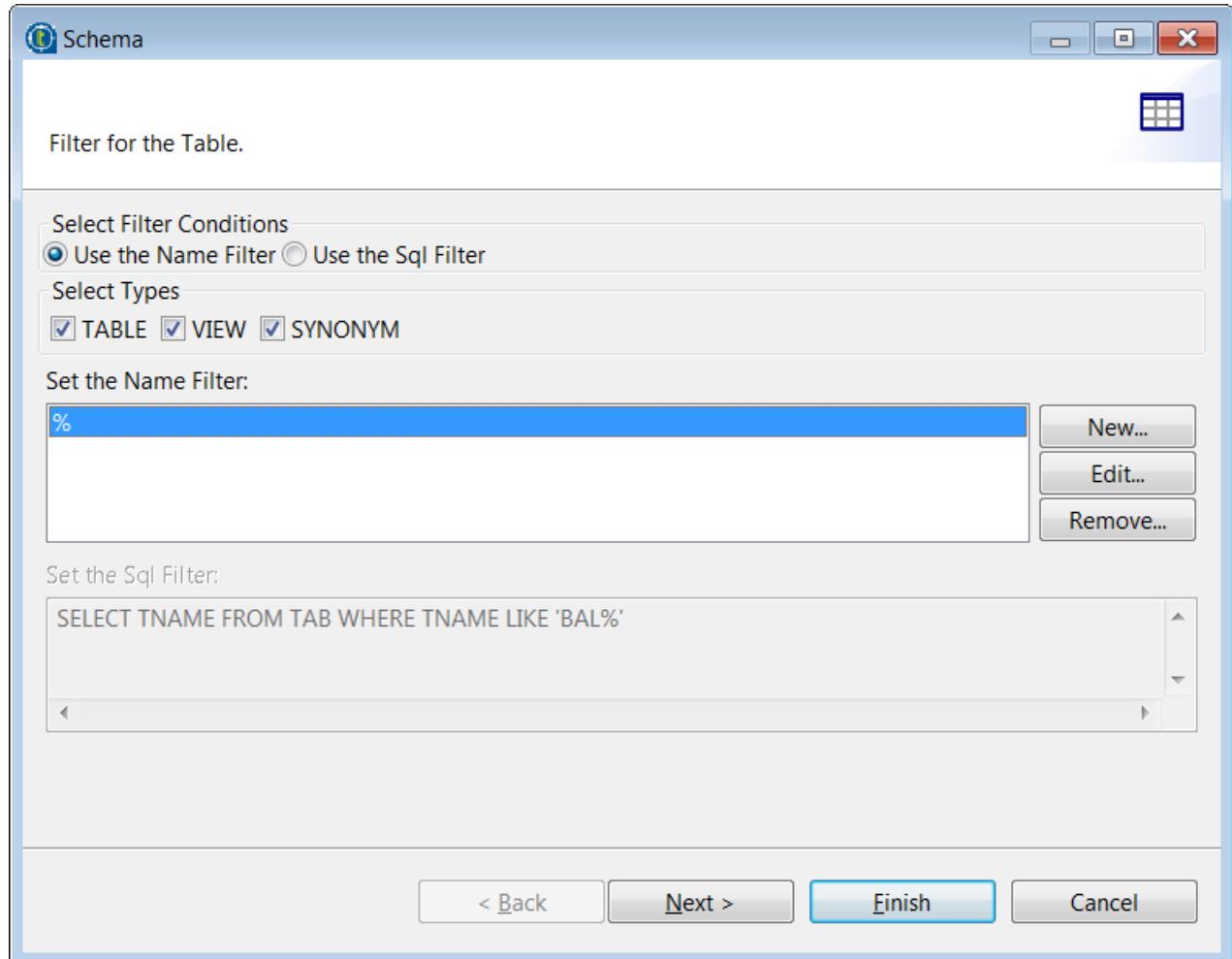
2. (Optional) Specify additional connection properties through the **Additional parameters** field in the **Database Settings** area.
3. Click **Check** to check your connection.  
If the connection fails, a message box is displayed to indicate this failure and from that message box. From that message 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.
4. 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 will be saved under the **Db Connections** node in the **Repository** tree view, and several folders for SQL queries and different types of schemas, such as **Calculation View schemas** (only for SAP HANA), **Synonym schemas** (for Oracle, IBM DB2 and MSSQL), **Table schemas**, and **View schemas** will be created under the database connection node.

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.

## Retrieving table schemas

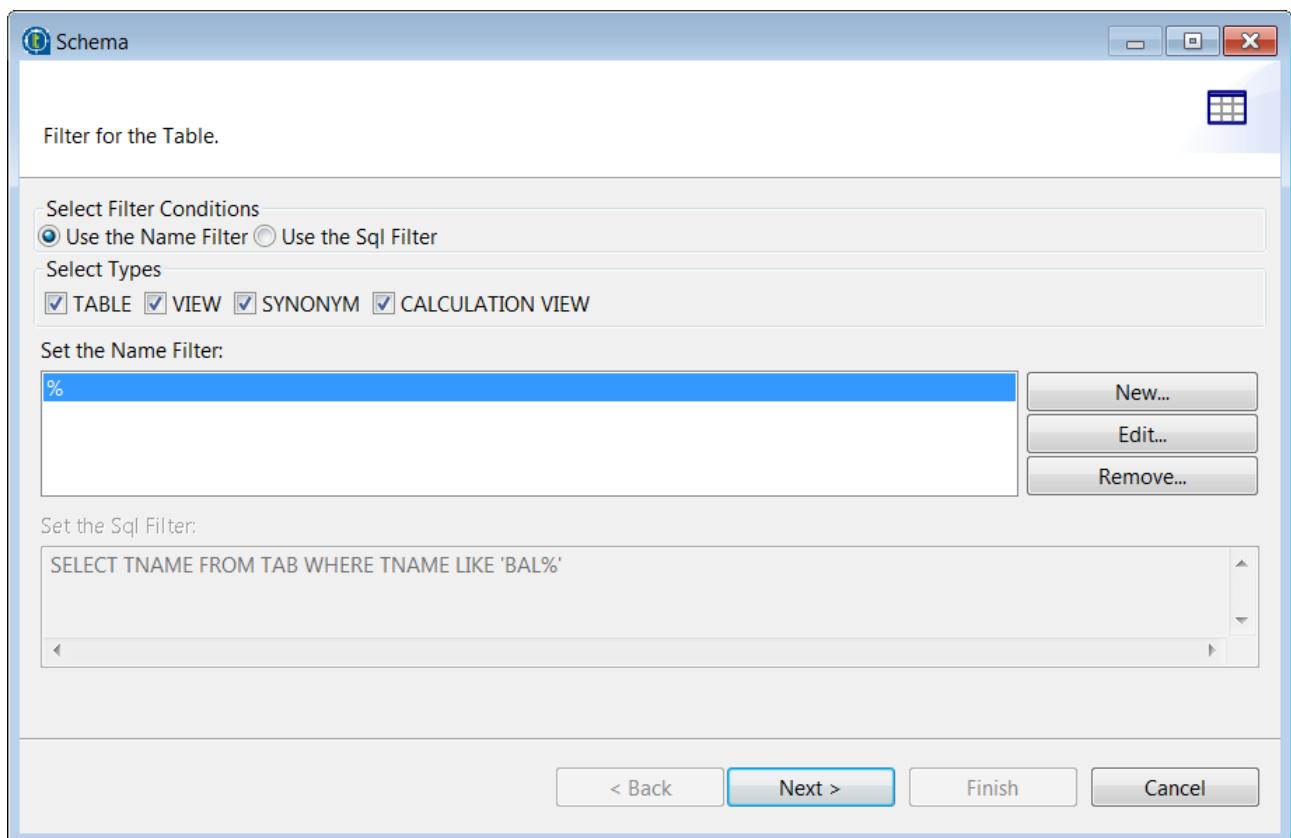
To retrieve table schemas from the database connection you have just set up, right-click the connection item from the **Repository** tree view, and select **Retrieve schema** from the contextual menu.



### Note:

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 specify the filter for searching different database objects such as table, view, synonym (for Oracle, IBM DB2 and MSSQL) and calculation view (only for SAP HANA).



## 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.

### Filtering database tables based on their names

#### Procedure

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.

**Note:** 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.

#### Example

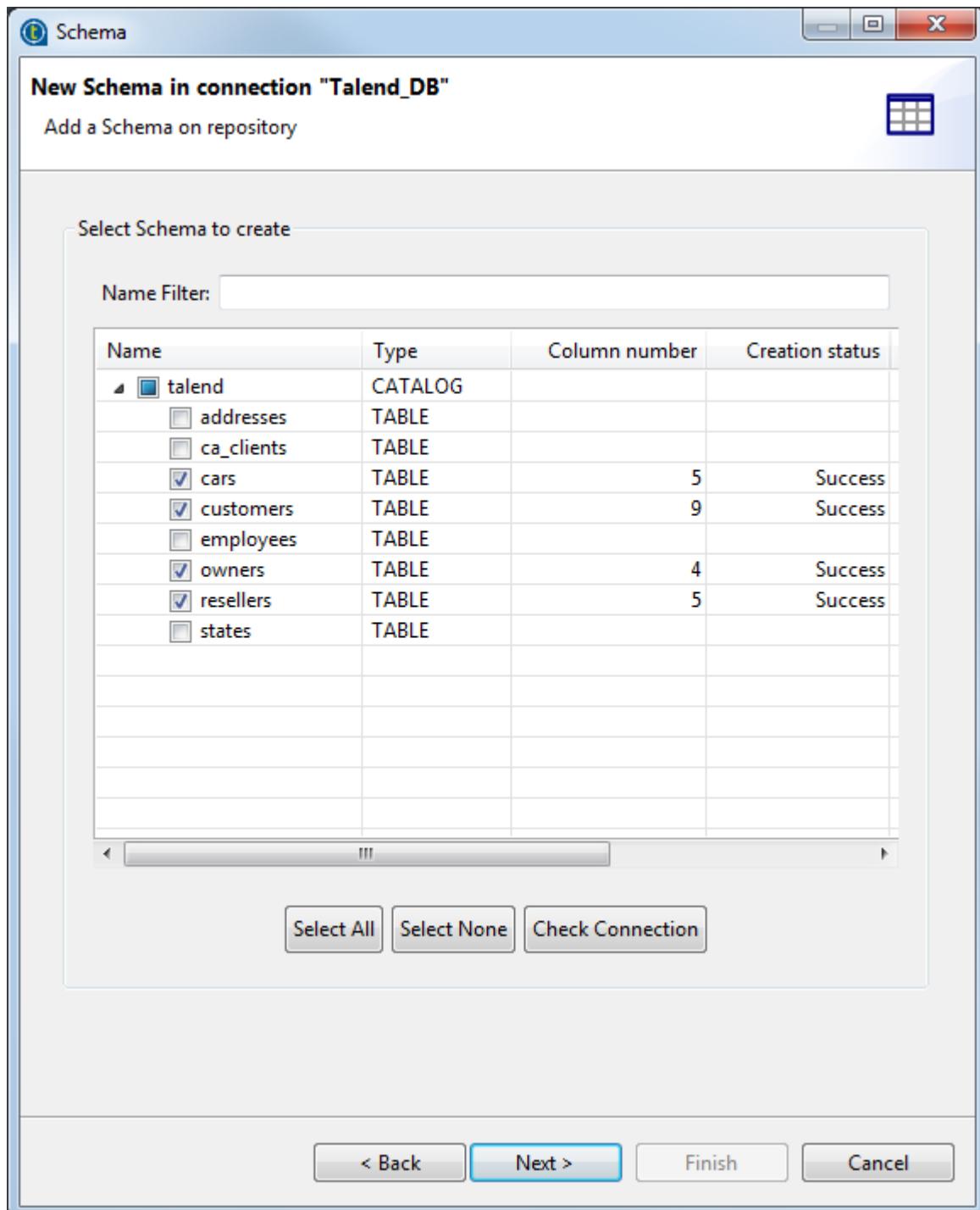
For example, if you want to recuperate the database objects which names start with "A", enter the filter A%, or if you want to recuperate all database objects which names end with "type", enter %type as your filter.

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.

## Filtering database objects using an SQL query

### Procedure

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.



## Selecting tables and defining table schemas

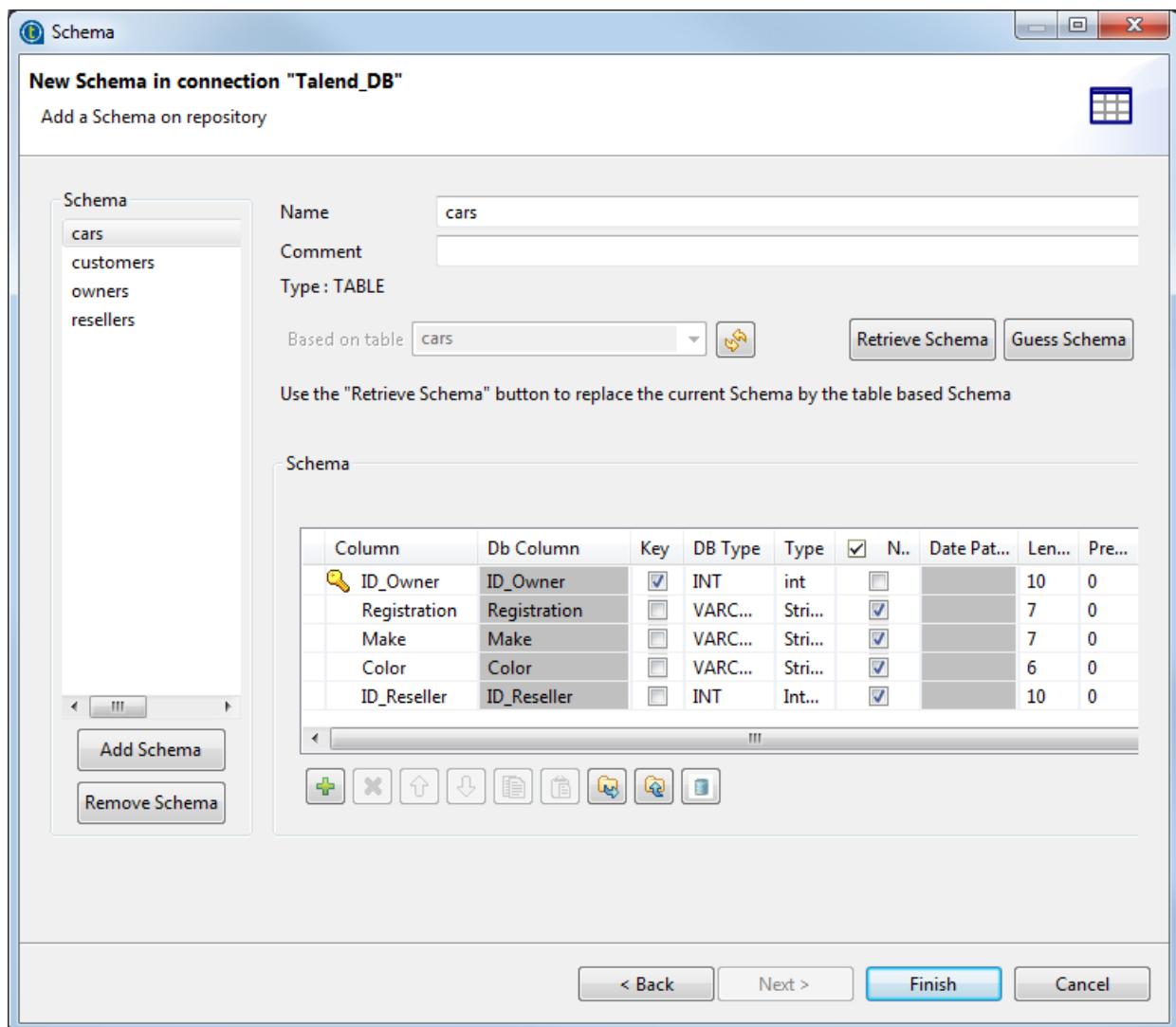
### About this task

Once you have the filtered list of the database objects, do the following to load the schemas of the desired objects onto your Repository:

### Procedure

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.

**Note:** 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.

**Warning:** 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. For more information, see Talend Help Center (<https://help.talend.com>).

**Tip:** If you find a certain data type of the database not yet supported by **Talend**, you can edit the mapping file for that database to enable conversion between the database data type and the corresponding **Talend** data type. For more information, see [Type mapping](#) on page 376.

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 will be saved in the corresponding schema folders 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 [Using centralized metadata in a Job](#) on page 344 and [Setting a repository schema in a Job](#) on page 41.

## 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.

For an example of using a JDBC connection, see Data Integration Job Examples on Talend Help Center (<https://help.talend.com>).

### Creating a JDBC connection and importing a database driver

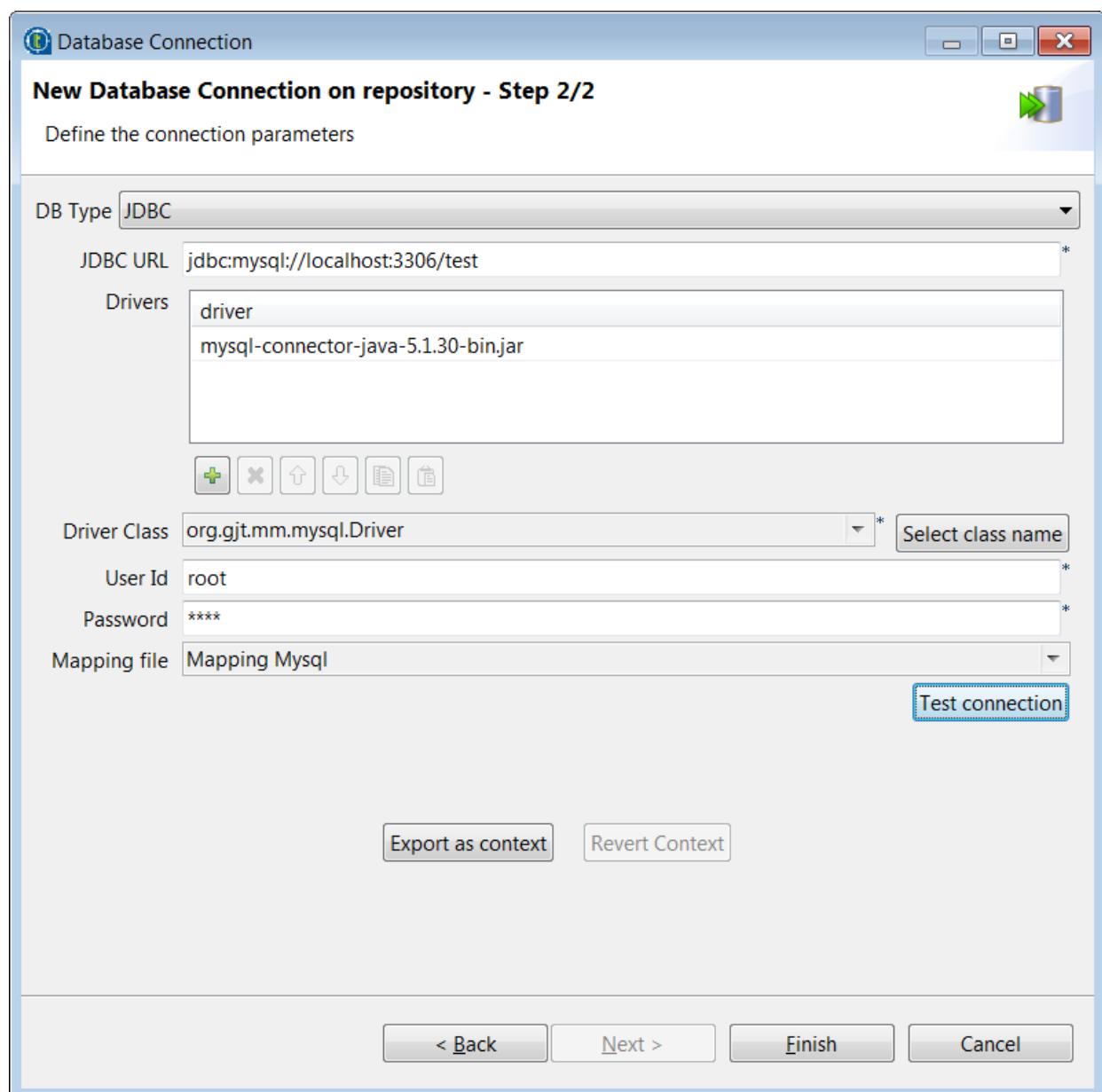
#### Procedure

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](#) on page 190.
3. Select **JDBC** from the **DB Type** list.



4. 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.

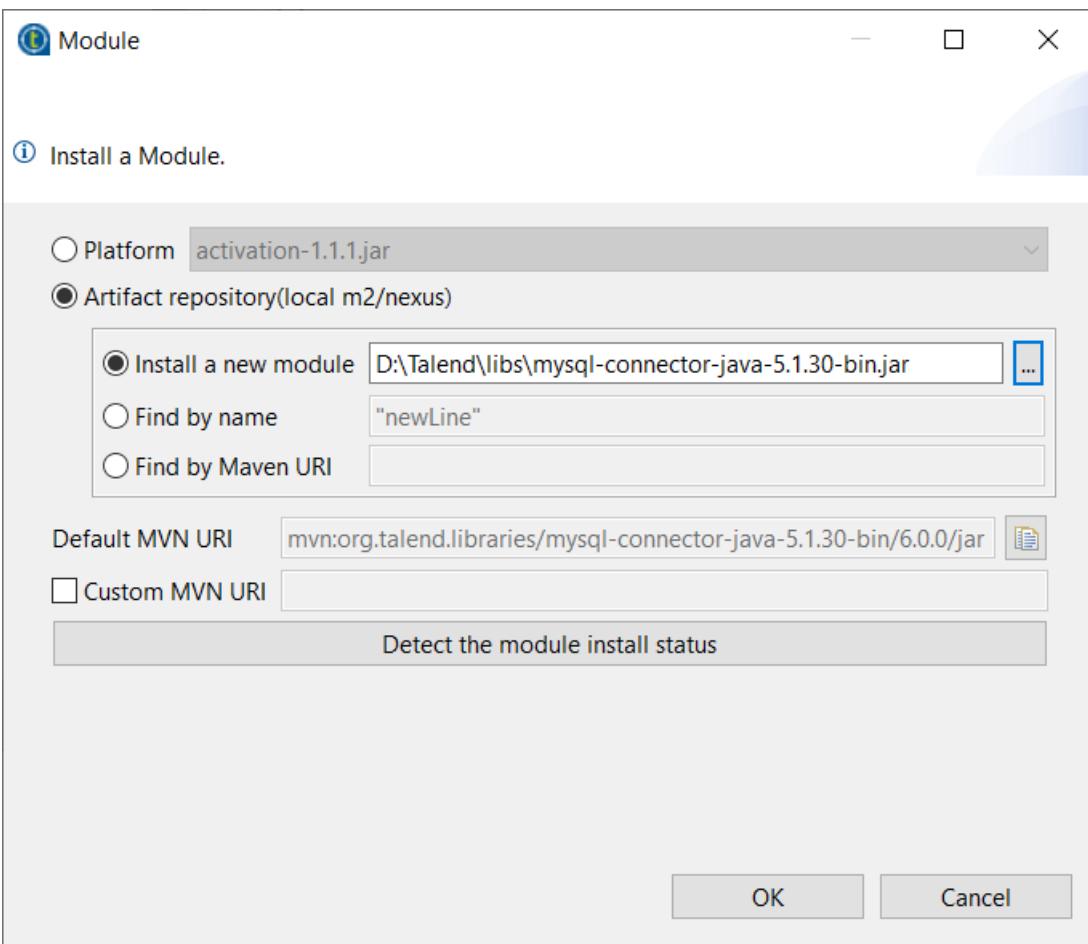
**Note:** Depending on the connection, you may need to import several driver files. For example, when connecting to Google BigQuery, import each jar file extracted from the zip file. For more information, see the procedure to [install and use the JDBC driver](#).

5. In the **Drivers** table, add one row to the table by clicking the **[+]** button.

driver
'newLine'

**Actions:**

6. Click the newly added row and click the [...] button to open the **Module** dialog box where you can import the external library.



7. If you have installed the library using the **Modules** view:

- Select the **Platform** option and then select the library from the list.
- Select the **Artifact repository (local m2/nexus)** > **Find by name** or **Artifact repository (local m2/nexus)** > **Find by Maven URI** option, then specify the full name or Maven URI of the library module, and click the **Detect the module install status** button to validate its installation status.

8. If you have stored the library file in a local directory:

- a) Select the **Artifact repository (local m2/nexus)** option.
- b) Select the **Install a new module** option, and click the [...] button to browse to library file.
- c) If you need to customize the Maven URI of the library, select the **Custom MVN URI** check box, specify the new URI, and then click the **Detect the module install status** button to validate its installation status.

**Note:**

Changing the Maven URI for an external module will affect all the components and metadata connections that use that module within the project.

When working on a remote project, your custom Maven URI settings will be automatically synchronized to the Talend Artifact Repository and will be used when other users working on the same project install the external module.

9. Click **OK** to confirm your changes.

The imported library file is listed in the **Drivers** table.

**Note:** You can replace or delete the imported library, or import new libraries if needed.

## Completing the JDBC connection details

### Procedure

1. Fill in the connection details as follows:
  - Fill in the **JDBC URL** used to access the database server.
  - In the **Driver Class** field, specify the main class of the driver allowing to communicate with the database.
  - Fill in your database user authentication data in **User Id** and **Password** fields.
  - In the **Mapping file** list, select the mapping that allows the database Type to match the Java type of data on the schema according to the type of database you are connecting to.

**Note:** The mapping files are XML files that you can manage via **Window > Preferences > Talend > Specific Settings > Metadata of TalendType**. For more information, see [Accessing mapping files and defining type mappings on page 376](#).

2. Click **Test connection** to check your connection.
3. Click **Finish** to close the connection setup wizard.

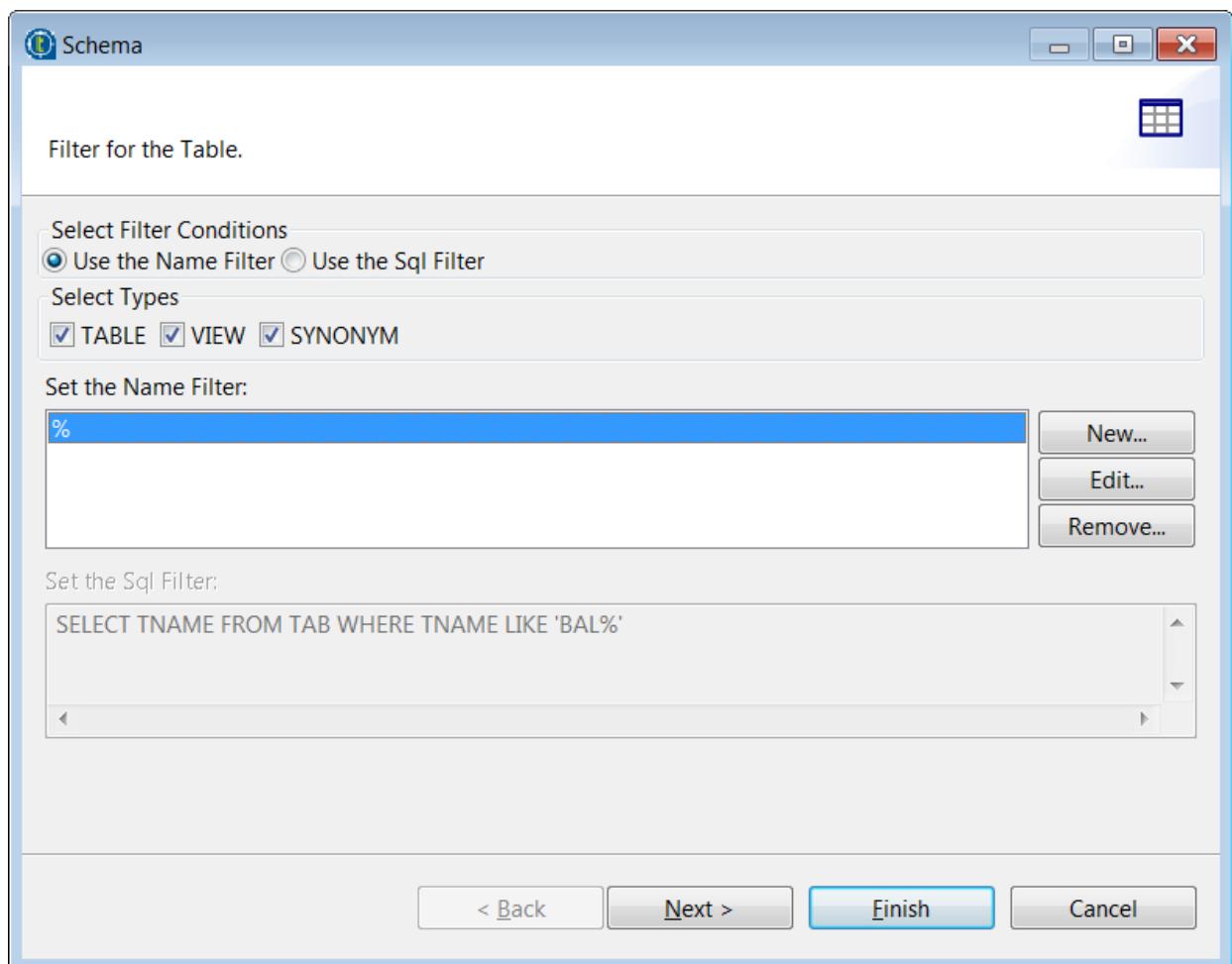
The newly created JDBC connection is now available in the **Repository** tree view and it displays several folders including **Queries** (for the SQL queries you save) and **Table schemas** that will gather all schemas linked to this DB connection upon schema retrieval.

## Retrieving table schemas

### Procedure

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](#) on page 195.  
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.

**Warning:** 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. For more information, see Talend Help Center (<https://help.talend.com>).

**Tip:** If you find a certain data type of the database not yet supported by **Talend**, you can edit the mapping file for that database to enable conversion between the database data type and the corresponding **Talend** data type. For more information, see [Type mapping](#) on page 376.

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 [Using centralized metadata in a Job](#) on page 344 and [Setting a repository schema in a Job](#) on page 41.

## 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.

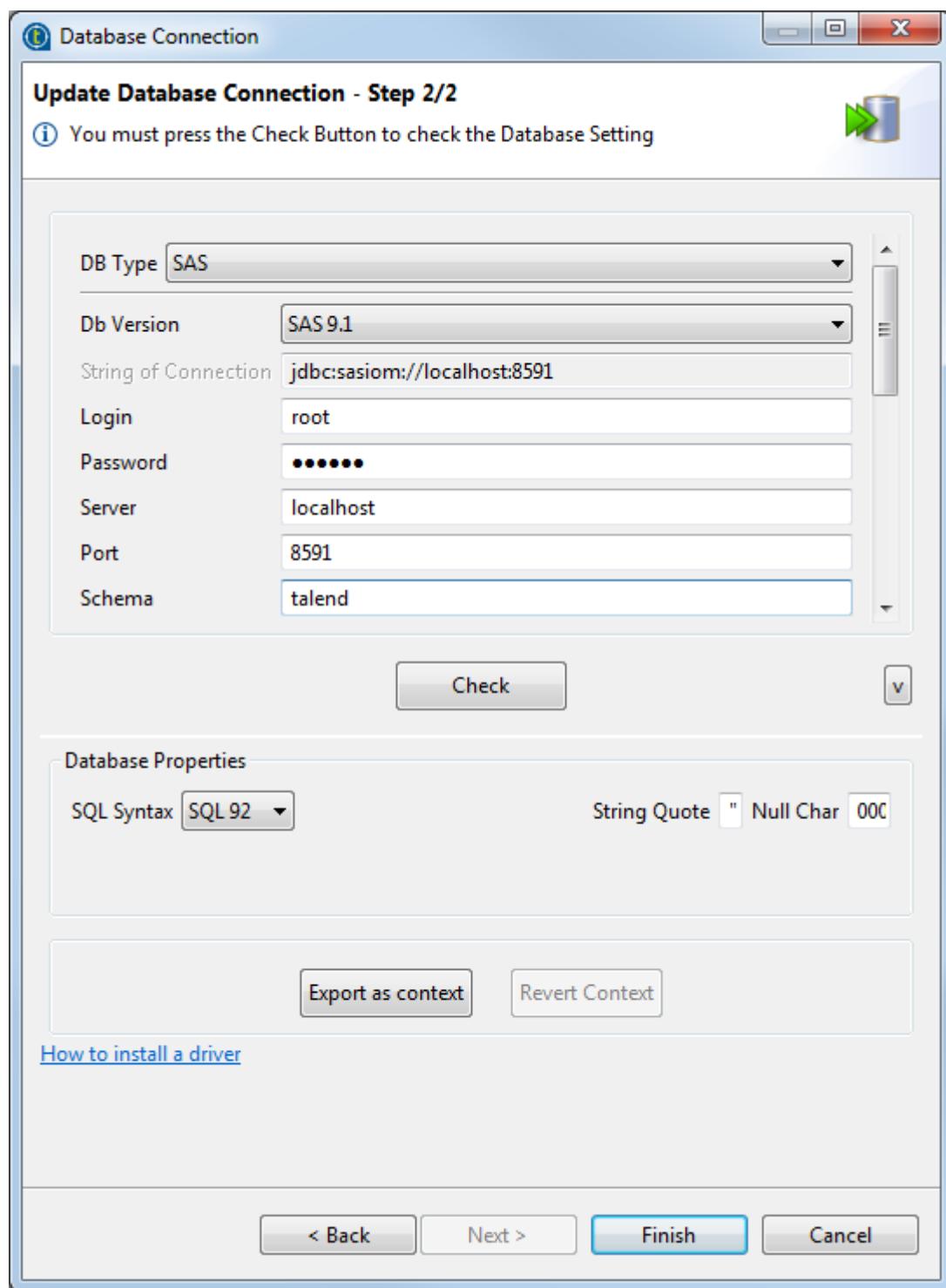
### Prerequisites:

- 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.
- 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.

## Setting up a SAS connection

### Procedure

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](#) on page 190.
3. In the **DB Type** field of the **Database Connection** wizard, select **SAS** and fill in the fields that follow with SAS connection information.



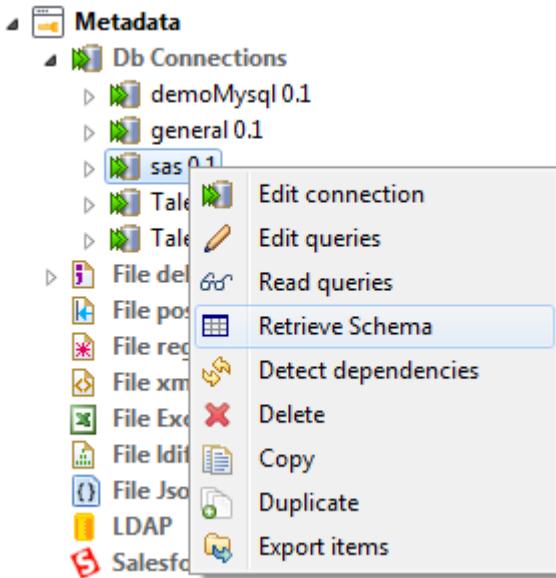
4. If needed, click the **Check** tab to verify if your connection is successful.
5. If needed, define the properties of the database in the corresponding fields in the **Database Properties** area.
6. Click **Finish** to validate your changes and close the wizard.

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.

## Retrieving SAS table schemas

## Procedure

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](#) on page 194.

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 [Using centralized metadata in a Job](#) on page 344 and [Setting a repository schema in a Job](#) on page 41.

## 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.

**Note:**

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.

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 [Using centralized metadata in a Job](#) on page 344 and [Setting a repository schema in a Job](#) on page 41.

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.

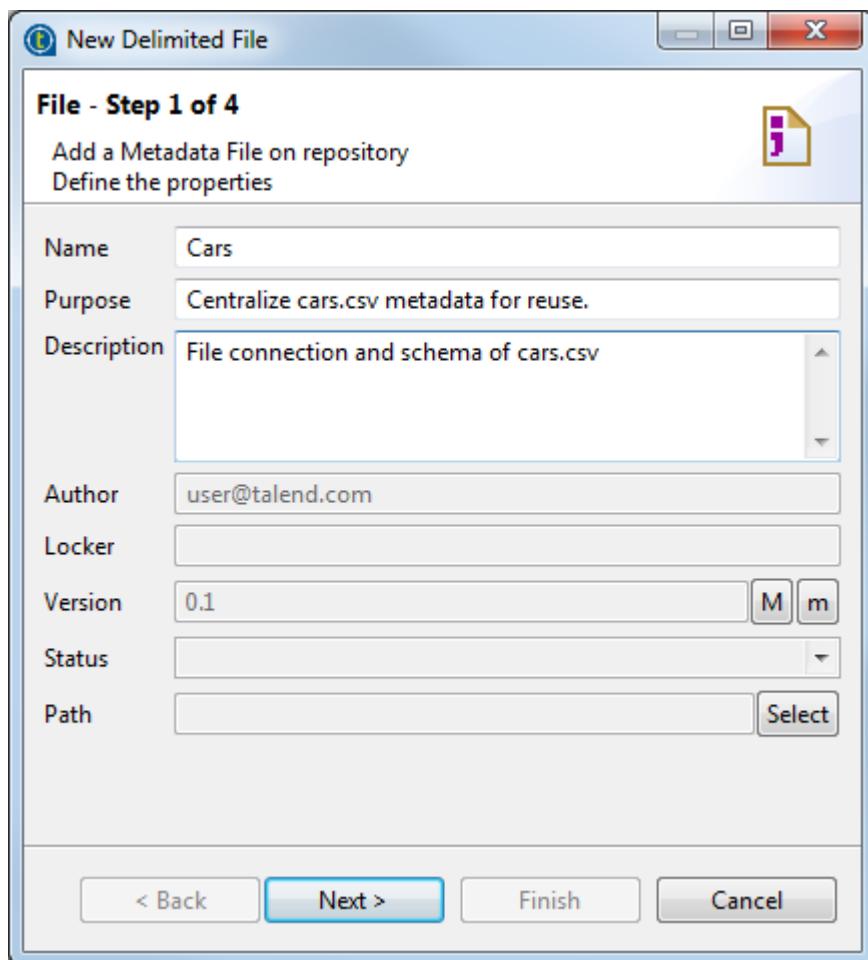
To add a new schema to an existing file connection, right-click the connection from the **Repository** tree view and select **Retrieve Schema** from the contextual menu.

To edit an existing file schema, right-click the schema from the **Repository** tree view and select **Edit Schema** from the contextual menu.

## Defining the general properties

### Procedure

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.



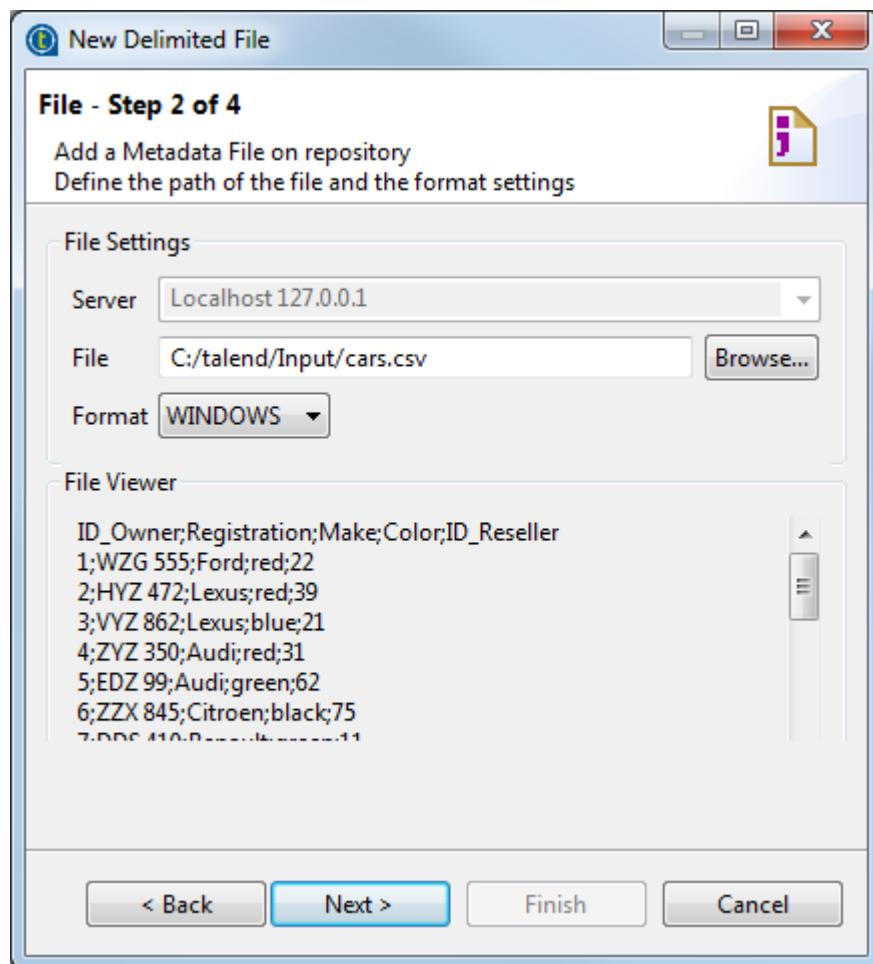
2. If needed, set the version and status in the **Version** and **Status** fields respectively. You can also manage the version and status of a repository item in the **Project Settings** dialog box. For more information, see [Version management](#) on page 382 and [Status management](#) on page 383 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

### Procedure

1. Specify the full path of the source file in the **File** field, or click the **Browse...** button to search for the file.

**Note:** The Universal Naming Convention (UNC) path notation is not supported. If your source file is on a LAN host, you can first map the network folder into a local drive.

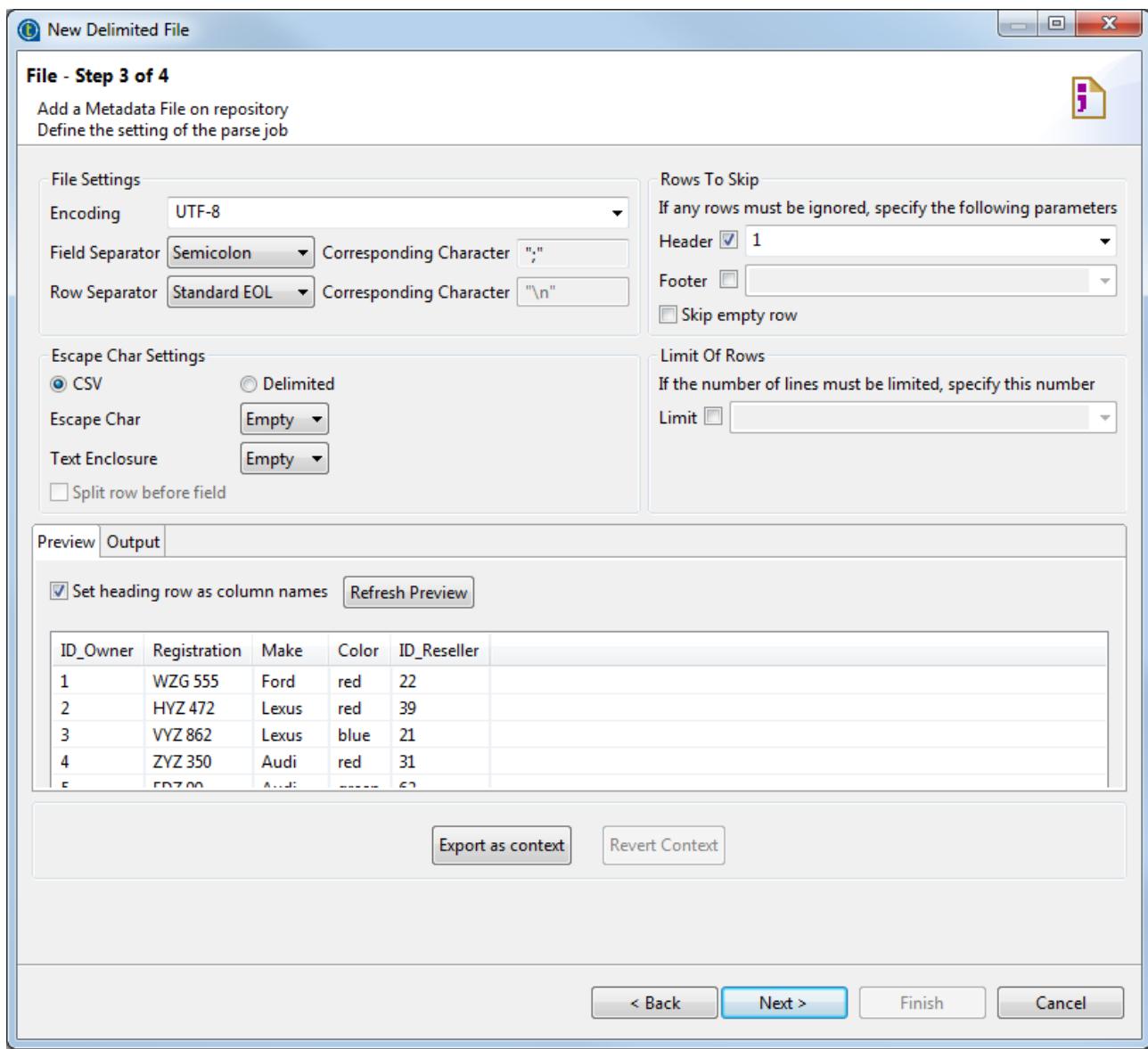


2. Select the OS **Format** the file was created in. This information is used to prefill subsequent step fields. If the list doesn't include the appropriate format, ignore it.
3. The **File viewer** gives an instant picture of the file loaded. Check the file consistency, the presence of header and more generally the file structure.
4. Click **Next** to proceed to the next step.

## Defining the file parsing parameters

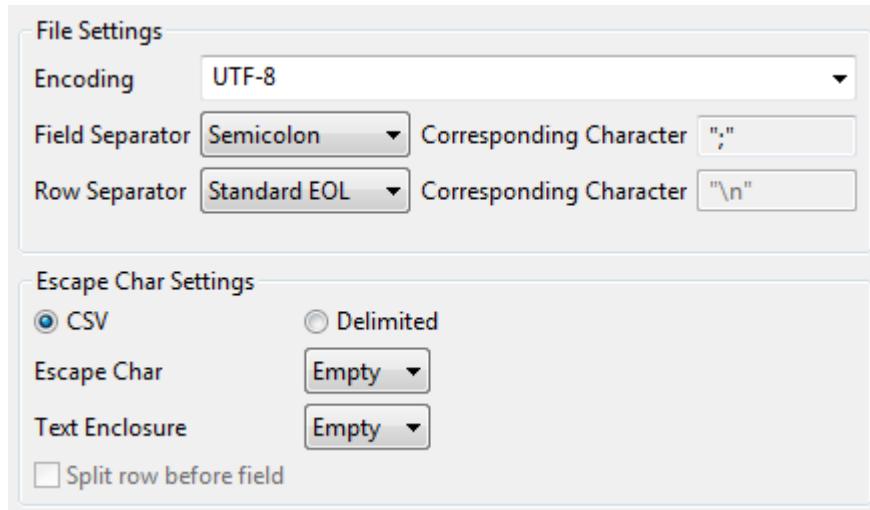
### About this task

On this view, you can refine the various settings of your file so that the file schema can be properly retrieved.



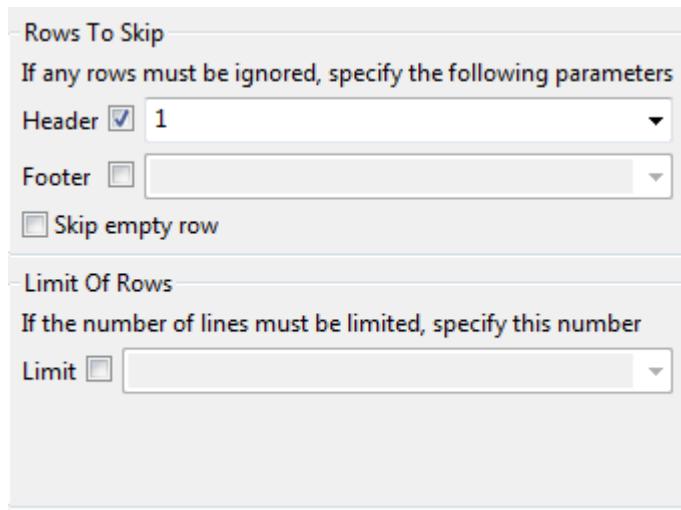
## Procedure

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.

- 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.



- 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.
- In the **File Preview** panel, view the new settings impact.
- 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**

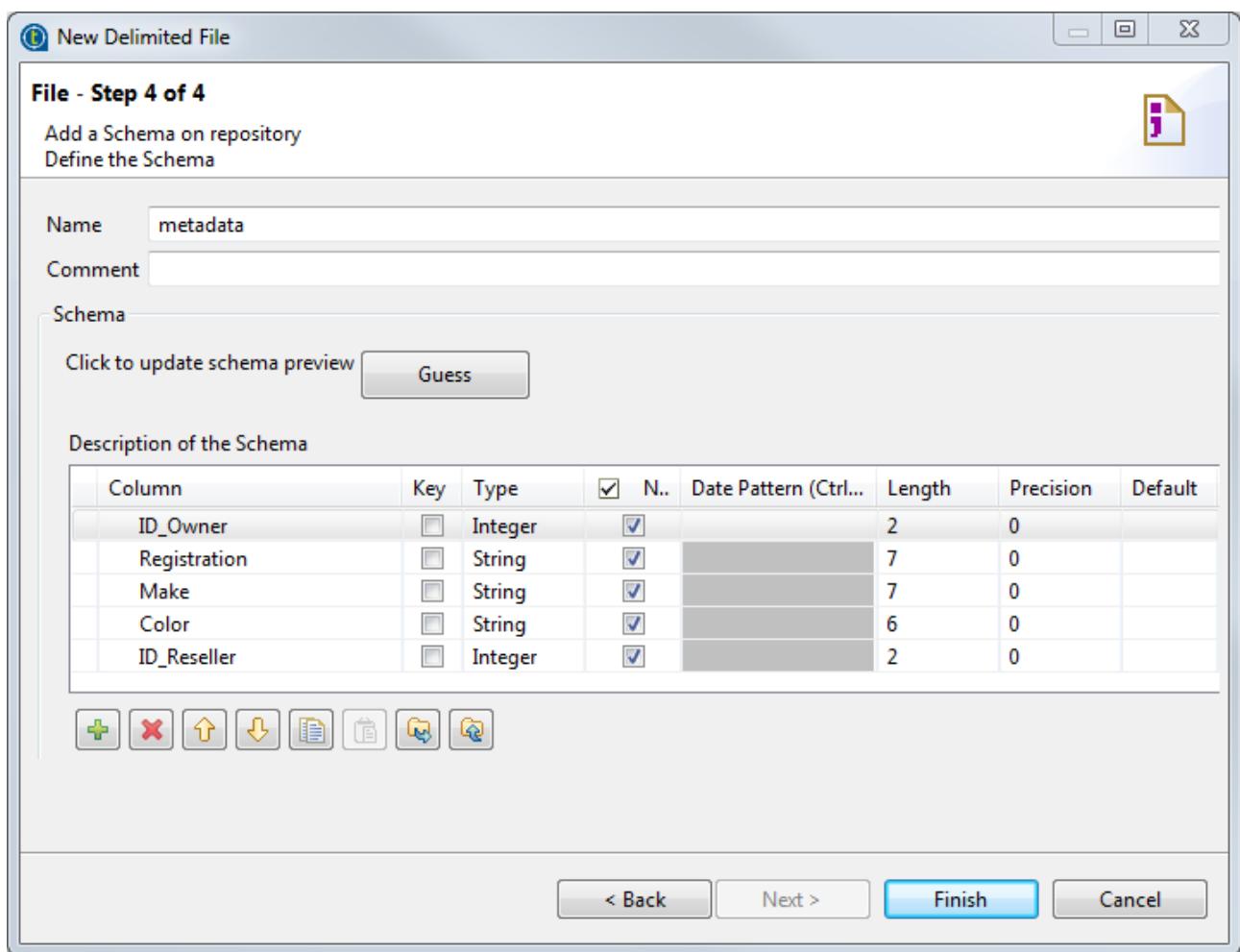
ID_Owner	Registration	Make	Color	ID_Reseller	
1	WZG 555	Ford	red	22	
2	HYZ 472	Lexus	red	39	
3	VYZ 862	Lexus	blue	21	
4	ZYZ 350	Audi	red	31	
5	EDZ 99	Audi	green	62	
6	ZZX 845	Citroen	black	75	

- Click **Refresh** on the preview panel for the settings to take effect and view the result on the viewer.
- Click **Next** to proceed to the final step to check and customize the generated file schema.

## Checking and customizing the file schema

### About this task

The last step shows the Delimited File schema generated. You can customize the schema using the toolbar underneath the table.



## Procedure

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. 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.
3. Click **Finish**. The new schema is displayed under the relevant **File Delimited** connection node in the **Repository** tree view.

## 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.

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 [Using centralized metadata in a Job on page 344](#) and [Setting a repository schema in a Job on page 41](#).

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.

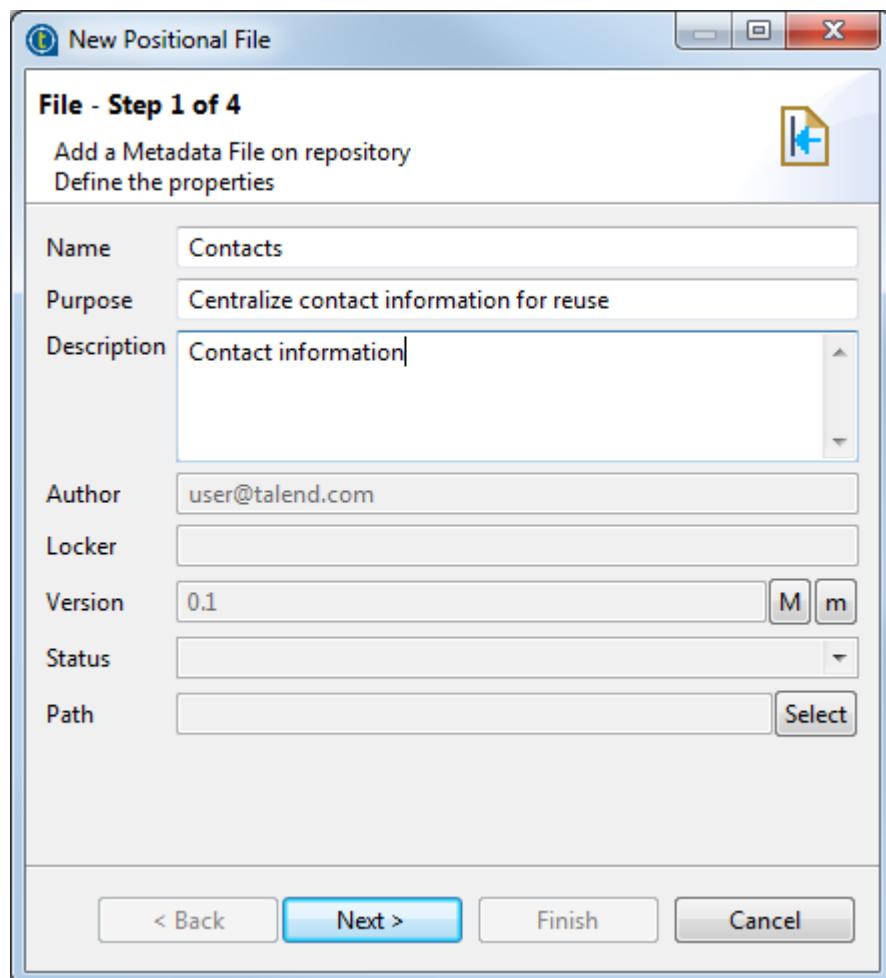
To add a new schema to an existing file connection, right-click the connection from the **Repository** tree view and select **Retrieve Schema** from the contextual menu.

To edit an existing file schema, right-click the schema from the **Repository** tree view and select **Edit Schema** from the contextual menu.

### Defining the general properties of the File Positional connection

#### Procedure

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](#) on page 382 and [Status management](#) on page 383 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.

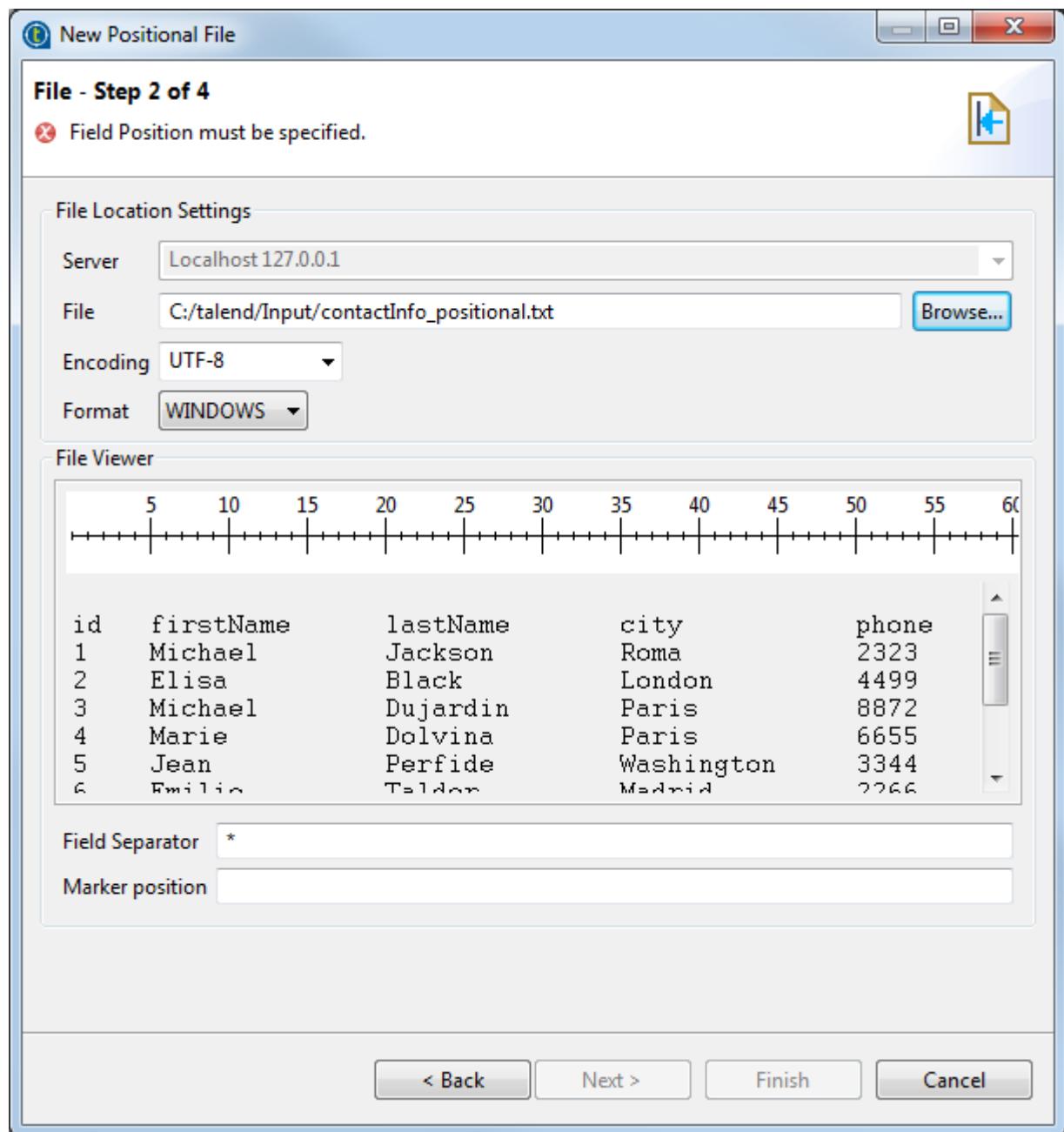
## Defining the file path, format and marker positions

### Procedure

1. Specify the full path of the source file in the **File** field, or click the **Browse...** button to search for the file.

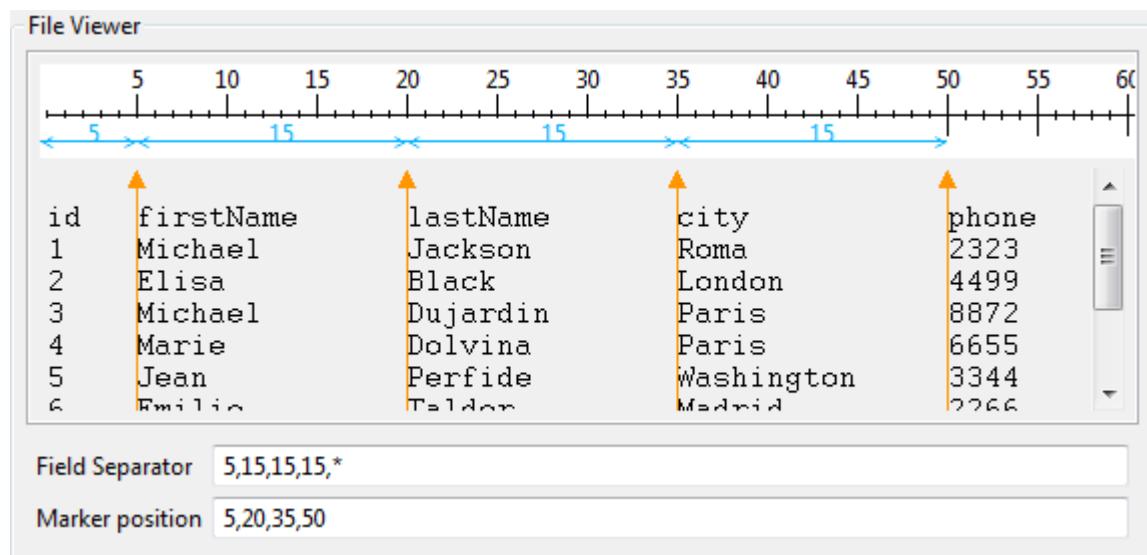
**Note:** The Universal Naming Convention (UNC) path notation is not supported. If your source file is on a LAN host, you can first map the network folder into a local drive.

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.

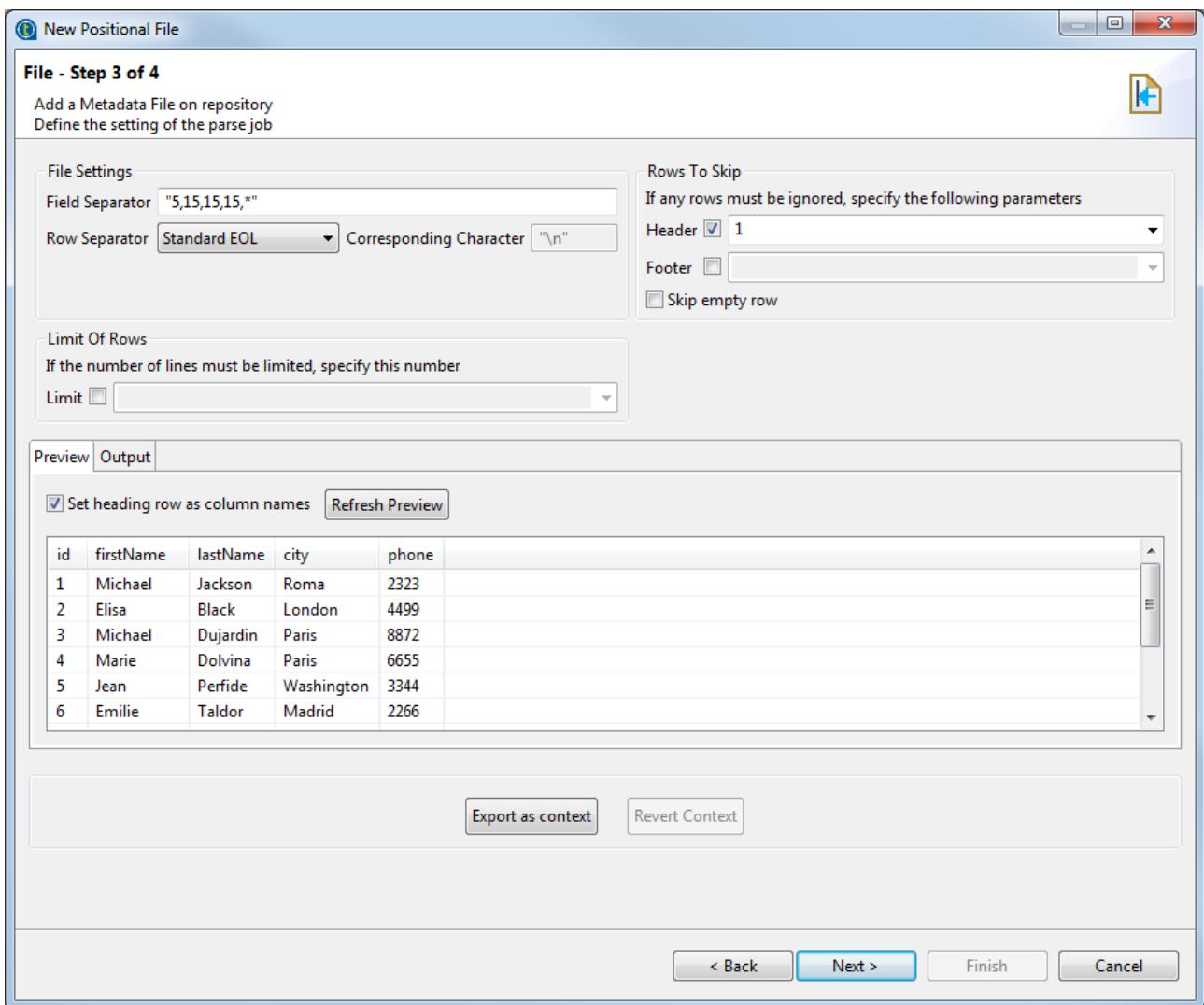
To move a marker, press its arrow and drag it to the new position. To remove a marker, press its arrow and drag it towards the ruler until a **X** icon appears.

4. Click **Next** to continue.

## Defining the parsing parameters of your positional file

### About this task

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.



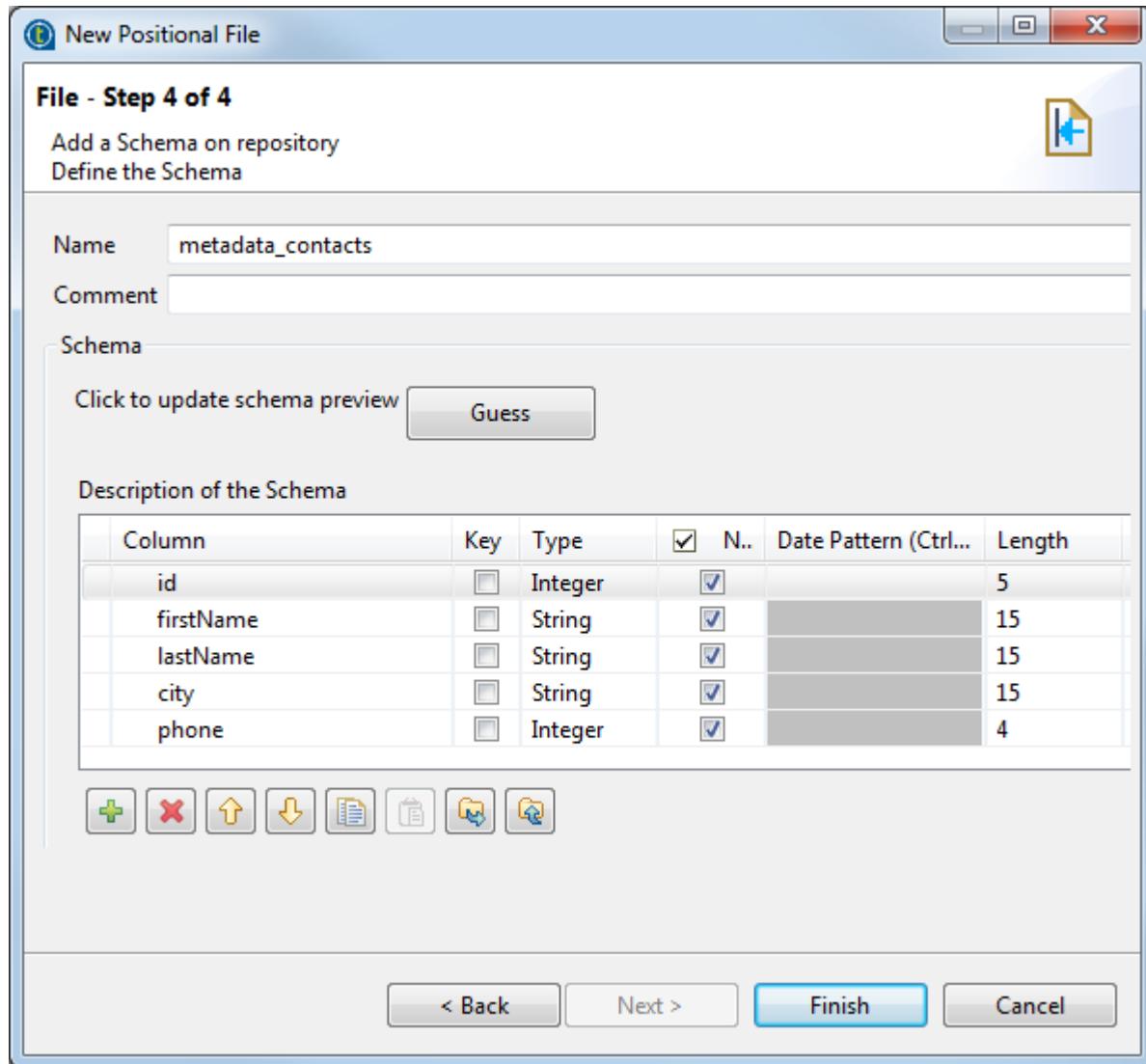
## Procedure

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 schema of your positional file

### About this task

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.



### Procedure

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.

## 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.

**Note:**

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.

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 [Using centralized metadata in a Job](#) on page 344 and [Setting a repository schema in a Job](#) on page 41.

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.

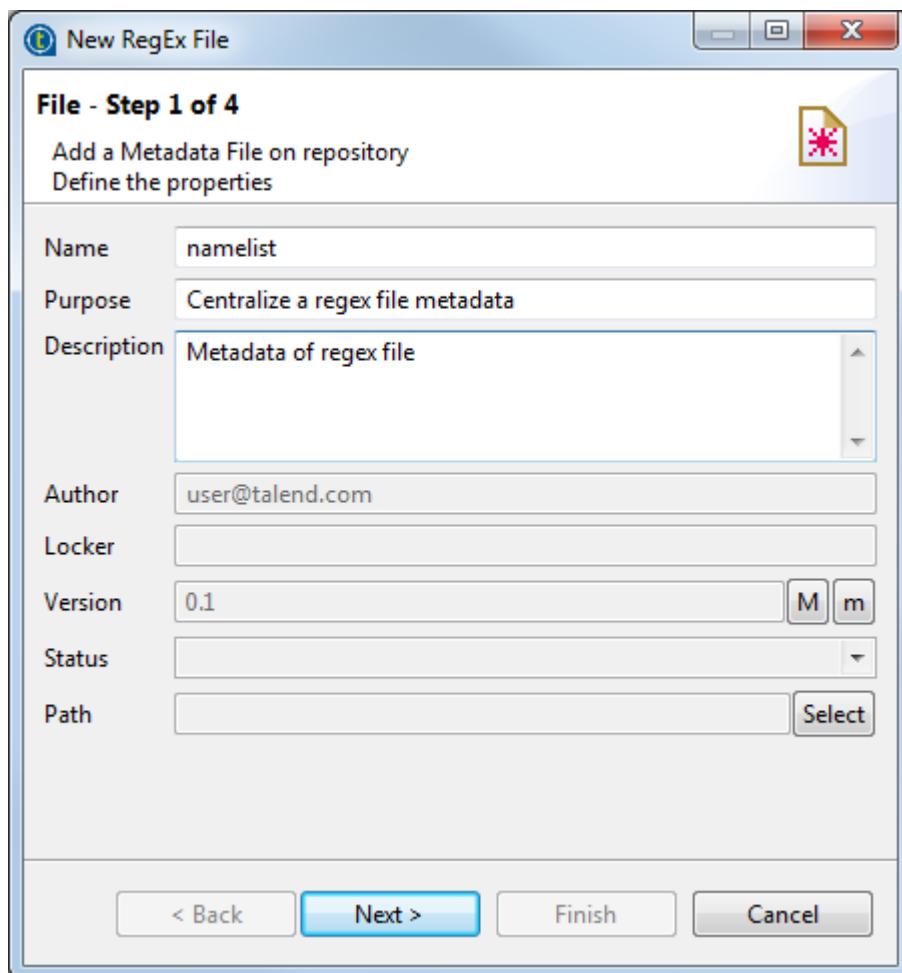
To add a new schema to an existing file connection, right-click the connection from the **Repository** tree view and select **Retrieve Schema** from the contextual menu.

To edit an existing file schema, right-click the schema from the **Repository** tree view and select **Edit Schema** from the contextual menu.

### Defining the general properties of the File Regex connection

#### Procedure

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](#) on page 382 and [Status management](#) on page 383 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.

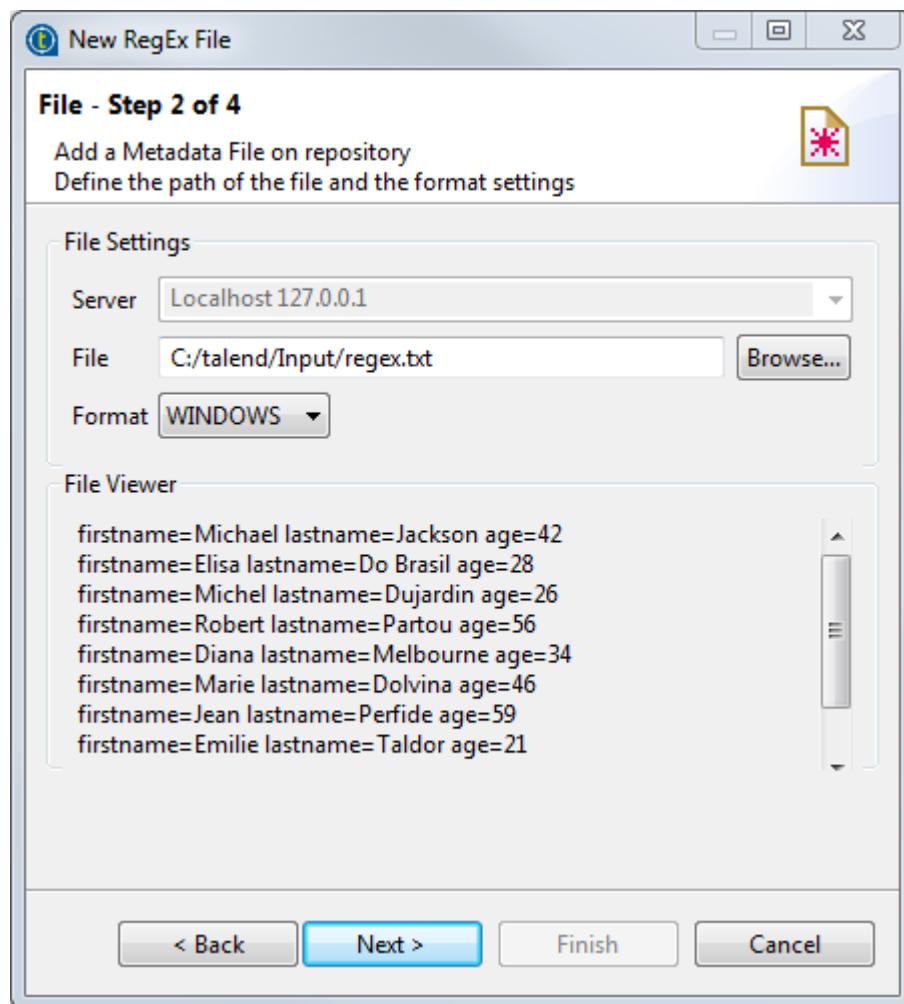
## Defining the path and format of your Regex file

### Procedure

1. Specify the full path of the source file in the **File** field, or click the **Browse...** button to search for the file.

**Note:** The Universal Naming Convention (UNC) path notation is not supported. If your source file is on a LAN host, you can first map the network folder into a local drive.

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.

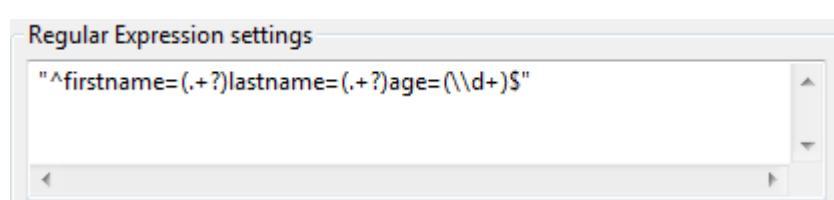
## Defining the parsing parameters of your Regex file

### About this task

On this view, you define the file parsing parameters so that the file schema can be properly retrieved.

### Procedure

- 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.



**Warning:**

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.

The screenshot shows a 'Preview' tab selected in a software interface. At the top, there are two tabs: 'Preview' (which is active) and 'Output'. Below the tabs are two buttons: 'Set heading row as column names' (unchecked) and 'Refresh Preview'. The main area displays a table with three columns labeled 'Column 0', 'Column 1', and 'Column 2'. The data consists of six rows:

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 schema of your Regex file

### Procedure

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.

## 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](#) on page 223.

For further information about writing an XML file, see [Setting up XML metadata for an output file](#) on page 233.

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.

## 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.

### Defining the general properties of the File XML connection

#### About this task

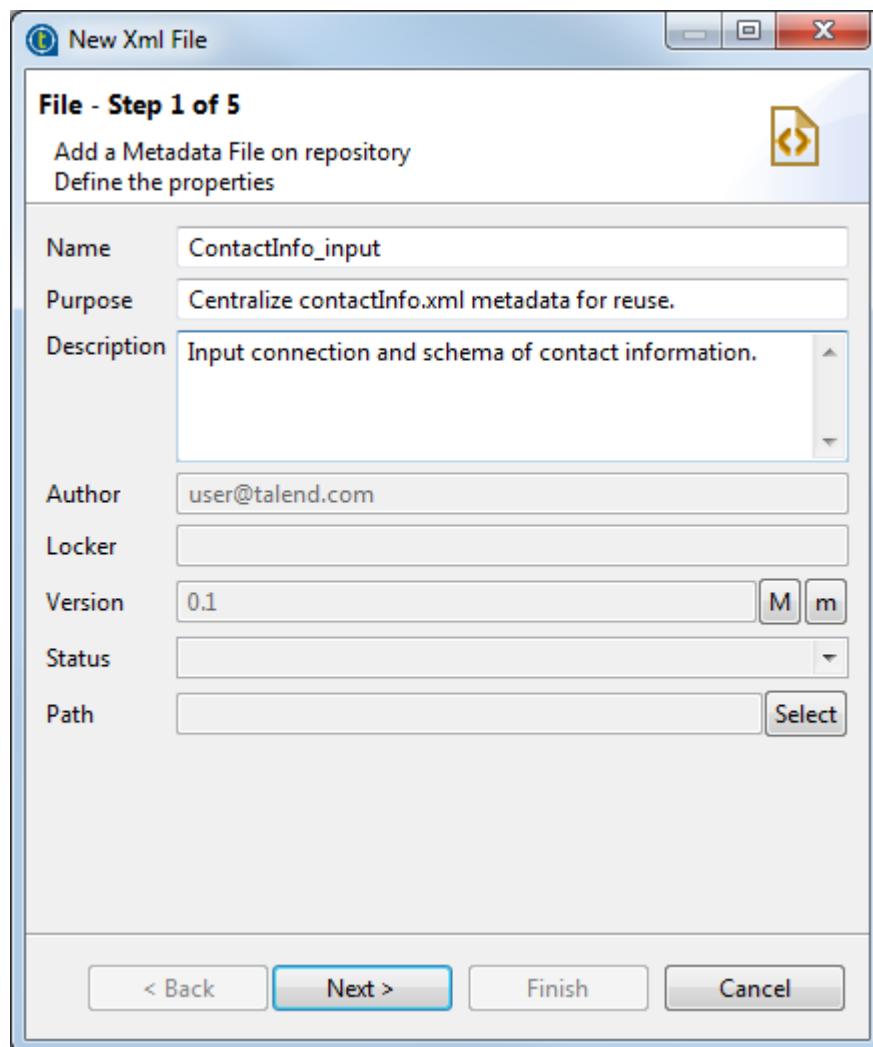
In this step, the general metadata properties such as the **Name**, **Purpose** and **Description** are set.

#### Procedure

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.

#### Note:

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](#) on page 382 and [Status management](#) on page 383 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.

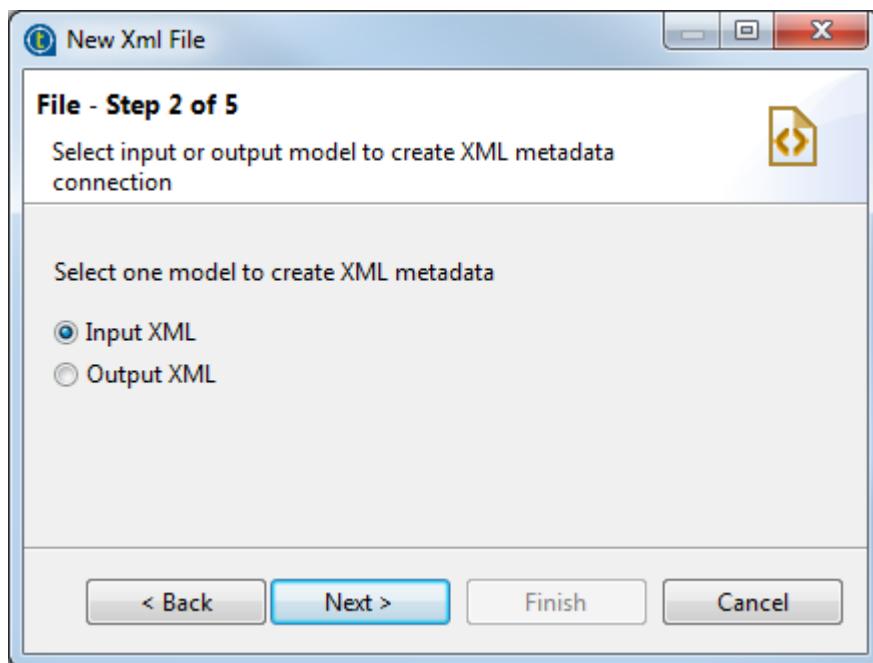
#### [Setting the type of metadata \(input\)](#)

#### [About this task](#)

In this step, the type of metadata is set as either input or output. For this procedure, the metadata of interest is input.

#### [Procedure](#)

1. In the dialog box, select **Input XML**.



- Click **Next** to upload the input file.

### Uploading an XML file

#### About this task

This procedure describes how to upload an XML file to obtain the XML tree structure.

The example input XML file used to demonstrate this step contains some contact information, and the structure is like the following:

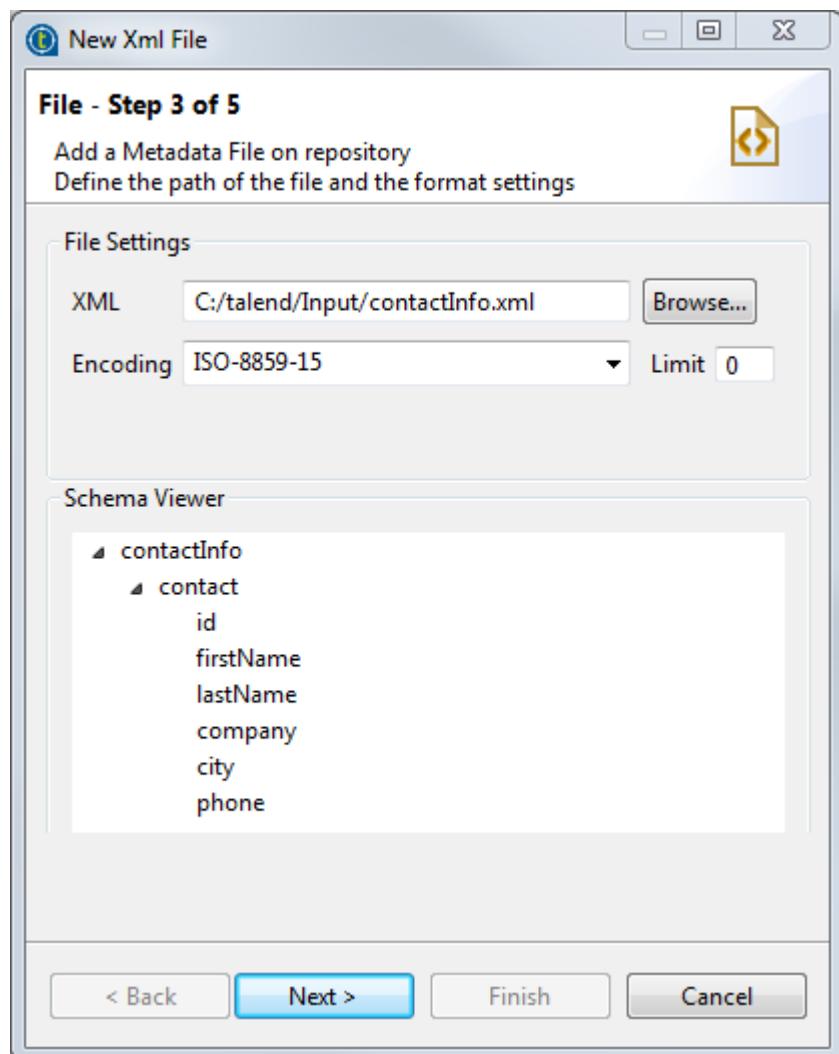
```
<contactInfo>
  <contact>
    <id>1</id>
    <firstName>Michael</firstName>
    <lastName>Jackson</lastName>
    <company>Talend</company>
    <city>Paris</city>
    <phone>2323</phone>
  </contact>
  <contact>
    <id>2</id>
    <firstName>Elisa</firstName>
    <lastName>Black</lastName>
    <company>Talend</company>
    <city>Paris</city>
    <phone>4499</phone>
  </contact>
  ...
</contactInfo>
```

To upload an XML file, do the following:

#### Procedure

- 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

#### About this task

This procedure describes how to upload an XSD file to obtain the XML tree structure.

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.

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
  <xs:element name="contactInfo">
    <xs:complexType>
      <xs:sequence>
        <xs:element maxOccurs="unbounded" ref="contact"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="contact">
    <xs:complexType>
      <xs: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>.

#### Note:

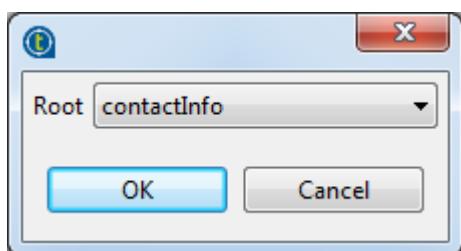
When loading an XSD file,

- 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.

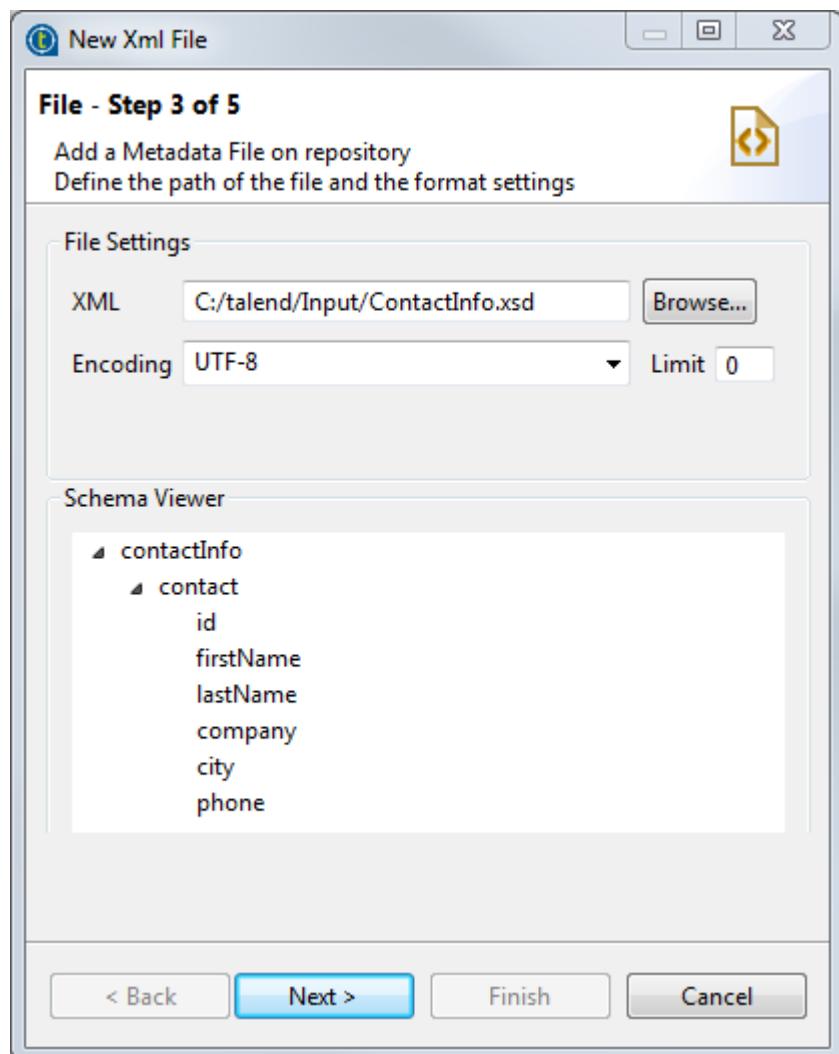
To load an XSD file, do the following:

#### Procedure

- Click **Browse...** and browse your directory to the XSD file to be uploaded. Alternatively, enter the access path to the file.
- 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.

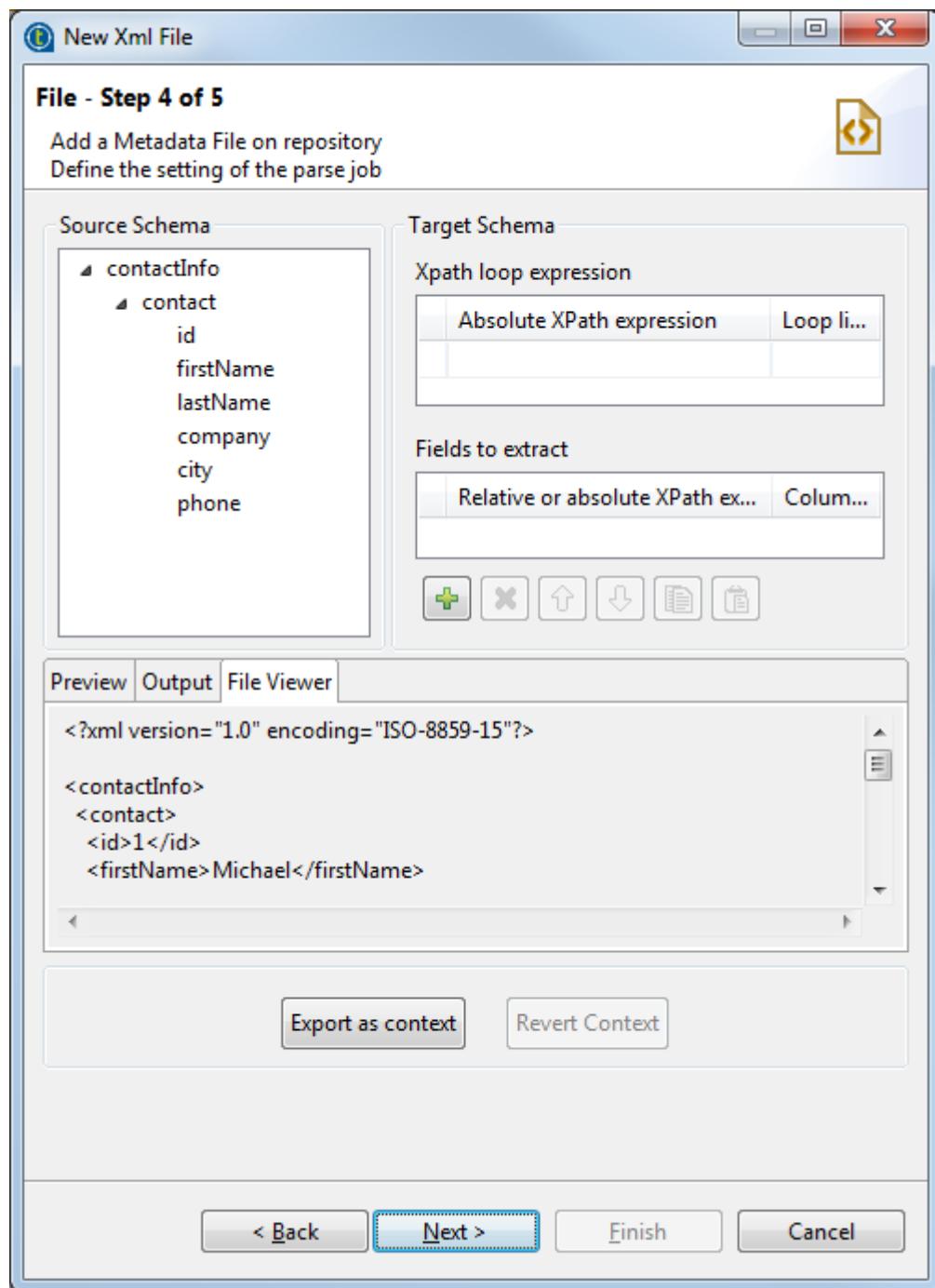


3. Enter the **Encoding** type in the corresponding field if the system does not detect it automatically.
4. In the **Limit** field, enter the number of columns on which the XPath query is to be executed, or 0 if you want to run it against all of the columns.
5. Click **Next** to define the schema parameters.

### Defining the schema

#### About this task

In this step, the schema parameters are set.



The schema definition window is composed of four views:

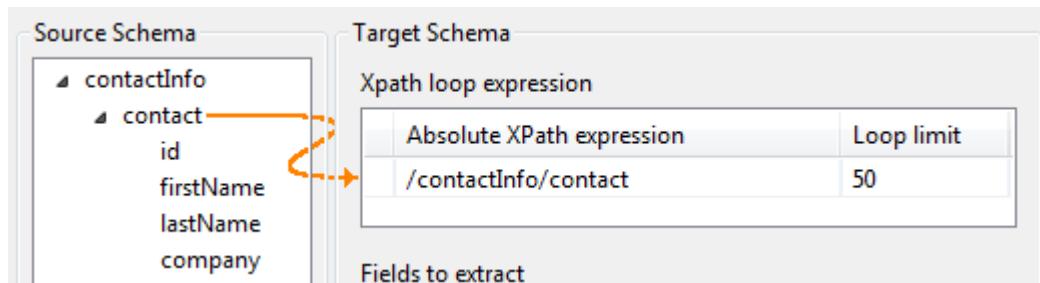
View	Description
Source Schema	Tree view of the XML file.
Target Schema	Extraction and iteration information.
Preview	Preview of the target schema, together with the input data of the selected columns displayed in the defined order.  <b>Note:</b> The preview functionality is not available if you loaded an XSD file.
File Viewer	Preview of the brute data.

First define an Xpath loop and the maximum number of times the loop can run. To do so:

### Procedure

- 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.

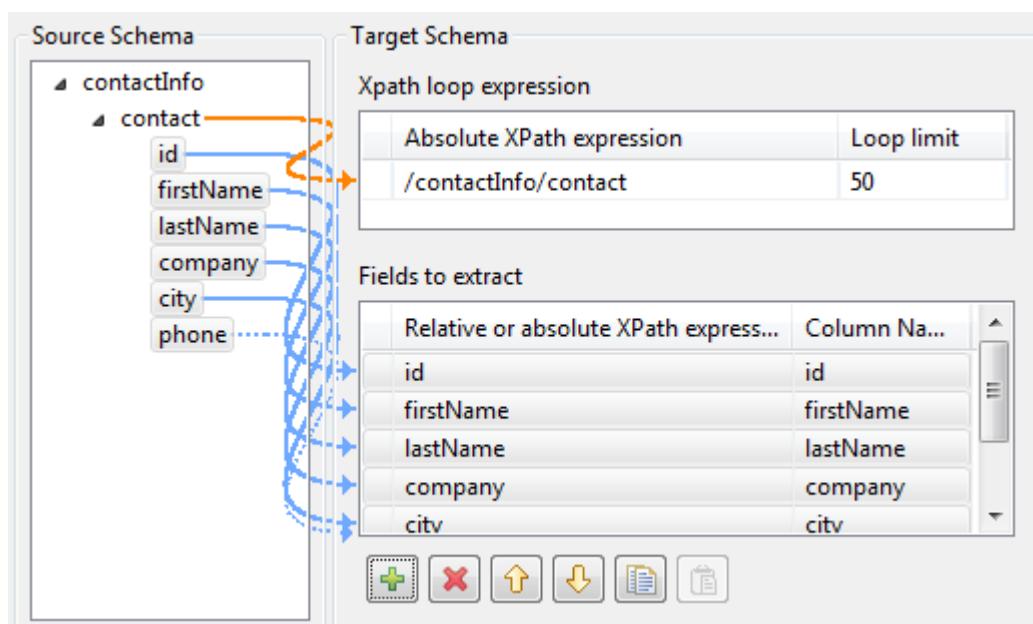
An orange arrow links the node to the corresponding expression.



**Note:** The **Xpath loop expression** field is mandatory.

- 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.
- 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.

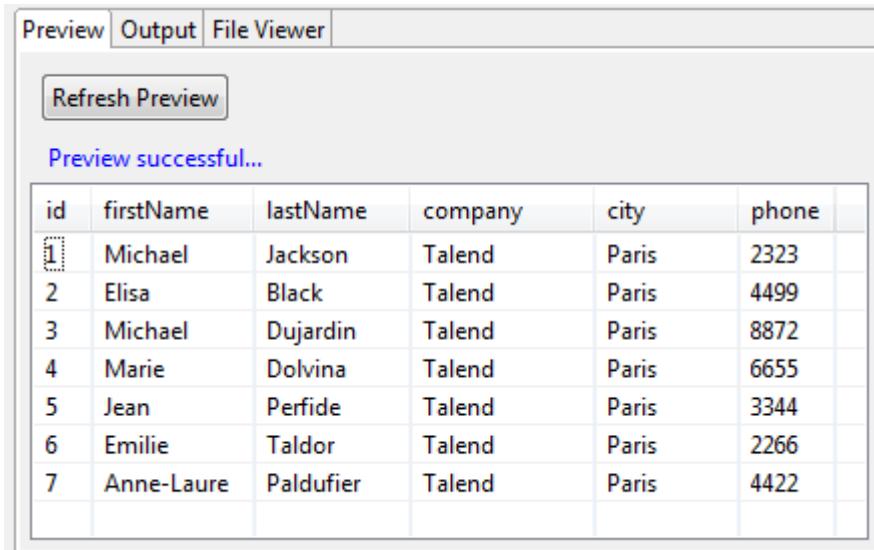
**Note:** You can select several nodes to drop on the table by pressing **Ctrl** or **Shift** and clicking the nodes of interest. The arrow linking an individual node selected on the **Source Schema** to the **Fields to extract** table are blue in colour. The other ones are gray.



- 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.

**Note:** The preview functionality is not available if you loaded an XSD file.



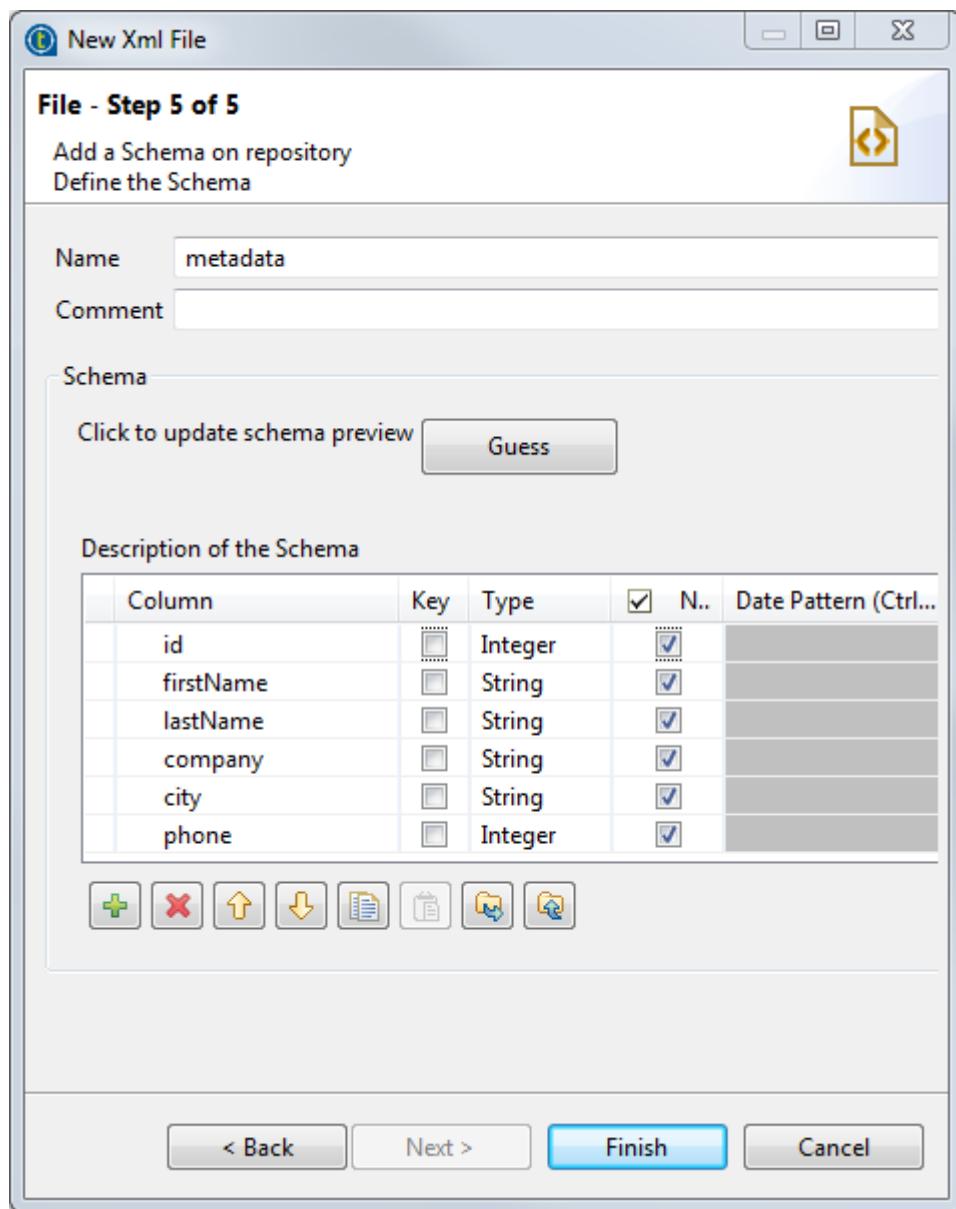
id	firstName	lastName	company	city	phone
1	Michael	Jackson	Talend	Paris	2323
2	Elisa	Black	Talend	Paris	4499
3	Michael	Dujardin	Talend	Paris	8872
4	Marie	Dolvina	Talend	Paris	6655
5	Jean	Perfide	Talend	Paris	3344
6	Emilie	Taldor	Talend	Paris	2266
7	Anne-Laure	Paldufier	Talend	Paris	4422

7. Click **Next** to check and edit the end schema.

### Finalizing the end schema

#### About this task

The schema generated displays the columns selected from the XML file and allows you to further define the schema.



## Procedure

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.

## Results

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 [Using centralized metadata in a Job](#) on page 344 and [Setting a repository schema in a Job](#) on page 41.

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.

## 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.

### Defining the general properties of the File XML connection for an output file

#### About this task

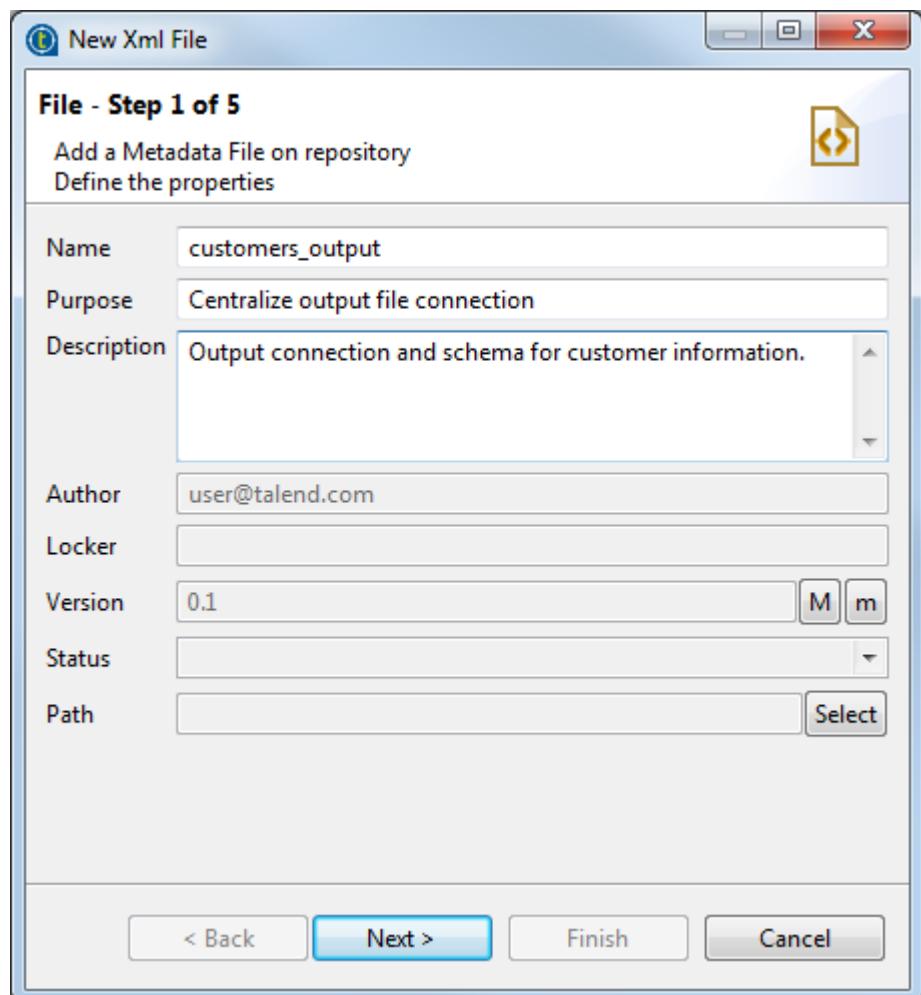
In this step, the general metadata properties such as the **Name**, **Purpose** and **Description** are set.

#### Procedure

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.

#### Note:

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](#) on page 382 and [Status management](#) on page 383 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.

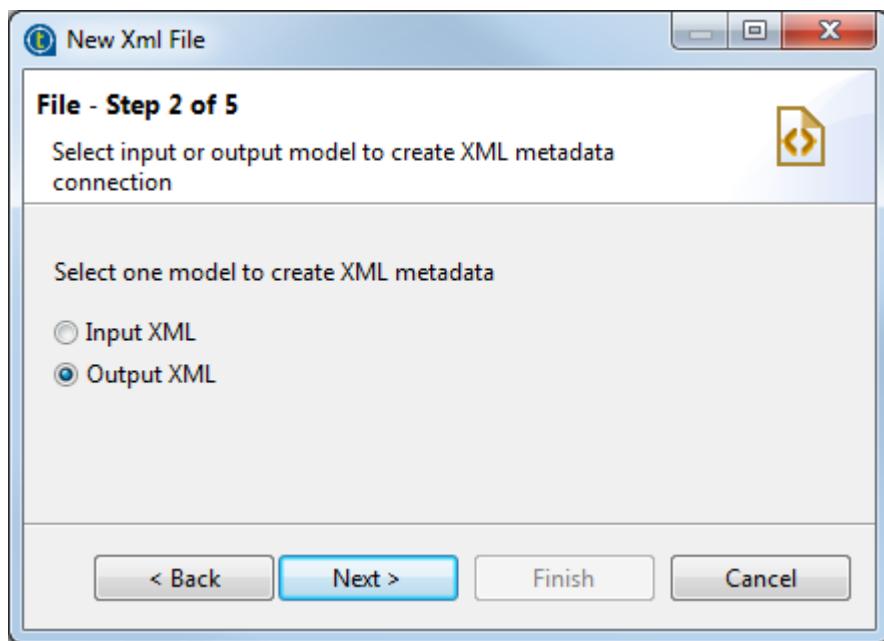
#### [Setting the type of metadata \(output\)](#)

#### **About this task**

In this step, the type of metadata is set as either input or output. For this procedure, the metadata of interest is output.

#### **Procedure**

1. From the dialog box, select **Output XML**.



- 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

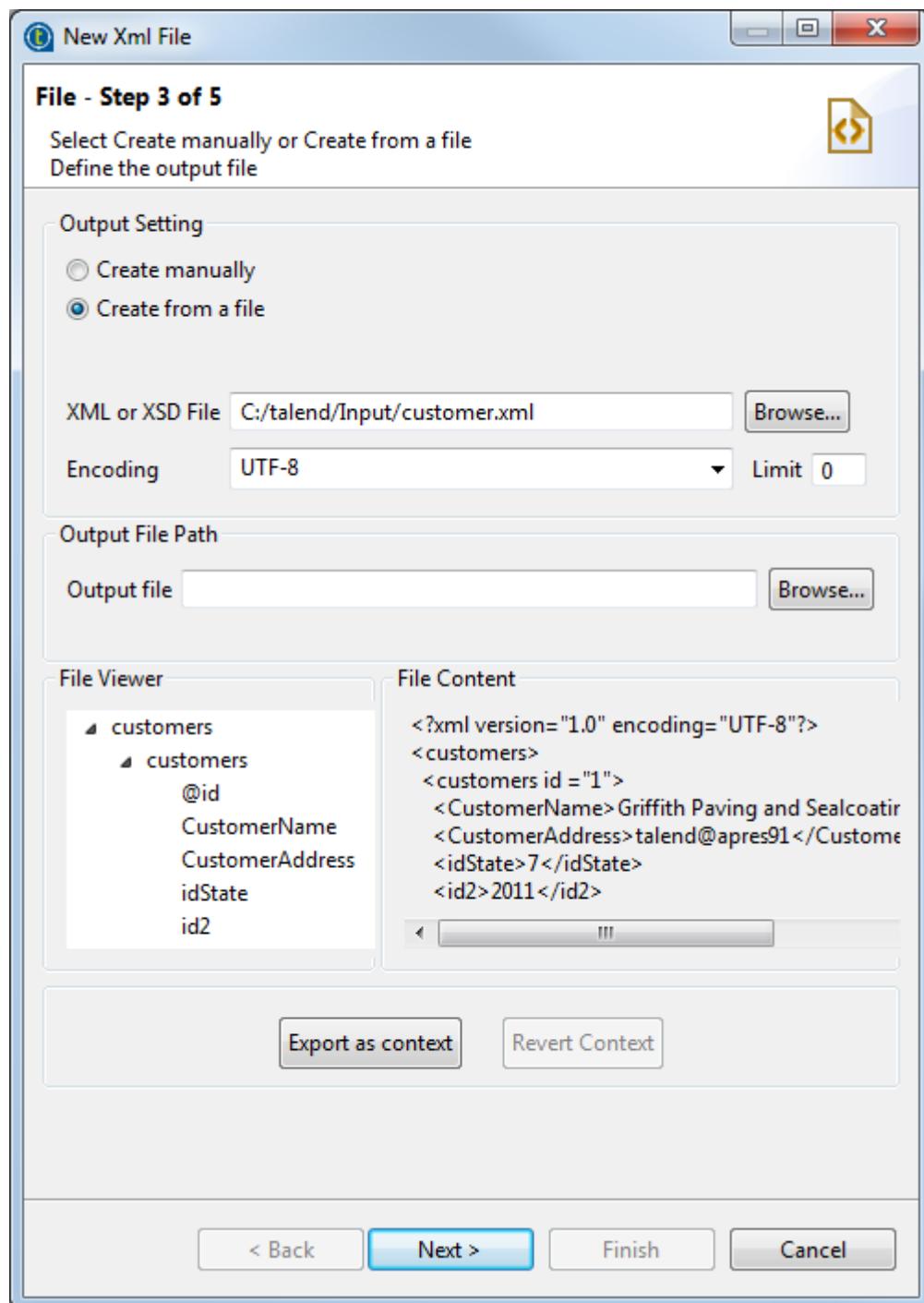
#### About this task

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 **tAdvancedFileOutputXML**.

To create the output XML structure from an XML file, do the following:

#### Procedure

- Select the **Create from a file** option.
  - 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

##### About this task

This procedure describes how to define the output XML file structure from an XSD file.

**Note:**

When loading an XSD file,

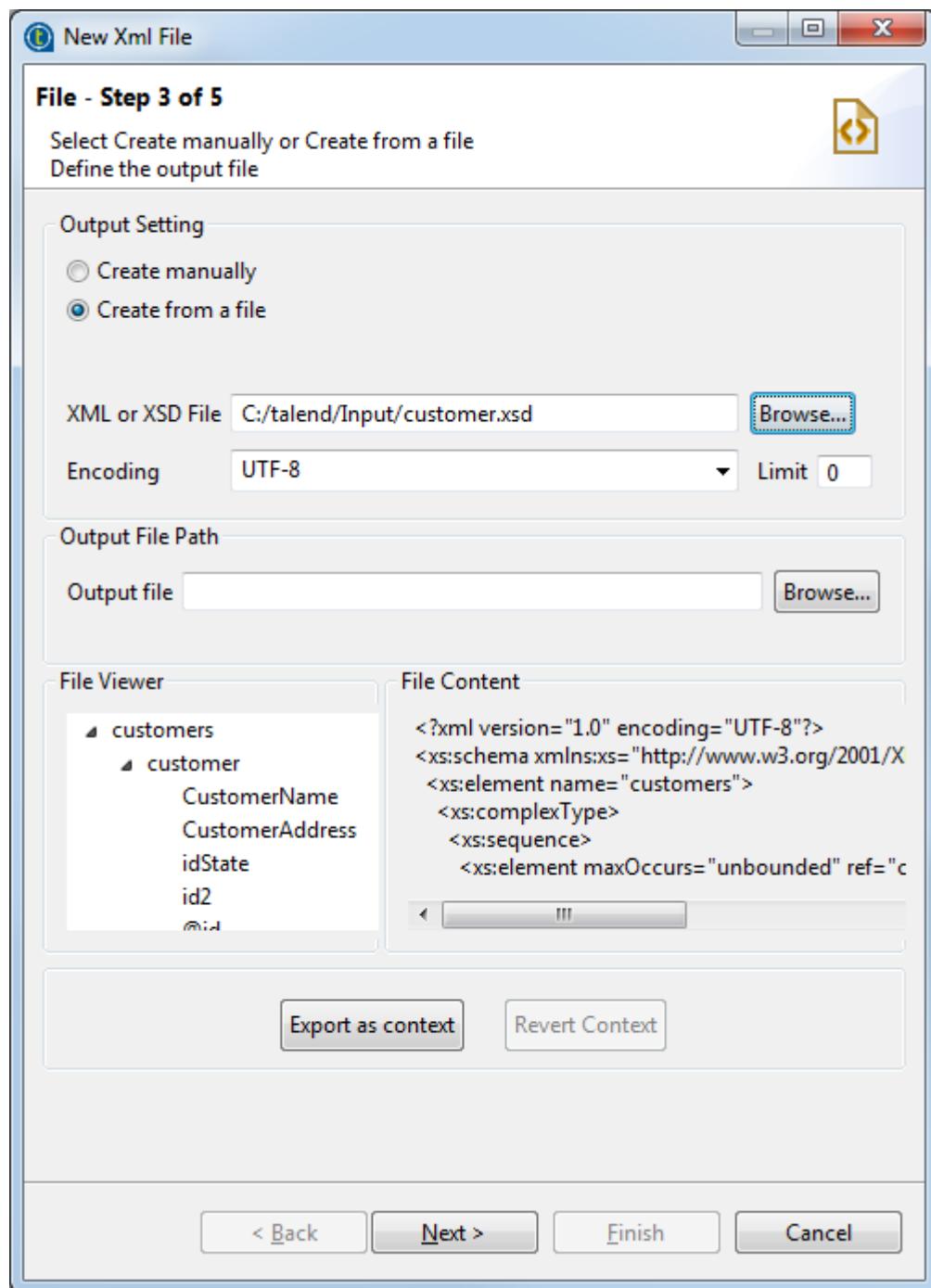
- the data will be saved in the **Repository**, and therefore the metadata will not be affected by the deletion or displacement of the file.
- you can choose an element as the root of your XML tree.

To create the output XML structure from an XSD file, do the following:

**Procedure**

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.

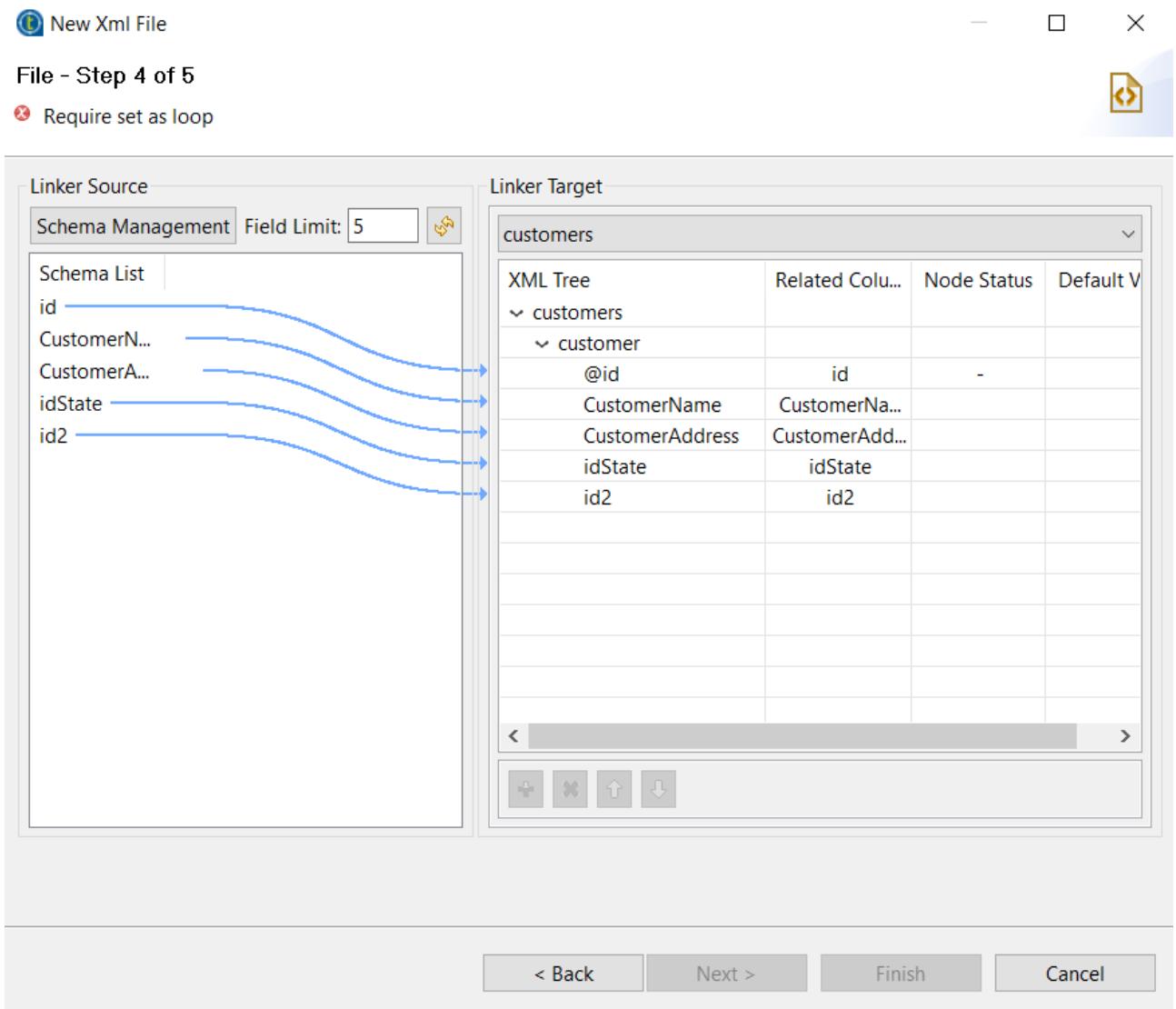


4. Enter the **Encoding** type in the corresponding field if the system does not detect it automatically.
5. In the **Limit** field, enter the number of columns on which the XPath query is to be executed, or enter 0 if you want it to be run against all of the columns.
6. In the **Output File** field, 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 of your output file

#### About this task

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.
Specify the maximum number of columns to be displayed in the schema list	In the <b>Field Limit</b> field of the <b>Linker Source</b> area, specify the maximum number of columns you want to display if the number of columns of the source file exceeds the limit defined in the Studio preference and any columns you want to pass to the output schema are not shown in the schema list. Then click the refresh button.
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. <p><b>Note:</b> It is a mandatory operation to define an element to run a loop on.</p>

To...	Perform the following...
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><b>Note:</b> 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.</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> <li>Select the element of interest, click the <b>[+]</b> 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 <b>[+]</b> 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 <b>x</b> 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><b>Note:</b> 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><b>Note:</b></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>

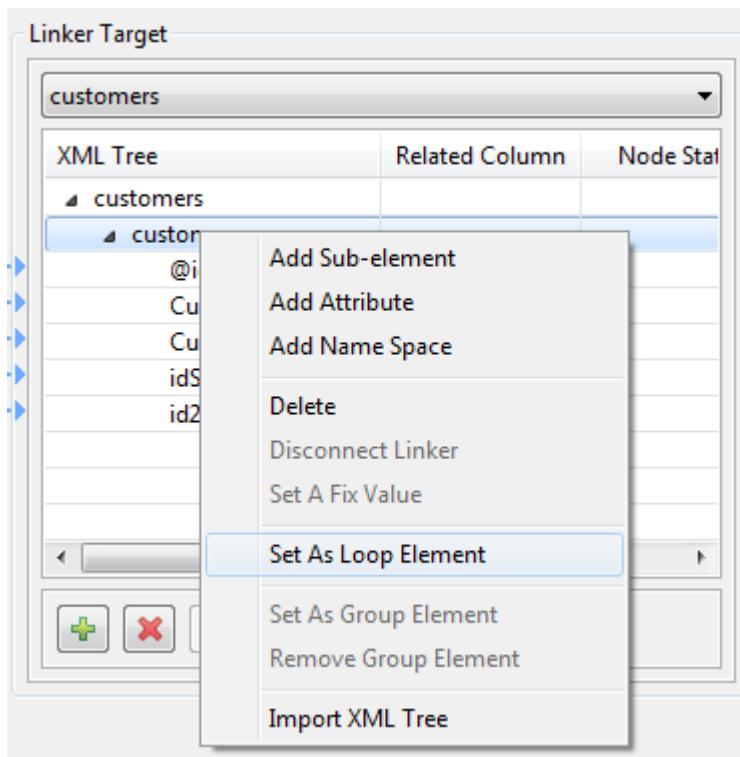
To...	Perform the following...
Create a source-target mapping	Select the column of interest in the <b>Linker Source</b> area, drop it onto the node of interest in the <b>Linker Target</b> area, and select <b>Create as sub-element of target node</b> , <b>Create as attribute of target node</b> , or <b>Add linker to target node</b> according to your need in the dialog box that appears, and click <b>OK</b> .  If you choose an option that is not permitted for the target node, you will see a warning message and your operation will fail.
Remove a source-target mapping	In the <b>Linker Target</b> area, right-click the node of interest and select <b>Disconnect Linker</b> from the contextual menu.
Create an XML tree from another XML or XSD file	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.

**Note:**

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.

**Procedure**

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.

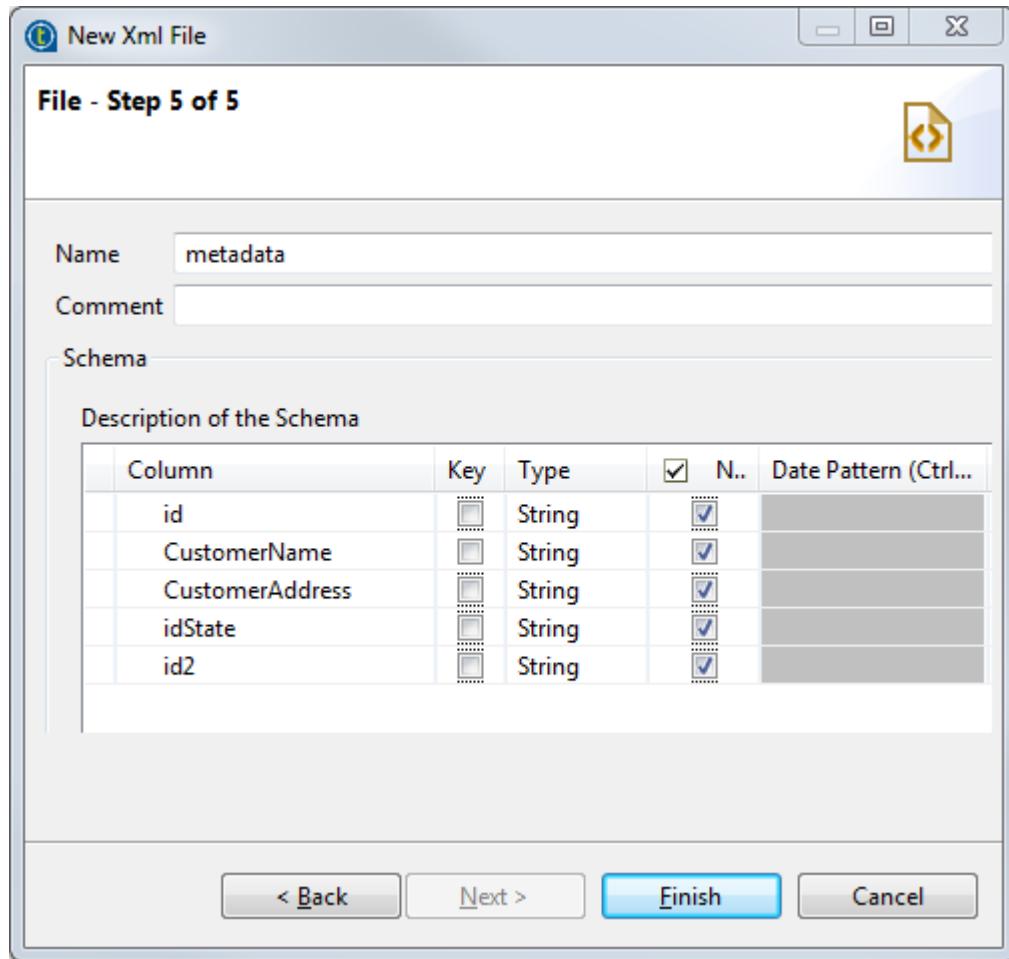


2. Define other output file properties as needed, and then click **Next** to view and customize the end schema.

## Finalizing the end schema of your output file

### About this task

Step 5 of the wizard displays the end schema generated and allows you to further define the schema.



### Procedure

- 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.

### Results

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.

## 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 of the File Excel connection](#) on page 244.
- Load the file of interest. See [Loading the file](#) on page 244.
- Parse the file to retrieve the file schema. See [Parsing the file](#) on page 246.
- Finalize the file schema. See [Finalizing the end schema of your Excel file](#) on page 247.

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 [Using centralized metadata in a Job](#) on page 344 and [Setting a repository schema in a Job](#) on page 41.

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.

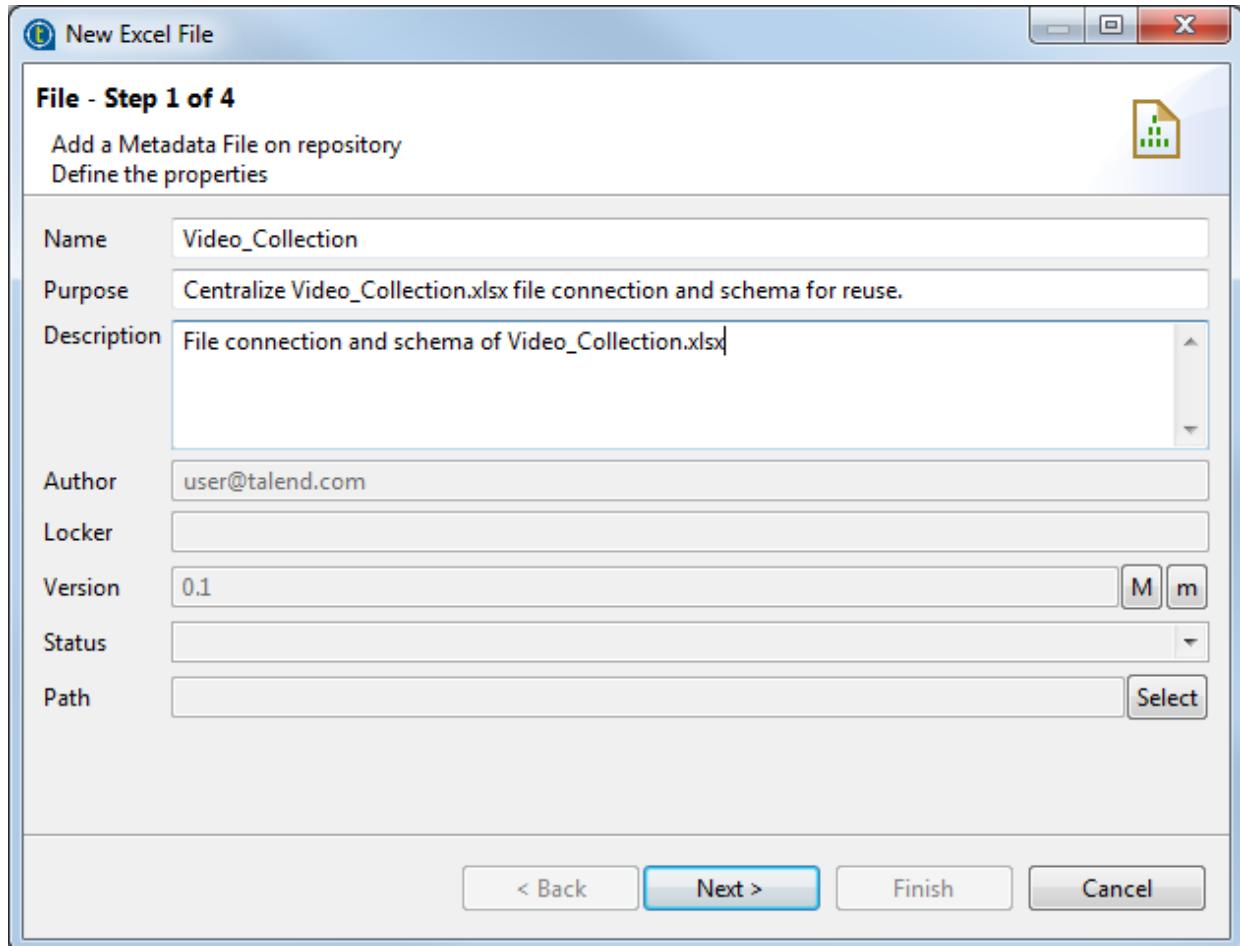
To add a new schema to an existing file connection, right-click the connection from the **Repository** tree view and select **Retrieve Schema** from the contextual menu.

To edit an existing file schema, right-click the schema from the **Repository** tree view and select **Edit Schema** from the contextual menu.

## Defining the general properties of the File Excel connection

### Procedure

- 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.



- 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](#) on page 382 and [Status management](#) on page 383 respectively.
- 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.
- Click **Next** to proceed with file settings.

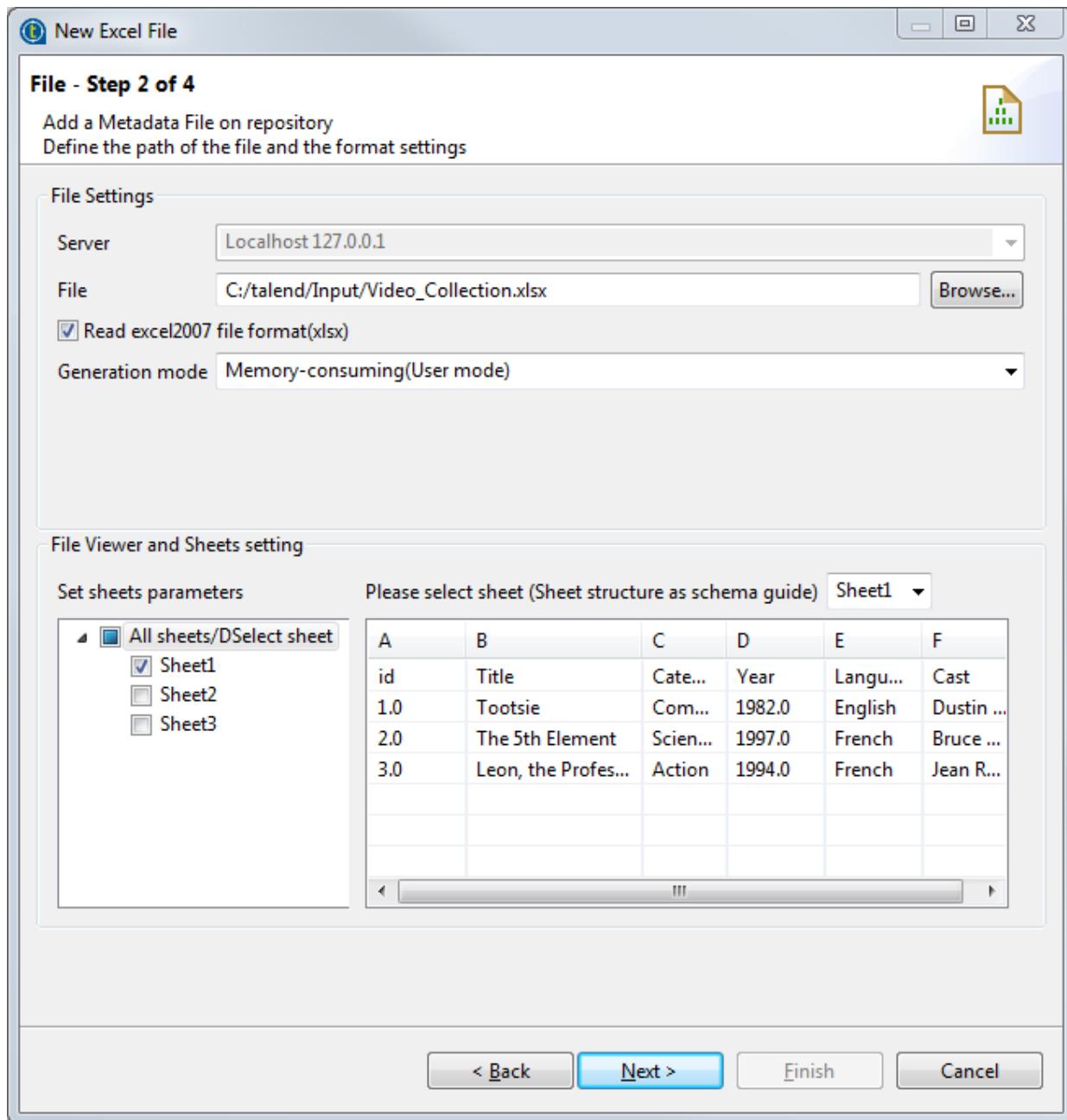
## Loading the file

### Procedure

- Specify the full path of the source file in the **File** field, or click the **Browse...** button to browse to the file.

**Note:** The Universal Naming Convention (UNC) path notation is not supported. If your source file is on a LAN host, you can first map the network folder into a local drive.

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.

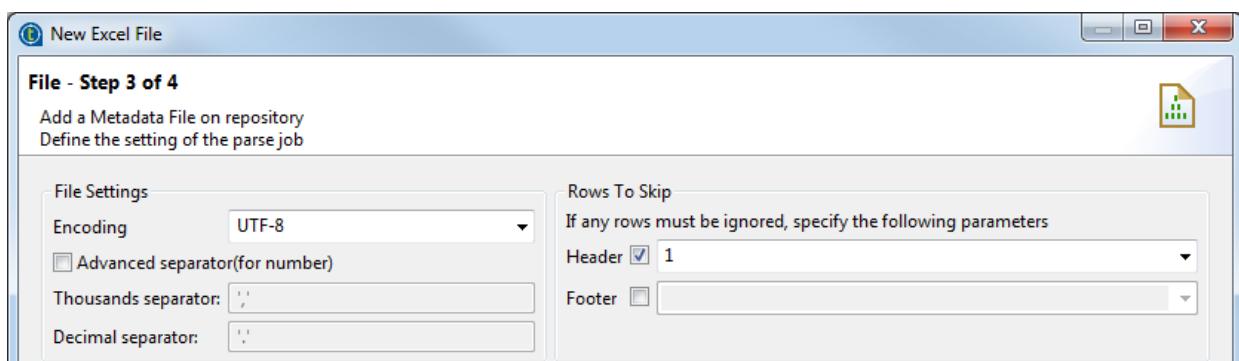
## Parsing the file

### About this task

In this step of the wizard, you can define the various settings of your file so that the file schema can be properly retrieved.

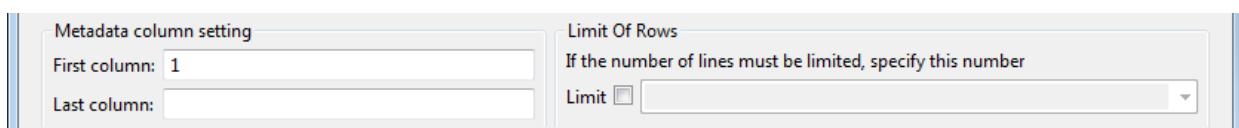
### Procedure

1. 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.



2. 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.



3. 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.

The screenshot shows the 'Preview' tab of a data integration wizard. At the top, there are two tabs: 'Preview' (which is selected) and 'Output'. Below the tabs, there is a checkbox labeled 'Set heading row as column names' which is checked, and a 'Refresh Preview' button. The main area displays a table with the following data:

id	Title	Category	Year	Language	Cast
1	Tootsie	Comedy	1982	English	Dustin Hoffman, Jessica Lange, Sydney
2	The 5th Element	Science fiction	1997	French	Bruce Willis, Gary Oldman, Milla Jovovich
3	Leon, the Professor	Action drama	1994	French	Jean Reno, Gary Oldman, Nathalie Portman

Below the table, there are two buttons: 'Export as context' and 'Revert Context'. At the bottom of the preview window, there are four buttons: '< Back', 'Next >', 'Finish', and 'Cancel'.

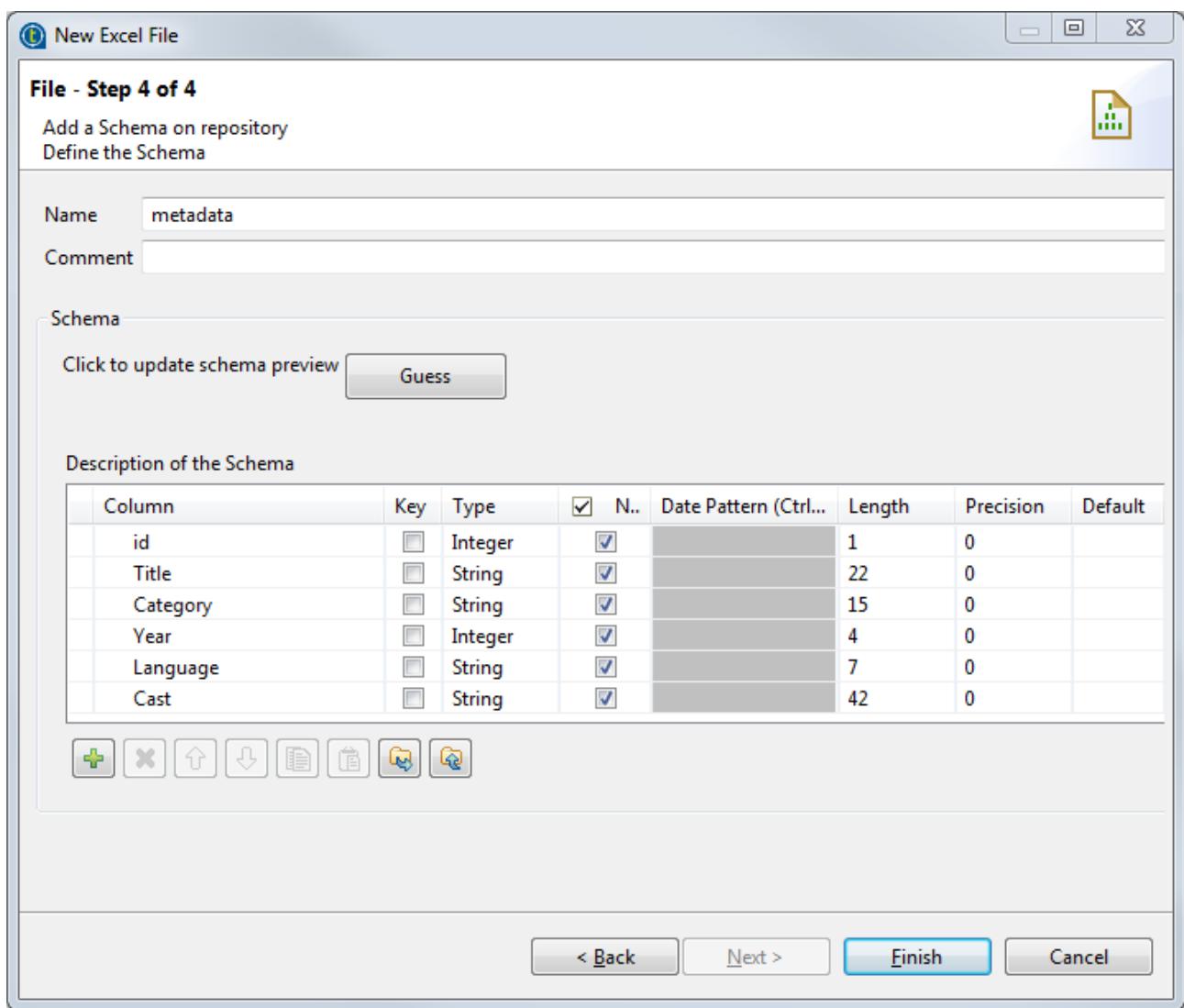
4. Then click **Next** to continue.

## Finalizing the end schema of your Excel file

### About this task

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.



## Procedure

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.

## Centralizing File LDIF metadata

### About this task

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.

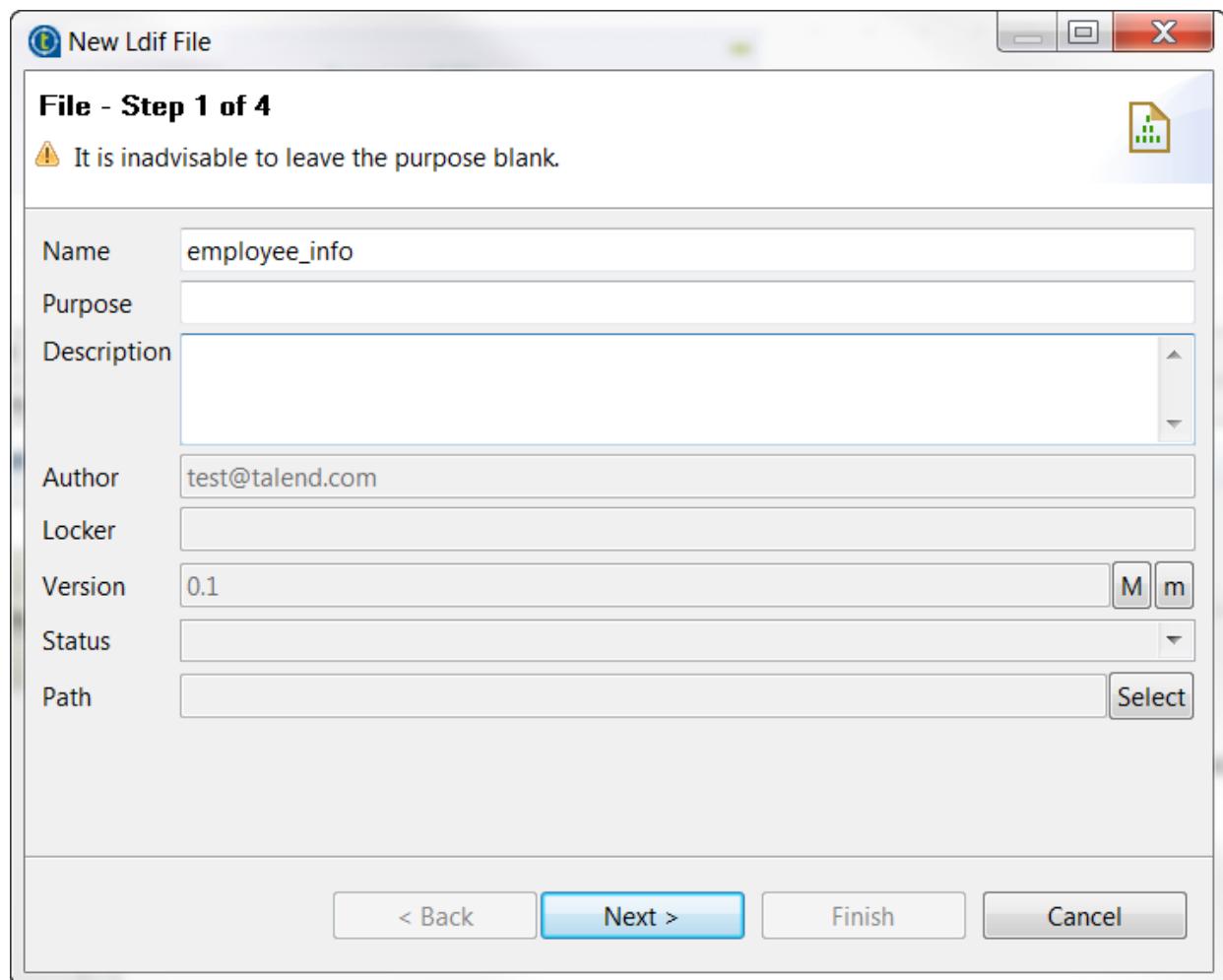
To centralize a file connection and its schema you have already defined in a Job, click the  icon in the **Basic settings** view of the relevant component, with its **Property Type** set to **Built-in**, to open the file metadata setup wizard.

Then complete these steps following the wizard:

### Procedure

1. Fill in the general information in the relevant fields to identify the LDIF file metadata, including **Name**, **Purpose** and **Description**.

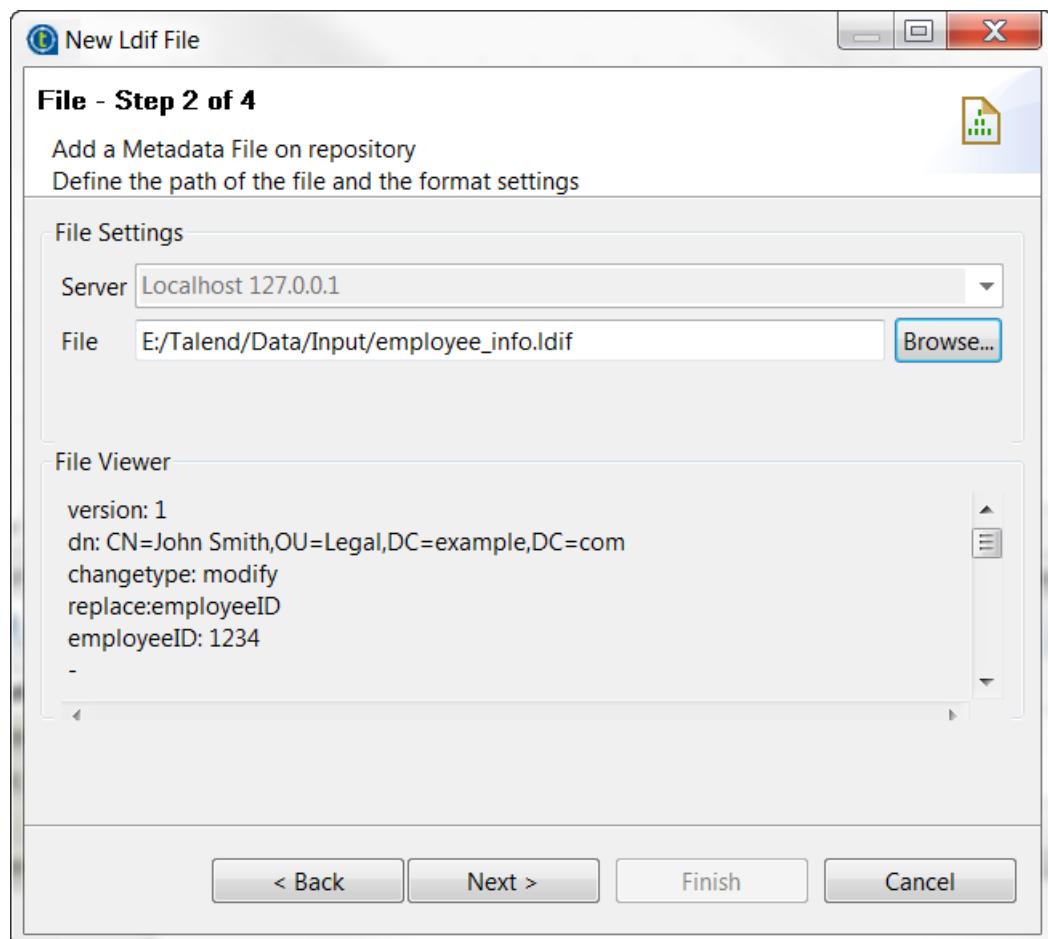
The **Name** field is required, and the information you provide in the **Description** field will appear as a tooltip when you move your mouse pointer over the file connection.



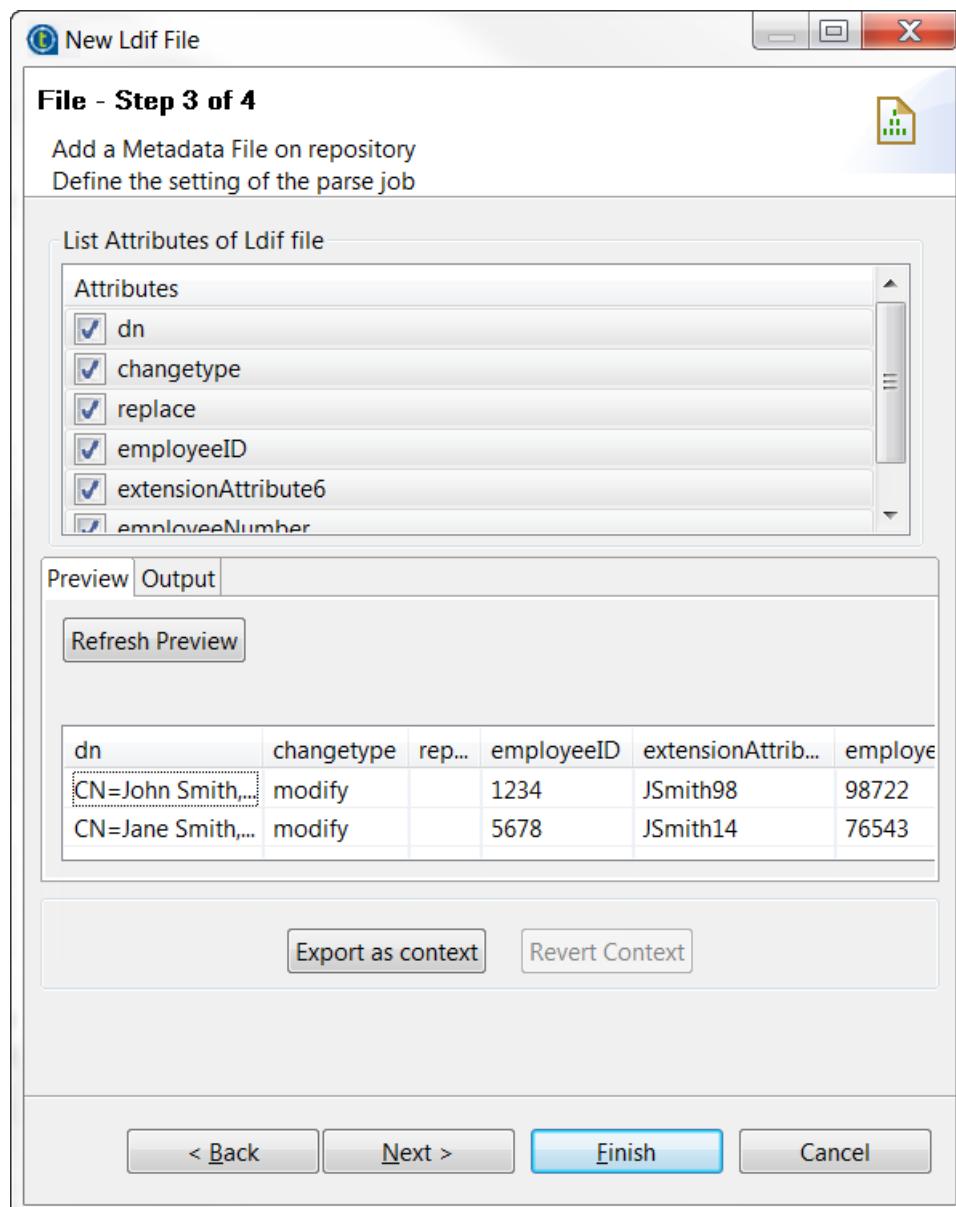
2. If needed, set the version and status in the **Version** and **Status** fields respectively. You can also manage the version and status of a repository item in the **Project Settings** dialog box. For more information, see [Version management](#) on page 382 and [Status management](#) on page 383 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. Specify the full path of the source file in the **File** field, or click the **Browse...** button to browse to the file.

**Note:** The Universal Naming Convention (UNC) path notation is not supported. If your source file is on a LAN host, you can first map the network folder into a local drive.

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.

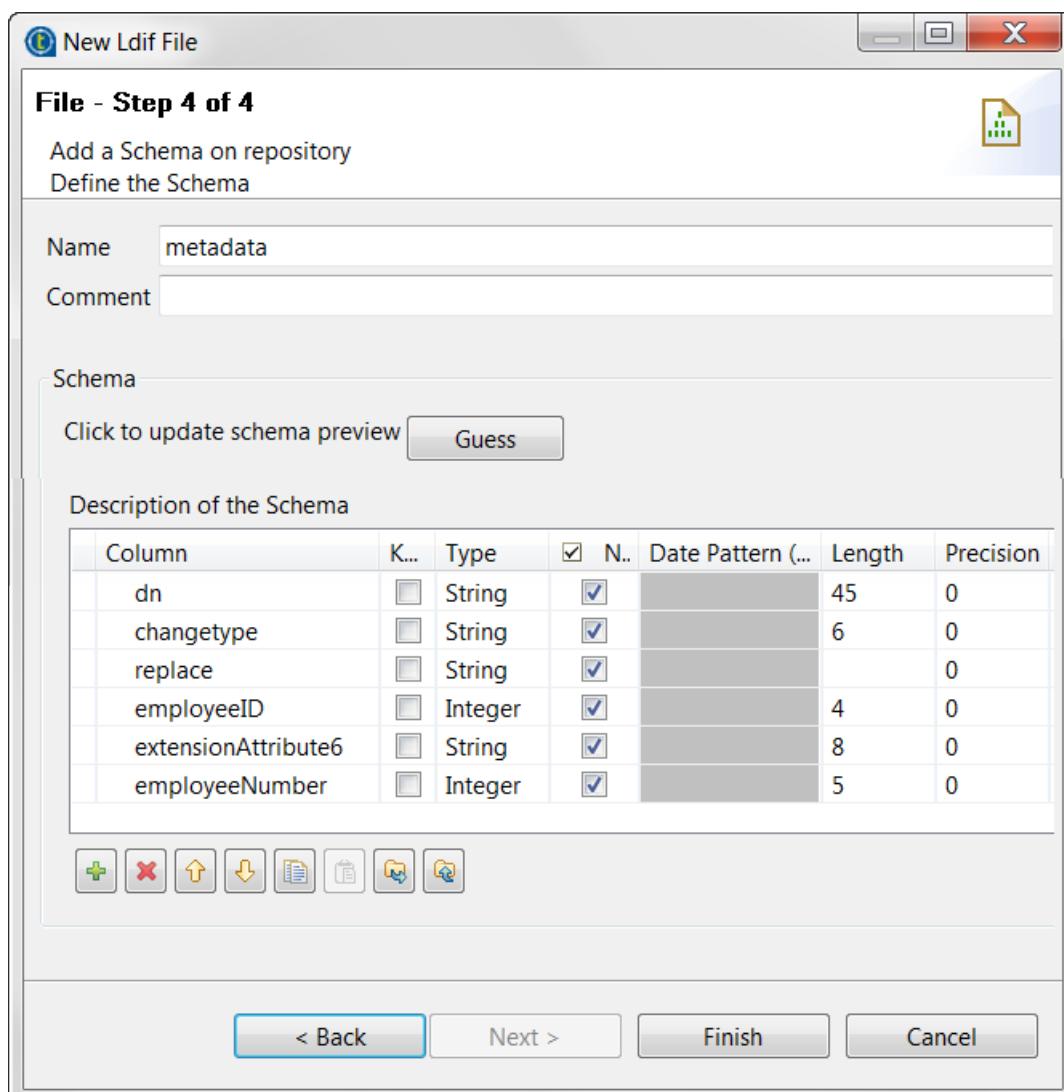


5. Check the first 50 rows of the file in the **File Viewer** area and click **Next** to continue.
6. From the list of attributes of the loaded file, select the attributes you want to include the file schema, and click **Refresh Preview** to preview the selected attributes.  
Then click **Next** to proceed 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.

## Results

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 [Using centralized metadata in a Job](#) on page 344 and [Setting a repository schema in a Job](#) on page 41.

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.

## 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](#) on page 254.

For information about setting up output JSON metadata, see [Setting up JSON metadata for an output file](#) on page 262.

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.

## 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.

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 [Using centralized metadata in a Job](#) on page 344 and [Setting a repository schema in a Job](#) on page 41.

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.

### Defining the general properties of the File JSON connection

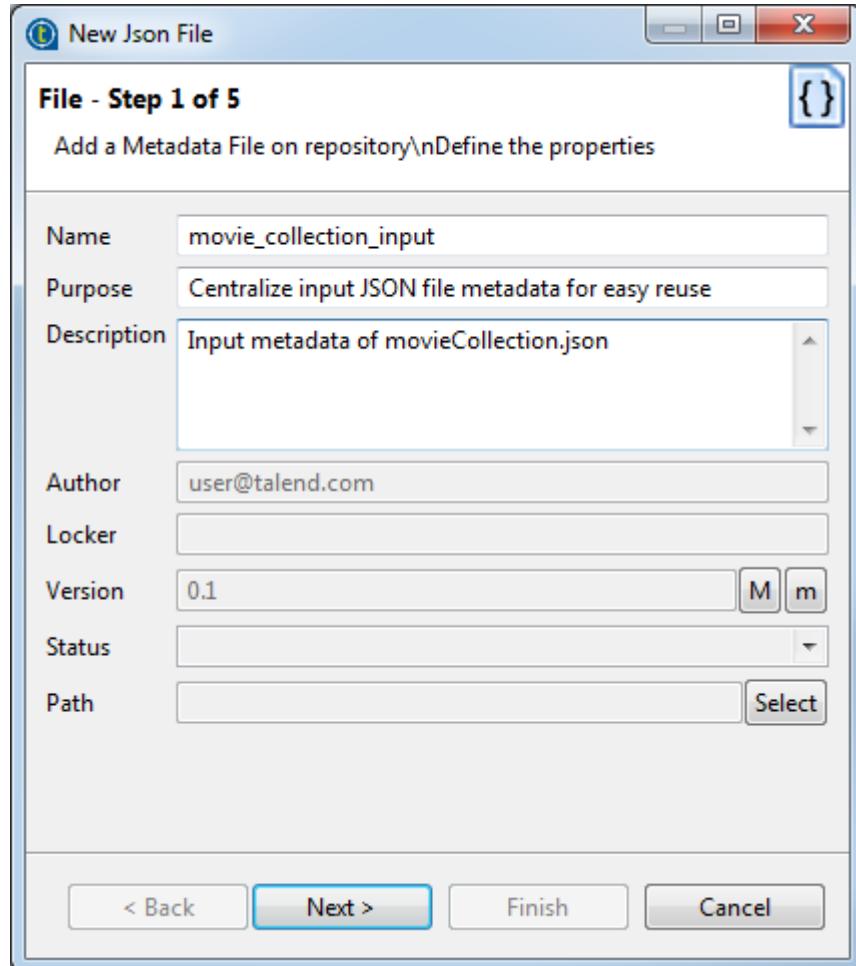
#### Procedure

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.

**Note:**

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.

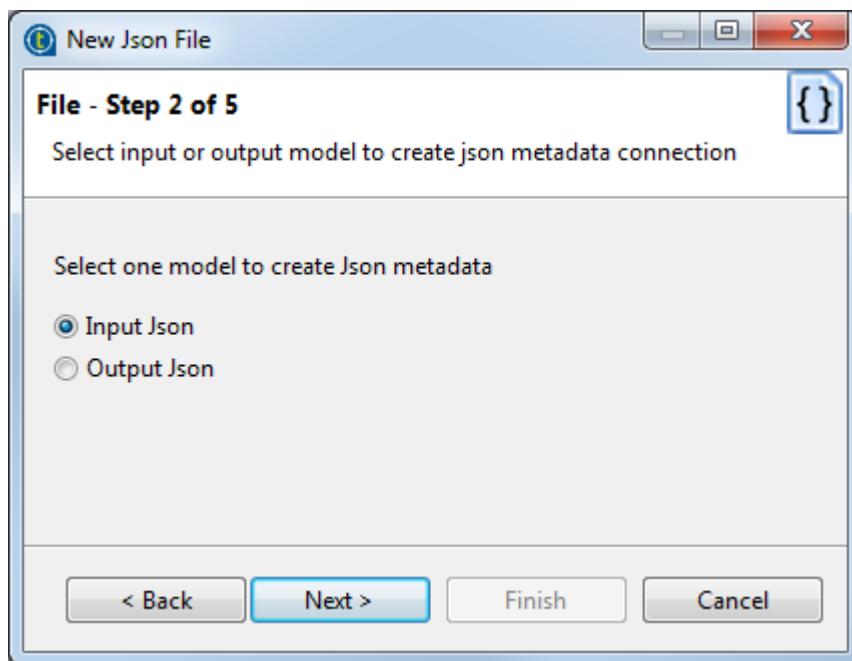


2. If needed, set the version and status in the **Version** and **Status** fields respectively. You can also manage the version and status of a repository item in the **Project Settings** dialog box. For more information, see [Version management](#) on page 382 and [Status management](#) on page 383 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

##### Procedure

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.



- 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.

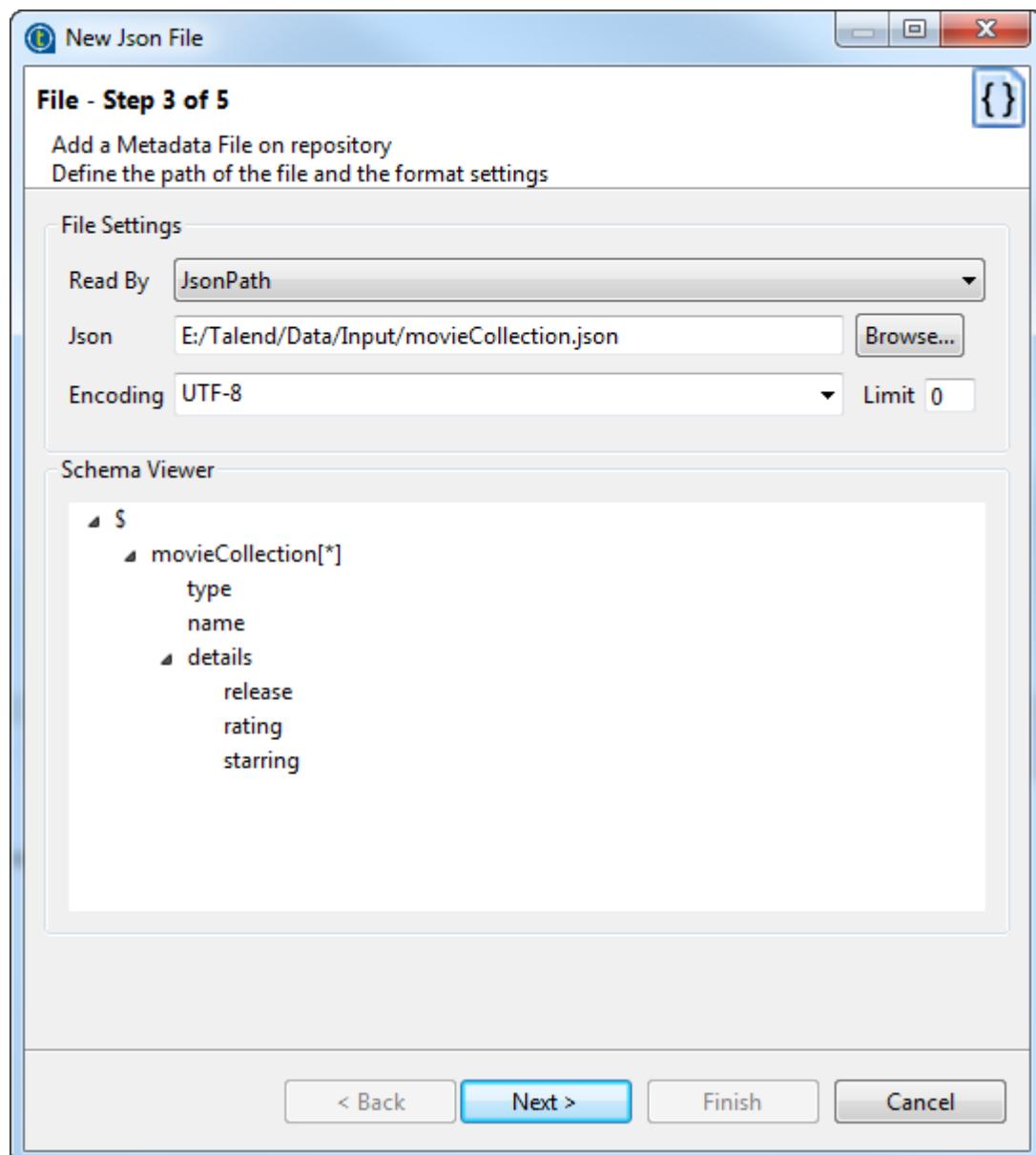
- Xpath:** read the JSON data based on an XPath query.

- 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.

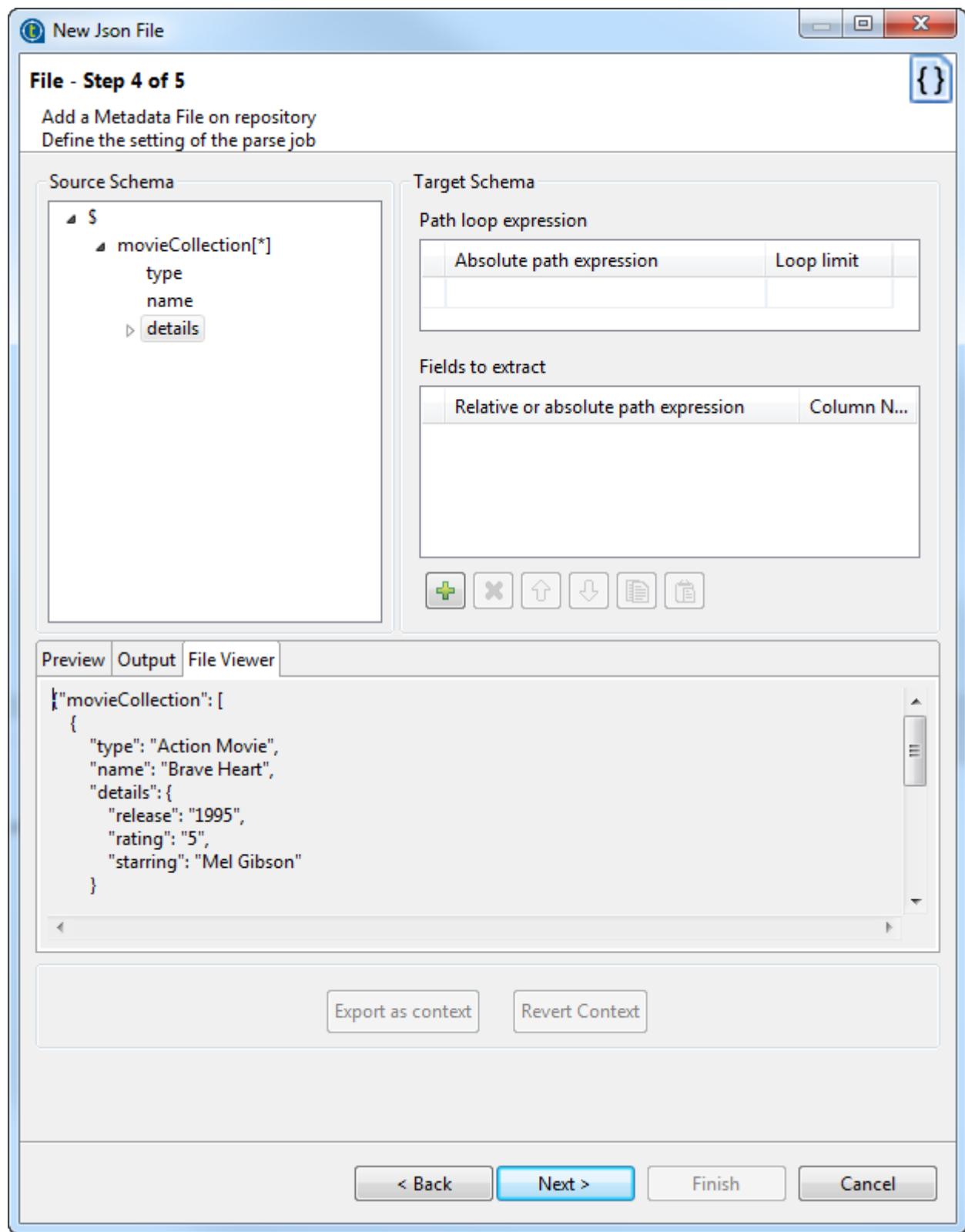


4. Enter the **Encoding** type in the corresponding field if the system does not detect it automatically.
5. In the **Limit** field, enter the number of levels in the JSON hierarchical depth to which you want to limit the JsonPath or XPath query, 0 for no limits.  
Setting this parameter to a value less 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 of your JSON file

##### About this task

In this step, you will set the schema parameters.



The schema definition window is composed of four views:

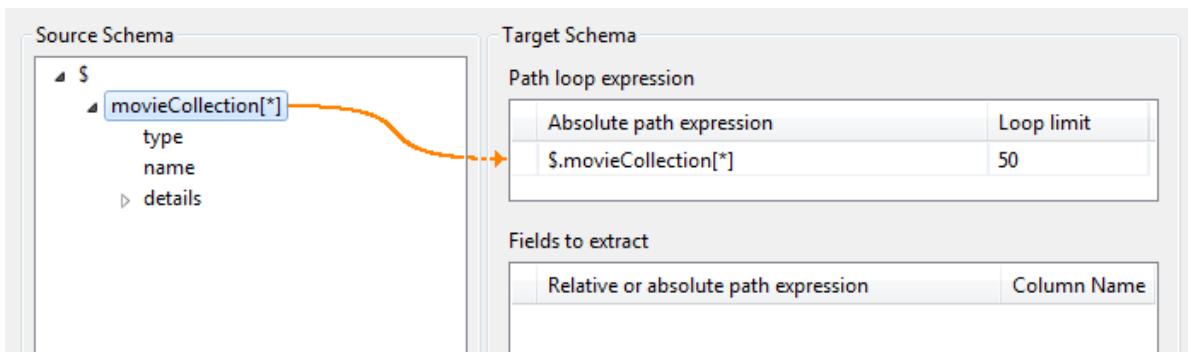
View	Description
Source Schema	Tree view of the JSON file.
Target Schema	Extraction and iteration information.

View	Description
Preview	Preview of the target schema, together with the input data of the selected columns displayed in the defined order.
File Viewer	Preview of the JSON file's data.

## Procedure

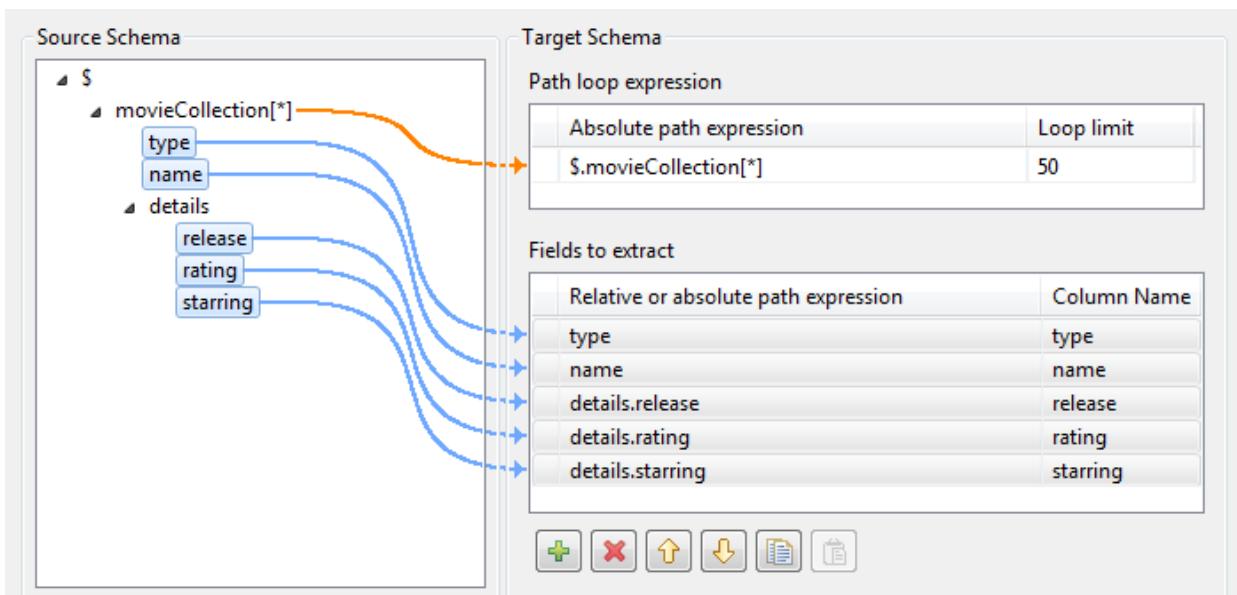
1. Populate the **Path loop expression** field with the absolute JsonPath or XPath expression, depending on the type of query you have selected, for the node to be iterated upon. There are two ways to do this, either:
  - enter the absolute JsonPath or XPath expression for the node to be iterated upon (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.



**Note:** 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.



**Note:** 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.

The screenshot shows a user interface for previewing a schema. At the top, there are three tabs: 'Preview' (which is selected), 'Output', and 'File Viewer'. Below the tabs is a button labeled 'Refresh Preview'. Underneath the button, the text 'Preview successful...' is displayed in blue. A table is shown below this text, representing the schema and its data. The table has columns labeled 'type', 'name', 'release', 'rating', and 'starring'. The first row contains 'Action Movie', 'Brave Heart', '1995', '5', and 'Mel Gibson'. The second row contains 'Action Movie', 'Edge of Darkness', '2010', '5', and 'Mel Gibson'.

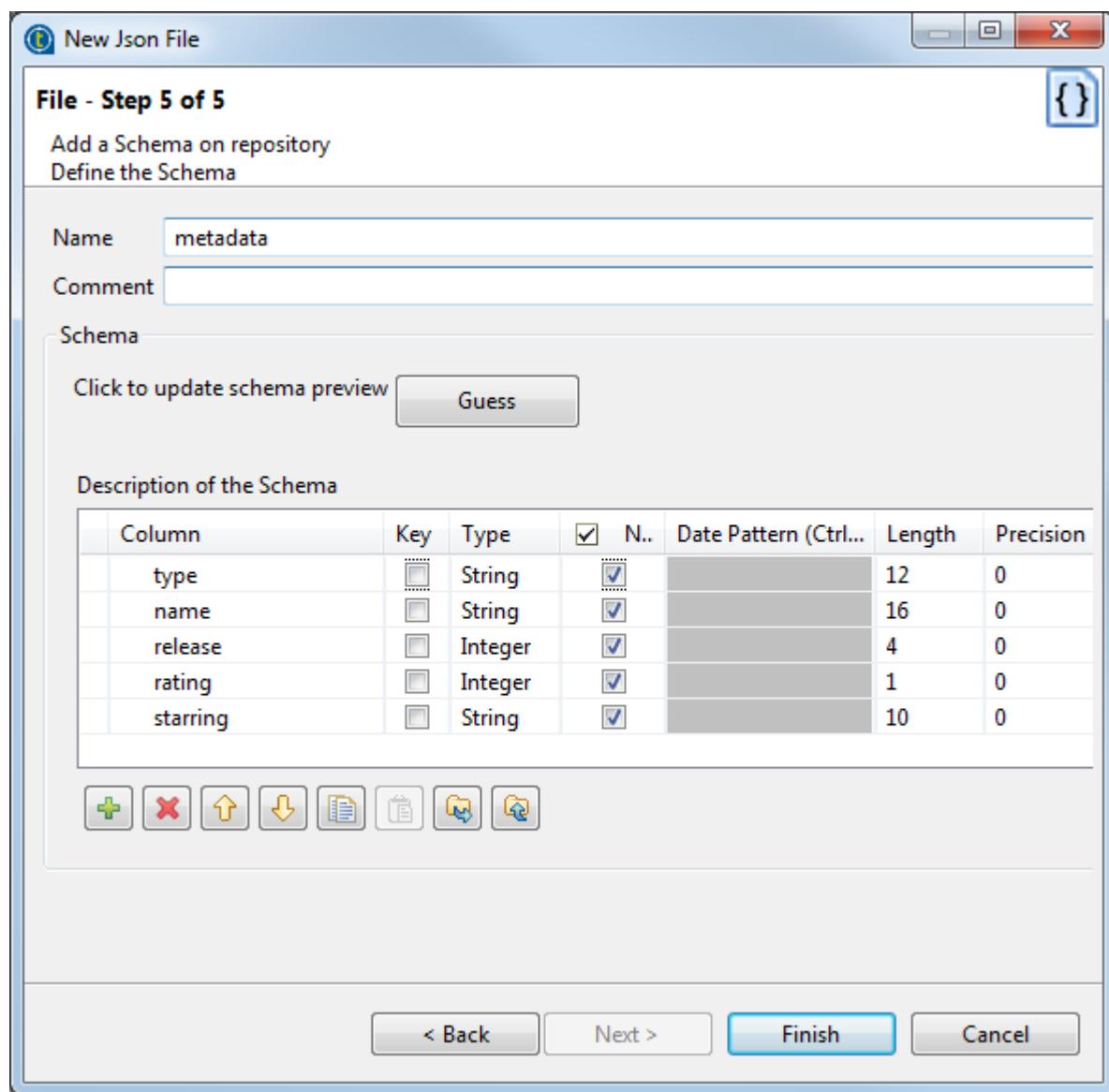
type	name	release	rating	starring
Action Movie	Brave Heart	1995	5	Mel Gibson
Action Movie	Edge of Darkness	2010	5	Mel Gibson

7. Click **Next** to finalize the schema.

### Finalizing the schema of your JSON file

#### About this task

The last step of the wizard shows the end schema generated and allows you to customize the schema according to your needs.



## Procedure

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.

## Setting up JSON metadata for an output file

This section describes how to define JSON metadata for an output file.

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 [Using centralized metadata in a Job](#) on page 344 and [Setting a repository schema in a Job](#) on page 41.

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.

### Defining general properties of the File JSON connection for an output file

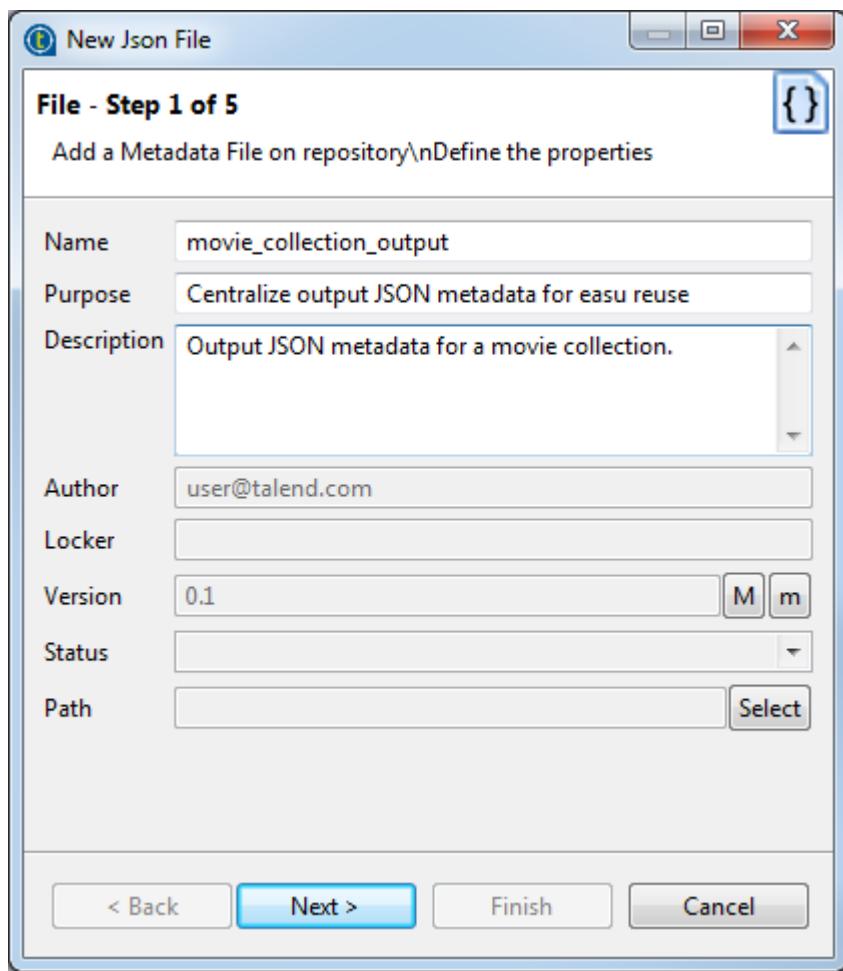
#### Procedure

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.

#### Note:

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](#) on page 382 and [Status management](#) on page 383 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.

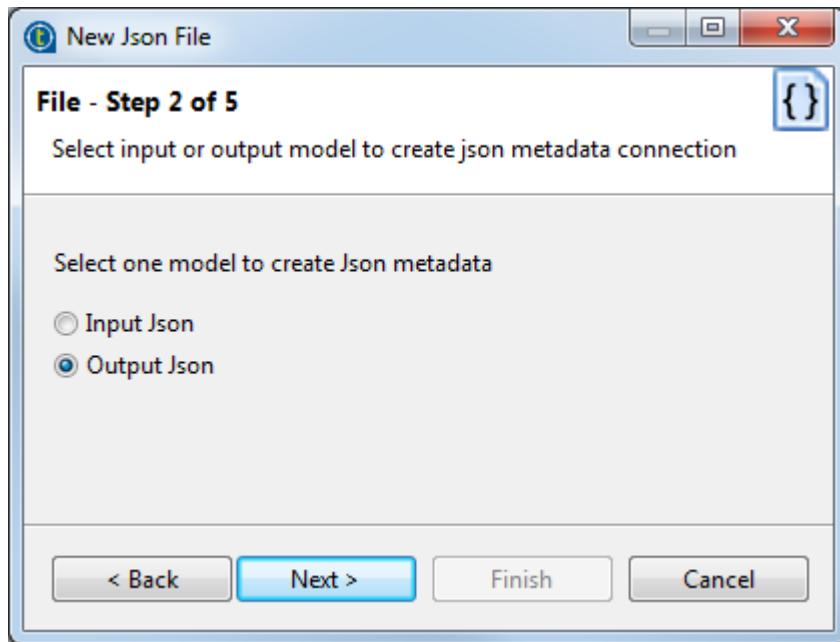
#### Setting the type of metadata and loading the template JSON file

##### About this task

In this step, the type of schema is set as either input or output. For this procedure, the schema of interest is output.

##### Procedure

1. From the dialog box, select **Output JSON** click **Next** to proceed to the next step of the wizard.



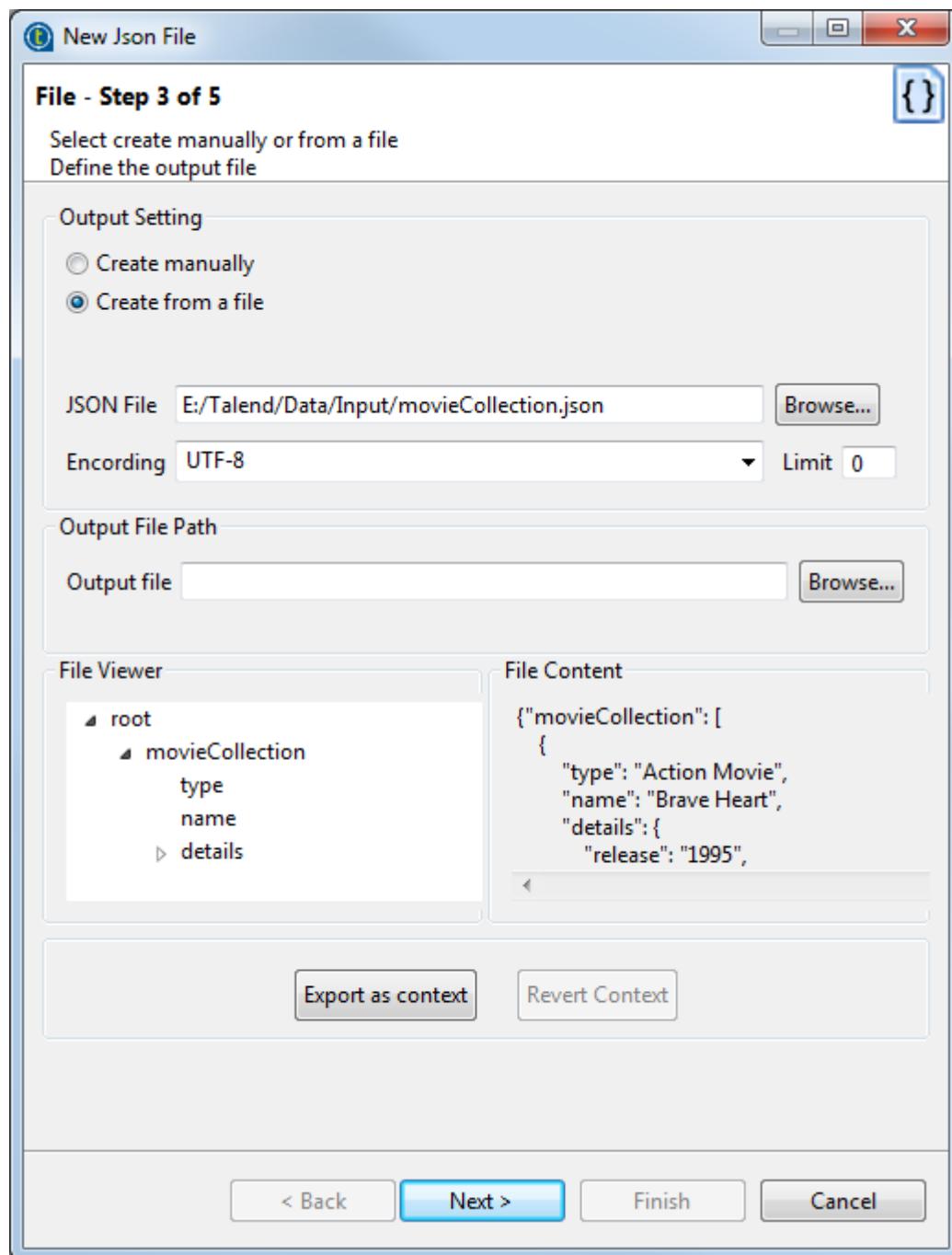
2. Choose whether to create the output metadata manually or from an existing JSON file as a template.

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.

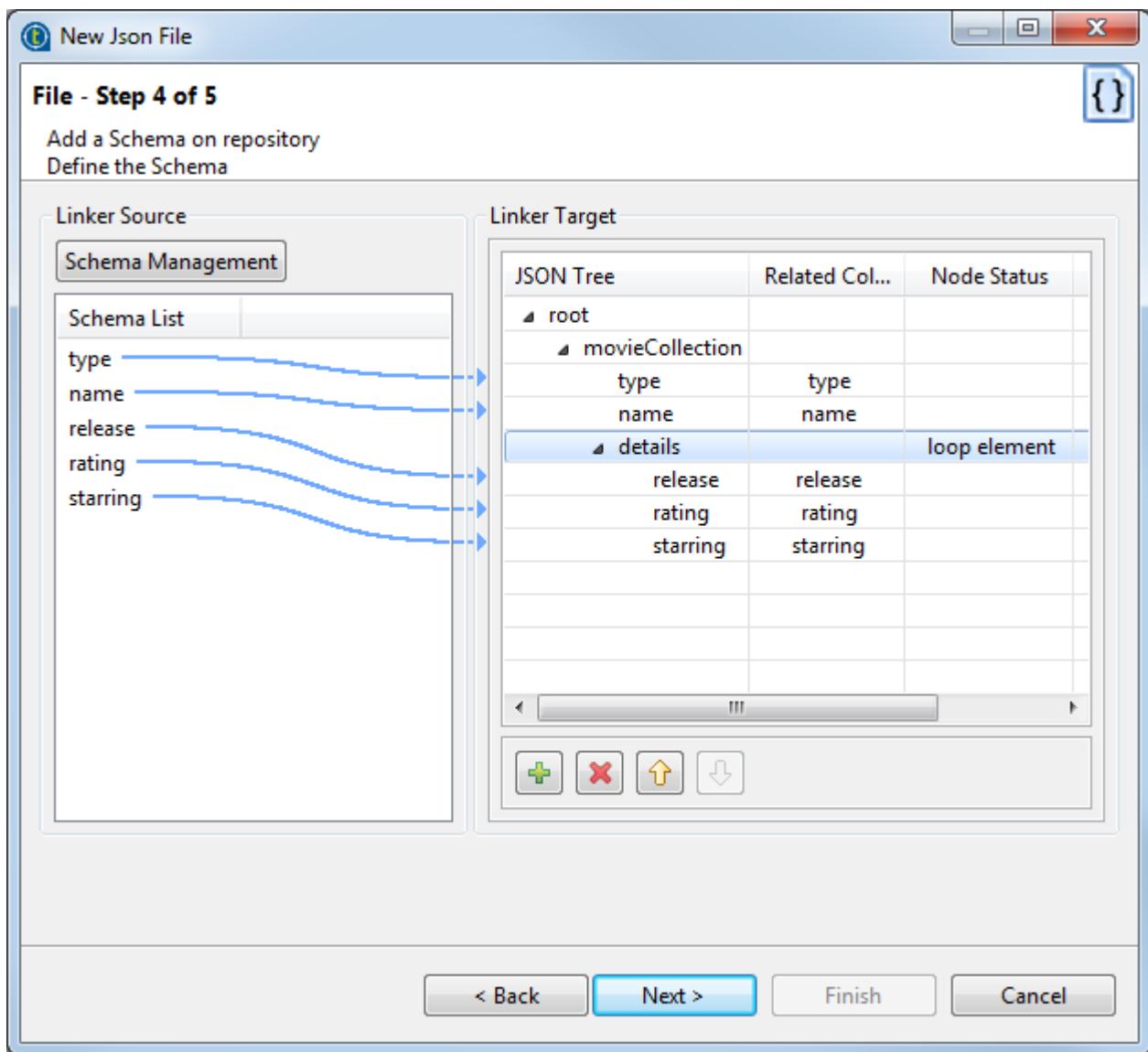


4. Enter the **Encoding** type in the corresponding field if the system does not detect it automatically.
5. In the **Limit** field, enter the number of levels in the JSON hierarchical depth to which you want to limit the JsonPath or XPath query, 0 for no limits.  
Setting this parameter to a value less 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 JSON schema of your output file

##### About this task

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> <div style="border: 1px solid #ccc; padding: 5px; margin-top: 10px;"> <b>Note:</b>            It is a mandatory operation to define an element to run a loop on.         </div>
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> <div style="border: 1px solid #ccc; padding: 5px; margin-top: 10px;"> <b>Note:</b>            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.         </div>

To...	Perform the following...
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> <li>Select the element of interest, click the <b>[+]</b> 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 <b>[+]</b> 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 <b>x</b> 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> <div style="border: 1px solid #ccc; padding: 5px; margin-top: 10px;"> <p><b>Note:</b></p> <p>Deleting an element will also delete its children, if any.</p> </div>
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> <div style="border: 1px solid #ccc; padding: 5px; margin-top: 10px;"> <p><b>Note:</b></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> </div>
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>

To...	Perform the following...
Remove a source-target mapping	In the <b>Linker Target</b> area, right-click the node of interest and select <b>Disconnect Linker</b> from the contextual menu.
Create a JSON tree from another JSON file	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.

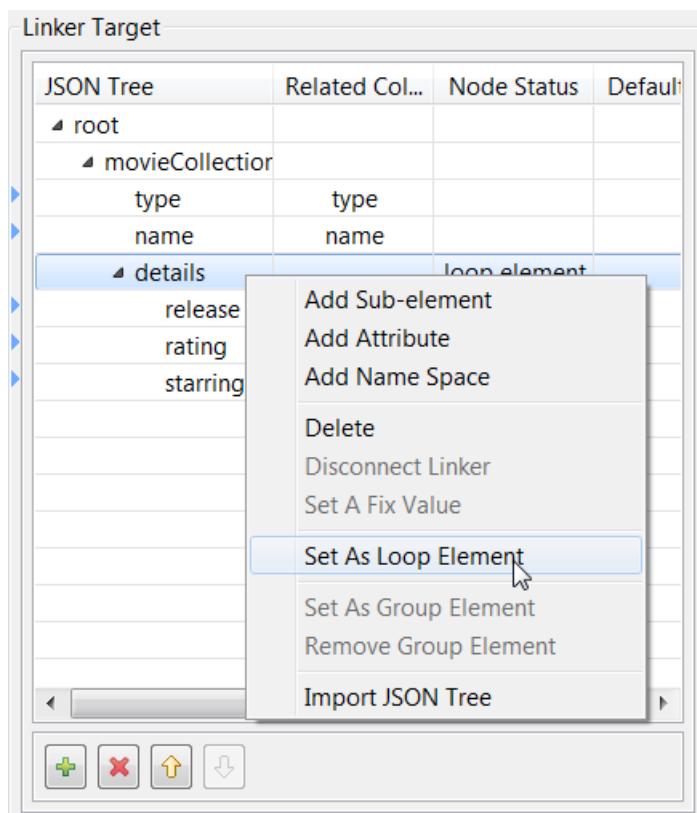
**Note:**

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.

**Procedure**

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.

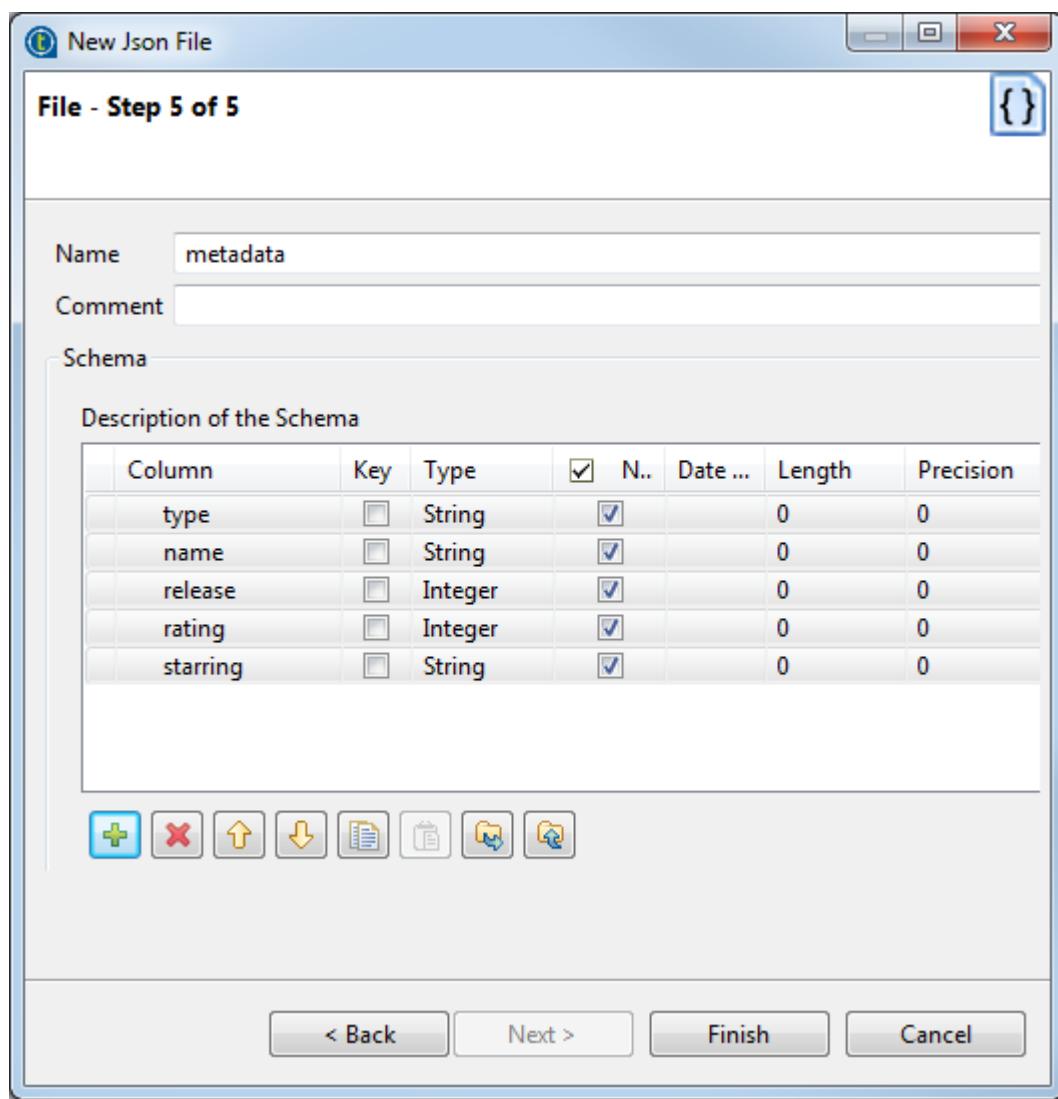
In this example, define a loop to run on the **details** element.



2. Customize the mappings if needed.
3. Click **Next** to finalize the schema.

**Finalizing the end schema JSON of your output file****About this task**

The last step of the wizard shows the end schema generated and allows you to customize the schema according to your needs.



## Procedure

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.

## 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.

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.

## Defining the general properties of the LDAP connection

### Procedure

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](#) on page 382 and [Status management](#) on page 383 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

### Procedure

- 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**

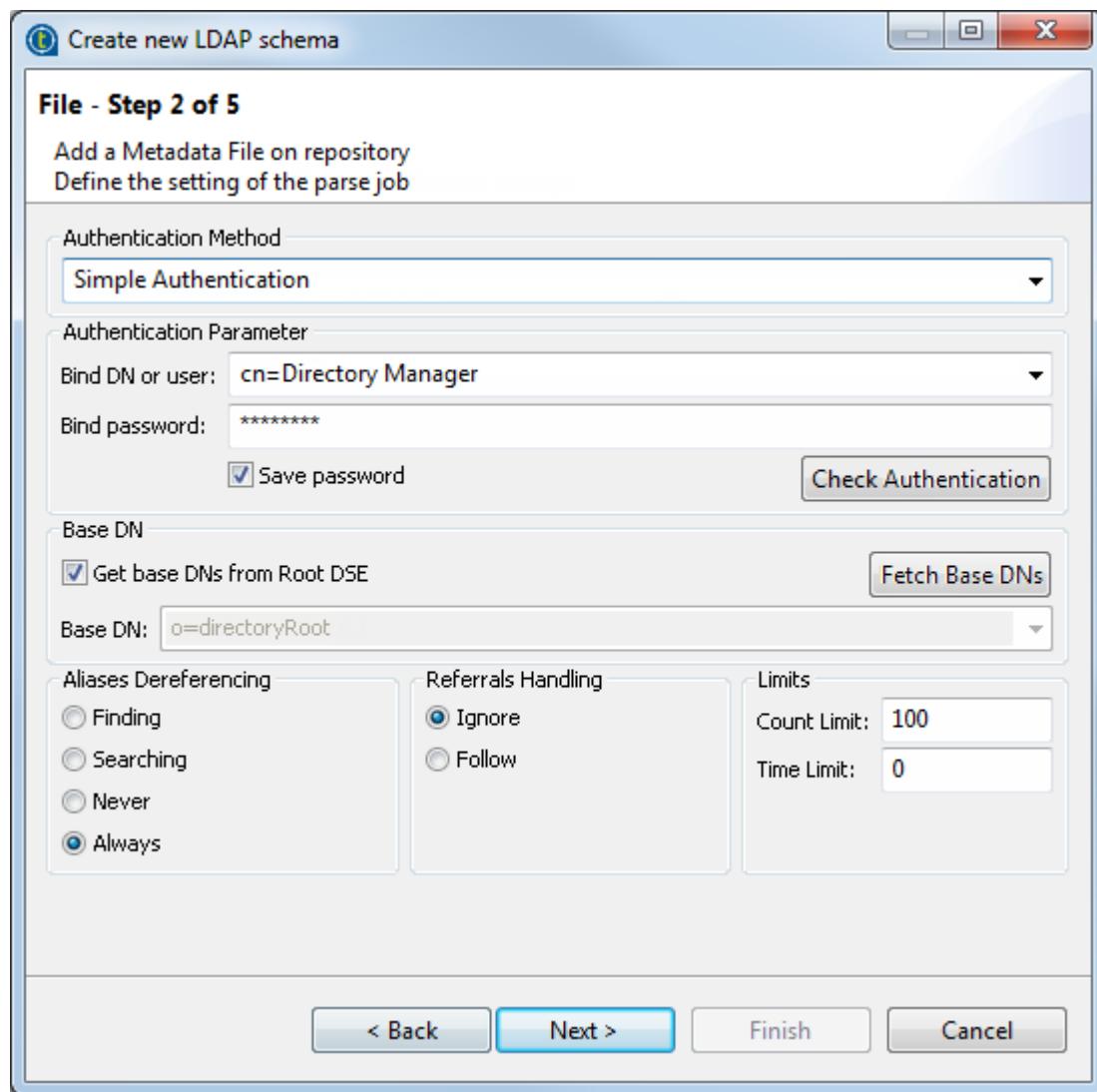
Field	Description
Host	LDAP Server host name or IP address
Port	Listening port to the LDAP directory
Encryption method	<b>LDAP</b> : no encryption is used <b>LDAPS</b> : secured LDAP <b>TLS</b> : certificate is used

- Then check your connection using **Check Network Parameter** to verify the connection and activate the **Next** button.
- Click **Next** to continue.

## Configuring LDAP access parameters

### Procedure

- 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

Field	Description
Referral Handling	Redirection of user request: <b>Ignore</b> : does not handle request redirections <b>Follow</b> : does handle request redirections
Limit	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.

## Defining the schema of your LDAP directory

### Procedure

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 of 'Attributes' with checkboxes: 'description', 'userpassword', 'uid' (which is checked), 'sn', and 'cn'. To the right of the attributes 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'.

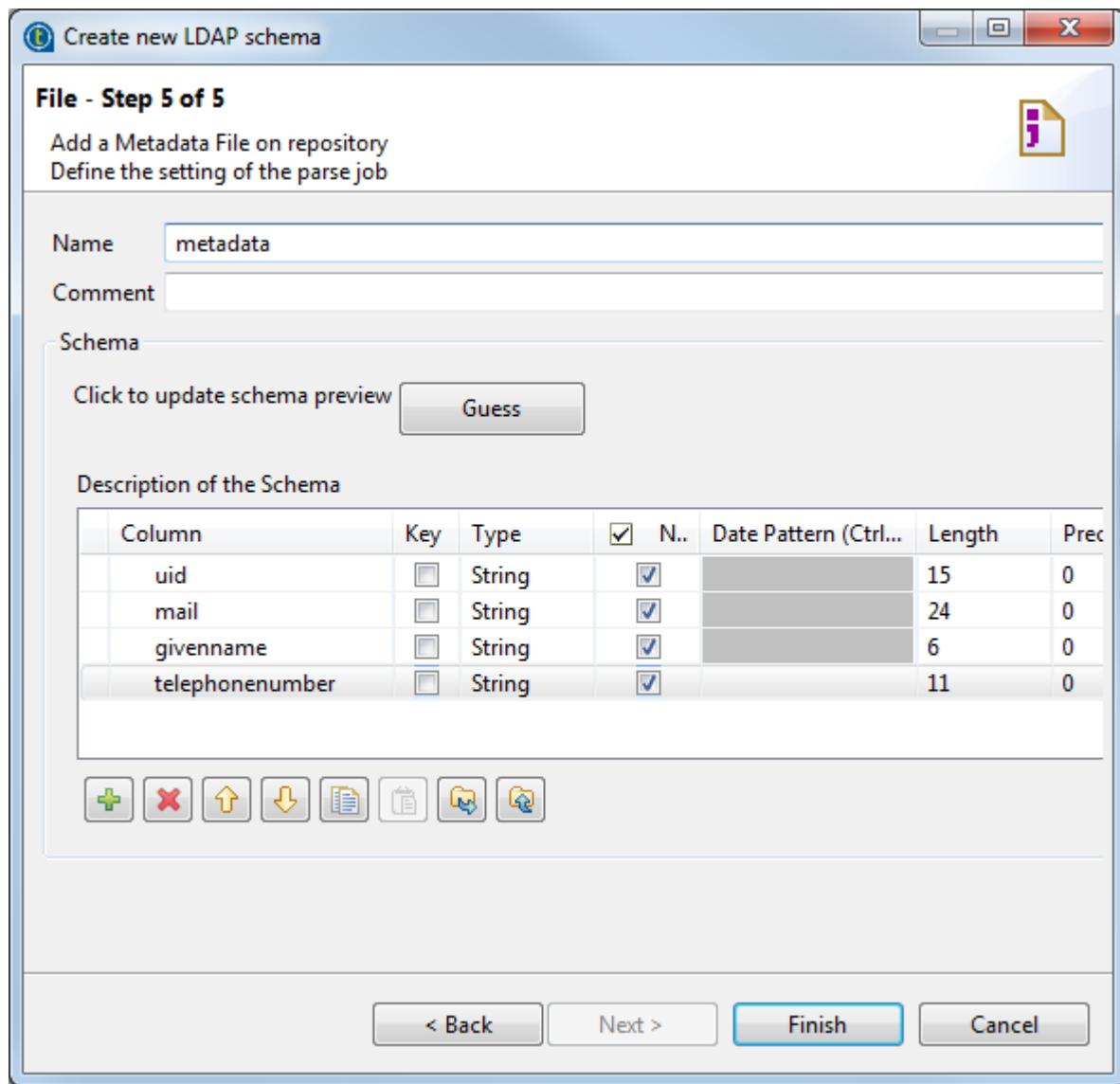
uid	mail	givenname	telephonenumber
PIERRE DUPONT	Pierre.Dupont@talend.com	PIERRE	00149684750
PIERRE DUPON...	mhirt78@talend.com	PIERRE	
mhirt			
greg			

2. Click **Refresh Preview** to display the selected column and a sample of the data.
3. Click **Next** to continue.

## Finalizing the end schema of your LDAP directory

### About this task

The last step shows the LDAP schema generated and allows you to further customize the end schema.



### Procedure

- 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.

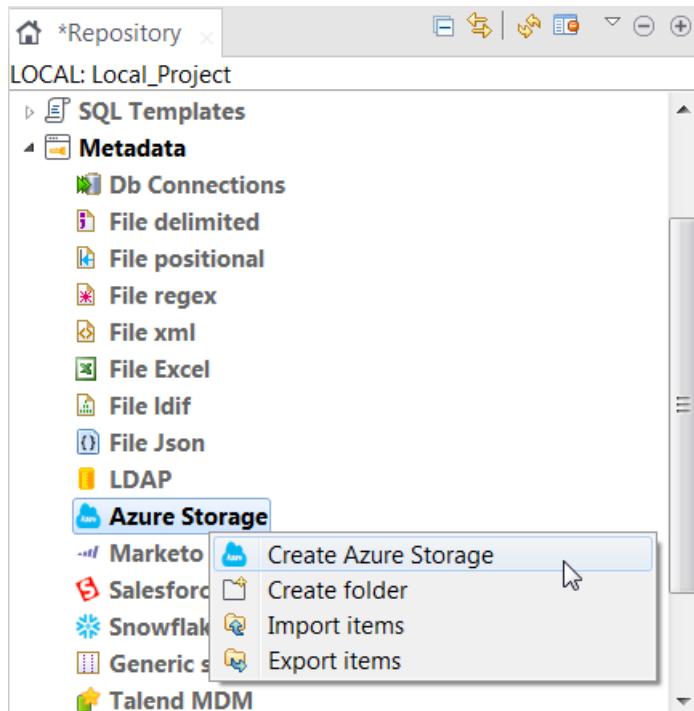
## Centralizing Azure Storage metadata

### About this task

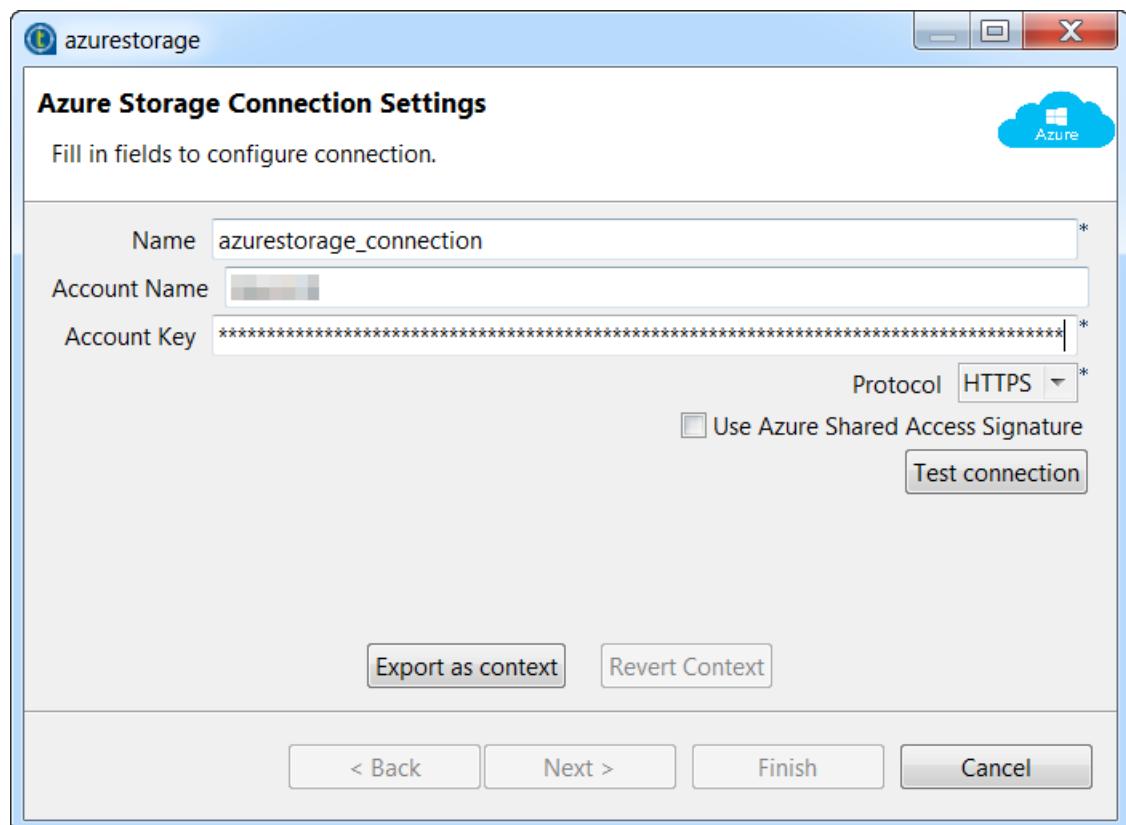
You can use the Azure Storage metadata wizard provided by Talend Studio to set up quickly a connection to Azure Storage and retrieve the schema of your interested container(s), queue(s), and table(s).

### Procedure

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.

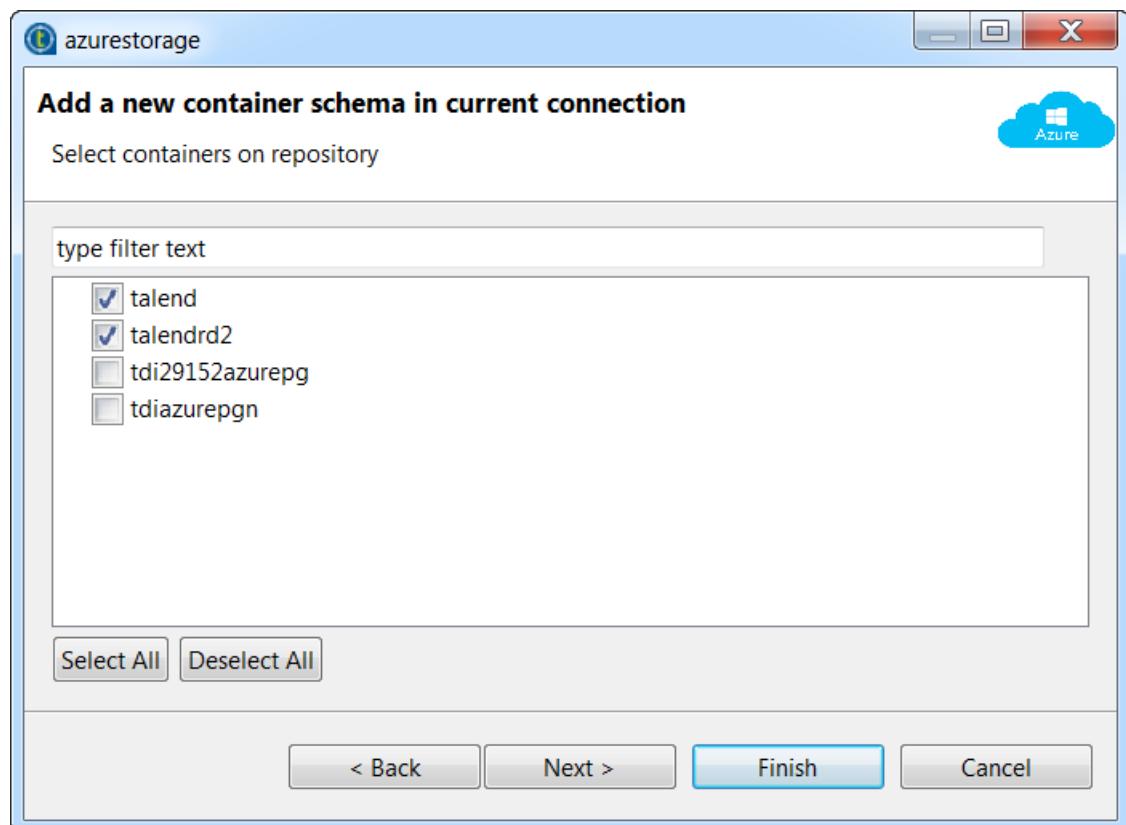


Property	Description
<b>Name</b>	Enter the name for the connection to be created.
<b>Account Name</b>	Enter the name of the storage account you need to access. A storage account name can be found in the Manage Access Keys dashboard of the Microsoft Azure Storage system to be used.
<b>Account Key</b>	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> .

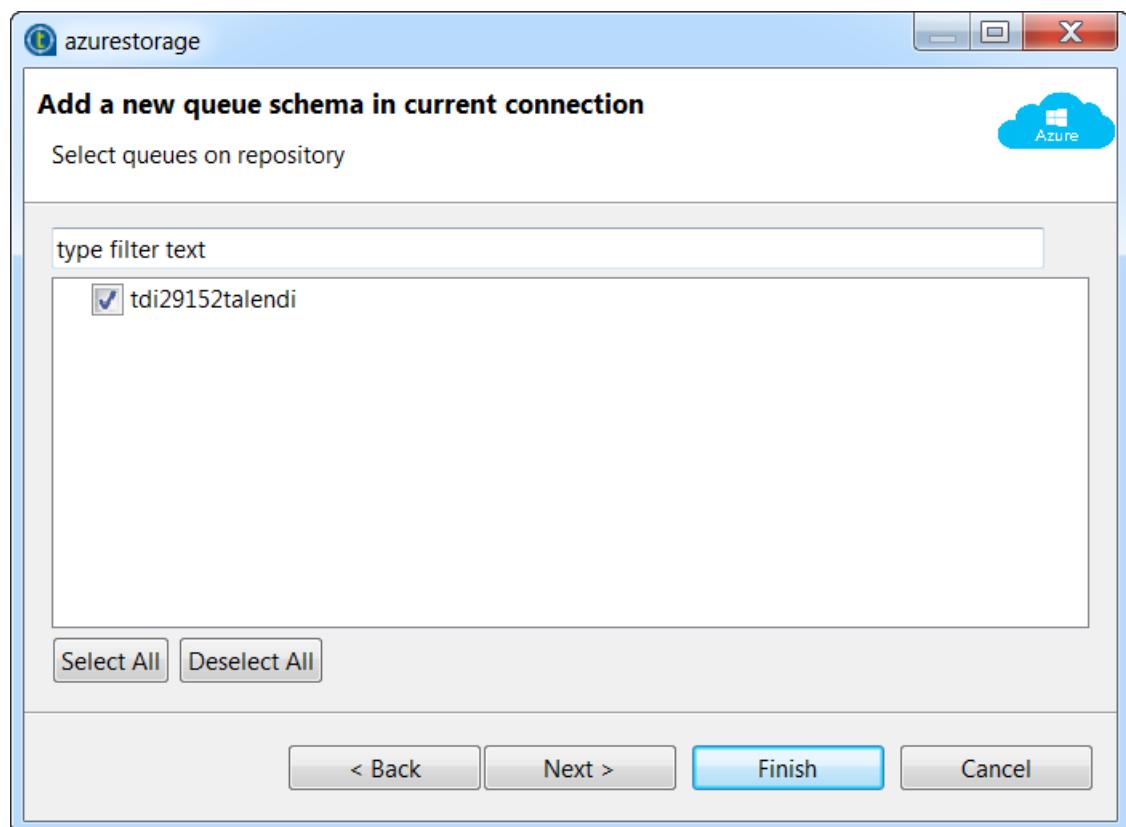
3. Click **Test connection** to verify the configuration.

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.

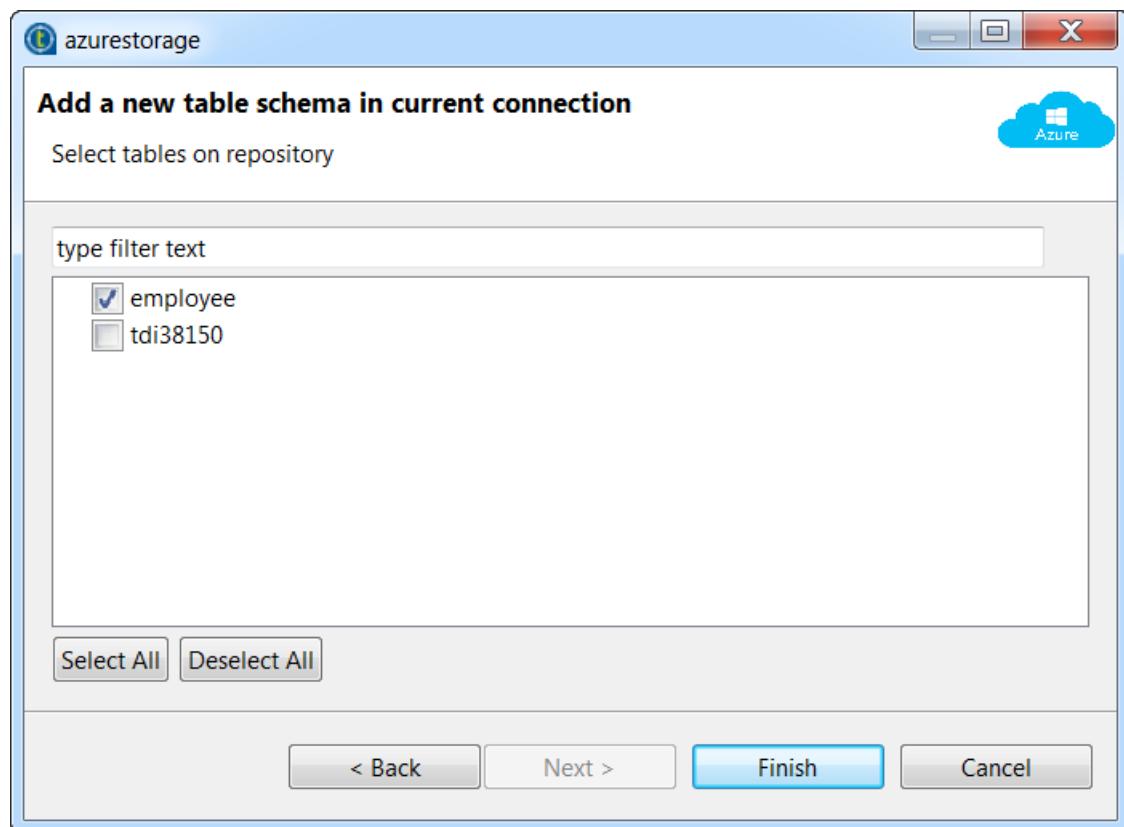
4. Click **Next** and in the **Add a new container schema in current connection** dialog box displayed, select your interested container(s) whose schema you want to retrieve.



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.

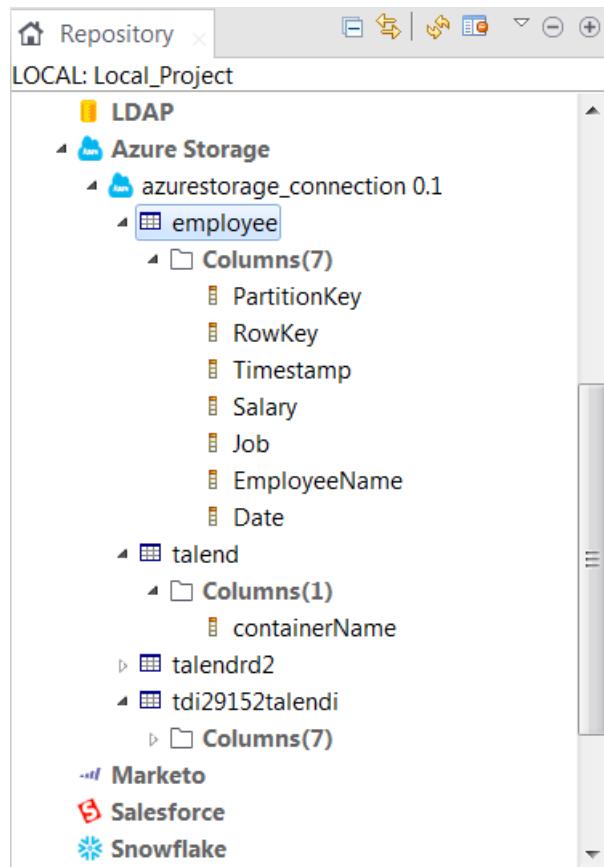


6. Click **Next** and in the **Add a new table schema in current connection** dialog box displayed, select your interested table(s) whose schema you want to retrieve.



- 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 [Using centralized metadata in a Job](#) on page 344. 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.

## Centralizing Google Drive metadata

Talend Studio enables you to centralize the details of your Google Drive connection under the **Metadata** folder in the **Repository** tree view. You can then use the established connection to connect to your Google Drive when using the Google Drive components.

### Procedure

1. In the **Repository** tree view, expand the **Metadata** node, right-click the **Google Drive** tree node, and select **New GoogleDrive Connection** from the contextual menu to open the **New Google Drive Connection** wizard.
2. Specify the values for the properties listed in the following table according to the OAuth method you are using.

Property	Description
<b>Name</b>	The name for the Google Drive connection to be created.
<b>Application Name</b>	The application name required by Google Drive to get access to its APIs.
<b>OAuth Method</b>	<p>Select an OAuth method used to access Google Drive from the drop-down list.</p> <ul style="list-style-type: none"> <li>• <b>Access Token (deprecated)</b>: uses an access token to access Google Drive.</li> <li>• <b>Installed Application (Id &amp; Secret)</b>: uses the client ID and client secret created through Google API Console to access Google Drive. For more information about this method, see <a href="#">Google Identity Platform &gt; Installed applications</a>.</li> <li>• <b>Installed Application (JSON)</b>: uses the client secret JSON file that is created through Google API Console and contains the client ID, client secret, and other OAuth 2.0 parameters to access Google Drive.</li> <li>• <b>Service Account</b>: uses a service account JSON file created through Google API Console to access Google Drive. For more information about this method, see <a href="#">Google Identity Platform &gt; Service accounts</a>.</li> </ul>

Property	Description
	For more detailed information about how to access Google Drive using each method, see the description of OAuth methods for accessing Google Drive in Talend Components Reference Guide or on Talend Help Center ( <a href="https://help.talend.com">https://help.talend.com</a> ).
<b>Access Token</b>	The access token generated through Google Developers OAuth 2.0 Playgroun. This property is available only when <b>Access Token</b> is selected from the <b>OAuth Method</b> drop-down list.
<b>Client ID and Client Secret</b>	The client ID and client secret. These two properties are available only when <b>Installed Application (Id &amp; Secret)</b> is selected from the <b>OAuth Method</b> drop-down list.
<b>Client Secret JSON</b>	The path to the client secret JSON file. This property is available only when <b>Installed Application (JSON)</b> is selected from the <b>OAuth Method</b> drop-down list.
<b>Service Account JSON</b>	The path to the service account JSON file. This property is available only when <b>Service Account</b> is selected from the <b>OAuth Method</b> drop-down list.
<b>DataStore Path</b>	The path to the credential file that stores the refresh token. This property is available only when <b>Installed Application (Id &amp; Secret)</b> or <b>Installed Application (JSON)</b> is selected from the <b>OAuth Method</b> drop-down list.
<b>Use Proxy</b>	Select this check box when you are working behind a proxy. With this check box selected, you need to specify the value for the following parameters: <ul style="list-style-type: none"> <li>• <b>Host:</b> The IP address of the HTTP proxy server.</li> <li>• <b>Port:</b> The port number of the HTTP proxy server.</li> </ul>
<b>Use SSL</b>	Select this check box if an SSL connection is used to access Google Drive. With this check box selected, you need to specify the value for the following parameters: <ul style="list-style-type: none"> <li>• <b>Algorithm:</b> The name of the SSL cryptography algorithm.</li> <li>• <b>Keystore File:</b> The path to the certificate TrustStore file that contains the list of certificates the client trusts.</li> <li>• <b>Password:</b> The password used to check the integrity of the TrustStore data.</li> </ul>

**3.** Click **Test connection** to verify the configuration.

If you are using the OAuth method **Access Token** (deprecated), **Installed Application (Id & Secret)**, or **Installed Application (JSON)**, a window will pop up in your web browser, asking you to choose your account and allow the access to your Google

Drive. After the authentication in web browser, a connection successful dialog box will prompt up in Talend Studio.

4. Click **OK** to close the connection successful dialog box and then click **Finish**.

The newly created Google Drive connection is displayed under the **Google Drive** node in the **Repository** tree view.

You can now add a Google Drive component onto the design workspace by dragging and dropping the new Google Drive connection node to reuse the connection information. For more information about dropping component metadata in the design workspace, see [Using centralized metadata in a Job](#) on page 344. For more information about the usage of the Google Drive components, see the related documentation for the Google Drive components.

To modify the Google Drive connection metadata created, right-click the connection node in the **Repository** tree view and select **Edit GoogleDrive Connection** from the contextual menu to open the metadata setup wizard.

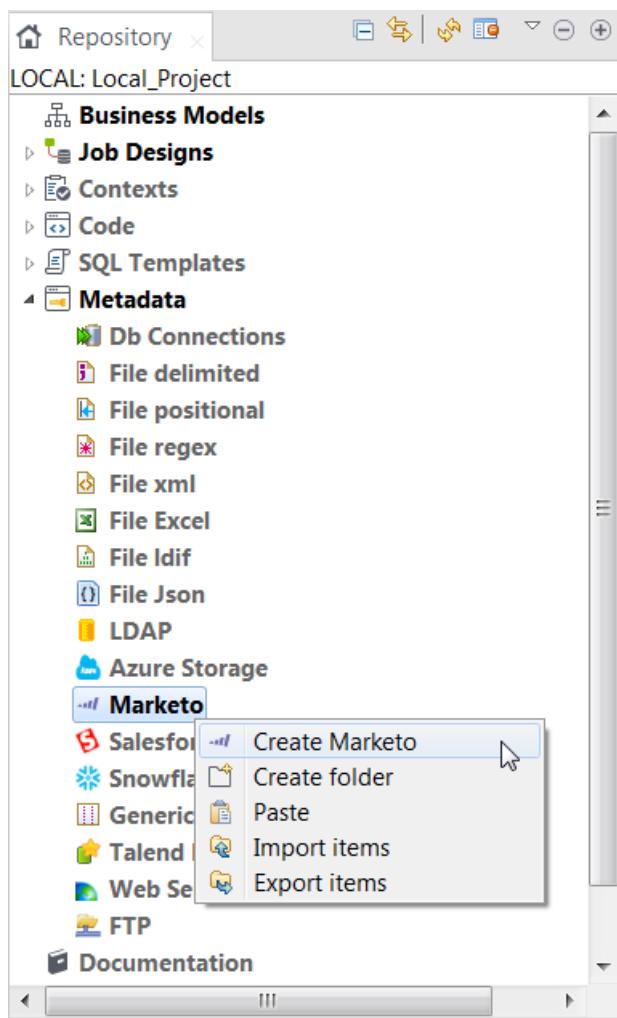
## Centralizing Marketo metadata

### About this task

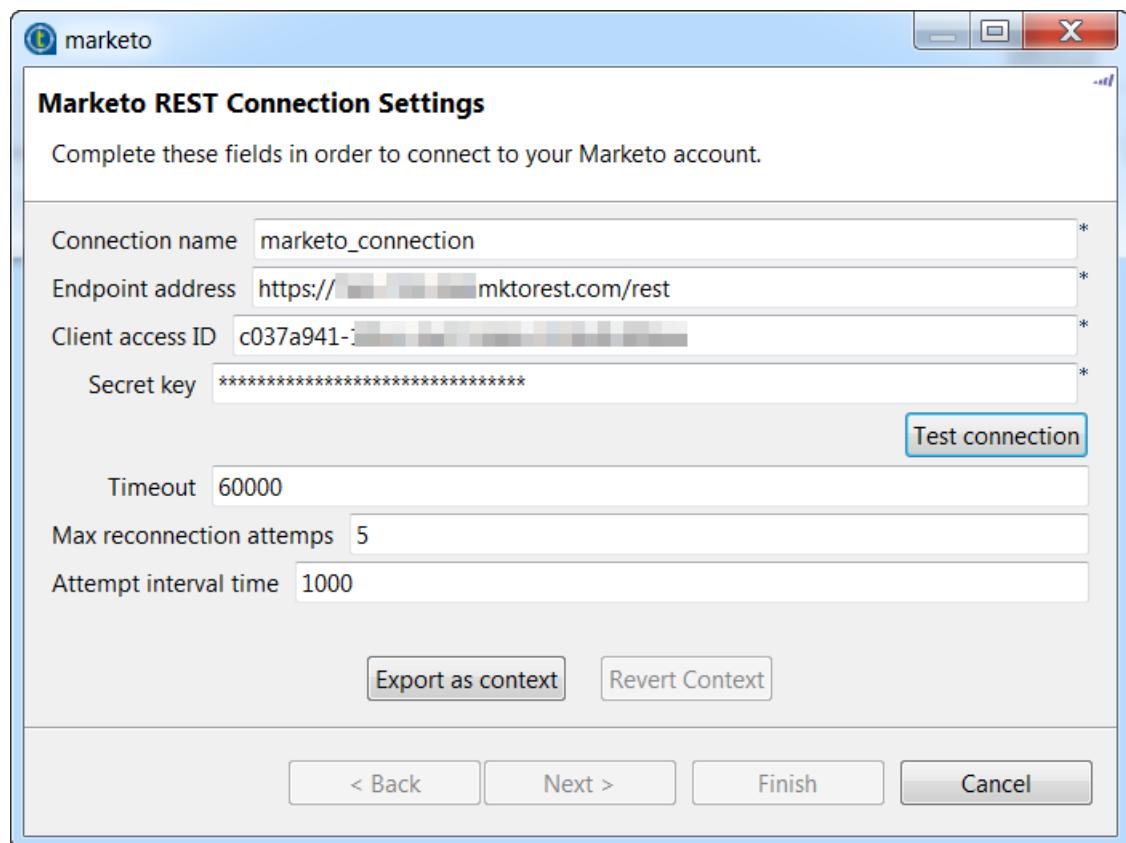
You can use the Marketo metadata wizard provided by Talend Studio to set up quickly a connection to Marketo and retrieve the schema of your interested custom objects using REST API.

### Procedure

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.

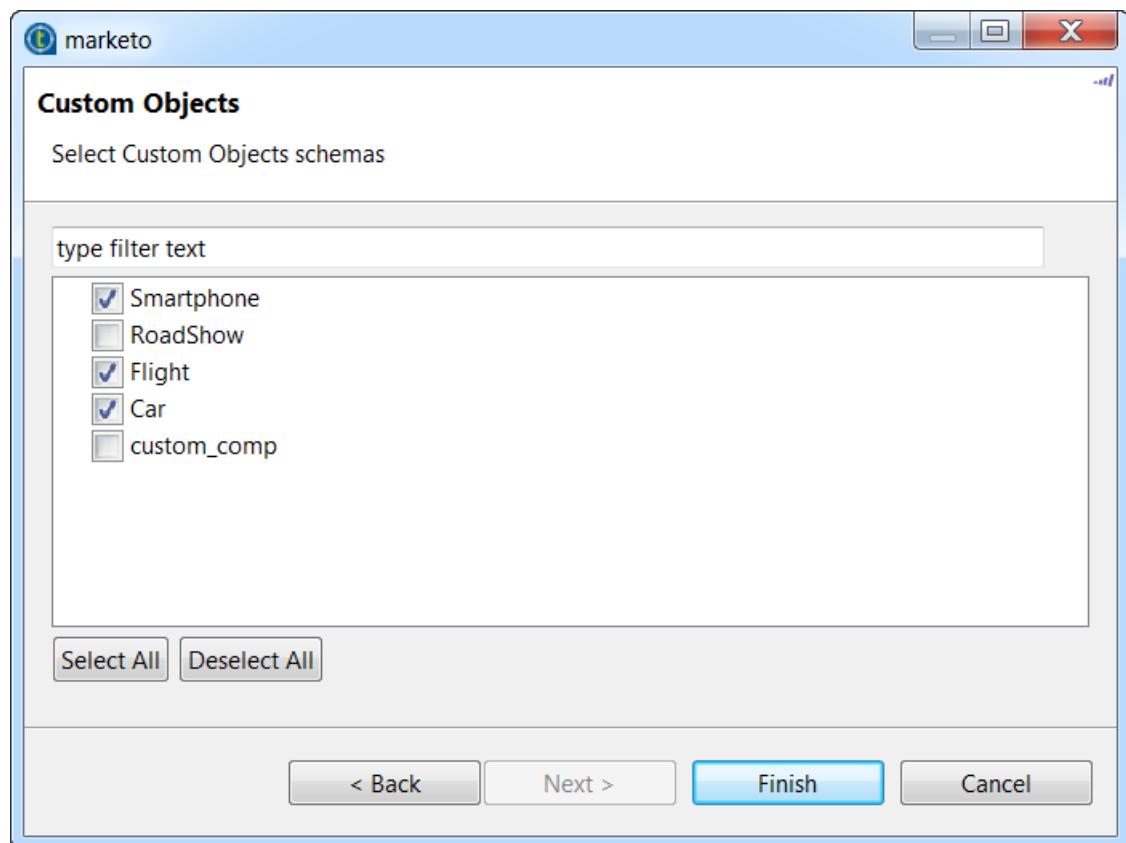


Property	Description
<b>Connection name</b>	Enter the name for the connection to be created.
<b>Endpoint address</b>	Enter the API Endpoint URL of the Marketo Web Service. The API Endpoint URL can be found on the Marketo <b>Admin &gt; Web Services</b> panel.
<b>Client access ID</b>	Enter the client Id 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.

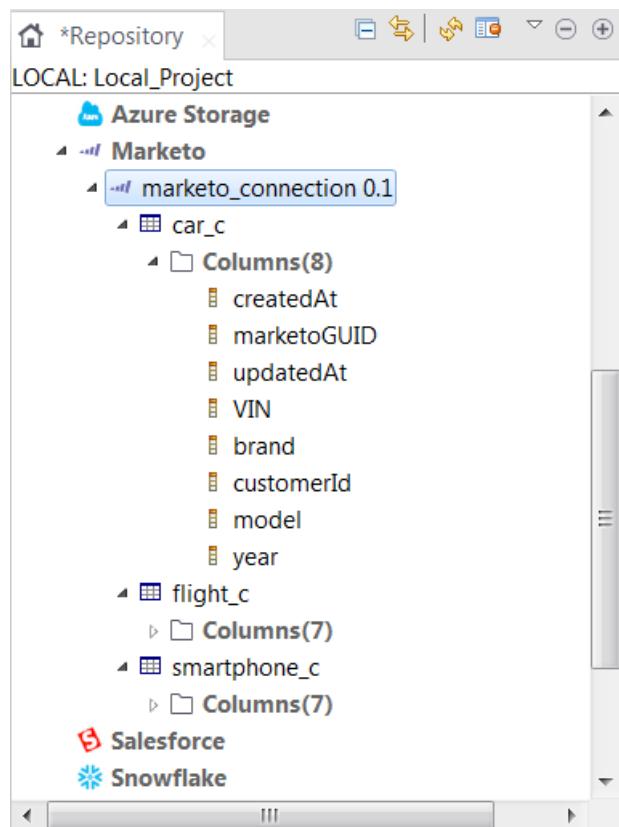
A connection successful dialog box will prompt up if the connection information provided is correct. Then click **OK** to close the dialog box. The **Next** button will be available to use.

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 [Using centralized metadata in a Job](#) on page 344. 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.

## Centralizing Salesforce metadata

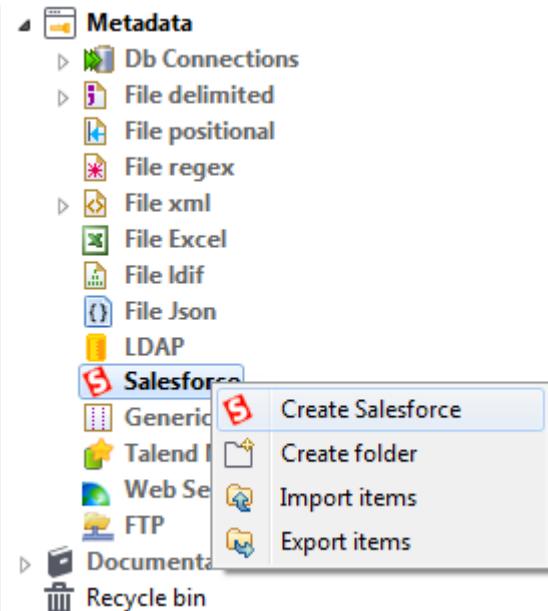
You can use the Salesforce metadata wizard provided by Talend Studio to set up quickly a connection to a Salesforce system so that you can reuse your Salesforce metadata across Jobs.

### About this task

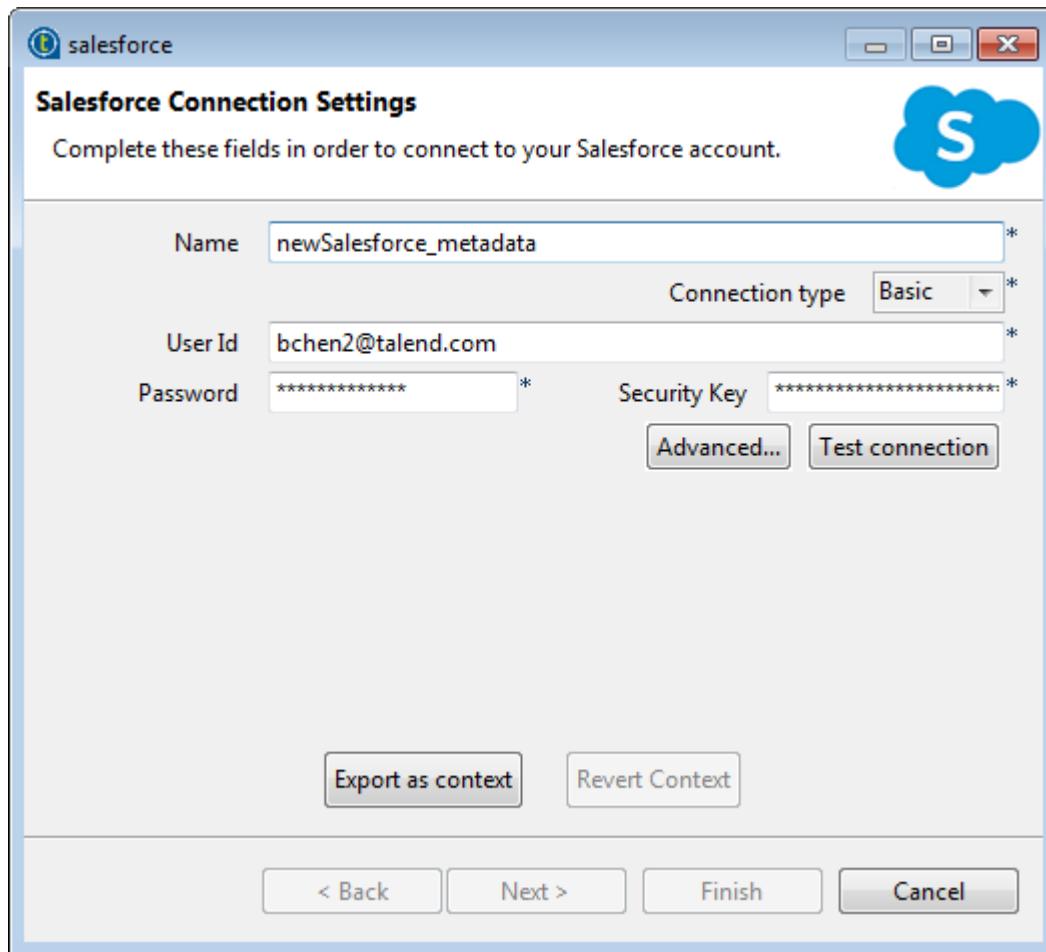
You can use the Salesforce metadata wizard provided by Talend Studio to set up quickly a connection to a Salesforce system so that you can reuse your Salesforce metadata across Jobs.

### Procedure

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 **Use or save the connection session** check box and in the **Session directory** field displayed, specify the path to the connection session file to be saved or used.
- This session file can be shared by different Jobs to retrieve a connection session as long as the correct user ID is provided by the component. This way, you do not need to connect to the server to retrieve the session.

When an expired session is detected, if the correct connection information (the user ID, password, and security key) is provided, the component will connect to the server to retrieve the new session information and update the connection session file.

This check box is available only when **Basic** is selected from the **Connection type** drop-down list.

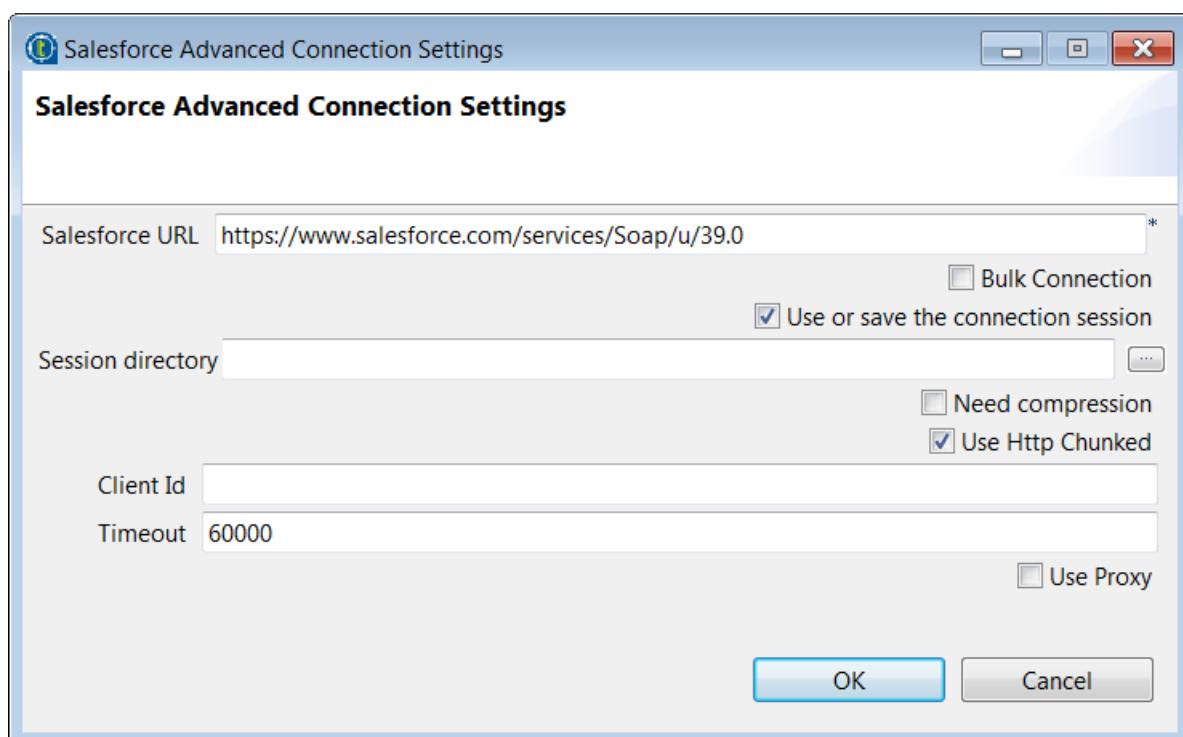
- select the **Need compression** check box to activate SOAP message compression, which can result in increased performance levels.
- select the **Trace HTTP message** check box to output the HTTP interactions on the console.

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.

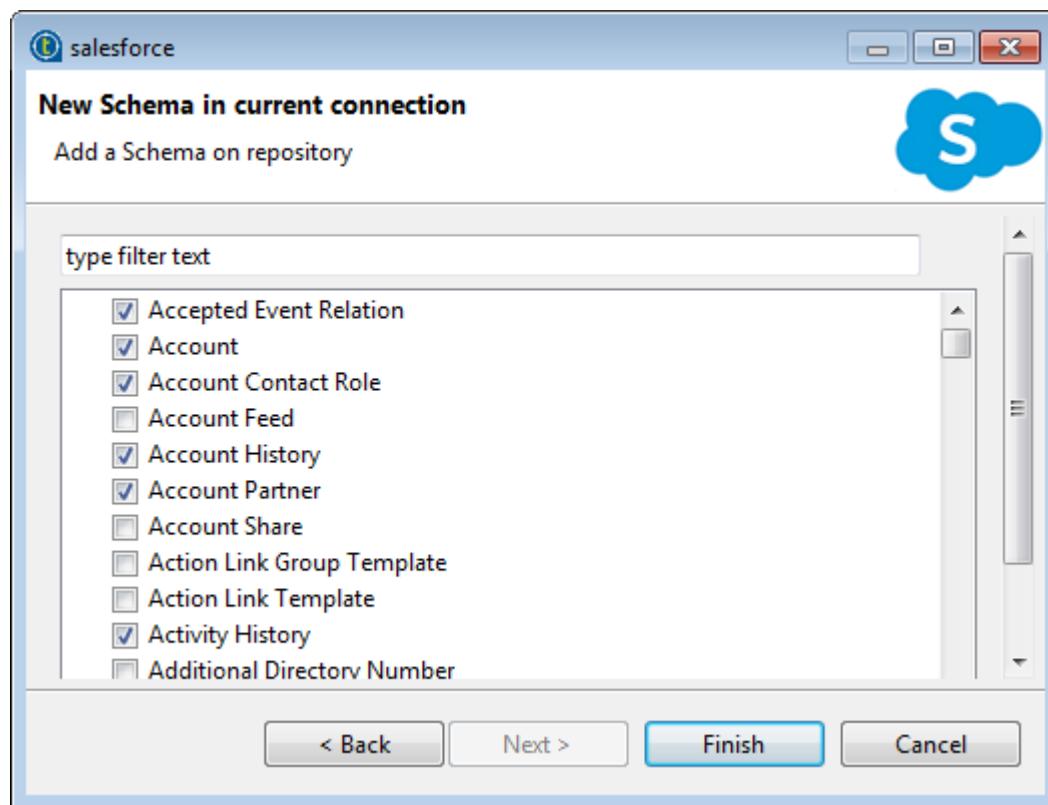
This option is not available if the **Bulk Connection** check box is selected.

- 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.
- If needed, select the **Use Proxy** check box to set the SOCKS type proxy and enter the corresponding setting details. Note that you can also set the HTTP type proxy via **Window > Preferences > General > Network Connections**.

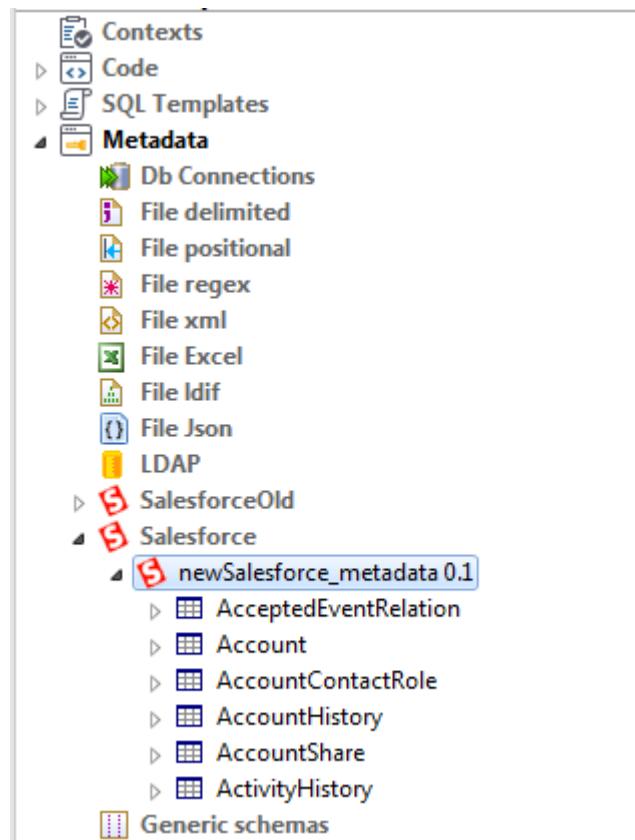


- Click **Test connection** to verify the connection settings, and when the connection check success message appears, click **OK** for confirmation. Then click **Next** to go to the next step to select the modules you want to retrieve the schema of.
- Select the check boxes for the modules of interest 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.



## Results

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 [Using centralized metadata in a Job](#) on page 344.

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.

## Centralizing Snowflake metadata

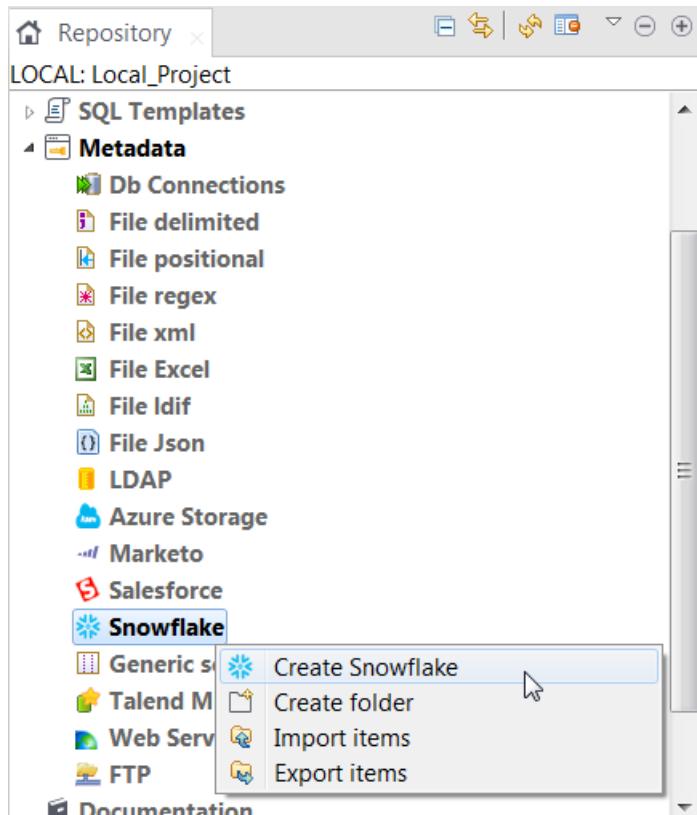
### About this task

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.

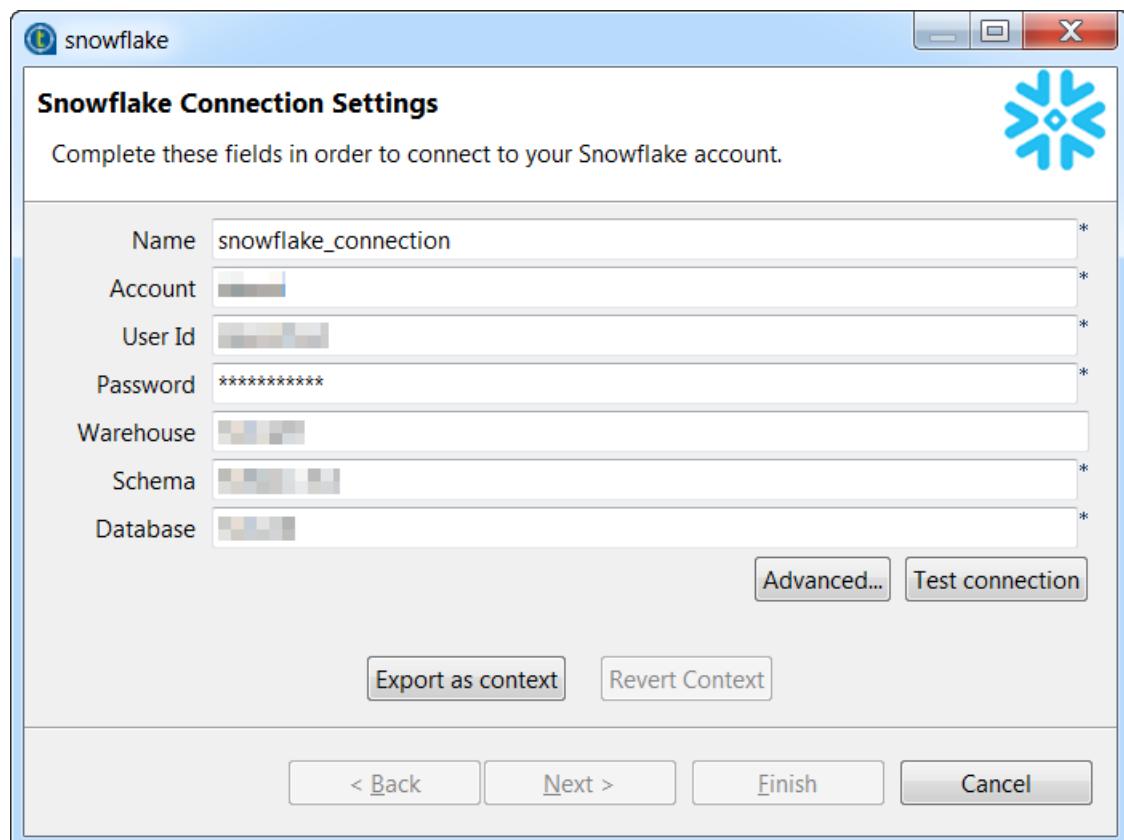
**Note:** The Snowflake metadata wizard doesn't support handling Snowflake views for now.

### Procedure

- 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.

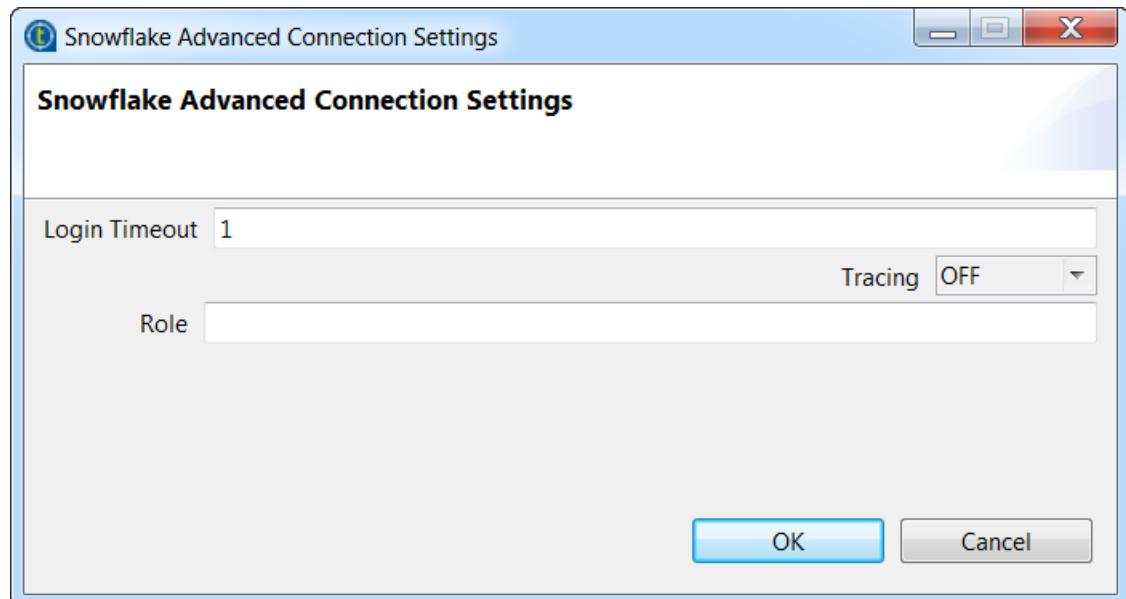


- In the **Snowflake Connection Settings** dialog box, specify the values for the properties listed in the following table.



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.

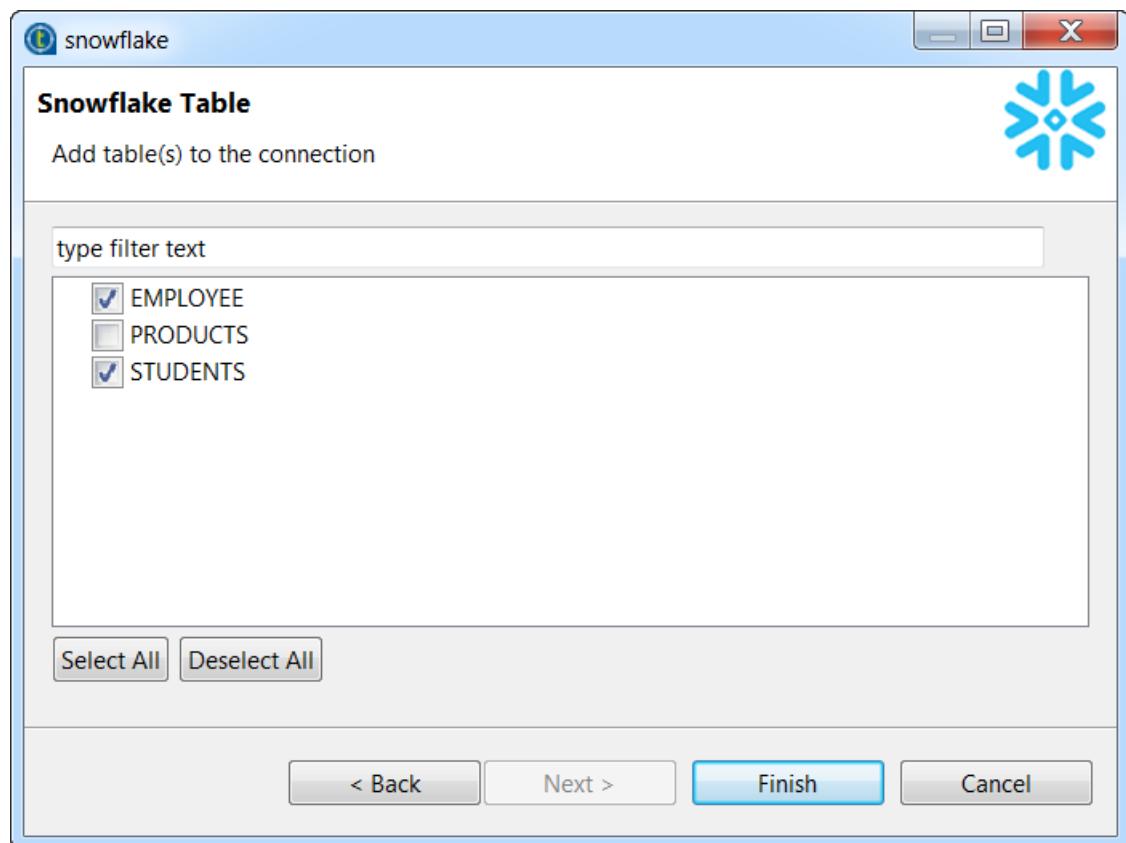


Property	Description
<b>Login Timeout</b>	Specify how long to wait for a response when connecting to Snowflake before returning an error.
<b>Tracing</b>	Select the log level for the Snowflake JDBC driver. If enabled, a standard Java log is generated.
<b>Role</b>	Enter the default access control role 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 <a href="#">Understanding the Access Control Model</a> .

4. 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.

5. 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 [Using centralized metadata in a Job](#) on page 344. 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.

## 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](#) on page 293,
- from a schema definition XML file. For details, see [Setting up a generic schema from an XML file](#) on page 296, and
- from the schema defined in a component. For details, see [Saving a component schema as a generic schema](#) on page 298.

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.

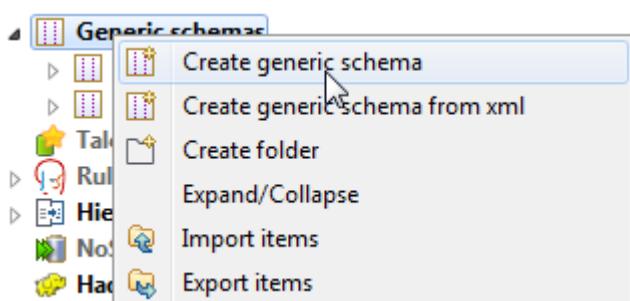
## Setting up a generic schema from scratch

### About this task

To create a generic schema from scratch, proceed as follows:

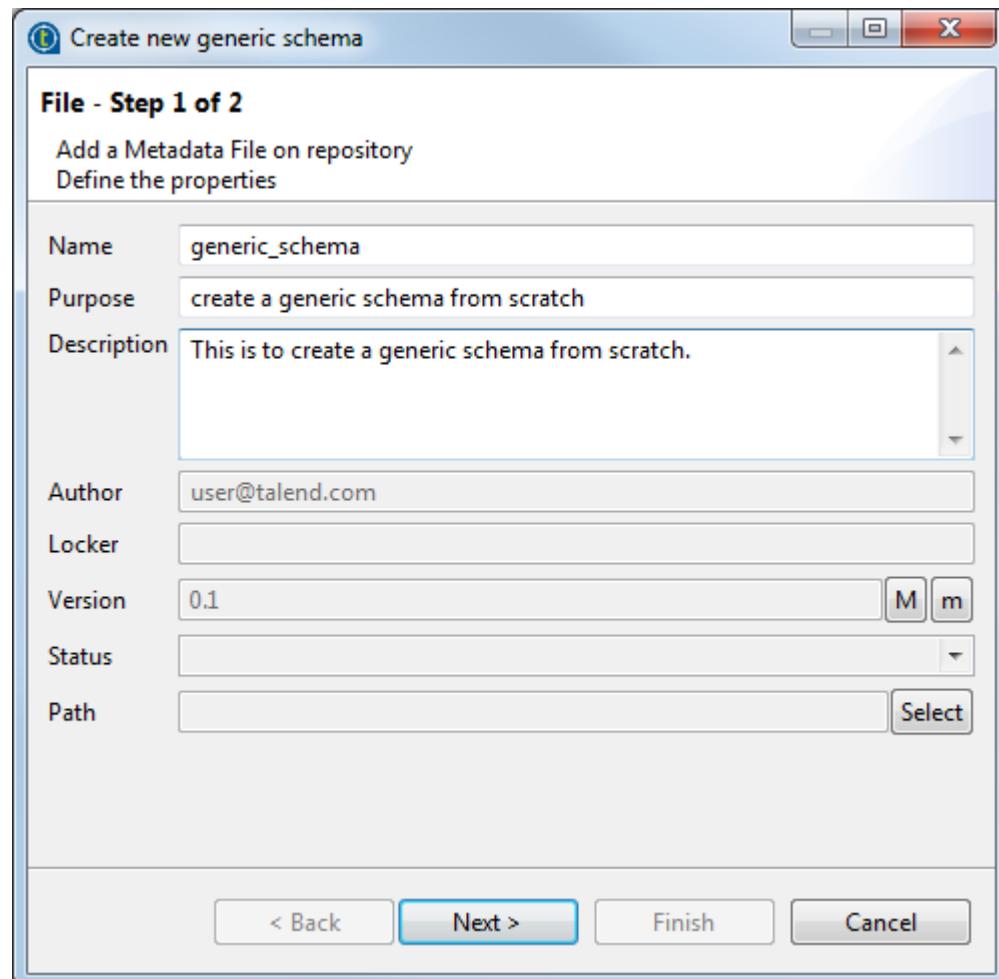
### Procedure

- Right-click **Generic schemas** under the **Metadata** node in the **Repository** tree view, and select **Create generic schema**.

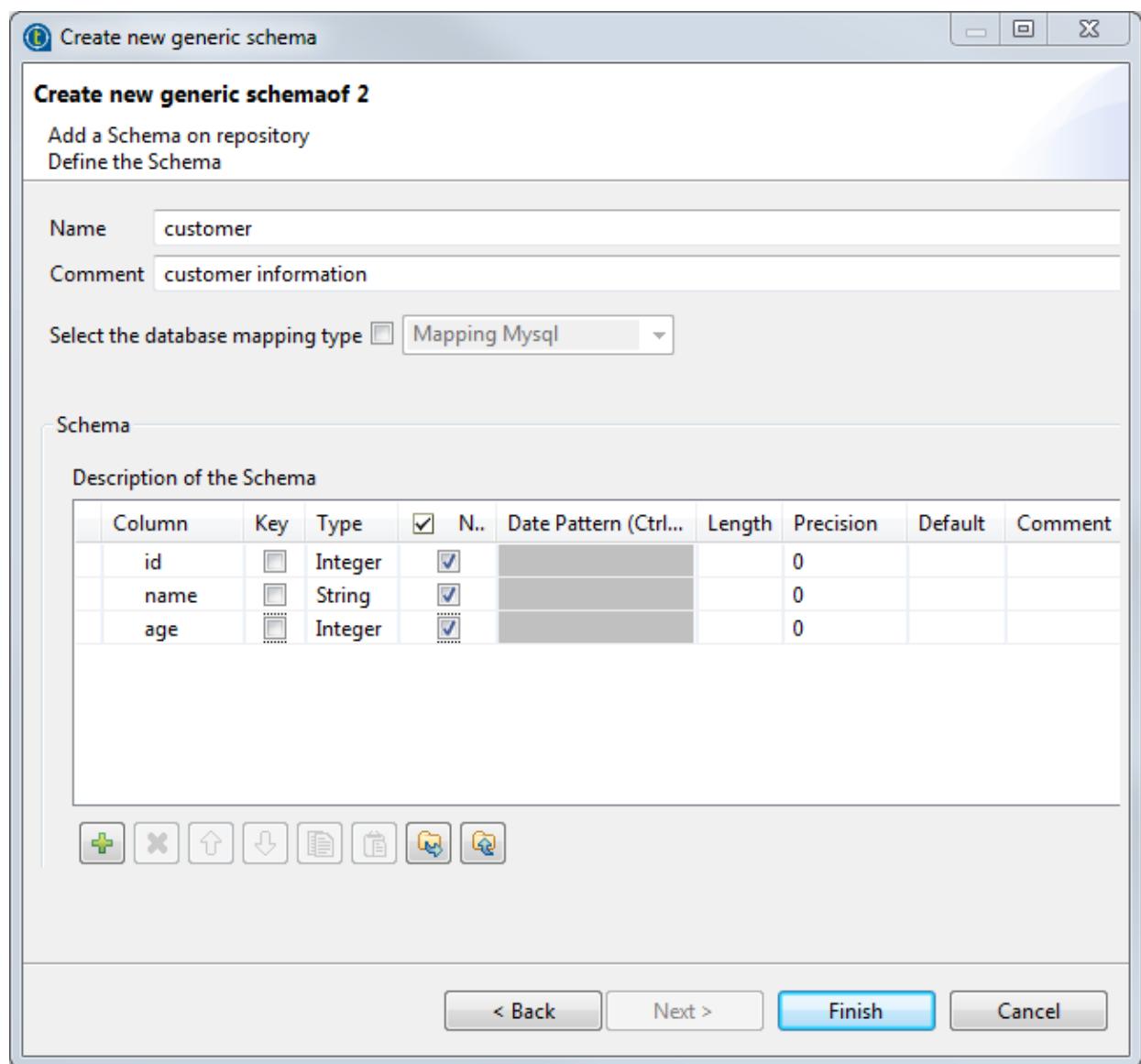


2. In the schema creation wizard that appears, fill in the generic schema properties such as schema **Name** and **Description**. The **Status** field is a customized field. For more information about how to define the field, see [Status settings](#) on page 389.

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.



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.

## Setting up a generic schema from an XML file

### About this task

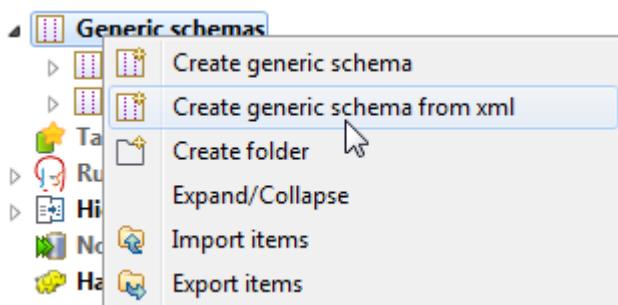
**Warning:**

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:

### Procedure

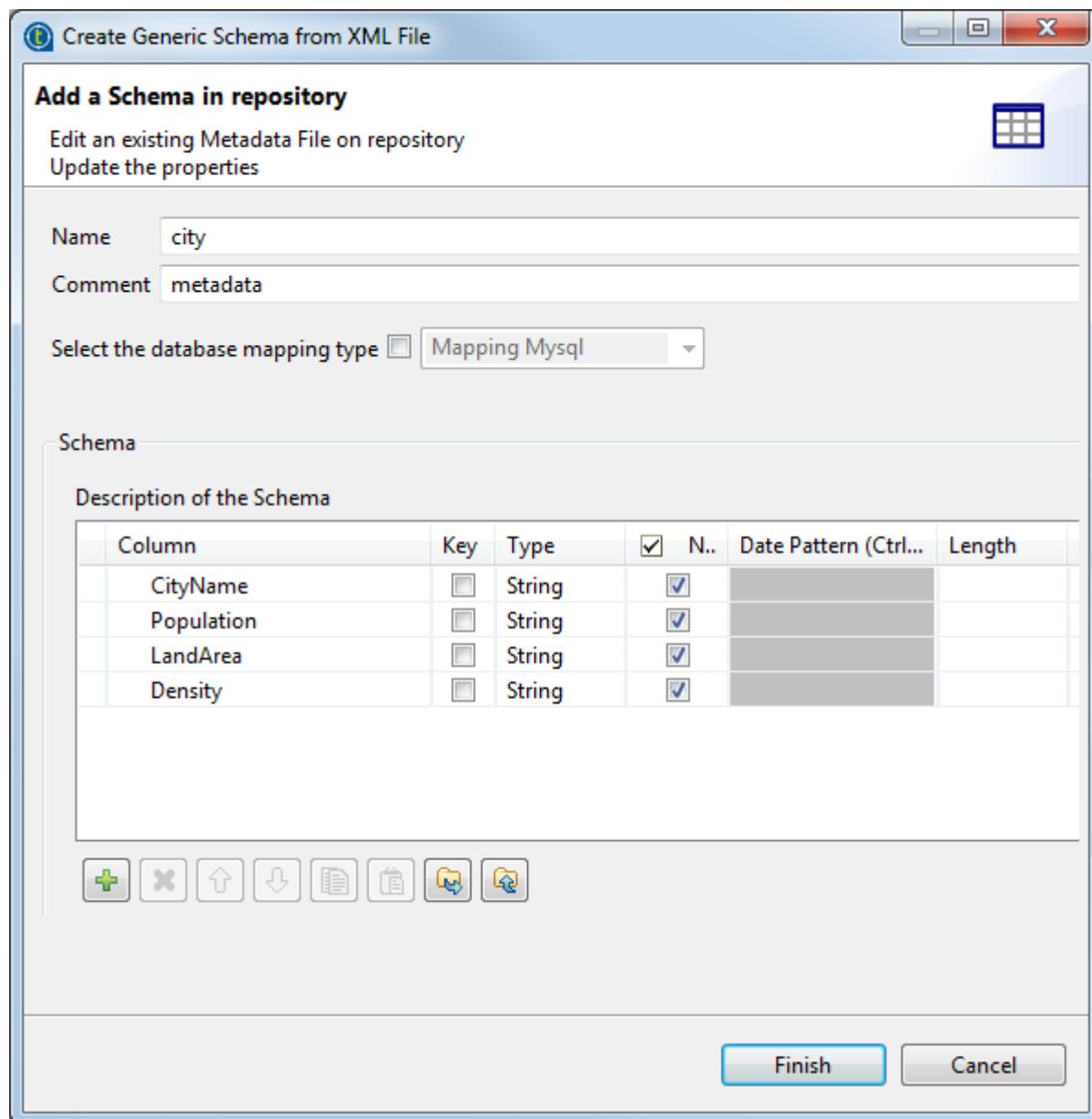
1. Right-click **Generic schemas** in the **Repository** tree view, and select **Create generic schema from xml**.



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.



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.

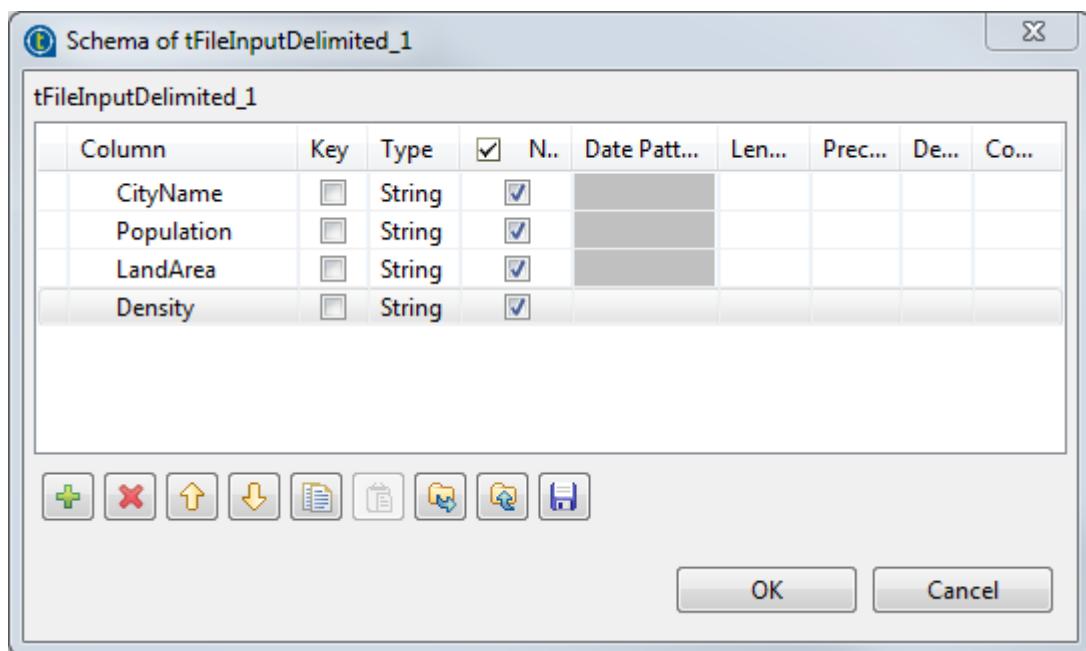
## Saving a component schema as a generic schema

### About this task

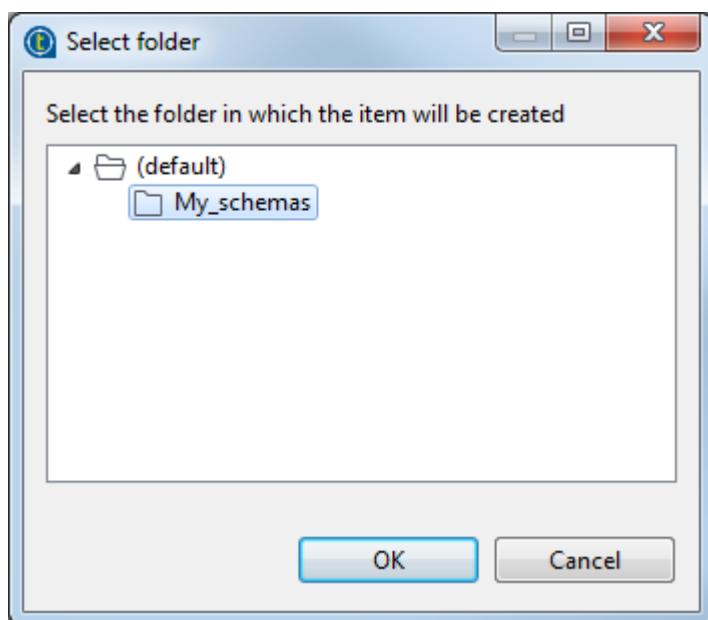
You can create a generic schema by saving the schema defined in a component. To do so, follow the steps below:

### Procedure

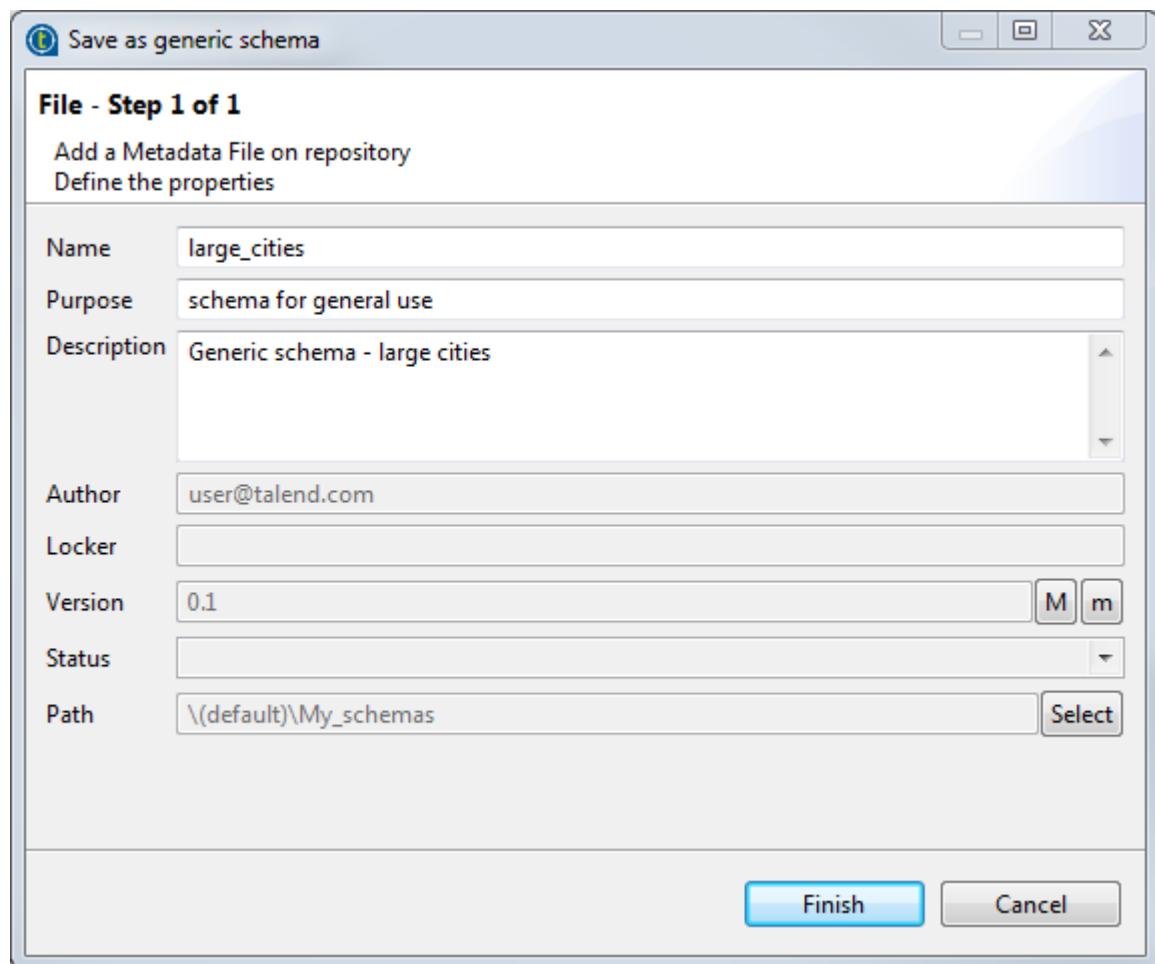
1. Open the **Basic settings** view of the component that has the schema you want to create a generic schema from, and click the [...] button next to **Edit schema** to open the **Schema** dialog box.



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.



4. 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.

## 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.

**Note:**

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.

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.

## Setting up the connection

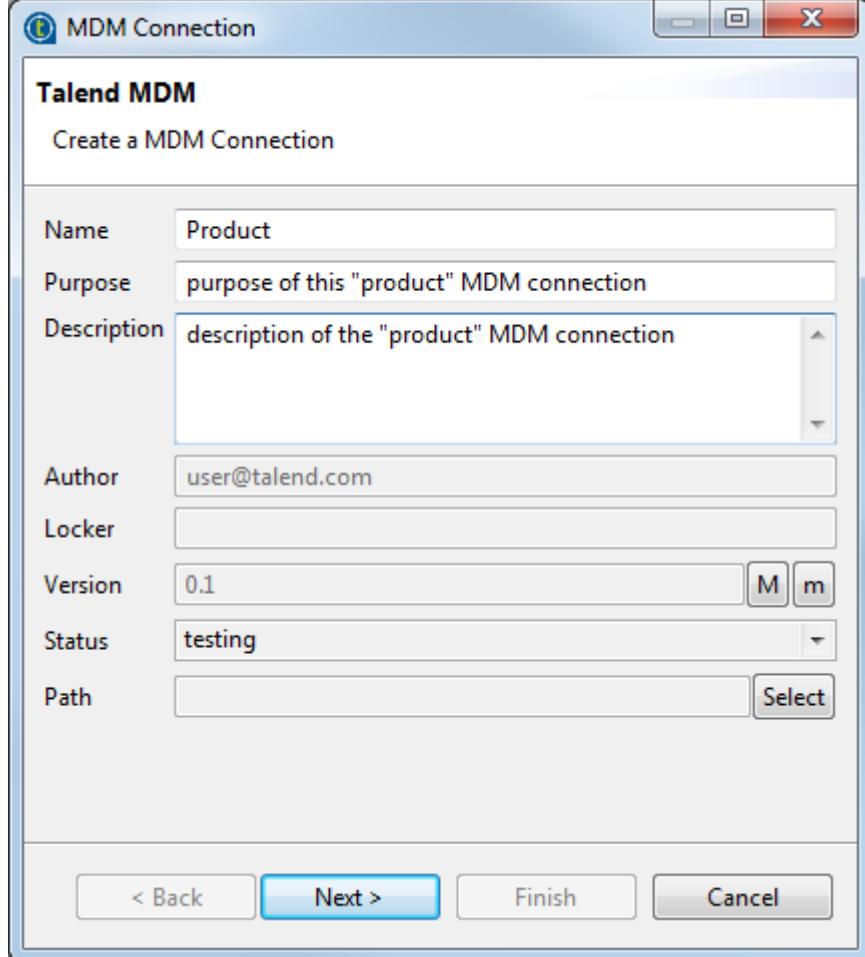
### About this task

To establish an MDM connection, complete the following:

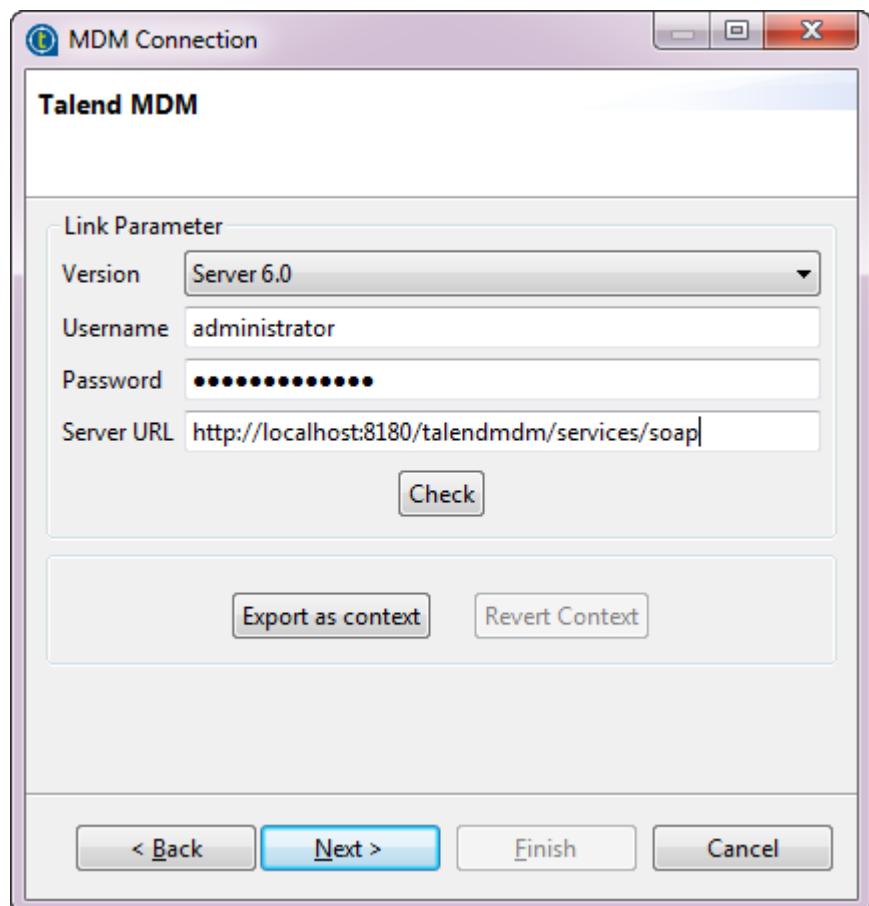
### Procedure

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](#) on page 389.
4. Click **Next** to proceed to the next step.

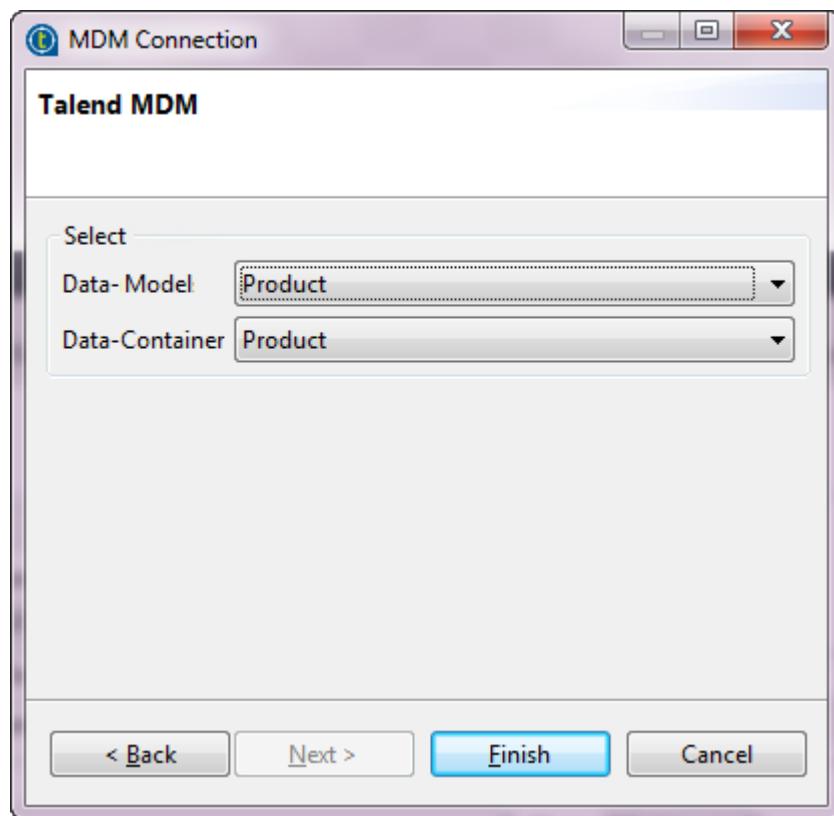


- From the **Version** list, select the version of the MDM server to which you want to connect.

**Note:**

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](#) on page 334.
- 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.

## Results

You need now to retrieve the XML schema of the business entities linked to this MDM connection.

## Defining MDM schema

### 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](#) on page 308.

### Retrieve entity values for an MDM connection

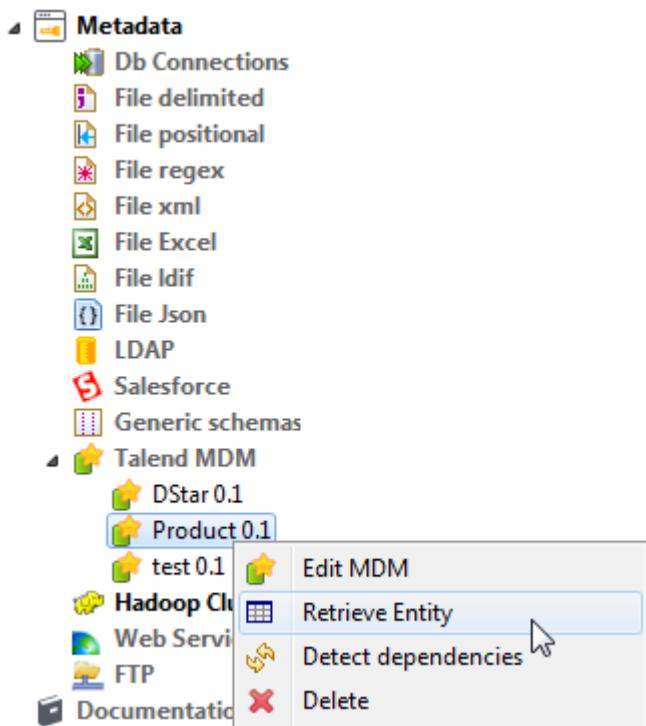
#### About this task

To set the values to be fetched from one or more entities linked to a specific MDM connection, complete the following:

#### Procedure

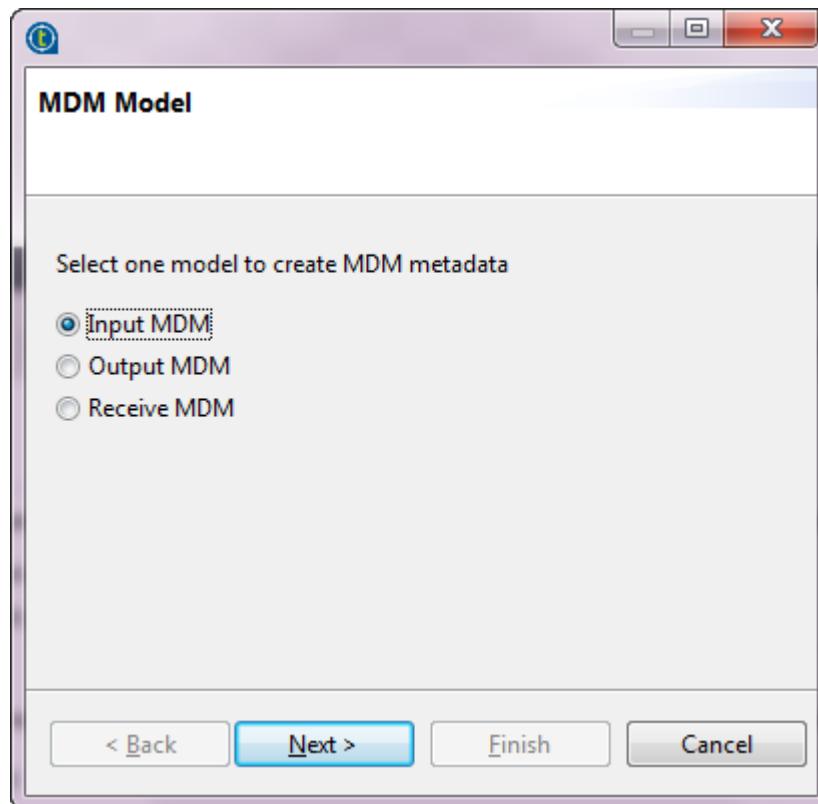
1. In the **Repository** tree view, expand **Metadata** and right-click the MDM connection for which you want to retrieve the entity values, and select **Retrieve Entity** from the contextual menu.

## Example



2. In the **MDM Model** dialog box, select the **Input MDM** option in order to download an input XML schema and then click **Next** to proceed to the following step.

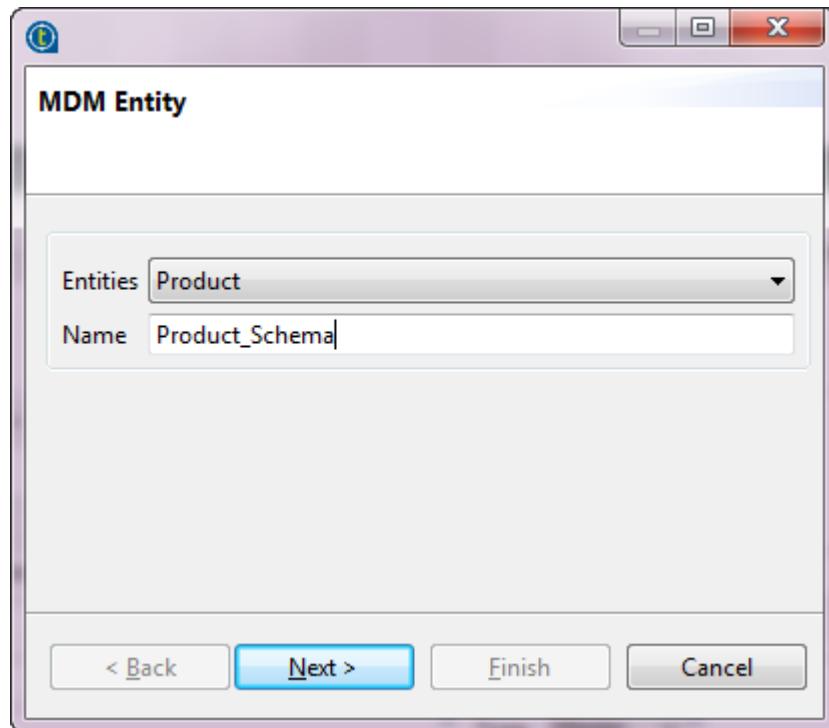
## Example



3. 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.

#### Example



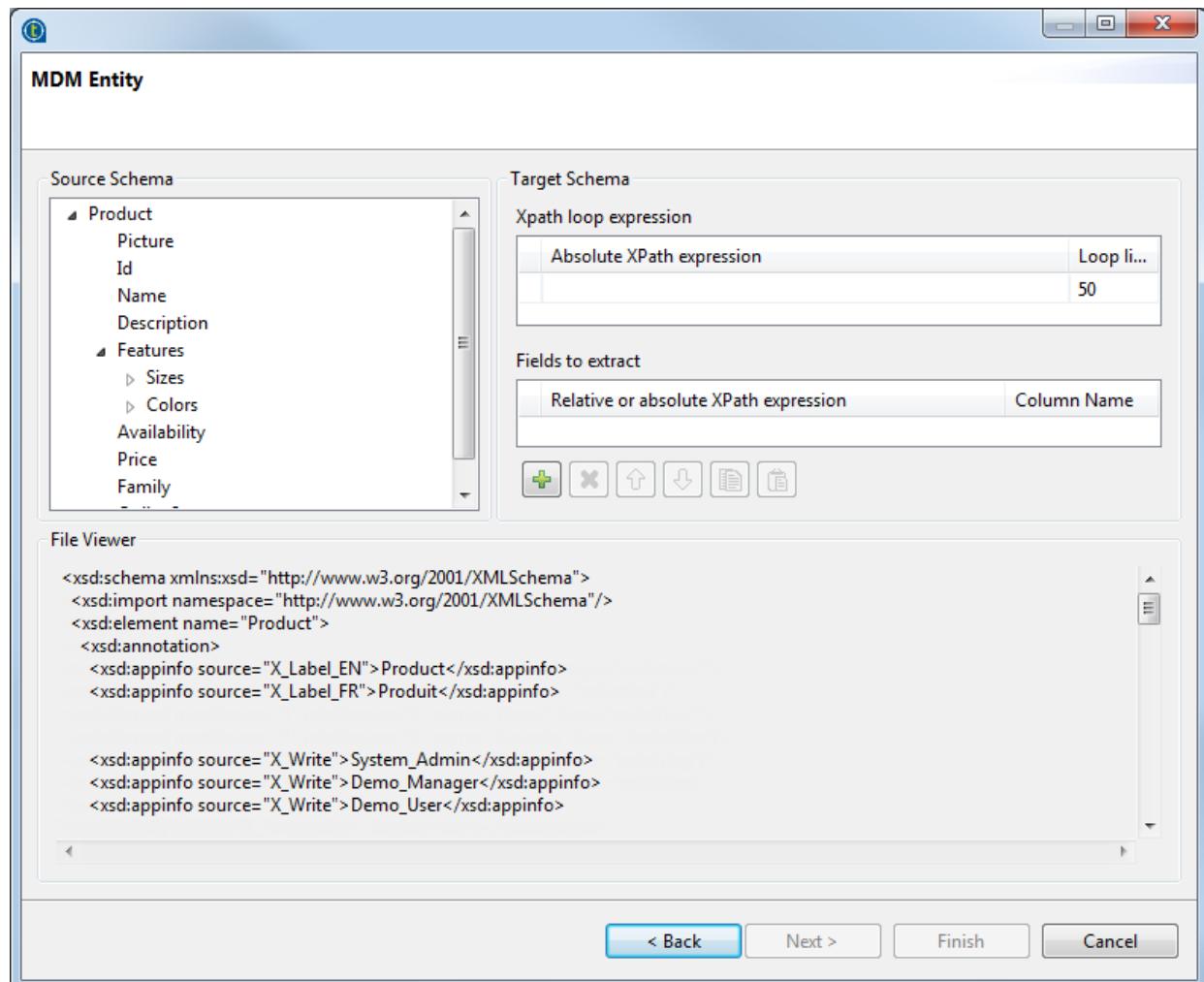
#### Example

**Note:** 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.

4. 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.

## Example



The schema dialog box is divided into four different panels as follows:

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.

5. 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.

**Note:** The **Xpath loop expression** field is compulsory.

6. If required, define a **Loop limit** to restrict the iteration to a number of nodes.

## Example

The screenshot shows the configuration of a Talend MDM connection. On the left, the **Source Schema** panel displays a hierarchical tree of fields for a **Product** entity. The **Features** node is highlighted with an orange arrow pointing to the **Xpath loop expression** field in the **Target Schema** panel on the right. The **Target Schema** panel includes an **Xpath loop expression** table and a **Fields to extract** table, both of which are currently empty.

## Example

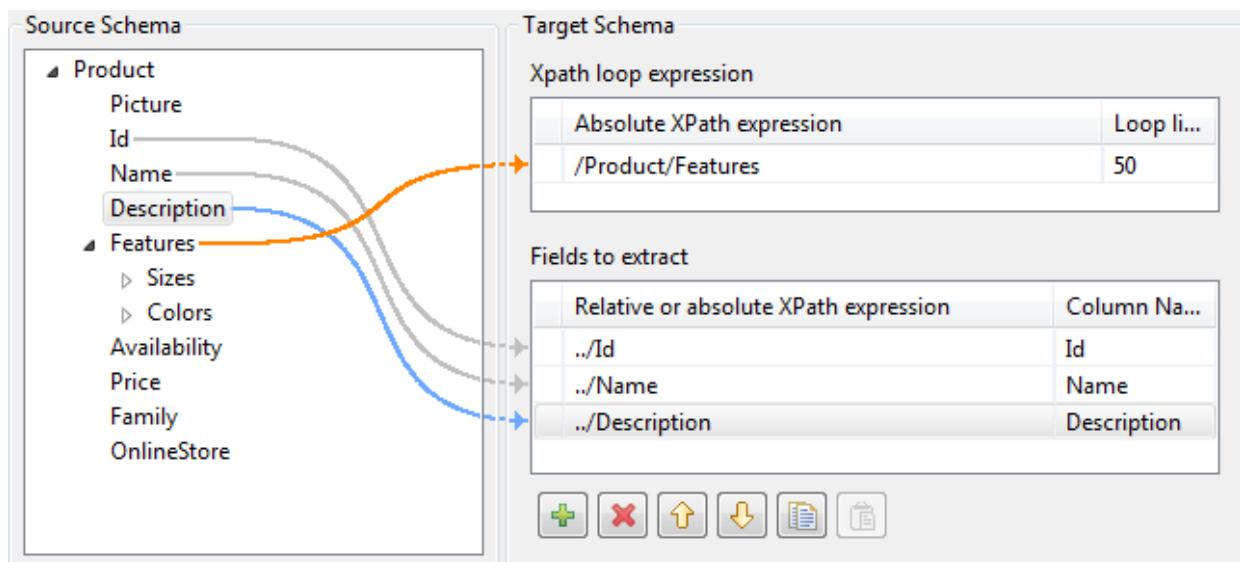
In the capture above, we use **Features** as the element to loop on because it is repeated within the **Product** entity as follows:

```
<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.

7. To define the fields to extract, drop the relevant node from the source schema to the **Relative or absolute XPath expression** field.

## Example



**Tip:** 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.

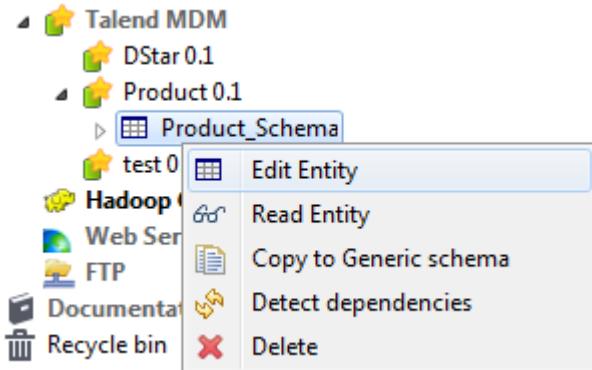
- If required, enter a name to each of the retrieved columns in the **Column name** field.

**Tip:** 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.

- Click **Finish** to validate your modifications and close the dialog box.

## Results

The newly created schema is listed under the corresponding MDM connection in the **Repository** tree view.



## Modifying the created schema

### Procedure

- 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 is displayed.

**Description of the Schema**

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		

Click Guess button to update the schema below according to your settings **Guess**

Tool bar icons: +, -, ↑, ↓, ↻, ↻, ↻, ↻.

### 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.

## Results

The MDM input connection (**tMDMInput**) is now ready to be dropped in any of your Jobs.

## Defining output MDM schema

### About this task

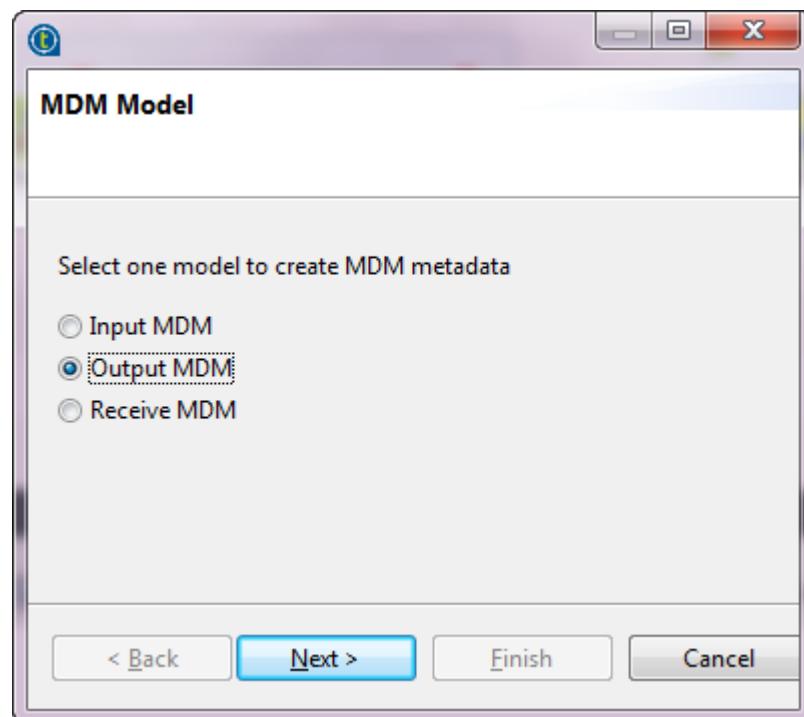
This section describes how to define and download an output MDM XML schema. To define and download an input MDM XML schema, see [Setting up the connection](#) on page 300.

To set the values to be written in one or more entities linked to a specific MDM connection, complete the following:

## Procedure

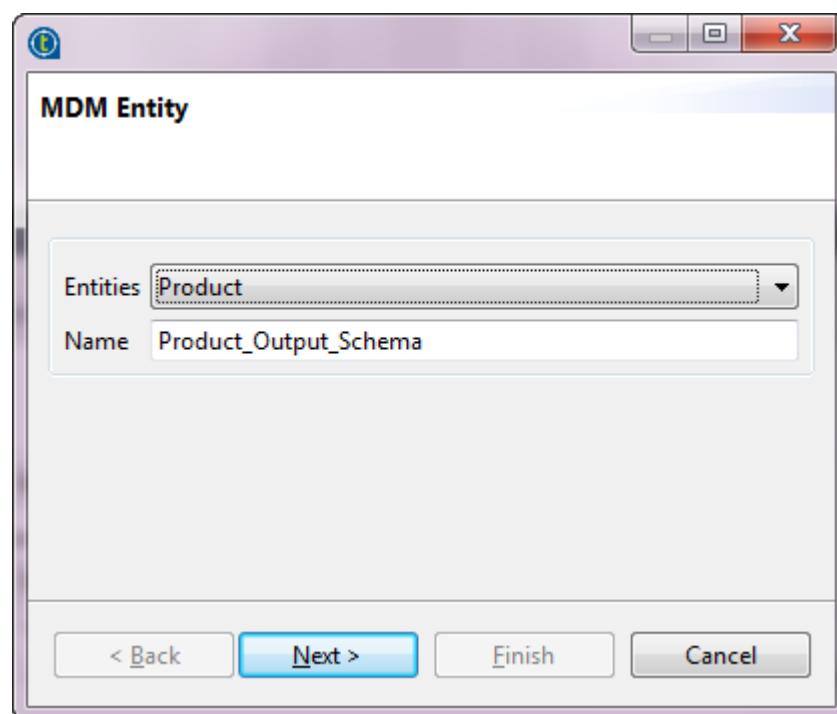
1. In the **Repository** tree view, expand **Metadata** and right-click the MDM connection for which you want to write the entity values, and select **Retrieve Entity** from the contextual menu.
2. In the **MDM Model** dialog box, select the **Output MDM** option in order to define an output XML schema and then click **Next** to proceed to the following step.

## Example



3. From the **Entities** field, select the business entity (XML schema) in which you want to write values.

## Example



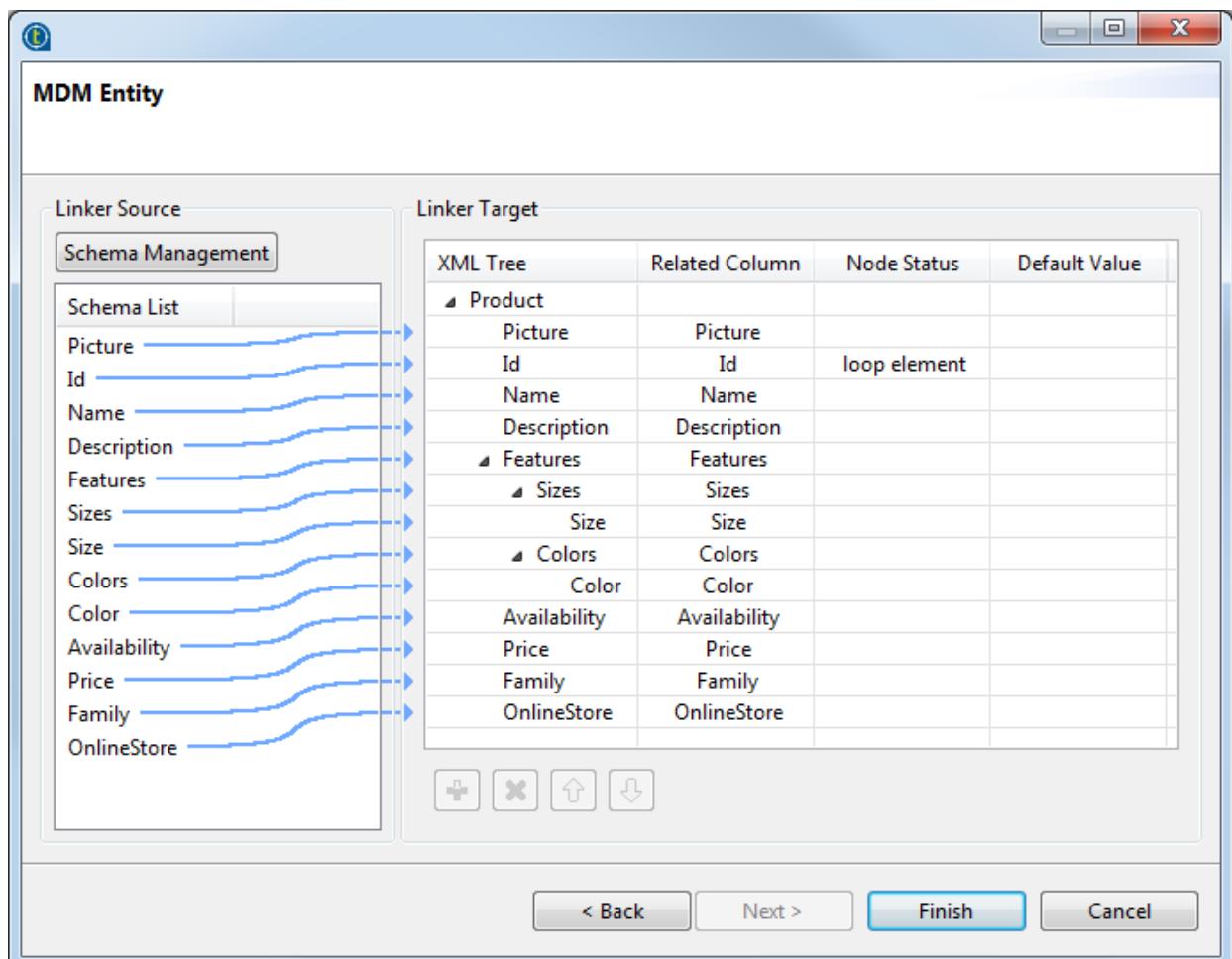
The name is displayed automatically in the **Name** field.

**Note:** 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.

- 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.

### Example



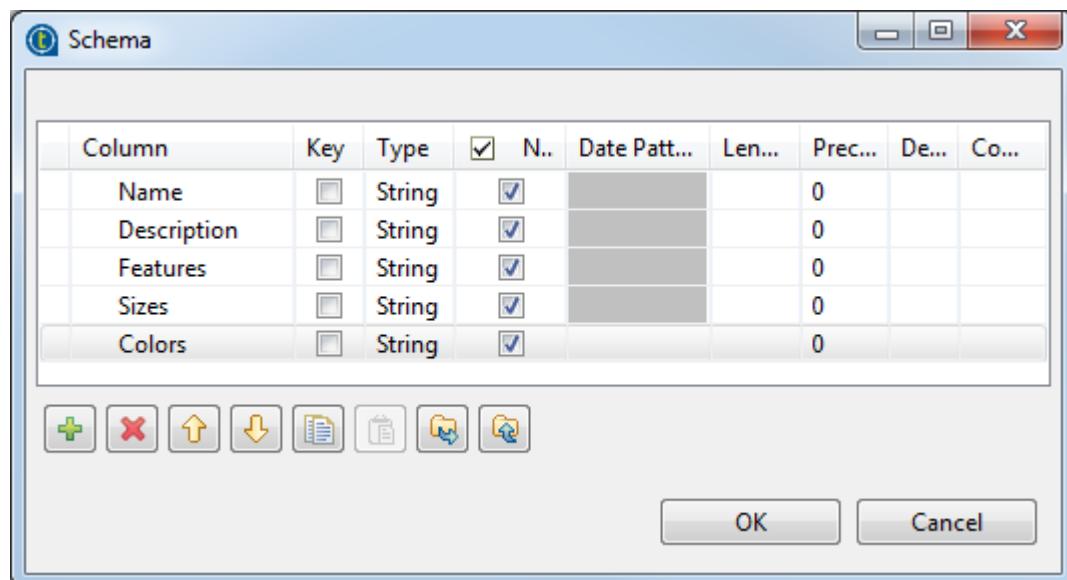
- Click **Schema Management** to display a dialog box.
- Do necessary modifications to define the XML schema you want to write in the selected entity.

**Warning:** 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.

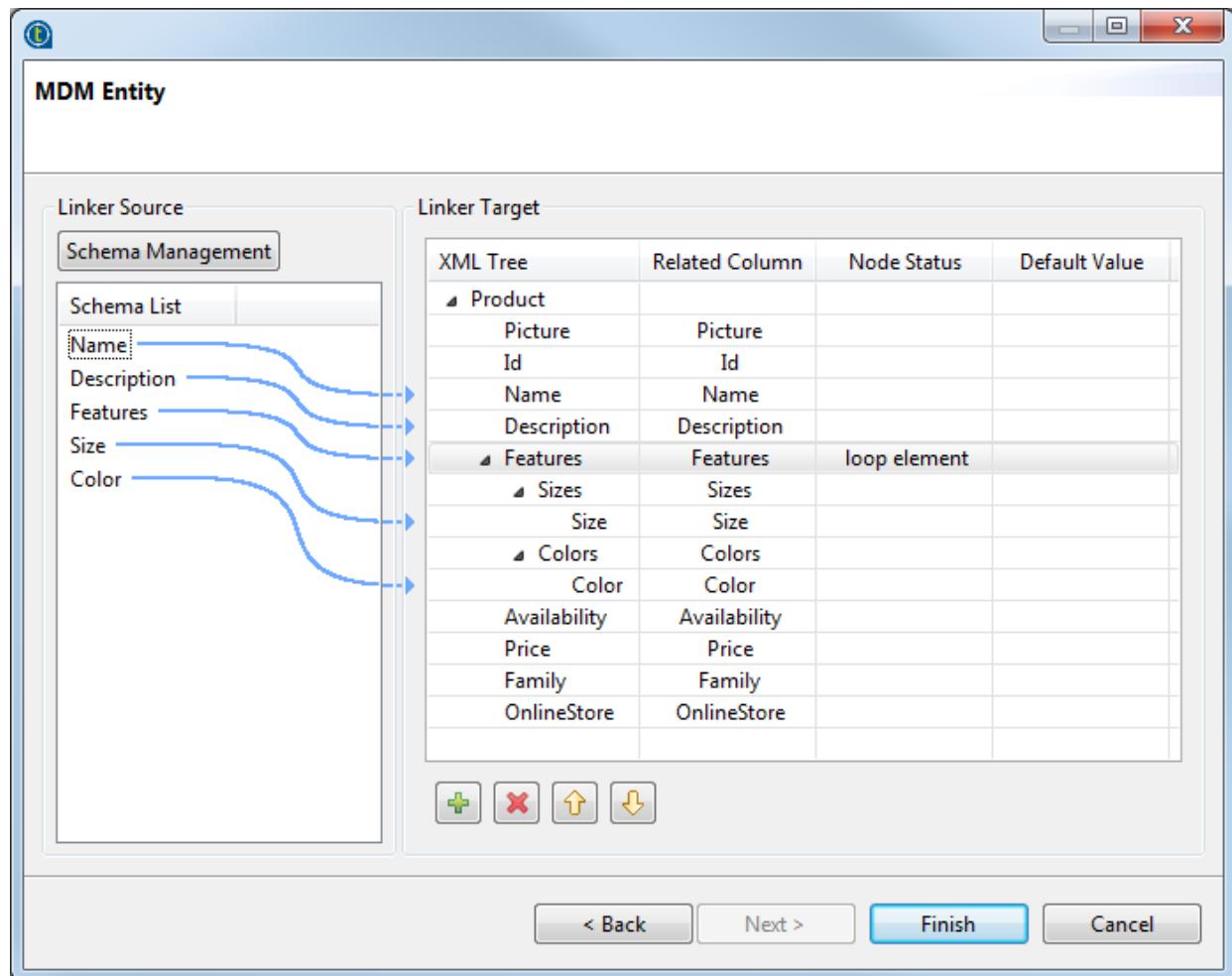
The defined schema is displayed under **Schema list**.

## Example



8. 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.

## Example



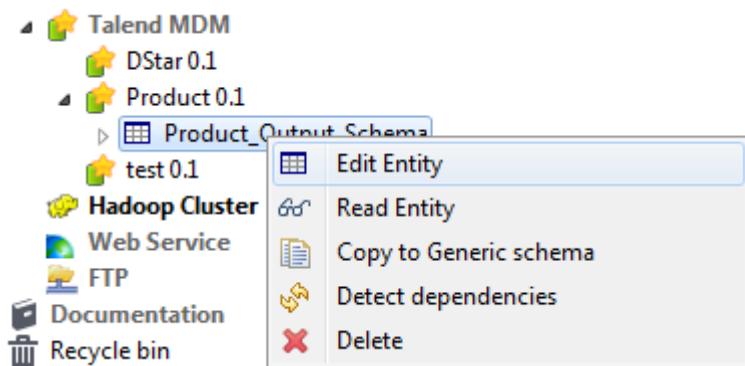
**Tip:** You can prioritize the order of the fields to write by selecting the field and using the up and down arrows.

- Click **Finish** to validate your modifications and close the dialog box.

## Results

The newly created schema is listed under the corresponding MDM connection in the **Repository** tree view. You can modify the created schema according to your needs and drop the connection as a **tMDMOutput** in any of your Jobs.

For more information on how to modify the schema, see [Modifying the created schema](#) on page 307.



## Defining Receive MDM schema

### Before you begin

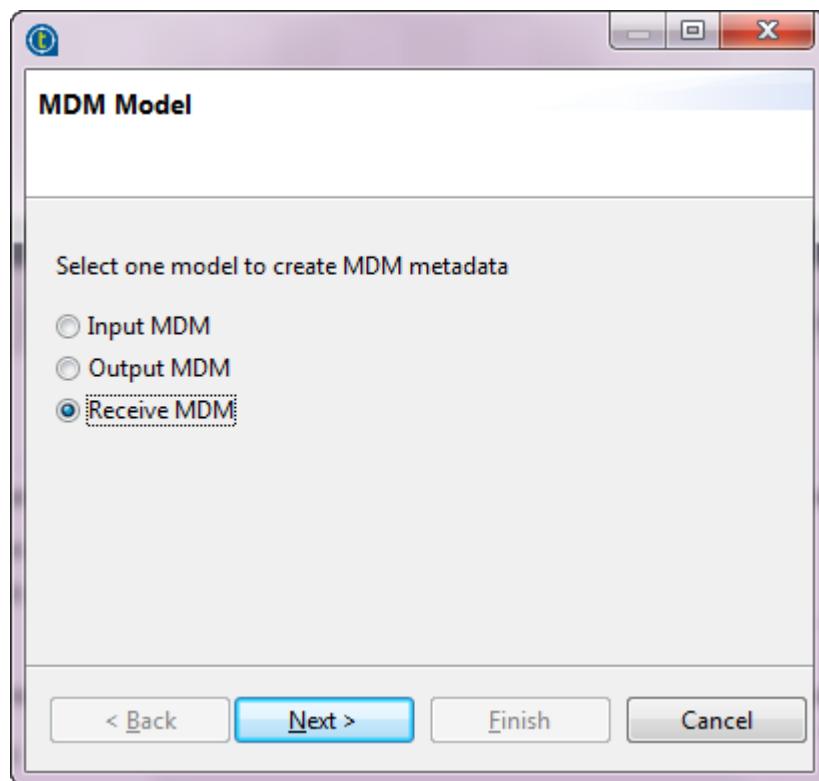
This section describes how to define a receive MDM XML schema based on the MDM connection.

To set the XML schema you want to receive in accordance with a specific MDM connection, complete the following:

### Procedure

1. In the **Repository** tree view, expand **Metadata** and right-click the MDM connection for which you want to retrieve the entity values, and select **Retrieve Entity** from the contextual menu.
2. In the **MDM Model** dialog box, select the **Receive MDM** option in order to define a receive XML schema and then click **Next** to proceed to the following step.

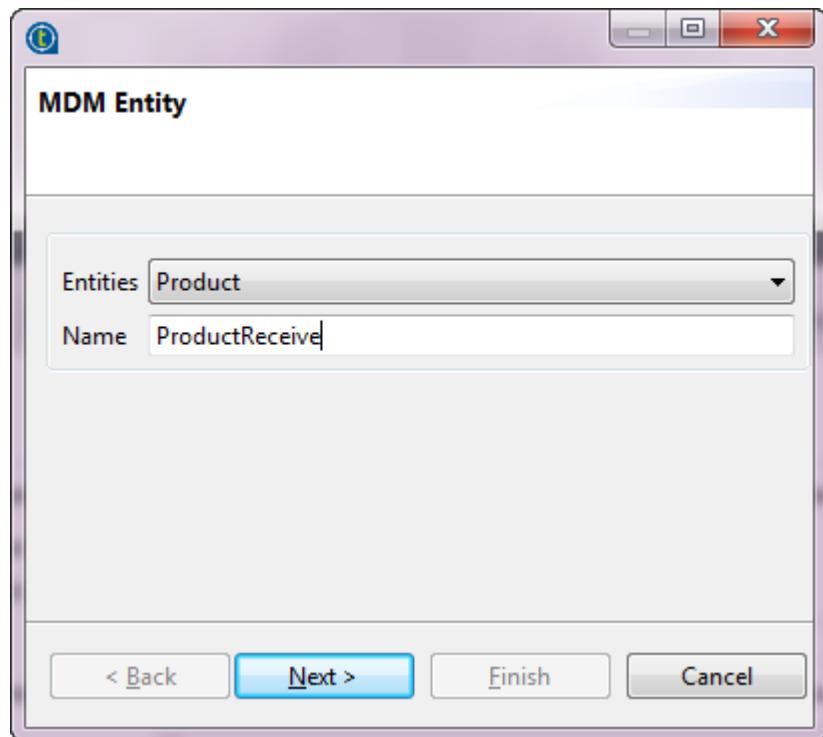
### Example



3. 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.

## Example

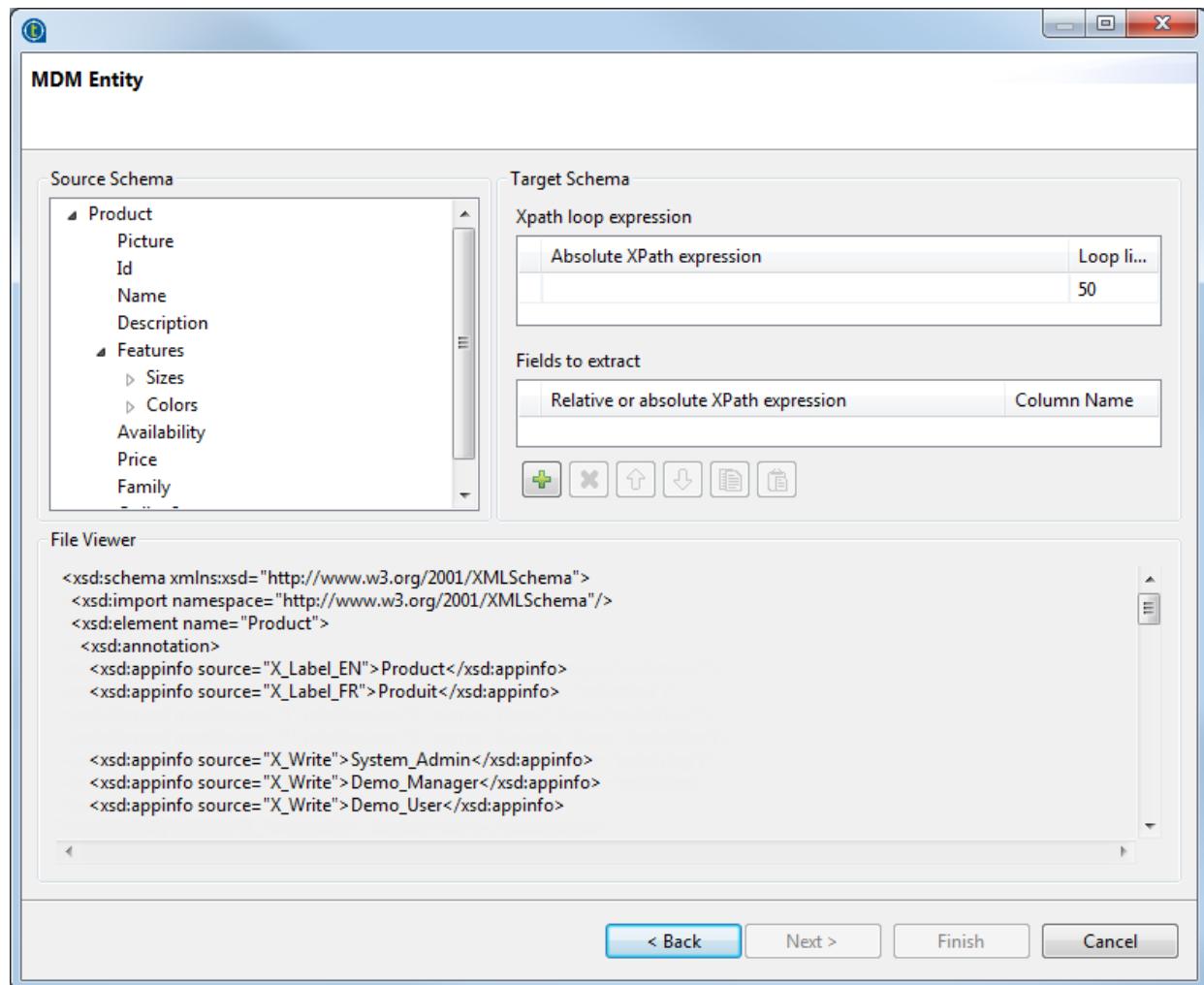


**Note:** You can enter any text in this field, although you would likely put the name of the entity according to which you want to receive the XML schema.

4. 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.

## Example



The schema dialog box is divided into four different panels as follows:

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.

5. 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.

**Note:** The **Xpath loop expression** field is compulsory.

6. If required, define a **Loop limit** to restrict the iteration to one or more nodes.

## Example

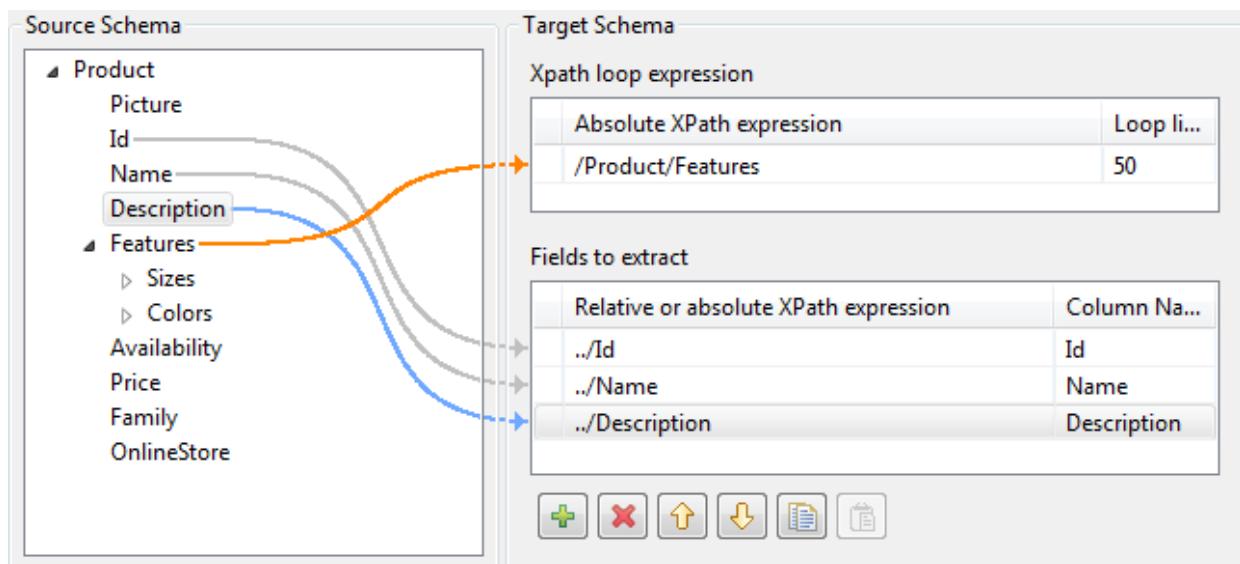
The screenshot shows the configuration of a Talend MDM connection. On the left, the **Source Schema** panel lists fields for a **Product** entity, including Picture, Id, Name, Description, Features, Sizes, Colors, Availability, Price, Family, and OnlineStore. The **Features** field is highlighted with an orange arrow pointing to the **Xpath loop expression** in the Target Schema panel. The **Target Schema** panel contains an **Xpath loop expression** table with one row: **Absolute XPath expression** /Product/Features and **Loop li...** 50. Below it is a **Fields to extract** section with a table and a set of icons for managing rows.

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>
        ...
        <Feature>Color blue</Feature>
        <Feature>Thermos</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.

- To define the fields to receive, drop the relevant node from the source schema to the **Relative or absolute XPath expression** field.



**Tip:** 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.

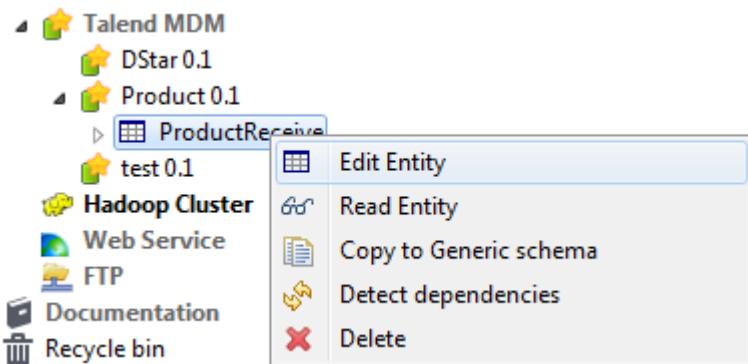
**Tip:** You can prioritize the order of the fields you want to receive by selecting the field and using the up and down arrows. The link of the selected field is blue, and all other links are grey.

- Click **Finish** to validate your modifications and close the dialog box.

## Results

The newly created schema is listed under the corresponding MDM connection in the **Repository** tree view. You can modify the created schema according to your needs and drop the connection as a **tMDMReceive** in any of your Jobs.

For more information on how to modify the schema, see [Modifying the created schema](#) on page 307.



## 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.

**Note:**

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](#) on page 318.

To create an advanced schema, see [Setting up an advanced schema](#) on page 323.

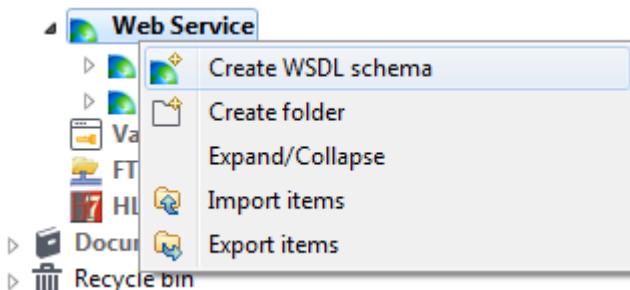
## Setting up a simple schema

This section describes how to define a simple Web Service schema (Simple WSDL).

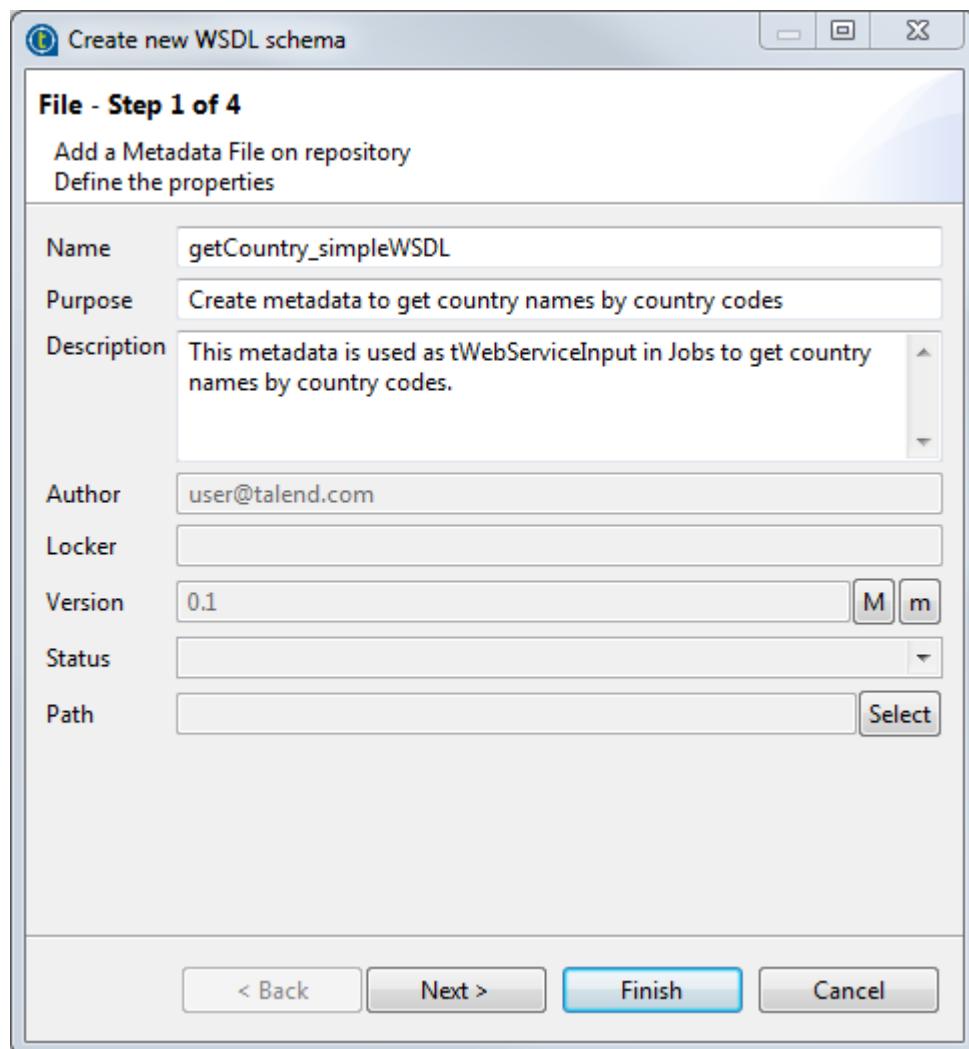
### Defining general properties of the simple Web Service schema

#### Procedure

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.

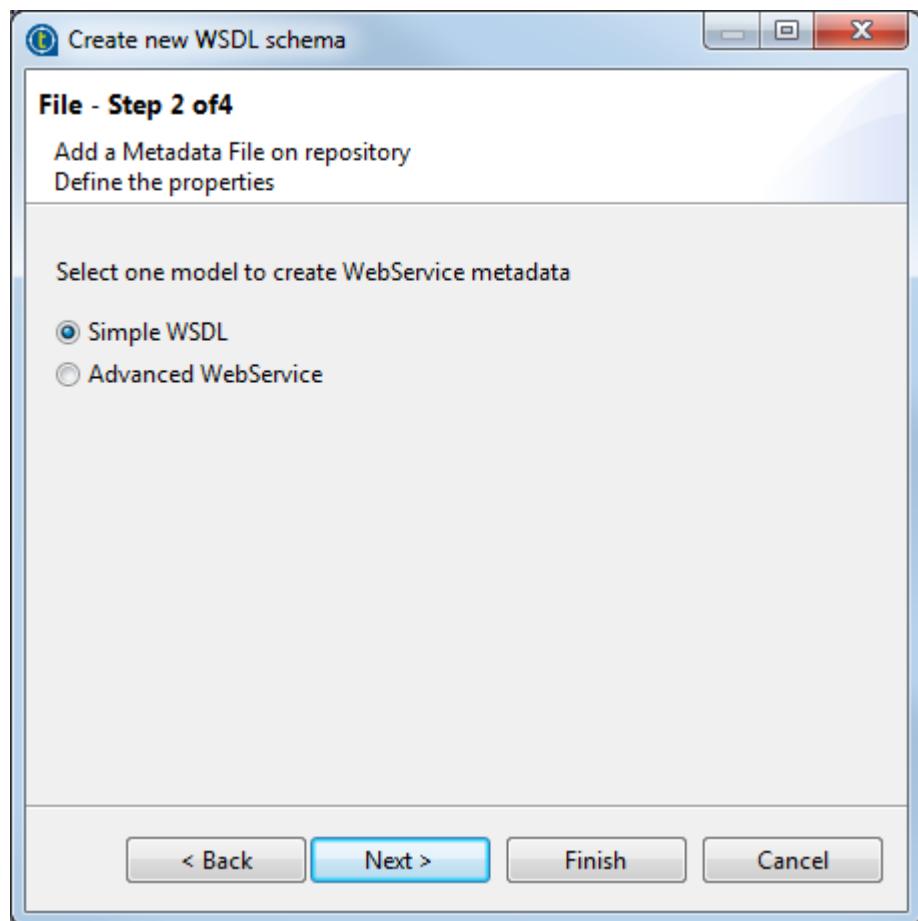
#### Selecting the type of schema (Simple)

##### About this task

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.

##### Procedure

1. In the dialog box, select the **Simple WSDL** option.

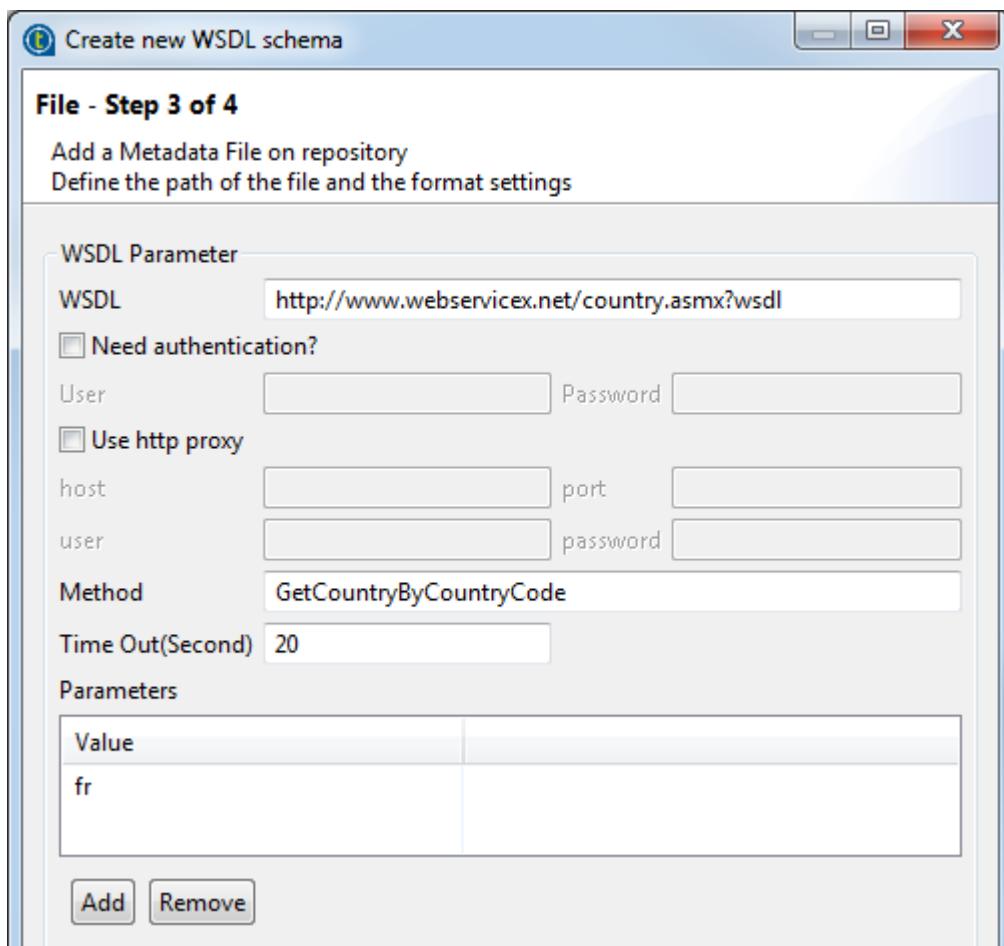


2. Click **Next** to continue.

### Specifying the URI and method

#### About this task

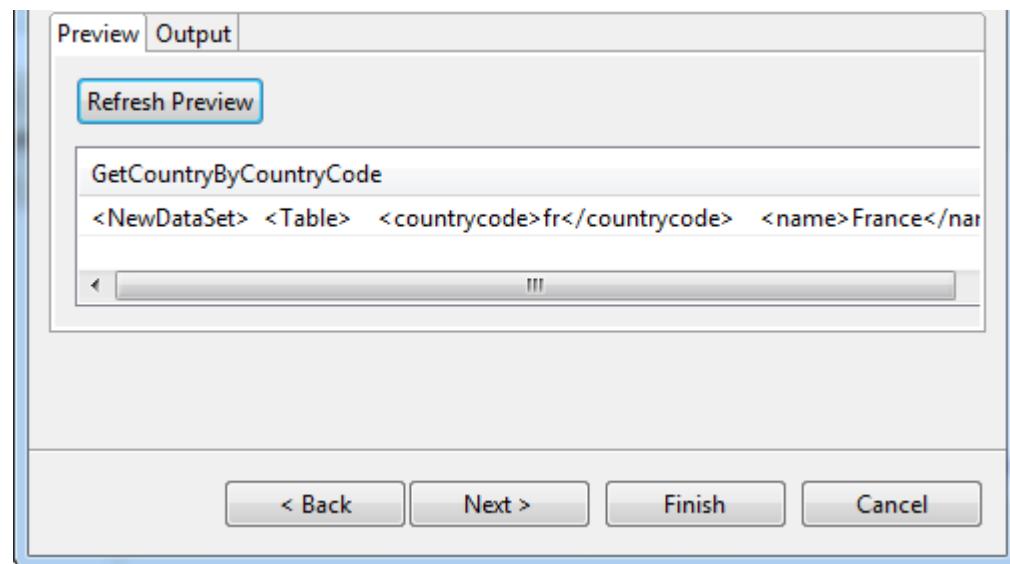
This step involves the definition of the URI and other parameters required to obtain the desired values.



In the **Web Service Parameter** zone:

#### Procedure

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.

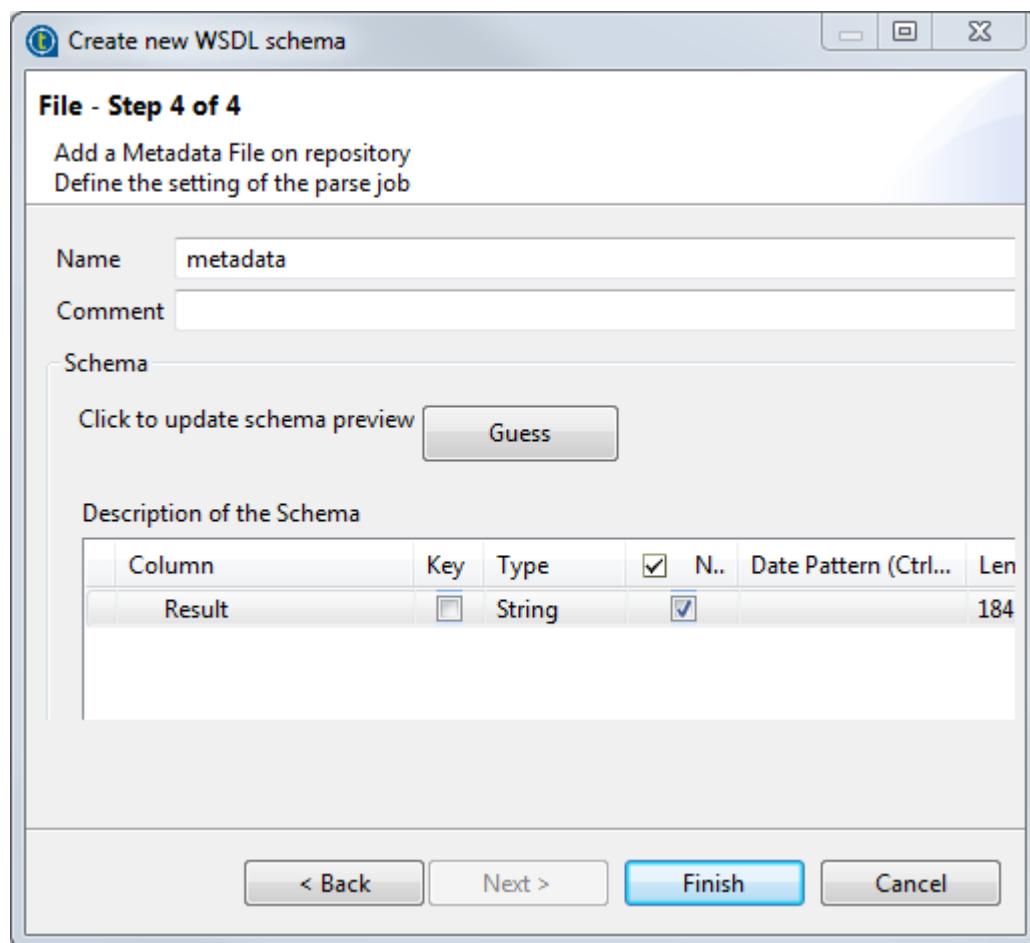


In the **Preview** tab, the values to be transmitted by the Web Service method are displayed, based on the parameters entered.

## Finalizing the end schema (Simple WSDL)

### About this task

You can modify the schema name (metadata, by default) and modify the schema itself using the tool bar.



## Procedure

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.

## Setting up an advanced schema

This section describes how to define an **Advanced WebService** schema.

Next, you need to define the input and output schemas and schema-parameter mappings in the **Input mapping** and **Output mapping** tabs.

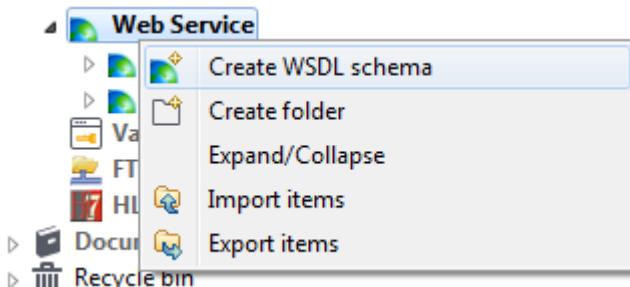
**Note:**

Depending on the type of the output, you can choose to normalize or denormalize the results by clicking the **Normalize** and **Denormalize** buttons.

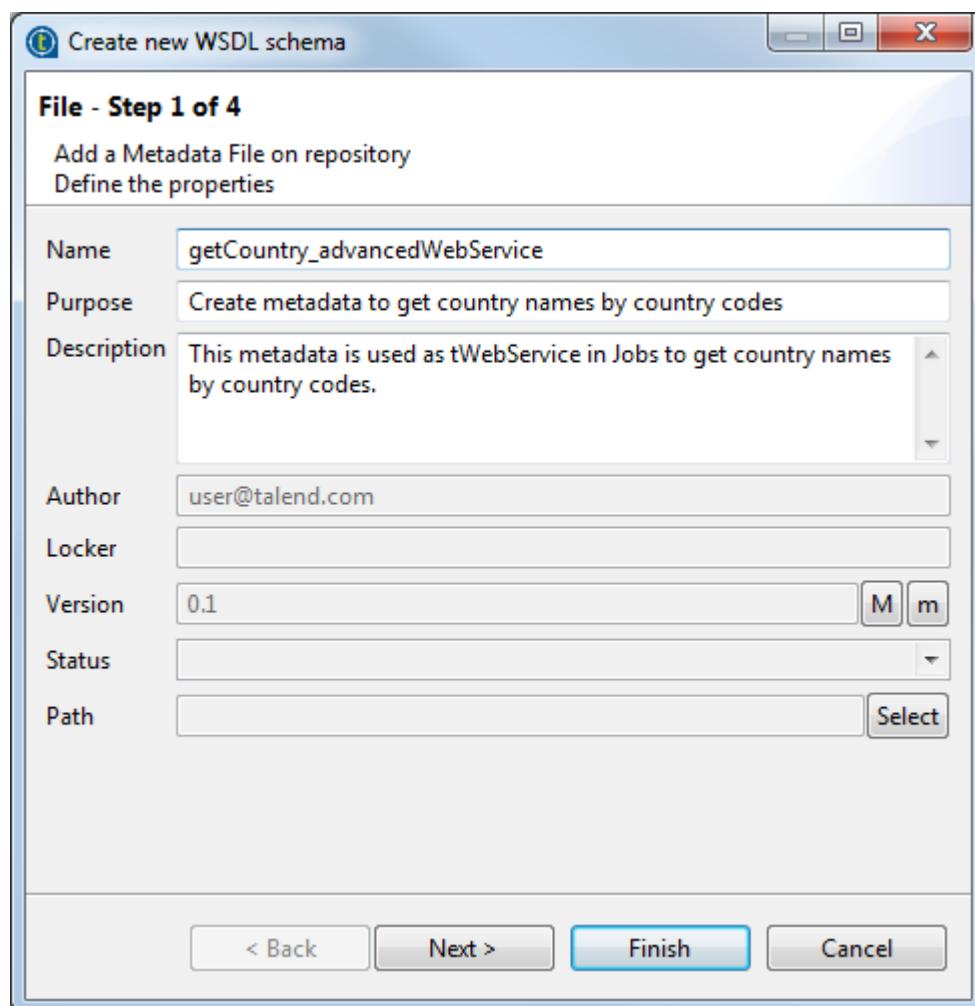
## Defining general properties of the advanced Web Service schema

### Procedure

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**.



- Click **Next** to select the schema type in step 2.

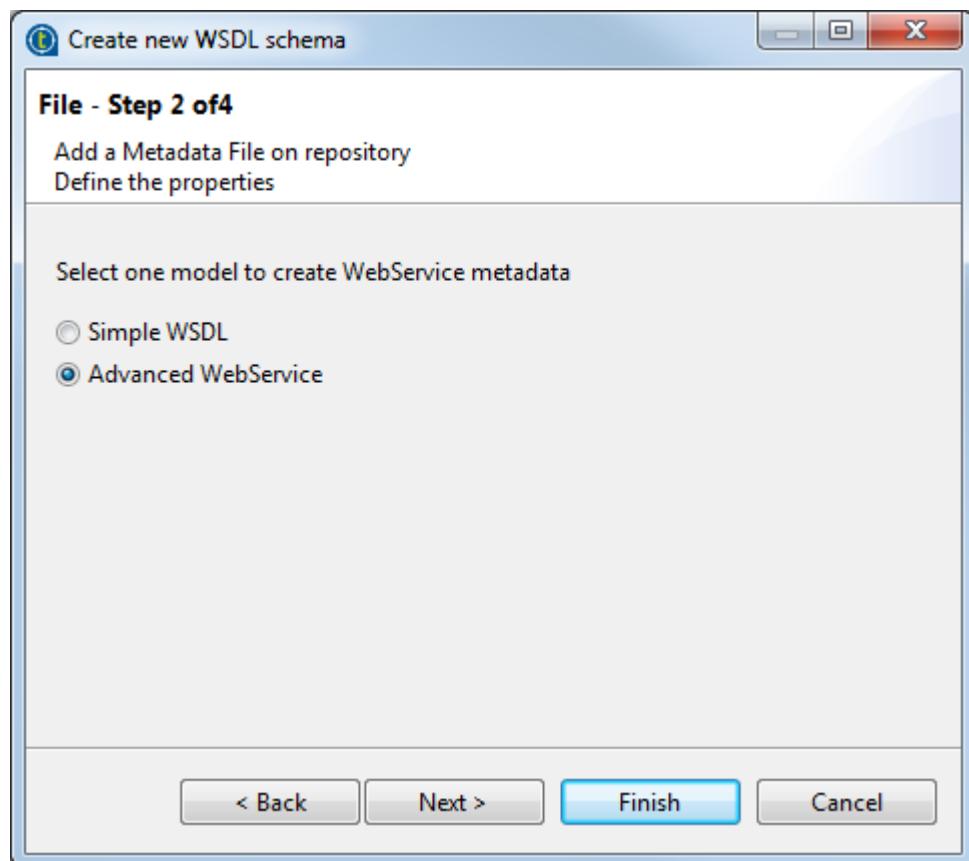
#### Selecting the type of schema (Advanced)

##### About this task

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.

##### Procedure

- In the dialog box, select the **Advanced WebService** option.

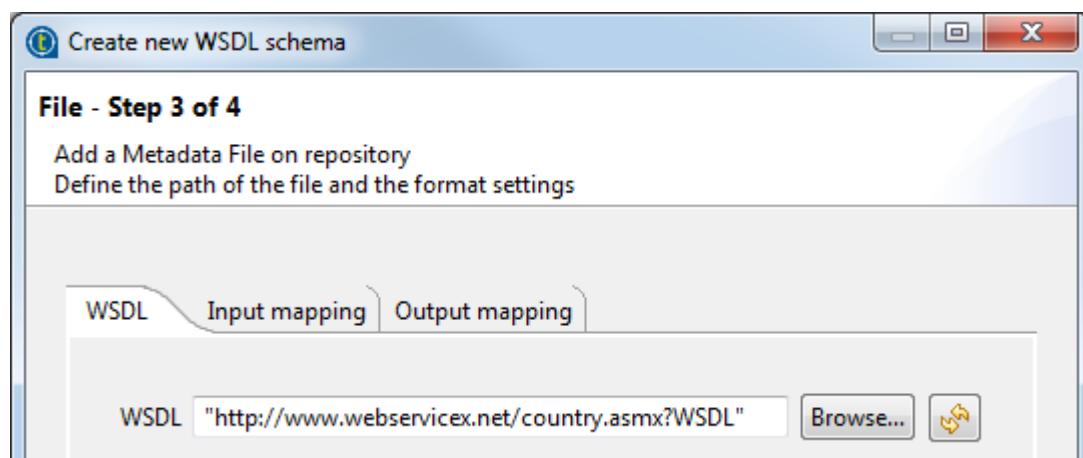


- Click **Next** to define precise Web Service parameters.

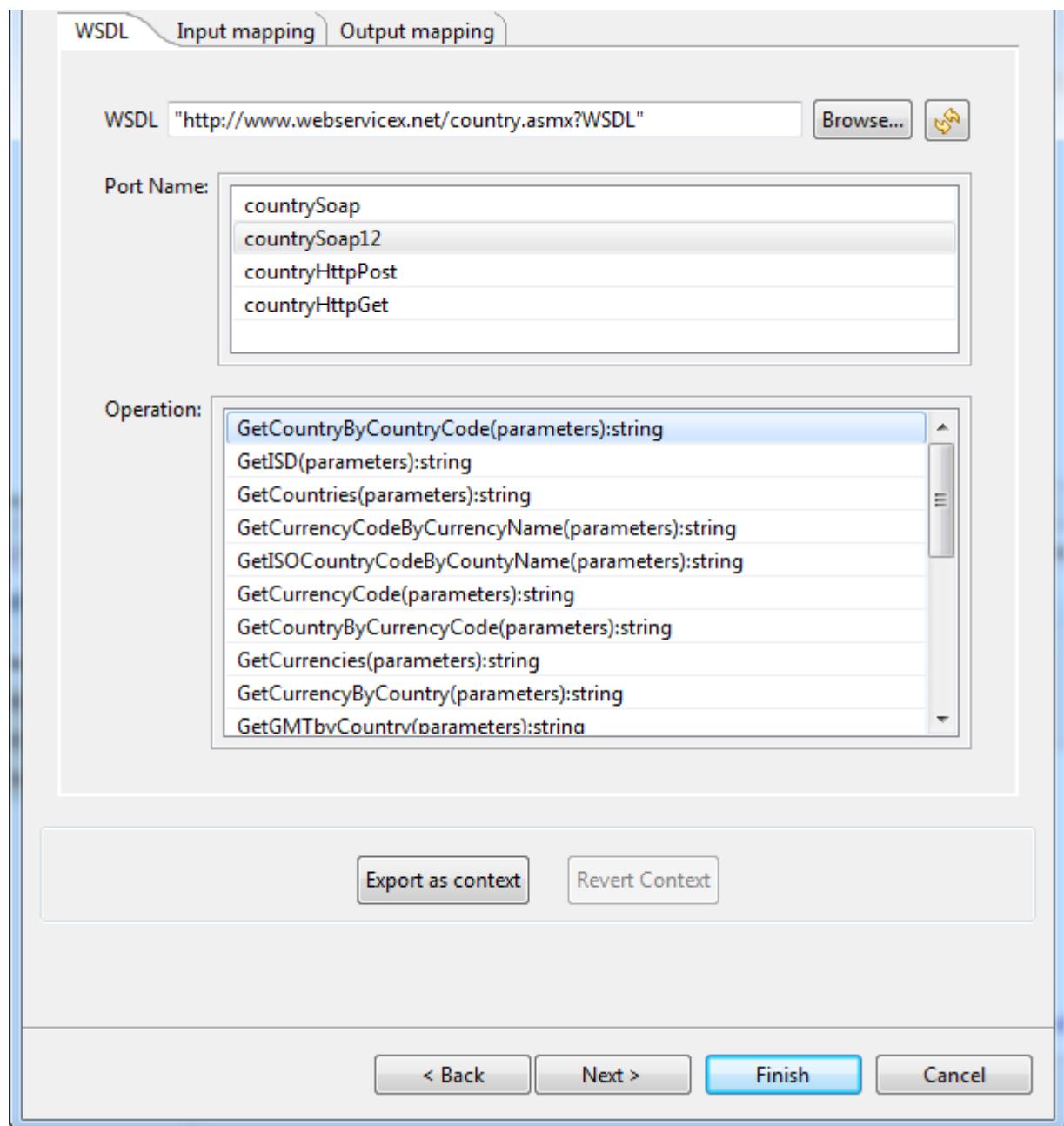
### Defining the port name and operation

#### Procedure

- 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.



- 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.

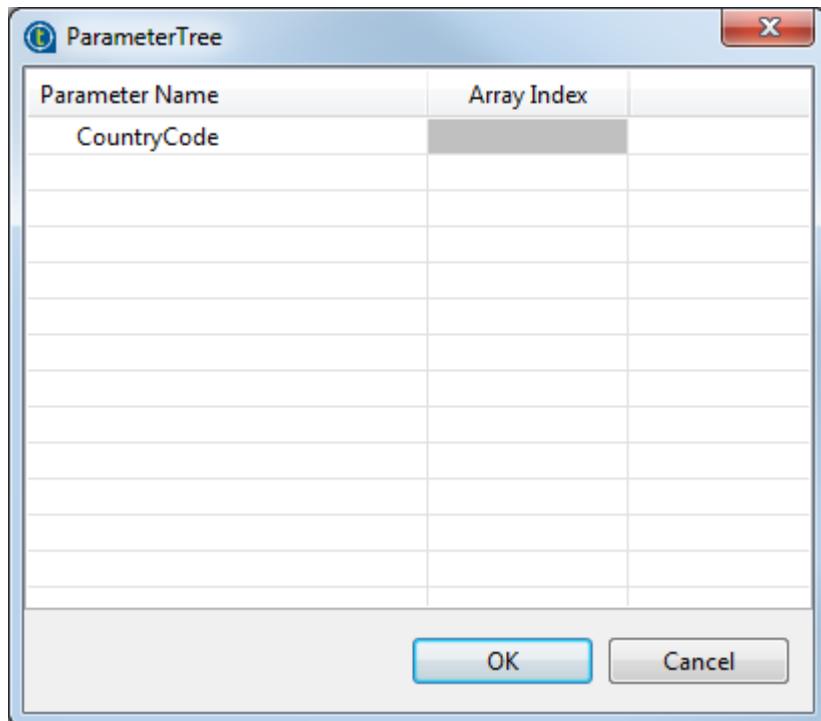
### Defining the input schemas and mappings

#### About this task

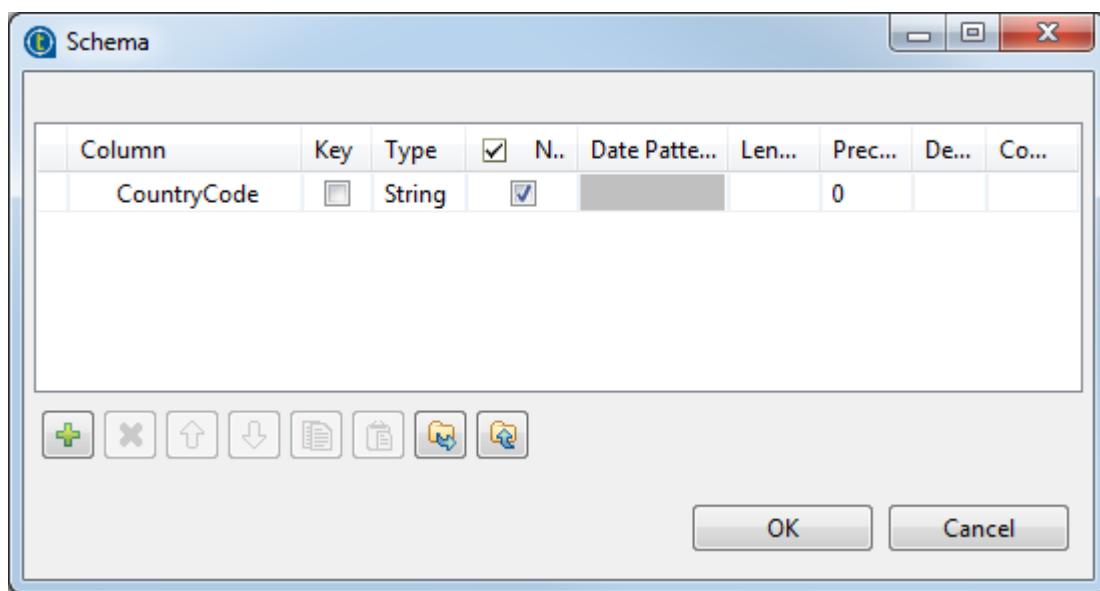
To define the input schema and mappings, do the following:

#### Procedure

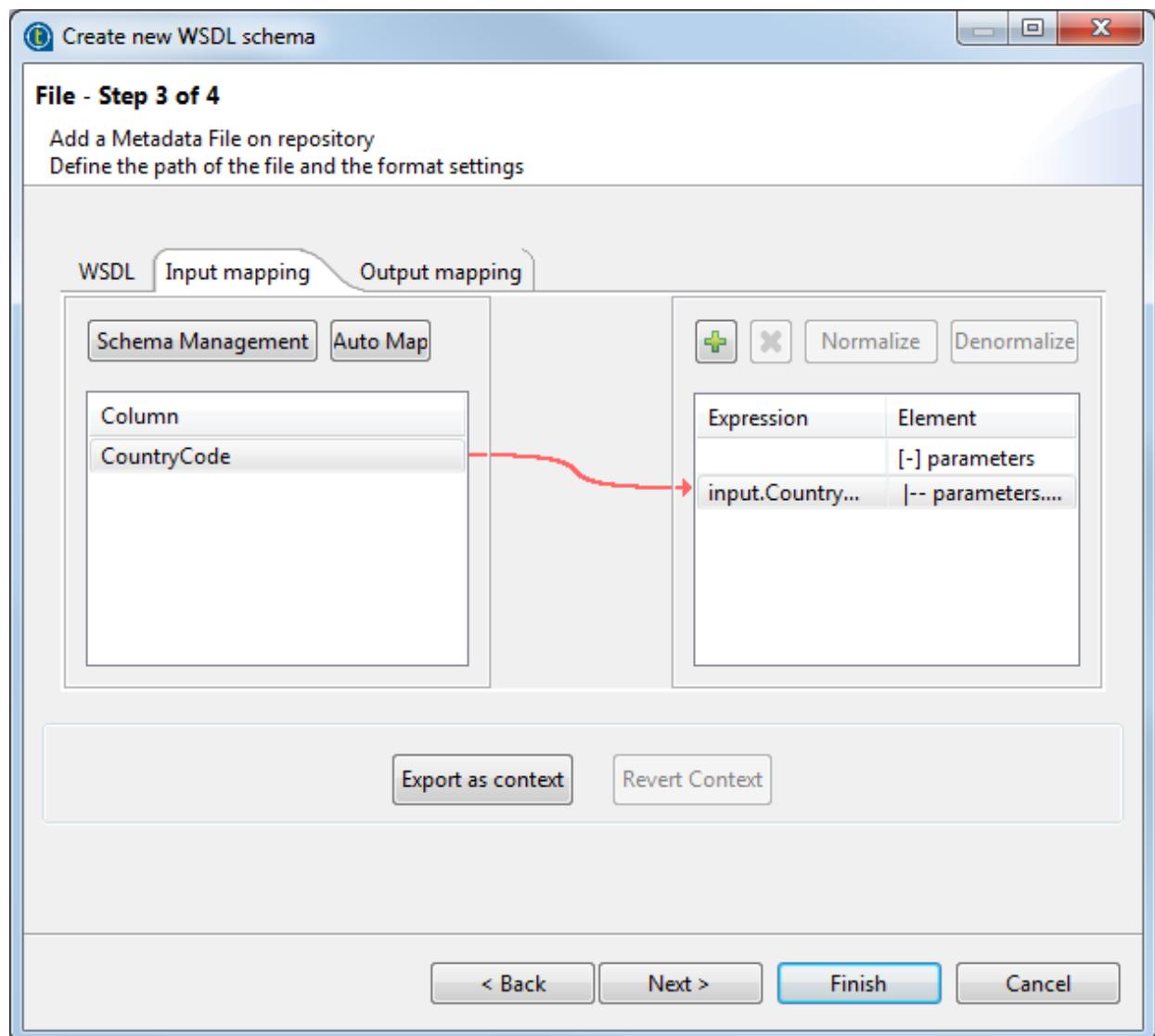
1. Click the **Input mapping** tab to define the input schema and set the parameters required to execute the operation.
2. In the table to the right, select the **parameters** row and click the **[+]** button to open the **ParameterTree** dialog box.



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.



#### Note:

If available, use the **Auto Map** button situated to the top of the tab, to carry out the mapping automatically.

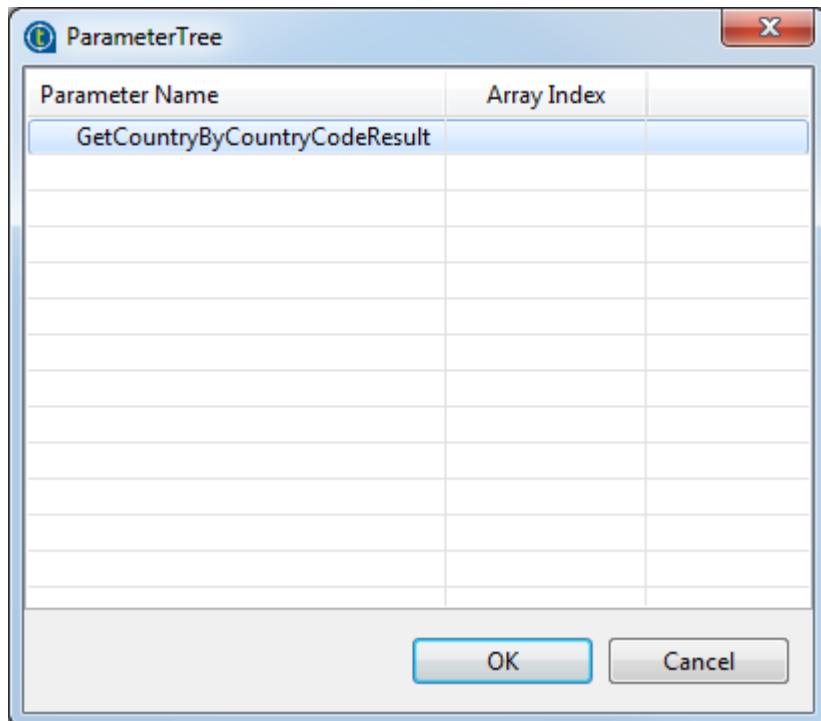
## Defining the output schemas and mappings

### About this task

To define the output schema and mappings, proceed as follows:

### Procedure

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.

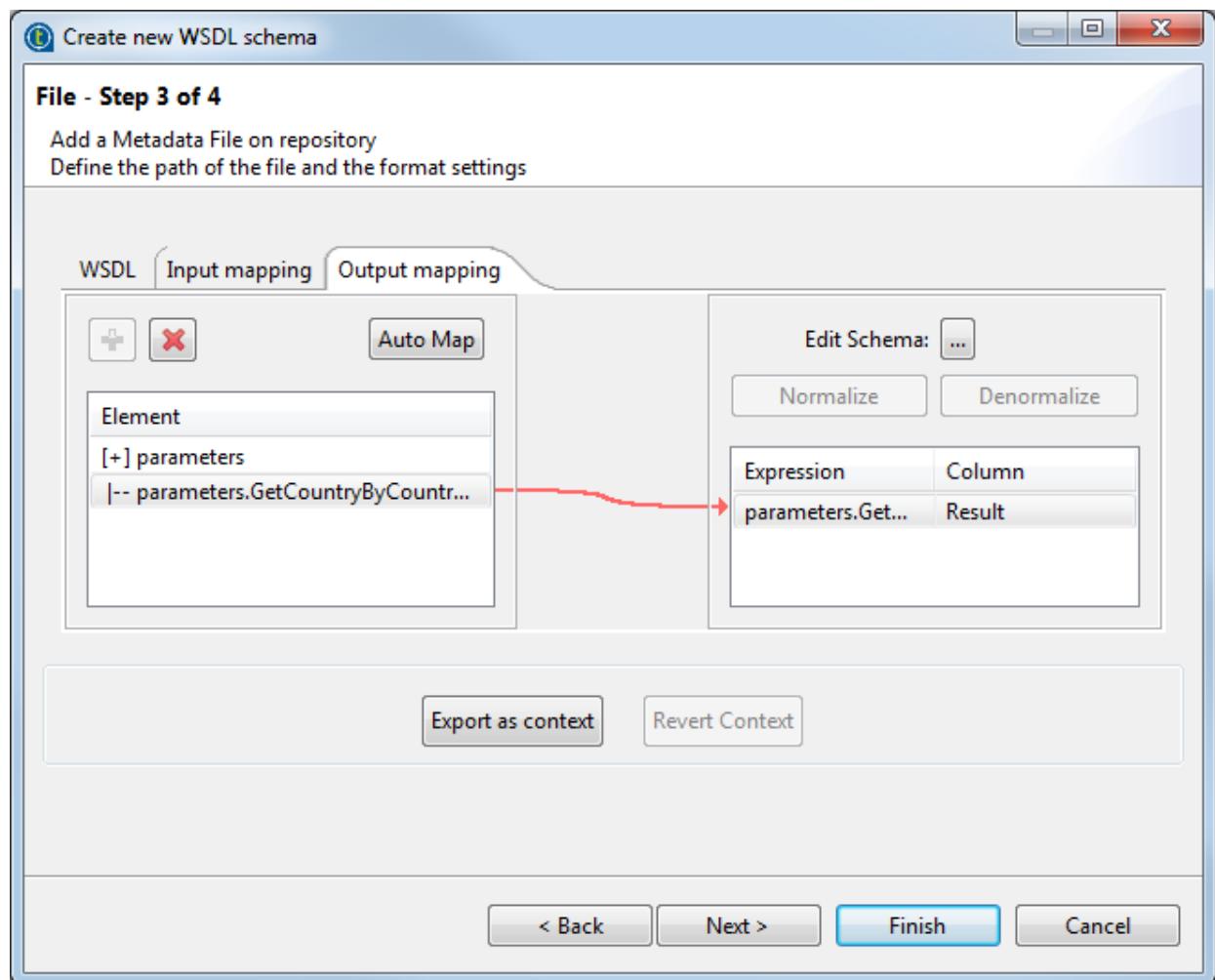
5. Define the output schema.

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.

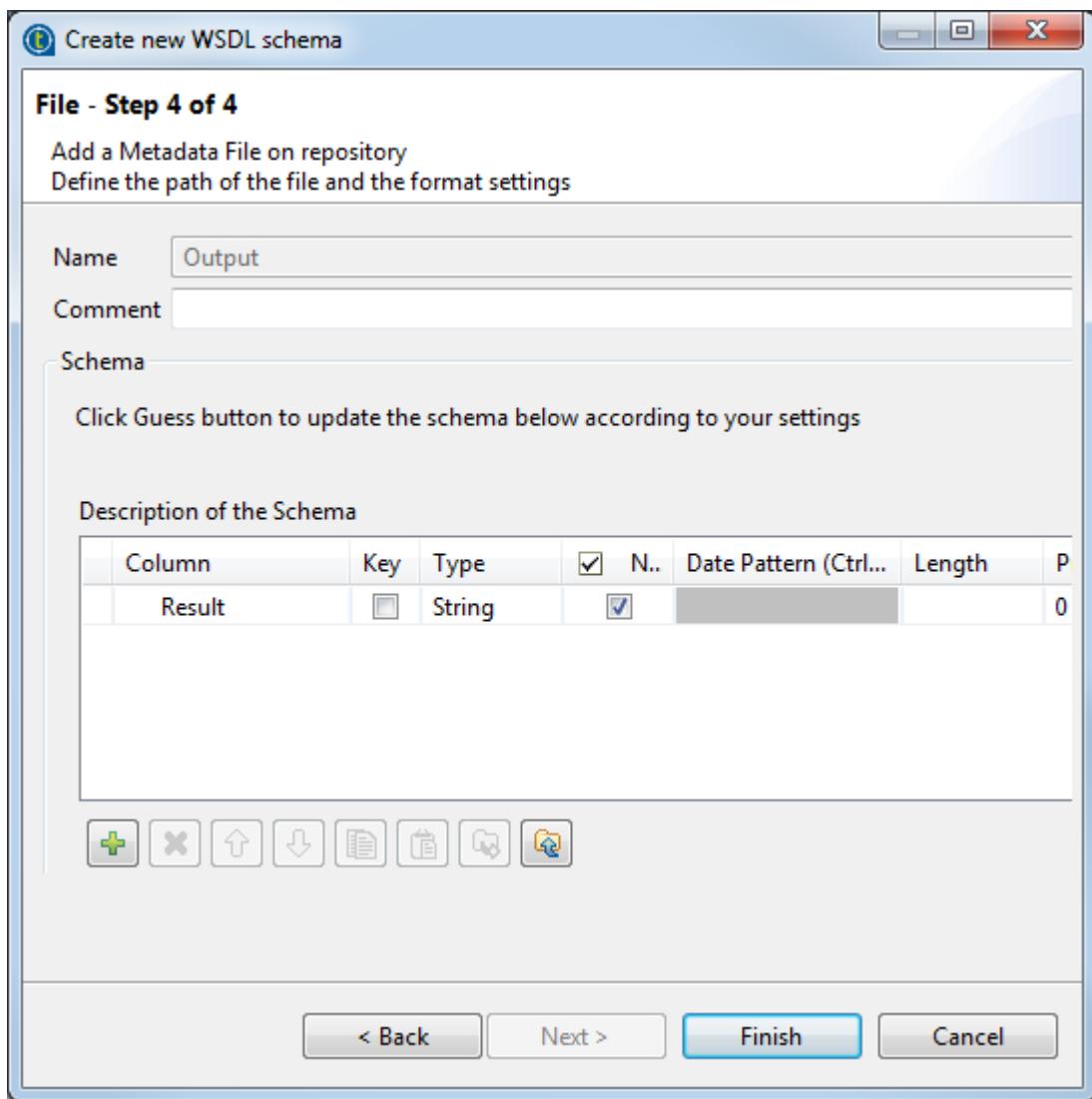


8. Click **Next** to finalize the schema.

### Finalizing the end schema (Advanced WebService)

#### About this task

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:

#### Procedure

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.

## 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.

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 [Using centralized metadata in a Job](#) on page 344.

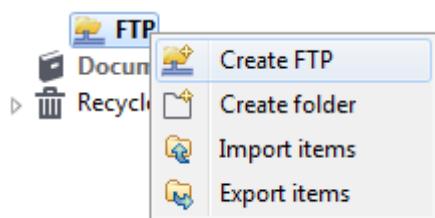
## Defining the general properties of the connection FTP

### About this task

To create a connection to an FTP server, follow the steps below:

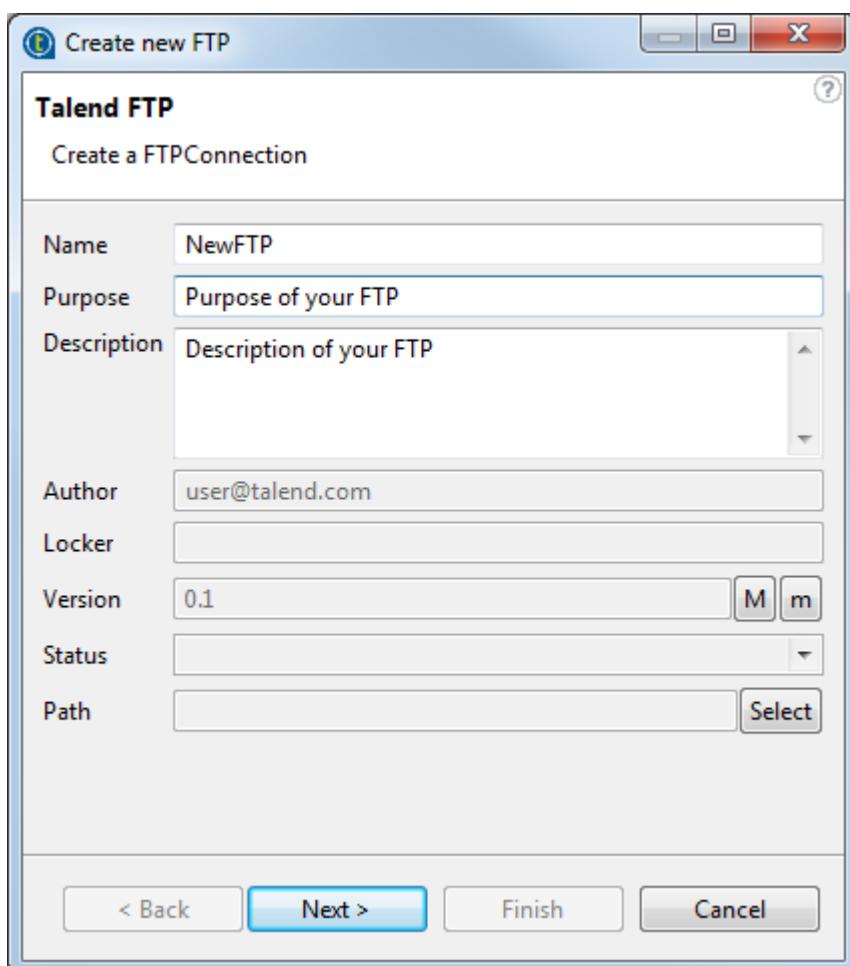
### Procedure

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**.

**Note:**

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 on page 404](#).

- When you have finished, click **Next** to enter the FTP server connection information.

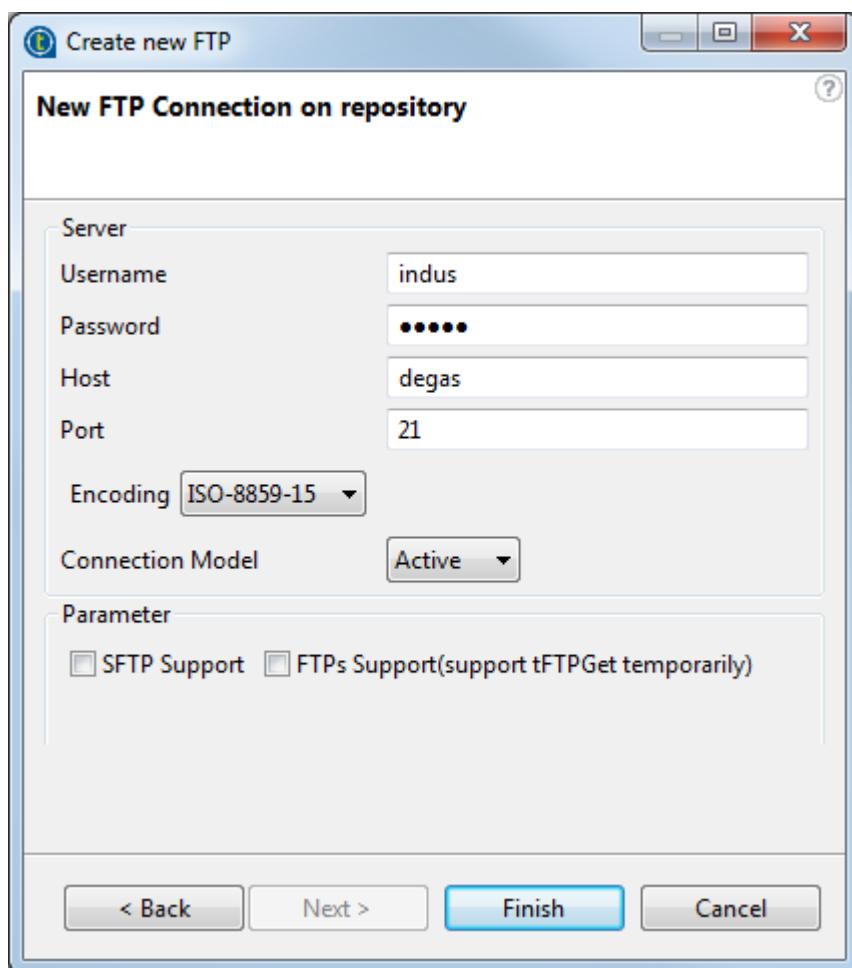
## Connecting to an FTP server

### About this task

In this step we shall define the connection information and parameters.

### Procedure

- Enter your **Username** and **Password** in the corresponding fields.



- In the **Host** field, enter the name of your FTP server host.
- Enter the **Port** number in the corresponding field.
- Select the **Encoding** type from the list.
- 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.

## Working with Hierarchical Mapper

Talend Studio enables you to access structures, maps and namespace containers created in the **Mapping** perspective by expanding the **Hierarchical Mapper** node, which is located under the **Metadata** folder in the **Repository** tree view in the **Integration** perspective.

For more information on working with these elements, see the Talend Data Mapper User Guide.

## Exporting metadata as context and reusing context parameters to set up a connection

If the **Export as context** option is available for a metadata connection, 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.

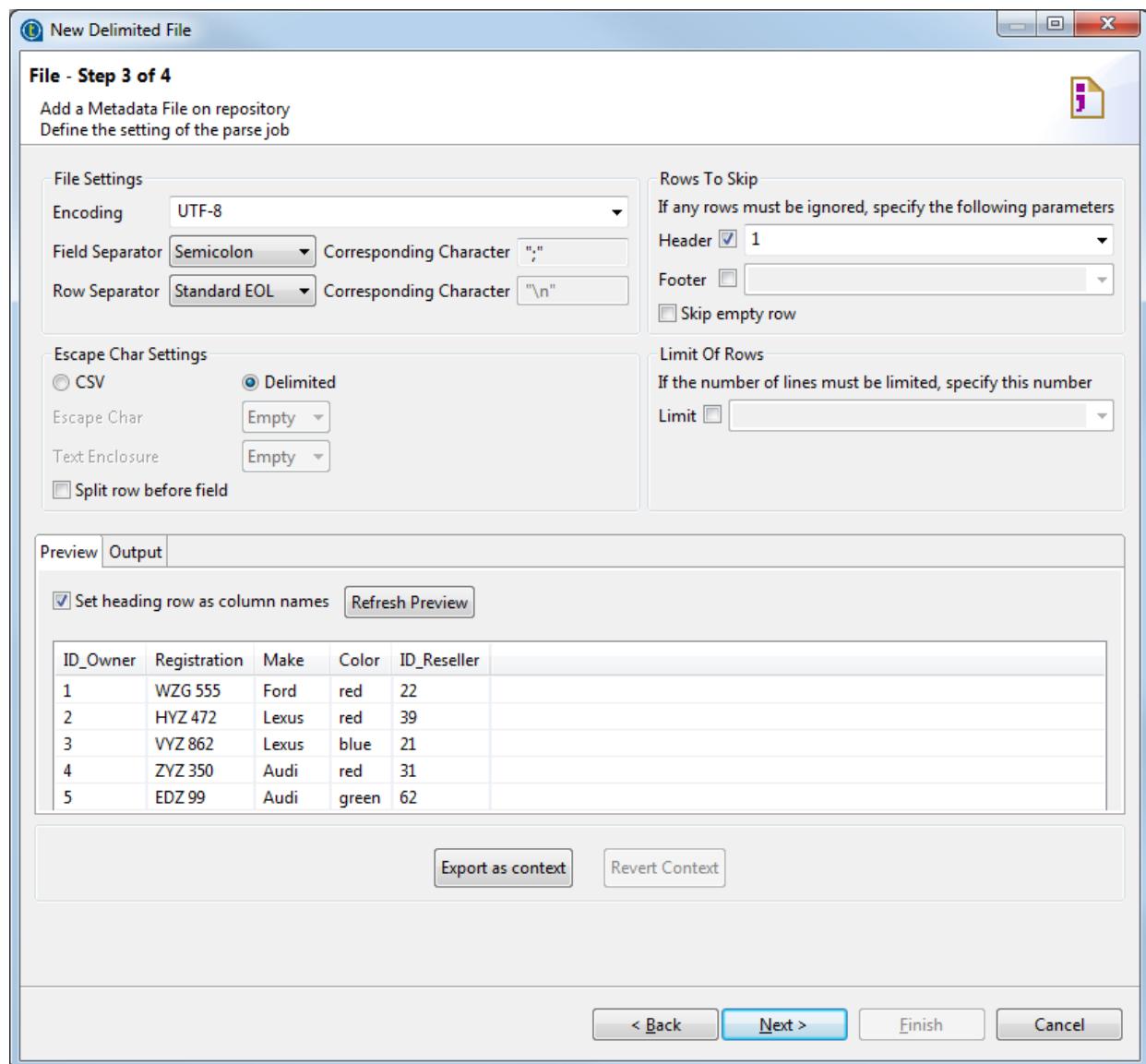
### Exporting connection details as context variables

#### About this task

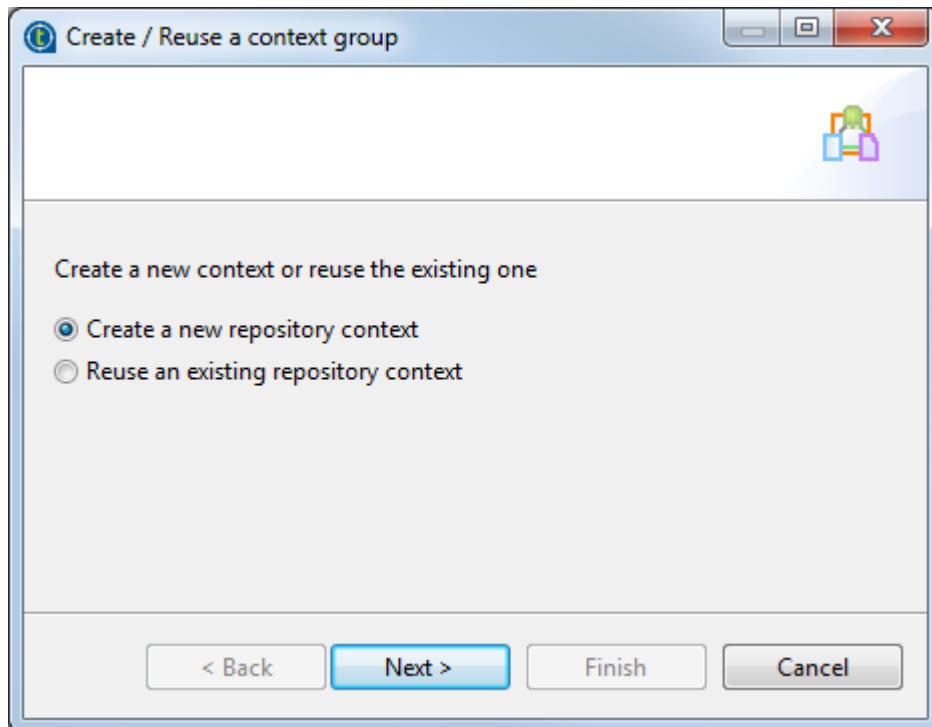
To export connection details as context variables in a new context group in the Repository, follow the steps below:

#### Procedure

1. Upon creating or editing a metadata connection in the wizard, click **Export as context**.

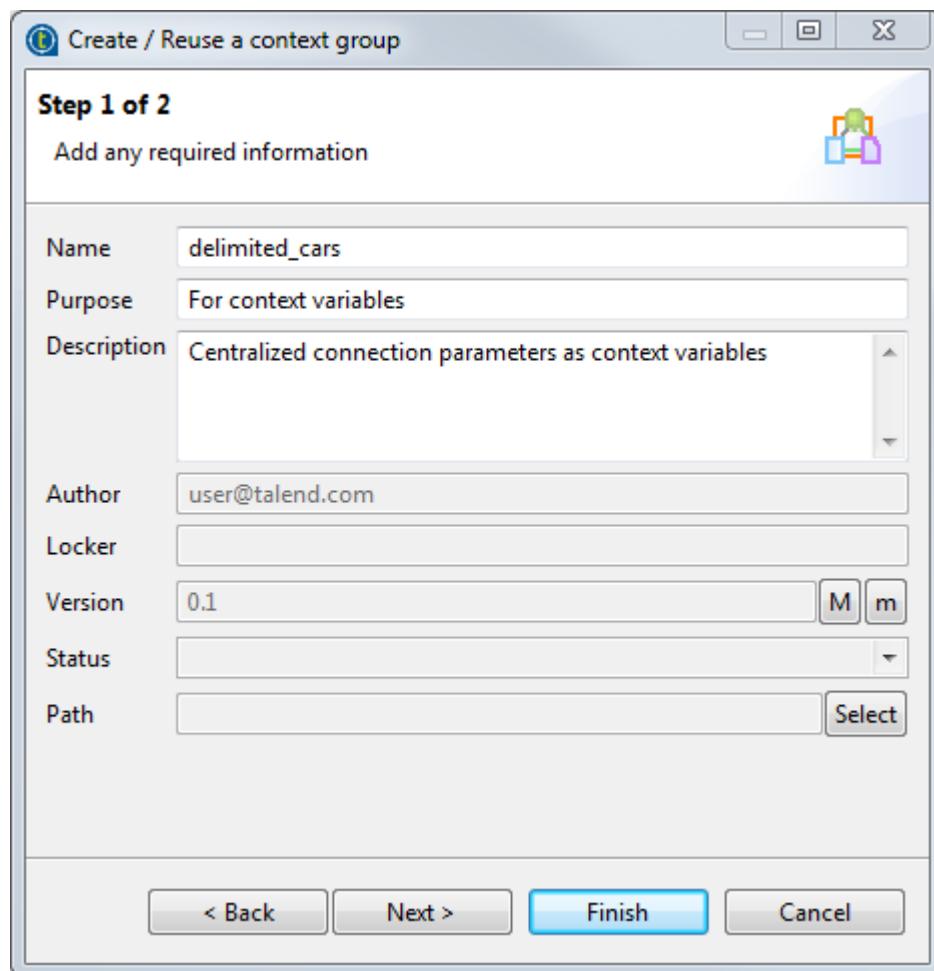


2. In the **Create / Reuse a context group** wizard that opens, select **Create a new repository context** and click **Next**.



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.



4. 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**.

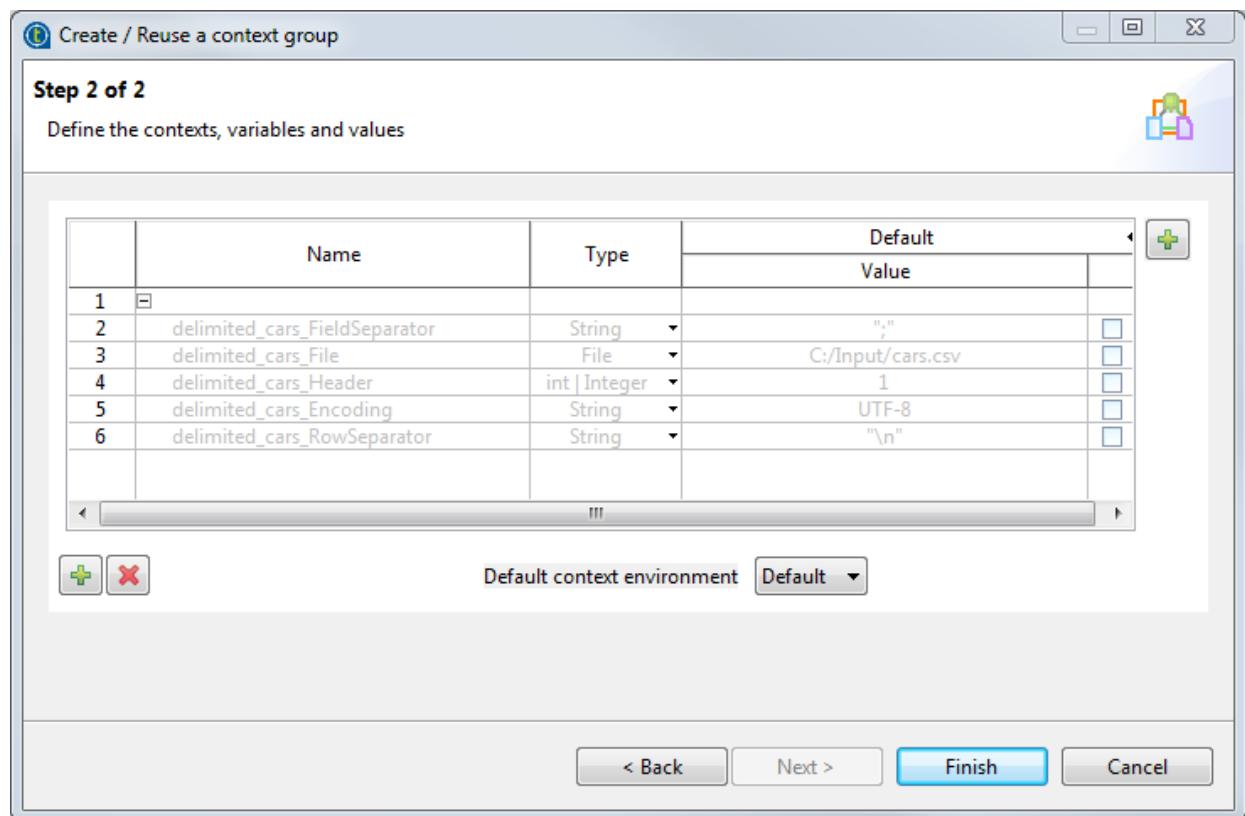
5. Check the context group generation result.

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.

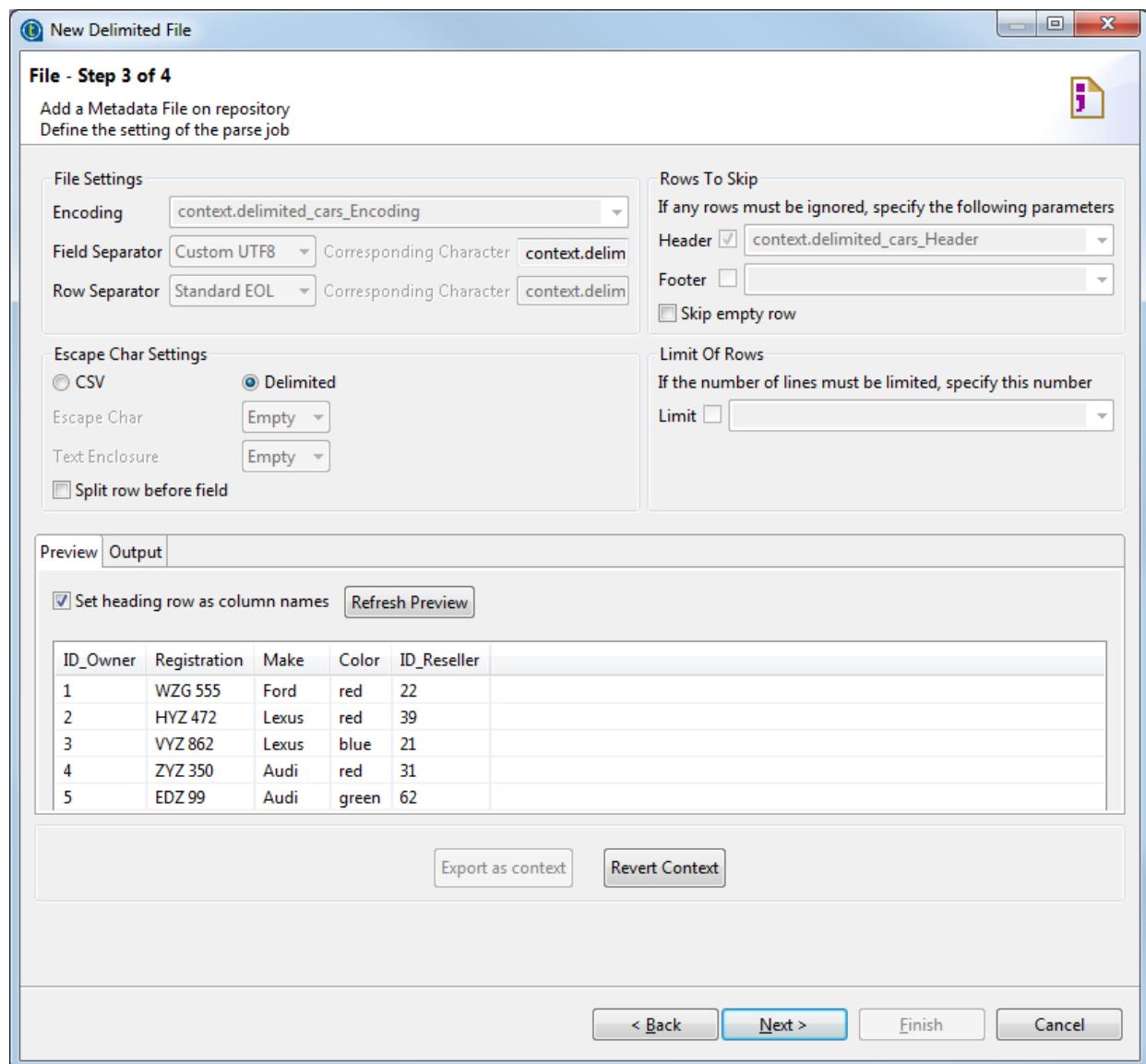
To edit the default context, or add new contexts, click the **[+]** button at the upper right corner of the wizard.

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](#) on page 66.



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.

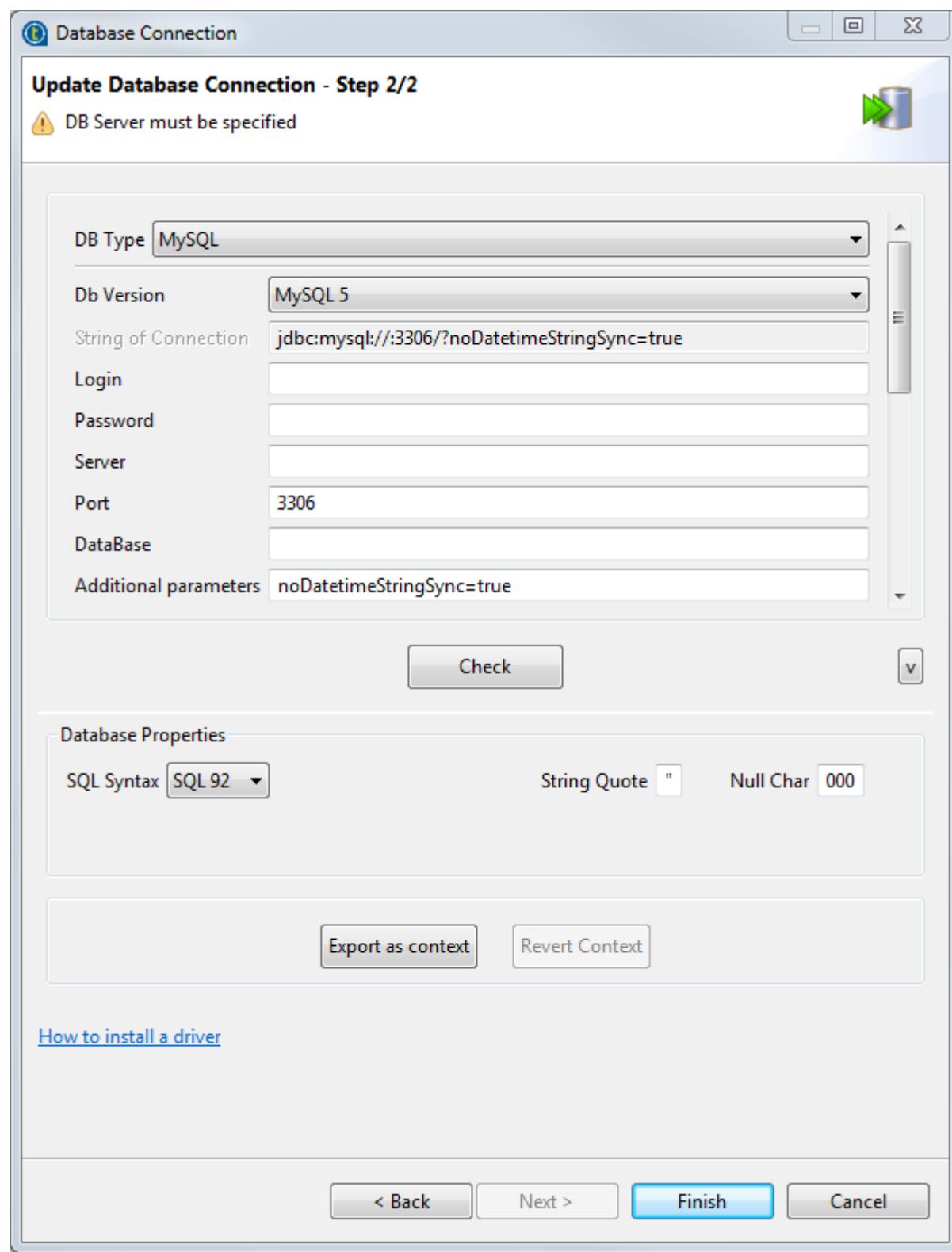
## Using variables of an existing context group to set up a connection

### About this task

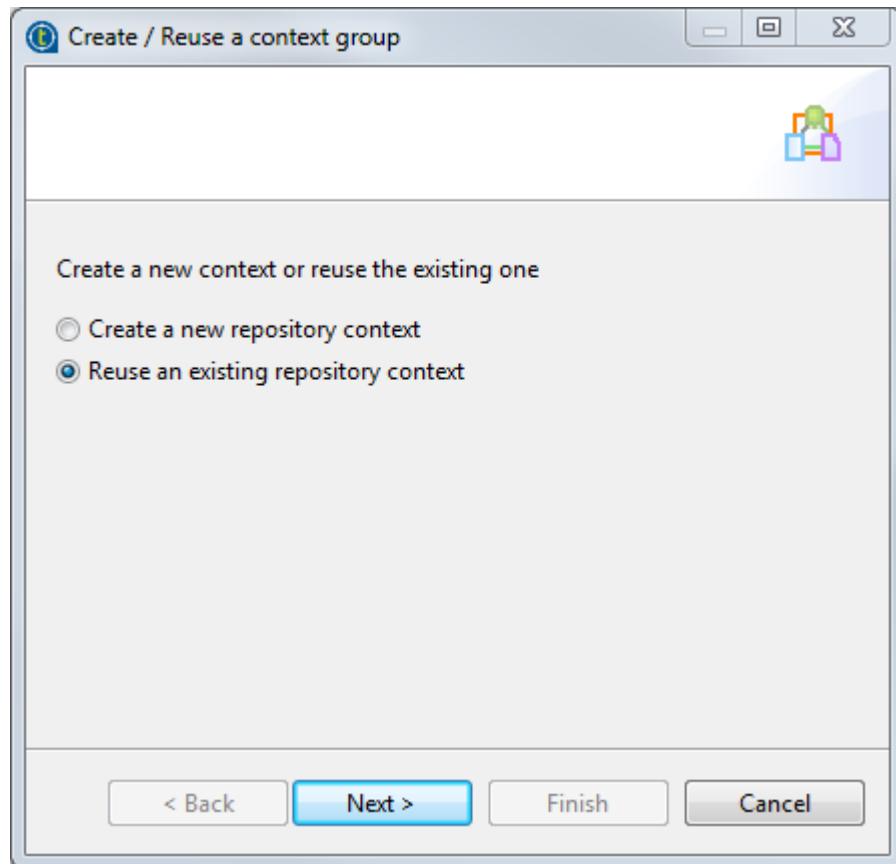
To use variables of an existing context group centrally stored in the Repository to set up a connection, follow the steps below:

### Procedure

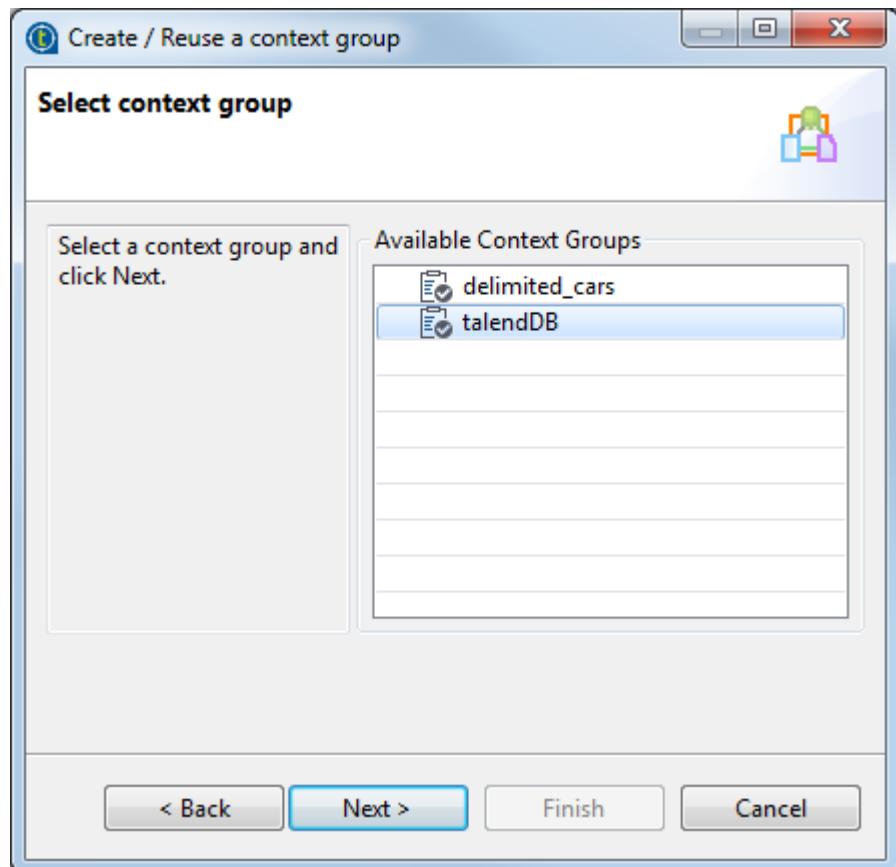
- When creating or editing a metadata connection in the wizard, click **Export as context**.



2. In the **Create / Reuse a context group** wizard that opens, select **Reuse an existing repository context** and click **Next**.

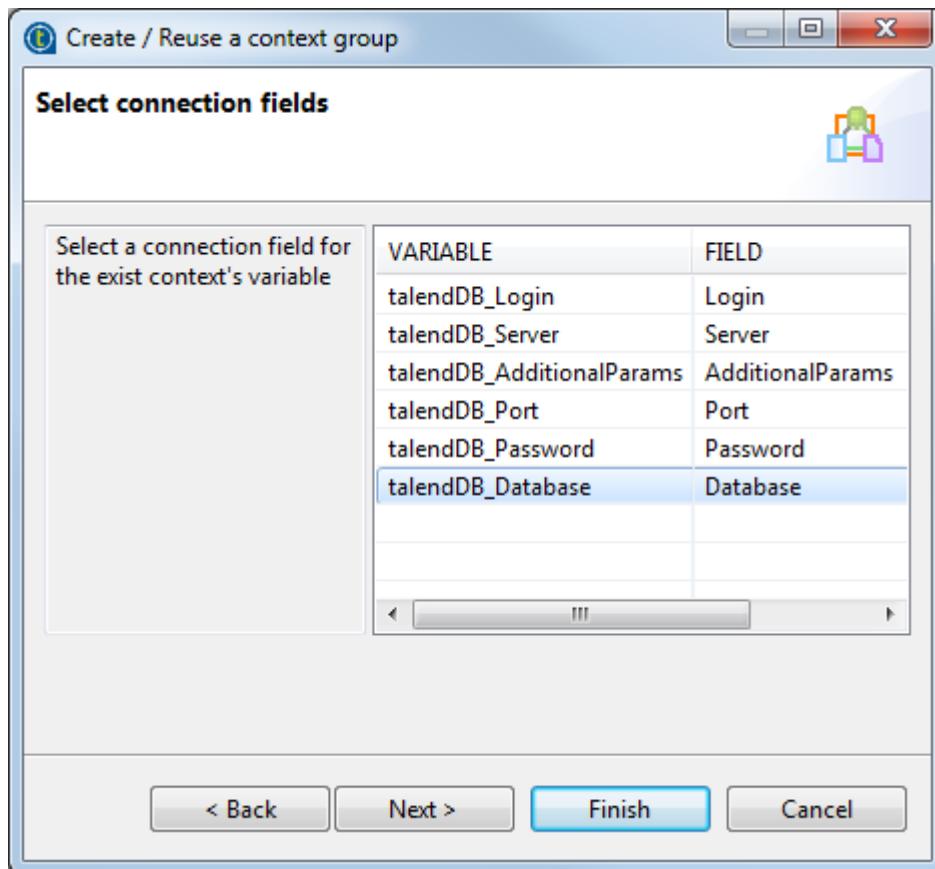


3. Select a context group from the list and click **Next**.



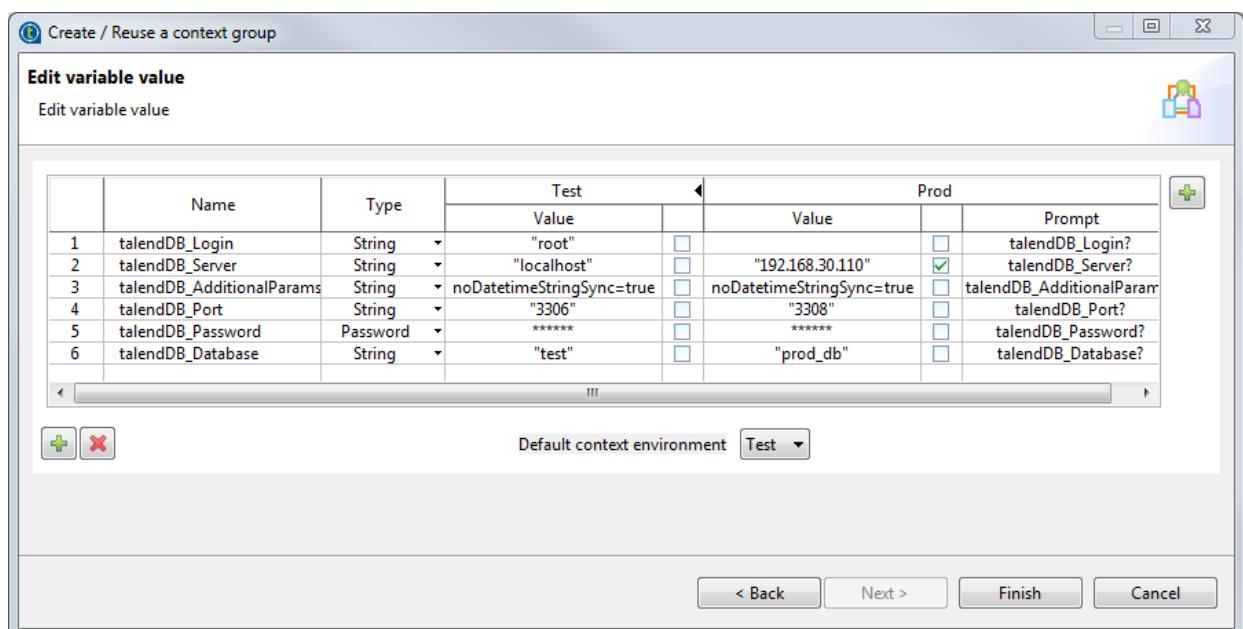
4. For each variable, select the corresponding field of the connection details, and then click **Next** to view and edit the context variables, or click **Finish** to show the connection setup result directly.

In this example, click **Next**.

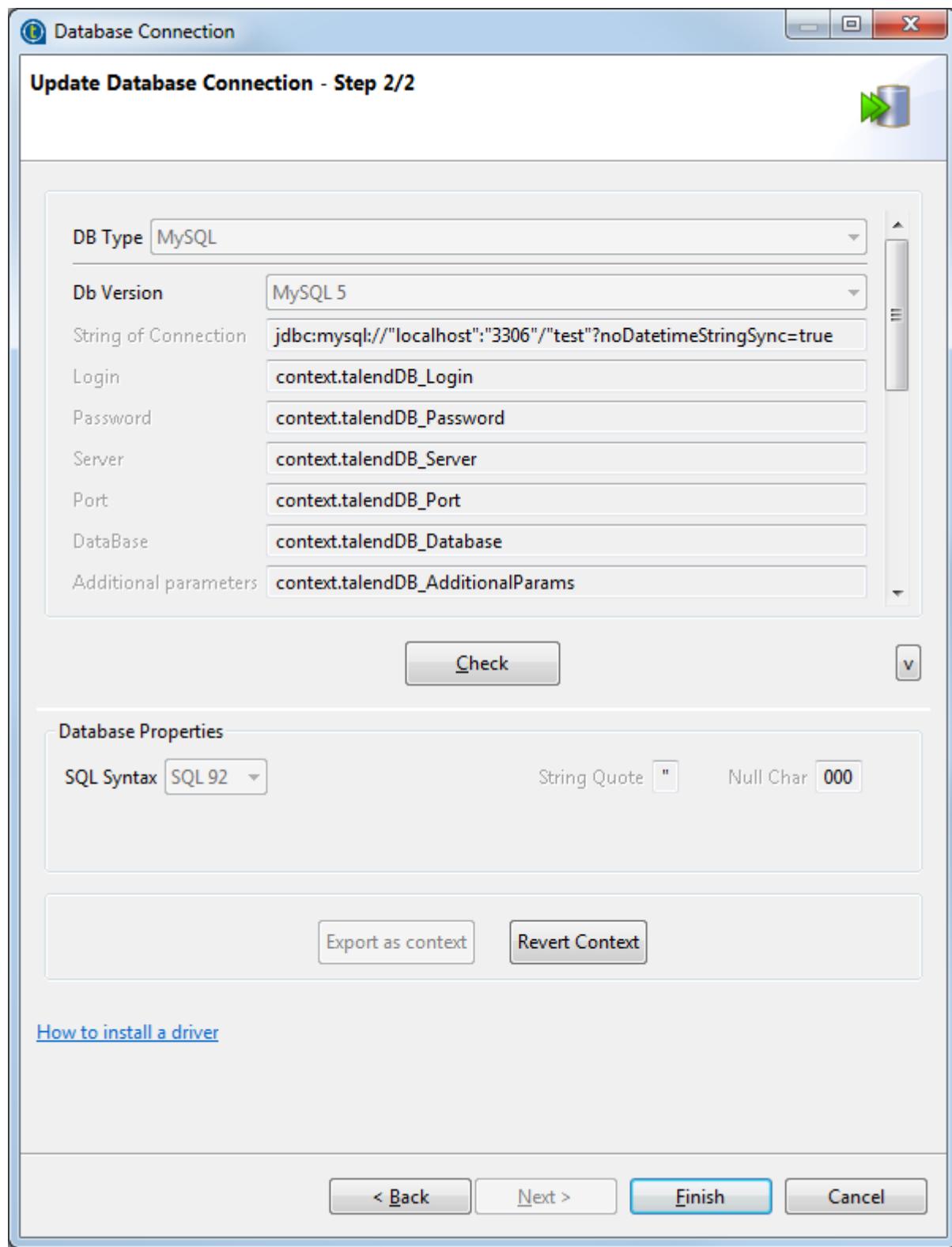


5. Edit the contexts and/or context variables if needed. If you make any changes, your centralized context group will be updated automatically.

For more information on handling contexts and variables, see [Using contexts and variables](#) on page 66.



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.

# Using centralized metadata in a Job

## About this task

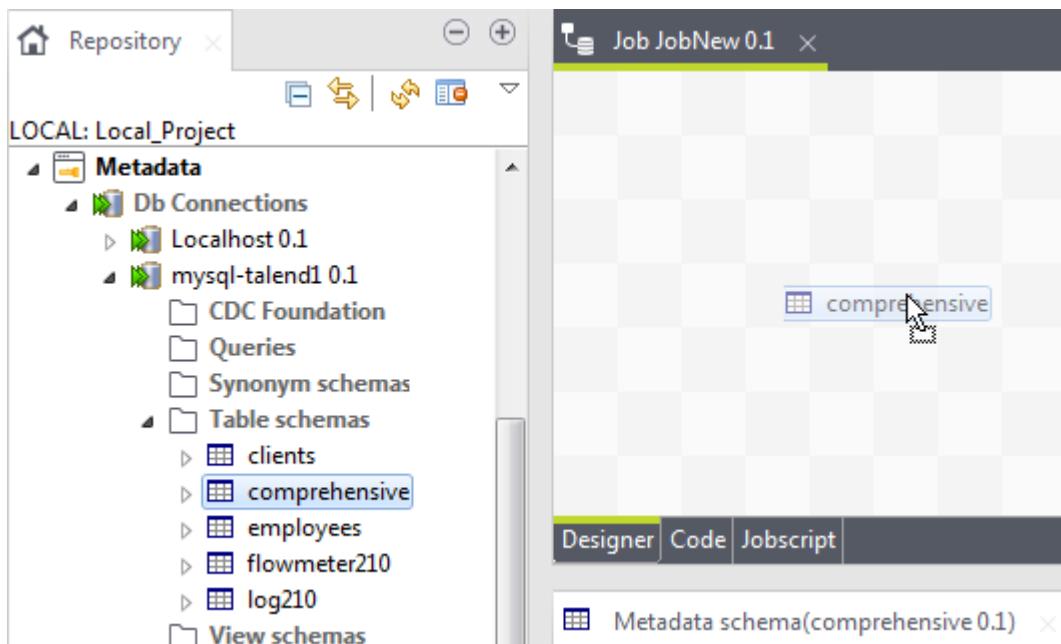
For recurrent use of files and database connections in various Jobs, we recommend you to store the connection and schema metadata in the **Repository** tree view under the **Metadata** node. Different folders under the **Metadata** node will group the established connections including those to databases, files and systems.

Different wizards will help you centralize connection and schema metadata in the **Repository** tree view.

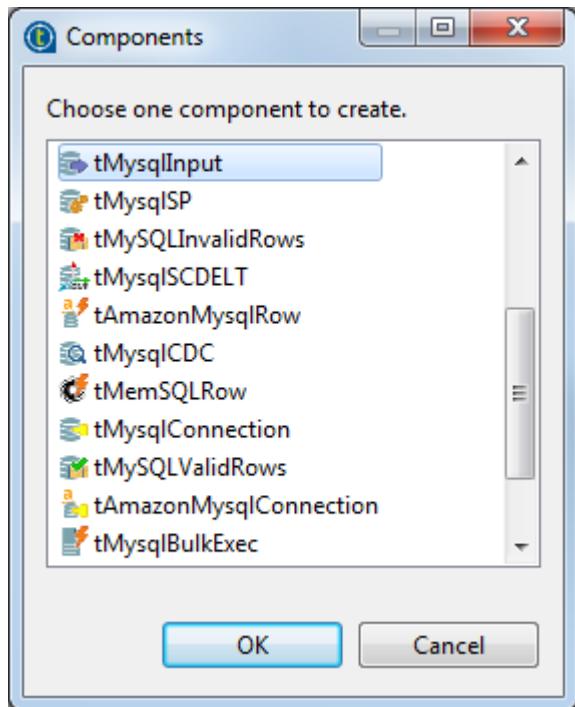
Once the relevant metadata is stored under the **Metadata** node, you will be able to drop the corresponding components directly onto the design workspace.

## Procedure

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.

## Results

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.

### Note:

If you select the connection without selecting a schema, then the properties will be filled with the first encountered schema.

# Using routines

## Managing routines

### 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.

**Note:** 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](#) on page 346.

For further information about how to create user routines, see [Creating user routines](#) on page 348.

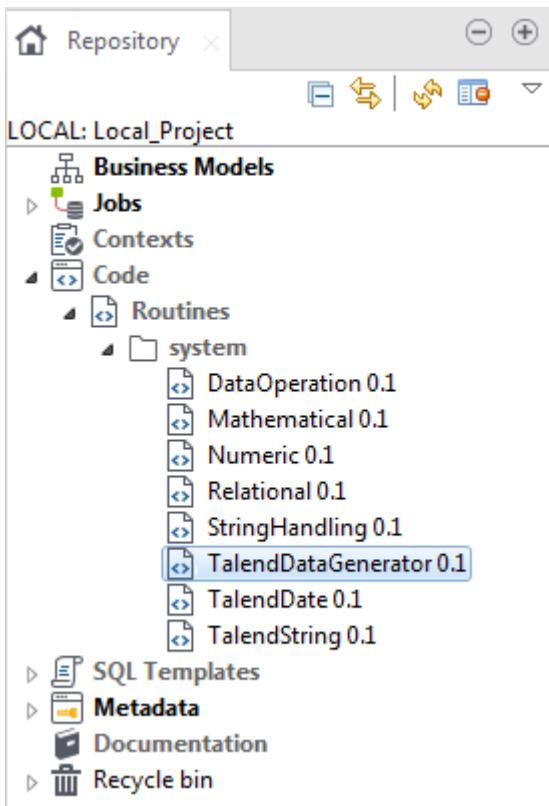
**Note:** You can also set up routine dependencies on Jobs. To do so, simply right click a Job on the **Repository** tree view and select **Set up routine dependencies**. In the dialog box which opens, all routines are set by default. You can use the tool bar to remove routines if required.

## Accessing the System Routines

### About this task

To access the system routines, click **Code > Routines > system**. The routines or functions are classed according to their usage.

**Note:** 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:

### Procedure

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.

**Note:** The syntax of routine call statements is case sensitive.

## Customizing the system routines

### About this task

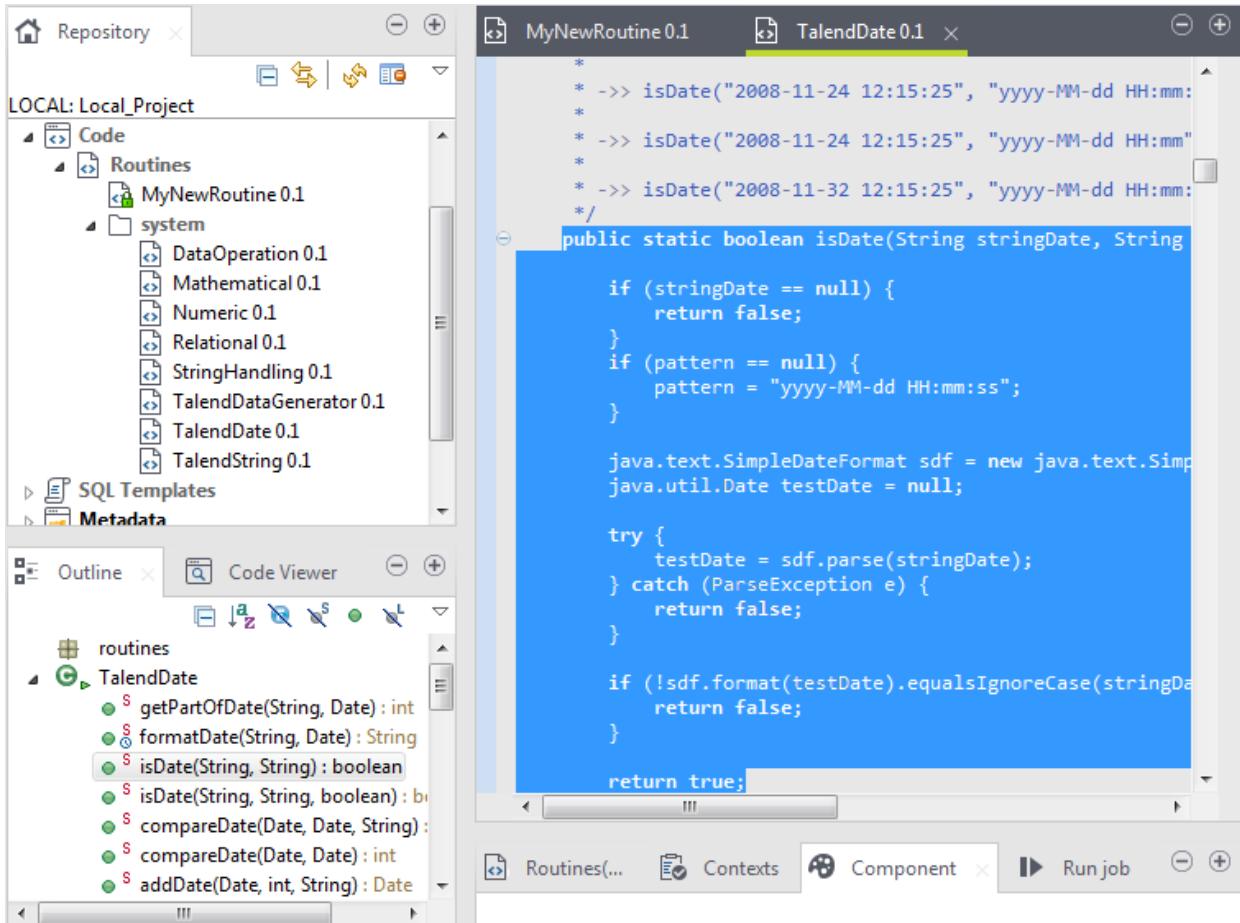
If the system routines are not adapted to your specific needs, you can customize them by copying and pasting the content in a user routine, then modify the content accordingly.

To customize a system routine:

### Procedure

1. First of all, create a user routine by following the steps outlined in [Creating user routines](#) on page 348. 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.

## 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.

### Creating user routines

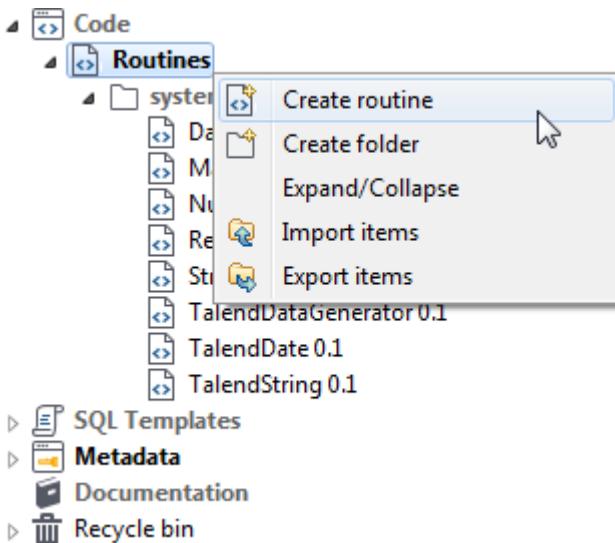
#### About this task

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.

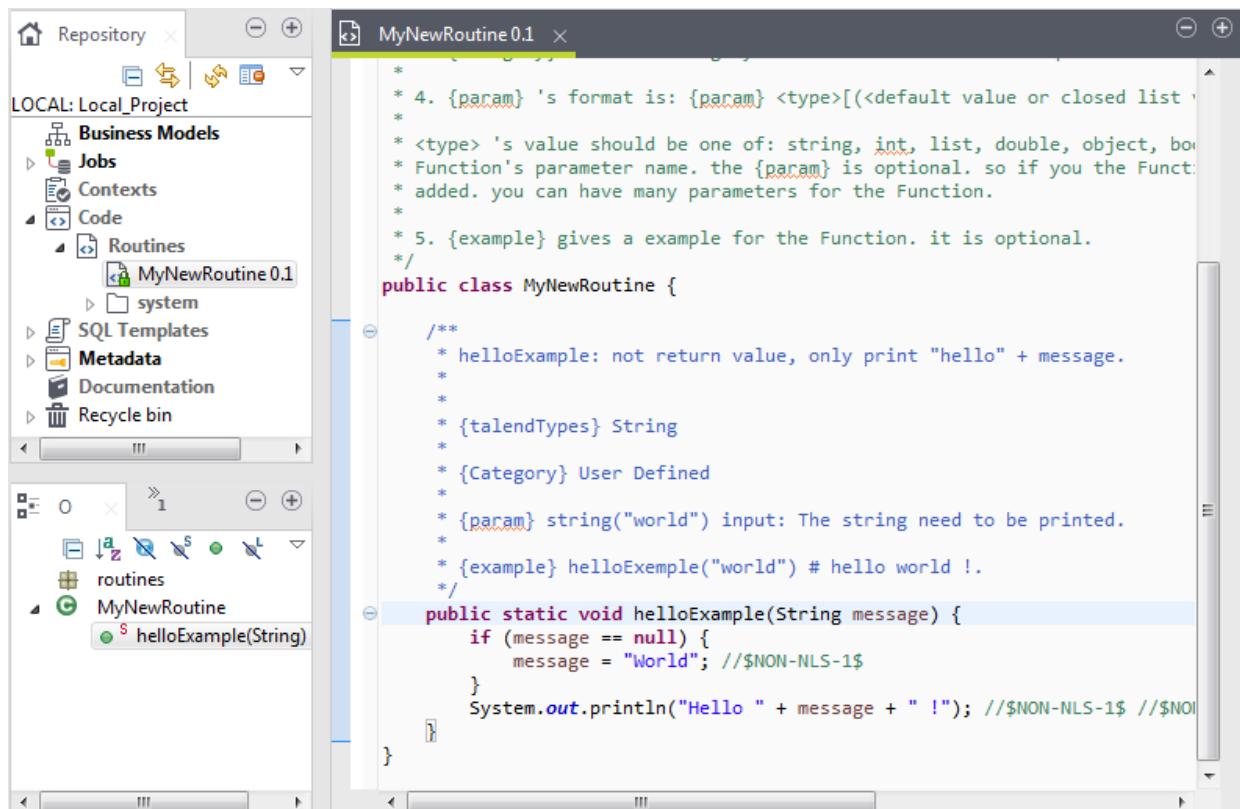
To create a new user routine, complete the following:

## Procedure

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.

**Note:** 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.

## Results

**Note:** 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.

## Editing user routines

### About this task

You can modify the user routines whenever you like.

**Note:** The **system** folder and all of the routines held within are read only.

To edit your user routines:

### Procedure

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.

## Results

If you want to reuse a system routine for your own specific needs, see [Customizing the system routines](#) on page 347.

## Editing user routine libraries

### About this task

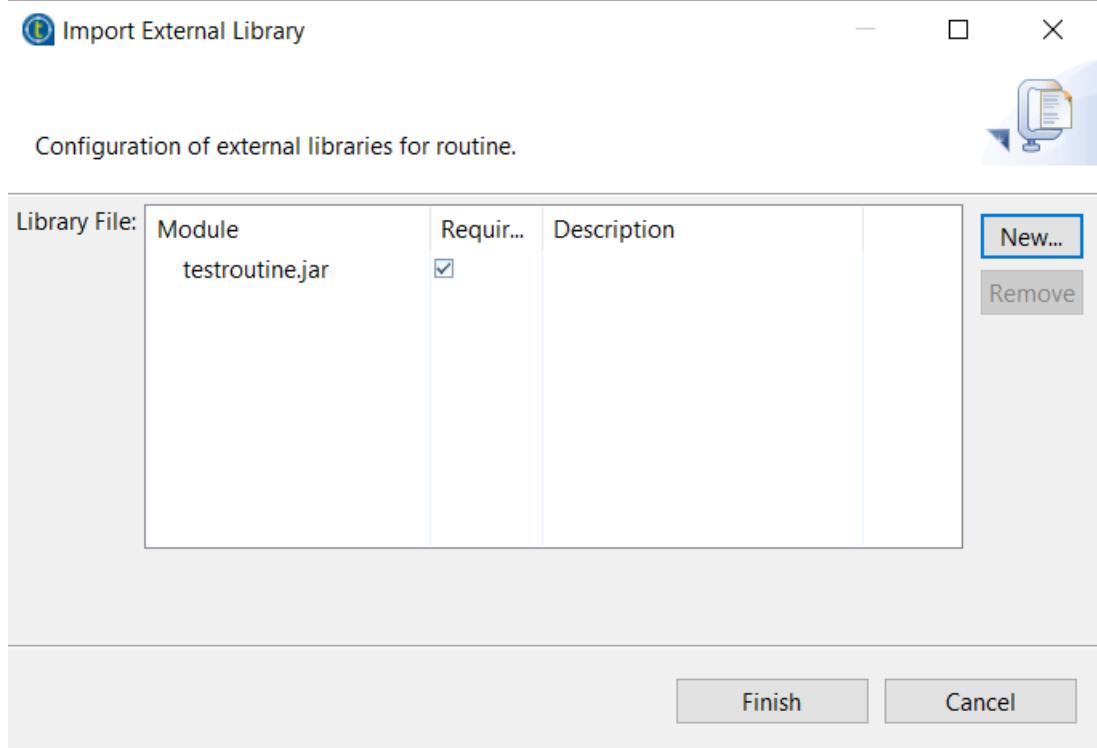
You can edit the library of any of the user routines by importing external libraries (usually .jar files) for the selected routine. These external library files will be listed, like modules, in the **Modules** view in your current Studio. For more information on the **Modules** view, see the *Talend Installation and Migration Guide*.

The imported library will be also listed in the library file of your current Studio.

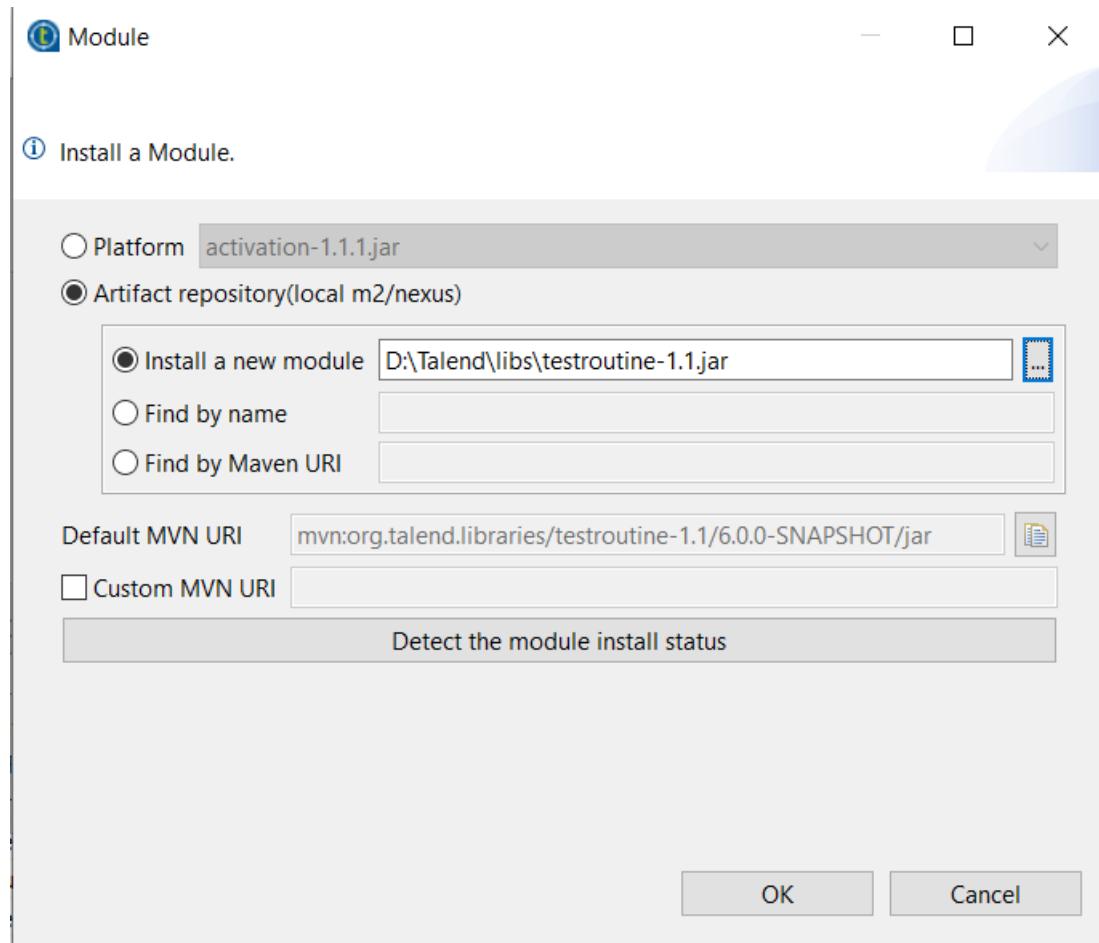
To edit a user routine library, complete the following:

## Procedure

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.



**5.** If you have installed the library using the **Modules** view:

- Select the **Platform** option and then select the library from the list.
- Select the **Artifact repository (local m2/nexus) > Find by name** or **Artifact repository (local m2/nexus) > Find by Maven URI** option, then specify the full name or Maven URI of the library module, and click the **Detect the module install status** button to validate its installation status.

**6.** If you have stored the library file in a local directory:

- a) Select the **Artifact repository (local m2/nexus)** option.
- b) Select the **Install a new module** option, and click the [...] button to browse to library file.
- c) If you need to customize the Maven URI of the library, select the **Custom MVN URI** check box, specify the new URI, and then click the **Detect the module install status** button to validate its installation status.

**Note:**

Changing the Maven URI for an external module will affect all the components and metadata connections that use that module within the project.

When working on a remote project, your custom Maven URI settings will be automatically synchronized to the Talend Artifact Repository and will be used when other users working on the same project install the external module.

**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.

**Note:** 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.

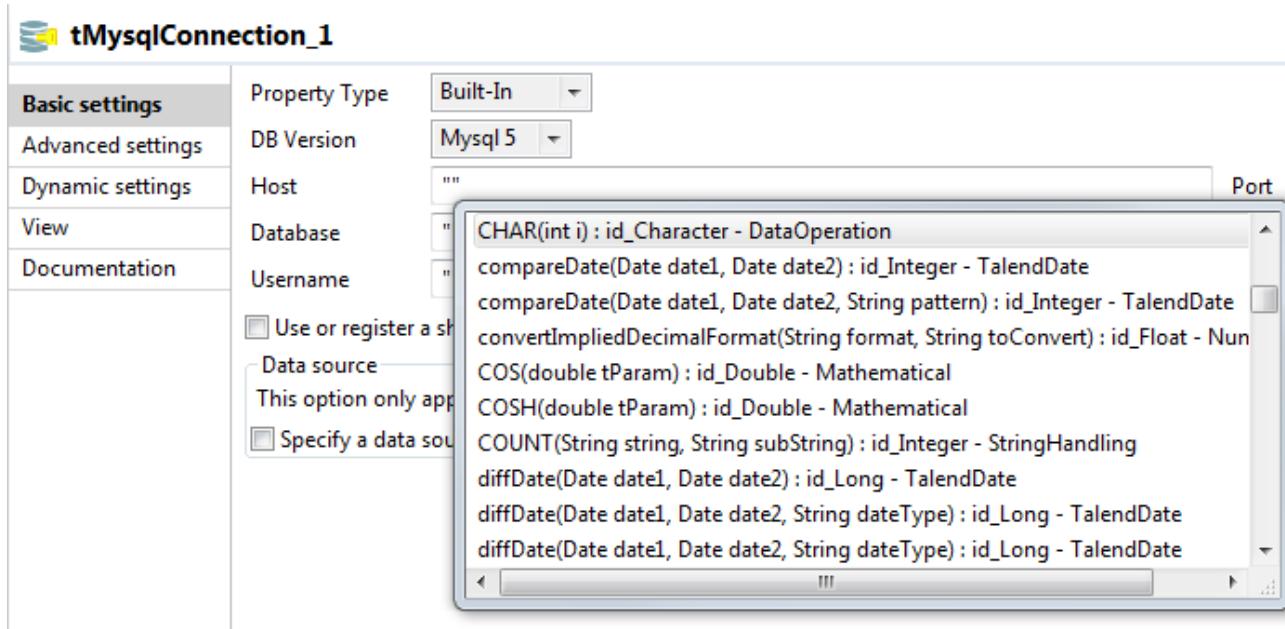
- Click **Finish** to close the dialog box.

## 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](#) on page 25.

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>
```

## Use case: Creating a file for the current date

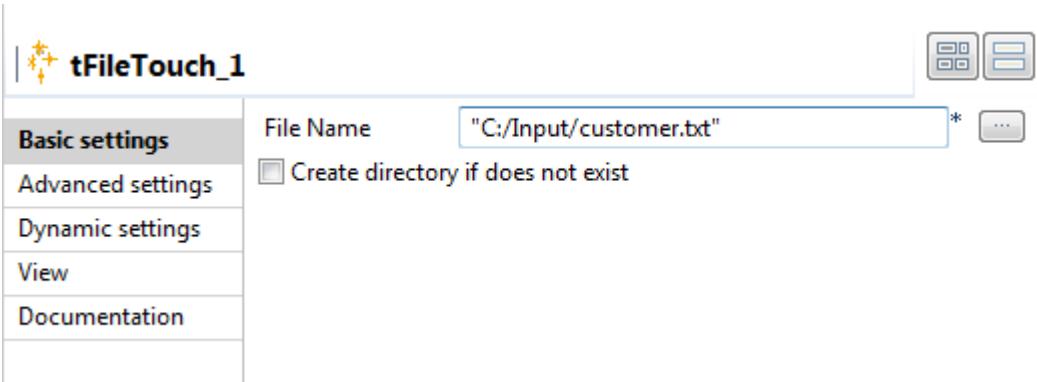
### About this task

This scenario describes how to use a routine. The Job uses just one component, which calls a system routine.

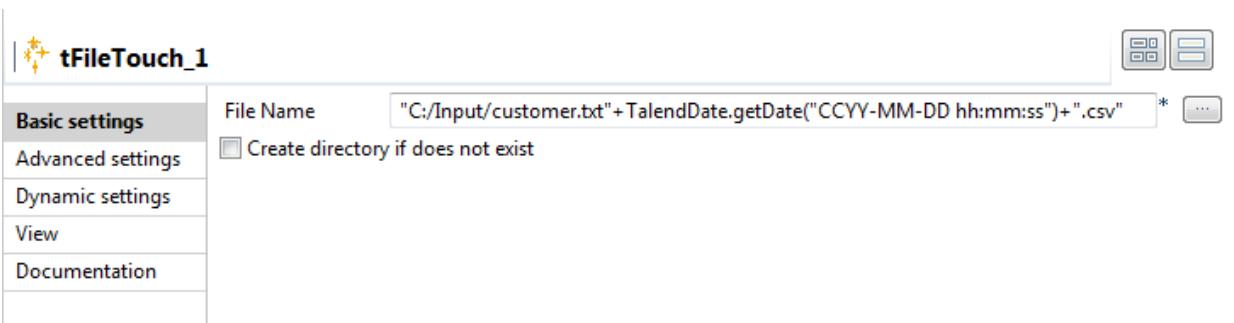


## Procedure

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.



**Warning:** 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.



# System routines

## 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 the routines, double-click the **Numeric** category, in the **system** folder. The Numeric category contains several routines, notably `sequence`, `random` and `convertImpliedDecimalFormat` (decimal):

Routine	Description	Syntax
<code>sequence</code>	Returns an incremental numeric ID.	<code>Numeric.sequence("Parameter name", start value, increment value)</code>
<code>resetSequence</code>	Creates a sequence if it doesn't exist and attributes a new start value.	<code>Numeric.resetSequence (Sequence Identifier, start value)</code>
<code>removeSequence</code>	Removes a sequence.	<code>Numeric.RemoveSequence (Sequence Identifier)</code>
<code>random</code>	Returns a random whole number between the maximum and minimum values.	<code>Numeric.random(minimum start value, maximum end value)</code>
<code>convertImpliedDecimalFormat</code>	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.

### Creating 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
```

## Converting 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]
```

## 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
<code>ISNULL</code>	Checks if the variable provided is a null value.  It returns <code>true</code> if the value is NULL and <code>false</code> if the value is not NULL.	<code>Relational.ISNULL(variable)</code>
<code>NOT</code>	Returns the complement of the logical value of an expression.	<code>Relational.NOT(expression)</code>
<code>isNull</code>	Checks if the variable provided is a null value.  It returns 1 if the value is NULL and 0 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]
```

## 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
ALPHA	Checks whether the expression is arranged in alphabetical order. Returns the true or false boolean accordingly.	StringHandling.ALPHA("string to be checked")
IS_ALPHA	Checks whether the expression contains alphabetical characters only, or otherwise. Returns the true or false boolean accordingly.	StringHandling.IS_ALPHA("string to be checked")
CHANGE	Replaces an element of a string with a defined replacement element and returns the new string.	StringHandling.CHANGE("string to be checked", "string to be replaced", "replacement string")
COUNT	Returns the number of times a substring occurs within a string.	StringHandling.COUNT("string to be checked", "substring to be counted")
DOWNCASE	Converts all uppercase letters in an expression into lowercase and returns the new string.	StringHandling.DOWNCASE("string to be converted")
UPCASE	Converts all lowercase letters in an expression into uppercase and returns the new string.	StringHandling.UPCASE("string to be converted")
DQUOTE	Encloses an expression in double quotation marks.	StringHandling.DQUOTE("string to be enclosed in double quotation marks")
EREPLACE	Substitutes all substrings that match the given regular expression in the given old string with the given replacement and returns a new string.	StringHandling.EREPLACE(oldStr, regex, replacement)
INDEX	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.	StringHandling.INDEX("string to be checked", "substring specified")

Routine	Description	Syntax
LEFT	Specifies a substring which corresponds to the first n characters in a string.	StringHandling.LEFT( "string to be checked" , number of characters)
RIGHT	Specifies a substring which corresponds to the last n characters in a string.	StringHandling.RIGHT( "string to be checked" , number of characters)
LEN	Calculates the length of a string.	StringHandling.LEN( "string to check" )
SPACE	Generates a string consisting of a specified number of blank spaces.	StringHandling.SPACE(number of blank spaces to be generated)
SQUOTE	Encloses an expression in single quotation marks.	StringHandling.SQUOTE("string to be enclosed in single quotation marks")
STR	Generates a particular character a the number of times specified.	StringHandling.STR('character to be generated' , number of times)
TRIM	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.	StringHandling.TRIM( "string to be checked" )
BTRIM	Deletes all the spaces and tabs after the last non-blank character in a string and returns the new string.	StringHandling.BTRIM( "string to be checked" )
FTRIM	Deletes all the spaces and tabs preceding the first non-blank character in a string.	StringHandling.FTRIM( "string to be checked" )
SUBSTR	Returns a portion of a string. It counts all characters, including blanks, starting at the beginning of the string.	StringHandling.SUBSTR(string, start, length) <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>
LTRIM	Removes blanks or characters from the beginning of a string.	StringHandling.LTRIM(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 beginning of the string. LTRIM will compare the <b>trim_set</b> 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 <b>trim_set</b>. If this parameter is not specified, LTRIM will remove any blanks from the beginning of the string.</li> </ul>

Routine	Description	Syntax
RTRIM	Removes blanks or characters from the end of a string.	<pre>StringHandling.RTRIM(string[, trim_set])</pre> <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. RTRIM will compare the <b>trim_set</b> 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 <b>trim_set</b>. If this parameter is not specified, RTRIM will remove any blanks from the ending of the string.</li> </ul>
LPAD	Converts a string to a specified length by adding blanks or characters to the beginning of the string.	<pre>StringHandling.LPAD(first_string, length[, second_string])</pre> <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 <b>first_string</b>.</li> </ul>
RPAD	Converts a string to a specified length by adding blanks or characters to the end of the string.	<pre>StringHandling.RPAD(first_string, length[, second_string])</pre> <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 <b>first_string</b>.</li> </ul>
INSTR	Returns the position of a character set in a string, counting from left to right and starting from 1.  Note that it returns 0 if the search is unsuccessful and NULL if the search value is NULL.	<pre>StringHandling.INSTR(string, search_value, start, occurrence)</pre> <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, <code>StringHandling.INSTR( "Talend Technology", "e", 3, 2)</code>, it will start the search from the third character 1 and return 7, the position of the second character e.</p>
TO_CHAR	Converts numeric values to text strings.	<pre>StringHandling.TO_CHAR(numeric_value)</pre>

### Storing 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
```

## Checking 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.println(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]
```

## Replacing 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.println(StringHandling.CHANGE( "hello world!", "world", "guy" ));
```

The routine replaces the old element with the new element specified.

```
|hello guy!
```

## Checking 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.println(StringHandling.INDEX( "hello world!", "hello"));
System.out.println(StringHandling.INDEX( "hello world!", "world"));
System.out.println(StringHandling.INDEX( "hello world!", "!" ));
System.out.println(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]
```

## Calculating 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.println(StringHandling.LEN( "hello world!" ));
```

The check returns a whole number which indicates the length of the chain, including spaces and blank characters.

|12

## Deleting blank characters

The FTRIM routine is easy to use, along with a **tJava** component, to delete blank characters from the start of a string:

```
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]
```

## 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
getFirstName	returns a first name taken randomly from a fictitious list.	TalendDataGenerator.getFirstName()
getLastName	returns a random surname from a fictitious list.	TalendDataGenerator.getLastName()
getUsStreet	returns an address taken randomly from a list of common American street names.	TalendDataGenerator.getUsStreet()
getUsCity	returns the name of a town taken randomly from a list of American towns.	TalendDataGenerator.getUsCity()
getUsState	returns the name of a State taken randomly from a list of American States.	TalendDataGenerator.getUsState()
getUsStateId	returns an ID randomly taken from a list of IDs attributed to American States.	TalendDataGenerator.getUsStateId()

**Note:** No entry parameter is required as **Talend** provides the list of fictitious data.

You can customize the fictitious data by modifying the TalendDataGenerator routines. For further information on how to customize routines, see [Customizing the system routines](#) on page 347.

## Generating 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.println(TalendDataGenerator.getLastName());
System.out.println(TalendDataGenerator.getUsCity());
System.out.println(TalendDataGenerator.getUsState());
System.out.println(TalendDataGenerator.getUsStateId());
System.out.println(TalendDataGenerator.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]
```

## 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
addDate	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("initial date string", "date format - eg.: yyyy/MM/dd", integer n, "format of the part of the date to which n is to be added - eg.:yyyy").
compareDate	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")
diffDate	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")
diffDateFloor	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")

Routine	Description	Syntax
formatDate	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)
formatDateLocale	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")
getCurrentDate	Returns the current date. No entry parameter is required.	TalendDate.getCurrentDate()
getDate	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 of DD/MM/CCYY.	TalendDate.getDate("Format of the string - ex: CCYY-MM-DD")
getFirstDayOfMonth	Changes the date of an event to the first day of the current month and returns the new date.	TalendDate.getFirstDayMonth(Date)
getLastDayOfMonth	Changes the date of an event to the last day of the current month and returns the new date.	TalendDate.getLastDayMonth(Date)
getPartOfDay	Returns part of a date according to the format specified. This string can contain fixed character strings or variables linked to the date.	TalendDate.getPartOfDay("String indicating the part of the date to be retrieved, "String in the format of the date to be parsed")
getRandomDate	Returns a random date, in the ISO format.	TalendDate.getRandomDate("format date of the character string", String minDate, String maxDate)
isDate	Checks whether the date string corresponds to the format specified. Returns the boolean value true or false according to the outcome.	TalendDate.isDate(Date() to be checked, "format of the date to be checked - eg.: yyyy-MM-dd HH:mm:ss")
parseDate	Changes a string into a Date. Returns a date in the standard format.	TalendDate.parseDate("format date of the string to be parsed", "string in the format of the date to be parsed")
parseDateInUTC	changes a string into a Date in UTC. Returns a date in the UTC format.	TalendDate.parseDateInUTC("format date of the string to be parsed", "string in the format of the date to be parsed", "boolean about whether parsing is set to be lenient, that is to say, accepting the heuristic match with the format")
parseDateLocale	Parses a string according to a specified format and extracts the date. Returns the date according to the local format specified.	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")

Routine	Description	Syntax
setDate	Modifies part of a date according to the part and value of the date specified and the format specified.	TalendDate.setDate(Date, whole n, "format of the part of the date to be modified - eg.:yyyy")
TO_CHAR	Converts a date to a character string.	<p>TalendDate.TO_CHAR(date[,format])</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>
TO_DATE	Converts a character string to a Date/Time datatype.	<p>TalendDate.TO_DATE(string[, format])</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 <b>string</b> argument. If not specified, the <b>string</b> value must be in the date format <code>MM/dd/yyyy HH:mm:ss.SSS</code>.</li> </ul> <p>For example, <code>TalendDate.TO_DATE( "04/24/2017 13:55:42.123" )</code> will return <code>Mon Apr 24 13:55:42 CST 2017</code>.</p>
ADD_TO_DATE	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>TalendDate.ADD_TO_DATE(date, format, amount)</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.</li> <li>• Valid format strings for year: Y, YY, YYYY, and YYYY.</li> <li>• Valid format strings for month: MONTH, MM, and MON.</li> <li>• Valid format strings for day: D, DD, DDD, DAY, and DY.</li> <li>• Valid format strings for hour: HH, HH12, and HH24.</li> <li>• Valid format string for minute: MI.</li> <li>• Valid format string for second : SS.</li> <li>• Valid format string for millisecond: MS.</li> <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> <pre>if TalendDate.getCurrentDate() returns Mon Apr 24 14:26:03 CST 2017, TalendDate.ADD_TO_DATE(TalendDate.getCurrentDate(), "YY", 1) will return Tue Apr 24 14:26:03 CST 2018.</pre>

### Warning:

Although "yyyy" and "YYYY" in the date format return the same year number in most cases , "YYYY" may not work as expected when used:

- at the first week of a year if the year does not start by the first day of the week.
- at the last week of a year if the year does not end by the last day of the week.

For example, when calculating what date it is 3 days before January 2, 2016, the code below would return a wrong date:

```
System.out.println(TalendDate.formatDate("YYYY-MM-dd", TalendDate.addDate(TalendDate.  
e_TO_DATE("01/02/2016 08:10:30.123"), -3, "dd")));
```

while the following code would work as expected:

```
System.out.println(TalendDate.formatDate("yyyy-MM-dd", TalendDate.addDate(TalendDate.  
e_TO_DATE("01/02/2016 08:10:30.123"), -3, "dd")));
```

Therefore, you should typically use "yyyy", which represents calendar year.

## Formatting a Date

The **formatDate** routine is easy to use, along with a **tJava** component:

```
System.out.println(TalendDate.formatDate("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]*

## Checking 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]*

## Comparing 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", "2025/11/24")));
```

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]*

## Configuring 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]*

## Parsing 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]*

## Retrieving part of a Date

It is easy to use the `getPartOfDay` 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.getPartOfDay("DAY_OF_MONTH", D));
System.out.println(TalendDate.getPartOfDay("MONTH", D));
System.out.println(TalendDate.getPartOfDay("YEAR", D));
System.out.println(TalendDate.getPartOfDay("DAY_OF_YEAR", D));
System.out.println(TalendDate.getPartOfDay("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]
```

**Note:** 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.

## Formatting 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.println(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]
```

## TalendString Routines

The TalendString routines allow you to carry out various operations on alphanumerical expressions.

To access these routines, double-click **TalendString** under the **system** folder. The TalendString class contains the following routines:

Routine	Description	Syntax
replaceSpecialCharForXML	returns a string from which the special characters (eg.: <, >, &...) have been replaced by equivalent XML characters.	<code>TalendString.replaceSpecialCharForXML("string containing the special characters - eg.: Thelma &amp; Louise")</code>
checkCDATAForXML	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.	<code>TalendString.checkCDATAForXML("string to be parsed")</code>

Routine	Description	Syntax
talendTrim	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)
removeAccents	removes accents from a string and returns the string without the accents.	TalendString.removeAccents("String")
getAsciiRandomString	generates a random string with a specific number of characters.	TalendString.getAsciiRandomString(whole number indicating the length of the string)

## Formatting an XML string

It is easy to run the `replaceSpecialCharForXML` routine along with a **tJava** component, to format a string for XML:

```
System.out.println(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]
```

## Trimming 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.println(TalendString.talendTrim("**talend open studio****",'*', -1));  
System.out.println(TalendString.talendTrim("**talend open studio****",'*', 1));  
System.out.println(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]
```

## Removing 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.println(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]
```

## 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
DECODE	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> <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 null.</li> <li>• <b>search</b>: the value for which you want to search. The search value must have the same datatype as the <b>value</b> 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.

# Appendices

## Customizing Talend Studio and setting Studio preferences

### Customizing project settings

#### About this task

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:

#### Procedure

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.

#### Results

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.

### Setting the compiler compliance level

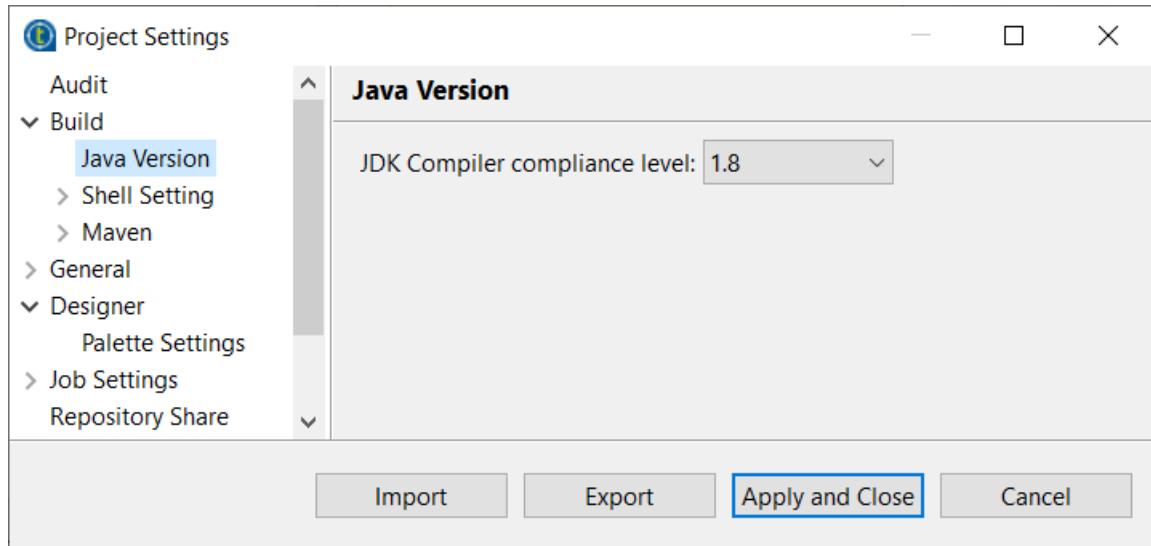
#### About this task

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 Migration Guide .

#### Procedure

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 **Apply and Close**.



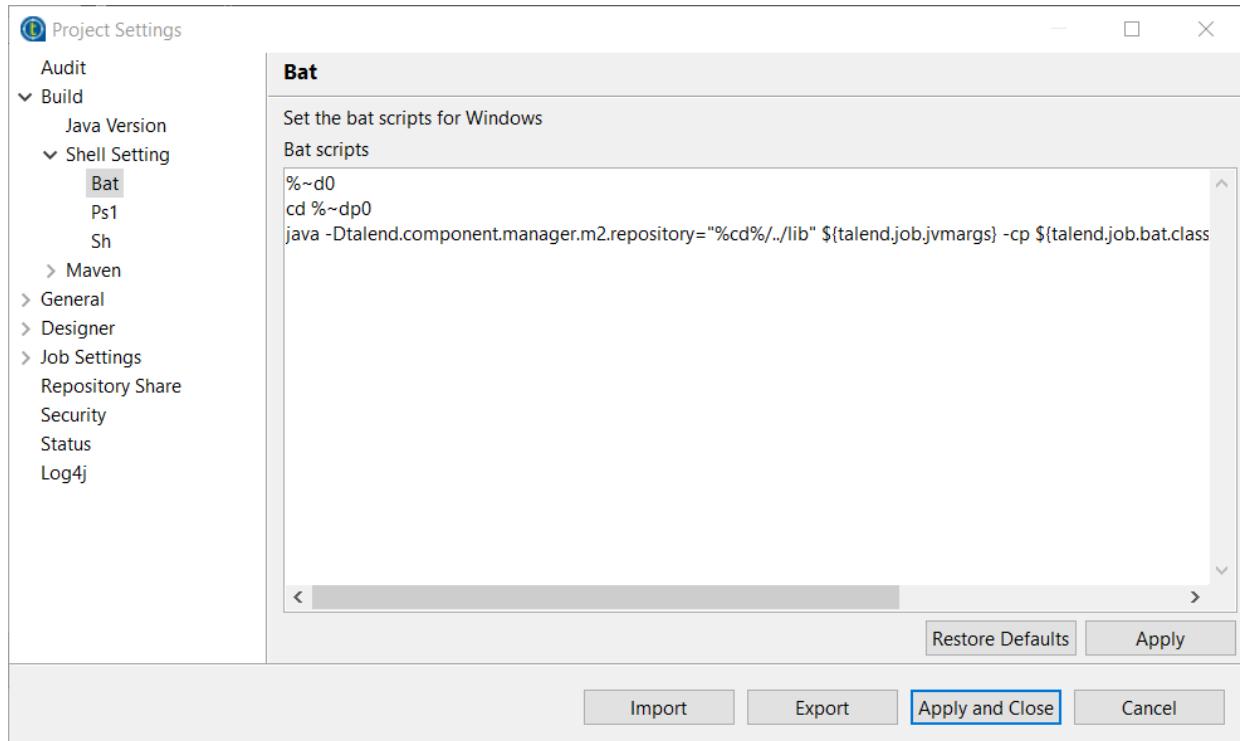
## Customizing shell command templates

### About this task

Your Talend Studio provides a set of templates for the shell commands used to launch built Jobs. You can customize those templates based on your needs.

### Procedure

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 > Shell Setting** nodes and click **Bat**, **Ps1**, or **Sh** depending on the operating system on which your built Jobs are going to run.



3. Edit the code in the corresponding view based on your needs and click **Apply and Close** to finish your customization.

## Customizing Maven build script templates

Your Talend Studio provides the following default templates for generating build scripts:

- Docker image build settings
- 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.

### Customizing the global build script templates

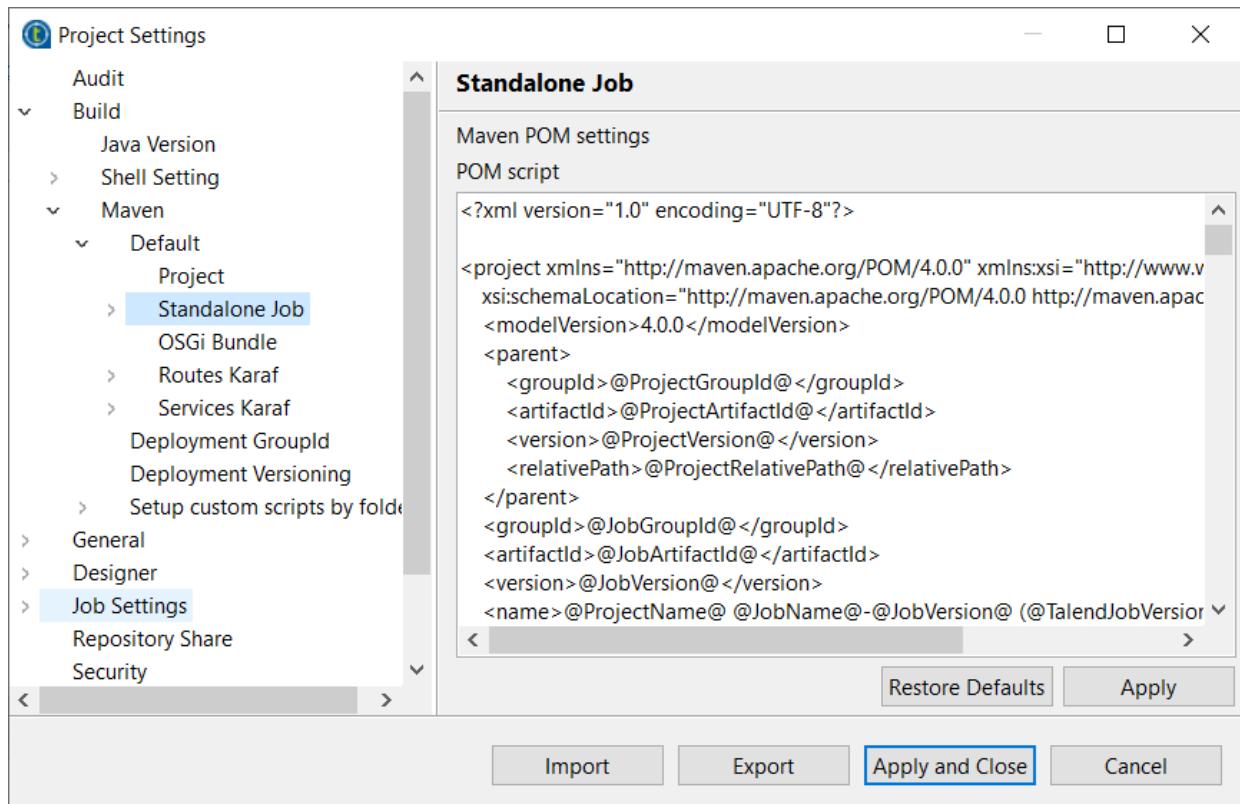
#### About this task

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 or Routes 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:

#### Procedure

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.



**Note:**

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 **Apply and Close** to finish your customization.

### Customizing the folder-level build script templates

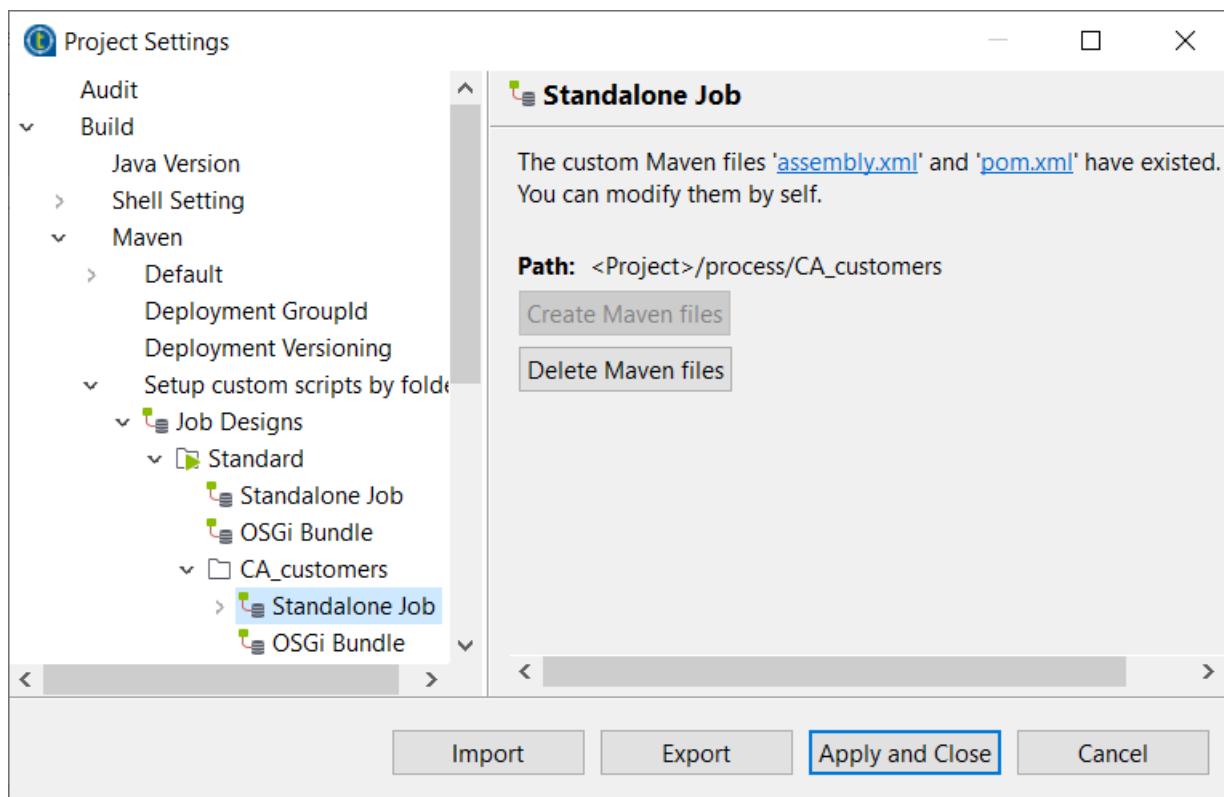
#### About this task

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:

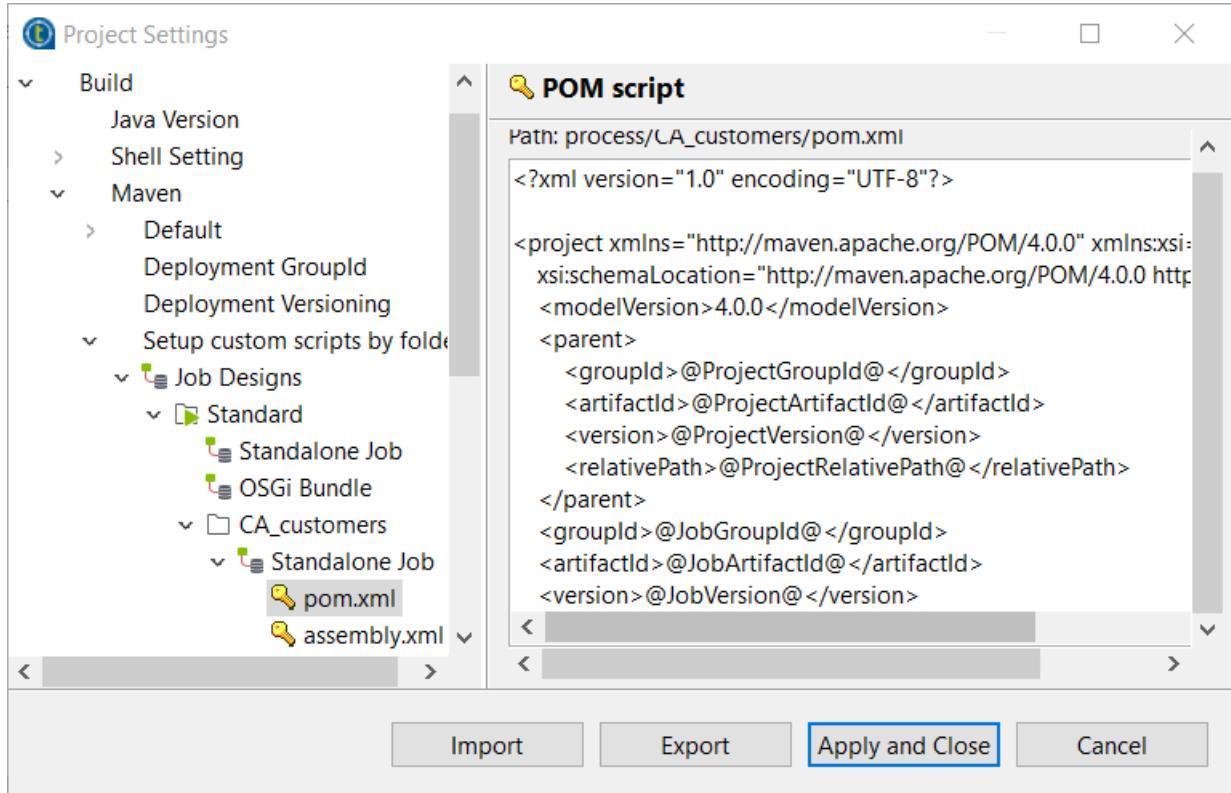
#### Procedure

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.

**Note:**

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 **Apply and Close** 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.

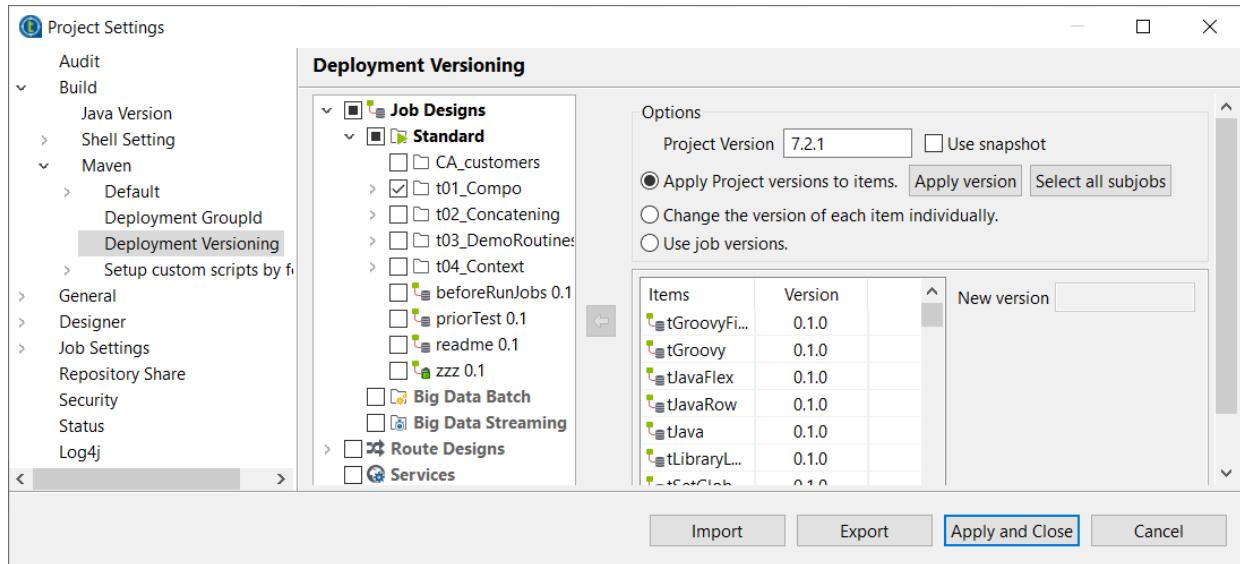
## Managing deployment versions of Jobs

### About this task

Through the **Project Settings** dialog box, you can manage in a batch manner or individually the deployment version of each Job item to be published to the artifact repository.

### Procedure

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 **Build > Maven** and select **Deployment Versioning** to open the corresponding view.



- In the **Repository** tree view, expand the node holding the items you want to manage their deployment versions and then select the check boxes of these items.

The selected items are displayed in the **Items** list to the right along with their current version in the **Version** column.

- Make changes as required:

- To set a deployment version for all the items, enter the version in the **Project version** field, select the **Apply project version to items** option, and click **Apply version**.
- Click **Select all subJobs** if you want to update all of the subJobs dependent on the selected items at the same time.
- To set the deployment version for one or more items individually, select the **Change the version of each item individually** option, select the item or items in the table below the **Options** area, enter the deployment version **New version** field, and click **Apply**.
- Select the **Use job versions** option if you want to use the latest Job versions as the deployment versions for the selected items.
- If you want to publish a snapshot version of the item or items, select the **Use snapshot** check box before applying the version setting.

- Click **Apply and Close** to apply your changes and close the dialog box.

- If any of the selected Jobs is opened in the design workspace, click **OK** in a new dialog box and then save your Job to apply the new deployment version to it.

You can also set the deployment version of a Job in the **Job** view of it once opened in the design workspace. For more information, see [Customizing deployment of a Job](#).

The deployment version settings you make in the **Project Settings** dialog box will be reflected in the **Deployment** tab in the **Job** view of the relevant item or items, and vice versa.

## Palette Settings

### About this task

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:

## Procedure

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.

**Note:**

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 **Apply and Close** to close the dialog box.

## Results

**Note:**

To get back to the **Palette** default settings, click **Restore Defaults**.

For more information on the **Palette**, see [Changing the Palette layout and settings](#) on page 391.

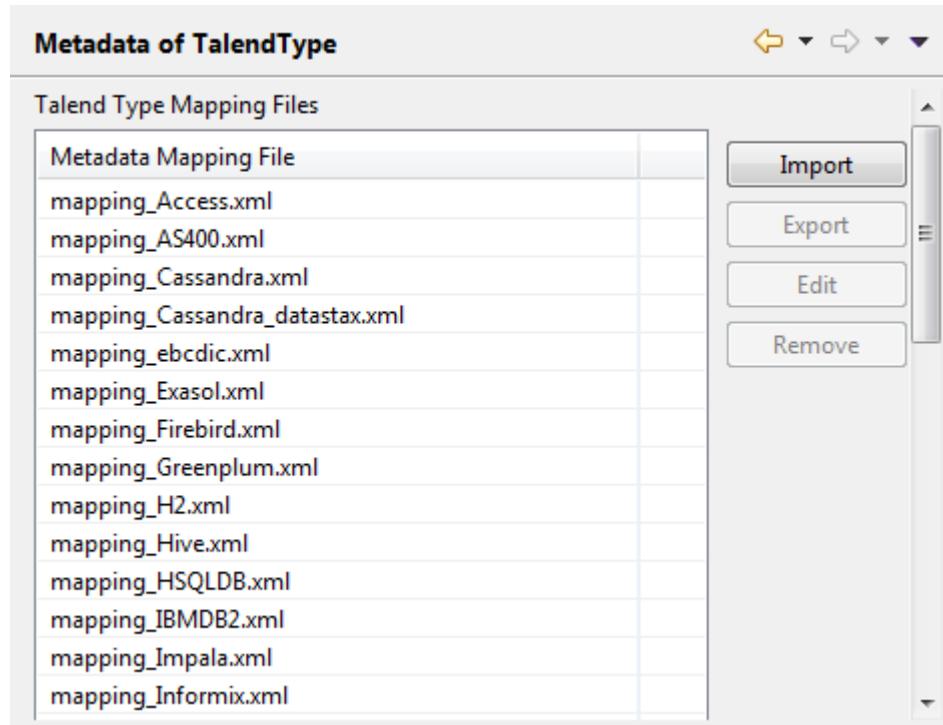
## Type mapping

You can set the parameters for type conversion in Talend Studio, from Java towards databases and vice versa.

### Accessing mapping files and defining type mappings

## Procedure

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 a conversion file according to your needs by double-clicking the file or selecting the file and clicking the **Edit** button to open the **Edit Mapping File** dialog box and then modify the XML code directly in the open dialog box.

Note that when you define a type mapping, you need to map both from Talend type to database type and from database type to Talend type.

- The `<dbTypes>` element with its child `<dbType>` elements defines the supported database types. To add a new database type in the mapping file, you need to add a `<dbType>` element under the `<dbTypes>` element. The example below adds two database types `BOOLEAN` and `YESNO`.

```
<dbType type="BOOLEAN" ignoreLen="true" ignorePre="true"/>
<dbType type="YESNO" ignoreLen="true" ignorePre="true" />
```

- The `<talendToDbType>` element with its child `<talendType>` elements defines a suggested database type list and the default database type when setting a Talend type for a metadata column. To map a Talend type to one or more database types, you need to add a `talendType` element under the `<talendToDbType>` element. The example below maps the Talend type `id_Boolean` to two database types `BOOLEAN` and `YESNO`.

```
<talendType type="id_Boolean">
  <dbType type="YESNO" default="true" />
  <dbType type="BOOLEAN" />
</talendType>
```

- The `<dbToTalendTypes>` element with its child `<dbType>` elements defines a suggested Talend type list and the default Talend type when retrieving schema from the database. To map a database type to one or more Talend types, you need to add a `dbType` element under

the <dbToTalendTypes> element. The example below maps the database type YESNO to the Talend type id\_Boolean.

```
<dbType type="YESNO">
  <talendType type="id_Boolean" default="true" />
</dbType>
```

Below is the XML metadata mapping file for the Access database:

```

<?xml version="1.0"?>
<mapping>
    <dbms product="ACCESS" id="access_id" label="Mapping Access" default="true">
        <dbTypes>
            <dbType type="BIT" ignoreLen="true" ignorePre="true"/>
            <dbType type="BOOLEAN" ignoreLen="true" ignorePre="true"/>
            <dbType type="COUNTER"/>
            <dbType type="DATE" ignoreLen="true" ignorePre="true"/>
            <dbType type="DOUBLE" ignoreLen="true" ignorePre="true"/>
            <dbType type="DECIMAL" ignoreLen="true" ignorePre="true"/>
            <dbType type="FLOAT" ignoreLen="true" ignorePre="true"/>
            <dbType type="INTEGER" ignoreLen="true" ignorePre="true"/>
            <dbType type="NUMERIC" ignoreLen="true" ignorePre="true"/>
            <dbType type="REAL" ignoreLen="true" ignorePre="true"/>
            <dbType type="SMALLINT" ignoreLen="true" ignorePre="true"/>
            <dbType type="TINYINT" ignoreLen="true" ignorePre="true"/>
            <dbType type="TIME" ignoreLen="true" ignorePre="true"/>
            <dbType type="TIMESTAMP" ignoreLen="true" ignorePre="true"/>
            <dbType type="VARCHAR" default="true" defaultLength="200"
ignorePre="true"/>
            <dbType type="DATETIME" ignoreLen="true" ignorePre="true"/>
            <dbType type="MEMO" ignoreLen="true" ignorePre="true"/>
            <dbType type="YESNO" ignoreLen="true" ignorePre="true"/>
        </dbTypes>

        <language name="java">
            <talendToDbTypes>
                <!-- Adviced mappings -->
                <talendType type="id_List"/>
                <talendType type="id_Boolean">
                    <dbType type="YESNO" default="true"/>
                    <dbType type="BOOLEAN"/>
                </talendType>
                <talendType type="id_Byte">
                    <dbType type="TINYINT" default="true"/>
                    <dbType type="SMALLINT"/>
                    <dbType type="INTEGER"/>
                </talendType>
                <talendType type="id_byte[]"> </talendType>
                <talendType type="id_Character">
                    <dbType type="VARCHAR" default="true"/>
                </talendType>
                <talendType type="id_Date">
                    <dbType type="DATE" default="true"/>
                    <dbType type="TIMESTAMP"/>
                    <dbType type="TIME"/>
                    <dbType type="DATETIME"/>
                </talendType>
                <talendType type="id_BigDecimal">
                    <dbType type="NUMERIC" default="true"/>
                    <dbType type="DOUBLE"/>
                    <dbType type="FLOAT"/>
                    <dbType type="DECIMAL"/>
                    <dbType type="REAL"/>
                </talendType>
                <talendType type="id_Double">
                    <dbType type="DOUBLE" default="true"/>
                    <dbType type="NUMERIC"/>
                    <dbType type="FLOAT"/>
                    <dbType type="DECIMAL"/>
                    <dbType type="REAL"/>
                </talendType>
                <talendType type="id_Float">
                    <dbType type="FLOAT" default="true"/>
                    <dbType type="DOUBLE"/>
                    <dbType type="NUMERIC"/>
                    <dbType type="DECIMAL"/>
                    <dbType type="REAL"/>
                </talendType>
                <talendType type="id_Integer">
                    <dbType type="INTEGER" default="true"/>
                    <dbType type="SMALLINT"/>
                </talendType>
            </talendToDbTypes>
        </language>
    </dbms>
</mapping>
```

```

        <dbType type="TINYINT" />
        <dbType type="COUNTER" />
    </talendType>
    <talendType type="id_Long">
        <dbType type="INTEGER" default="true" />
        <dbType type="SMALLINT" />
        <dbType type="TINYINT" />
        <dbType type="COUNTER" />
    </talendType>
    <talendType type="id_Object" /> </talendType>
    <talendType type="id_Short">
        <dbType type="SMALLINT" default="true" />
        <dbType type="INTEGER" />
        <dbType type="TINYINT" />
        <dbType type="COUNTER" />
    </talendType>
    <talendType type="id_String">
        <dbType type="VARCHAR" default="true" />
        <dbType type="MEMO" />
    </talendType>
</talendToDbTypes>
<dbToTalendTypes>
    <dbType type="BIT">
        <talendType type="id_Boolean" default="true" />
    </dbType>
    <dbType type="BOOLEAN">
        <talendType type="id_Boolean" default="true" />
    </dbType>
    <dbType type="COUNTER">
        <talendType type="id_Integer" default="true" />
    </dbType>
    <dbType type="DATE">
        <talendType type="id_Date" default="true" />
    </dbType>
    <dbType type="DECIMAL">
        <talendType type="id_Double" />
        <talendType type="id_BigDecimal" default="true" />
        <talendType type="id_Float" />
    </dbType>
    <dbType type="DOUBLE">
        <talendType type="id_Double" default="true" />
        <talendType type="id_BigDecimal" />
        <talendType type="id_Float" />
    </dbType>
    <dbType type="FLOAT">
        <talendType type="id_Float" default="true" />
        <talendType type="id_BigDecimal" />
        <talendType type="id_Double" />
    </dbType>
    <dbType type="INTEGER">
        <talendType type="id_Integer" default="true" />
        <talendType type="id_Short" />
        <talendType type="id_Long" />
        <talendType type="id_Byte" />
    </dbType>
    <dbType type="NUMERIC">
        <talendType type="id_Float" />
        <talendType type="id_BigDecimal" default="true" />
        <talendType type="id_Double" />
    </dbType>
    <dbType type="REAL">
        <talendType type="id_Float" default="true" />
        <talendType type="id_BigDecimal" />
        <talendType type="id_Double" />
    </dbType>
    <dbType type="SMALLINT">
        <talendType type="id_Short" default="true" />
        <talendType type="id_Integer" />
        <talendType type="id_Long" />
        <talendType type="id_Byte" />
    </dbType>
    <dbType type="TINYINT">
        <talendType type="id_Byte" default="true" />
        <talendType type="id_Integer" />
        <talendType type="id_Short" />
        <talendType type="id_Long" />
    </dbType>

```

```

        </dbType>
        <dbType type="TIME">
            <talendType type="id_Date" default="true"/>
        </dbType>
        <dbType type="TIMESTAMP">
            <talendType type="id_Date" default="true"/>
        </dbType>
        <dbType type="VARCHAR">
            <talendType type="id_String" default="true"/>
        </dbType>
        <dbType type="DATETIME">
            <talendType type="id_Date" default="true"/>
        </dbType>
        <dbType type="MEMO">
            <talendType type="id_String" default="true"/>
        </dbType>
        <dbType type="YESNO">
            <talendType type="id_Boolean" default="true"/>
        </dbType>
    </dbToTalendTypes>
</language>
</dbms>
</mapping>

```

## Supported Talend types

Talend supports the following types.

**Note:** You cannot define custom types in Talend Studio.

Talend Type	Java Type
id_Boolean	java.lang.Boolean
id_Byte	java.lang.Byte
id_byte[]	byte[]
id_Character	java.lang.Character
id_Date	java.util.Date
id_Double	java.lang.Double
id_Float	java.lang.Float
id_BigDecimal	java.math.BigDecimal
id_Integer	java.lang.Integer
id_Long	java.lang.Long
id_Object	java.lang.Object
id_Short	java.lang.Short
id_String	java.lang.String
id_List	java.util.List

## Frequently used attributes in dbType element

Using attributes in the conversion mappings, you can define the default values or behavior of the relevant schema columns. The table below describes the frequently used attributes in the dbType element.

Attribute	Description
ignoreLen	When set to <code>true</code> , the length that you set for newly added columns of the type in schema (by <b>Edit schema</b> ) on components will be ignored.
ignorePre	When set <code>true</code> , the precision that you set for newly added columns of the type in schema (by <b>Edit schema</b> ) on components will be ignored.
default	When set <code>true</code> , the type defined in this element will be the default type of newly added columns (by <b>Edit schema</b> ) on components.
defaultLength	Sets the default length of newly added columns of the type (by <b>Edit schema</b> ) on components.
defaultPrecision	Sets the default precision of newly added columns of the type (by <b>Edit schema</b> ) on components.

## Version management

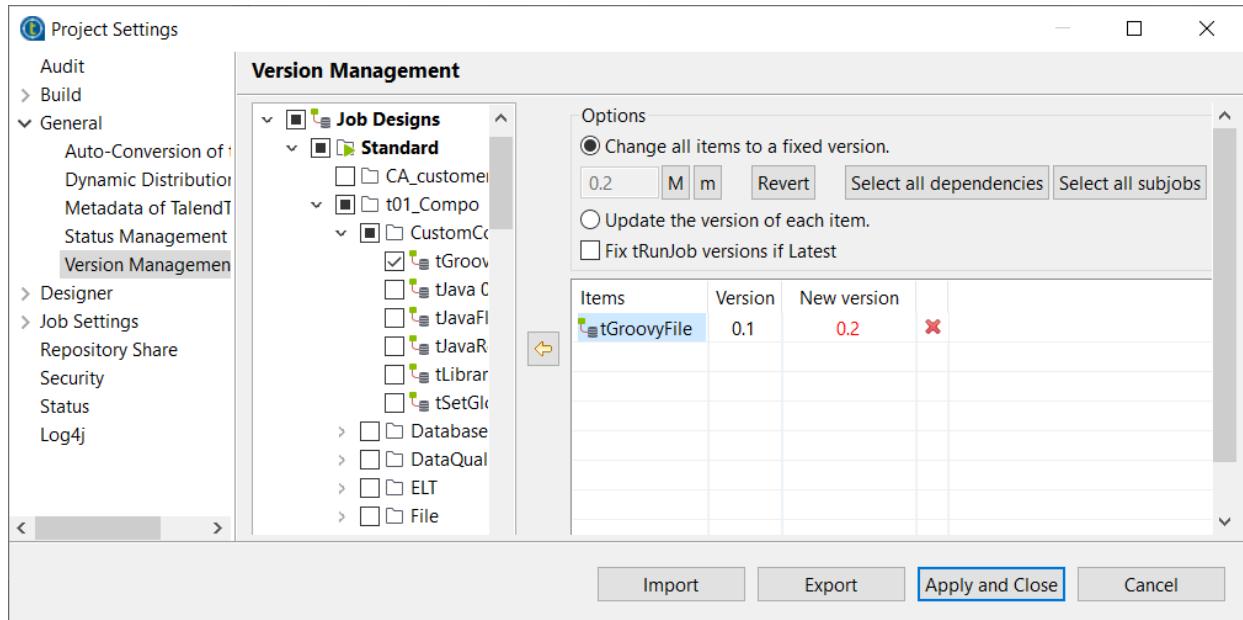
### About this task

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:

### Procedure

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.



3. 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.

4. 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 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.

**Warning:** 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).

5. Click **Apply and Close** to apply your changes and close the dialog box.

## Status management

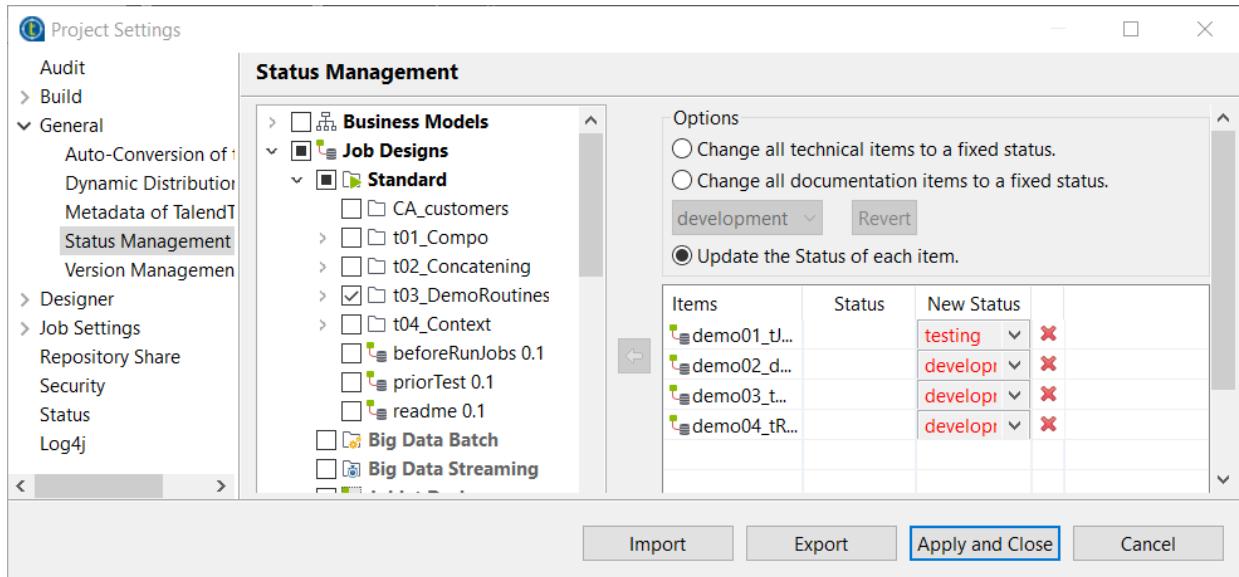
### About this task

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:

## Procedure

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 Status of each item** check box and change them manually.
7. Click **Apply and Close** to apply your changes and close the dialog box.

## Results

### Note:

For further information about Job status, see [Status settings on page 389](#).

## Job Settings

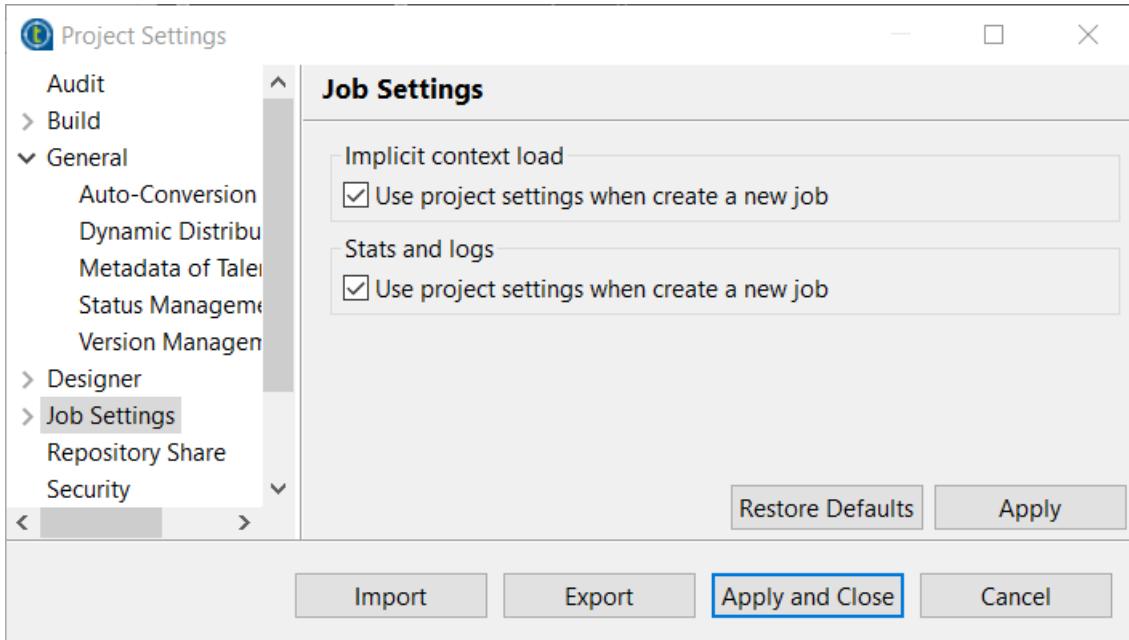
### About this task

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:

## Procedure

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 **Job Settings** node to open the corresponding view.
3. Select the **Use project settings when create a new job** check boxes of the **Implicit Context Load** and **Stats and Logs** areas.



4. Click **Apply** to validate your changes and then **Apply and Close** to close the dialog box.

## Stats & Logs

### About this task

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:

## Procedure

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 **Stats & Logs** to display the corresponding view.

## Stats & Logs

The screenshot shows the 'Stats & Logs' configuration dialog in Talend Studio. It includes the following settings:

- Log Types:**  Use statistics (tStatCatcher)  Use logs (tLogCatcher)  Use volumetrics (tFlowMeterCatcher)
- Storage Options:**  On Console  On Files
- File Paths:**
  - File Path: "C:/talend/Talend-Studio/workspace/.metadata"
  - Stats File Name: "stats\_file.txt"
  - Log File Name: "logs\_file.txt"
  - Meter file name: "meter\_file.txt"
- Encoding:** ISO-8859-15
- On Databases:**
  - Property Type: Repository (selected) DB (MYSQL):MySQL
  - Db Type: MySQL Db Version: MySQL 5
  - Host: "localhost" Port: "3306"
  - Db Name: "talend" Additional parameters: "noDatetimeStringSync=true"
  - User: "root" Password: \*\*\*\*\*
  - Stats Table: "statistics"
  - Logs Table: "logs"
  - Meter Table: "volumetrics"
- Catching Errors:**
  - Catch runtime errors
  - Catch user errors
  - Catch user warnings
  - Catch components statistics (tStatCatcher Statistics)

### Note:

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 [Automating the use of statistics & logs on page 94](#).

3. 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.
4. 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.

### Results

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.

### Note:

Alternatively, if you save your connection information in a Context, you can also access them through **Ctrl+Space**.

## Context settings

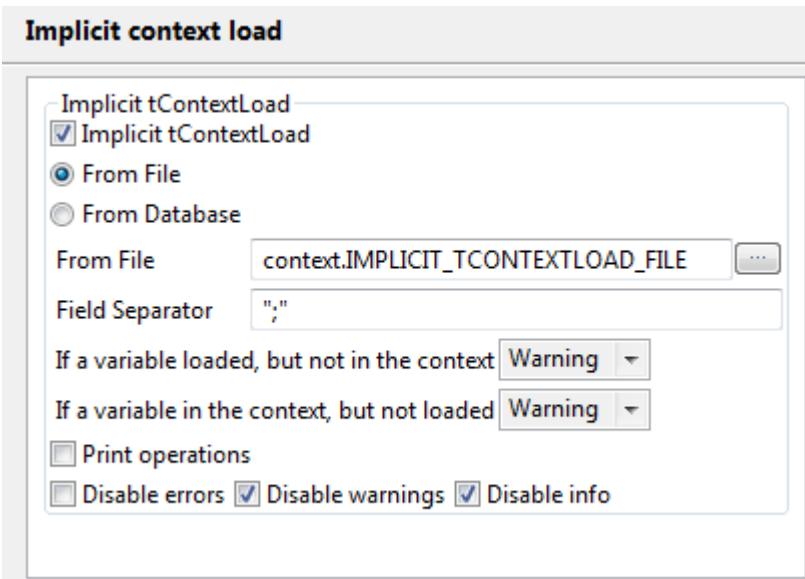
### About this task

You can define default context parameters you want to use in your Jobs.

To do so:

### Procedure

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 and Close** to apply your changes and close the dialog box.

### Applying Project Settings

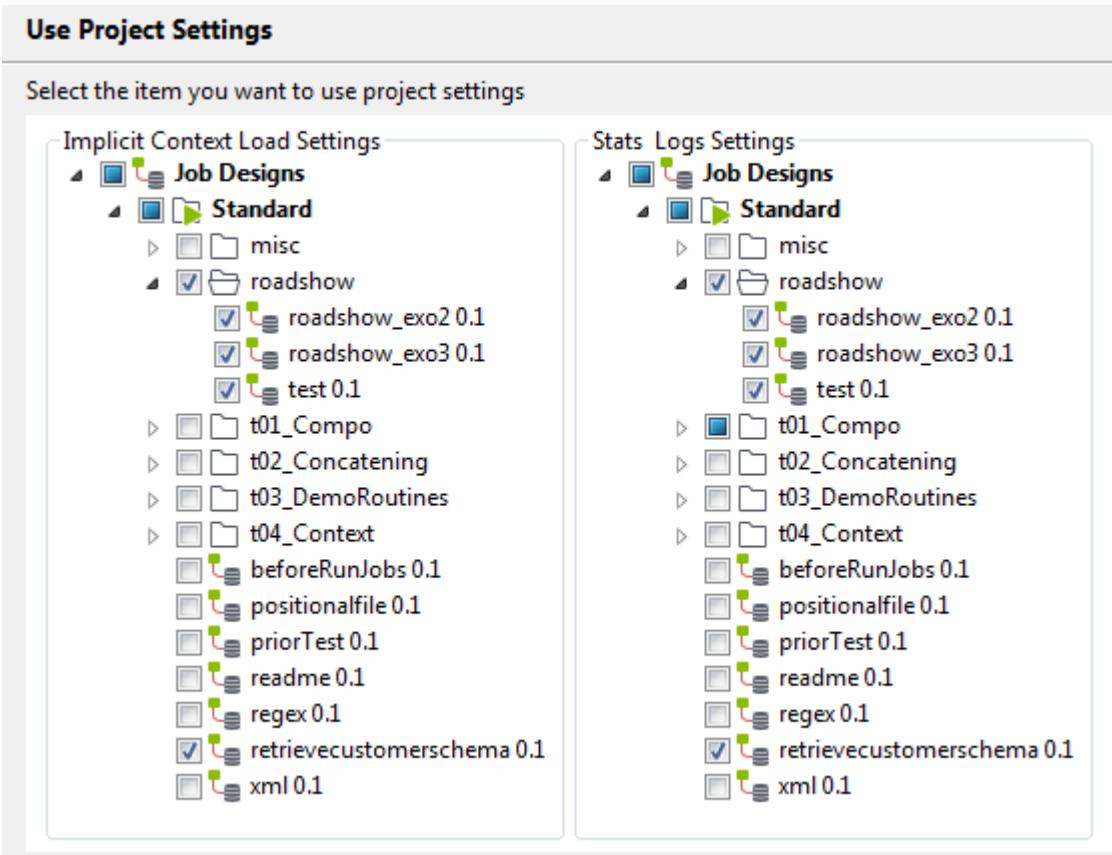
### About this task

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:

## Procedure

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.



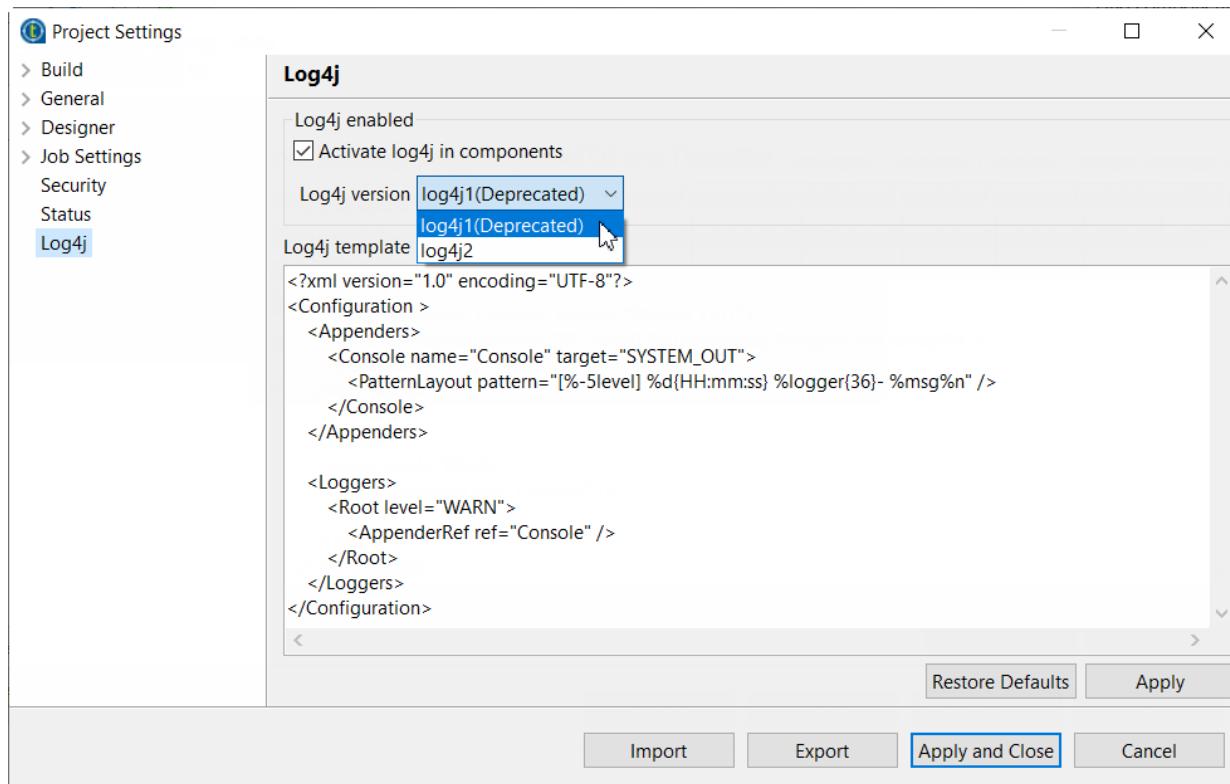
3. In the **Implicit Context Load Settings** area, select the check boxes corresponding to the Jobs in which you want to use the implicit context load option.
4. In the **Stats Logs Settings** area, select the check boxes corresponding to the Jobs in which you want to use the stats and logs option.
5. Click **Apply and Close** to apply your changes and close the dialog box.

## Configuring Log4j

Talend Studio includes the Apache logging utility Log4j to provide logging at runtime. You can enable or disable Log4j loggers in components and customize the Log4j configuration globally for the project.

## Procedure

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 **Log4j** node to open the **Log4j** view.



3. Select the **Activate log4j in components** check box to activate the Log4j feature.

By default, the Log4j feature is deactivated when a project is created.

When a project is imported from Talend 7.2 or an earlier version, which only supports Log4j 1, the deprecated Log4j 1 is selected from the **Log4j version** drop-down list by default. All POM files for the project will be synchronized when you change the Log4j version.

4. If needed, change the global Log4j configuration by modifying the XML instructions in the **Log4j template** area.

For example, to configure the root logger for Log4j 2 to output all debug or higher messages, go to the `Loggers` section and set the value of the `level` attribute of the `root` node to `debug`.

For more information about Log4j 2 configuration, see <http://logging.apache.org/log4j/2.x/manual/configuration.html#XML>.

For more information about Log4j 1 configuration, see <https://cwiki.apache.org/confluence/display/LOGGINGLOG4J/Log4jXmlFormat>.

## Status settings

### About this task

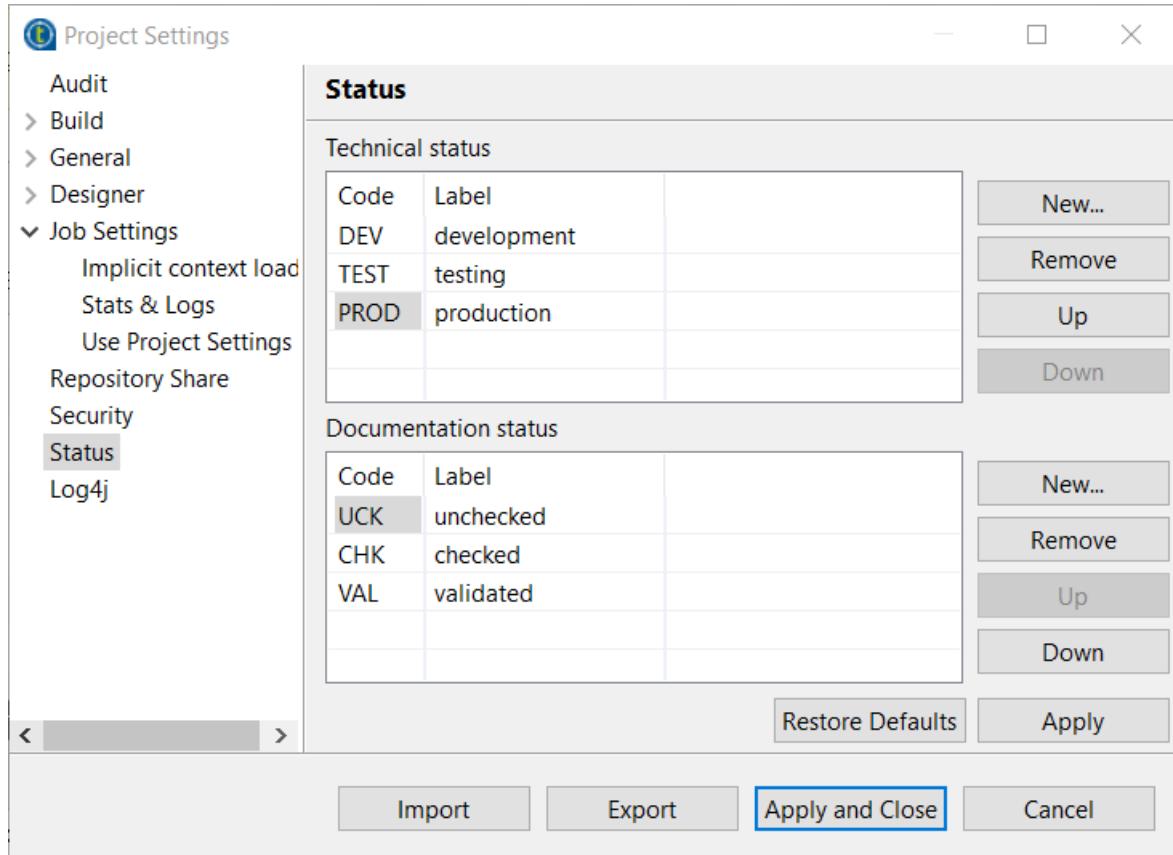
In the **Project Settings** dialog box, you can also define the Status.

To do so:

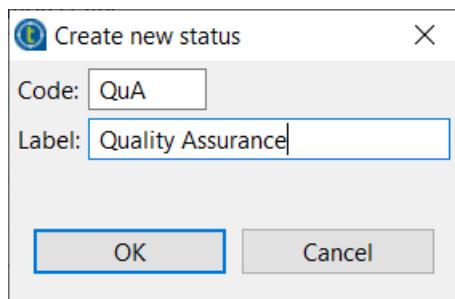
### Procedure

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 **Apply and Close** to close the dialog box.

## Security settings

### About this task

You can hide or show your passwords on your documentations, metadata, contexts, and so on when they are stored in the **Repository** tree view.

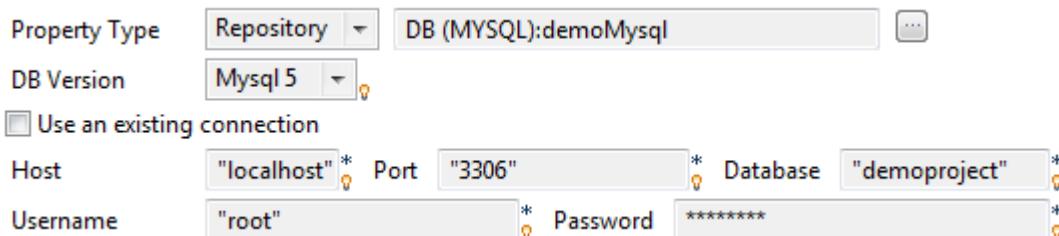
To hide your password:

### Procedure

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.

#### Note:

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 **Apply and Close** to close the dialog box.

## 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.

#### Note:

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/>.

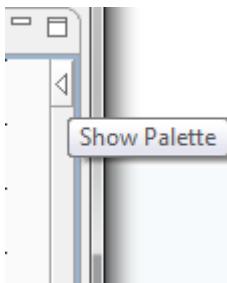
### Changing the Palette layout and settings

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**.

### Showing, hiding the Palette and changing its position

By default, the **Palette** might be hidden on the right hand side of your design workspace.



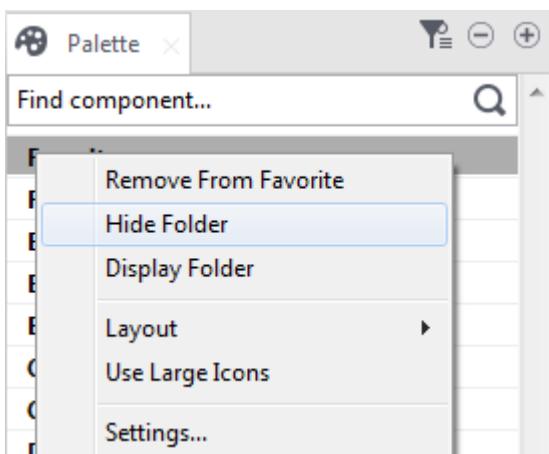
If you want the **Palette** to show permanently, click the left arrow, at the upper right corner of the design workspace, to make it visible at all times.

You can also move around the **Palette** outside the design workspace within the **Integration** perspective. To enable the standalone **Palette** view, select from the menu **Window > 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.

### Displaying/hiding 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.

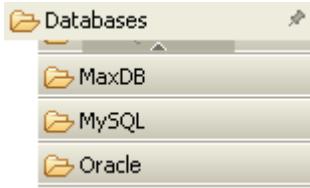


#### Note:

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 [Setting the Palette favorite on page 393](#).

## Maintaining 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.

## Filtering 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](#) on page 375.

For more information about adding components to the **Palette**, either from **Talend** or from your own development, see [Downloading/uploading Talend Community components](#) on page 53 and/or [How to define the user component folder \(Talend > Components\)](#) on page 408.

## Setting the Palette favorite

### About this task

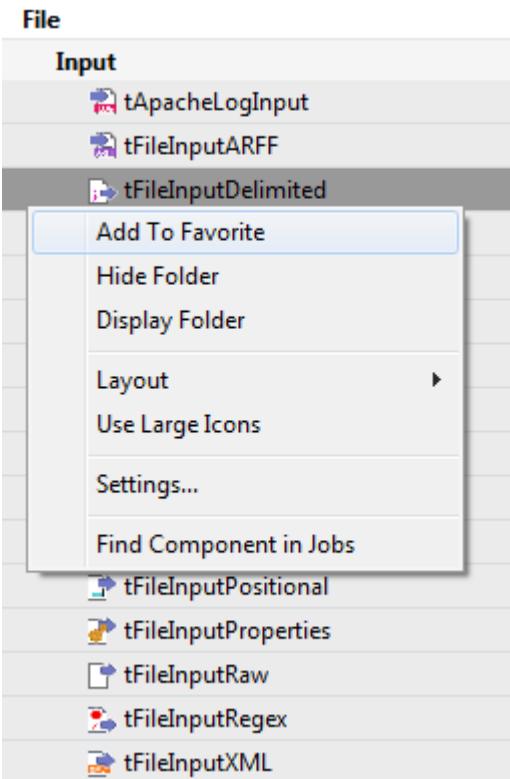
The **Palette** offers you search and favorite possibilities that by turn facilitate its usage.

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.

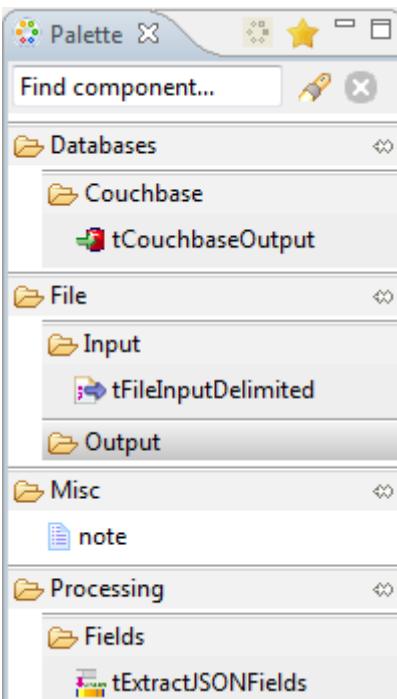
To do so:

### Procedure

1. From the **Palette**, right-click the component you want to add to **Palette** favorite and select **Add To Favorite**.



- 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**.

## Changing components layout in the Palette

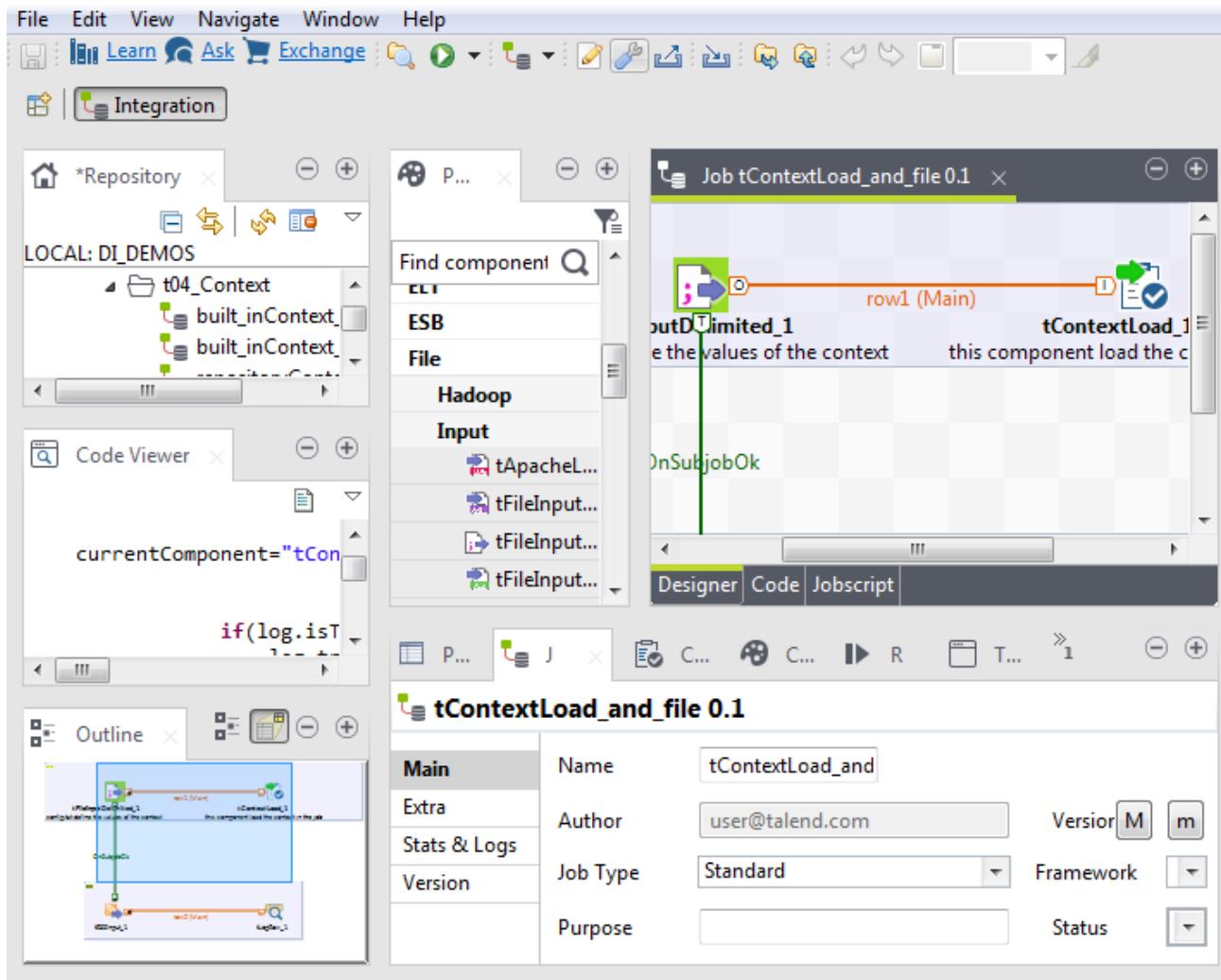
You can change the layout of the component list in the **Palette** to display them in columns or in lists, as icons only or as icons with short description.

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.

## Changing panels positions

All panels in the open Studio can be moved around according to your needs.



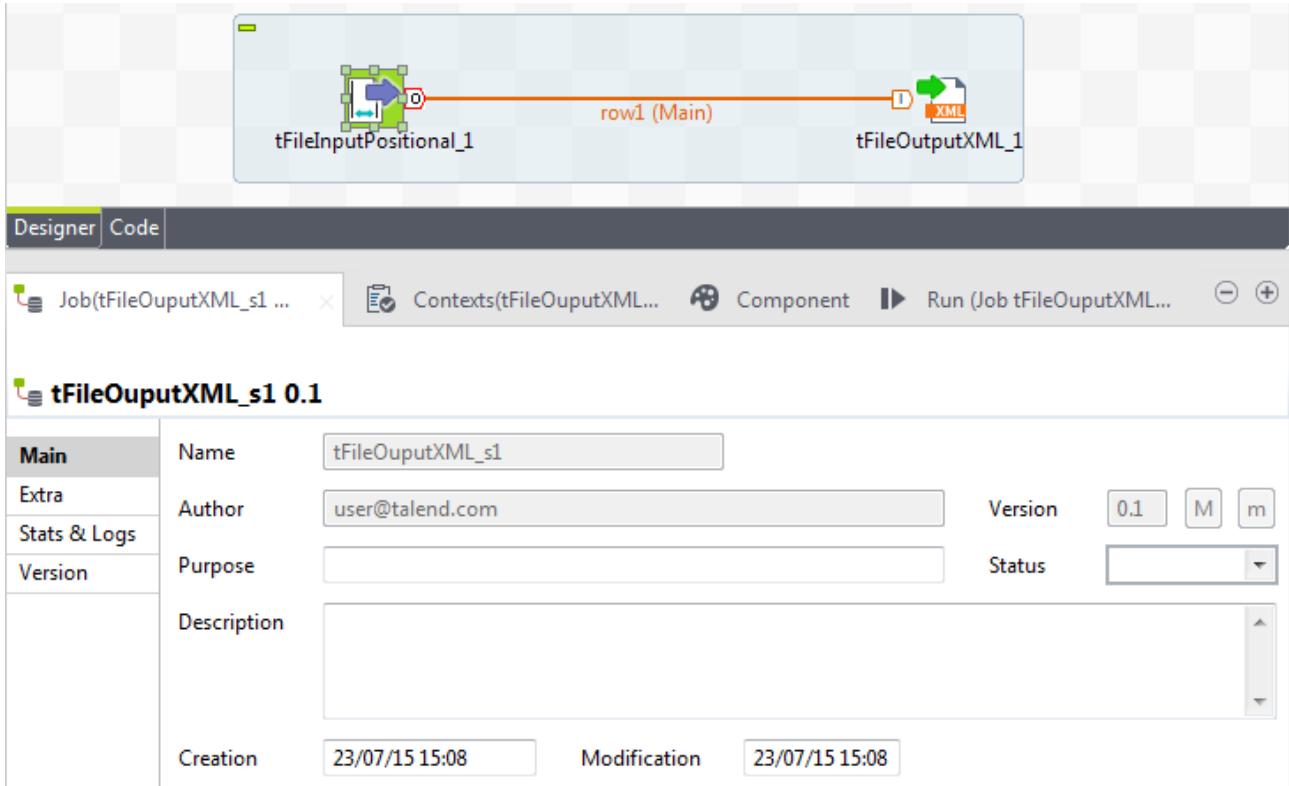
## Procedure

- Click the head border of a panel or click a tab, hold down the mouse button and drag the panel to the target destination, and then 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 [Displaying Job configuration tabs/views](#) on page 396.
- 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.

- 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.

### Displaying 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.

**Note:**

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 **Schedulerdeprecated** 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 **Schedulerdeprecated** 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.

## 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.

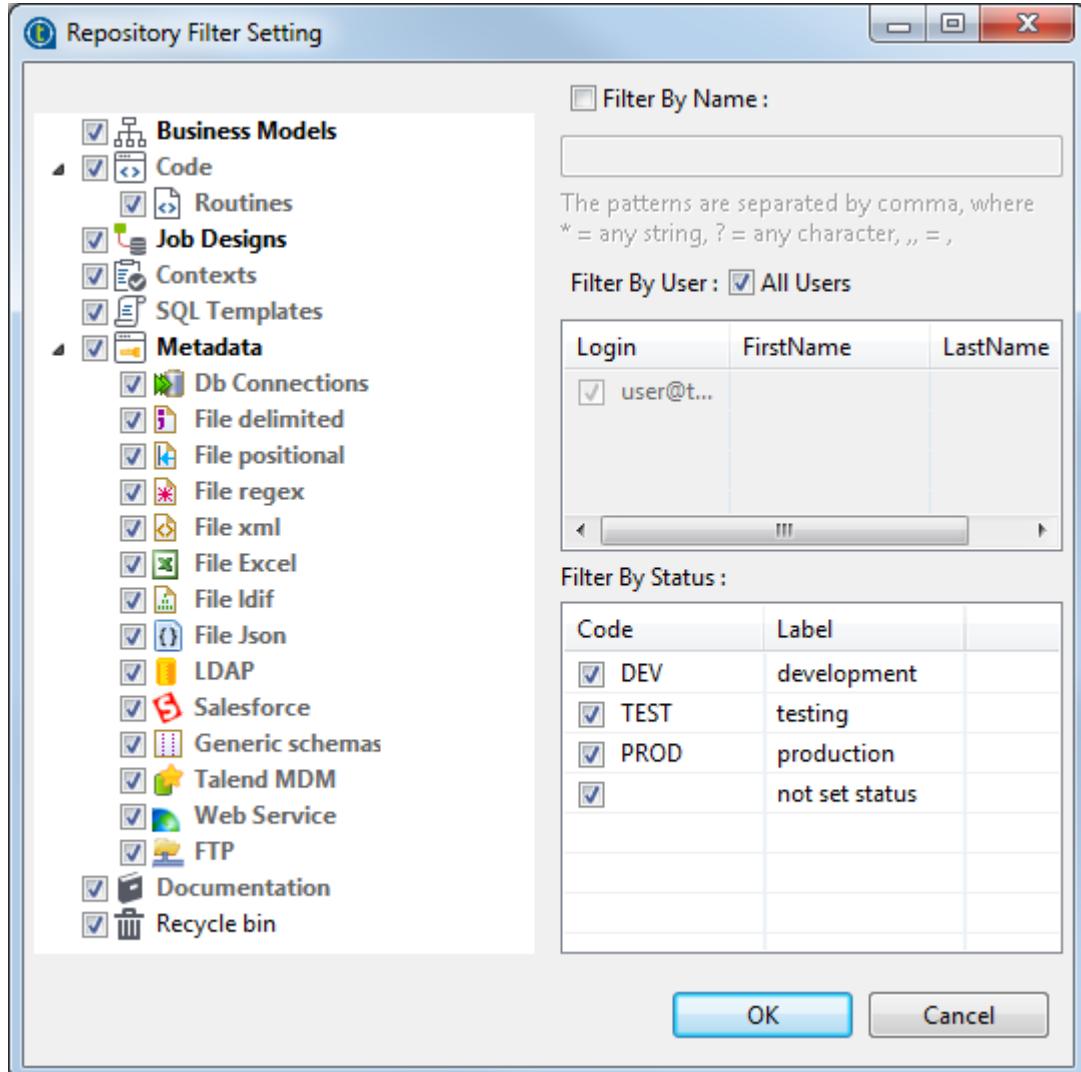
### Filtering by Job name

#### About this task

To filter Jobs listed in the **Repository** tree view by Job name, complete the following:

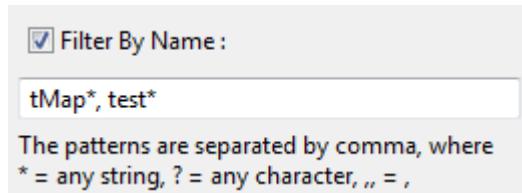
#### Procedure

1. In the Studio, click the  icon in the upper right corner of the **Repository** tree view and select **Filter Setting...** from the contextual menu.  
The **Repository Filter Setting** dialog box displays.

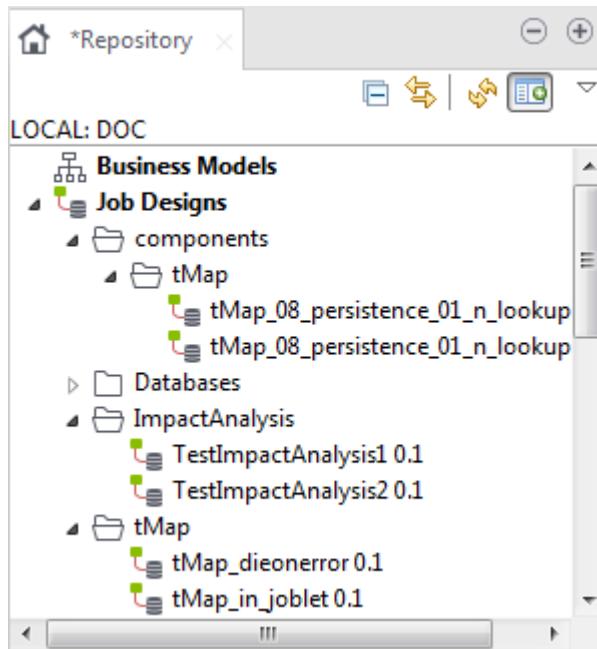


2. Select the **Filter By Name** check box.

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.



## Results

### Note:

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 ().

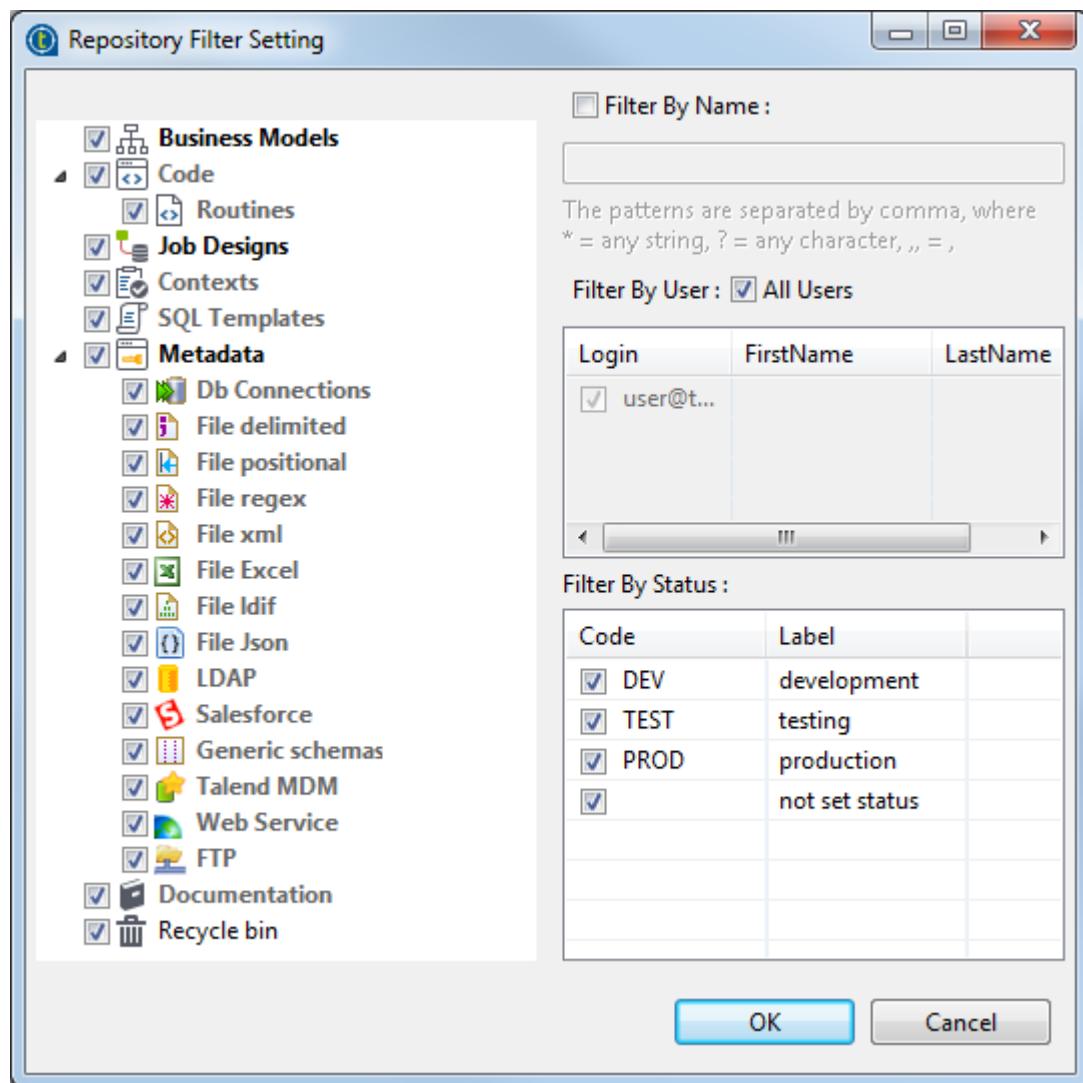
## Filtering by user

### About this task

To filter entries in the **Repository** tree view by the user who created the Jobs/items, complete the following:

### Procedure

1. In the Studio, click the icon in the upper right corner of the **Repository** tree view and select **Filter Setting...** from the contextual menu.  
The **Repository Filter** dialog box displays.



**2.** Clear the **All Users** check box.

The corresponding fields in the table that follows become available.

**Filter By User :**  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.

- 3.** 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.
- 4.** 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.

## Results

### Note:

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 ().

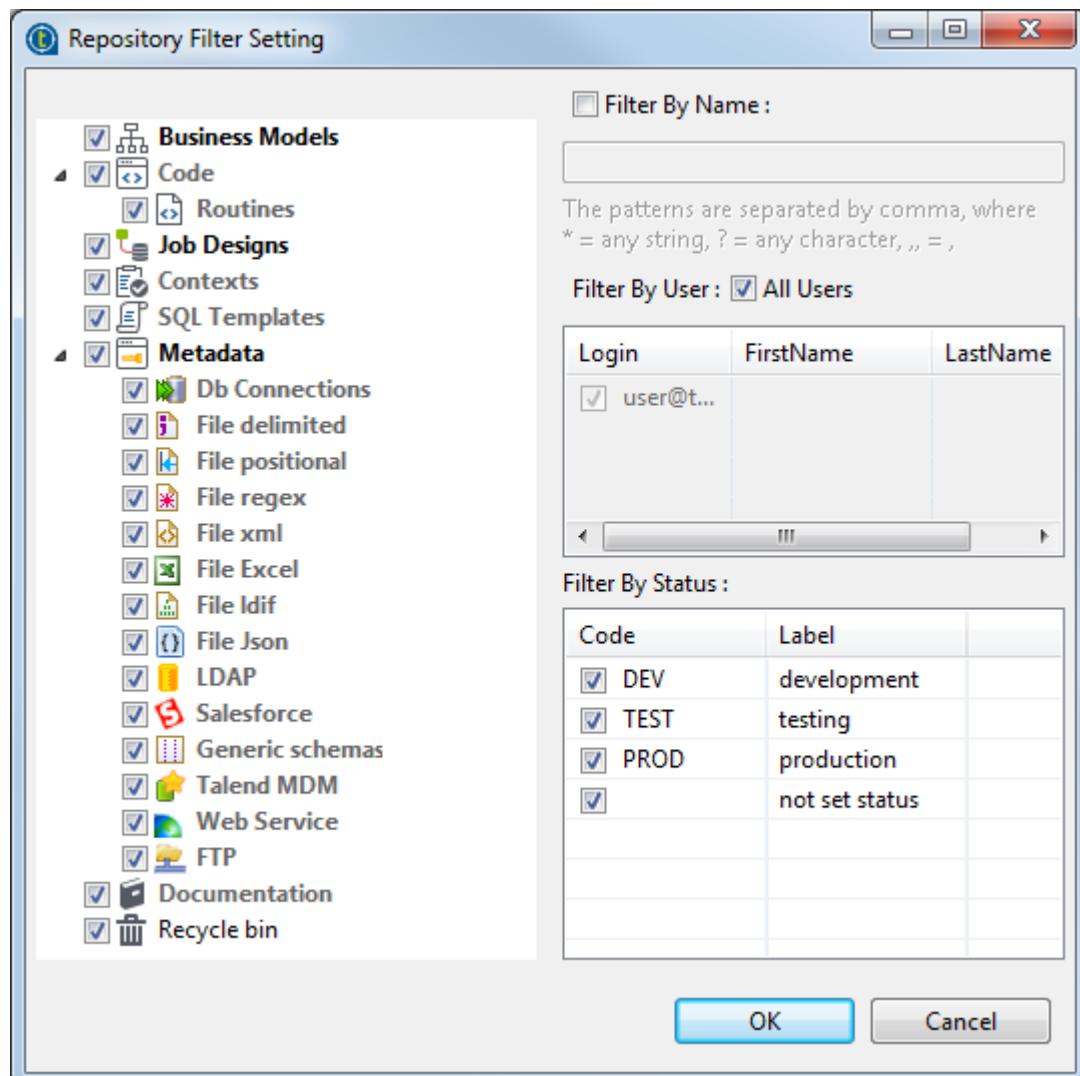
## Filtering by Job status

### About this task

To filter Jobs in the **Repository** tree view by the job status, complete the following:

### Procedure

- In the Studio, click the icon in the upper right corner of the **Repository** tree view and select **Filter Setting...** 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.

3. Click **OK** to validate your changes and close the dialog box.  
All Jobs that have the specified status will disappear from the tree view.

## Results

### Note:

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 ().

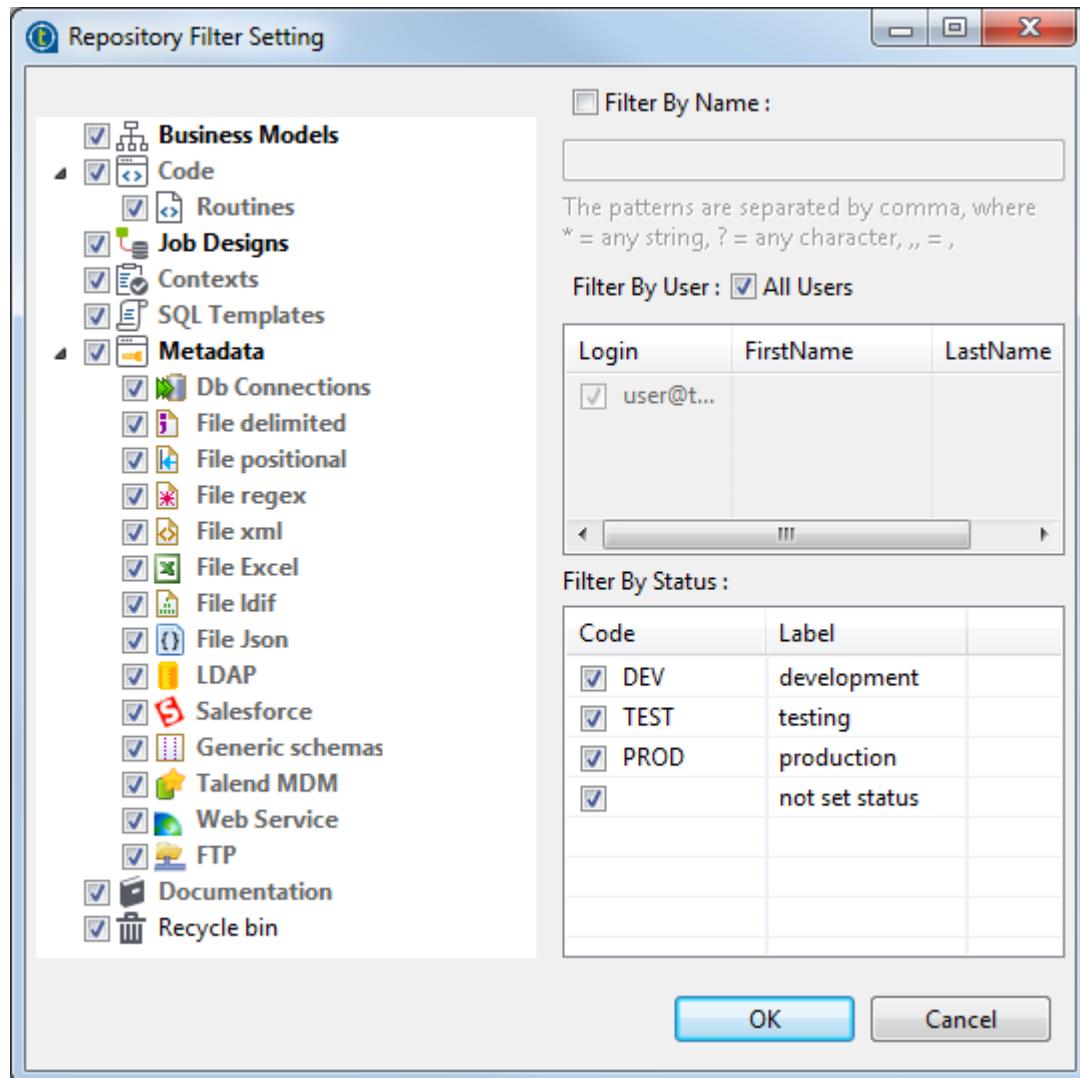
## Choosing what repository nodes to display

### About this task

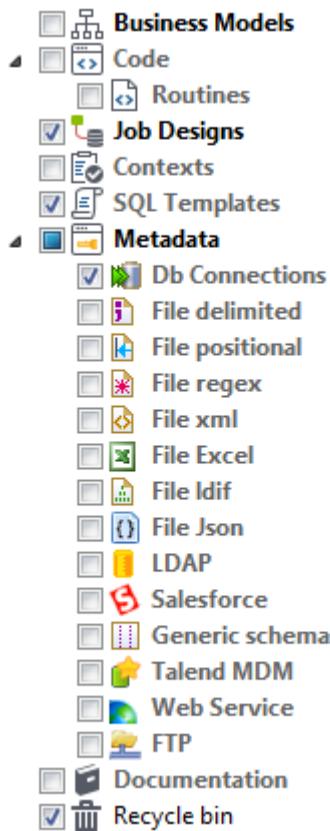
To filter repository nodes, complete the following:

### Procedure

1. In the **Integration** perspective of the Studio, click the icon in the upper right corner of the **Repository** tree view and select **Filter Setting...** from the contextual menu.  
The **Repository Filter** dialog box displays.



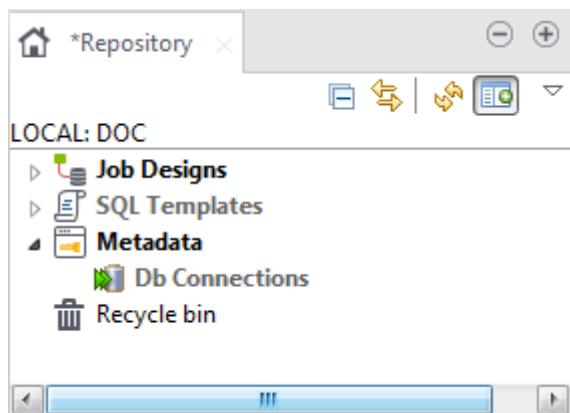
2. Select the check boxes next to the nodes you want to display in the **Repository** tree view.



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.



## Results

### Note:

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 [Filtering by Job name](#) on page 397.

## 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.

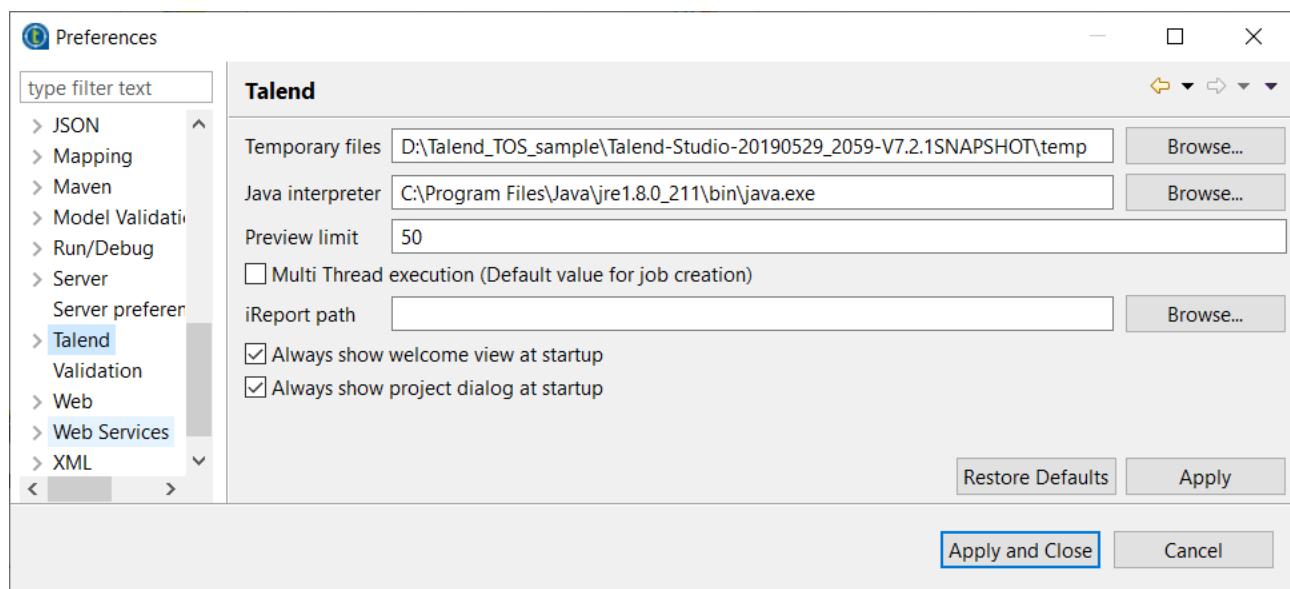
The following sections describe specific settings that you can set as preference.

First, click the **Window** menu of Talend Studio, then select **Preferences**.

### Java Interpreter path (Talend)

#### About this task

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:

#### Procedure

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.

#### Results

On the same view, you can also change the preview limit and the path to the temporary files or the OS language.

### Designer preferences (Talend > Appearance)

#### About this task

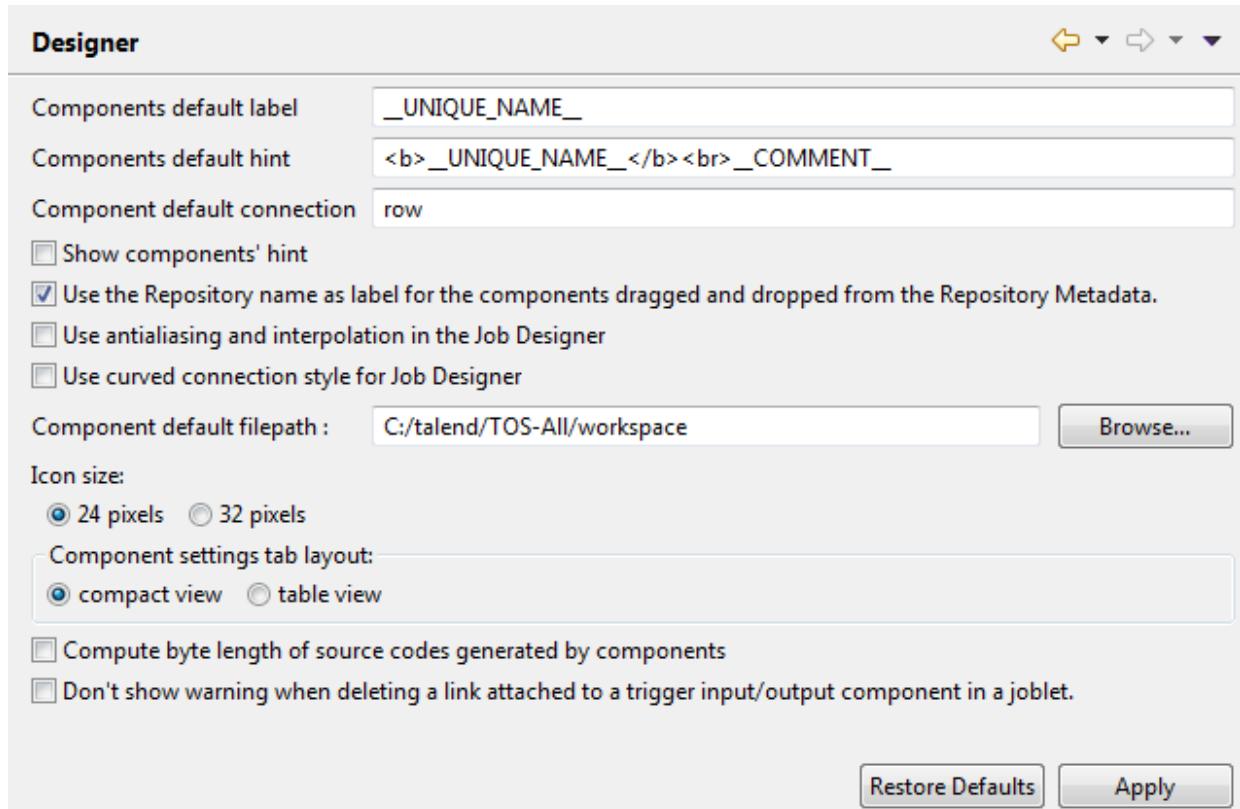
You can set component and Job design preferences to let your settings be permanent in the Studio.

#### Procedure

1. From the menu bar, click **Window > Preferences** to open the **Preferences** dialog box.

2. Expand the **Talend > Appearance** nodes.
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.

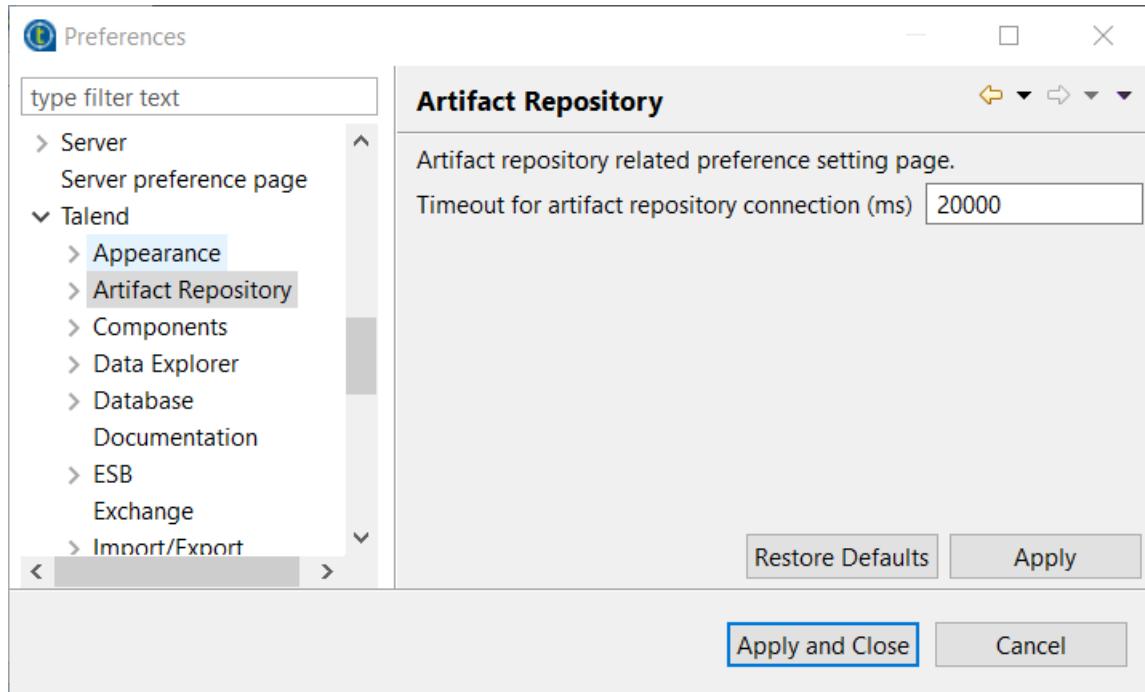
### [Artifact repository connection preferences \(Talend > Artifact Repository\)](#)

#### About this task

You can configure how long your Talend Studio keeps connection with the Artifact repository, from which your Talend Studio retrieves updates and custom libraries and to which you can publish your artifacts.

#### Procedure

1. From the menu bar, click **Window > Preferences** to open the **Preferences** dialog box.
2. Expand the **Talend** node and click **Artifact Repository** to display the relevant view.



3. In the **Timeout for artifact repository connection (ms)** field, specify the time in milliseconds you want your Talend Studio to wait for an interaction with the Artifact repository server before cutting the connection, 0 for an infinite timeout.
4. Click **Apply** to apply your changes; click **Apply and Close** to validate the settings and close the **Preferences** dialog box.

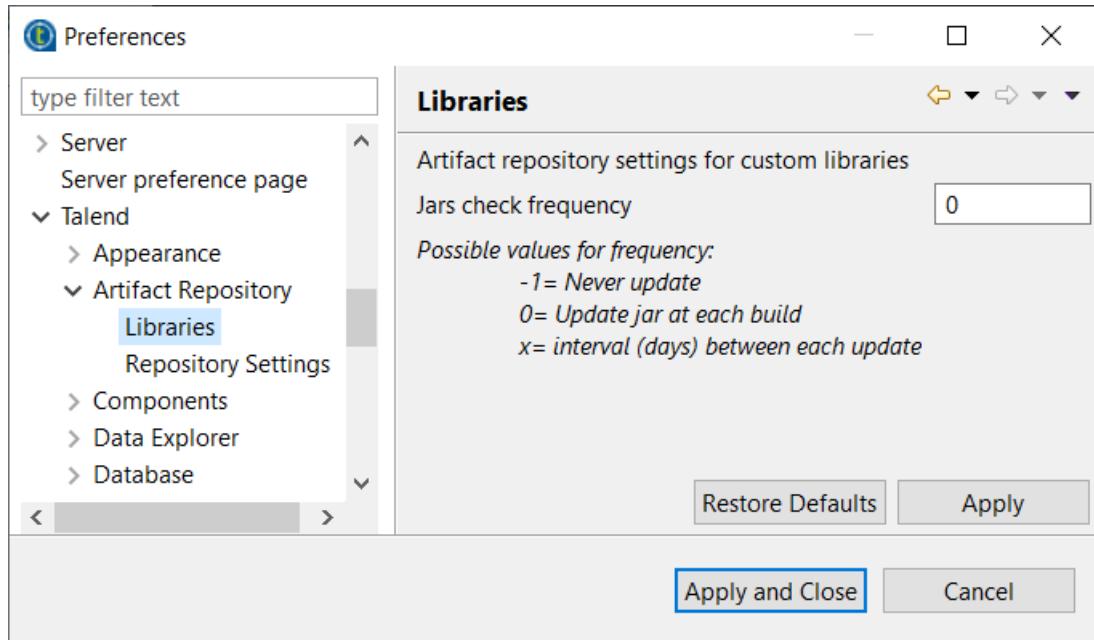
#### [Update server preferences \(Talend > Artifact Repository > Libraries\)](#)

##### **About this task**

You can set preferences for Talend Studio to the check for updates of custom libraries on the Artifact repository server.

##### **Procedure**

1. From the menu bar, click **Window > Preferences** to open the **Preferences** dialog box.
2. Expand the **Talend** and **Artifact Repository** nodes in succession and then click **Libraries** to display the relevant view.



3. Set the preferences according to your needs:
  - In the **Jars check frequency** field, specify how often you want your Talend Studio to check for updates:
    - -1 if you don't want your Talend Studio to check for updates at all.
    - 0 if you want your Talend Studio to check for updates at any action that needs a Jar or Jars, for example when a Job is built or executed from the Studio.
    - the number of days between two checks.
4. Click **Apply** to apply your changes; click **Apply and Close** to validate the settings and close the **Preferences** dialog box.

#### [Bonita BPM Portal preferences \(Talend > Bonita BPM Portal\)](#)

##### About this task

When creating a BPM service, you can set its URI as well as the connection information to the Bonita BPM Portal.

##### Procedure

1. From the menu bar, click **Window > Preferences** to open the **Preferences** dialog box.
2. Expand the **Talend > Bonita BPM Portal** node.
3. Fill in the information as follows.

Field Name	Action
Username and Password	Enter the username and password to connect to the Bonita BPM Portal. By default, it is <code>install</code> and <code>install</code> .
Address	Enter the URL of the Bonita BPM Portal server. By default, it is <code>http://localhost:8040/bonita</code> .

4. Click **Apply** and then **Apply and Close** to validate the set preferences and close the dialog box.

## How to define the user component folder (Talend > Components)

A new framework is available to build custom components and must be used from this version onwards. Refer to *Developing a component using Talend Component Kit* on Talend Help Center (<https://help.talend.com>). This document still applies for components downloaded from **Talend Exchange**.

You can download and install custom components for use in the **Integration** perspective of Talend Studio.

For further information about downloading and installing components from **Talend Exchange**, see *How to install and update a custom component* on Talend Help Center (<https://help.talend.com>).

The following procedure applies only to the external components.

The user component folder is the folder that contains the components you created and/or the ones you downloaded from **Talend Exchange**. To define it, proceed as follows:

### Procedure

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** to validate the preferences. You can also click **Apply and Close** to validate the preferences and close the dialog box.

The Studio restarts and the external components are added to the **Palette**.

### Results

This configuration is stored in the metadata of the workspace. If the workspace of Talend Studio changes, you have to reset this configuration again.

## How to change specific component settings (Talend > Components)

### About this task

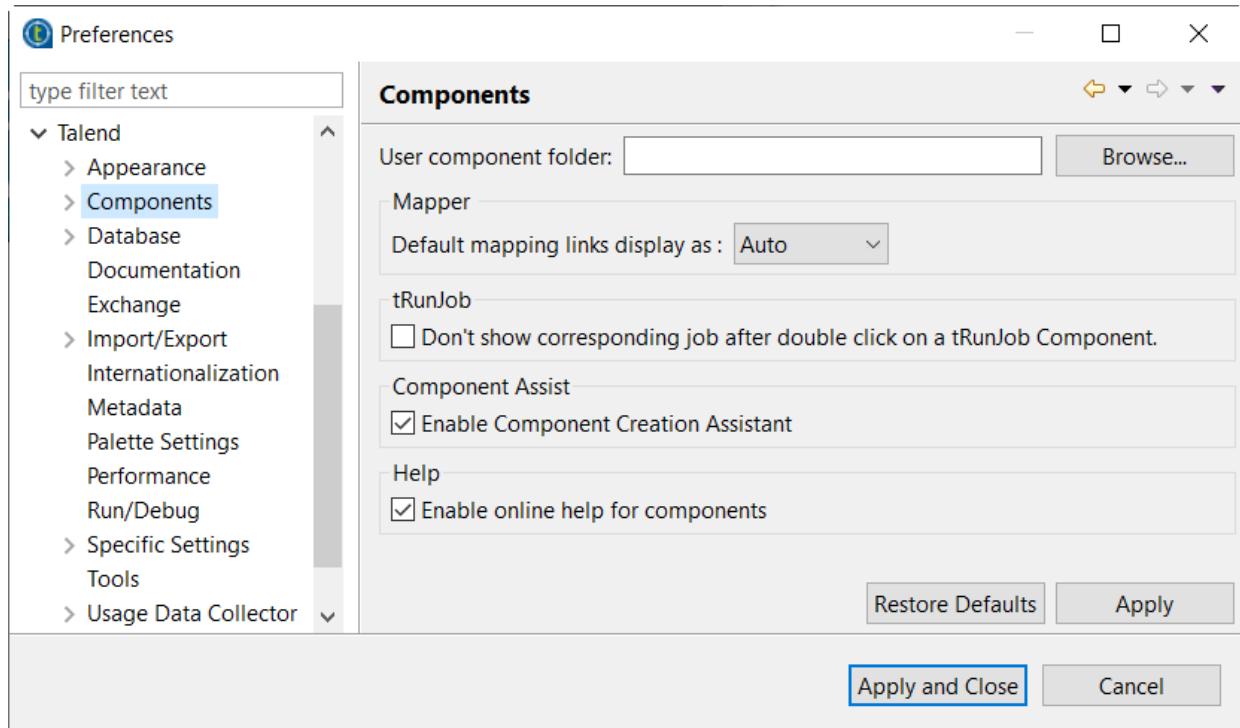
You can modify some specific component settings such as the default mapping link display.

The following procedure applies to the external components and to the components included in the Studio.

To modify those specific components settings, proceed as follows:

### Procedure

1. In the tree view of the **Preferences** dialog box, expand the **Talend** node and select **Components**.



**2.** Set your preferences as needed.

- From the **Default mapping links display as** list, select the mapping link type you want to use in the **tMap**.
- Under **tRunJob**, select the check box if you do not want the corresponding Job to open upon double clicking a **tRunJob** component.

**Note:** You will still be able to open the corresponding Job by right clicking the **tRunJob** component and selecting **Open tRunJob Component**.

- 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](#) on page 27.
- Select the **Enable online help for components** check box if you want to enable online help which shows you our latest content on Talend Help Center (<https://help.talend.com>). Clear the check box if you want to enable embedded help. By default, the check box is selected and online help is enabled.

**3.** Click **Apply** to validate the set preferences. You can also click **Apply and Close** to validate the set preferences and close the dialog box.

## Results

This configuration is stored in the metadata of the workspace. If the workspace of Talend Studio changes, you have to reset this configuration again.

### Documentation preferences (Talend > Documentation)

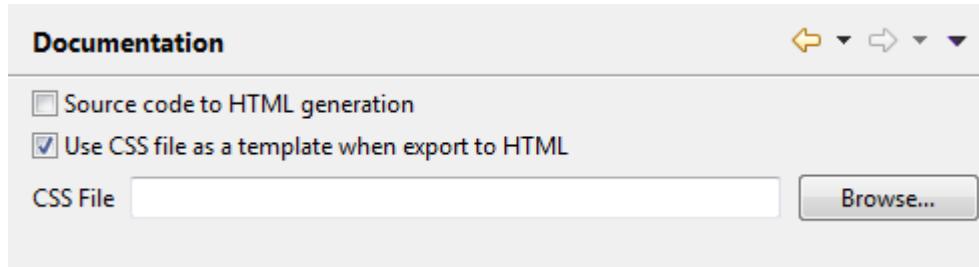
#### About this task

You can include the source code on the generated documentation.

#### Procedure

- 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.

## Results

For more information on documentation, see [Generating HTML documentation](#) on page 115 and [Documentation tab](#) on page 47.

### Exchange preferences (Talend > Exchange)

#### Before you begin

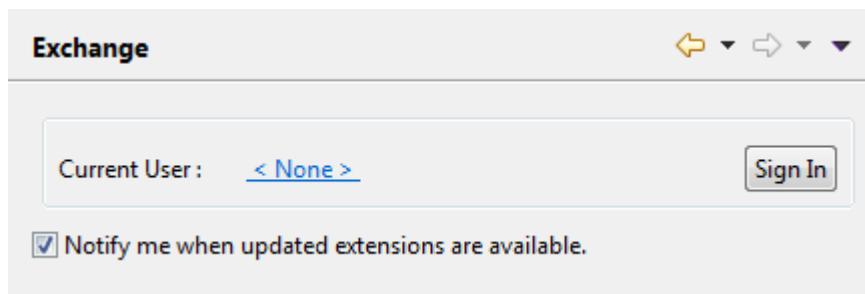
**Warning:** Make sure that the `-Dtalend.disable.internet` parameter is not present in the Studio .ini file or is set to false.

#### About this task

You can set preferences related to your connection with **Talend** Exchange, which is part of the **Talend** Community, in Talend Studio. To do so:

#### Procedure

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 Talend Community](#) 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.

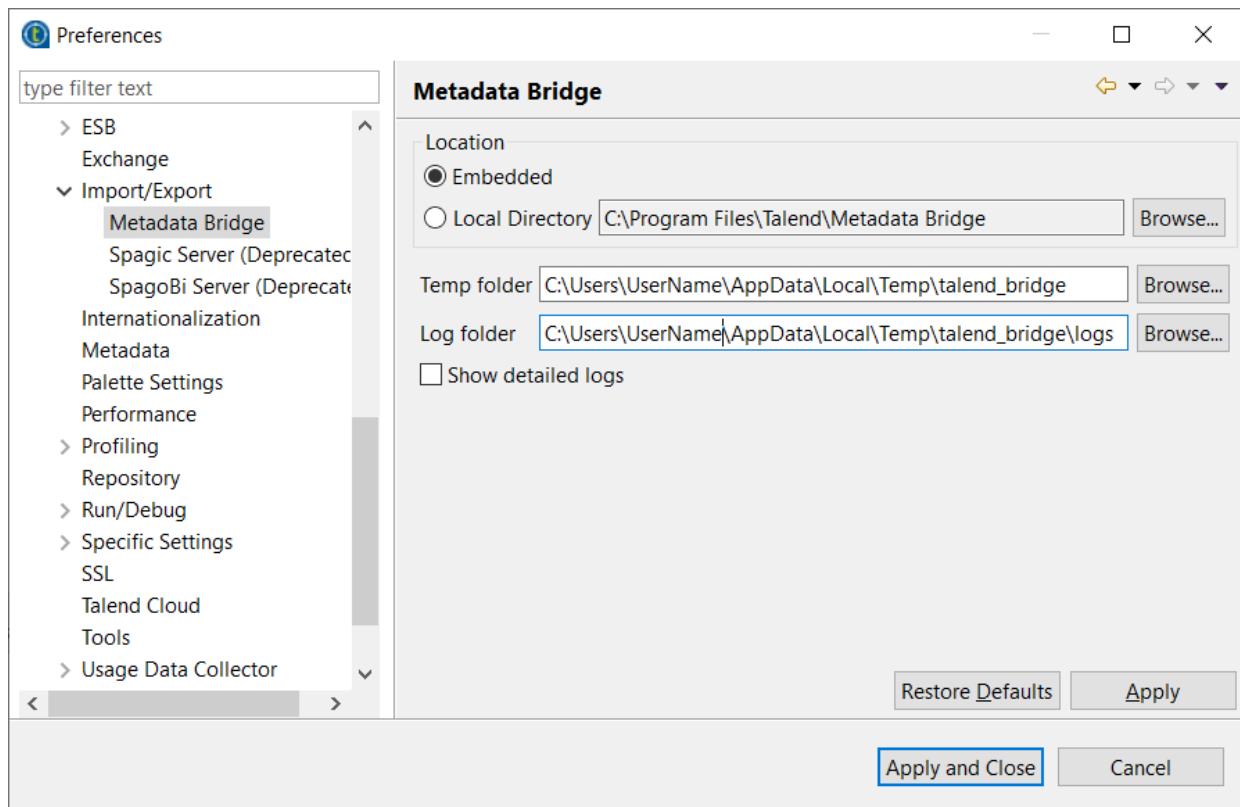
## Metadata Bridge preferences (Talend > Import/Export)

### About this task

You can set preferences for the **Talend** Metadata Bridge to make it work the way you want.

### Procedure

1. From the menu bar, click **Window > Preferences** to display the **Preferences** dialog box.
2. Expand the **Talend** and **Import/Export** nodes in succession and then click **Metadata Bridge** to display the relevant view.



3. Set the preferences according to your use of the **Talend** Metadata Bridge:

- In the **Location** area, select the **Embedded** option to use the MIMB tool embedded in the **Talend** Metadata Bridge. This is the default option.

To use the MIMB tool you have installed locally, select **Local Directory** and specify the installation directory of the MIMB tool.

- In the **Temp folder** field, specify the directory to hold the temporary files generated during metadata import/export executions, if you do not want to use the default directory.
- In the **Log folder** field, specify the directory to hold the logs files generated during metadata import/export executions, if you do not want to use the default directory.

- Select the **Show detailed logs** check box to generate detailed log files during metadata import/export executions.
4. Click **Apply** to apply your changes; click **Apply and Close** to validate the settings and close the **Preferences** dialog box.

## Results

For more information on using the **Talend** Metadata Bridge to import/export metadata, see *Importing and exporting metadata using Talend Metadata Bridge* on Talend Help Center (<https://help.talend.com>).

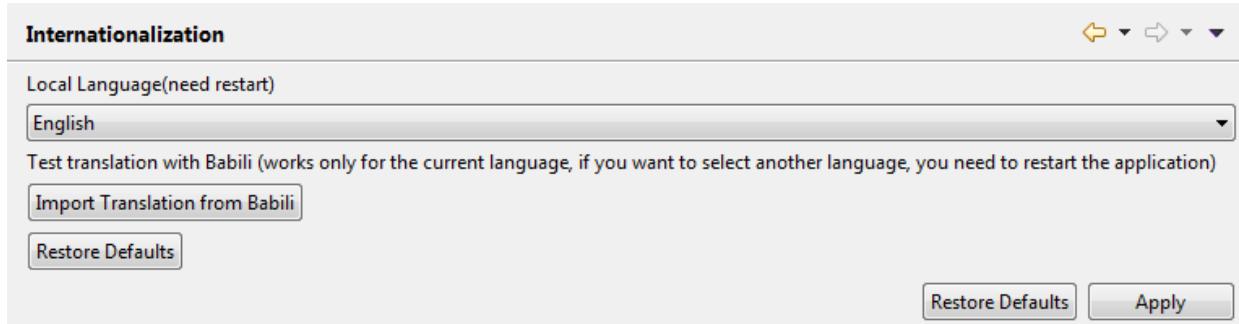
## Language preferences (Talend > Internationalization)

### About this task

You can set language preferences in Talend Studio. To do so:

### Procedure

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 **Apply and Close** to validate your change and close the **Preferences** dialog box.
5. Restart the Studio to display the graphical interface in the selected language.

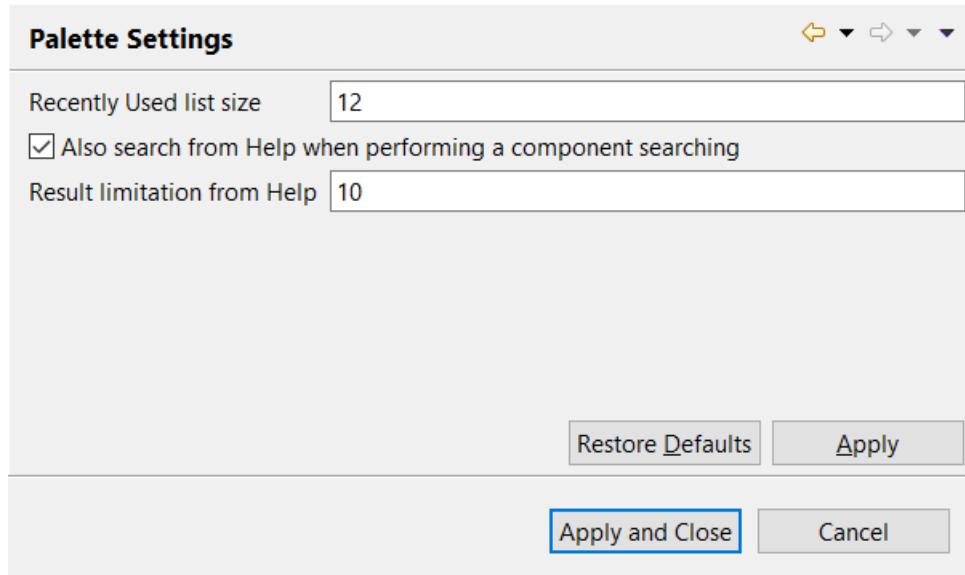
## Palette preferences (Talend> Palette Settings)

### About this task

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**.

### Procedure

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.



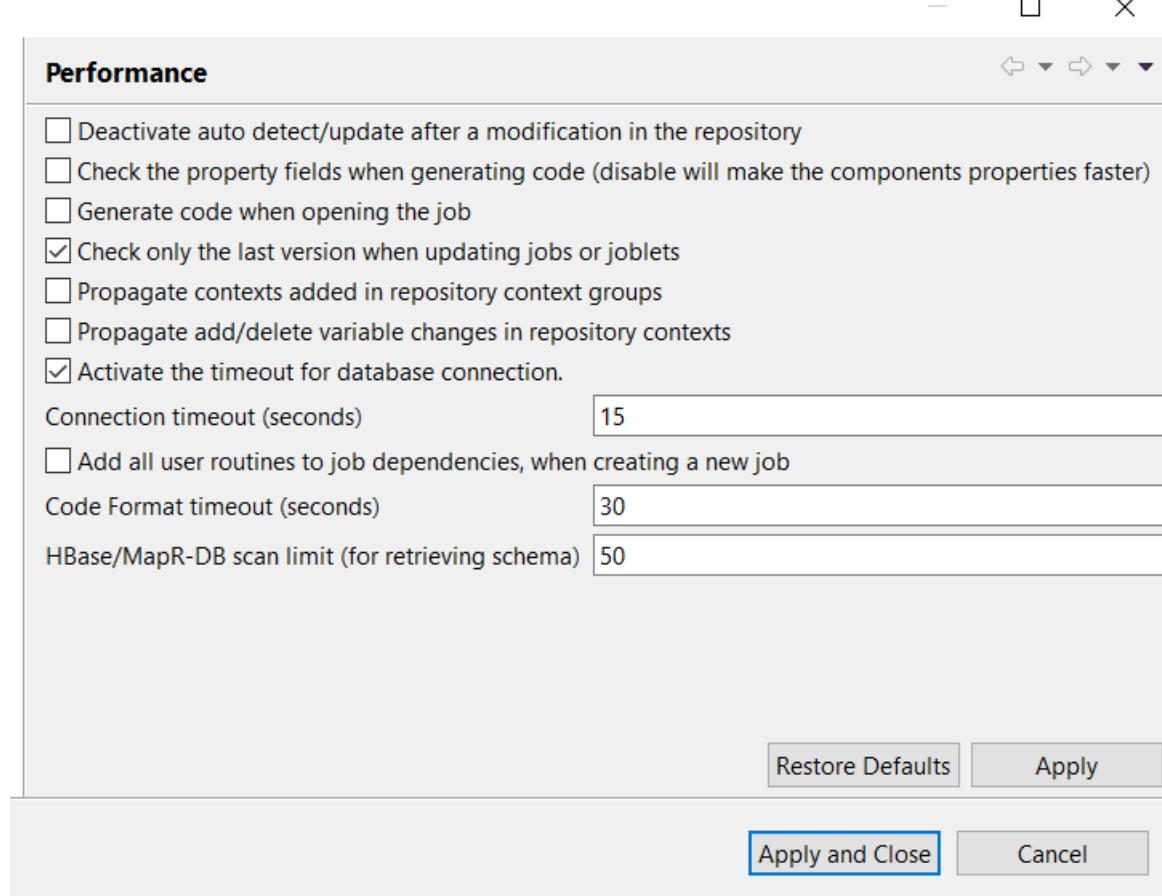
3. To limit number of components that can be displayed on the **Recently Used** list, enter your preferred number in the **Recently used list size** field.
4. To enable searching a component using a phrase that describes the function or purpose of the component as search keywords in the search field 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 key words, enter your preferred number in the **Result limitation from Help** field.

### Performance preferences (Talend > Performance)

You can set performance preferences according to your use of Talend Studio. To do so, proceed as follows:

#### Procedure

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.



**Note:** 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 field is not correctly filled in, the component is surrounded by red on the design workspace.

**Note:** 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 contexts added in repository context groups** check box to allow propagating contexts newly added in repository context groups to Jobs.

With this option enabled, each time you open a Job that uses a repository context group, you will see a dialog box asking you whether you want to perform a context propagation if any context has been added in the context group but not synchronized to the Job yet.

This option is disabled by default.

- Select the **Propagate add/delete variable changes in repository contexts** check box to allow propagating 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.
- In the **Code Format timeout (seconds)** field, specify the number of seconds in which you want your Talend Studio to stop formatting the source code upon code generation, for example when you switch from the **Designer** view to the **Code** view or when you build a Job. The value must be an integer greater than 0. Setting a small timeout value helps prevent performance issues at the price of lower readability of the source code, especially for a large, complex Job.

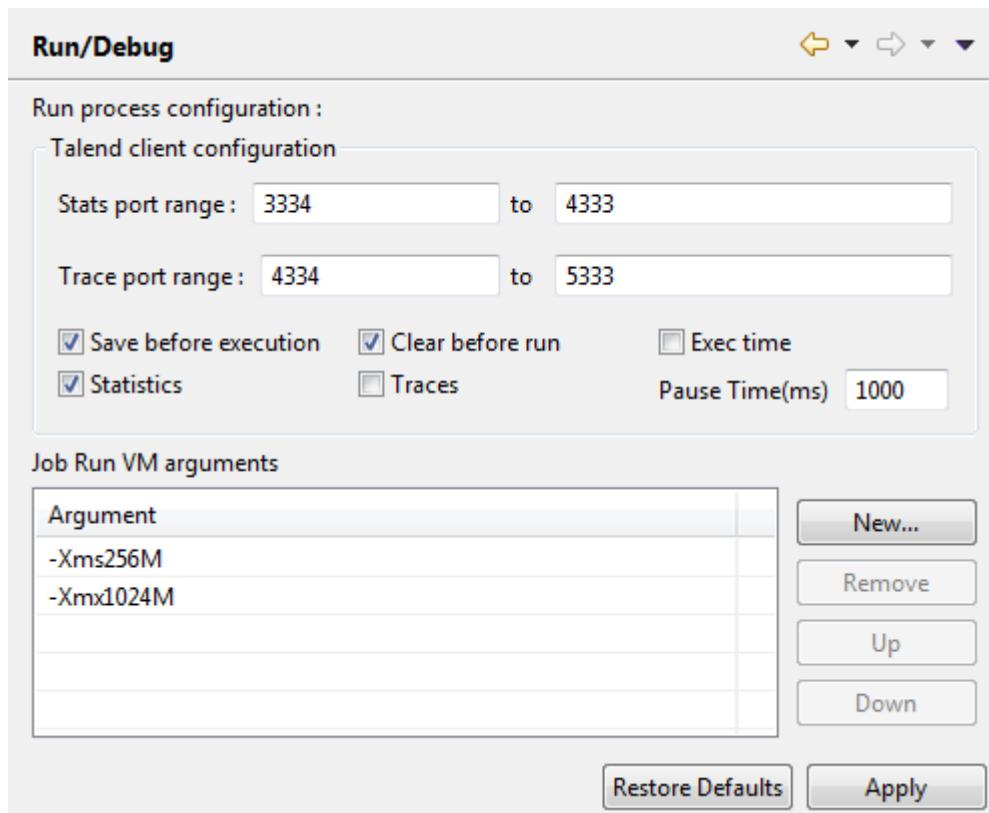
## Debug and Job execution preferences (Talend > Run/Debug)

### About this task

You can set your preferences for debug and job executions in Talend Studio. To do so:

### Procedure

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.



3. Set the parameters according to your needs.

- 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 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 [Setting advanced execution settings](#) on page 124.

For more information about possible parameters, check the site <http://www.oracle.com/technetwork/java/javase/tech/vmoptions-jsp-140102.html>.

## Displaying special characters for schema columns (Talend > Specific settings)

### About this task

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:

### Procedure

- From the menu bar, click **Window > Preferences** to open the **Preferences** dialog box.
- On the tree view of the opened dialog box, expand the **Talend** node.
- Click the **Specific settings** node to display the corresponding view on the right of the dialog box.
- Select the **Allow specific characters (UTF8,...) for columns of schemas** check box.



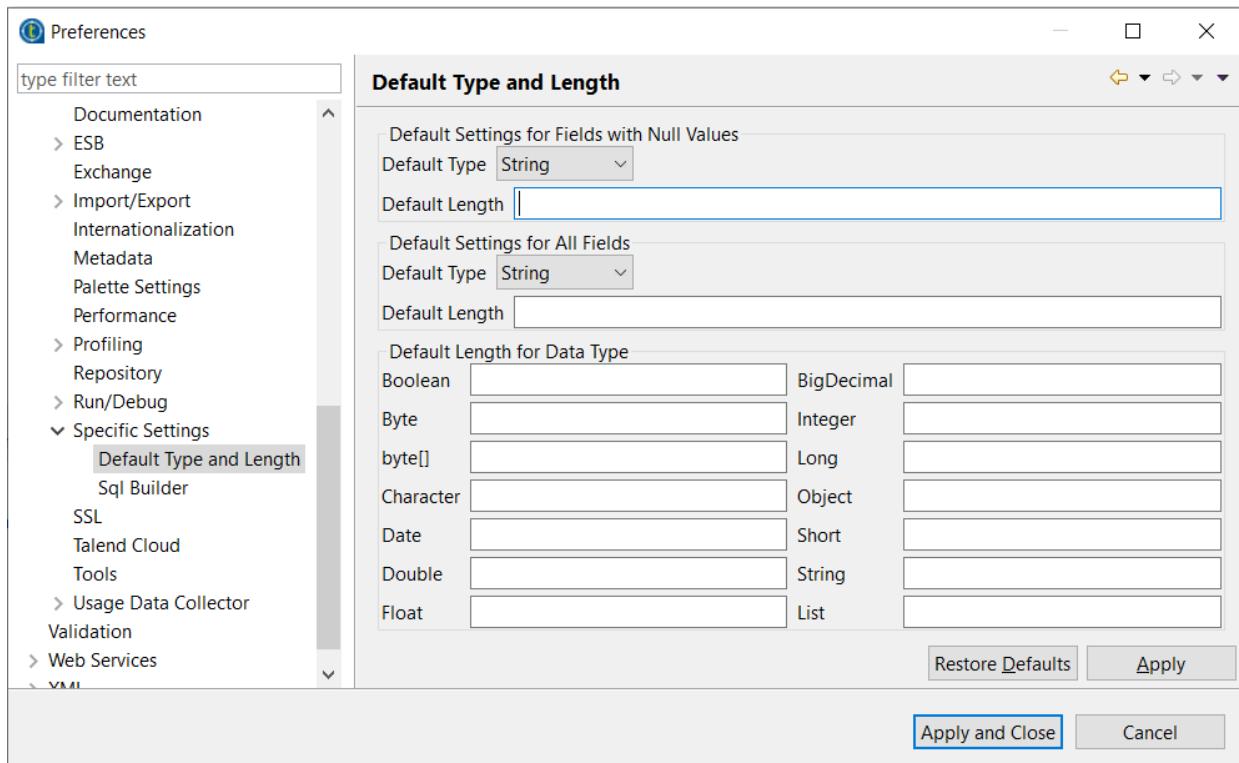
## Schema preferences (Talend > Specific Settings)

### About this task

You can define the default data length and type of the schema fields of your components.

### Procedure

- From the menu bar, click **Window > Preferences** to open the **Preferences** dialog box.
- Expand the **Talend** node, and click **Specific Settings > Default Type and Length** to display the data length and type of your schema.



- 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.

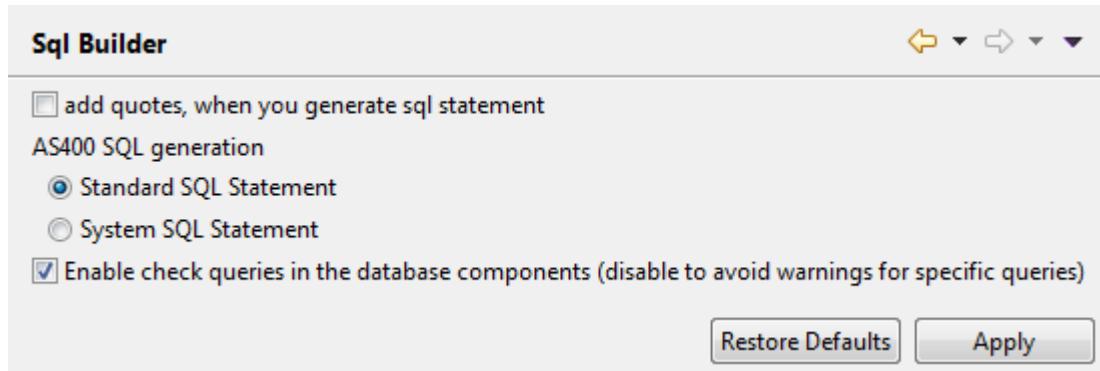
## SQL Builder preferences (Talend > Specific Settings)

### About this task

You can set your preferences for the SQL Builder. To do so:

### Procedure

- 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.



**3.** 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.

### [SSL settings preferences \(Talend> SSL\)](#)

#### **About this task**

You can set SSL preferences to configure your Talend Studio for secure communications with remote servers.

#### **Procedure**

1. From the menu bar, click **Window > Preferences** to display the **Preferences** dialog box.
2. Expand the **Talend** node and then click **SSL** to display the relevant view.
3. Define the Keystore Configuration for the local certificate to be sent to the remote host:
  - a) Click **Browse** next to the **Path** field and browse to the keystore file that stores your local credentials.
  - b) In the **Password** field, enter the keystore password.
  - c) From the **Keystore Type** list, select the type of keystore to use.
4. Define the Truststore Configuration for verification of the remote host's certificate:
  - a) Click **Browse** next to the **Path** field and browse to the truststore file.
  - b) In the **Password** field, enter the truststore password.
  - c) From the **Keystore Type** list, select the type of keystore to use.
5. Click **Apply** to apply your changes; click **Apply and Close** to validate the settings and close the **Preferences** dialog box.
6. Restart your Talend Studio for the configurations to take effect.

## Usage Data Collector preferences (Talend > Usage Data Collector)

### About this task

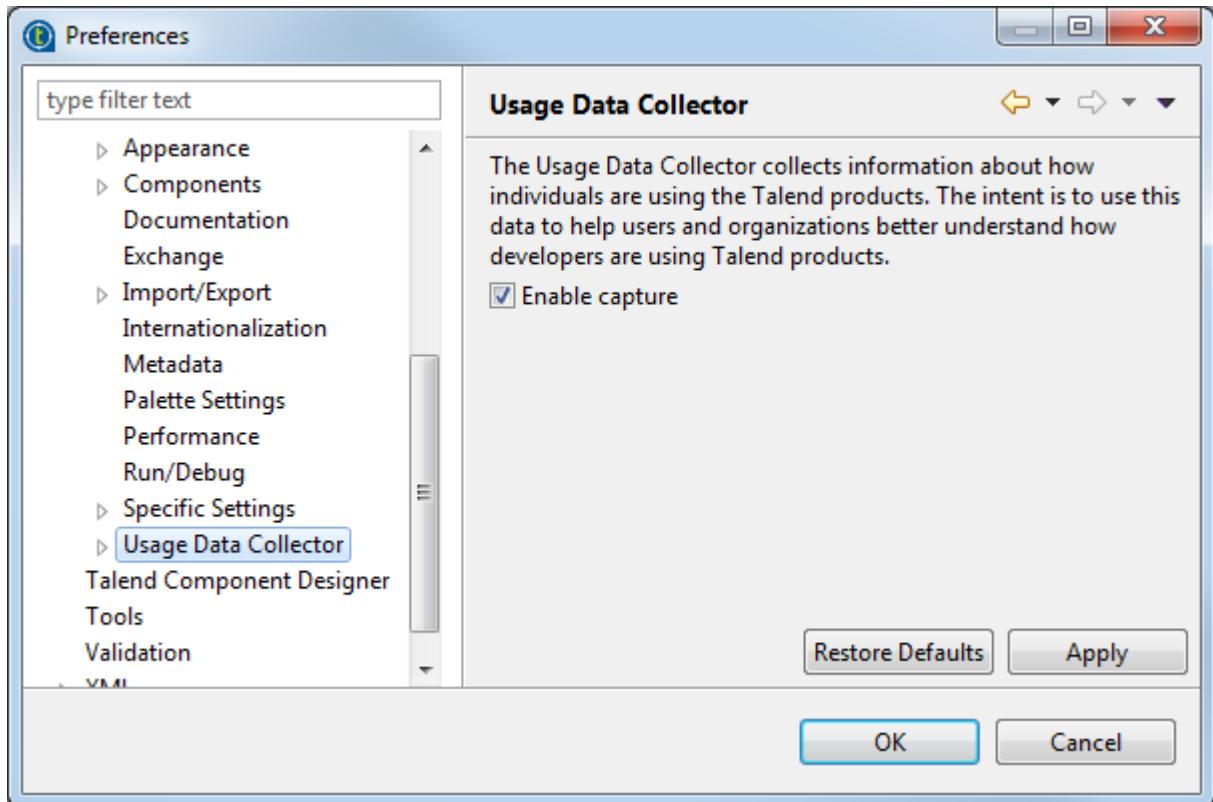
By allowing Talend Studio to collect your Studio usage statistics, you help users better understand **Talend** products and help **Talend** better learn how users are using the products, thus enabling **Talend** to improve product quality and performance to serve users better.

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.

**Note:** 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**.

### Procedure

1. From the menu bar, click **Window > Preferences** to display the **Preferences** dialog box.
2. 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	cSrRusnn4QmFywMvoHoAxHP3MR71UtrF3...
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.

## Setting an authentication-enabled Https proxy for an Azure storage connection

This task sets an Https proxy with user authentication enabled for an existing Azure storage connection.

### About this task

This task enables Talend studio to connect Azure storage through an Https proxy server with user authentication enabled. It assumes that the address of the Https server, the username and password set on the proxy server are available.

### Procedure

1. In the **Repository** pane, expand the **Metadata>Azure Storage** node.  
The Azure storage connection appears under **Azure Storage**.
2. Right-click the Azure storage connection and select **Azure Storage Connection** from the context menu.  
The **Azure Storage** dialog box appears.
3. Click **Proxy Setting** in the **Azure Storage** dialog box.  
The **Preferences (Filtered)** dialog box appears.
4. Select **Enable Basic User Authentication Header** and then select **General>Network Connection**.  
The **Network Connections** pane appears in the right part of the dialog box.
5. Select the **HTTPS** row in **Proxy entries** table and then click **Edit....**  
The **Edit Proxy Entry** dialog box appears.
6. Provide the following settings in the corresponding fields:
  - **Host**, the IP address of the Https server;
  - **Port**, the port number used;
  - **Require Authentication**, select this option to enter username and password.
  - **User**, username for authentication;
  - **Password**, password for authentication.
7. Click **OK** and then **Apply and Close** to validate the settings.

## Using SQL templates

### 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](#) on page 60.

Based on the advantages and disadvantages of ELT, the SQL templates are designed as the ELT facilitation requires.

## 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.

**Note:**

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.

## 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.

### Types of system SQL templates

This section gives detail information related to the different types of the pre-defined SQL templates.

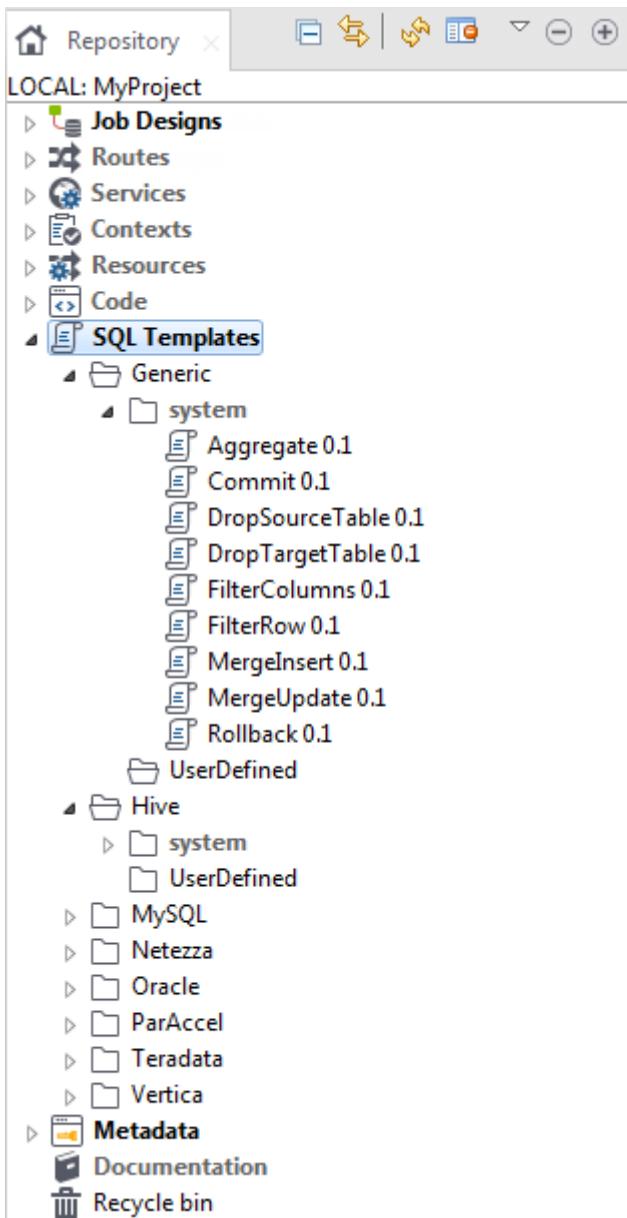
Even though the statements of each group of templates vary from database to database, according to the operations they are intended to accomplish, they are also grouped on the basis of their types in each folder.

The below table provides these types and their related information.

Name	Function	Associated components	Required component parameters
Aggregate	Realizes aggregation (sum, average, count, etc.) over a set of data.	<b>tSQLTemplateAggregate</b>	Database name Source table name Target table name
Commit	Sends a Commit instruction to RDBMS.	<b>tSQLTemplate</b> <b>tSQLTemplateAggregate</b> <b>tSQLTemplateCommit</b> <b>tSQLTemplateFilterColumns</b> <b>tSQLTemplateFilterRows</b> <b>tSQLTemplateMerge</b> <b>tSQLTemplateRollback</b>	Null
Rollback	Sends a Rollback instruction to RDBMS.	<b>tSQLTemplate</b> <b>tSQLTemplateAggregate</b> <b>tSQLTemplateCommit</b> <b>tSQLTemplateFilterColumns</b> <b>tSQLTemplateFilterRows</b> <b>tSQLTemplateMerge</b> <b>tSQLTemplateRollback</b>	Null
DropSourceTable	Removes a source table.	<b>tSQLTemplate</b> <b>tSQLTemplateAggregate</b> <b>tSQLTemplateFilterColumns</b> <b>tSQLTemplateFilterRows</b>	Table name (when use <b>tSQLTemplate</b> ) Source table name
DropTargetTable	Removes a target table.	<b>tSQLTemplateAggregate</b> <b>tSQLTemplateFilterColumns</b> <b>tSQLTemplateFilterRows</b>	Target table name
FilterColumns	Selects and extracts a set of data from given columns in RDBMS.	<b>tSQLTemplateAggregate</b> <b>tSQLTemplateFilterColumns</b> <b>tSQLTemplateFilterRows</b>	Target table name (and schema) Source table name (and schema)
FilterRow	Selects and extracts a set of data from given rows in RDBMS.	<b>tSQLTemplateAggregate</b> <b>tSQLTemplateFilterColumns</b> <b>tSQLTemplateFilterRows</b>	Target table name (and schema) Source table name (and schema) Conditions
MergeInsert	Inserts records from the source table to the target table.	<b>tSQLTemplateMerge</b> <b>tSQLTemplateCommit</b>	Target table name (and schema) Source table name (and schema) Conditions
MergeUpdate	Updates the target table with records from the source table.	<b>tSQLTemplateMerge</b> <b>tSQLTemplateCommit</b>	Target table name (and schema) Source table name (and schema) Conditions

## Accessing 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.

**Note:**

The **system** folders and their content are read only.

From the **Repository** tree view, proceed as follows to open an SQL template:

## Procedure

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 INSERT INTO <%=__TABLE_NAME_TARGET__%> (<%=StringUtils.list(__OPERATION_OUTPUT_COLUMN, ",")%> , <%= StringUtils.list(__GROUPBY_OUTPUT_COLUMN, ",") %> )
21 SELECT <%= operation %>, <%= StringUtils.list(__GROUPBY_INPUT_COLUMN, ",") %> FROM <%= __TABLE_NAME__ %>
22 GROUP BY <%=StringUtils.list(__GROUPBY_INPUT_COLUMN, ",", "", "") %>;
23

```

## Results

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:

Main	Name	Aggregate
Version	Author	test@talend.com
	Purpose	
	Description	
	Creation	5/22/15 10:55 AM
	Modification	5/22/15 10:55 AM
	Version	0.1 M m
	Status	

For further information regarding the different types of SQL templates, see [Types of system SQL templates](#) on page 422.

## Creating user-defined SQL templates

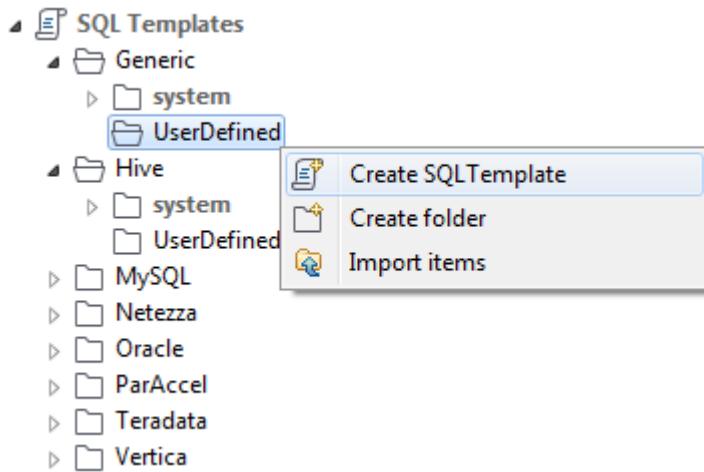
As the transformation you need to accomplish in ELT may exceed the scope of what the given SQL templates can achieve, Talend Studio allows you to develop your own SQL templates according to some writing rules. These SQL templates are stored in the **UserDefined** folders grouped according to the database type in which they will be used.

For more information on the SQL template writing rules, see [SQL statements](#) on page 431.

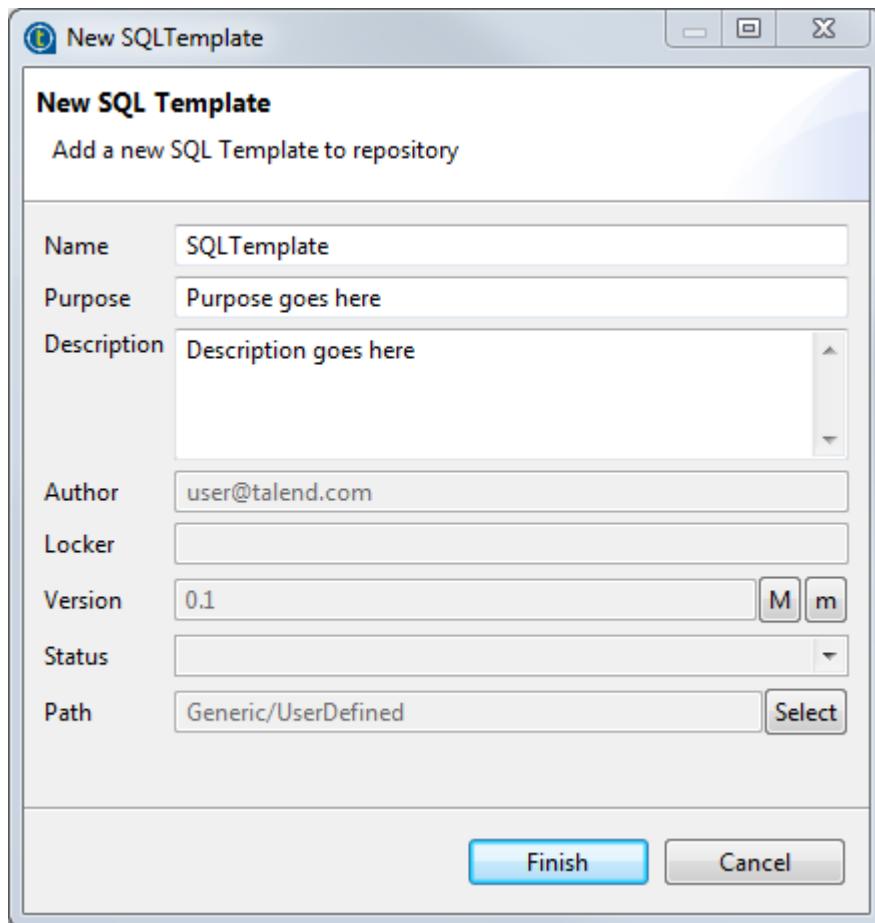
To create a user-defined SQL template:

## Procedure

1. In the **Repository** tree view, expand **SQL Templates** and then the category you want to create the SQL template in.



2. Right-click **UserDefined** and select **Create SQLTemplate** to open the **New SQLTemplate** wizard.



3. 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 the section about iterating on DB tables and deleting their content using a user-defined SQL template at [MySQL](#).

## 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:

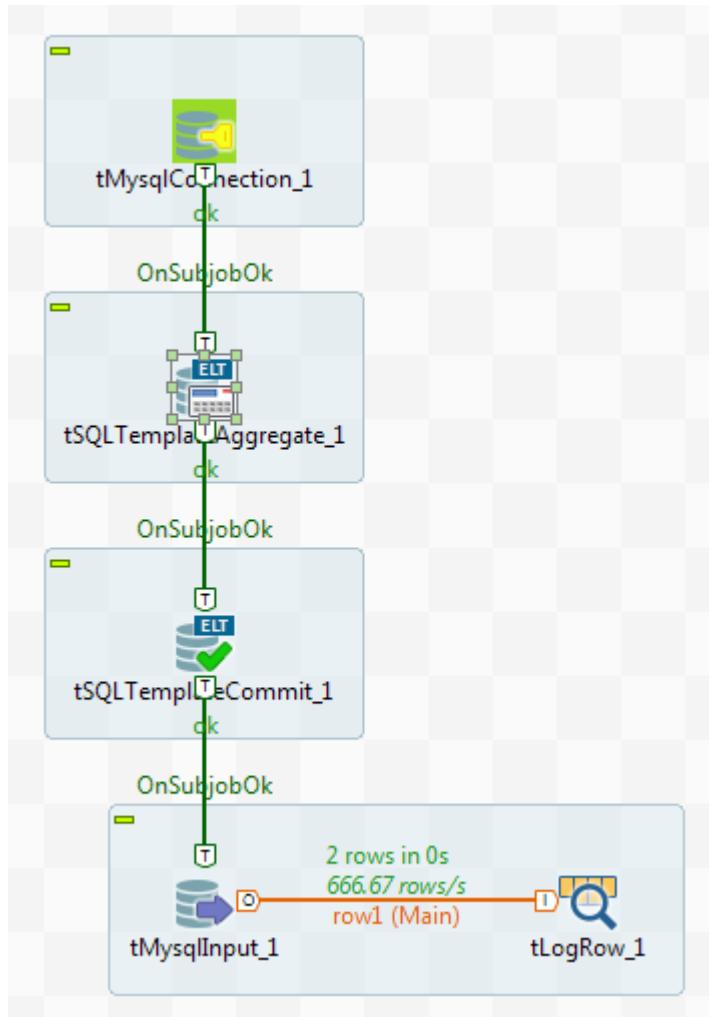
- opens a connection to a Mysql database.
- collects data grouped by specific value(s) from a database table and writes aggregated data in a target database table.
- deletes the source table where the aggregated data comes from.
- reads the target database table and lists the Job execution result.

To connect to the database and aggregate the database table columns:

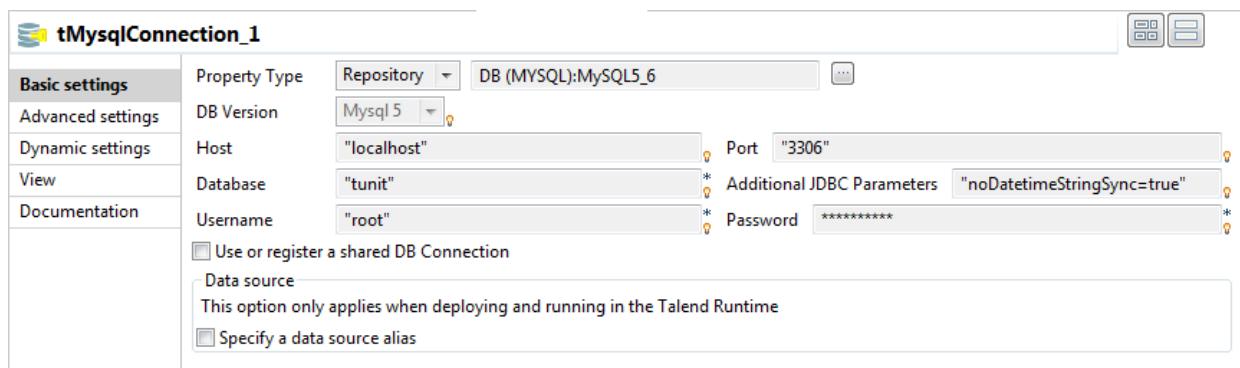
### Configuring a connection to a MySQL database

#### Procedure

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.



5. Double-click **tMysqlConnection** to open its **Basic settings** view.
6. In the **Basic settings** view, set the database connection details manually.

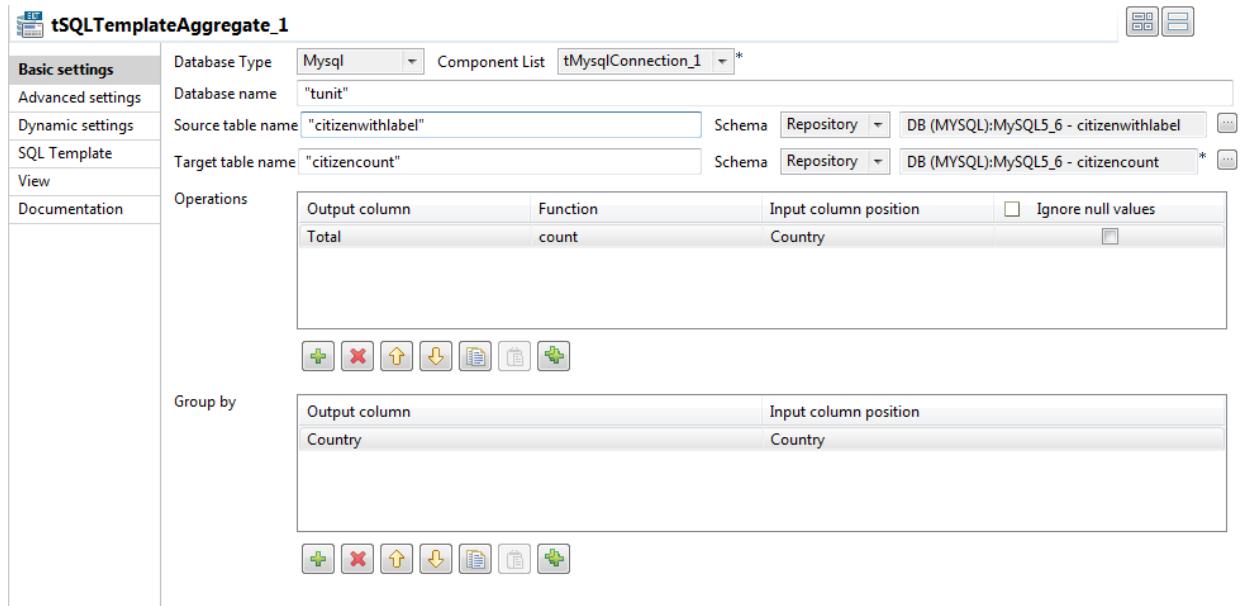


7. Double-click **tSQLTemplateCommit** to open its **Basic settings** view.
8. On the **Database Type** list, select the relevant database type, and from the **Component List**, select the relevant database connection component if more than one connection is used.

### Grouping data, writing aggregated data and dropping the source table

#### Procedure

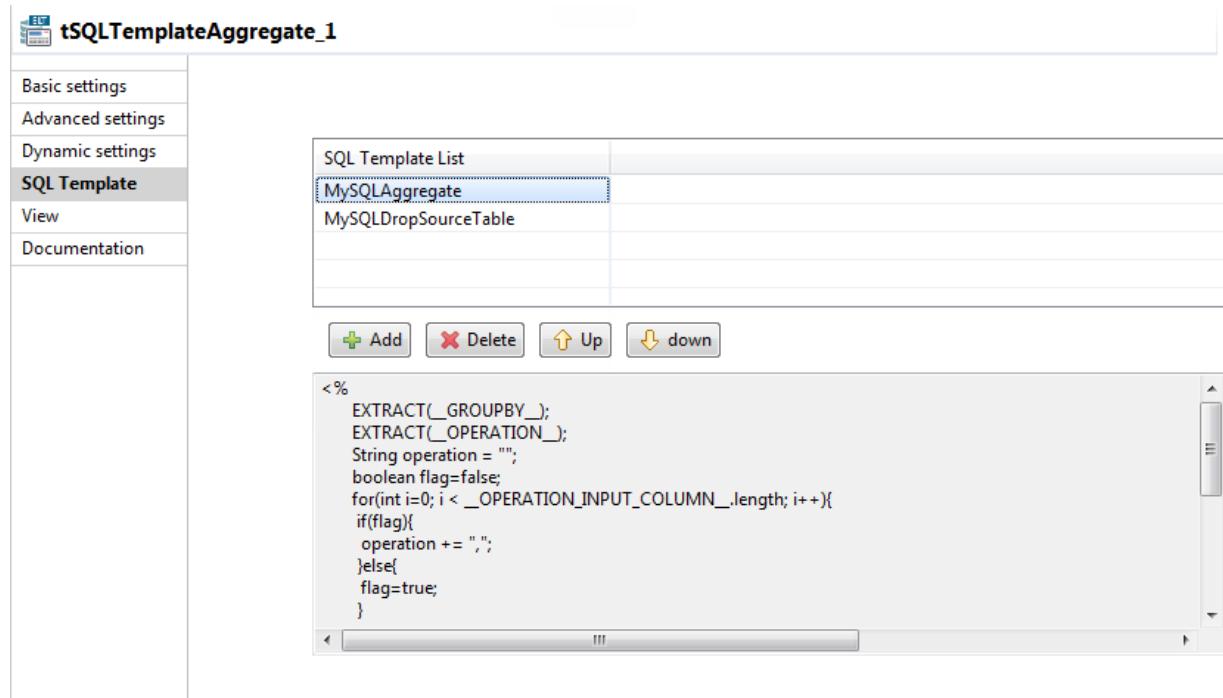
1. Double-click **tSQLTemplateAggregate** to open its **Basic settings** view.



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.

**Note:**

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.

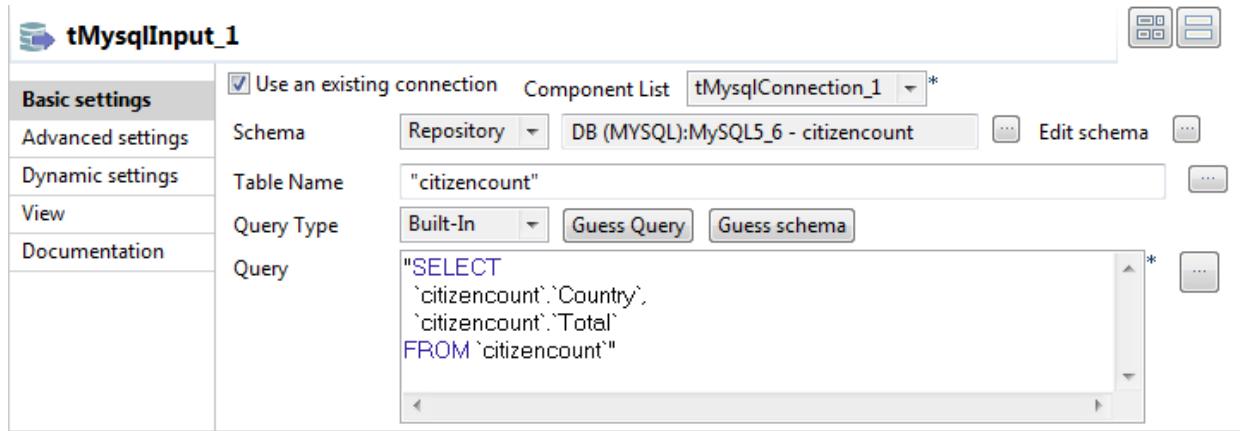
**Note:**

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

### Procedure

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 EITYudong 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 EITYudong 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.

## SQL template writing rules

### 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.

## Comment lines

A comment line starts with # or --. Any line that starts with # or -- will be ignored in code generating.

**Note:**

There is no exception to the lines in the middle part of a SQL statement or within the <%...%> syntax.

## 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.

**Note:**

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.

## 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 [What is a Job design?](#) on page 25.

For more information on the <% . . . %> syntax, see [The <%=...%> syntax](#) on page 432.

For more information on the </ . . . /> syntax, see the following section.

**Note:**

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.

## 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 <% . . . %> syntax, see [The <%=...%> syntax](#) on page 432.

For more information on the <%= . . . %> syntax, see [The <%=...%> syntax](#) on page 433.

The following sections present more specific code used to access more complicated parameters.

**Note:**

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.

## 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\_SHEMA variable name is used to get the component schema.

```
<%
String query = "select ";
SCHEMA(__ELT_METADATA_SHEMA__);
for (int i=0; i < __ELT_METADATA_SHEMA__.length ; i++) {
query += (__ELT_METADATA_SHEMA__[i].name + ",");
}
query += " from " + __TABLE_NAME__;
%>
<%=query %>;
```

In this example, and according to what you want to do, the \_\_ELT\_METADATA\_SHEMA\_\_[i].name code can be replaced by \_\_ELT\_METADATA\_SHEMA\_\_[i].dbType, \_\_ELT\_METADATA\_SHEMA\_\_[i].isKey, \_\_ELT\_METADATA\_SHEMA\_\_[i].length or \_\_ELT\_METADATA\_SHEMA\_\_[i].nullable to access the other fields of the schema column.

The extract statement is SCHEMA(\_\_ELT\_METADATA\_SHEMA\_\_); In this statement, ELT\_METADATA\_SHEMA 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.

**Warning:**

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 on page 39](#).

## 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\_\_);. Uppcase 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.