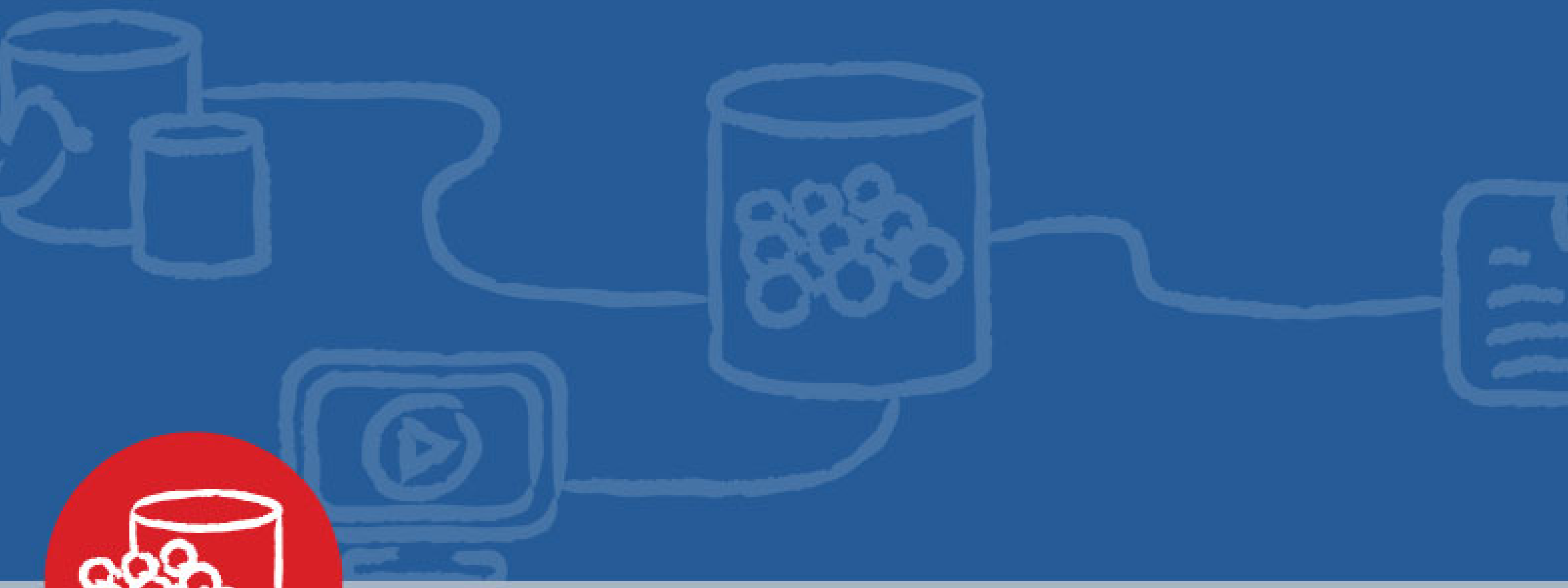
talend*

*open integration solutions



# Talend Open Studio for Big Data

User Guide

5.2.1

Adapted for Talend Open Studio for Big Data 5.2.1. Supersedes previous User Guide releases.

## Copyleft

## Notices

# Table of Contents

# Preface

# 1. General information

## 1.1. Purpose

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

Information presented in this document applies to *Talend Open Studio for Big Data* releases beginning with **5.2.1**.

## 1.2. Audience

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

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

## 1.3. Typographical conventions

This guide uses the following typographical conventions:

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

# 2. Feedback and Support

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

http://talendforge.org/forum

# Chapter 1. Data integration and Talend Studio

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

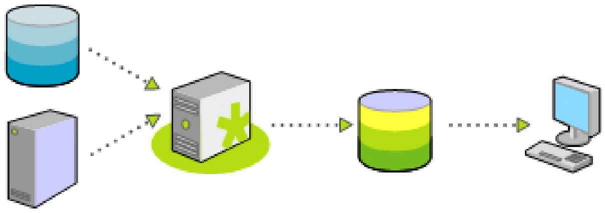
A third reason is the multiplication of data storage formats (XML files, positional flat files, delimited flat files, multi-valued files and so on), protocols (FTP, HTTP, SOAP, SCP and so on) and database technologies.

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

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

# 1.1. Data analytics

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



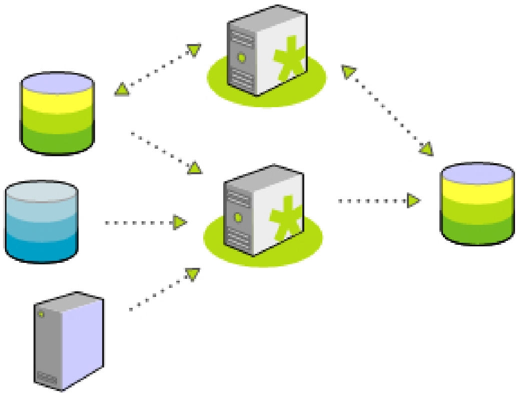
*Talend Open Studio for Big Data* offers nearly comprehensive connectivity to:

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

Most connectors addressing each of the above needs are detailed in *Talend Open Studio for Big Data Components Reference Guide*. For information about their orchestration in *Talend Open Studio for Big Data*, see chapter *Designing a data integration Job*.

# 1.2. Operational integration

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



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

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

Most connectors addressing each of the above needs are detailed in *Talend Open Studio for Big Data Components Reference Guide*. For information about their orchestration in *Talend Open Studio for Big Data*, see chapter

*Designing a data integration Job*. For information about designing a detailed data integration Job using the output stream feature, see section *Using the output stream feature*.

# Chapter 2. Getting started with Talend Studio

This chapter introduces *Talend Open Studio for Big Data*. It provides basic configuration information required to get started with *Talend Open Studio for Big Data*.

The chapter guides you through the basic steps in creating local projects. It also describes how to set preferences and customize the workspace in *Talend Open Studio for Big Data*.

Before starting any data integration processes, you need to be familiar with *Talend Open Studio for Big Data* Graphical User Interface (GUI). For more information, see appendix *GUI*.

# 2.1. Important concepts in *Talend Open Studio for Big Data*

When working with *Talend Open Studio for Big Data*, you will often come across words such as repository, project, workspace, Job, component and item.

Understanding the concept behind each of these words is crucial to grasping the functionality of *Talend Open Studio for Big Data*.

**What is a repository?** A repository is the storage location *Talend Open Studio for Big Data* uses to gather data related to all of the technical items that you use to design Jobs.

**What is a project?** Projects are structured collections of technical items and their associated metadata. All of the Jobs you design are organized in Projects.

You can create as many projects as you need in a repository. For more information about projects, see section *Working with projects*.

**What is a workspace?** A workspace is the directory where you store all your project folders. You need to have one workspace directory per connection (repository connection). *Talend Open Studio for Big Data* enables you to connect to different workspace directories, if you do not want to use the default one.

For more information about workspaces, see section *Working with different workspace directories*.

**What is a Job?** A Job is a graphical design, of one or more components connected together, that allows you to set up and run dataflow management processes. It translates business needs into code, routines and programs. Jobs address all of the different sources and targets that you need for data integration processes and all other related processes.

For detailed information about how to design data integration processes in *Talend Open Studio for Big Data*, see chapter *Designing a data integration Job*.

**What is a component?** A component is a preconfigured connector used to perform a specific data integration operation, no matter what data sources you are integrating: databases, applications, flat files, Web services, etc. A component can minimize the amount of hand-coding required to work on data from multiple, heterogeneous sources.

Components are grouped in families according to their usage and displayed in the **Palette** of the *Talend Open Studio for Big Data* main window.

For detailed information about components types and what they can be used for, see *Talend Open Studio for Big Data Components Reference Guide*.

**What is an item?** An item is the fundamental technical unit in a project. Items are grouped, according to their types, as: Job Design, Context, Code, etc. One item can include other items. For example, the Jobs you design are items, and routines you use inside your Jobs are items as well.

# 2.2. Launching *Talend Open Studio for Big Data*

## 2.2.1. How to launch the Studio for the first time

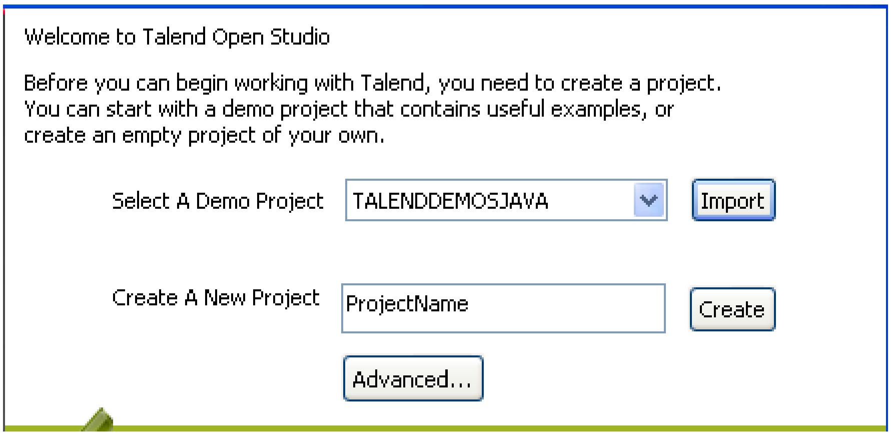
To open *Talend Open Studio for Big Data* for the first time, complete the following:

1. Unzip the *Talend Open Studio for Big Data* zip file and, in the folder, double-click the executable file corresponding to your operating system.

   The Studio zip archive contains binaries for several platforms including Mac OS X and Linux/Unix.

2. In the **[License]** window that appears, read and accept the terms of the end user license agreement to continue.

   The startup window appears.



Welcome to Talend Open Studio

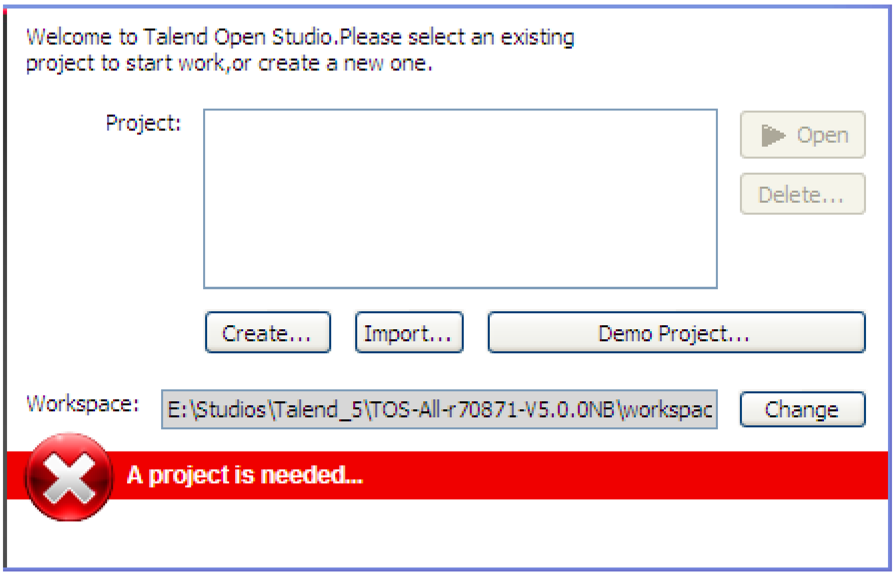
Before you can begin working with Talend, you need to create a project. You can start with a demo project that contains useful examples, or create an empty project of your own.

Select A Demo Project: TALENDDEMOSJAVA

   This screen appears only when you launch the *Talend Open Studio for Big Data* for the first time or if all existing projects have been deleted.

3. Click the **Import** button to import the selected demo project, or type in a project name in the **Create A New Project** field and click the **Create** button to create a new project, or click the **Advanced...** button to go to the Studio login window.

   In this procedure, click **Advanced...** to go to the Studio login widow. For more information about the other two options, see section *How to import the demo project* and section *How to create a project* respectively.



Welcome to Talend Open Studio.Please select an existing project to start work,or create a new one.

Workspace: E:\Studios\Talend_5\TOS-All-r70871-V5.0.0NB\workspac

A project is needed...

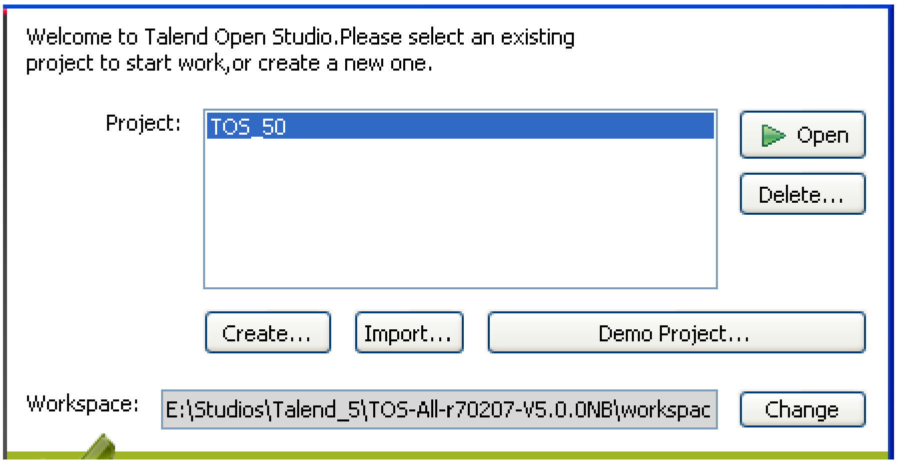
4. From the Studio login window:

| Click... | To... |
|---|---|
| **Create...** | create a new project that will hold all Jobs designed in the Studio.<br>For more information, see section *How to create a project*. |
| **Import...** | import one or more existing projects. |

| Click... | To... |
|---|---|
| | For more information, see section *How to import projects*. |
| **Demo Project...** | import the Demo project including numerous samples of ready-to-use Jobs. This Demo project can help you understand the functionalities of different **Talend** components.<br>For more information, see section *How to import the demo project*. |
| **Open** | open the selected existing project.<br>For more information, see section *How to open a project*. |
| **Delete...** | open a dialog box in which you can delete any created or imported project that you do not need anymore.<br>For more information, see section *How to delete a project*. |

As the purpose of this procedure is to create a new project, click **Create...** to open the **[New project]** dialog box.

5. In the dialog box, enter a name for your project and click **Finish** to close the dialog box. The name of the new project is displayed in the **Project** list.



Welcome to Talend Open Studio.Please select an existing project to start work,or create a new one.

Project: TOS_50

Workspace: E:\Studios\Talend_5\TOS-All-r70207-V5.0.0NB\workspac

6. Select the project, and click **Open**.

   The **Connect to TalendForge** page appears, inviting you to connect to the **Talend** Community so that you can check, download, install external components and upload your own components to the **Talend** Community to share with other **Talend** users directly in the **Exchange** view of your Job designer in the Studio.

   To learn more about the **Talend** Community, click the **read more** link. For more information on using and sharing community components, see section *How to download/upload Talend Community components*.

7. If you want to connect to the **Talend** Community later, click **Skip** to continue.

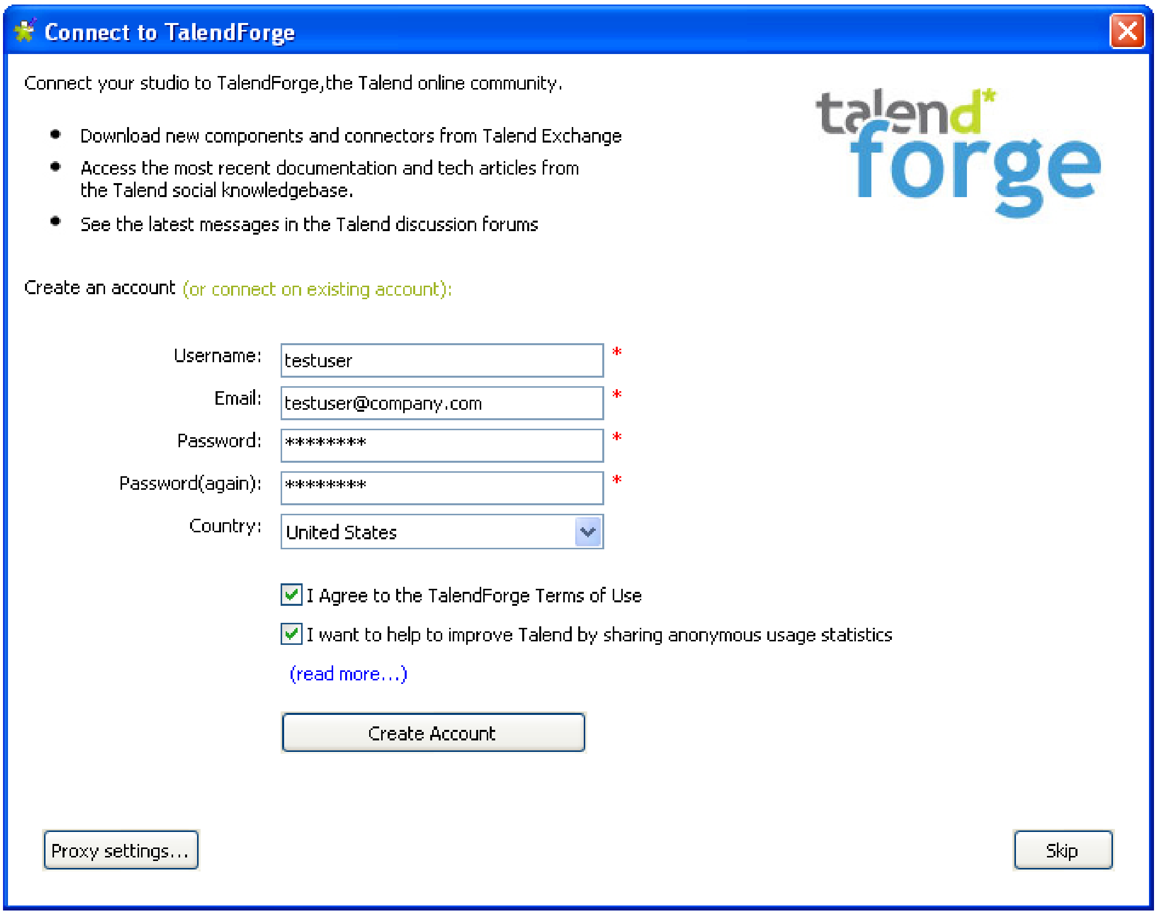
8. If you are working behind a proxy, click **Proxy setting** and fill in the **Proxy Host** and **Proxy Port** fields of the **Network setting** dialog box.

9. By default, the Studio will automatically collect product usage data and send the data periodically to servers hosted by **Talend** for product usage analysis and sharing purposes only. If you do not want the Studio to do so, clear the **I want to help to improve Talend by sharing anonymous usage statistics** check box.

   You can also turn on or off usage data collection in the Usage Data Collector preferences settings. For more information, see section *Usage Data Collector preferences (Talend > Usage Data Collector)*.

10. Fill in the required information, select the **I Agree to the TalendForge Terms of Use** check box, and click **Create Account** to create your account and connect to the **Talend** Community automatically. If you already have created an account at http://www.talendforge.org, click the **or connect on existing account** link to sign in.

Be assured that any personal information you may provide to **Talend** will never be transmitted to third parties nor used for any purpose other than joining and logging in to the **Talend** Community and being informed of **Talend** latest updates.



Connect your studio to TalendForge,the Talend online community.

- Download new components and connectors from Talend Exchange
- Access the most recent documentation and tech articles from the Talend social knowledgebase.
- See the latest messages in the Talend discussion forums

Create an account (or connect on existing account):

Username: testuser *

Email: testuser@company.com *

Password: ******** *

Password(again): ******** *

Country: United States

☑ I Agree to the TalendForge Terms of Use

☑ I want to help to improve Talend by sharing anonymous usage statistics

(read more...)

Create Account

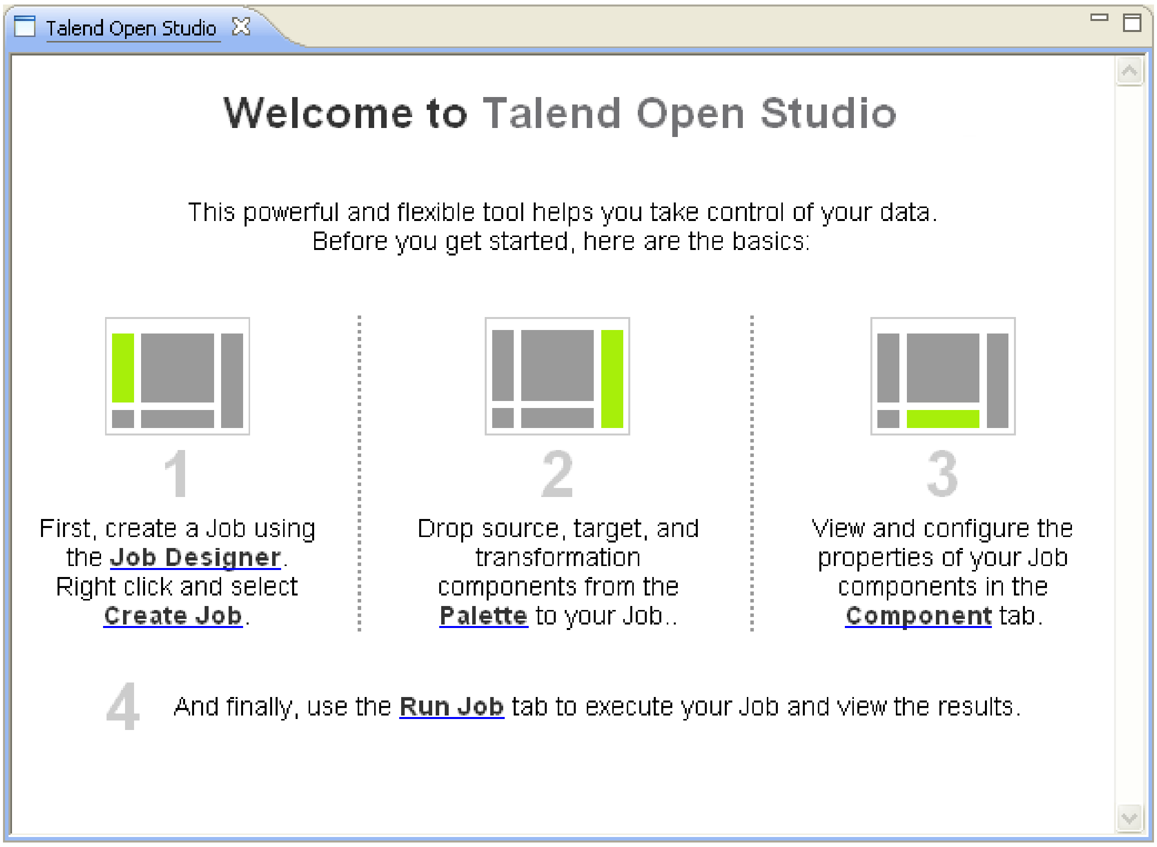
This page will not appear again at Studio startup once you successfully connect to the **Talend** Community or if you click **Skip** too many times. You can show this page again from the **[Preferences]** dialog box. For more information, see section *Exchange preferences (Talend > Exchange)*.

A progress information bar and a welcome window display consecutively. From this page you have direct links to user documentation, tutorials, **Talend** forum, **Talend Exchange** and **Talend** latest news.

11. Click **Start now!** to open *Talend Open Studio for Big Data* main window.

    The main window opens on a welcome page which has useful tips for beginners on how to get started with the Studio. Clicking an underlined link brings you to the corresponding tab view or opens the corresponding dialog box.

    For more information on how to open a project, see section *How to open a project*.



Talend Open Studio

Welcome to Talend Open Studio

This powerful and flexible tool helps you take control of your data.
Before you get started, here are the basics:

1 First, create a Job using the **Job Designer**. Right click and select **Create Job**.

2 Drop source, target, and transformation components from the **Palette** to your Job..

3 View and configure the properties of your Job components in the **Component** tab.

4 And finally, use the **Run Job** tab to execute your Job and view the results.

## 2.2.2. How to set up a project

To open the *Talend Open Studio for Big Data* main window, you must first set up a project.

You can set up a project by:

- creating a new project. For more information, see section *How to create a project*.
- importing one or more projects you already created in other sessions of *Talend Open Studio for Big Data*. For more information, see section *How to import projects*.
- importing the Demo project. For more information, see section *How to import the demo project*.

# 2.3. Working with different workspace directories

*Talend Open Studio for Big Data* makes it possible to create many workspace directories and connect to a workspace different from the one you are currently working on, if necessary.

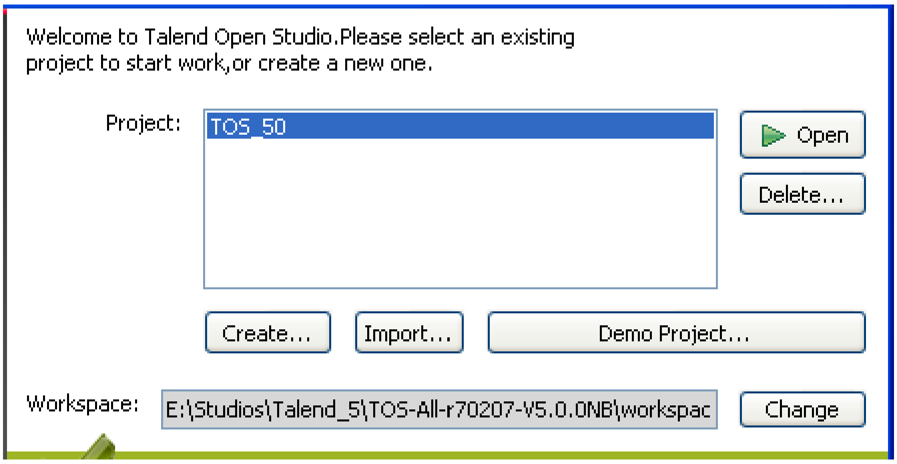
This flexibility enables you to store these directories wherever you want and give the same project name to two or more different projects as long as you store the projects in different directories.

## 2.3.1. How to create a new workspace directory

*Talend Open Studio for Big Data* is delivered with a default workspace directory. However, you can create as many new directories as you want and store your project folders in them according to your preferences.

To create a new workspace directory:

1. In the project login window, click **Change** to open the dialog box for selecting the directory of the new workspace.



Welcome to Talend Open Studio.Please select an existing project to start work,or create a new one.

Project: TOS_50

Workspace: E:\Studios\Talend_5\TOS-All-r70207-V5.0.0NB\workspac

2. In the dialog box, set the path to the new workspace directory you want to create and then click **OK** to close the view.

   On the login window, a message displays prompting you to restart the Studio.

3. Click **Restart** to restart the Studio.

4. On the re-initiated login window, set up a project for this new workspace directory.

   For more information, see section *How to set up a project*.

5. Select the project from the **Project** list and click **Open** to open *Talend Open Studio for Big Data* main window.

All Jobs you design in the current instance of the Studio will be stored in the new workspace directory you created. .

When you need to connect to any of the workspaces you have created, simply repeat the process described in this section.

# 2.4. Working with projects

In *Talend Open Studio for Big Data*, the highest physical structure for storing all different types of data integration Jobs, routines, etc. is the “project”.

From the login window of *Talend Open Studio for Big Data*, you can:

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

  For more information, see section *How to import the demo project*.

- create a local project. When connecting to *Talend Open Studio for Big Data* for the first time, there are no default projects listed. You need to create a project and open it in the Studio to store all the Jobs you create in it. When creating a new project, a tree folder is automatically created in the workspace directory on your repository server. This will correspond to the **Repository** tree view displaying on *Talend Open Studio for Big Data* main window.

  For more information, see section *How to create a project*.

- import projects you have already created with previous releases of *Talend Open Studio for Big Data* into your current *Talend Open Studio for Big Data* workspace directory by clicking **Import...** .

  For more information, see section *How to import projects*.

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

  For more information, see section *How to open a project*.

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

  For more information, see section *How to delete a project*.

Once you launch *Talend Open Studio for Big Data*, you can export the resources of one or more of the created projects in the current instance of the Studio. For more information, see section *How to export a project*.

## 2.4.1. How to create a project

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

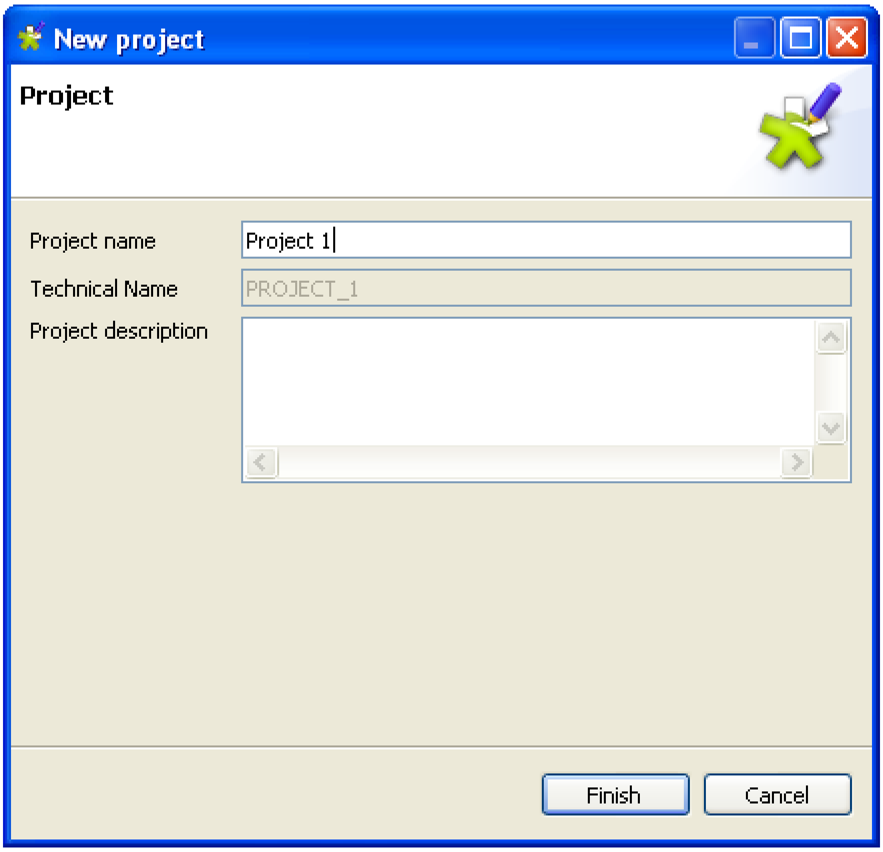
To create a project:

1. Launch *Talend Open Studio for Big Data*.

2. Use either of the following two options:

   - Enter a project name in the **Create A New Project** field and click **Create** to open the **[New project]** dialog box with the **Project name** field filled with the specified name.

   - Click **Advanced**, and then from the login window click **Create...** to open the **[New project]** dialog box with an empty **Project name** field.



New project

Project

Project name: Project 1

Technical Name: PROJECT_1

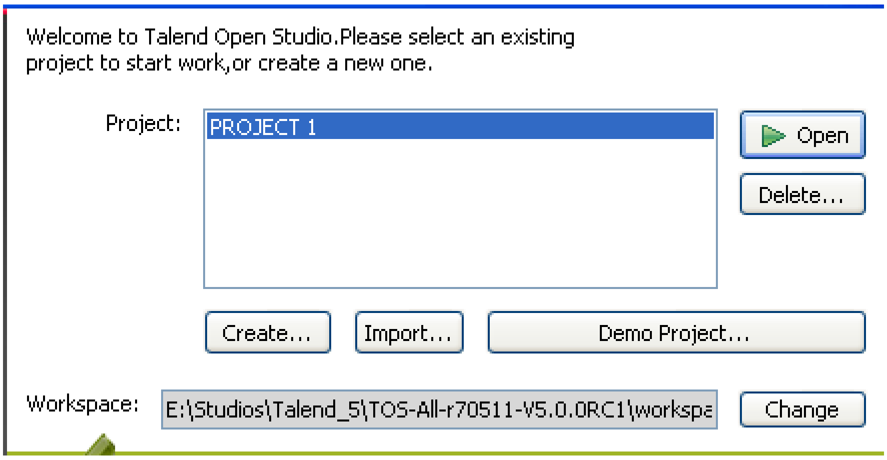
Project description:

3. In the **Project name** field, enter a name for the new project, or change the previously specified project name if needed. This field is mandatory.

   A message shows at the top of the wizard, according to the location of your pointer, to inform you about the nature of data to be filled in, such as forbidden characters

   The read-only “technical name” is used by the application as file name of the actual project file. This name usually corresponds to the project name, upper-cased and concatenated with underscores if needed.

4. Click **Finish**. The name of the newly created project is displayed in the **Project** list in *Talend Open Studio for Big Data* login window.



Welcome to Talend Open Studio.Please select an existing project to start work,or create a new one.

Project: PROJECT 1

Workspace: E:\Studios\Talend_5\TOS-All-r70511-V5.0.0RC1\workspa

From version 5.0 onwards, Java is the only language generated.

To open the newly created project in *Talend Open Studio for Big Data*, select it from the **Project** list and then click **Open**. A generation engine initialization window displays. Wait till the initialization is complete.

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

If you already used *Talend Open Studio for Big Data* and want to import projects from a previous release, see section *How to import projects*.

## 2.4.2. How to import the demo project

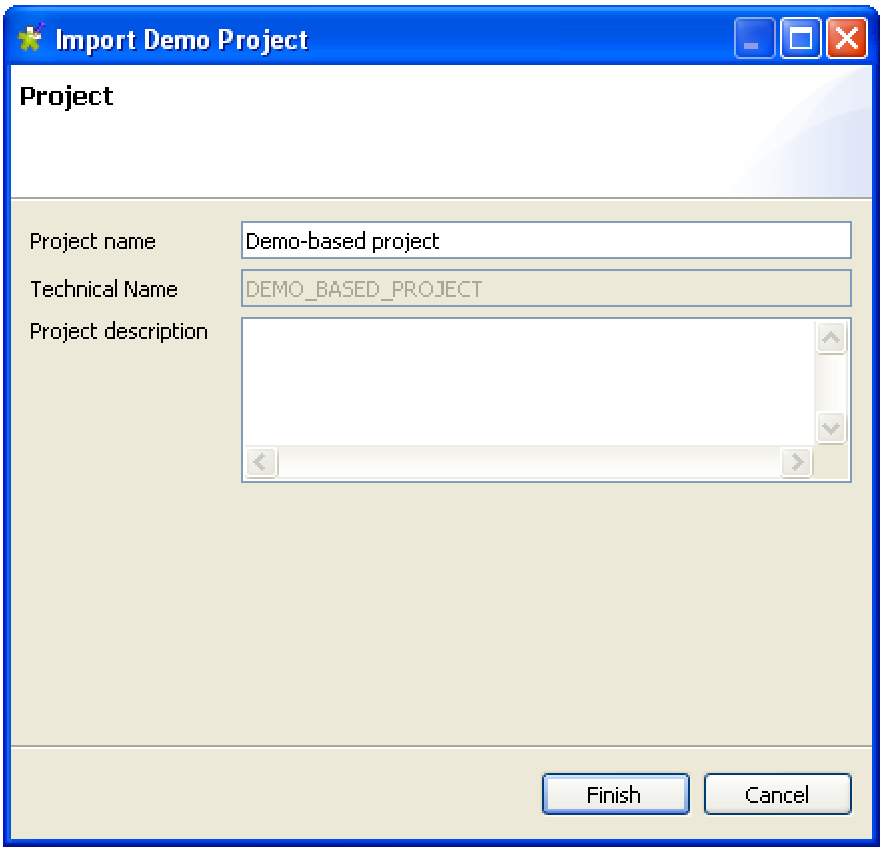
In *Talend Open Studio for Big Data*, you can import the demo project that includes numerous samples of ready to use Jobs. This demo project can help you understand the functionalities of different **Talend** components.

At the first launch of *Talend Open Studio for Big Data*, you can:

- create a new project in your repository using the demo project as a template,
- import the demo project *TALENDDEMOSJAVA* into your repository.

To create a new project based on the demo project:

1. Click the **Import** button next to the **Select A Demo Project** list box. The **[Import Demo Project]** dialog box opens.



Import Demo Project

Project

Project name: Demo-based project

Technical Name: DEMO_BASED_PROJECT

Project description:

2. Type in a name for the new project, and click **Finish** to create the project.

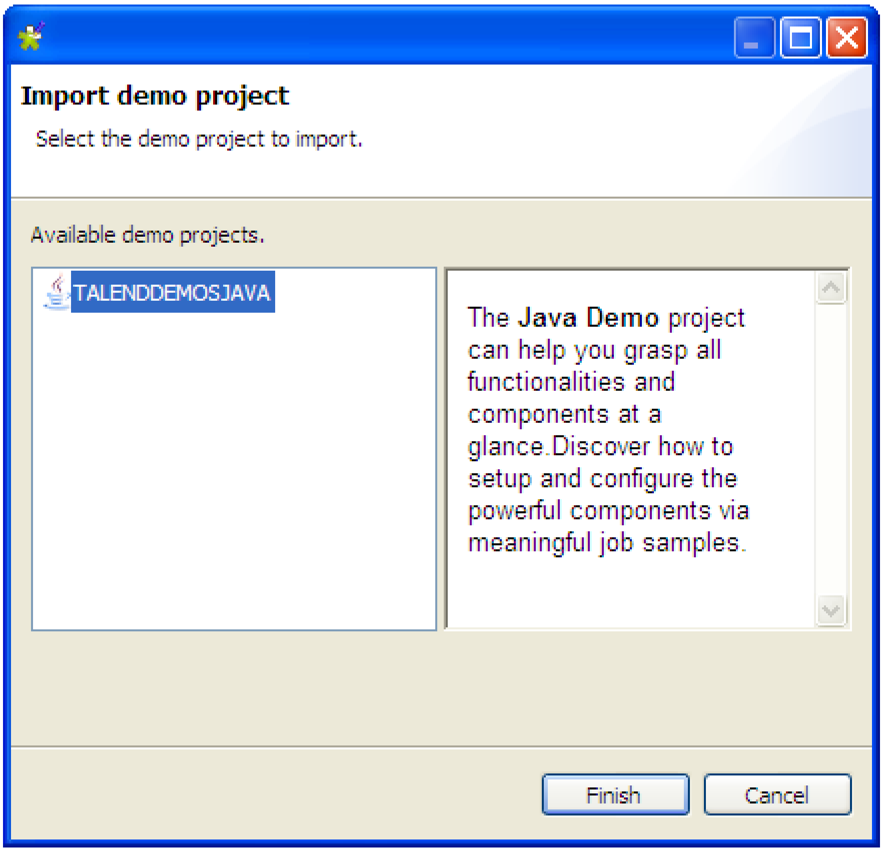
   A confirmation message is displayed, informing you that the demo project has been successfully imported in the current instance of the Studio.

3. Click **OK** to close the confirmation message.

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

To import the demo project *TALENDDEMOSJAVA* into your repository:

1. Click **Advanced...**, and then from the login window click **Demo Project...**. The **[Import demo project]** dialog box opens.



Import demo project

Select the demo project to import.

Available demo projects.

TALENDDEMOSJAVA

The **Java Demo** project can help you grasp all functionalities and components at a glance.Discover how to setup and configure the powerful components via meaningful job samples.

2. Select the demo project and then click **Finish** to close the dialog box.

   A confirmation message is displayed, informing your that the demo project has been successfully imported in the current instance of the Studio.

3. Click **OK** to close the confirmation message.

   The imported demo project displays in the **Project** list on the login window.

To open the imported demo project in *Talend Open Studio for Big Data*, select it from the **Project** list and then click **Open**. A generation engine initialization window displays. Wait till the initialization is complete.

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

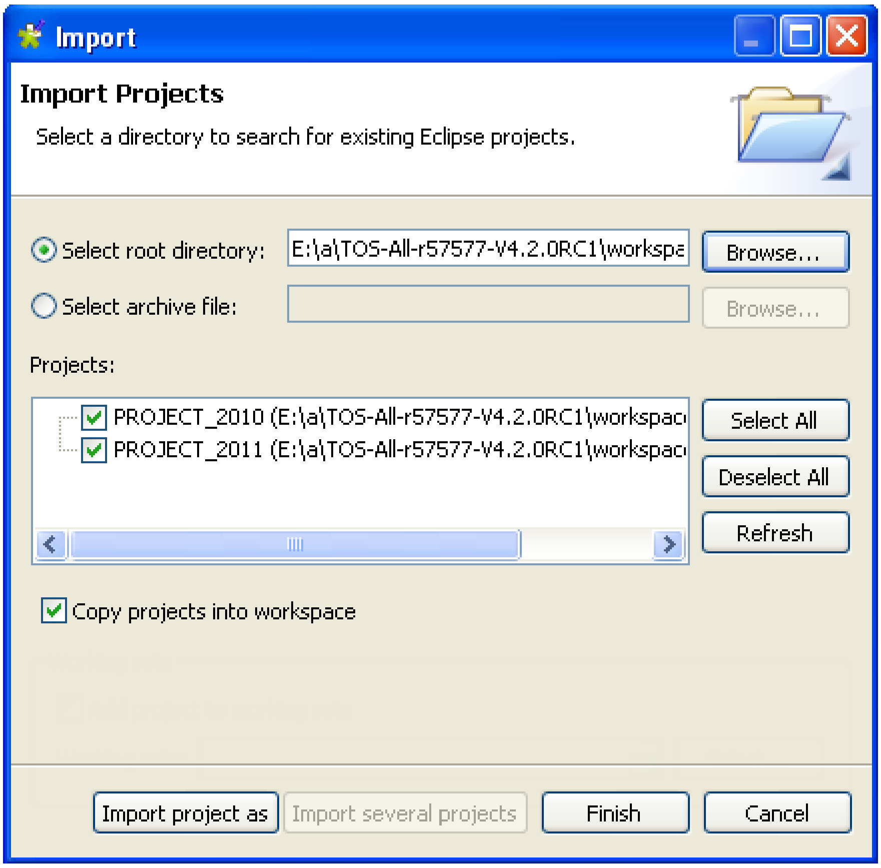
You can use these samples to get started with your own Job design.

## 2.4.3. How to import projects

In *Talend Open Studio for Big Data*, you can import projects you already created with previous releases of the Studio.

1. If you are launching *Talend Open Studio for Big Data* for the first time, click **Advanced...** to open to the login window.

2. From the login window, click **Import...** to open the **[Import]** wizard.



Import

Import Projects

Select a directory to search for existing Eclipse projects.

● Select root directory: E:\a\TOS-All-r57577-V4.2.0RC1\workspa

○ Select archive file:

Projects:

☑ PROJECT_2010 (E:\a\TOS-All-r57577-V4.2.0RC1\workspac

☑ PROJECT_2011 (E:\a\TOS-All-r57577-V4.2.0RC1\workspac

☑ Copy projects into workspace

3. Click **Import several projects** if you intend to import more than one project simultaneously.

4. Click **Select root directory** or **Select archive file** depending on the source you want to import from.

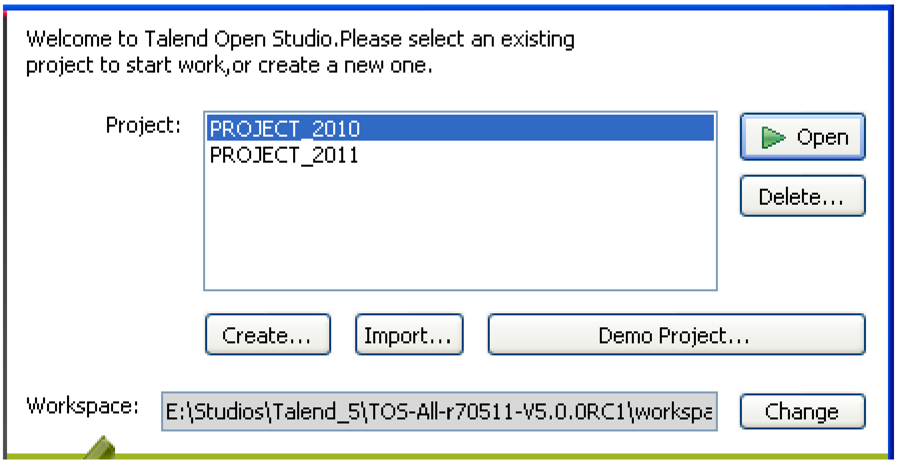
5. Click **Browse...** to select the workspace directory/archive file of the specific project folder. By default, the workspace in selection is the current release's one. Browse up to reach the previous release workspace directory or the archive file containing the projects to import.

6. Select the **Copy projects into workspace** check box to make a copy of the imported project instead of moving it.

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

7. From the **Projects** list, select the projects to import and click **Finish** to validate the operation.

In the login window, the names of the imported projects now appear on the **Project** list.



Welcome to Talend Open Studio.Please select an existing project to start work,or create a new one.

Project: PROJECT_2010
PROJECT_2011

Workspace: E:\Studios\Talend_5\TOS-All-r70511-V5.0.0RC1\workspa

You can now select the imported project you want to open in *Talend Open Studio for Big Data* and click **Open** to launch the Studio.

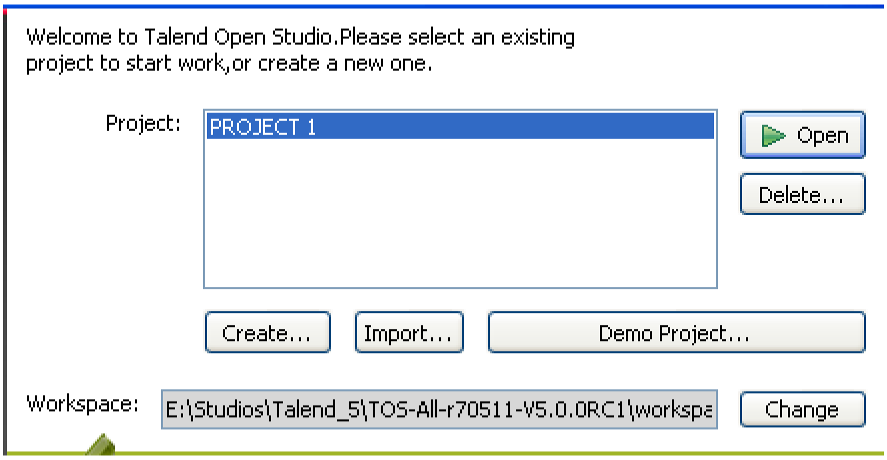
A generation initialization window might come up when launching the application. Wait until the initialization is complete.

## 2.4.4. How to open a project

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

To open a project in *Talend Open Studio for Big Data*:

On the Studio login screen, select the project from the **Project** list, and click **Open**.



Welcome to Talend Open Studio.Please select an existing project to start work,or create a new one.

Project: PROJECT 1

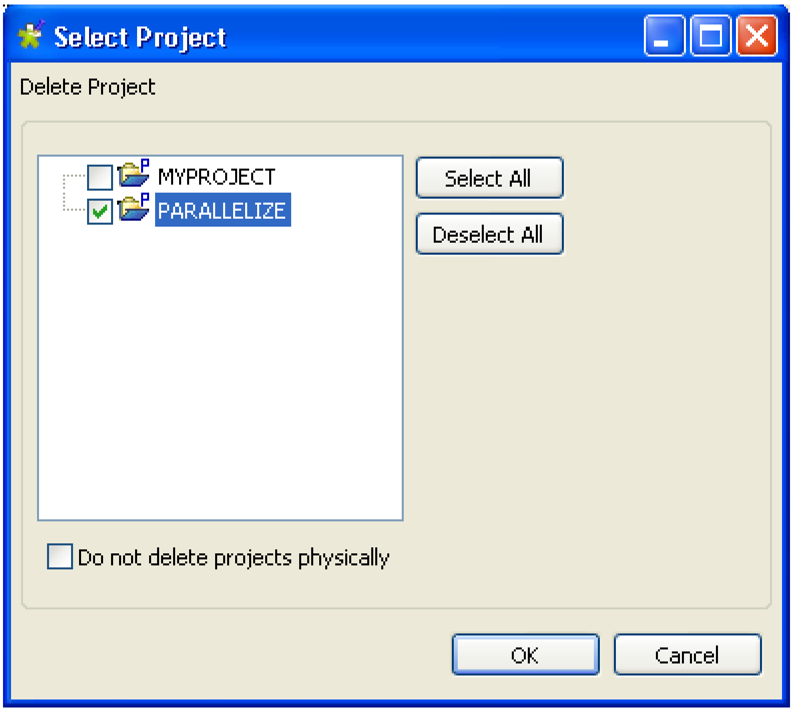
Workspace: E:\Studios\Talend_5\TOS-All-r70511-V5.0.0RC1\workspa

A progress bar appears, and the *Talend Open Studio for Big Data* main window opens. A generation engine initialization dialog bow displays. Wait till initialization is complete.

When you open a project imported from a previous version of the Studio, an information window pops up to list a short description of the successful migration tasks. For more information, see section *Migration tasks*.

## 2.4.5. How to delete a project

1. On the login screen, click **Delete...**to open the **[Select Project]** dialog box.



Select Project

Delete Project

□ MYPROJECT

☑ PARALLELIZE

□ Do not delete projects physically

2. Select the check box(es) of the project(s) you want to delete.

3. Click **OK** to validate the deletion.

   The project list on the login window is refreshed accordingly.

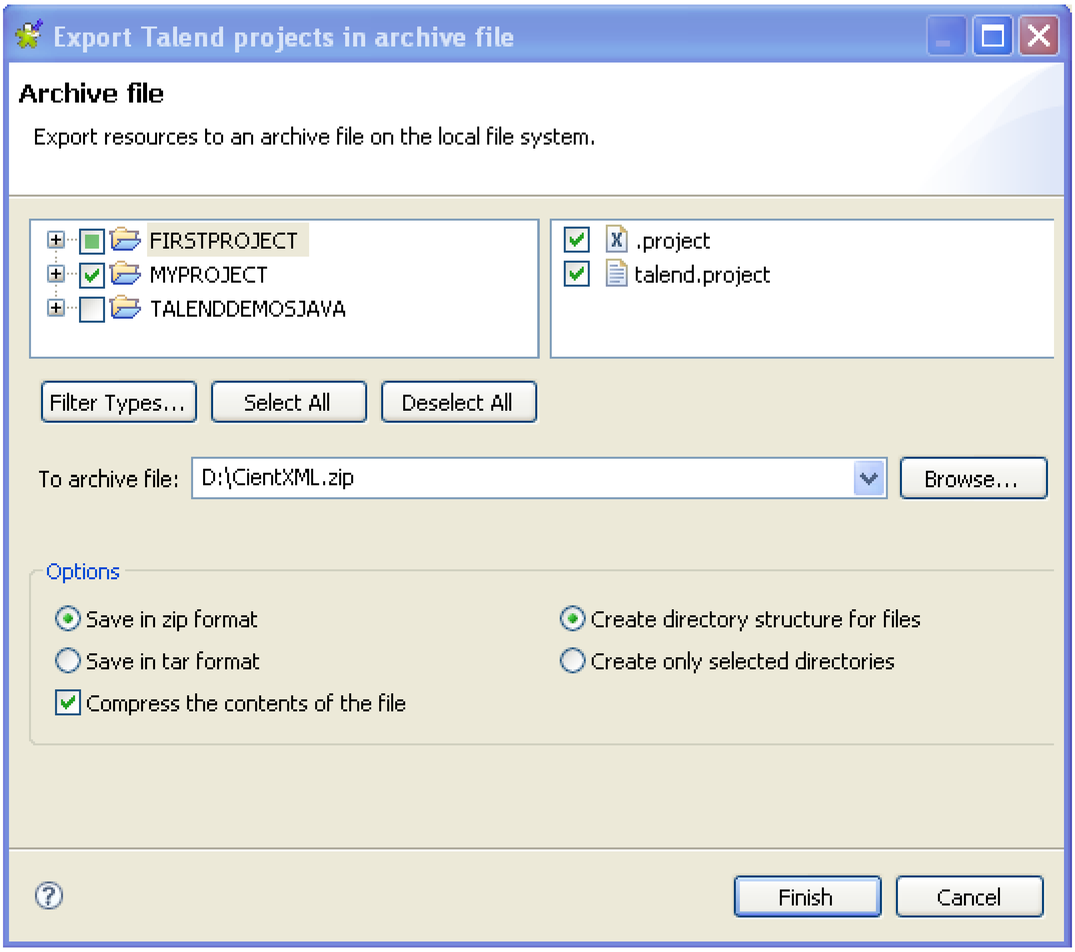
   *Be careful, this action is irreversible. When you click **OK**, there is no way to recuperate the deleted project(s).*

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

## 2.4.6. How to export a project

*Talend Open Studio for Big Data*, allows you to export projects created or imported in the current instance of *Talend Open Studio for Big Data*.

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



Export Talend projects in archive file

Archive file

Export resources to an archive file on the local file system.

- ⊞ ■ FIRSTPROJECT
- ⊞ ☑ MYPROJECT
- ⊞ □ TALENDDEMOSJAVA

- ☑ .project
- ☑ talend.project

To archive file: D:\CientXML.zip

Options

- ● Save in zip format
- ○ Save in tar format
- ☑ Compress the contents of the file
- ● Create directory structure for files
- ○ Create only selected directories

2. Select the check boxes of the projects you want to export. You can select only parts of the project through the **Filter Types...** link, if need be (for advanced users).

3. In the **To archive file** field, type in the name of or browse to the archive file where you want to export the selected projects.

4. In the **Option** area, select the compression format and the structure type you prefer.

5. Click **Finish** to validate the changes.

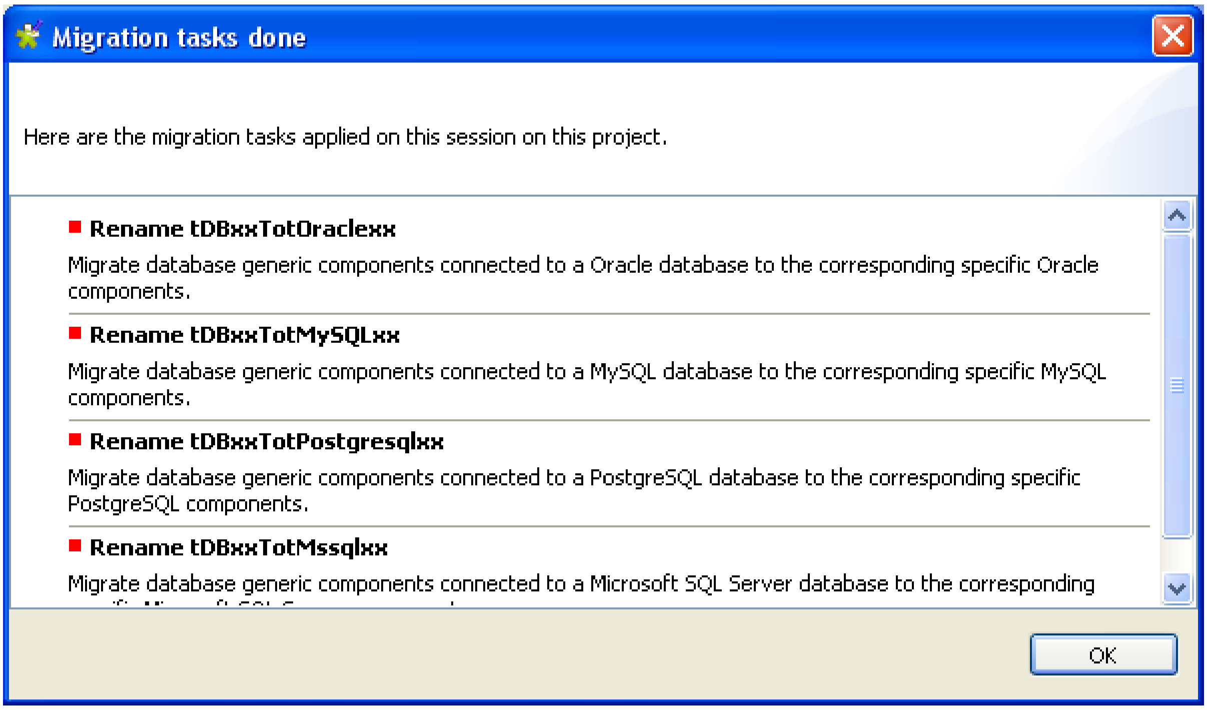
The archived file that holds the exported projects is created in the defined place.

## 2.4.7. Migration tasks

Migration tasks are performed to ensure the compatibility of the projects you created with a previous version of *Talend Open Studio for Big Data* with the current release.

As some changes might become visible to the user, we thought we'd share these update tasks with you through an information window.

This information window pops up when you launch the project you imported (created) in a previous version of *Talend Open Studio for Big Data*. It lists and provides a short description of the tasks which were successfully performed so that you can smoothly roll your projects.



Migration tasks done

Here are the migration tasks applied on this session on this project.

- **Rename tDBxxTotOraclexx**
  Migrate database generic components connected to a Oracle database to the corresponding specific Oracle components.
- **Rename tDBxxTotMySQLxx**
  Migrate database generic components connected to a MySQL database to the corresponding specific MySQL components.
- **Rename tDBxxTotPostgresqlxx**
  Migrate database generic components connected to a PostgreSQL database to the corresponding specific PostgreSQL components.
- **Rename tDBxxTotMssqlxx**
  Migrate database generic components connected to a Microsoft SQL Server database to the corresponding

Some changes that affect the usage of *Talend Open Studio for Big Data* include, for example:

- **tDBInput** used with a MySQL database becomes a specific **tDBMysqlInput** component the aspect of which is automatically changed in the Job where it is used.
- **tUniqRow** used to be based on the Input schema keys, whereas the current **tUniqRow** allows the user to select the column to base the unicity on.

## 2.5. Setting *Talend Open Studio for Big Data* preferences

You can define various properties of *Talend Open Studio for Big Data* main design workspace according to your needs and preferences.

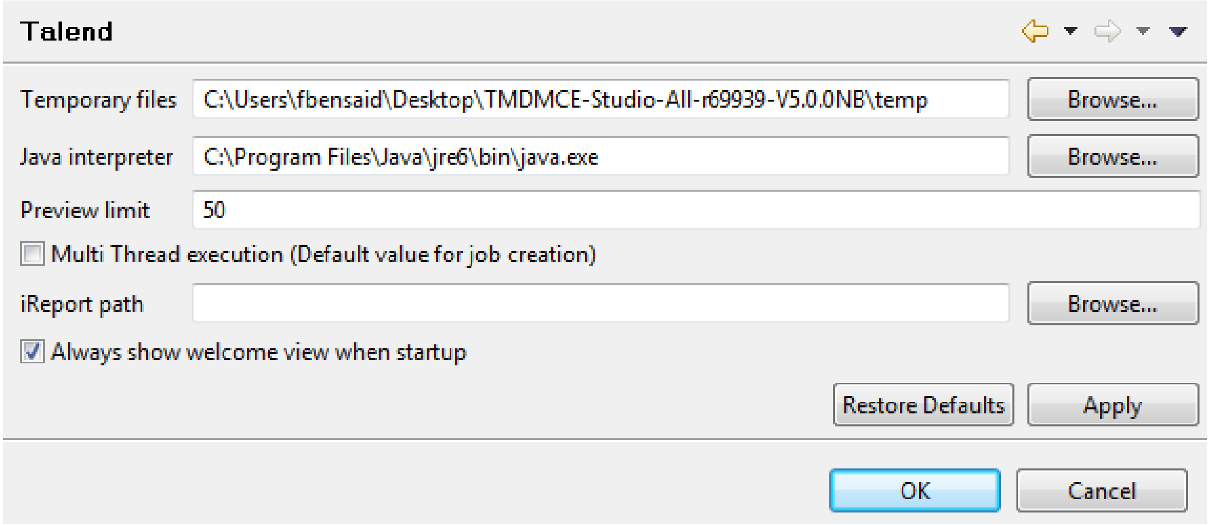
Numerous settings you define can be stored in the **Preference** and thus become your default values for all new Jobs you create.

The following sections describe specific settings that you can set as preference.

First, click the **Window** menu of your *Talend Open Studio for Big Data*, then select **Preferences**.

### 2.5.1. Java Interpreter path (Talend)

The Java Interpreter path is set default in the Java file of your computer (by default Program Files\Java\jre6\bin\java.exe).



Temporary files: C:\Users\fbensaid\Desktop\TMDMCE-Studio-All-r69939-V5.0.0NB\temp

Java interpreter: C:\Program Files\Java\jre6\bin\java.exe

Preview limit: 50

□ Multi Thread execution (Default value for job creation)

iReport path:

☑ Always show welcome view when startup

To customize your Java Interpreter path:

1. If needed, click the **Talend** node in the tree view of the **[Preferences]** dialog box.

2. Enter a path in the **Java interpreter** field if the default directory does not display the right path.

On the same view, you can also change the preview limit and the path to the temporary files or the OS language.

## 2.5.2. Designer preferences (Talend > Appearance)

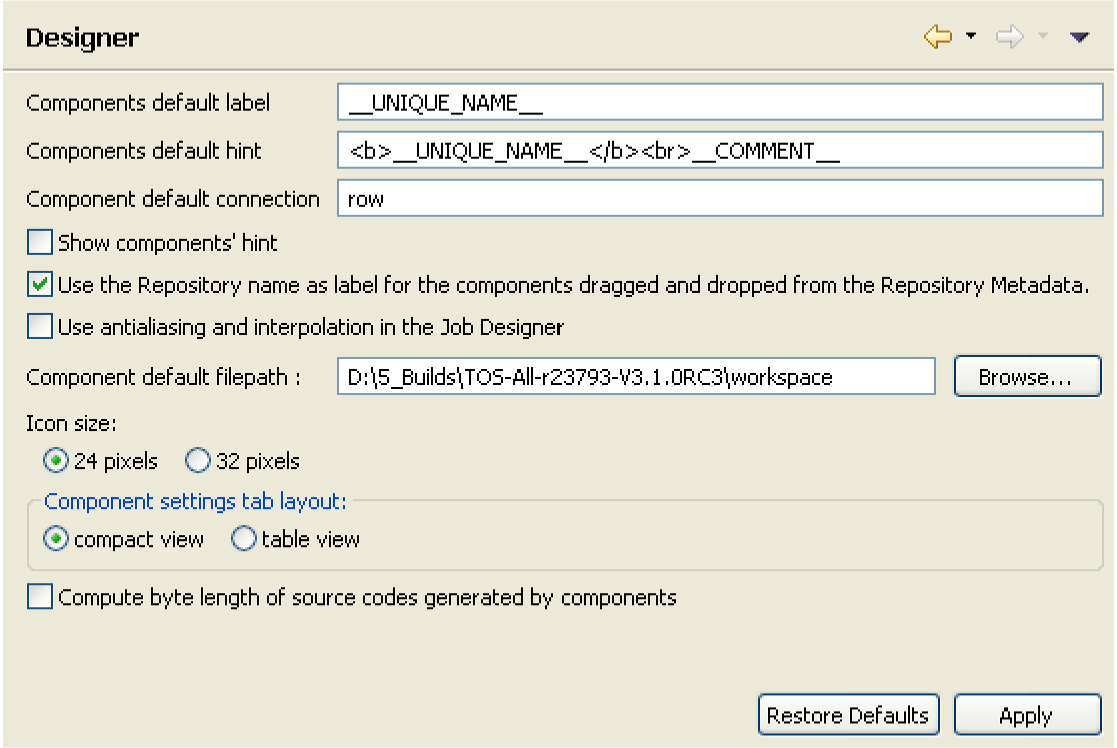
You can set component and Job design preferences to let your settings be permanent in the Studio.

1. From the menu bar, click **Window > Preferences** to open the **[Preferences]** dialog box.

2. Expand the **Talend** > **Appearance** node.

3. Click **Designer** to display the corresponding view.

   On this view, you can define the way component names and hints will be displayed.



Designer

Components default label: __UNIQUE_NAME__

Components default hint: <b>__UNIQUE_NAME__</b><br>__COMMENT__

Component default connection: row

□ Show components' hint

☑ Use the Repository name as label for the components dragged and dropped from the Repository Metadata.

□ Use antialiasing and interpolation in the Job Designer

Component default filepath : D:\5_Builds\TOS-All-r23793-V3.1.0RC3\workspace

Icon size:

● 24 pixels ○ 32 pixels

Component settings tab layout:

● compact view ○ table view

□ Compute byte length of source codes generated by components

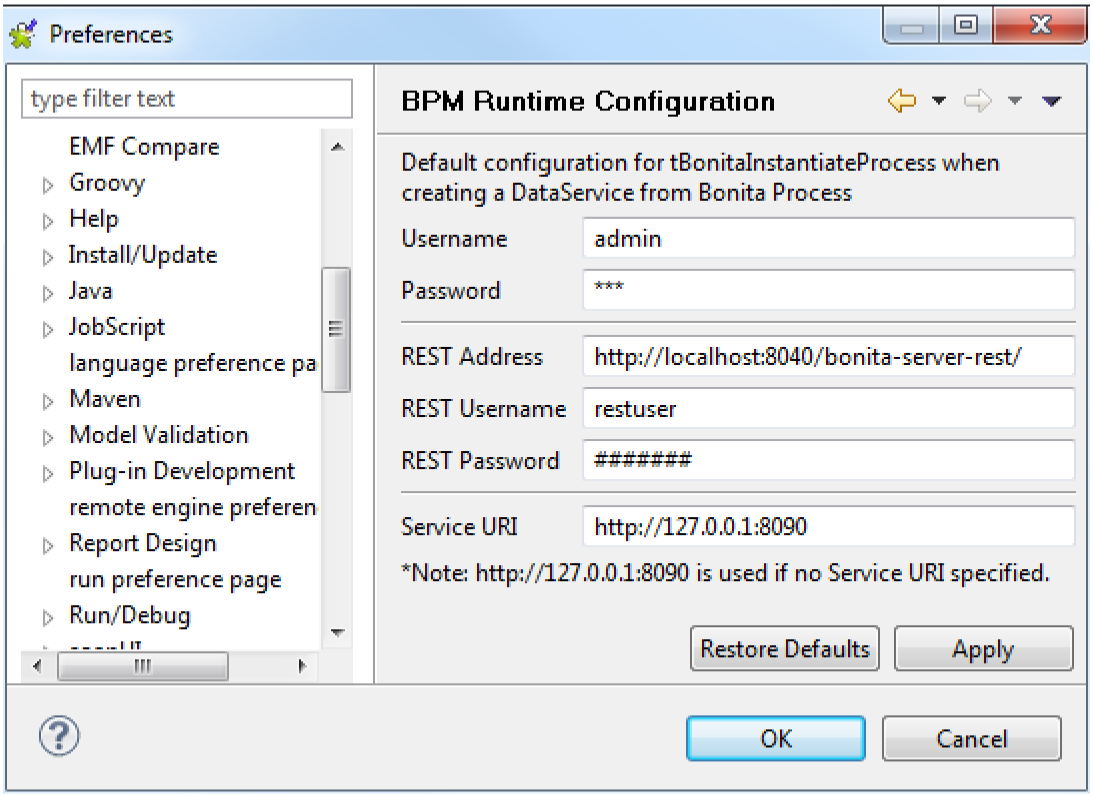
4. Select the relevant check boxes to customize your use of *Talend Open Studio for Big Data* design workspace.

## 2.5.3. BPM Runtime preferences (Talend > BPM Runtime Configuration)

When creating a BPM service, you can set its URI as well as the connection information to the BPM Web console.

1. From the menu bar, click **Window > Preferences** to open the **[Preferences]** dialog box.

2. Expand the **Talend** > **BPM Runtime Configuration** node.



BPM Runtime Configuration

Default configuration for tBonitaInstantiateProcess when creating a DataService from Bonita Process

Username: admin

Password: ***

REST Address: http://localhost:8040/bonita-server-rest/

REST Username: restuser

REST Password: #######

Service URI: http://127.0.0.1:8090

*Note: http://127.0.0.1:8090 is used if no Service URI specified.

3. Fill in the information as follows.

| Field Name | Action |
|---|---|
| Username and Password | Enter the username and password to connect to the BPM Web console. By default, it is *admin* and *bpm*. |
| REST Address | Enter the URL of the BPM REST server. By default, it is *http://localhost:8040/bonita-server-rest/*. |
| REST Username and REST Password | Enter the username and password to connect to the BPM REST server. By default, it is *restuser* and *restbpm*. |
| Service URI | Enter the URI of the BPM service. By default, it is *http://127.0.0.1:8090*. Note that this default URI will be used if no service URI is specified. |

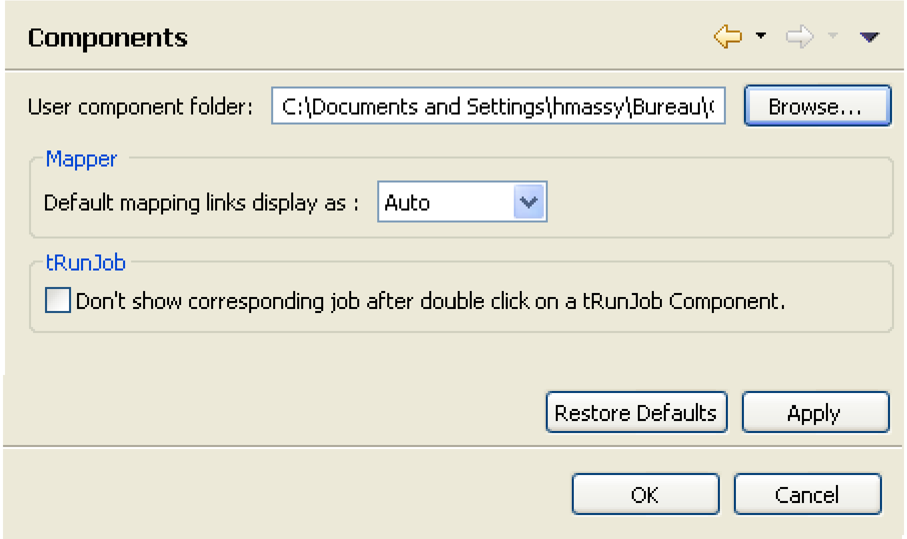
4. Click **Apply** and then **OK** to validate the set preferences and close the dialog box.

## 2.5.4. External or User components (Talend > Components)

You can create and develop your own components for use in *Talend Open Studio for Big Data*.

For further information about the creation and development of user components, refer to the component creation tutorial on our wiki at http://www.talendforge.org/wiki/doku.php?id=component_creation.

1. In the tree view of the **[Preferences]** dialog box, expand the **Talend** node and select **Components**.



User component folder: C:\Documents and Settings\hmassy\Bureau\(

Default mapping links display as : Auto

□ Don't show corresponding job after double click on a tRunJob Component.

2. Enter the **User components folder** path or browse to the folder that holds the components to be added to the *Talend Open Studio for Big Data* **Palette**.

3. From the **Default mapping links display as** list, select the mapping link type you want to use in the **tMap**.

4. Under **tRunJob**, select the check box if you do not want the corresponding Job to open upon double clicking a **tRunJob** component.

   You will still be able to open the corresponding Job by right clicking the **tRunJob** component and selecting **Open tRunJob Component**.

5. Click **Apply** and then **OK** to validate the set preferences and close the dialog box.

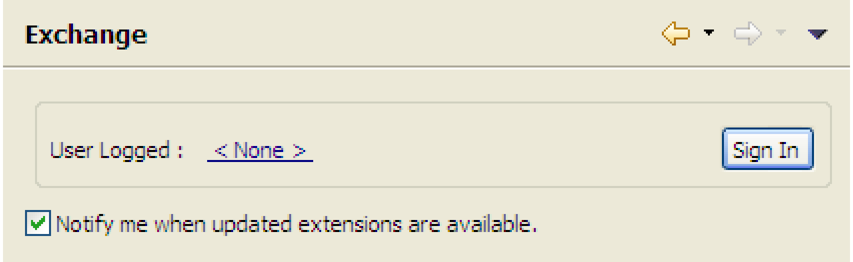
   The external components are added to the **Palette**.

## 2.5.5. Exchange preferences (Talend > Exchange)

You can set preferences related to your connection with **Talend** Exchange, which is part of the **Talend** Community, in *Talend Open Studio for Big Data*. To do so:

1. From the menu bar, click **Window** > **Preferences** to open the **[Preferences]** dialog box.

2. Expand the **Talend** node and click **Exchange** to display the **Exchange** view.



User Logged : < None >

☑ Notify me when updated extensions are available.

3. Set the Exchange preferences according to your needs:

   - If you are not yet connected to the **Talend** Community, click **Sign In** to go to the **Connect to TalendForge** page to sign in using your **Talend** Community credentials or create a **Talend** Community account and then sign in.

If you are already connected to the **Talend** Community, your account is displayed and the **Sign In** button becomes **Sign Out**. To get disconnected from the **Talend** Community, click **Sign Out**.

- By default, while you are connected to the **Talend** Community, whenever an update to an installed community extension is available, a dialog box appears to notify you about it. If you often check for community extension updates and you do not want that dialog box to appear again, clear the **Notify me when updated extensions are available** check box.

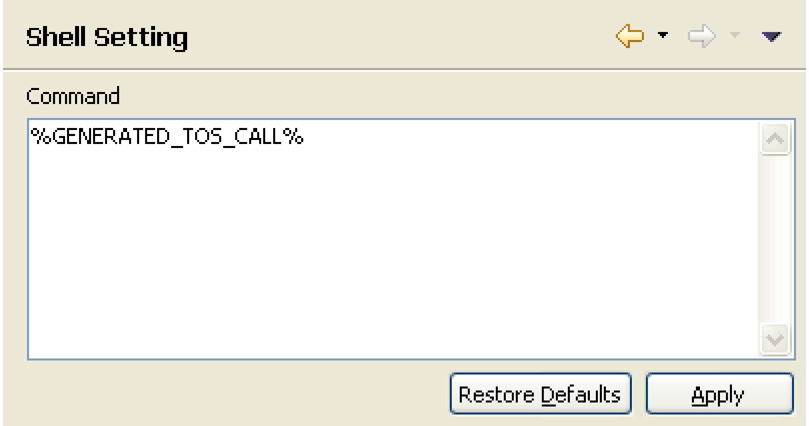
For more information on connecting to the **Talend** Community, see section *Launching Talend Open Studio for Big Data*. For more information on using community extensions in the Studio, see section *How to download/upload Talend Community components*.

## 2.5.6. Adding code by default (Talend > Import/Export)

You can add pieces of code by default at the beginning and at the end of the code of your Job.

1. From the menu bar, click **Window** > **Preferences** to open the **[Preferences]** dialog box.

2. Expand the **Talend** and **Import/Export** nodes in succession and then click **Shell Setting** to display the relevant view.



Shell Setting

Command

```
%GENERATED_TOS_CALL%
```

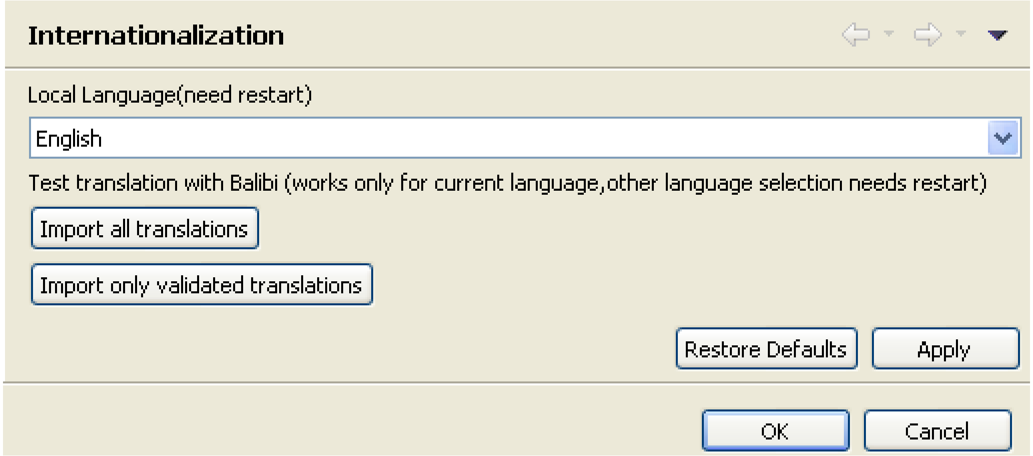
3. In the **Command** field, enter your piece/pieces of code before or after `%GENERATED_TOS_CALL%` to display it/them before or after the code of your Job.

## 2.5.7. Language preferences (Talend > Internationalization)

You can set language preferences in *Talend Open Studio for Big Data*. To do so:

1. From the menu bar, click **Window** > **Preferences** to open the **[Preferences]** dialog box.

2. Expand the **Talend** node and click **Internationalization** to display the relevant view.



Internationalization

Local Language(need restart)

English

Test translation with Balibi (works only for current language,other language selection needs restart)

3. From the **Local Language** list, select the language you want to use for *Talend Open Studio for Big Data* graphical interface.

4. Click **Apply** and then **OK** to validate your change and close the **[Preferences]** dialog box.

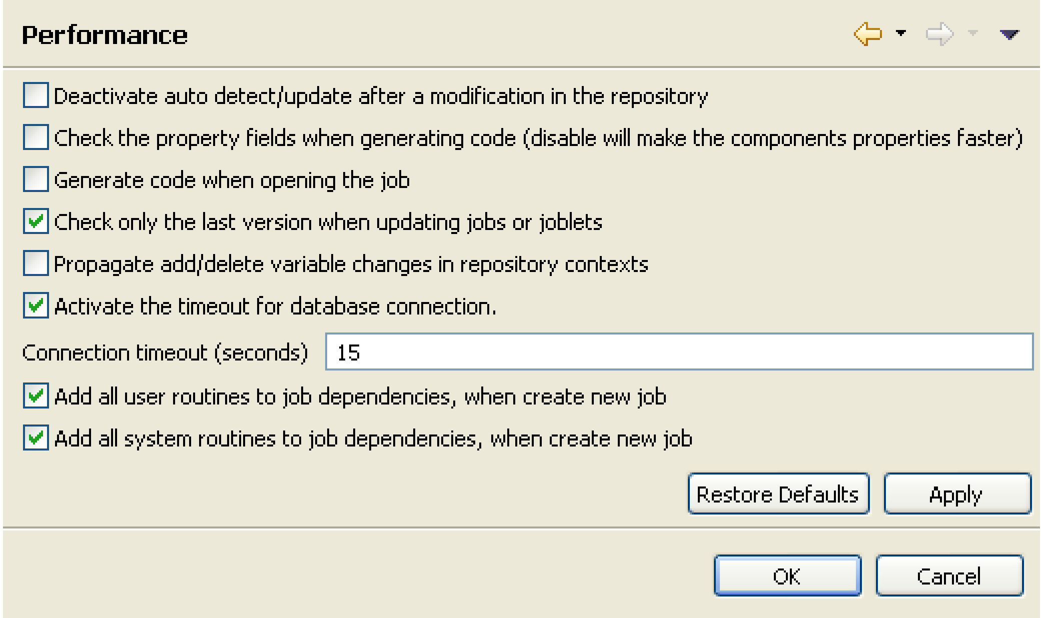
5. Restart *Talend Open Studio for Big Data* to display the graphical interface in the selected language.

## 2.5.8. Performance preferences (Talend > Performance)

You can set the **Repository** tree view preferences according to your use of *Talend Open Studio for Big Data*. To refresh the **Repository** view:

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.



Performance

□ Deactivate auto detect/update after a modification in the repository

□ Check the property fields when generating code (disable will make the components properties faster)

□ Generate code when opening the job

☑ Check only the last version when updating jobs or joblets

□ Propagate add/delete variable changes in repository contexts

☑ Activate the timeout for database connection.

Connection timeout (seconds) 15

☑ Add all user routines to job dependencies, when create new job

☑ Add all system routines to job dependencies, when create new job

You can improve your performance when you deactivate automatic refresh.

3. Set the performance preferences according to your use of *Talend Open Studio for Big Data*:

- Select the **Deactivate auto detect/update after a modification in the repository** check box to deactivate the automatic detection and update of the repository.
- Select the **Check the property fields when generating code** check box to activate the audit of the property fields of the component. When one property filed is not correctly filled in, the component is surrounded by red on the design workspace.

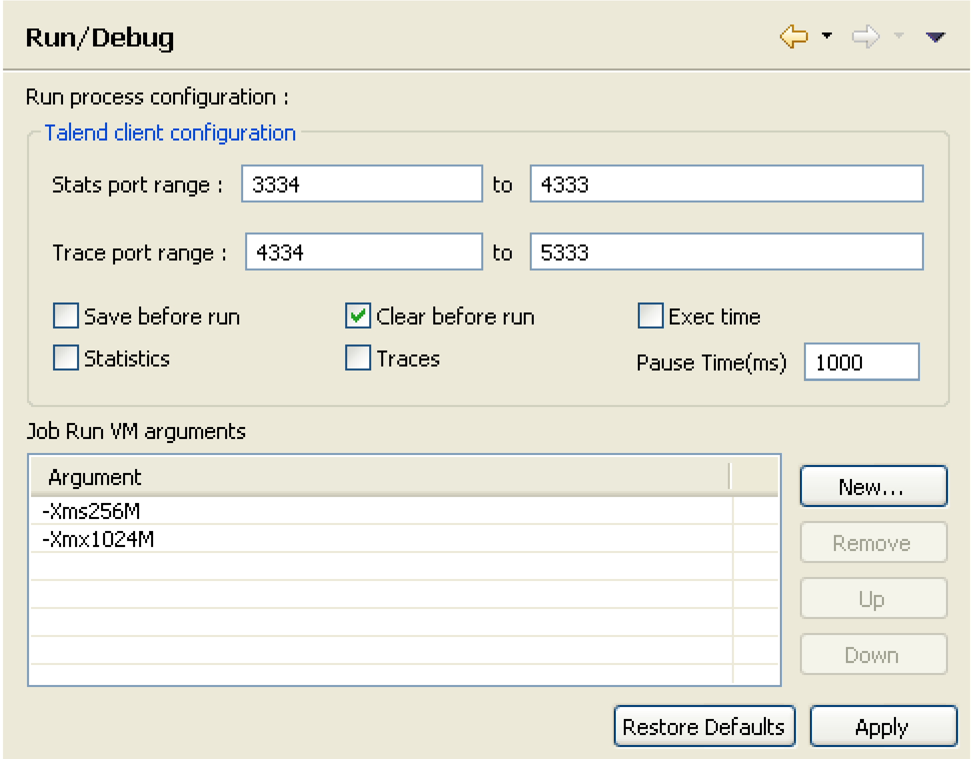
> You can optimize performance if you disable property fields verification of components, i.e. if you clear the **Check the property fields when generating code** check box.

- Select the **Generate code when opening the job** check box to generate code when you open a Job.
- Select the **Check only the last version when updating jobs or joblets** check box to only check the latest version when you update a Job.
- Select the **Propagate add/delete variable changes in repository contexts** to propagate variable changes in the Repository Contexts.
- Select the **Activate the timeout for database connection** check box to establish database connection time out. Then set this time out in the **Connection timeout (seconds)** field.
- Select the **Add all user routines to job dependencies, when create new job** check box to add all user routines to Job dependencies upon the creation of new Jobs.
- Select the **Add all system routines to job dependencies, when create job** check box to add all system routines to Job dependencies upon the creation of new Jobs.

## 2.5.9. Debug and Job execution preferences (Talend > Run/Debug)

You can set your preferences for debug and job executions in *Talend Open Studio for Big Data*. To do so:

1. From the menu bar, click **Window** > **Preferences** to display the **[Preferences]** dialog box.
2. Expand the **Talend** node and click **Run/Debug** to display the relevant view.



Run/Debug

Run process configuration :

Talend client configuration

Stats port range : 3334 to 4333

Trace port range : 4334 to 5333

□ Save before run ☑ Clear before run □ Exec time

□ Statistics □ Traces Pause Time(ms) 1000

Job Run VM arguments

| Argument |
|---|
| -Xms256M |
| -Xmx1024M |

- In the **Talend client configuration** area, you can define the execution options to be used by default:

| | |
|---|---|
| **Stats port range** | Specify a range for the ports used for generating statistics, in particular, if the ports defined by default are used by other applications. |
| **Trace port range** | Specify a range for the ports used for generating traces, in particular, if the ports defined by default are used by other applications. |
| **Save before run** | Select this check box to save your Job automatically before its execution. |
| **Clear before run** | Select this check box to delete the results of a previous execution before re-executing the Job. |
| **Exec time** | Select this check box to show Job execution duration. |
| **Statistics** | Select this check box to show the statistics measurement of data flow during Job execution. |
| **Traces** | Select this check box to show data processing during job execution. |
| **Pause time** | Enter the time you want to set before each data line in the traces table. |

- In the **Job Run VM arguments** list, you can define the parameter of your current JVM according to your needs. The by-default parameters **-Xms256M** and **-Xmx1024M** correspond respectively to the minimal and maximal memory capacities reserved for your Job executions.

  If you want to use some JVM parameters for only a specific Job execution, for example if you want to display the execution result for this specific Job in Japanese, you need open this Job's **Run** view and then in the **Run** view, configure the advanced execution settings to define the corresponding parameters.

For further information about the advanced execution settings of a specific Job, see section *How to set advanced execution settings*.

For more information about possible parameters, check the site http://www.oracle.com/technetwork/java/javase/tech/vmoptions-jsp-140102.html.

## 2.5.10. Displaying special characters for schema columns (Talend > Specific settings)

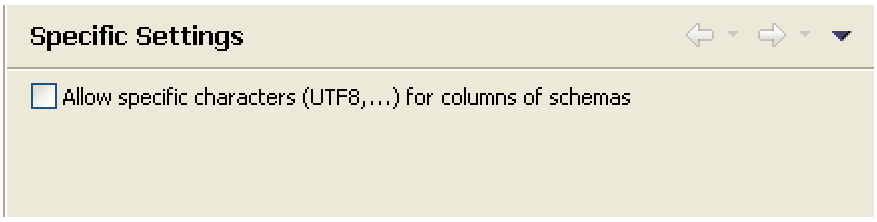
You may need to retrieve a table schema that contains columns written with special characters like Chinese, Japanese, Korean. In this case, you need to enable *Talend Open Studio for Big Data* to read the special characters. To do so:

1. From the menu bar, click **Window > Preferences** to open the **[Preferences]** dialog box.

2. On the tree view of the opened dialog box, expand the **Talend** node.

3. Click the **Specific settings** node to display the corresponding view on the right of the dialog box.

4. Select the **Allow specific characters (UTF8,...) for columns of schemas** check box.



Specific Settings

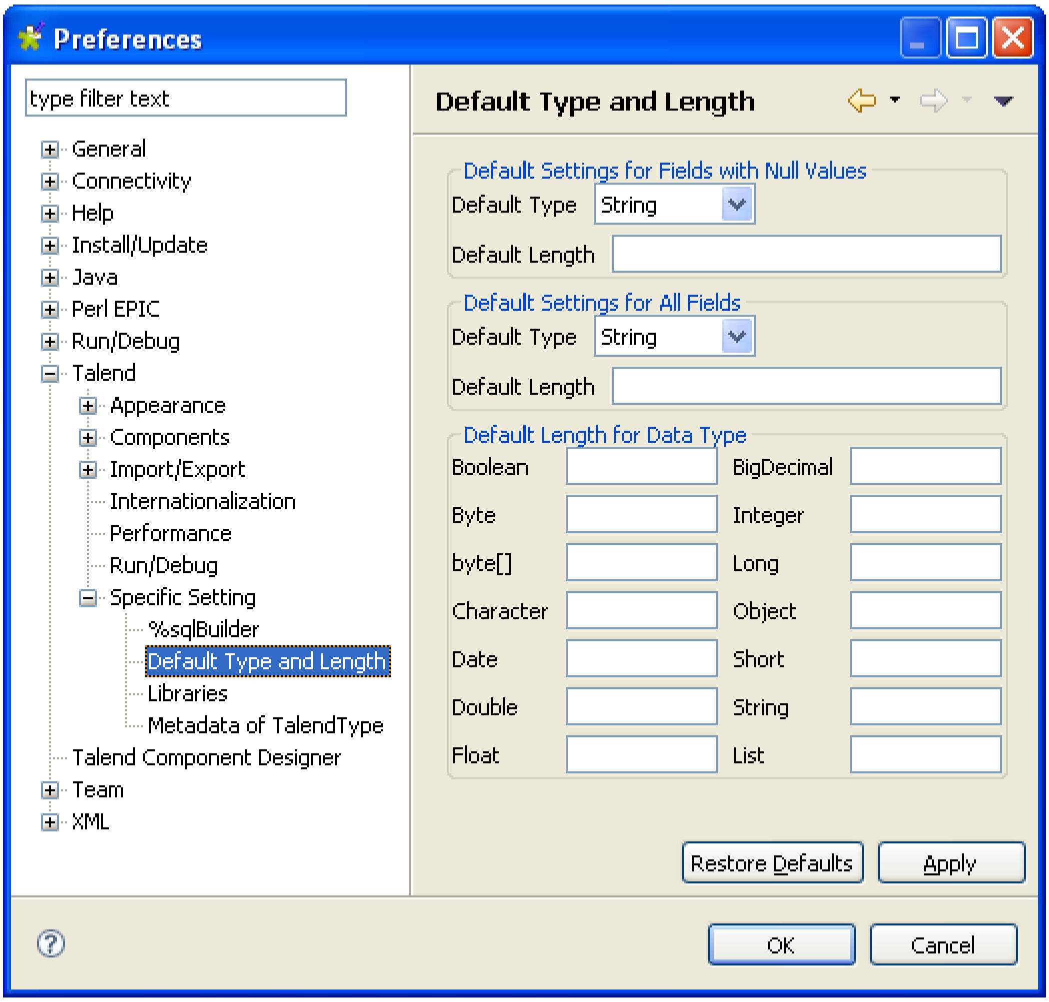
□ Allow specific characters (UTF8,...) for columns of schemas

## 2.5.11. Schema preferences (Talend > Specific Settings)

You can define the default data length and type of the schema fields of your components.

1. From the menu bar, click **Window** > **Preferences** to open the **[Preferences]** dialog box.

2. Expand the **Talend** node, and click **Specific Settings > Default Type and Length** to display the data length and type of your schema.



Default Type and Length

Default Settings for Fields with Null Values

Default Type: String

Default Length:

Default Settings for All Fields

Default Type: String

Default Length:

Default Length for Data Type

| | | | |
|---|---|---|---|
| Boolean | | BigDecimal | |
| Byte | | Integer | |
| byte[] | | Long | |
| Character | | Object | |
| Date | | Short | |
| Double | | String | |
| Float | | List | |

3. Set the parameters according to your needs:

   - In the **Default Settings for Fields with Null Values** area, fill in the data type and the field length to apply to the null fields.

   - In the **Default Settings for All Fields** area, fill in the data type and the field length to apply to all fields of the schema.

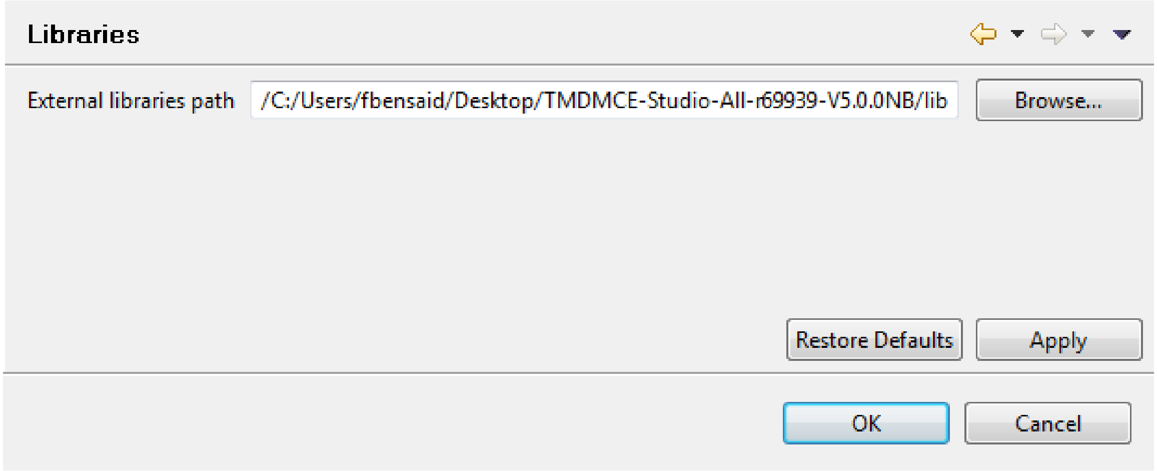
   - In the **Default Length for Data Type** area, fill in the field length for each type of data.

## 2.5.12. Libraries preferences (Talend > Specific Settings)

You can define the folder where to store the different libraries used in *Talend Open Studio for Big Data*. To do so:

1. From the menu bar, click **Window** > **Preferences** to display the **[Preferences]** dialog box.

2. Expand the **Talend** and **Specific Settings** nodes in succession and then click **Libraries** to display the relevant view.



Libraries

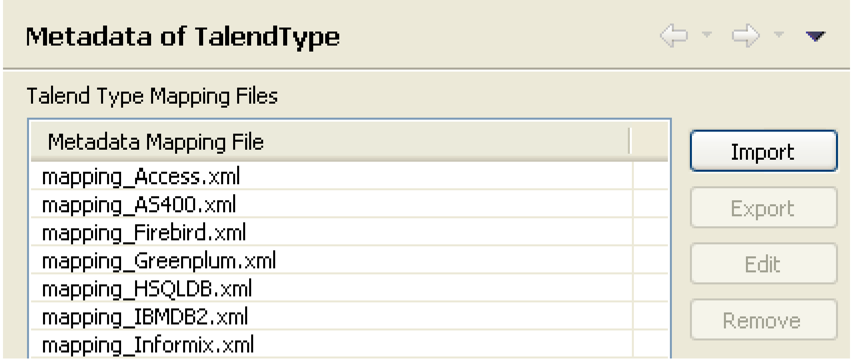
External libraries path /C:/Users/fbensaid/Desktop/TMDMCE-Studio-All-r69939-V5.0.0NB/lib

3. Set the access path in the **External libraries path** field through the **Browse...** button. The default path leads to the library of your current build.

## 2.5.13. Type conversion (Talend > Specific Settings)

You can set the parameters for type conversion in *Talend Open Studio for Big Data*, from Java towards databases and vice versa.

1. From the menu bar, click **Window** > **Preferences** to display the **[Preferences]** dialog box.
2. Expand the **Talend** and **Specific Settings** nodes in succession and then click **Metadata of Talend Type** to display the relevant view.



Metadata of TalendType

Talend Type Mapping Files

| Metadata Mapping File |
|---|
| mapping_Access.xml |
| mapping_AS400.xml |
| mapping_Firebird.xml |
| mapping_Greenplum.xml |
| mapping_HSQLDB.xml |
| mapping_IBMDB2.xml |
| mapping_Informix.xml |

The **Metadata Mapping File** area lists the XML files that hold the conversion parameters for each database type used in *Talend Open Studio for Big Data*.

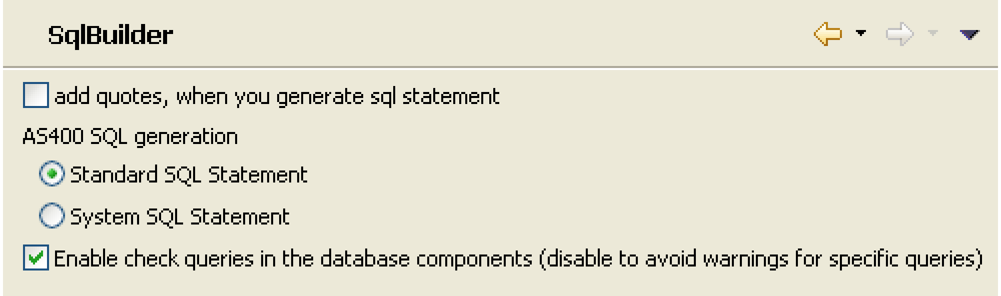
- You can import, export, or delete any of the conversion files by clicking **Import**, **Export** or **Remove** respectively.
- You can modify any of the conversion files according to your needs by clicking the **Edit** button to open the **[Edit mapping file]** dialog box and then modify the XML code directly in the open dialog box.

## 2.5.14. SQL Builder preferences (Talend > Specific Settings)

You can set your preferences for the SQL Builder. To do so:

1. From the menu bar, click **Window** > **Preferences** to open the **[Preferences]** dialog box.

2. Expand the **Talend** and **Specific Settings** nodes in succession and then click **Sql Builder** to display the relevant view.



SqlBuilder

☐ add quotes, when you generate sql statement

AS400 SQL generation

● Standard SQL Statement

○ System SQL Statement

☑ Enable check queries in the database components (disable to avoid warnings for specific queries)

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 AS400 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.

## 2.5.15. Usage Data Collector preferences (Talend > Usage Data Collector)

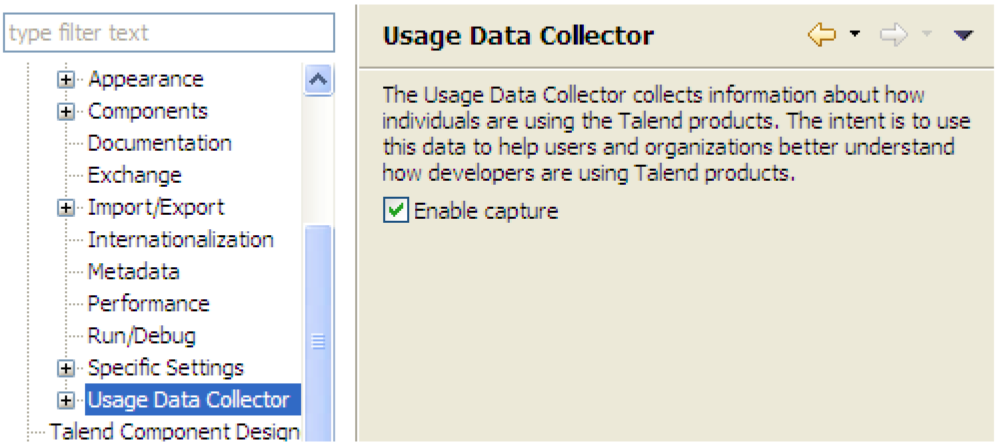
By allowing *Talend Open Studio for Big Data* to collect your Studio usage statistics, you help users better understand **Talend** products and help **Talend** better learn how users are using the products, thus enabling **Talend** to improve product quality and performance to serve users better.

By default, *Talend Open Studio for Big Data* automatically collects your Studio usage data and sends this data on a regular basis to servers hosted by **Talend**. You can view the usage data collection and upload information and customize the Usage Data Collector preferences according to your needs.

Be assured that only the Studio usage statistics data will be collected and none of your private information will be collected and transmitted to **Talend**.

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.



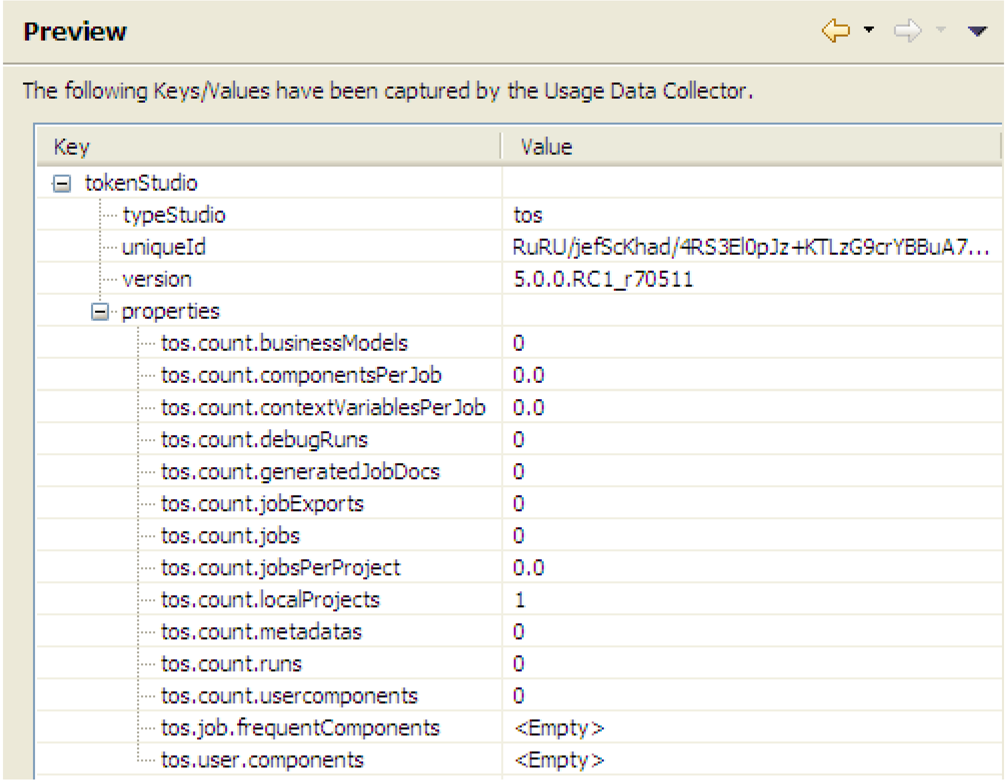
Usage Data Collector

The Usage Data Collector collects information about how individuals are using the Talend products. The intent is to use this data to help users and organizations better understand how developers are using Talend products.

☑ Enable capture

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**.

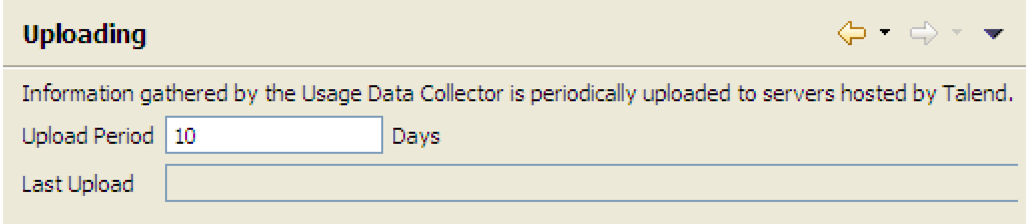


**Preview**

The following Keys/Values have been captured by the Usage Data Collector.

| Key | Value |
|---|---|
| tokenStudio | |
| typeStudio | tos |
| uniqueId | RuRU/jefScKhad/4RS3El0pJz+KTLzG9crYBBuA7... |
| version | 5.0.0.RC1_r70511 |
| properties | |
| tos.count.businessModels | 0 |
| tos.count.componentsPerJob | 0.0 |
| tos.count.contextVariablesPerJob | 0.0 |
| tos.count.debugRuns | 0 |
| tos.count.generatedJobDocs | 0 |
| tos.count.jobExports | 0 |
| tos.count.jobs | 0 |
| tos.count.jobsPerProject | 0.0 |
| tos.count.localProjects | 1 |
| tos.count.metadatas | 0 |
| tos.count.runs | 0 |
| tos.count.usercomponents | 0 |
| tos.job.frequentComponents | <Empty> |
| 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 10 Days

Last Upload

- 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.

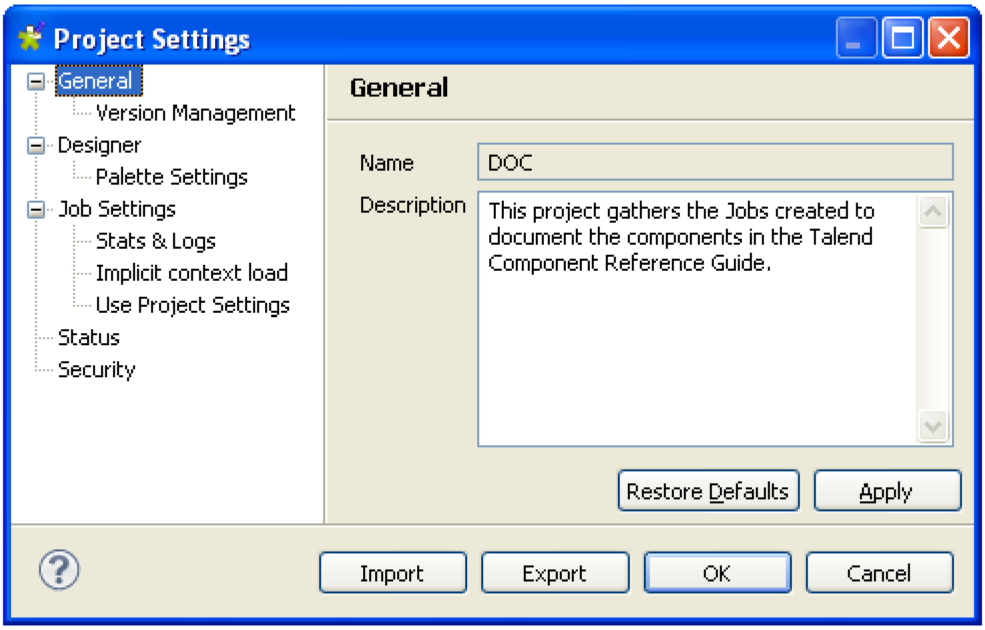
## 2.6. Customizing project settings

*Talend Open Studio for Big Data* enables you to customize the information and settings of the project in progress, including the **Palette**, Job settings, for example.

To customize project settings:

1. Click on the Studio tool bar, or select **File** > **Edit Project Properties** from the menu bar.

The **[Project Settings]** dialog box opens.



Name: DOC

Description: This project gathers the Jobs created to document the components in the Talend Component Reference Guide.

2. In the tree diagram to the left of the dialog box, select the setting you wish to customize and then customize it, using the options that appear to the right of the box.

From the dialog box you can also export or import the full assemblage of settings that define a particular project:

- To export the settings, click on the **Export** button. The export will generate an XML file containing all of your project settings.

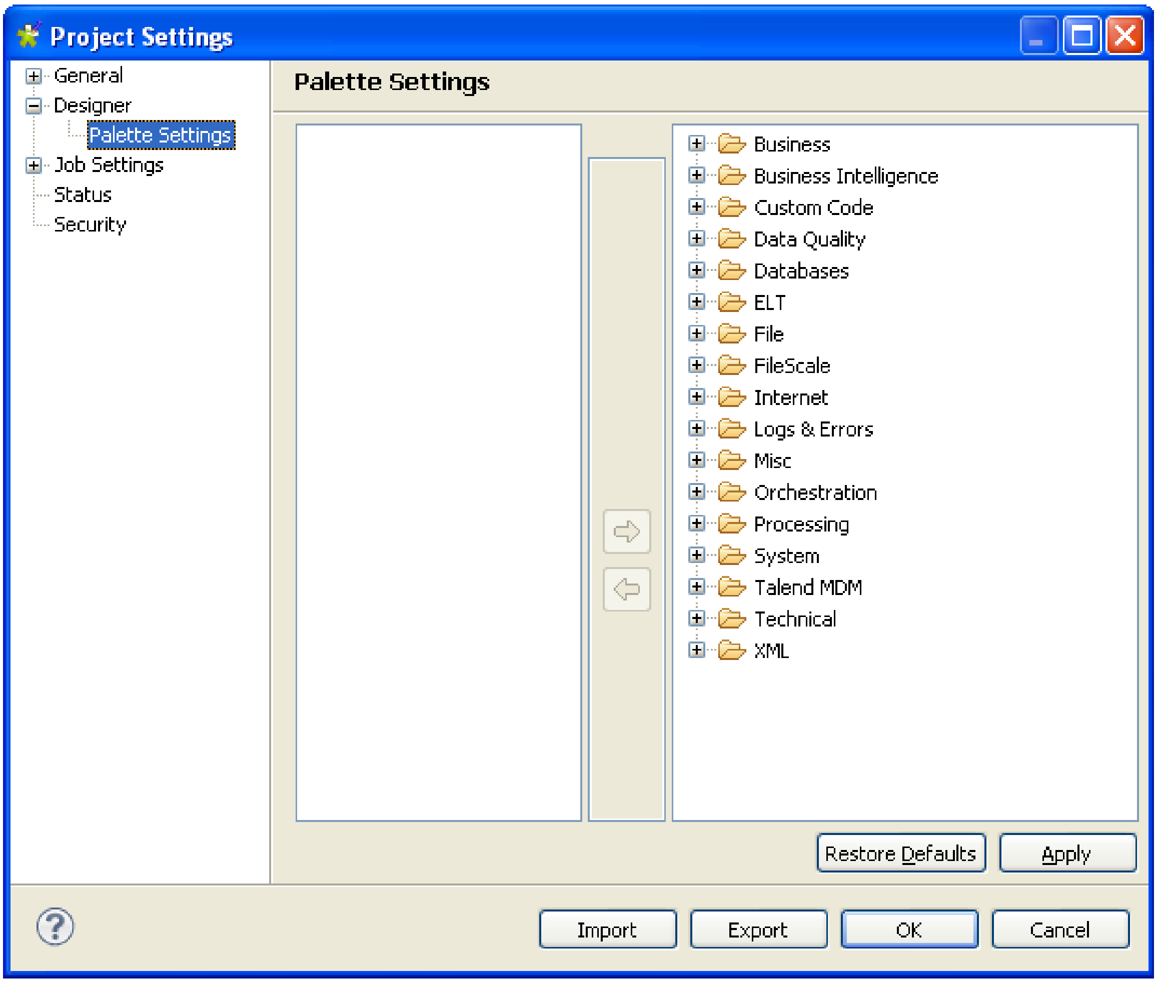
- To import settings, click on the **Import** button and select the XML file containing the parameters of the project which you want to apply to the current project.

## 2.6.1. Palette Settings

You can customize the settings of the **Palette** display so that only the components used in the project are loaded. This will allow you to launch the Studio more quickly.

To customize the **Palette** display settings:

1. On the toolbar of the Studio's main window, click  or click **File** > **Edit Project Properties** on the menu bar to open the **[Project Settings]** dialog box.



In the **General** view of the **[Project Settings]** dialog box, you can add a project description, if you did not do so when creating the project.

2. In the tree view of the **[Project Settings]** dialog box, expand **Designer** and select **Palette Settings**. The settings of the current **Palette** are displayed in the panel to the right of the dialog box.

3. Select one or several components, or even set(s) of components you want to remove from the current project's **Palette**.

4. Use the left arrow button to move the selection onto the panel on the left. This will remove the selected components from the **Palette**.

5. To re-display hidden components, select them in the panel on the left and use the right arrow button to restore them to the **Palette**.

6. Click **Apply** to validate your changes and **OK** to close the dialog box.

To get back to the **Palette** default settings, click **Restore Defaults**.

For more information on the **Palette**, see section *How to change the Palette layout and settings*.

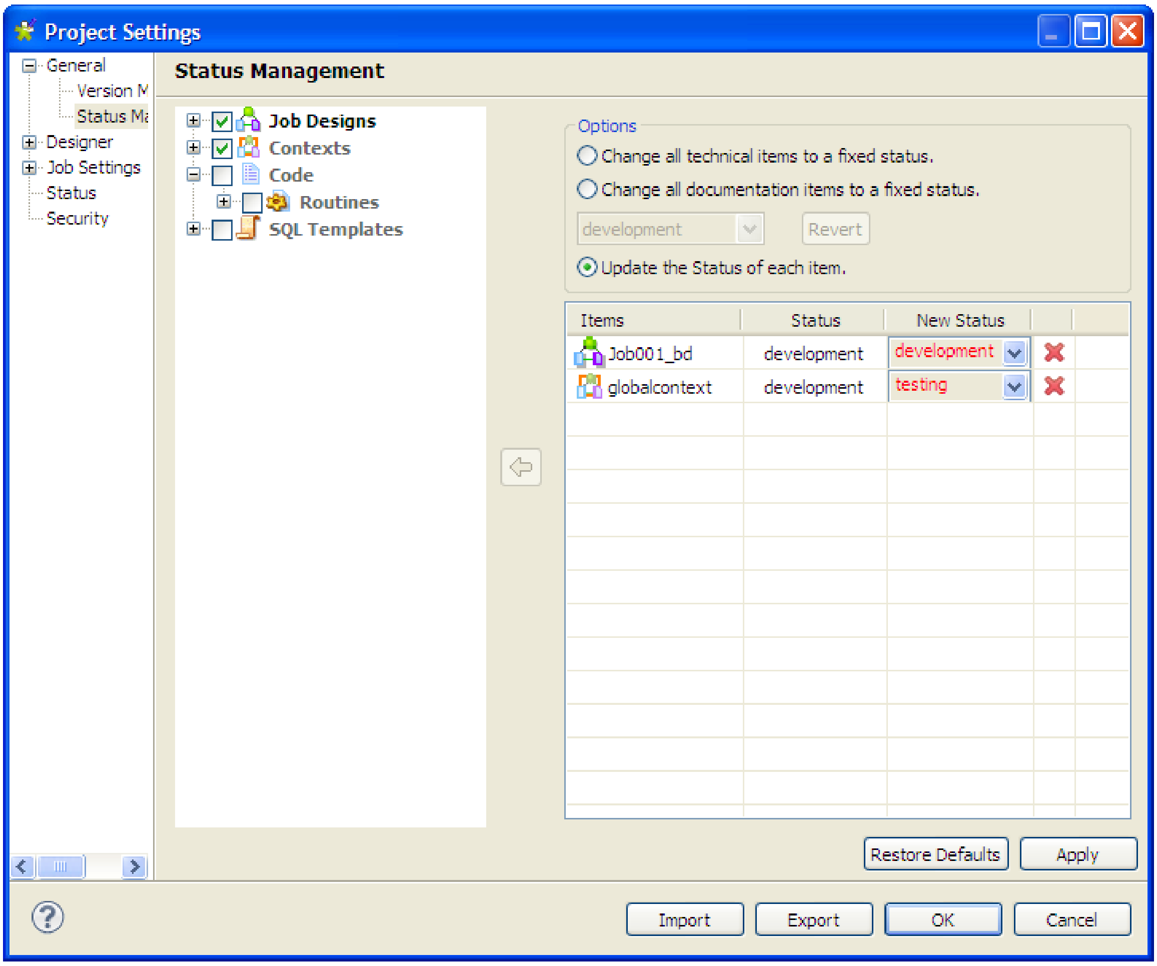
## 2.6.2. Status management

You can also manage the status of each item in the **Repository** tree view through **General** > **Status Management** of the **[Project Settings]** dialog box.

To do so:

1. On the toolbar of the Studio main window, click or click **File** > **Edit Project Properties** from the menu bar to open the **[Project Settings]** dialog box.

2. In the tree view of the dialog box, expand **General** and select **Status Management** to open the corresponding view.



☑ Job Designs
☑ Contexts
□ Code
□ Routines
□ SQL Templates

Options
○ Change all technical items to a fixed status.
○ Change all documentation items to a fixed status.
development Revert
● Update the Status of each item.

| Items | Status | New Status | |
|---|---|---|---|
| Job001_bd | development | development | ✖ |
| globalcontext | development | testing | ✖ |

3. In the **Repository** tree view, expand the node holding the items you want to manage their status and then select the check boxes of these items.

   The selected items display in the **Items** list to the right along with their current status in the **Status** column and the new status set in the **New Status** column.

4. In the **Options** area, select the **Change all technical items to a fixed status** check box to change the status of the selected items to the same fixed status.

5. Click **Revert** if you want to undo the changes.

6. To increment each status of the items, select the **Update the version of each item** check box and change them manually.

7. Click **Apply** to apply your changes and then **OK** to close the dialog box.

For further information about Job status, see section *Status settings*.

## 2.6.3. Job Settings

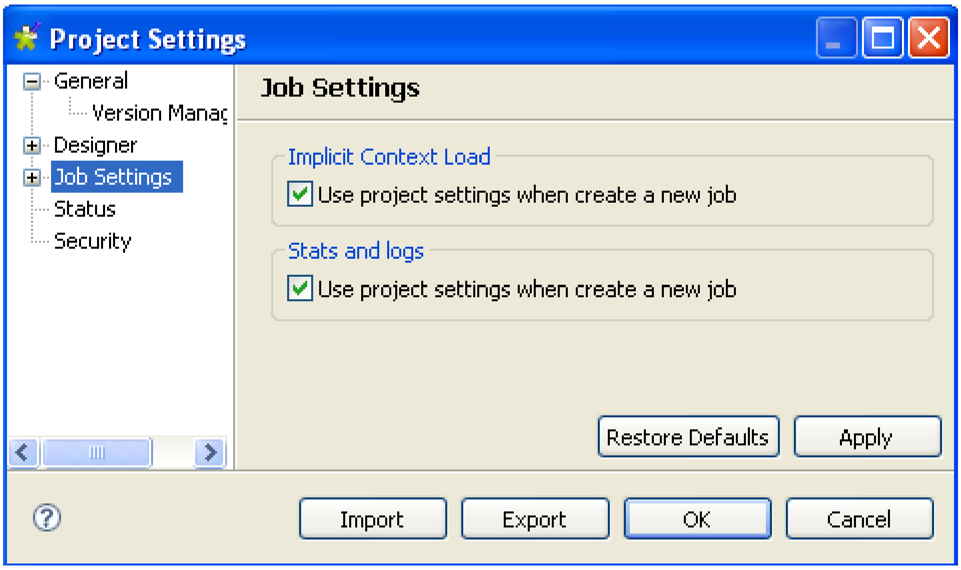
You can automatically use **Implicit Context Load** and **Stats and Logs** settings you defined in the **[Project Settings]** dialog box of the actual project when you create a new Job.

To do so:

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.



Job Settings

Implicit Context Load

☑ Use project settings when create a new job

Stats and logs

☑ Use project settings when create a new job

4. Click **Apply** to validate your changes and then **OK** to close the dialog box.

## 2.6.4. Stats & Logs

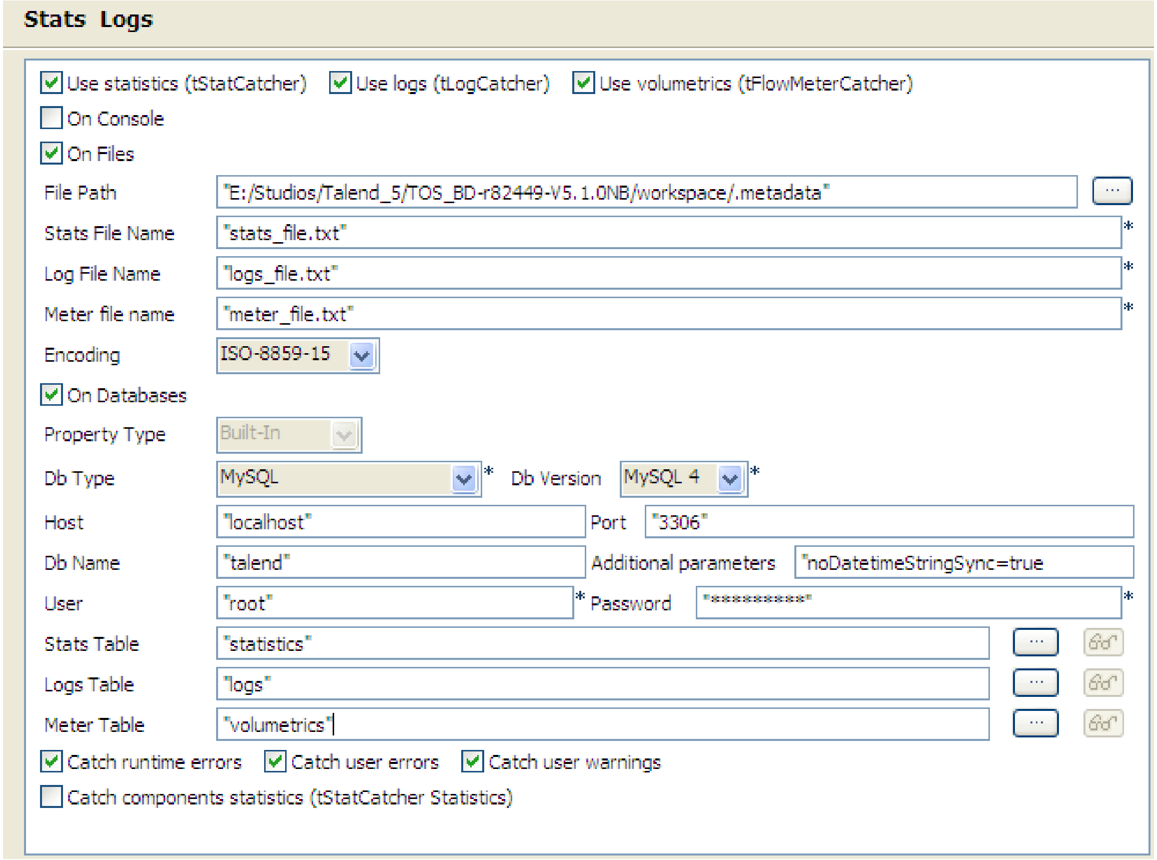
When you execute a Job, you can monitor the execution through the **tStatCatcher Statistics** option or through using a log component. This will enable you to store the collected log data in .csv files or in a database.

You can then set up the path to the log file and/or database once for good in the **[Project Settings]** dialog box so that the log data get always stored in this location.

To do so:

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**

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

□ On Console

☑ On Files

File Path: "E:/Studios/Talend_5/TOS_BD-r82449-V5.1.0NB/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: Built-In

Db Type: MySQL * Db Version: MySQL 4 *

Host: "localhost" Port: "3306"

Db Name: "talend" Additional parameters: "noDatetimeStringSync=true

User: "root" * Password: "**********" *

Stats Table: "statistics"

Logs Table: "logs"

Meter Table: "volumetrics"

☑ Catch runtime errors ☑ Catch user errors ☑ Catch user warnings

□ Catch components statistics (tStatCatcher Statistics)

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 section *How to automate the use of statistics & logs*.

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.

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.

Alternatively, if you save your connection information in a Context, you can also access them through **Ctrl+Space**.

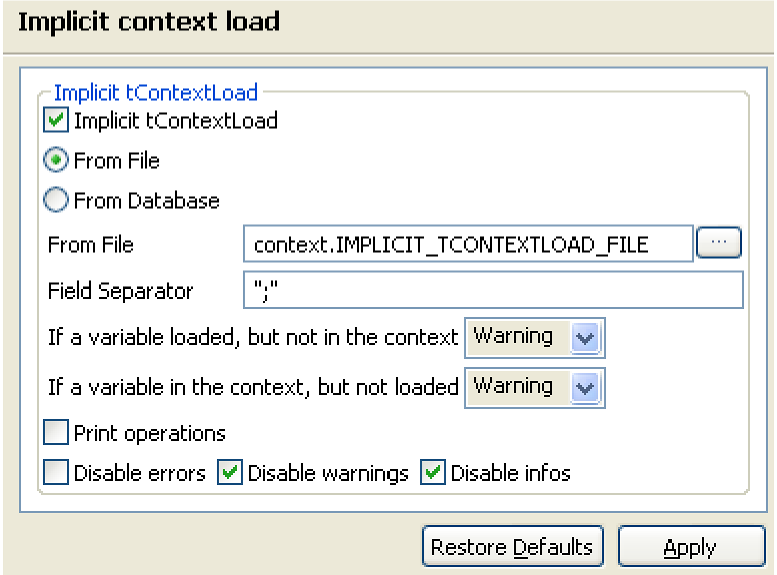
## 2.6.5. Context settings

You can define default context parameters you want to use in your Jobs.

To do so:

1. On the toolbar of the Studio main window, click or click **File** > **Edit Project Properties** from the menu bar to open the **[Project Settings]** dialog box.

2. In the tree view of the dialog box, expand the **Job Settings** node and then select the **Implicit Context Load** check box to display the configuration parameters of the Implicit **tContextLoad** feature.



Implicit context load

Implicit tContextLoad

☑ Implicit tContextLoad

● From File

○ From Database

From File: context.IMPLICIT_TCONTEXTLOAD_FILE

Field Separator: ";"

If a variable loaded, but not in the context: Warning

If a variable in the context, but not loaded: Warning

□ Print operations

□ Disable errors ☑ Disable warnings ☑ Disable infos

3. Select the **From File** or **From Database** check boxes according to the type of file you want to store your contexts in.

4. For files, fill in the file path in the **From File** field and the field separator in the **Field Separator** field.

5. For databases, select the **Built-in** or **Repository** mode in the **Property Type** list and fill in the next fields.

6. Fill in the **Table Name** and **Query Condition** fields.

7. Select the type of system message you want to have (warning, error, or info) in case a variable is loaded but is not in the context or vice versa.

8. Click **Apply** to validate your changes and then **OK** to close the dialog box.

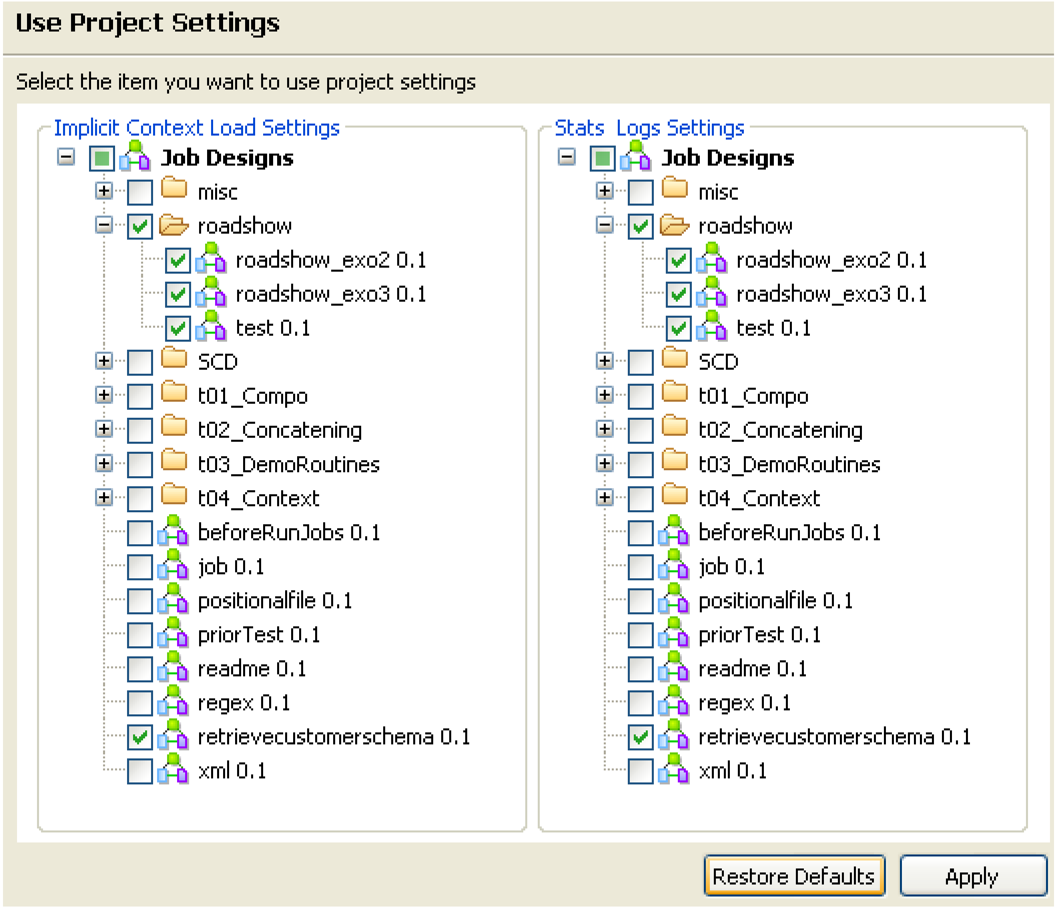
## 2.6.6. Project Settings use

From the **[Project Settings]** dialog box, you can choose to which Job in the **Repository** tree view you want to apply the **Implicit Context Load** and **Stats and Logs** settings.

To do so:

1. On the toolbar of the Studio main window, click or click **File** > **Edit Project Properties** from the menu bar to open the **[Project Settings]** dialog box.

2. In the tree view of the dialog box, expand the **Job Settings** node and then click **Use Project Settings** to display the use of **Implicit Context Load** and **Stats and Logs** option in the Jobs.



**Use Project Settings**

Select the item you want to use project settings

Implicit Context Load Settings

- ■ Job Designs
  - ☐ misc
  - ☑ roadshow
    - ☑ roadshow_exo2 0.1
    - ☑ roadshow_exo3 0.1
    - ☑ test 0.1
  - ☐ SCD
  - ☐ t01_Compo
  - ☐ t02_Concatening
  - ☐ t03_DemoRoutines
  - ☐ t04_Context
  - ☐ beforeRunJobs 0.1
  - ☐ job 0.1
  - ☐ positionalfile 0.1
  - ☐ priorTest 0.1
  - ☐ readme 0.1
  - ☐ regex 0.1
  - ☑ retrievecustomerschema 0.1
  - ☐ xml 0.1

Stats Logs Settings

- ■ Job Designs
  - ☐ misc
  - ☑ roadshow
    - ☑ roadshow_exo2 0.1
    - ☑ roadshow_exo3 0.1
    - ☑ test 0.1
  - ☐ SCD
  - ☐ t01_Compo
  - ☐ t02_Concatening
  - ☐ t03_DemoRoutines
  - ☐ t04_Context
  - ☐ beforeRunJobs 0.1
  - ☐ job 0.1
  - ☐ positionalfile 0.1
  - ☐ priorTest 0.1
  - ☐ readme 0.1
  - ☐ regex 0.1
  - ☑ retrievecustomerschema 0.1
  - ☐ xml 0.1

3. In the **Implicit Context Load Settings** area, select the check boxes corresponding to the Jobs in which you want to use the implicit context load option.

4. In the **Stats Logs Settings** area, select the check boxes corresponding to the Jobs in which you want to use the stats and logs option.

5. Click **Apply** to validate your changes and then **OK** to close the dialog box.

## 2.6.7. Status settings

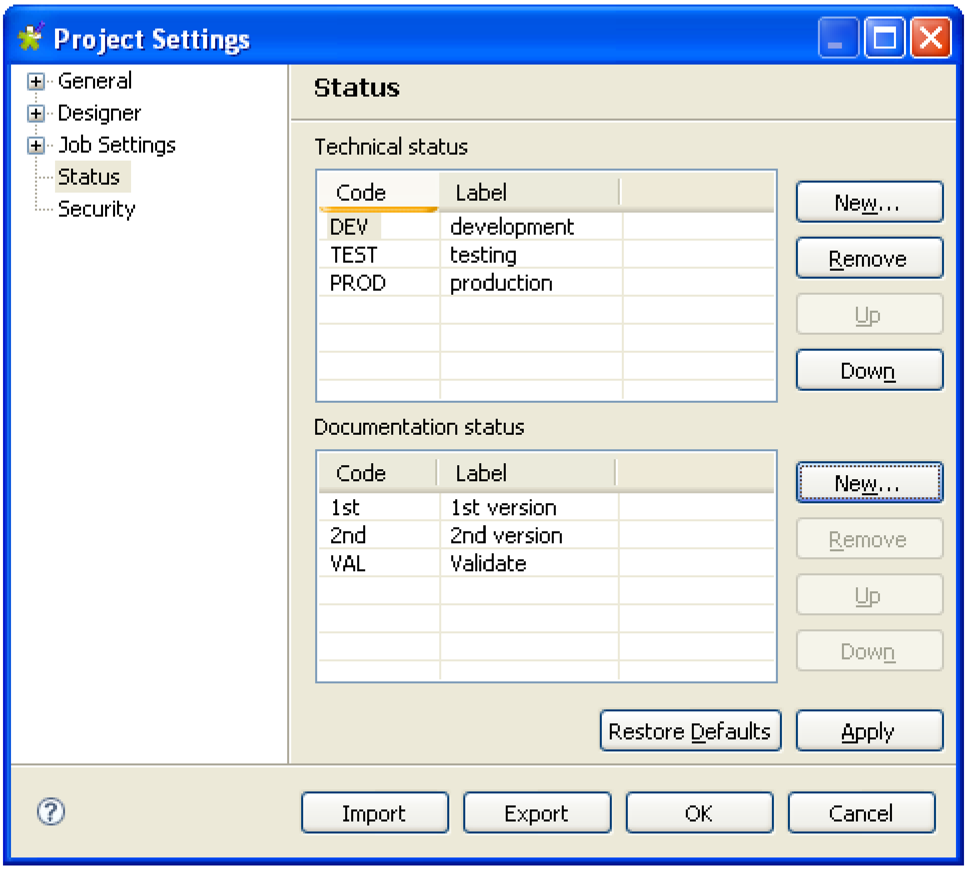
In the **[Project Settings]** dialog box, you can also define the Status.

To do so:

1. On the toolbar of the Studio main window, click or click **File** > **Edit Project Properties** from the menu bar to open the **[Project Settings]** dialog box.

2. In the tree view of the dialog box, click the **Status** node to define the main properties of your **Repository** tree view elements.

   The main properties of a repository item gathers information data such as **Name**, **Purpose**, **Description**, **Author**, **Version** and **Status** of the selected item. Most properties are free text fields, but the **Status** field is a drop-down list.



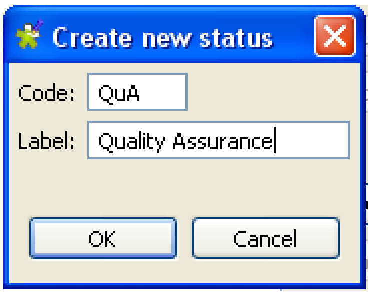
Technical status

| Code | Label |
|---|---|
| DEV | development |
| TEST | testing |
| PROD | production |

Documentation status

| Code | Label |
|---|---|
| 1st | 1st version |
| 2nd | 2nd version |
| VAL | Validate |

3. 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.



Code: QuA

Label: Quality Assurance

**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.

4. 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.

5. In the [**Project Settings]** dialog box, click **Apply** to validate your changes and then **OK** to close the dialog box.

### 2.6.8. Security settings

You can hide or show your passwords on your contexts, and so on when they are stored in the **Repository** tree view.

To hide your password:

1. On the toolbar of the Studio main window, click  or click **File** > **Edit Project Properties** from the menu bar to open the **[Project Settings]** dialog box.
2. In the tree view of the dialog box, click the **Security** node to open the corresponding view.
3. Select the **Hide passwords** check box to hide your password.
4. In the **[Project Settings]** dialog box, click **Apply** to validate your changes and then **OK** to close the dialog box.

## 2.7. Filtering entries listed in the Repository tree view

*Talend Open Studio for Big Data* provides the possibility to choose what nodes, Jobs or items you want to list in the **Repository** tree view.

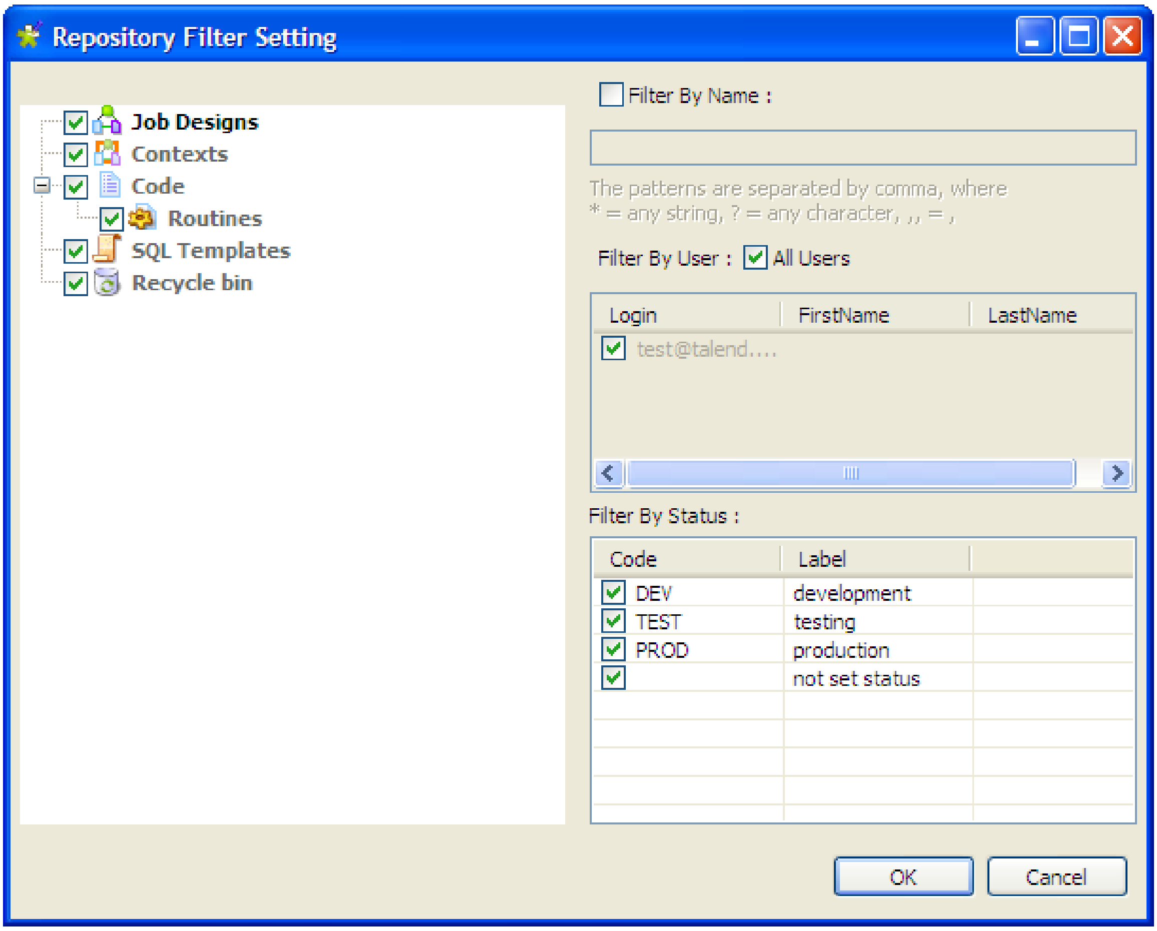
You can filter the **Repository** tree view by job name, Job status, the user who created the Job/items or simply by selecting/clearing the check box next to the node/ item you want to display/hide in the view. You can also set several filters simultaneously.

### 2.7.1. How to filter by Job name

To filter Jobs listed in the **Repository** tree view by Job name, complete the following:

1. In the Studio, click the  icon in the upper right corner of the **Repository** tree view and select **Filter settings** from the contextual menu.

   The **[Repository Filter]** dialog box displays.



☑ Job Designs
☑ Contexts
☑ Code
  ☑ Routines
☑ SQL Templates
☑ Recycle bin

□ Filter By Name :

The patterns are separated by comma, where
* = any string, ? = any character, ,, = ,

Filter By User : ☑ All Users

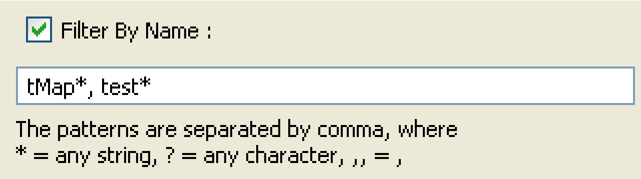
| Login | FirstName | LastName |
|---|---|---|
| ☑ test@talend.... | | |

Filter By Status :

| Code | Label |
|---|---|
| ☑ DEV | development |
| ☑ TEST | testing |
| ☑ PROD | production |
| ☑ | not set status |

2. Select the **Filter By Name** check box.

   The corresponding field becomes available.



☑ Filter By Name :

tMap*, test*

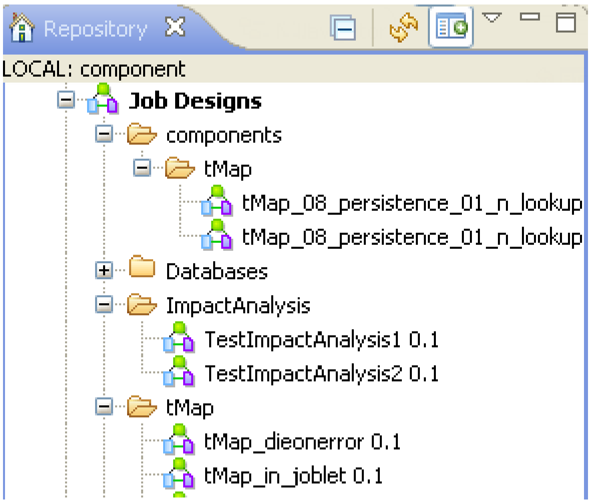
The patterns are separated by comma, where
* = any string, ? = any character, ,, = ,

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



Repository

LOCAL: component

- Job Designs
  - components
    - tMap
      - tMap_08_persistence_01_n_lookup
      - tMap_08_persistence_01_n_lookup
  - Databases
  - ImpactAnalysis
    - TestImpactAnalysis1 0.1
    - TestImpactAnalysis2 0.1
  - tMap
    - tMap_dieonerror 0.1
    - tMap_in_joblet 0.1

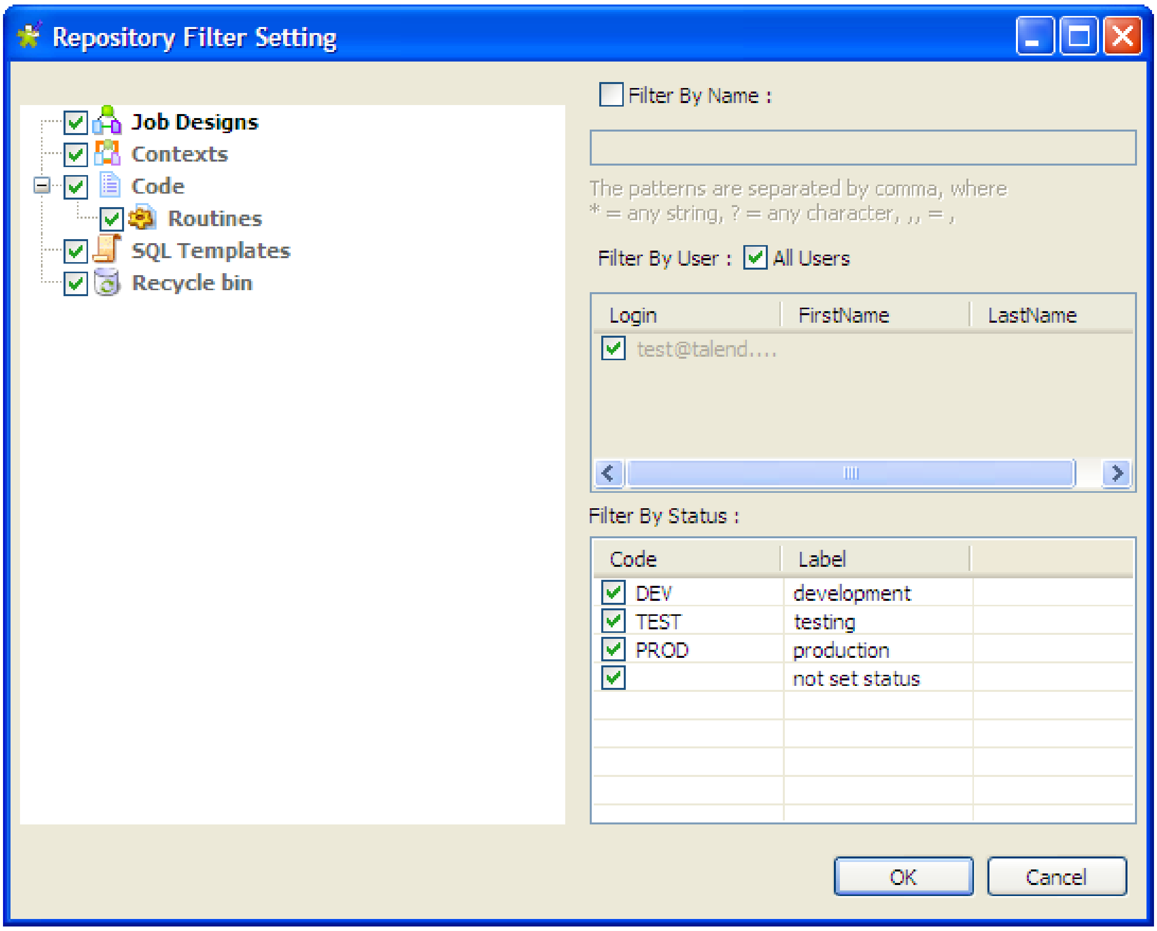
You can switch back to the by-default tree view, which lists all nodes, Jobs and items, by simply clicking the icon . This will cause the green plus sign appended on the icon to turn to a minus red sign ().

## 2.7.2. How to filter by user

To filter entries in the **Repository** tree view by the user who created the Jobs/items, complete the following:

1. In the Studio, click the icon in the upper right corner of the **Repository** tree view and select **Filter settings** from the contextual menu.

   The **[Repository Filter]** dialog box displays.



☑ Job Designs
☑ Contexts
☑ Code
☑ Routines
☑ SQL Templates
☑ Recycle bin

□ Filter By Name :

The patterns are separated by comma, where
* = any string, ? = any character, ,, = ,

Filter By User : ☑ All Users

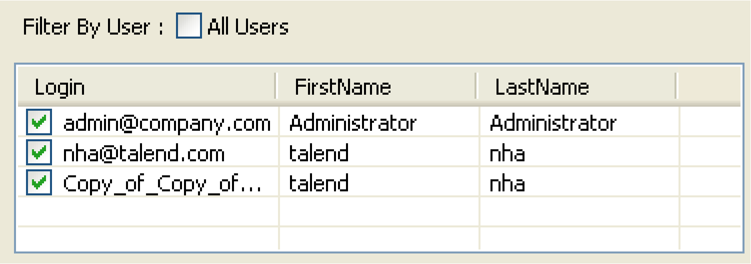
| Login | FirstName | LastName |
|---|---|---|
| ☑ test@talend.... | | |

Filter By Status :

| Code | Label |
|---|---|
| ☑ DEV | development |
| ☑ TEST | testing |
| ☑ PROD | production |
| ☑ | not set status |

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 |
|---|---|---|
| ☑ admin@company.com | Administrator | Administrator |
| ☑ nha@talend.com | talend | nha |
| ☑ Copy_of_Copy_of... | talend | nha |

   This table lists the authentication information of all the users who have logged in to *Talend Open Studio for Big Data* 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.

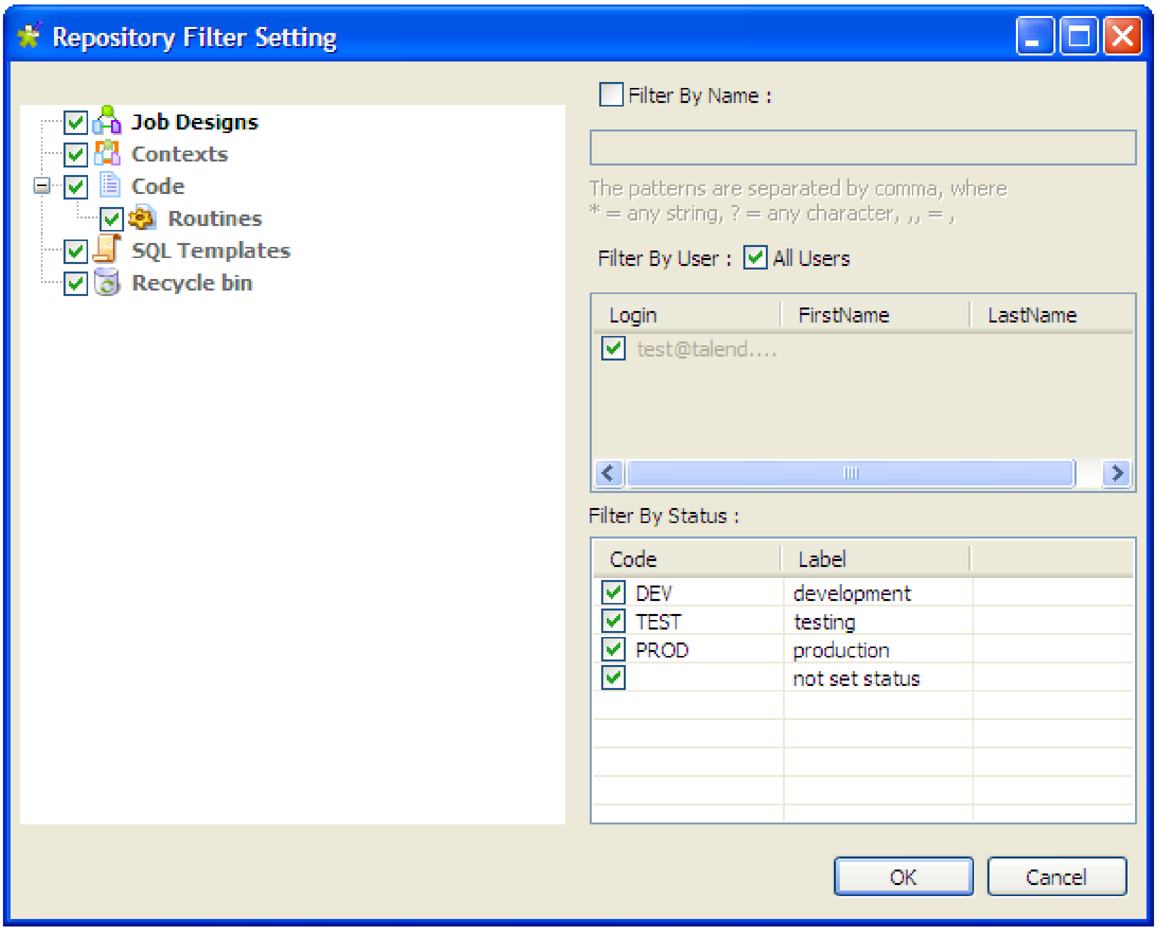
You can switch back to the by-default tree view, which lists all nodes, Jobs and items, by simply clicking the icon . This will cause the green plus sign appended on the icon to turn to a minus red sign ().

## 2.7.3. How to filter by job status

To filter Jobs in the **Repository** tree view by the job status, complete the following:

1. In the Studio, click the icon in the upper right corner of the **Repository** tree view and select **Filter settings** from the contextual menu.

   The **[Repository Filter]** dialog box displays.



Repository Filter Setting

- ☑ Job Designs
- ☑ Contexts
- ☑ Code
  - ☑ Routines
- ☑ SQL Templates
- ☑ Recycle bin

□ Filter By Name :

The patterns are separated by comma, where
* = any string, ? = any character, ,, = ,

Filter By User : ☑ All Users

| | Login | FirstName | LastName |
|---|---|---|---|
| ☑ | test@talend.... | | |

Filter By Status :

| | Code | Label |
|---|---|---|
| ☑ | DEV | development |
| ☑ | TEST | testing |
| ☑ | PROD | production |
| ☑ | | not set status |

2. 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.

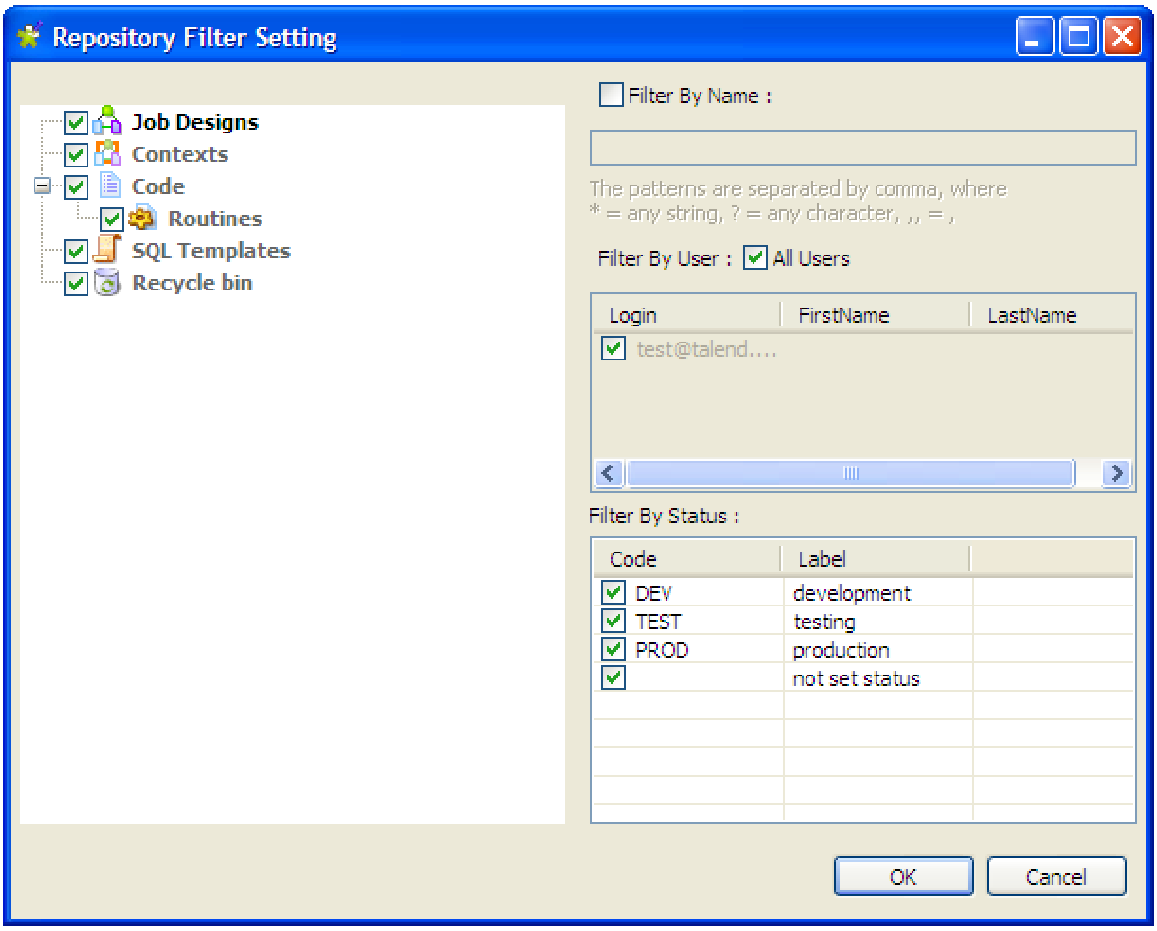
You can switch back to the by-default tree view, which lists all nodes, Jobs and items, by simply clicking the icon . This will cause the green plus sign appended on the icon to turn to a minus red sign ( ).

## 2.7.4. How to choose what repository nodes to display

To filter repository nodes, complete the following:

1. In the Studio, click the icon in the upper right corner of the **Repository** tree view and select **Filter settings** from the contextual menu.

The **[Repository Filter]** dialog box displays.



Repository Filter Setting

☑ Job Designs
☑ Contexts
☑ Code
 ☑ Routines
☑ SQL Templates
☑ Recycle bin

□ Filter By Name :

The patterns are separated by comma, where
* = any string, ? = any character, ,, = ,

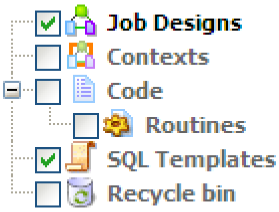
Filter By User : ☑ All Users

| Login | FirstName | LastName |
|---|---|---|
| ☑ test@talend.... | | |

Filter By Status :

| Code | Label |
|---|---|
| ☑ DEV | development |
| ☑ TEST | testing |
| ☑ PROD | production |
| ☑ | not set status |

2. Select the check boxes next to the nodes you want to display in the **Repository** tree view.

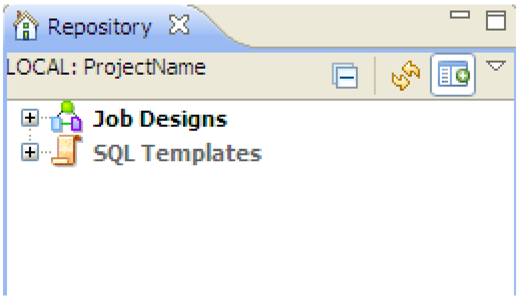


☑ Job Designs
□ Contexts
□ Code
 □ Routines
☑ SQL Templates
□ Recycle bin

Consider, for example, that you want to show in the tree view all the Jobs listed under the **Job Designs** node, and the **SQL Templates** node.

3. Click **OK** to validate your changes and close the dialog box.

Only the nodes/folders for which you selected the corresponding check boxes are displayed in the tree view.



If you do not want to show all the Jobs listed under the **Job Designs** node, you can filter the Jobs using the **Filter By Name** check box. For more information on filtering Jobs, see section *How to filter by Job name*.

# Chapter 3. Designing a data integration Job

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

This chapter helps you to design data integration Jobs that allow you to put in place up and run dataflow management processes.

Before starting any data integration processes, you need to be familiar with the *Talend Open Studio for Big Data* Graphical User Interface (GUI). For more information, see appendix *GUI*.

# 3.1. What is a Job design

A Job Design 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 Open Studio for Big Data*, you can:

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

⚠ *In order to be able to execute the Jobs you design in Talend Open Studio for Big Data, you need to install an Oracle JVM 1.6 or later (IBM JVM is not supported). You can download it from http://www.oracle.com/technetwork/java/javase/downloads/index.html.*

# 3.2. Getting started with a basic Job design

⚠ *Until a Job is created, the design workspace is unavailable and the Palette does not display.*

A Job design is made of one or several subjobs, which are themselves defined by one or, most likely, several components linked together. The properties of each component require to be configured individually, in order to function properly.

For more information, see section *How to connect components together* and section *How to define component properties*.

## 3.2.1. How to create a Job

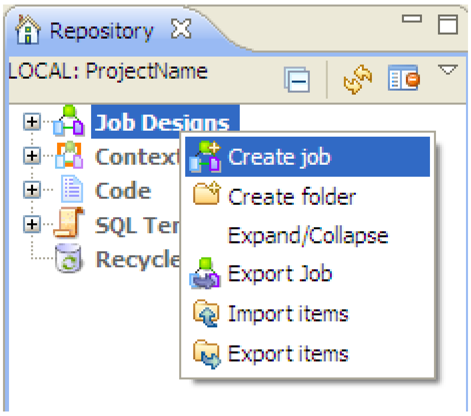
*Talend Open Studio for Big Data* enables you to create a Job Design by dropping different technical components from the **Palette** onto the design workspace and then connecting these components together.

You can also create different folders to better classify these Jobs.

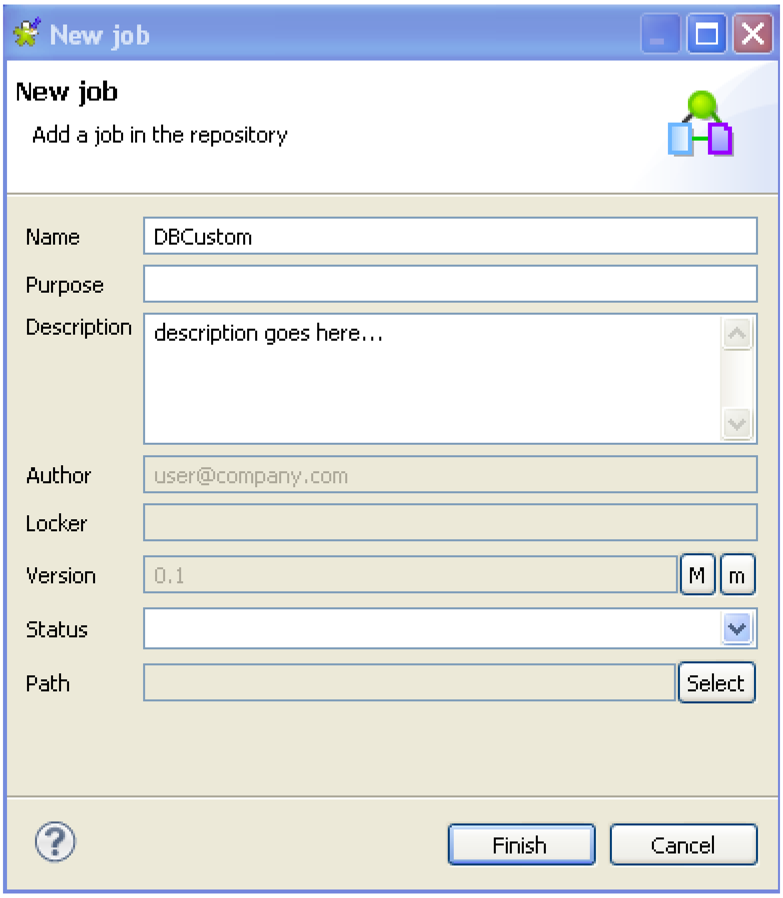
To create a Job, complete the following:

1. Open *Talend Open Studio for Big Data* following the procedure detailed in section *Launching Talend Open Studio for Big Data*.

2. In the **Repository** tree view, 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.



Name: DBCustom

Purpose:

Description: description goes here...

Author: user@company.com

Locker:

Version: 0.1

3. Enter the Job properties according to the following table:

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

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

4. Drop the components you want to use in your Job design from the **Palette** onto the design workspace and connect them together. For more information, see section *How to drop components to the workspace* and section *How to connect components together*.

5. Define the properties of each of the components used in the Job. For more information, see section *How to define component properties*.

6. Save your Job and then press **F6** to execute it. For more information, see section *How to run a Job*.

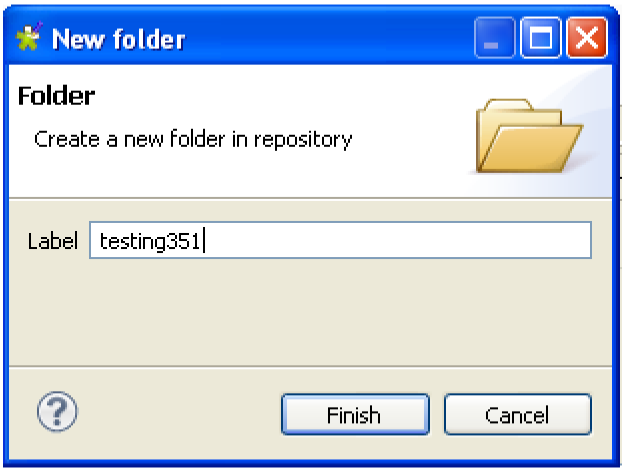
   The Job you created is now listed under the **Job Designs** node in the **Repository** tree view.

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

To create different folders for your Jobs, complete the following:

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

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



New folder

Folder

Create a new folder in repository

Label testing351

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

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

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

For a scenario showing how to create a real-life data integration Job, see appendix *Theory into practice: Job examples*.

## 3.2.2. How to drop components to the workspace

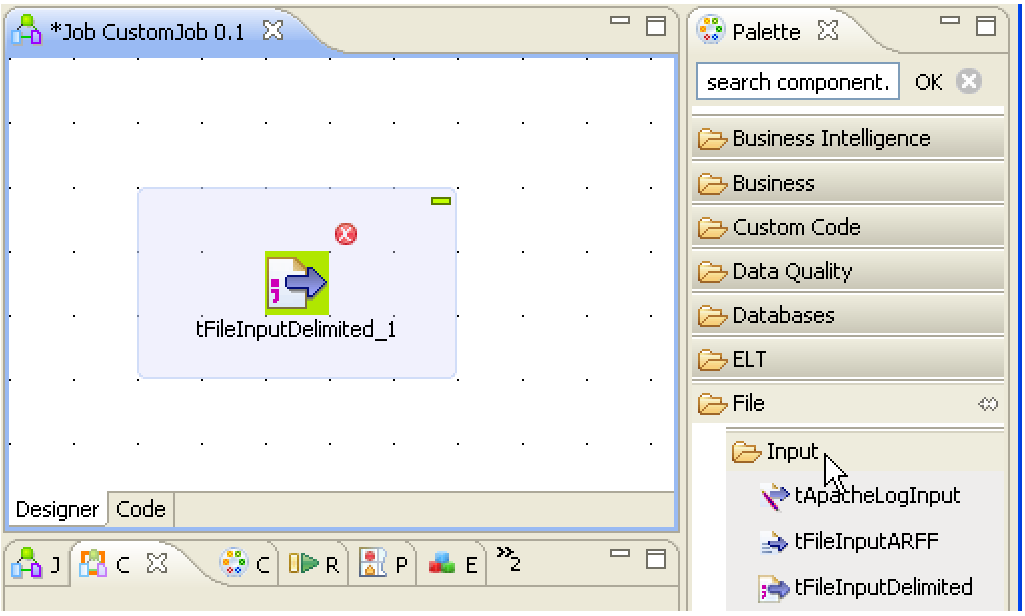
### 3.2.2.1. How to drop components from the Palette

To actually start building a Job, click a component on the **Palette**. Then click again on the design workspace to drop it there and add it to your Job Design.

If the **Palette** does not show in the Studio, see section *How to show, hide the Palette and change its position*.

You can drop a note to your Job the same way you drop components. For more information, see section *How to add notes to a Job design*.

Each newly added component displays generally in a blue square that symbolizes it as an individual Subjob.



Connect components together in a logical order using the connections offered, in order to build a full Job or subjob. For more information about component connection types, see section *Connection types*.

The Job or subjob gets highlighted in one single blue rectangle. For more information about Job and subjob background color, see section *How to manage the subjob display*.

Multiple information or warnings may show next to the component. Browse over the component icon to display the information tooltip. This will display until you fully completed your Job design and defined all basic (and sometimes advanced) component properties of the **Component** view.

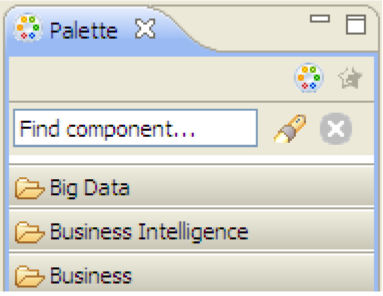
*You will be required to use Java code for your project.*

Related topics:

- section *How to connect components together*.
- section *Warnings and error icons on components*.
- section *How to define component properties*.

## 3.2.3. How to search components in the Palette

If you do not want to browse the components families in the **Palette** to find the components you want to use in your Job, you can search the desired component directly in the search field at the top of the **Palette**.

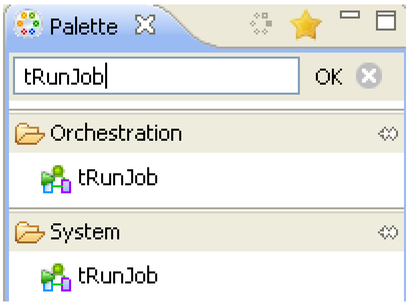


To search for a component, do the following:

1. Click ⊗ to clear the search field of any text.

2. Enter the name of the component you want to look for and click **OK**.

   The **Palette** displays only the family/families that hold(s) the component.



   To go back to the default **Palette** settings, click ⊗.

## 3.2.4. How to connect components together

A Job or a subjob is defined by a group of components interconnected in a logical way. The Job itself can be built with several subjobs carrying out various processings.

The component forming a subjob, as well as the subjobs are connected to each other using various types of connections.

Also, a Job (made of one or more subjobs) can be preceded by a pre-job and followed by a post-job components, in order to ensure that some specific tasks (often not related to the actual data processing) are performed first or last in the process. For more information, see section *How to use the tPrejob and tPostjob components*.

To connect two components, right-click the source component on your design workspace, select your type of connection from the contextual menu, and click the target component.

When dragging the link from your source component towards the target component, a graphical plug indicates if the destination component is valid or not. The black crossed circle disappears only when you reach a valid target component.

Only the connections authorized for the selected component are listed on the right-click contextual menu.

The types of connections proposed are different for each component according to its nature and role within the Job, i.e. if the connection is meant to transfer data (from a defined schema) or if no data is handled.

The types of connections available also depend if data comes from one or multiple input files and gets transferred towards one or multiple outputs.

For more information about the various types of connections and their specific settings, see section *Using connections*.

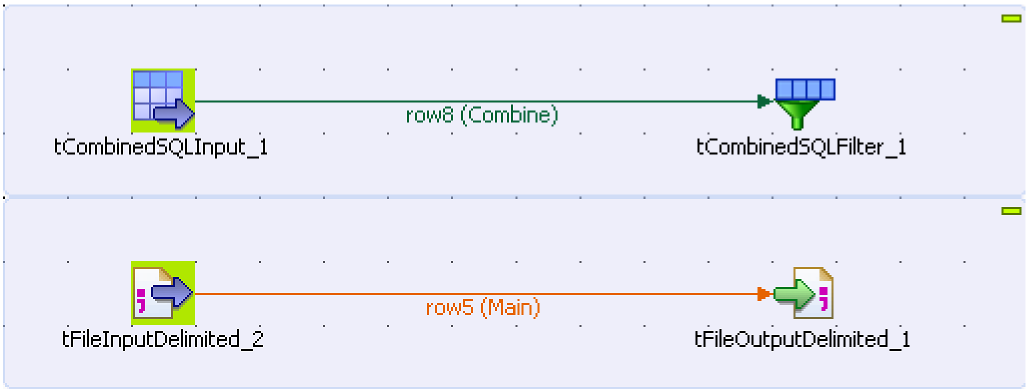
## 3.2.5. How to drop components in the middle of a Row link

When creating a Job, *Talend Open Studio for Big Data* enables you to insert a component in the middle of a **Row** > **Main**, **Row** > **Reject** or **Row** > **Combine** connection.

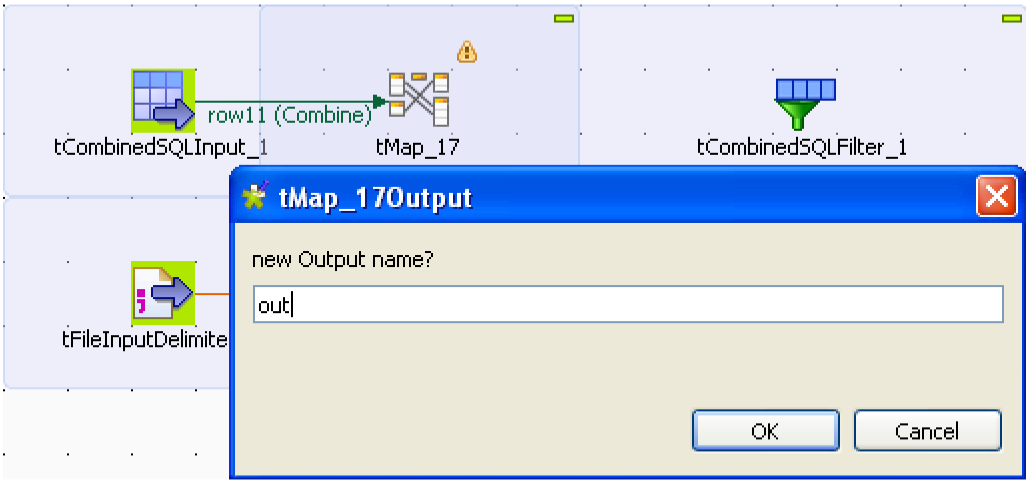
To do so, complete the following:

1. Drop two combine and two file components from the **Palette** onto the design workspace.

2. Connect the component pairs using a **Row** > **Main** (or a **Row** > **Reject**) connection and a **Row** > **Combine** one.



3. Drop the component you want to insert in the middle of the row. The link gets bold and then a dialog box displays, prompting you to type in a name for the output link.



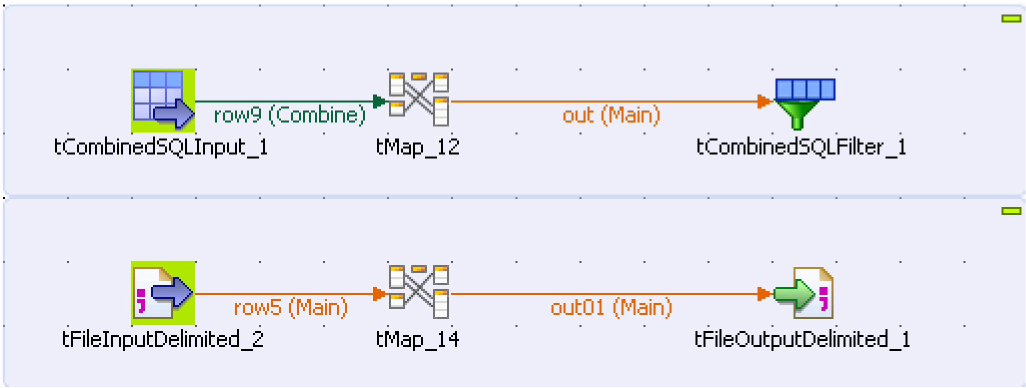
new Output name?

out

4. Type in a name and click **OK** to close the dialog box.

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

The component is inserted in the middle of the link, which is now divided in two links.



# 3.2.6. How to define component properties

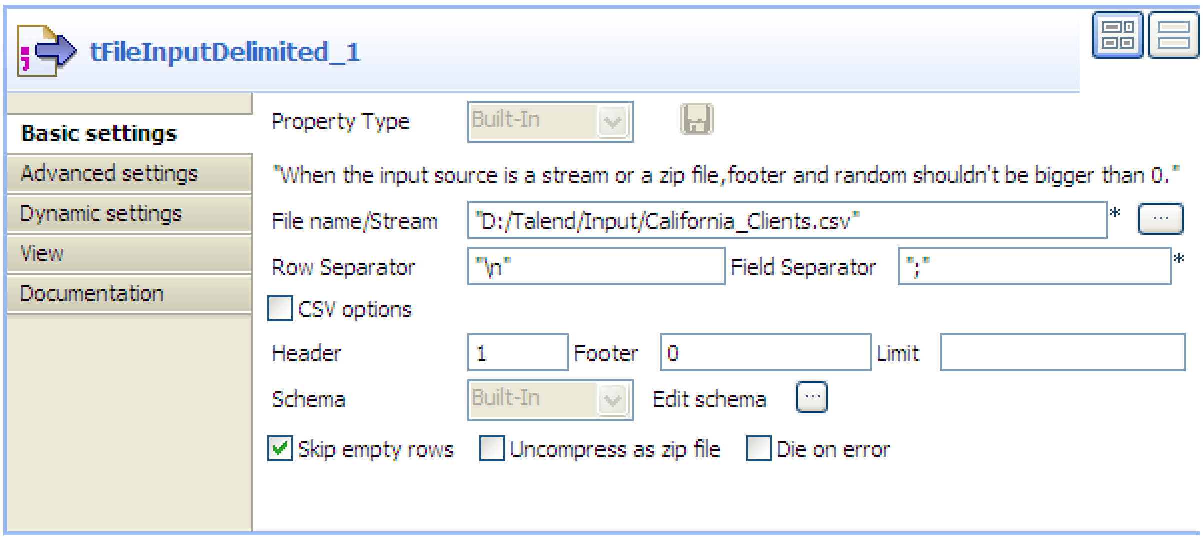
The properties information for each component forming a Job or a subjob allows to set the actual technical implementation of the active Job.

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

For detailed configuration for each component displaying in the **Palette**, check *Talend Open Studio for Big Data Components Reference Guide*.

## 3.2.6.1. Basic Settings tab

The **Basic Settings** tab is part of the **Component** view, which is located on the lower part of the designing editor of *Talend Open Studio for Big Data*.



Property Type Built-In

"When the input source is a stream or a zip file,footer and random shouldn't be bigger than 0."

File name/Stream "D:/Talend/Input/California_Clients.csv" *

Row Separator "\n" Field Separator ";" *

□ CSV options

Header 1 Footer 0 Limit

Schema Built-In Edit schema

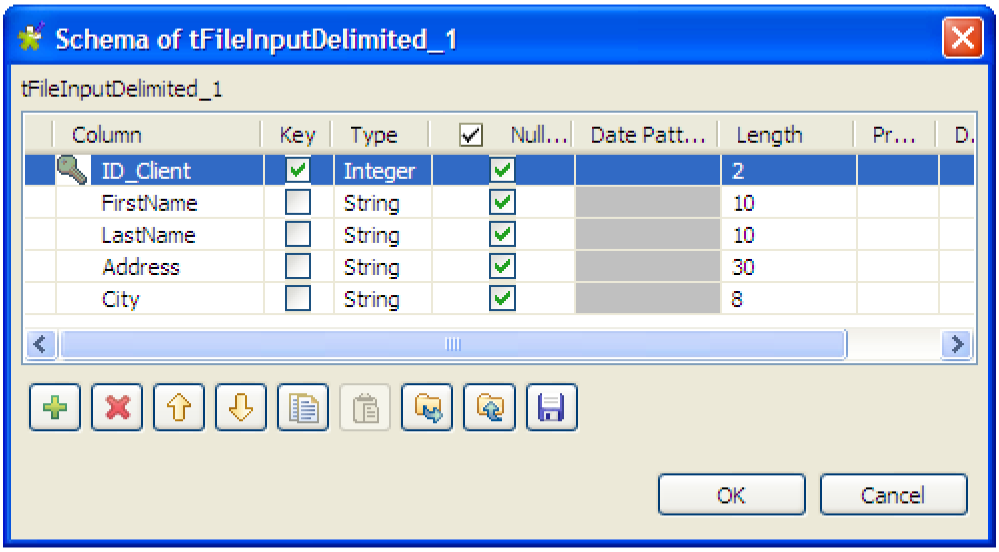
☑ Skip empty rows □ Uncompress as zip file □ Die on error

Each component has specific basic settings according to its function requirements within the Job. For a detailed description of each component properties and use, see *Talend Open Studio for Big Data Components Reference Guide*.

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

### How to set a schema

Click the **Edit Schema** button to create your built-in schema by adding columns and describing their content, according to the input file definition.



tFileInputDelimited_1

| Column | Key | Type | ☑ Null... | Date Patt... | Length | Pr... | D. |
|---|---|---|---|---|---|---|---|
| ID_Client | ☑ | Integer | ☑ | | 2 | | |
| FirstName | □ | String | ☑ | | 10 | | |
| LastName | □ | String | ☑ | | 10 | | |
| Address | □ | String | ☑ | | 30 | | |
| City | □ | String | ☑ | | 8 | | |

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.

Some extra information is required. For more information about Date pattern for example, check out: http://docs.oracle.com/javase/6/docs/api/index.html.

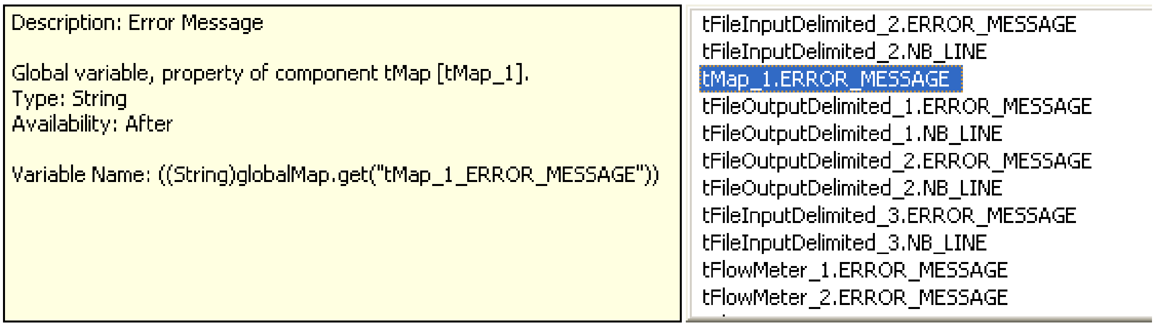
### How to set a field dynamically (Ctrl+Space bar)

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

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

2. Press **Ctrl+Space bar** to access the proposal list.

3. Select on the list the relevant parameters you need. Appended to the variable list, a information panel provides details about the selected parameter.



Description: Error Message

Global variable, property of component tMap [tMap_1].
Type: String
Availability: After

Variable Name: ((String)globalMap.get("tMap_1_ERROR_MESSAGE"))

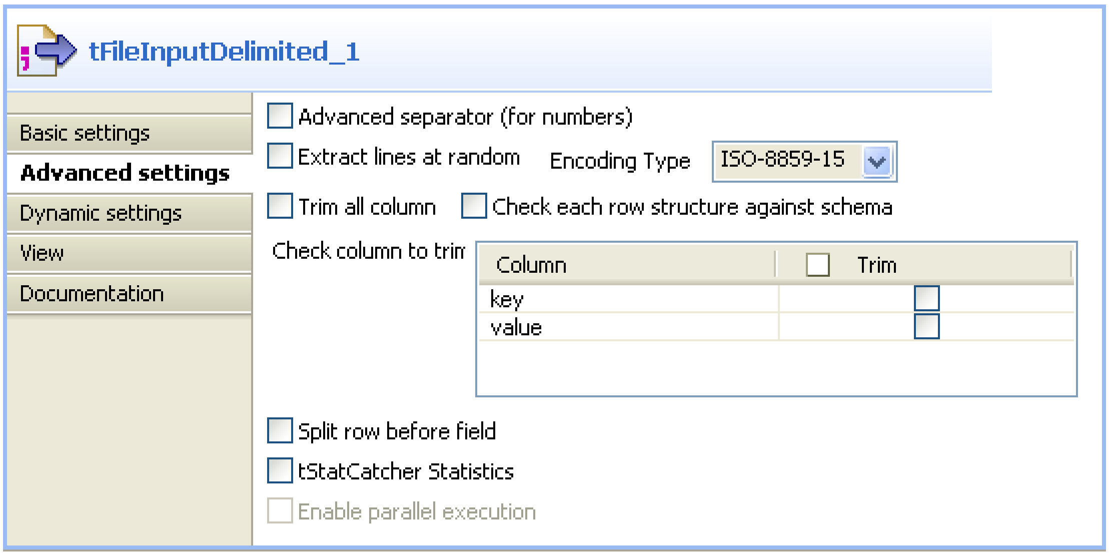
tFileInputDelimited_2.ERROR_MESSAGE
tFileInputDelimited_2.NB_LINE
tMap_1.ERROR_MESSAGE
tFileOutputDelimited_1.ERROR_MESSAGE
tFileOutputDelimited_1.NB_LINE
tFileOutputDelimited_2.ERROR_MESSAGE
tFileOutputDelimited_2.NB_LINE
tFileInputDelimited_3.ERROR_MESSAGE
tFileInputDelimited_3.NB_LINE
tFlowMeter_1.ERROR_MESSAGE
tFlowMeter_2.ERROR_MESSAGE

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: section *How to centralize contexts and variables*.

## 3.2.6.2. Advanced settings tab

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



□ Advanced separator (for numbers)

□ Extract lines at random  Encoding Type ISO-8859-15

□ Trim all column  □ Check each row structure against schema

Check column to trim

| Column | □ Trim |
|---|---|
| key | □ |
| value | □ |

□ Split row before field

□ tStatCatcher Statistics

□ Enable parallel execution

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

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

### How to measure data flows

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

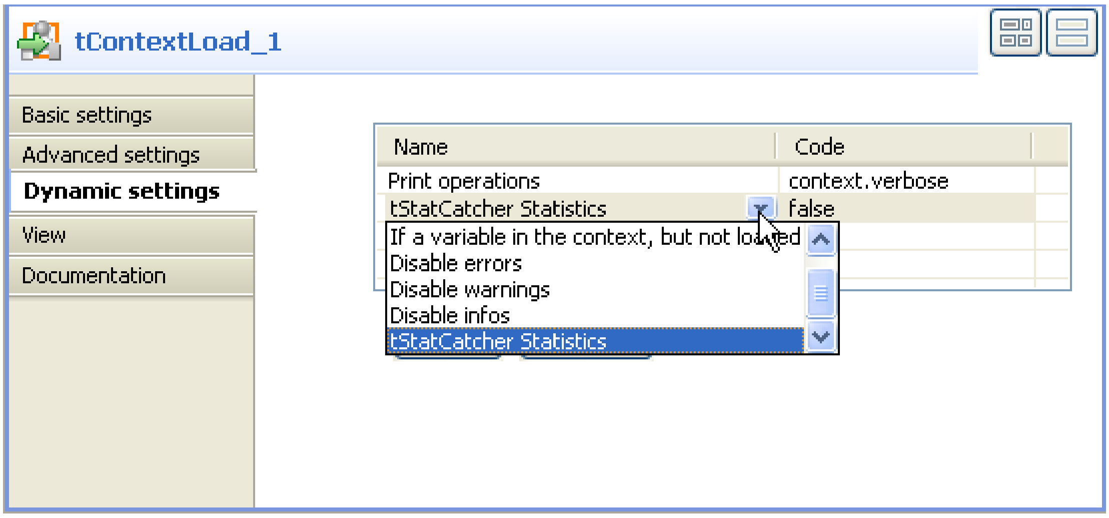
## 3.2.6.3. Dynamic settings tab

The **Basic settings** and **Advanced settings** tabs of all all components display various check boxes and drop-down lists for component parameters. Usually, available values for these types of parameters are either *true* or *false* and 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 |
| tStatCatcher Statistics | false |

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

1. Select the relevant component basic settings or advanced settings view that contains the parameter you want to define as a variable.

2. Click the **Dynamic settings** tab.

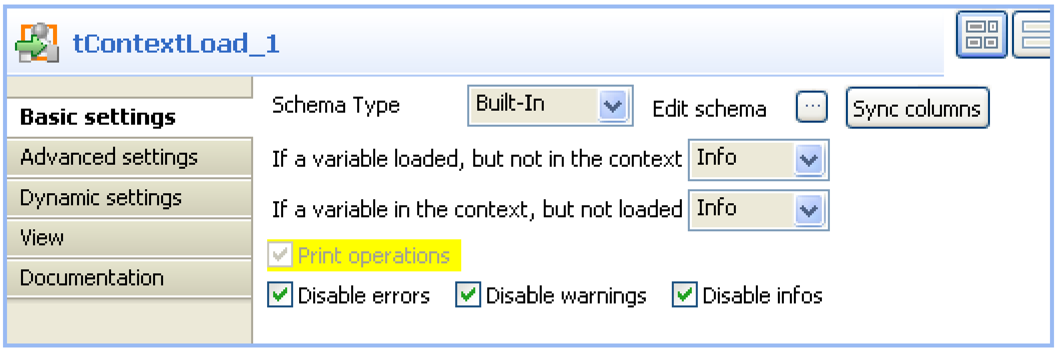
3. Click the **plus** button to display a new parameter line in the table.

4. Click the **Name** of the parameter displaying to show the list of available parameters. For example: *Print operations*

5. Then click in the facing **Code** column cell and set the code to be used. For example: *context.verbose* if you create the corresponding context variable, called *verbose*.

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

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



Schema Type: Built-In

If a variable loaded, but not in the context: Info

If a variable in the context, but not loaded: Info

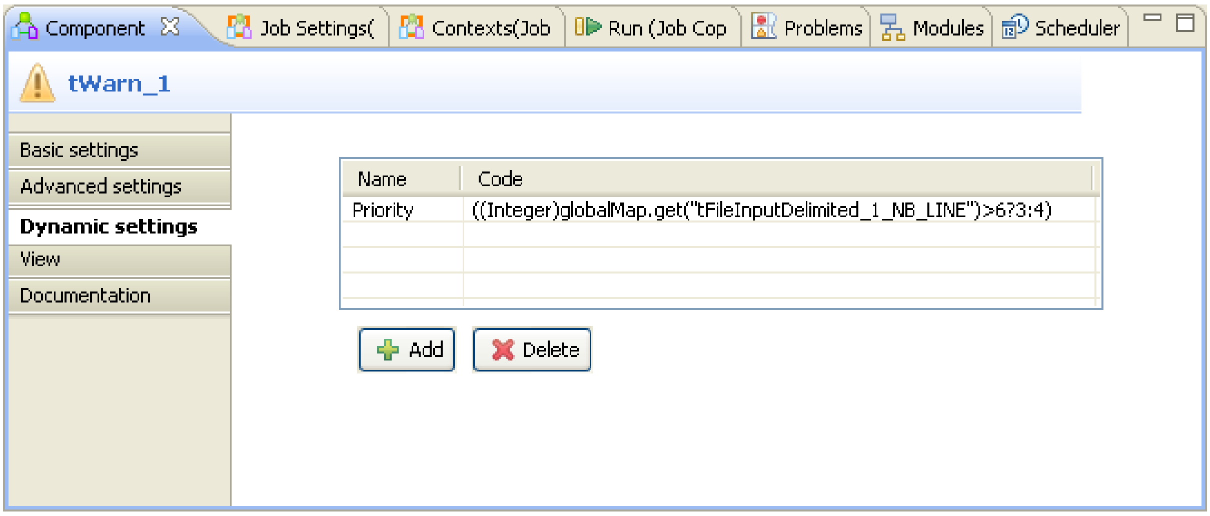
☑ Print operations

☑ Disable errors ☑ Disable warnings ☑ Disable infos

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

For more information regarding the context variable definition, see section *How to use variables in the Contexts view*.

You can also use a global variable or pieces of Java code to store the values to be used for each parameter.



| Name | Code |
|---|---|
| Priority | ((Integer)globalMap.get("tFileInputDelimited_1_NB_LINE")>6?3:4) |

For example, use some global variable available through the **Ctrl+Space** bar keys, and adapt it to your context.

## 3.2.6.4. View tab

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

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

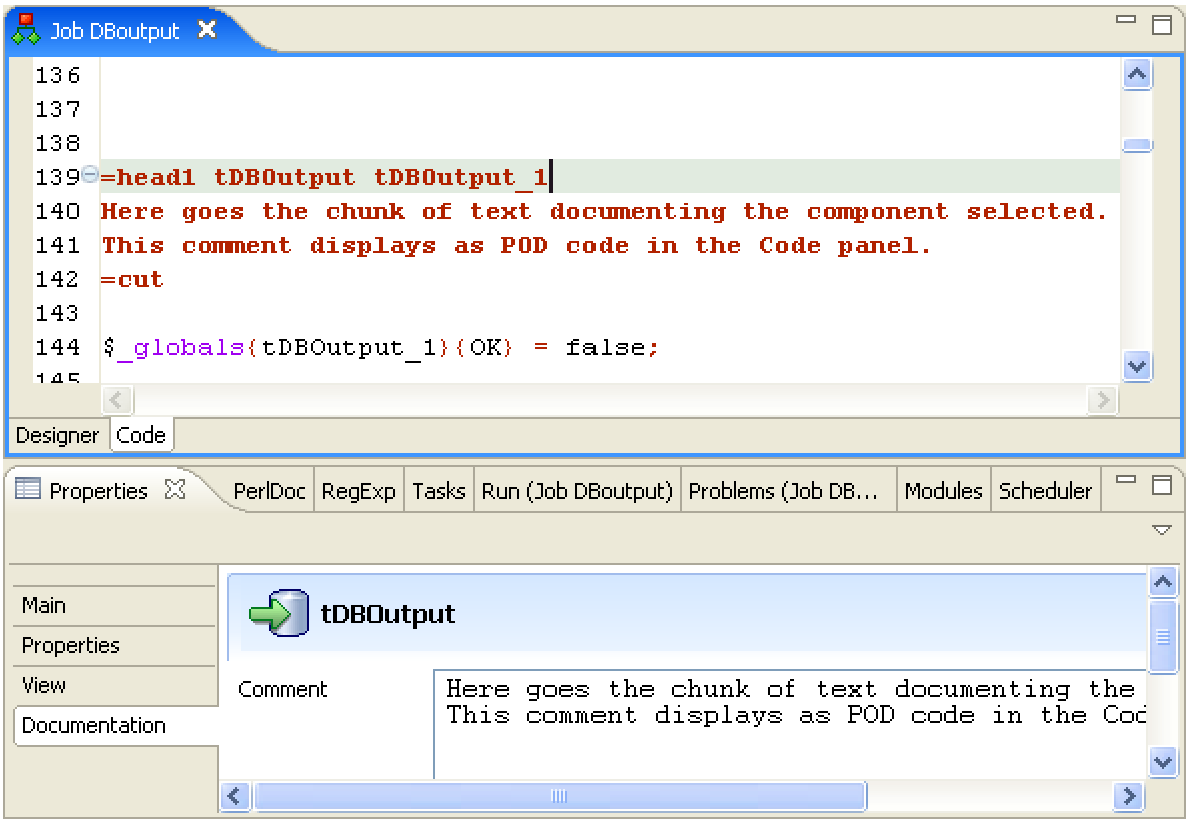
You can graphically highlight both **Label** and **Hint** text with HTML formatting tags:

- Bold: <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**.

## 3.2.6.5. Documentation tab

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



```


=head1 tDBOutput tDBOutput_1
Here goes the chunk of text documenting the component selected.
This comment displays as POD code in the Code panel.
=cut

$_globals{tDBOutput_1}{OK} = false;
```

Comment: Here goes the chunk of text documenting the
This comment displays as POD code in the Cod

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.

## 3.2.7. How to run a Job

You can execute a Job in several ways. This mainly depends on the purpose of your Job execution and on your user level.

If you are an advanced Java user and want to execute your Job step by step to check and possibly modify it on the run, see section *How to run a Job in Java Debug mode*.

If you do not have advanced Java knowledge and want to execute and monitor your Job in normal mode, see section *How to run a Job in normal mode*.

For how to run a Job on a remote Hadoop server via **Oozie scheduler**, see section *How to run a Job on a remote HDFS server*.

### 3.2.7.1. How to run a Job in normal mode

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

To run your Job in a normal mode, complete 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.

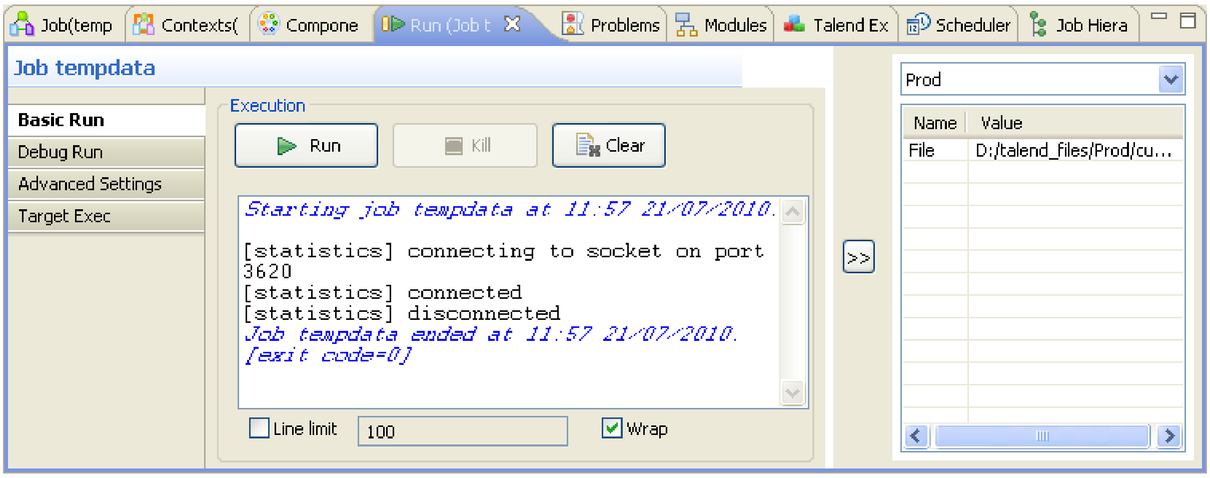
If you have not defined any particular execution context, the context parameter table is empty and the context is the default one. Related topic: section *How to centralize contexts and variables*.

1. Click **Run** to start the execution.

2. On the same view, the console displays the progress of the execution. The log includes any error message as well as start and end messages. It also shows the Job output in case of a **tLogRow** component is used in the Job design.

3. To define the lines of the execution progress to be displayed in the console, select the **Line limit** check box and type in a value in the field.

4. Select the **Wrap** check box to wrap the text to fit the console width. This check box is selected by default. When it is cleared, a horizontal scrollbar appears, allowing you to view the end of the lines.



```
Starting job tempdata at 11:57 21/07/2010.

[statistics] connecting to socket on port
3620
[statistics] connected
[statistics] disconnected
Job tempdata ended at 11:57 21/07/2010.
[exit code=0]
```

□ Line limit 100 ☑ Wrap

Prod

| Name | Value |
|---|---|
| File | D:/talend_files/Prod/cu... |

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 Open Studio for Big Data* 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.

### 3.2.7.2. How to run a Job in Java Debug mode

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

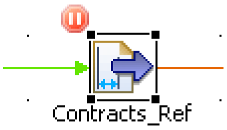
To access the Debug mode:

1. Click the **Run** view to access it.

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

⚠ *In order to be able to run a Job in Debug mode, you need the EPIC module to be installed.*

Before running your Job in Debug mode, add breakpoints to the major steps of your Job flow.



This will allow you to get the Job to automatically stop at each breakpoint. This way, components and their respective variables can be verified individually and debugged if required.

To add breakpoints to a component, right-click it on the design workspace, and select **Add breakpoint** on the contextual menu.

A pause icon displays next to the component where the break is added.

To switch to debug mode, click the **Java Debug** button on the **Debug Run** tab of the **Run** panel. *Talend Open Studio for Big Data*'s main window gets reorganized for debugging.

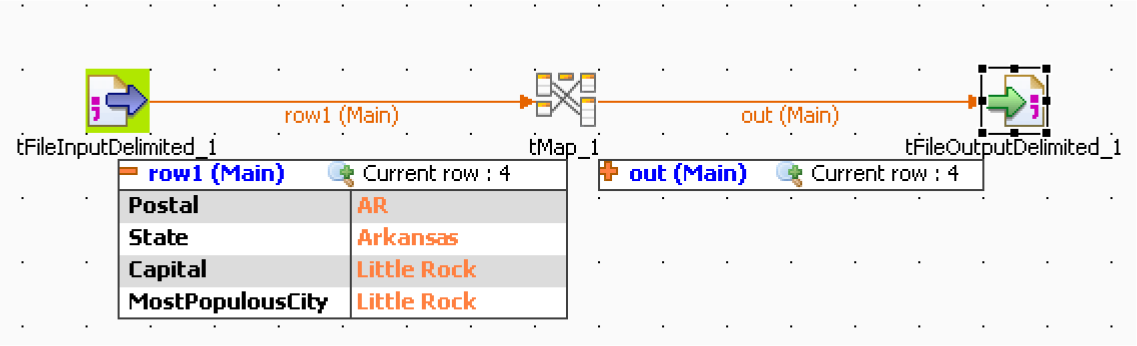
You can then run the Job step by step and check each breakpoint component for the expected behavior and variable values.

To switch back to *Talend Open Studio for Big Data* designer mode, click **Window**, then **Perspective** and select *Talend Open Studio for Big Data*.

## 3.2.7.3. How to run a Job in Traces Debug mode

The traces feature allows you to monitor data processing when running a Job in *Talend Open Studio for Big Data*.

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.



| row1 (Main) | Current row : 4 |
|---|---|
| Postal | AR |
| State | Arkansas |
| Capital | Little Rock |
| MostPopulousCity | Little Rock |

out (Main) Current row : 4

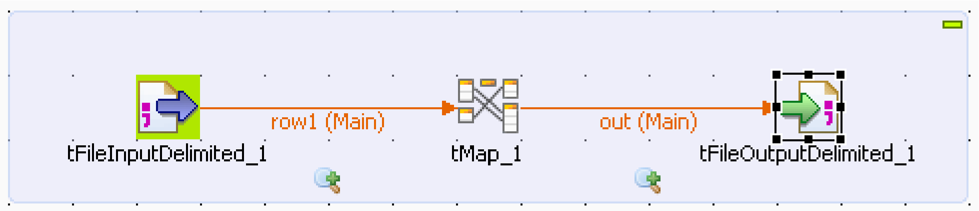
This feature allows you to monitor all the components of a Job, without switching to the debug mode, hence without requiring advanced Java knowledge.

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

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

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

To activate the **Traces** mode in a Job:



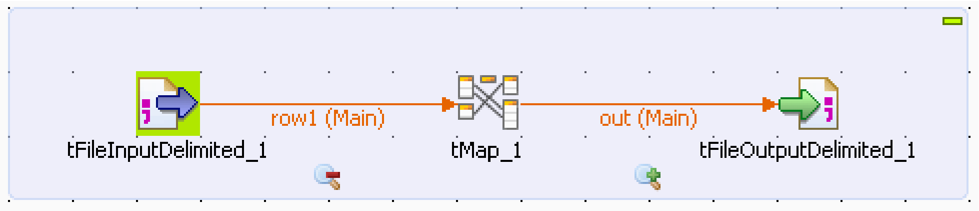
1. Click the **Run** view.

2. Click the **Debug Run** tab to access the debug and traces execution modes.

3. Click the down arrow of the **Java Debug** button and select the **Traces Debug** option. An icon displays under every flow of your Job to indicate that process monitoring is activated.

4. Click the **Traces Debug** to execute the Job in Traces mode.

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

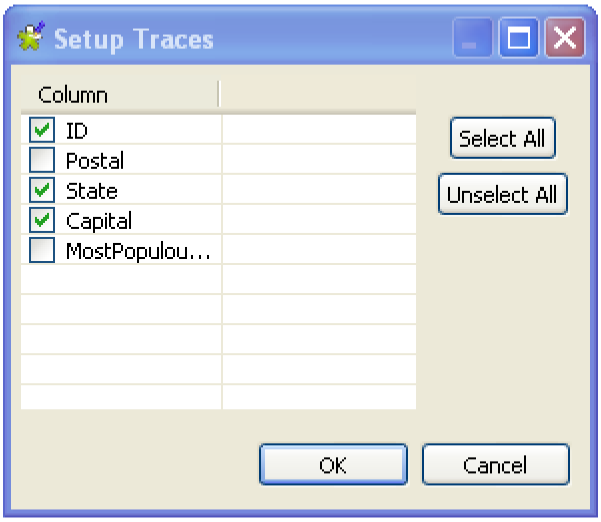


1. Right-click the **Traces** icon under the relevant flow.

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:

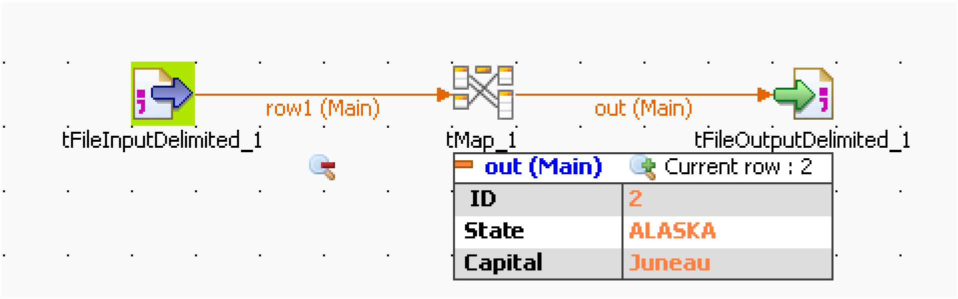
1. Right-click the **Traces** icon for the relevant flow, then select **Setup Traces** from the list. The **[Setup Traces]** dialog box appears.



| Column |
|---|
| ☑ ID |
| □ Postal |
| ☑ State |
| ☑ Capital |
| □ MostPopulou... |

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.



out (Main) Current row : 2

| | |
|---|---|
| ID | 2 |
| State | ALASKA |
| Capital | Juneau |

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.

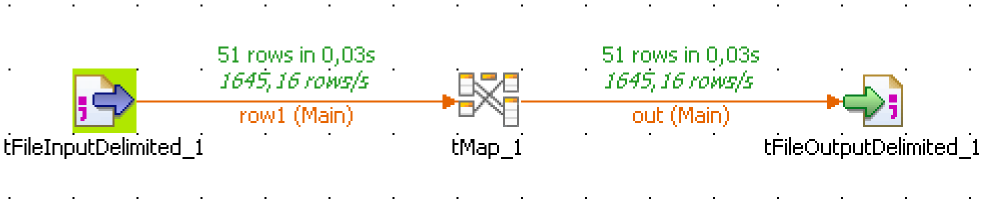
## 3.2.7.4. How to set advanced execution settings

Several advanced execution settings are available to make the execution of the Jobs handier:

- **Statistics**, this feature displays processing performance rate. For more information, see section *How to display Statistics*.
- **Exec time**, this feature displays the execution time in the console at the end of the execution. For more information, see section *How to display the execution time and other options*.
- **Save Job before execution**, this feature allows to automatically save the Job before its execution.
- **Clear before run**, this feature clears all the results of a previous execution before re-executing the Job.
- **JVM Setting**, this feature allows you to define the parameters of your JVM according to your needs, for example the parameters used to display special characters.

### How to display Statistics

The **Statistics** feature displays each component performance rate, under the flow links on the design workspace.



It shows the number of rows processed and the processing time in row per second, allowing you to spot straight away any bottleneck in the data processing flow.

For trigger links like **OnComponentOK**, **OnComponentError**, **OnSubjobOK**, **OnSubjobError** and **If**, the **Statistics** option displays the state of this trigger during the execution time of your Job: Ok or Error and True or False.

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

In the **Run** view, click the **Advanced settings** tab and select the **Statistics** check box to activate the Stats feature and clear the box to disable it.

The calculation only starts when the Job execution is launched, and stops at the end of it.

Click the **Clear** button from the **Basic** or **Debug Run** views to remove the calculated stats displayed. Select the **Clear before Run** check box to reset the Stats feature before each execution.

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

You can also save your Job before the execution starts. Select the relevant option check box.

## How to display the execution time and other options

To display the Job total execution time after Job execution, select in the **Advanced settings** tab of the **Run** view the **Exec time** check box before running the Job.

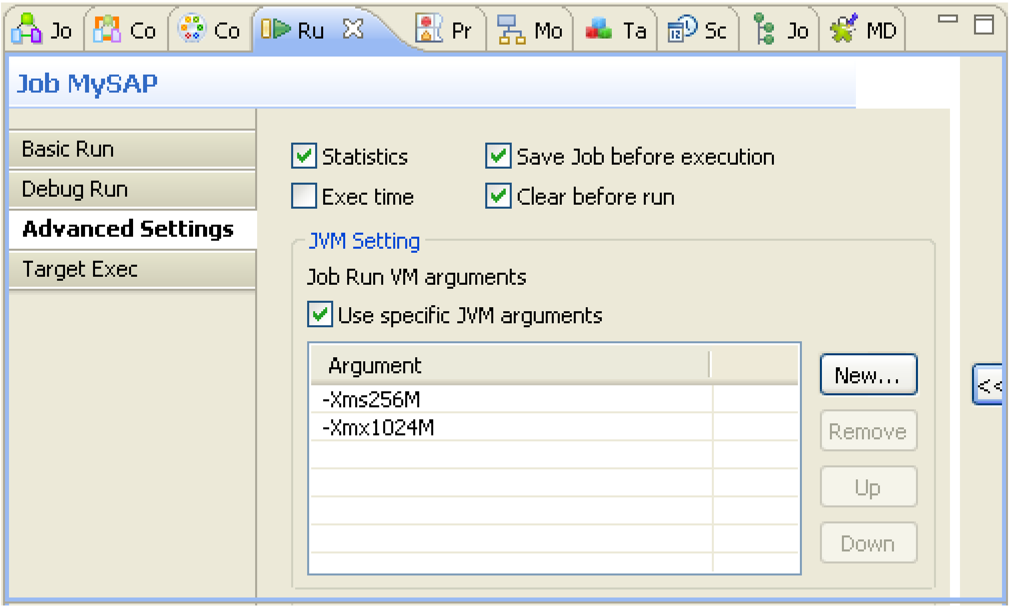
This way you can test your Job before going to production.

You can also clear the design workspace before each Job execution by selecting the check box **Clear before Run**.

You can also save your Job before the execution starts. Select the relevant option check box.

## How to display special characters in the console

*Talend Open Studio for Big Data* 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:



Job MySAP

☑ Statistics ☑ Save Job before execution

□ Exec time ☑ Clear before run

JVM Setting

Job Run VM arguments

☑ Use specific JVM arguments

| Argument |
|---|
| -Xms256M |
| -Xmx1024M |

1. Select the **Advanced settings** tab.

2. In the **JVM settings** area of the tab view, select the **Use specific JVM arguments** checkbox to activate the **Argument** table.

3. Next to the **Argument** table, click the **New...** button to pop up the **[Set the VM argument]** dialog box.

4. In the dialog box, type in `-Dfile.encoding=UTF-8`.

5. Click **OK** to close the dialog box.

   This argument can be applied for all of your Job executions in *Talend Open Studio for Big Data*. For further information about how to apply this JVM argument for all of the Job executions, see section *Debug and Job execution preferences (Talend > Run/Debug)*.

## 3.2.7.5. How to run a Job on a remote HDFS server

*Talend Open Studio for Big Data* provides an **Oozie scheduler**, a feature that enables you to schedule executions of a Job you have created or run it immediately on a remote Hadoop Distributed File System (HDFS) server, and to monitor the execution status of your Job. For more information on Apache Oozie and Hadoop, check http://oozie.apache.org/ and http://hadoop.apache.org/.

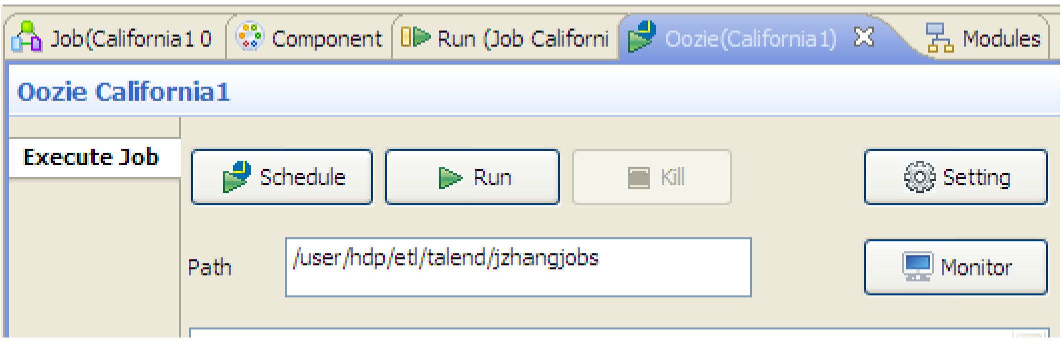
Before you can run or schedule executions of a Job on an HDFS server, you need first to define the HDFS connection details either in the **Oozie scheduler** view or in the *Talend Open Studio for Big Data* preference settings, and specify the path where your Job will be deployed.

### How to set HDFS connection details

#### Defining HDFS connection details in Oozie scheduler view

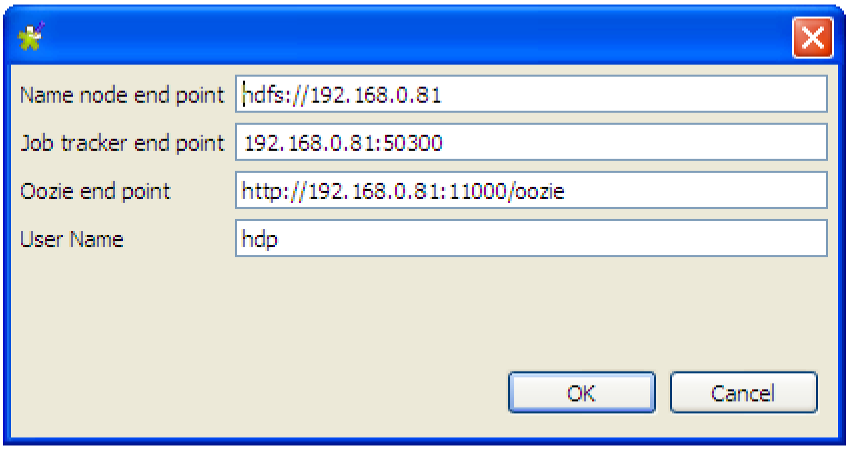
To define HDFS connection details in the **Oozie scheduler** view, do the following:

1. Click the **Oozie schedule** view beneath the design workspace.



Path: /user/hdp/etl/talend/jzhangjobs

2. Click **Setting** to open the connection setup dialog box.



Name node end point: hdfs://192.168.0.81

Job tracker end point: 192.168.0.81:50300

Oozie end point: http://192.168.0.81:11000/oozie

User Name: hdp

*The connection settings shown above are for an example only.*

3. Fill in the required information in the corresponding fields, and click **OK** close the dialog box.

| Field | Description |
|---|---|
| Name Node End Point | URI of the name node, the centerpiece of the HDFS file system. |
| Job Tracker End Point | URI of the Job Tracker node, which farms out MapReduce tasks to specific nodes in the cluster. |

| Field | Description |
|---|---|
| Oozie End Point | URI of the Oozie endpoint, for Job execution monitoring. |
| User Name | Login user name. |

Once the connection details are defined via the **Oozie scheduler** view, the Oozie preference settings are automatically updated, and vice versa. For information on Oozie preference settings, see section *Defining HDFS connection details in preference settings*.

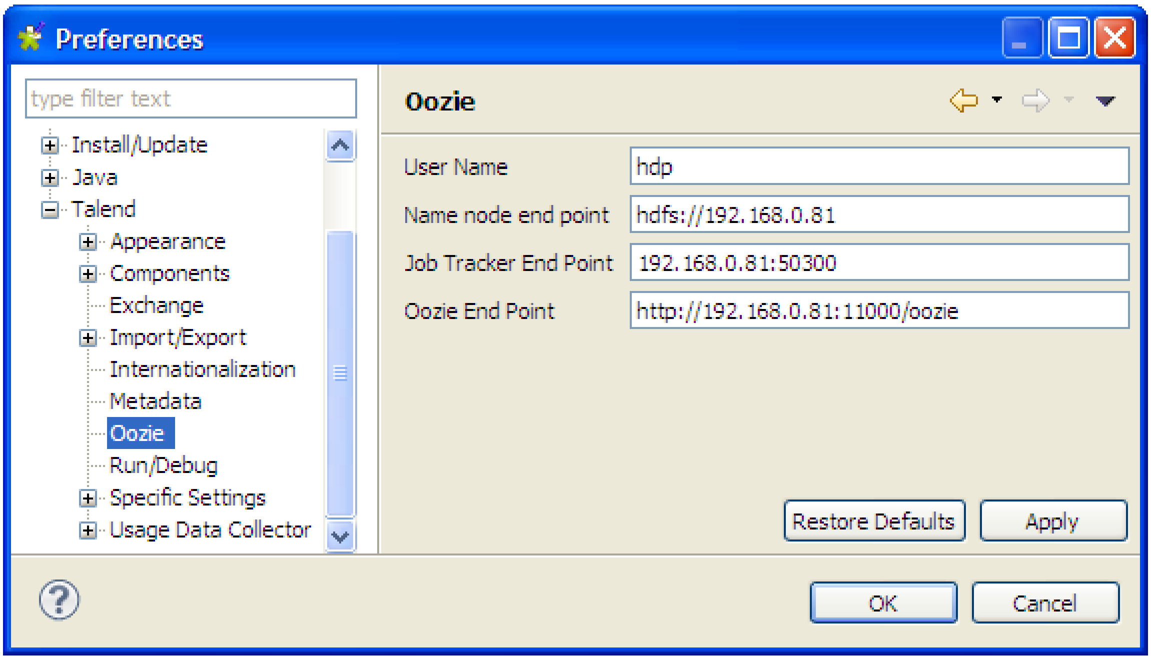
Once the connection details are defined in the **Oozie scheduler** view, the HDFS connection settings in the **[Preferences]** dialog box are automatically updated, and vice versa.

Upon defining the deployment path in the **Oozie scheduler** view, you are ready to schedule executions of your Job, or run it immediately, on the HDFS server.

### Defining HDFS connection details in preference settings

To define HDFS connection details in the *Talend Open Studio for Big Data* preference settings, do the following:

1. From the menu bar, click **Window** > **Preferences** to open the **[Preferences]** dialog box.
2. Expand the **Talend** node and click **Oozie** to display the Oozie preference view.



User Name: hdp

Name node end point: hdfs://192.168.0.81

Job Tracker End Point: 192.168.0.81:50300

Oozie End Point: http://192.168.0.81:11000/oozie

*The Oozie settings shown above are for an example only.*

3. Fill in the required information in the corresponding fields:

| Field | Description |
|---|---|
| User Name | Login user name. |
| Name Node End Point | URI of the name node, the centerpiece of the HDFS file system. |
| Job Tracker End Point | URI of the Job Tracker node, which farms out MapReduce tasks to specific nodes in the cluster. |
| Oozie End Point | URI of the Oozie endpoint, for Job execution monitoring. |

Once the connection details are defined in the**[Preferences]** dialog box, the HDFS connection settings in the **Oozie scheduler** view are automatically updated, and vice versa. For information on the **Oozie scheduler** view, see section *How to run a Job on a remote HDFS server*.

## How to run a Job on the HDFS server

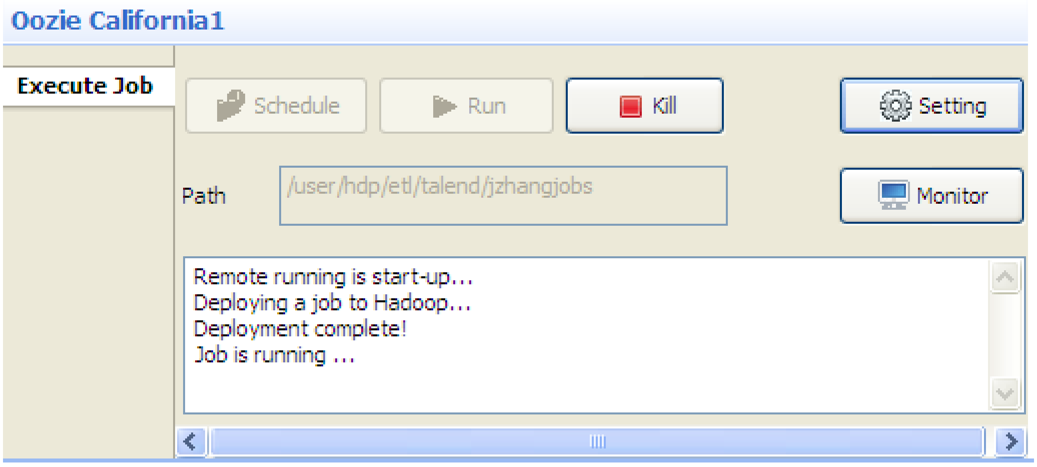
To run a Job on the HDFS server,

1. In the **Path** field on the **Oozie scheduler** tab, enter the path where your Job will be deployed on the HDFS server.

2. Click the **Run** button to start Job deployment and execution on the HDFS server.

Your Job data is zipped, sent to, and deployed on the HDFS server based on the server connection settings and automatically executed. Depending on your connectivity condition, this may take some time. The console displays the Job deployment and execution status.

To stop the Job execution before it is completed, click the **Kill** button.



Path: /user/hdp/etl/talend/jzhangjobs

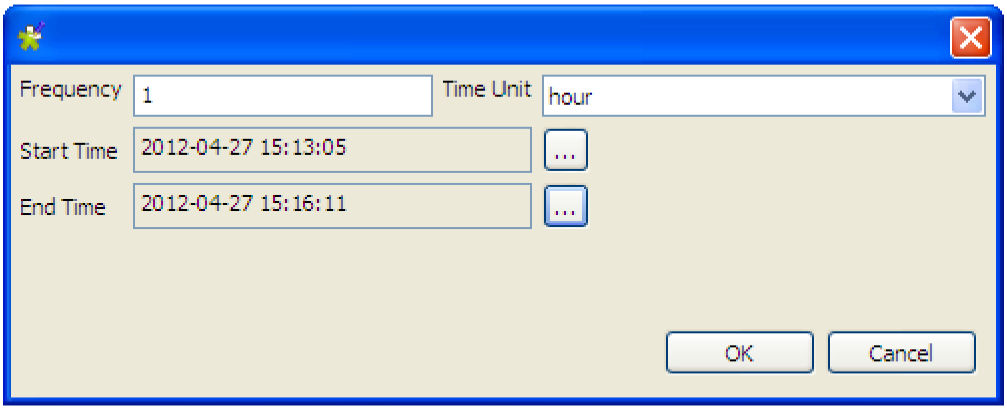
```
Remote running is start-up...
Deploying a job to Hadoop...
Deployment complete!
Job is running ...
```

## How to schedule the executions of a Job

The **Oozie scheduler** feature integrated in *Talend Open Studio for Big Data* enables you to schedule executions of your Job on the HDFS server. Thus, your Job will be executed based on the defined frequency within the set time duration. To configure Job scheduling, do the following:

1. In the **Path** field on the **Oozie scheduler** tab, enter the path where your Job will be deployed on the HDFS server if the deployment path is not yet defined.

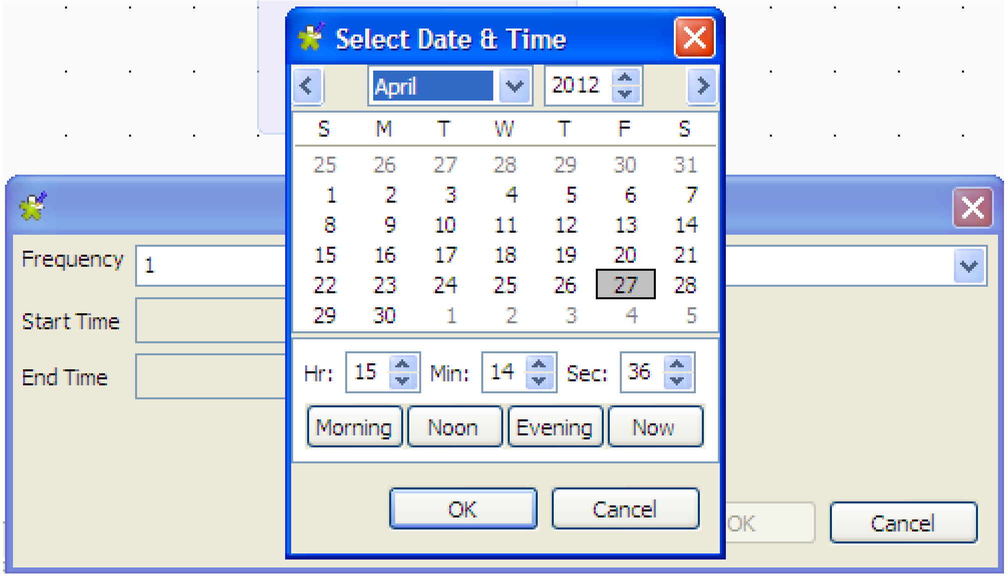
2. Click the **Schedule** button on the **Oozie scheduler** tab to open the scheduling setup dialog box.



Frequency: 1
Time Unit: hour
Start Time: 2012-04-27 15:13:05
End Time: 2012-04-27 15:16:11

3. Fill in the **Frequency** field with an integer and select a time unit from the **Time Unit** list to define the Job execution frequency.

4. Click the **[...]** button next to the **Start Time** field to open the **[Select Date & Time]** dialog box, select the date, hour, minute, and second values, and click **OK** to set the Job execution start time. Then, set the Job execution end time in the same way.



Frequency 1
Start Time
End Time

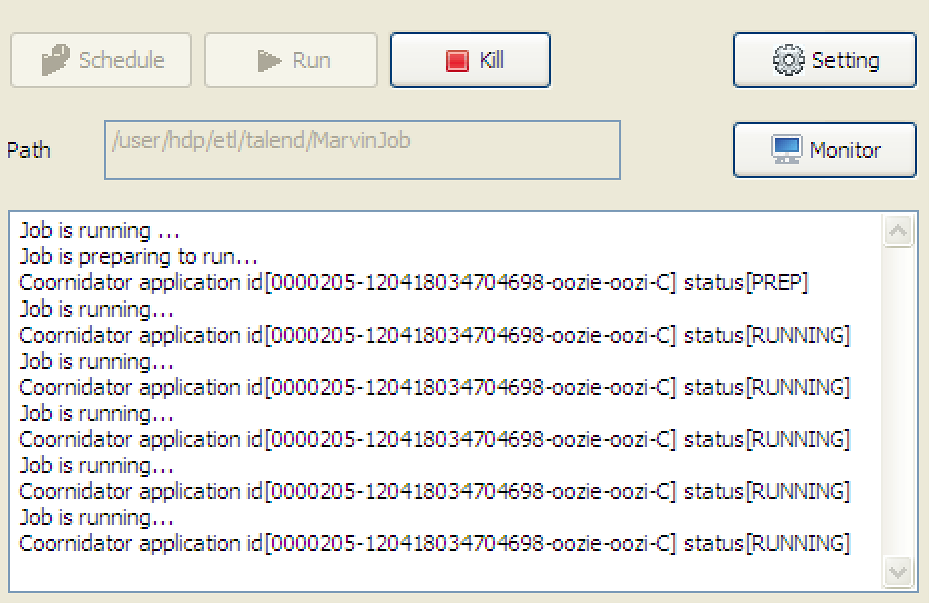
April 2012

| S | M | T | W | T | F | S |
|---|---|---|---|---|---|---|
| 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 8 | 9 | 10 | 11 | 12 | 13 | 14 |
| 15 | 16 | 17 | 18 | 19 | 20 | 21 |
| 22 | 23 | 24 | 25 | 26 | 27 | 28 |
| 29 | 30 | 1 | 2 | 3 | 4 | 5 |

Hr: 15 Min: 14 Sec: 36

5. Click **OK** to close the dialog box and start scheduled executions of your Job.

   The Job automatically runs based on the defined scheduling parameters. To stop the Job, click **Kill**.

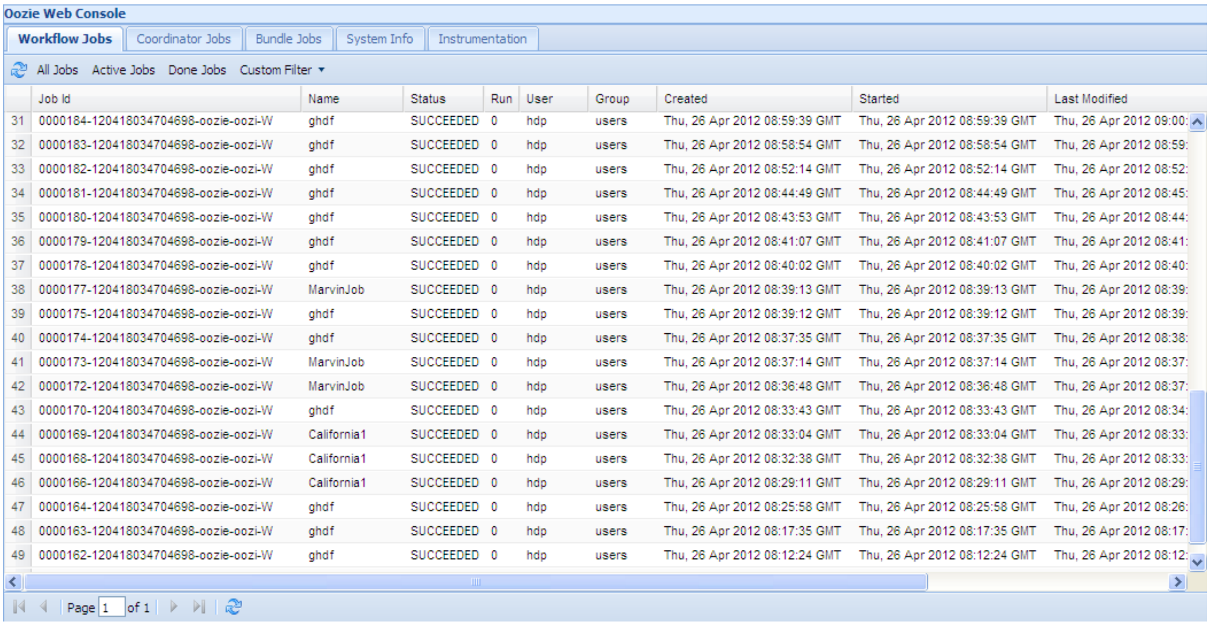


Path /user/hdp/etl/talend/MarvinJob

```
Job is running ...
Job is preparing to run...
Coornidator application id[0000205-120418034704698-oozie-oozi-C] status[PREP]
Job is running...
Coornidator application id[0000205-120418034704698-oozie-oozi-C] status[RUNNING]
Job is running...
Coornidator application id[0000205-120418034704698-oozie-oozi-C] status[RUNNING]
Job is running...
Coornidator application id[0000205-120418034704698-oozie-oozi-C] status[RUNNING]
Job is running...
Coornidator application id[0000205-120418034704698-oozie-oozi-C] status[RUNNING]
Job is running...
Coornidator application id[0000205-120418034704698-oozie-oozi-C] status[RUNNING]
```

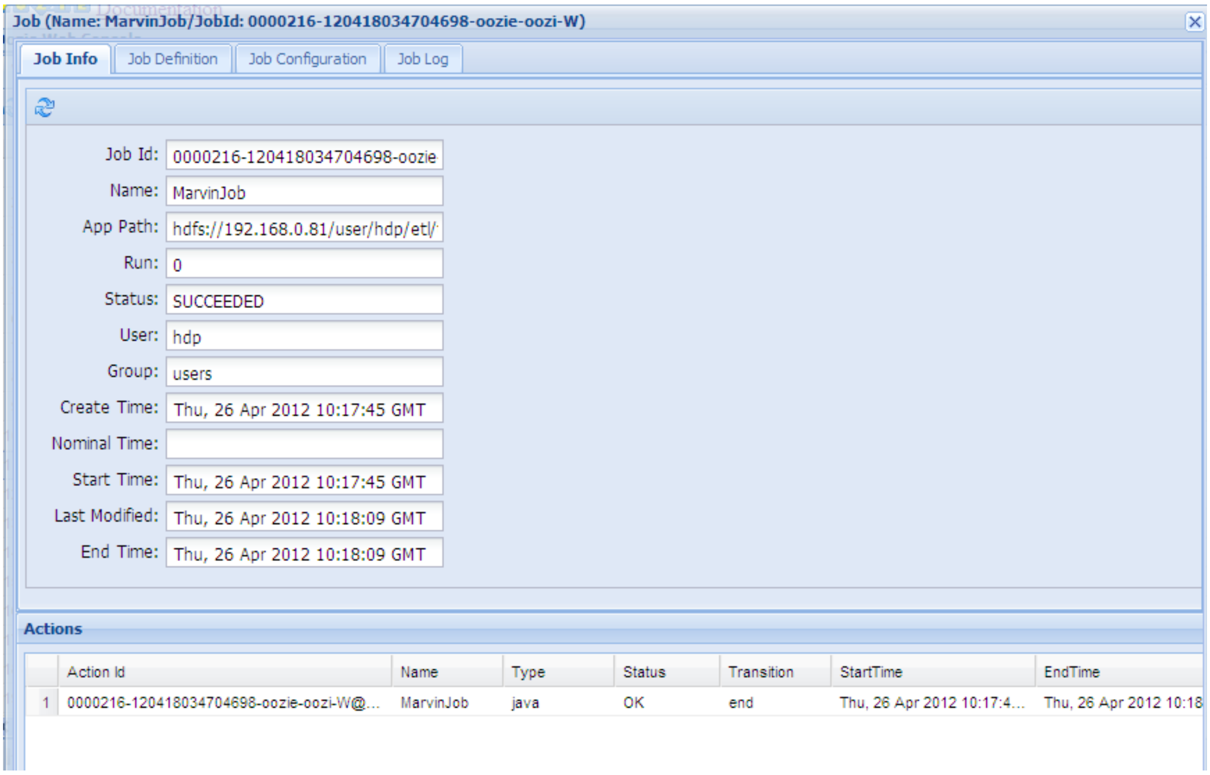
## How to monitor Job execution status

To monitor Job execution status and results, click the **Monitor** button on the **Oozie scheduler** tab. The Oozie end point URI opens in your Web browser, displaying the execution information of the Jobs on the HDFS server.



| | Job Id | Name | Status | Run | User | Group | Created | Started | Last Modified |
|---|---|---|---|---|---|---|---|---|---|
| 31 | 0000184-120418034704698-oozie-oozi-W | ghdf | SUCCEEDED | 0 | hdp | users | Thu, 26 Apr 2012 08:59:39 GMT | Thu, 26 Apr 2012 08:59:39 GMT | Thu, 26 Apr 2012 09:00: |
| 32 | 0000183-120418034704698-oozie-oozi-W | ghdf | SUCCEEDED | 0 | hdp | users | Thu, 26 Apr 2012 08:58:54 GMT | Thu, 26 Apr 2012 08:58:54 GMT | Thu, 26 Apr 2012 08:59: |
| 33 | 0000182-120418034704698-oozie-oozi-W | ghdf | SUCCEEDED | 0 | hdp | users | Thu, 26 Apr 2012 08:52:14 GMT | Thu, 26 Apr 2012 08:52:14 GMT | Thu, 26 Apr 2012 08:52: |
| 34 | 0000181-120418034704698-oozie-oozi-W | ghdf | SUCCEEDED | 0 | hdp | users | Thu, 26 Apr 2012 08:44:49 GMT | Thu, 26 Apr 2012 08:44:49 GMT | Thu, 26 Apr 2012 08:45: |
| 35 | 0000180-120418034704698-oozie-oozi-W | ghdf | SUCCEEDED | 0 | hdp | users | Thu, 26 Apr 2012 08:43:53 GMT | Thu, 26 Apr 2012 08:43:53 GMT | Thu, 26 Apr 2012 08:44: |
| 36 | 0000179-120418034704698-oozie-oozi-W | ghdf | SUCCEEDED | 0 | hdp | users | Thu, 26 Apr 2012 08:41:07 GMT | Thu, 26 Apr 2012 08:41:07 GMT | Thu, 26 Apr 2012 08:41: |
| 37 | 0000178-120418034704698-oozie-oozi-W | ghdf | SUCCEEDED | 0 | hdp | users | Thu, 26 Apr 2012 08:40:02 GMT | Thu, 26 Apr 2012 08:40:02 GMT | Thu, 26 Apr 2012 08:40: |
| 38 | 0000177-120418034704698-oozie-oozi-W | MarvinJob | SUCCEEDED | 0 | hdp | users | Thu, 26 Apr 2012 08:39:13 GMT | Thu, 26 Apr 2012 08:39:13 GMT | Thu, 26 Apr 2012 08:39: |
| 39 | 0000175-120418034704698-oozie-oozi-W | ghdf | SUCCEEDED | 0 | hdp | users | Thu, 26 Apr 2012 08:39:12 GMT | Thu, 26 Apr 2012 08:39:12 GMT | Thu, 26 Apr 2012 08:39: |
| 40 | 0000174-120418034704698-oozie-oozi-W | ghdf | SUCCEEDED | 0 | hdp | users | Thu, 26 Apr 2012 08:37:35 GMT | Thu, 26 Apr 2012 08:37:35 GMT | Thu, 26 Apr 2012 08:38: |
| 41 | 0000173-120418034704698-oozie-oozi-W | MarvinJob | SUCCEEDED | 0 | hdp | users | Thu, 26 Apr 2012 08:37:14 GMT | Thu, 26 Apr 2012 08:37:14 GMT | Thu, 26 Apr 2012 08:37: |
| 42 | 0000172-120418034704698-oozie-oozi-W | MarvinJob | SUCCEEDED | 0 | hdp | users | Thu, 26 Apr 2012 08:36:48 GMT | Thu, 26 Apr 2012 08:36:48 GMT | Thu, 26 Apr 2012 08:37: |
| 43 | 0000170-120418034704698-oozie-oozi-W | ghdf | SUCCEEDED | 0 | hdp | users | Thu, 26 Apr 2012 08:33:43 GMT | Thu, 26 Apr 2012 08:33:43 GMT | Thu, 26 Apr 2012 08:34: |
| 44 | 0000169-120418034704698-oozie-oozi-W | California1 | SUCCEEDED | 0 | hdp | users | Thu, 26 Apr 2012 08:33:04 GMT | Thu, 26 Apr 2012 08:33:04 GMT | Thu, 26 Apr 2012 08:33: |
| 45 | 0000168-120418034704698-oozie-oozi-W | California1 | SUCCEEDED | 0 | hdp | users | Thu, 26 Apr 2012 08:32:38 GMT | Thu, 26 Apr 2012 08:32:38 GMT | Thu, 26 Apr 2012 08:33: |
| 46 | 0000166-120418034704698-oozie-oozi-W | California1 | SUCCEEDED | 0 | hdp | users | Thu, 26 Apr 2012 08:29:11 GMT | Thu, 26 Apr 2012 08:29:11 GMT | Thu, 26 Apr 2012 08:29: |
| 47 | 0000164-120418034704698-oozie-oozi-W | ghdf | SUCCEEDED | 0 | hdp | users | Thu, 26 Apr 2012 08:25:58 GMT | Thu, 26 Apr 2012 08:25:58 GMT | Thu, 26 Apr 2012 08:26: |
| 48 | 0000163-120418034704698-oozie-oozi-W | ghdf | SUCCEEDED | 0 | hdp | users | Thu, 26 Apr 2012 08:17:35 GMT | Thu, 26 Apr 2012 08:17:35 GMT | Thu, 26 Apr 2012 08:17: |
| 49 | 0000162-120418034704698-oozie-oozi-W | ghdf | SUCCEEDED | 0 | hdp | users | Thu, 26 Apr 2012 08:12:24 GMT | Thu, 26 Apr 2012 08:12:24 GMT | Thu, 26 Apr 2012 08:12: |

To display the detailed information of a particular Job, click any field of that Job to open a separate page showing the details of the Job.


Job (Name: MarvinJob/JobId: 0000216-120418034704698-oozie-oozi-W)
Job Info
Job Definition
Job Configuration
Job Log
Actions

| | |
|---|---|
| Job Id: | 0000216-120418034704698-oozie |
| Name: | MarvinJob |
| App Path: | hdfs://192.168.0.81/user/hdp/etl/ |
| Run: | 0 |
| Status: | SUCCEEDED |
| User: | hdp |
| Group: | users |
| Create Time: | Thu, 26 Apr 2012 10:17:45 GMT |
| Nominal Time: | |
| Start Time: | Thu, 26 Apr 2012 10:17:45 GMT |
| Last Modified: | Thu, 26 Apr 2012 10:18:09 GMT |
| End Time: | Thu, 26 Apr 2012 10:18:09 GMT |

| | Action Id | Name | Type | Status | Transition | StartTime | EndTime |
|---|---|---|---|---|---|---|---|
| 1 | 0000216-120418034704698-oozie-oozi-W@... | MarvinJob | java | OK | end | Thu, 26 Apr 2012 10:17:4... | Thu, 26 Apr 2012 10:18 |

## 3.2.8. How to customize your workspace

When using *Talend Open Studio for Big Data* to design a data integration Job, you can customize the **Palette** layout and setting according to your needs. You can as well change the position of any of the panels that exist in the Studio to meet your requirements.

All the panels, tabs, and views described in this documentation are specific to *Talend Open Studio for Big Data*. 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/.

### 3.2.8.1. How to change the Palette layout and settings

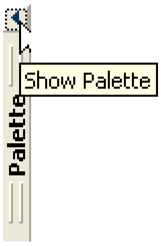
The **Palette** contains all basic technical components necessary to create the most complex Jobs in the design workspace. These components are grouped in families and sub-families.

For specific component configuration, check *Talend Open Studio for Big Data Components Reference Guide*.

*Talend Open Studio for Big Data* 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**.

#### How to show, hide the Palette and change its position

By default, the **Palette** might be hidden on the right hand side of your design workspace.



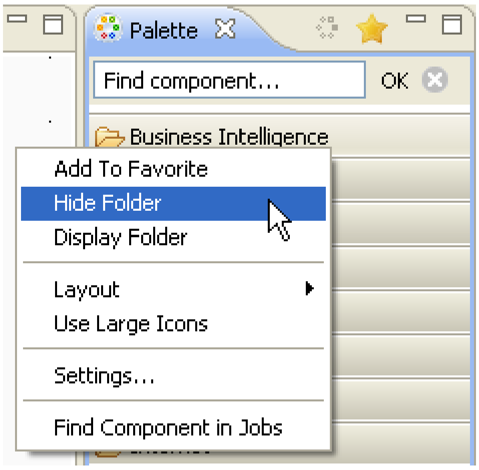
If you want the **Palette** to show permanently, click the left arrow, at the upper right corner of the design workspace, to make it visible at all times.

You can also move around the **Palette** outside the design workspace within *Talend Open Studio for Big Data*'s main window. To enable the standalone **Palette** view, click the **Window** menu > **Show View...** > **General** > **Palette**.

If you want to set the Palette apart in a panel, right-click the **Palette** head bar and select **Detached** from the contextual menu. The **Palette** opens in a separate view that you can move around wherever you like within *Talend Open Studio for Big Data*'s main window.

#### How to display/hide components families

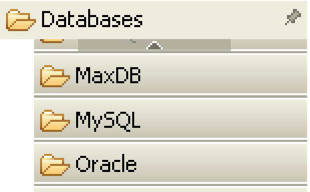
You can display/hide components families according to your needs in case of visibility problems, for example. To do so, right-click the **Palette** and select **Display folder** to display components families and **Hide folder** to display components without their families.



This display/hide option can be very useful when you are in the **Favorite** view of the **Palette**. In this view, you usually have a limited number of components that if you display without their families, you will have them in an alphabetical list and thus facilitate their usage. for more information about the **Palette** favorite, see section *How to set the Palette favorite*.

## How to maintain a component family open

If you often use one or many component families, you can add a pin on their names to stop them from collapsing when you select components from other families.



To add a pin, click the pin icon on the top right-hand corner of the family name.

## How to filter the Palette

You can select the components to be shown or hidden on your **Palette**. You can also add to the **Palette** the components that you developed yourself.

For more information about filtering the **Palette**, see section *Palette Settings*.

For more information about adding components to the **Palette**, either from **Talend Exchange** or from your own development, see section *How to download/upload Talend Community components* and/or section *External or User components (Talend > Components)*.

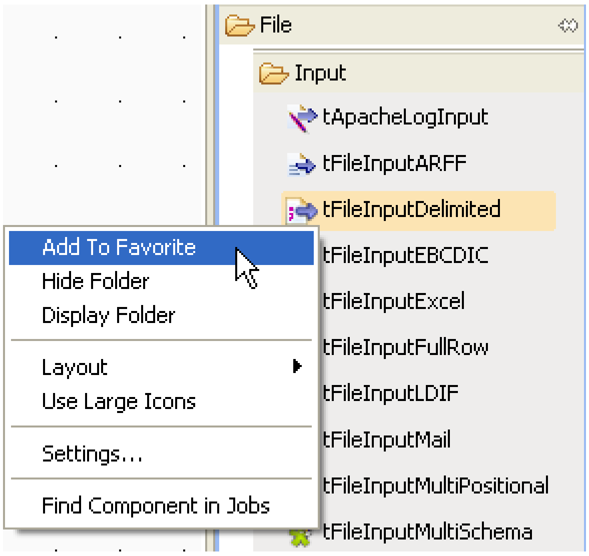
## How to set the Palette favorite

The **Palette** offers you search and favorite possibilities that by turn facilitate its usage.

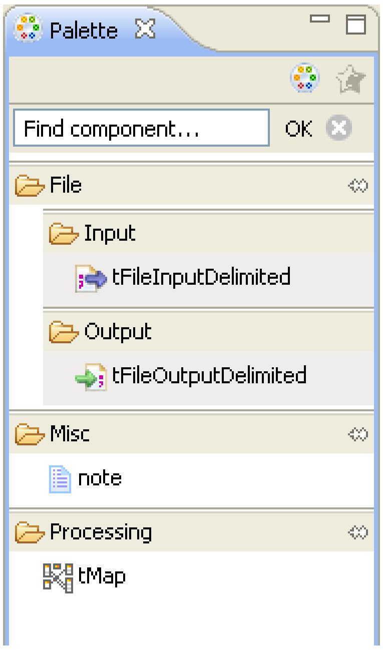
You can add/remove components to/from the **Palette** favorite view of *Talend Open Studio for Big Data* in order to have a quick access to all the components that you mostly use.

To do so:

1. From the **Palette**, right-click the component you want to add to **Palette** favorite and select **Add To Favorite**.



2. Do the same for all the components you want to add to the **Palette** favorite then click the **Favorite** ⭐ button in the upper right corner of the **Palette** to display the **Palette** favorite.



Only the components added to the favorite are displayed.

To delete a component from the **Palette** favorite, right-click the component you want to remove from the favorite and select **Remove From Favorite**.

To restore the **Palette** standard view, click the **Standard** button in the upper right corner of the **Palette**.

### How to change components layout in the Palette

You can change the layout of the component list in the **Palette** to display them in columns or in lists, as icons only or as icons with short description.

You can also enlarge the component icons for better readability of the component list.

To do so, right-click any component family in the **Palette** and select the desired option in the contextual menu or click **Settings** to open the **[Palette Settings]** window and fine-tune the layout.

### How to add external components to the Palette

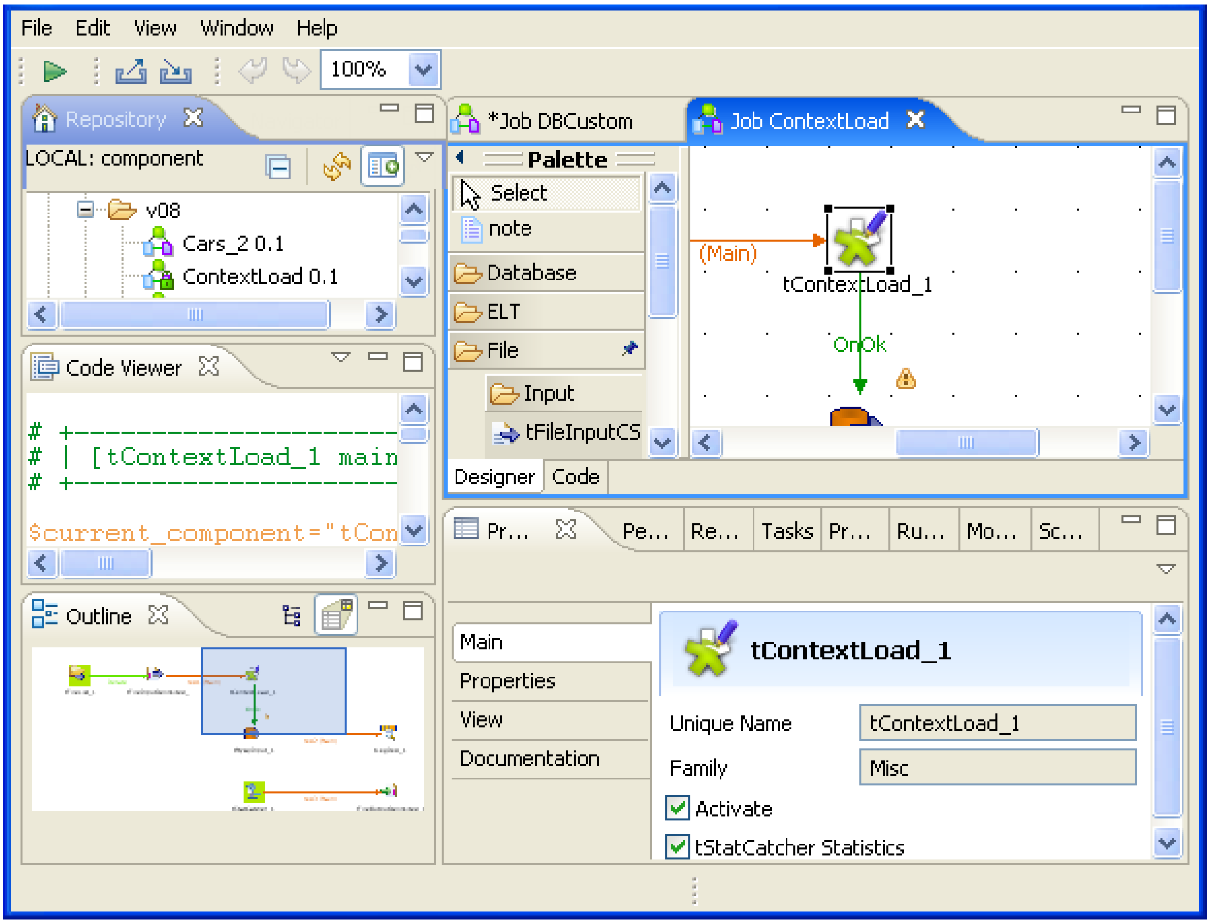
*Talend Open Studio for Big Data* enables you to add external components to the **Palette** of your Studio and use them in your Job designs.

For more information about the creation and development of user components, refer to the component creation tutorial on our wiki at http://www.talendforge.org/wiki/doku.php?id=component_creation.

For more information about how to download user components in your Studio, see section *How to download/upload Talend Community components*.

## 3.2.8.2. How to change panels positions

All panels in the open Studio can be moved around according to your needs.



```
# +---------------------
# | [tContextLoad_1 main
# +---------------------

$current_component="tCon
```

Unique Name: tContextLoad_1

Family: Misc

☑ Activate

☑ tStatCatcher Statistics

All you need to do is to click the head border of a panel or to click a tab, hold down the mouse button and drag the panel to the target destination. Release to change the panel position.

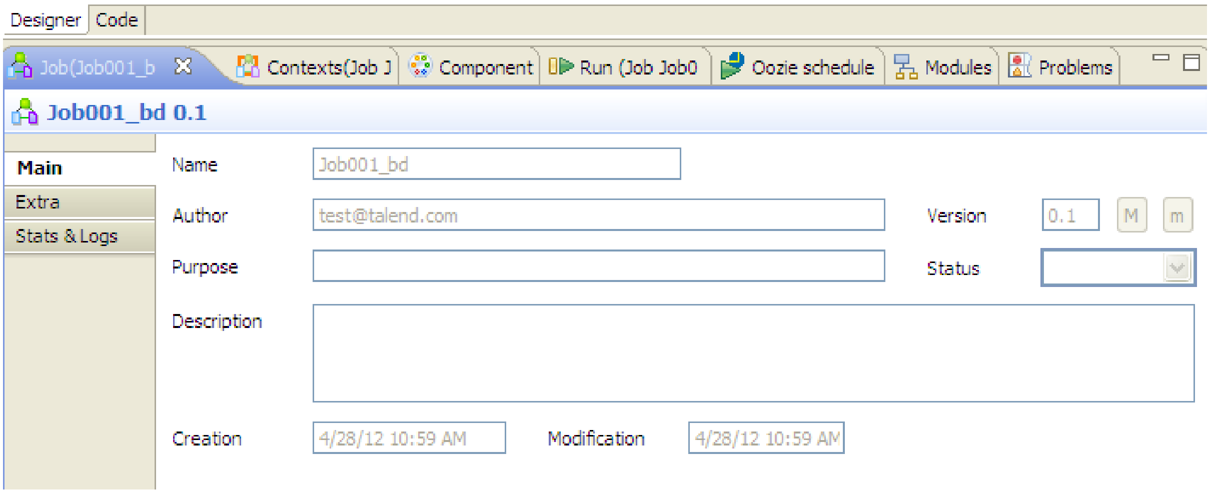
Click the minimize/maximize icons (▭/▢) to minimize the corresponding panel or maximize it. For more information on how to display or hide a panel/view, see section *How to display Job configuration tabs/views*.

Click the close icon (☒) to close a tab/view. To reopen a view, click **Window** > **Show View** > **Talend**, then click the name of the panel you want to add to your current view or see section *Shortcuts and aliases* .

If the **Palette** does not show or if you want to set it apart in a panel, go to **Window** > **Show view...**> **General** > **Palette**. The **Palette** opens in a separate view that you can move around wherever you like within *Talend Open Studio for Big Data*'s main window.

### 3.2.8.3. How to display Job configuration tabs/views

The configuration tabs are located in the lower half of the design workspace. Each tab opens a view that displays detailed information about the selected element in the design workspace.



Name: Job001_bd

Author: test@talend.com

Version: 0.1

Purpose:

Status:

Description:

Creation: 4/28/12 10:59 AM

Modification: 4/28/12 10:59 AM

The **Component**, **Run Job**, **Contexts**, and **Oozie scheduler** views gather all information relative to the graphical elements selected in the design workspace or the actual execution of the open Job.

By default, when you launch *Talend Open Studio for Big Data* for the first time, the **Problems** tab will not be displayed until the first Job is created. After that, **Problems** tab will be displayed in the tab system automatically.

The **Modules** and **Scheduler[deprecated]** tabs are located in the same tab system as the **Component**, **Logs** and **Run Job** tabs. Both views are independent from the active or inactive Jobs open on the design workspace.

Some of the configuration tabs are hidden by default such as the **Error Log, Navigator**, **Job Hierarchy**, **Problems**, **Modules** and **Scheduler[deprecated]** tabs. You can show hidden tabs in this tab system and directly open the corresponding view if you select **Window** > **Show view** and then, in the open dialog box, expand the corresponding node and select the element you want to display.

For detailed description about these tabs, see section *Configuration tabs* .

# 3.3. Using connections

In *Talend Open Studio for Big Data*, a Job or a subjob is composed of a group of components logically linked to one another via connections. This section will describe the types of connections and their related settings.

## 3.3.1. Connection types

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

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

The sections below describe all available connection types.

### 3.3.1.1. Row connection

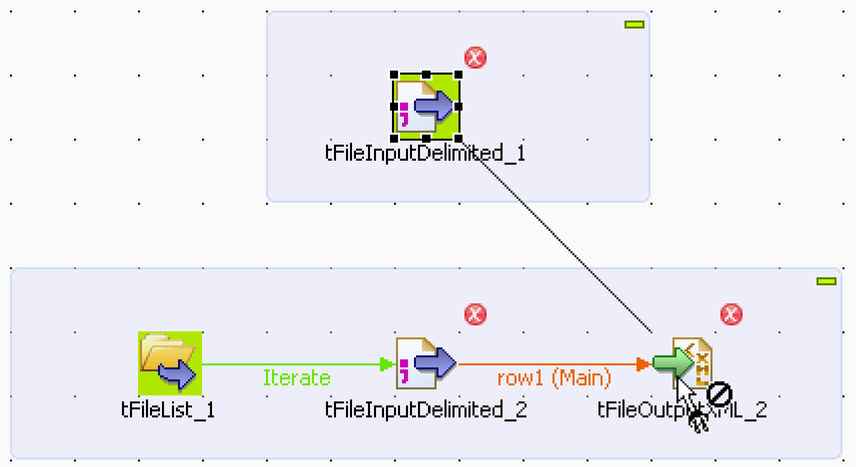
A **Row** connection handles the actual data. The **Row** connections can be **main**, **lookup**, **reject** or **output** according to the nature of the flow processed.

## Main

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

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

You cannot connect two Input components together using a **main Row** 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 linking a component will be called **Lookup**.



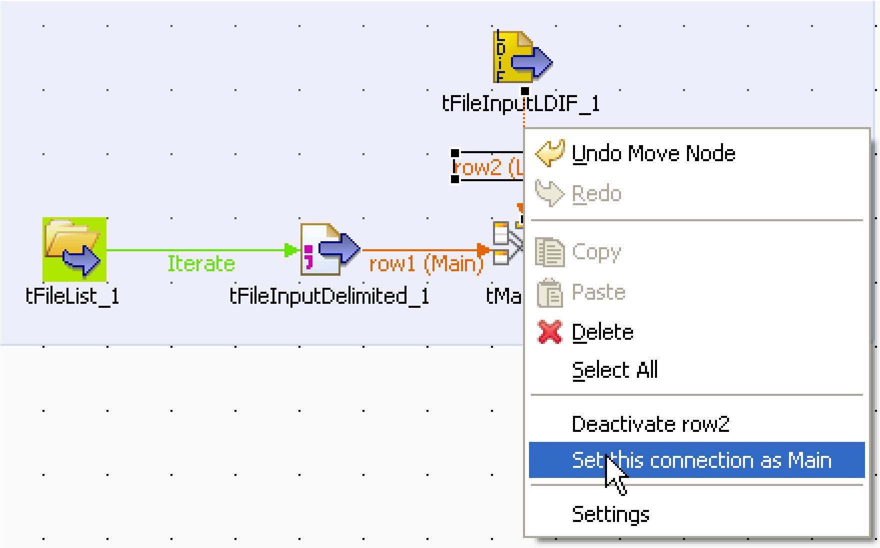
To connect two components using a Main connection, right-click the input component and select **Row** > **Main** on the connection list.

Alternatively, you can click the component to highlight it, then right-click it 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 section *Multiple Input/Output*.

## Lookup

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



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

Related topic: section *Multiple Input/Output*.

### Filter

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

### Rejects

This row link connects a processing component to an output component. This row link gathers the data that does NOT match the filter or are not valid for the expected output. This link allows you to track the data that could not be processed for any reason (wrong type, undefined null value, etc.). On some components, this link is enabled when the **Die on error** option is deactivated. For more information, refer to the relevant component properties available in *Talend Open Studio for Big Data Components Reference Guide*.

### ErrorReject

This row link connects a **tMap** component to an output component. This link 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: section *Handling errors*.

### Output

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

The system also remembers deleted output link 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: section *Multiple Input/Output*.

### Uniques/Duplicates

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

The **Uniques** link 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** link 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 chapter *Mapping data flows*.

For properties regarding the **tMap** component as well as use case scenarios, see *Talend Open Studio for Big Data Components Reference Guide*.

## 3.3.1.2. Iterate connection

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

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

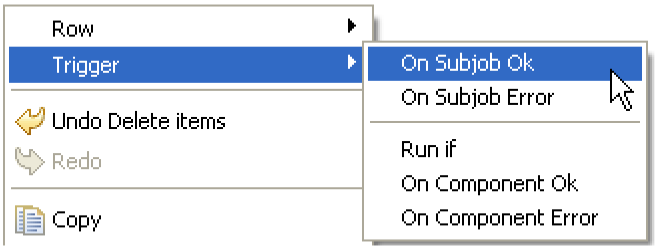
Some components such as the **tFileList** component are meant to be connected through an iterate link with the next component. For how to set an **Iterate** connection, see section *Iterate connection settings*.

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

## 3.3.1.3. Trigger connections

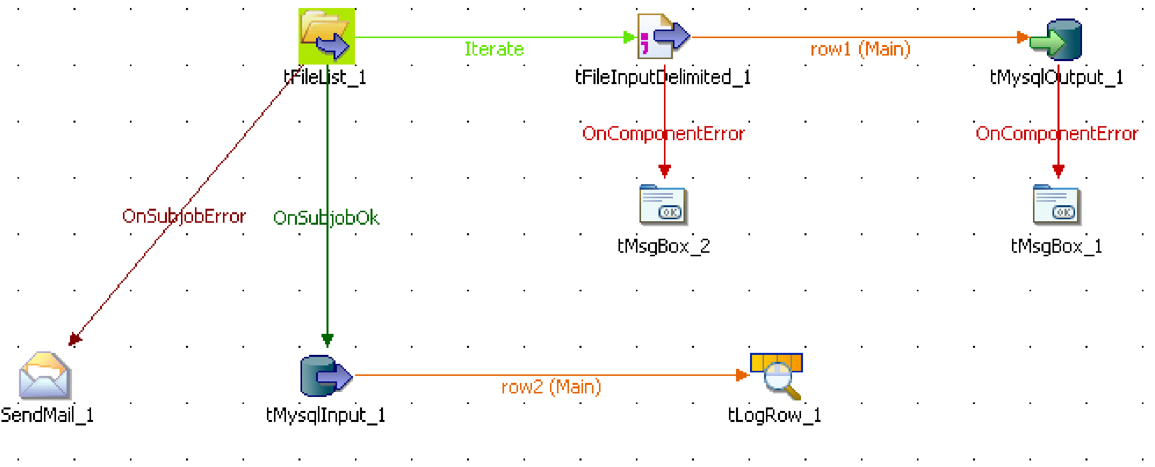
Trigger connections define the processing sequence, i.e. no data is handled through these connections.

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

Related topic: section *How to define the Start component*.

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

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

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

**Run if** triggers a subjob or component in case the condition defined is met. For how to set a trigger condition, see section *Run if connection settings*.

## 3.3.1.4. Link connection

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

Related topics: ELT components in *Talend Open Studio for Big Data Components Reference Guide*.

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

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

*Be aware that the name you provide to the link MUST reflects the actual table name.*

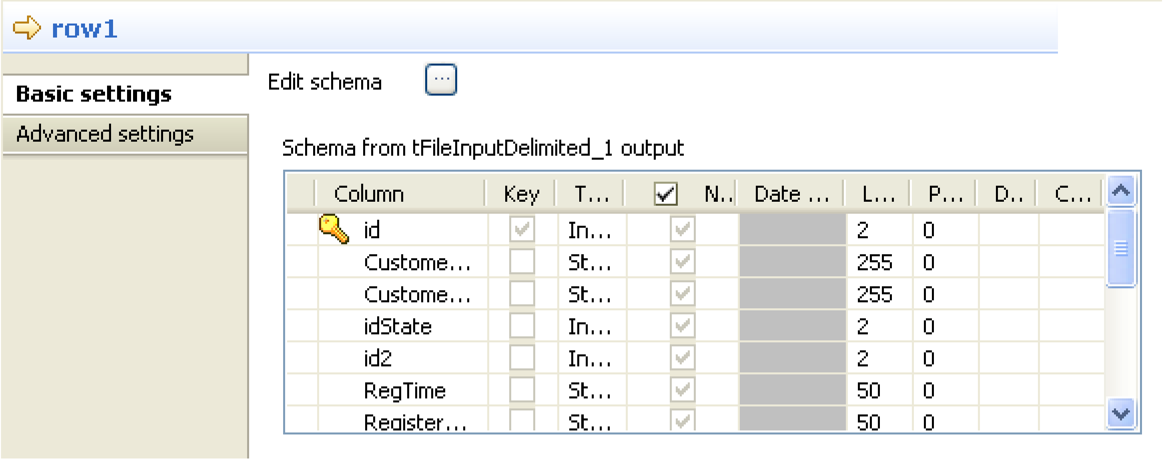
In fact, the link name will be used in the SQL statement generated through the ETL Mapper, therefore the same name should never be used twice.

# 3.3.2. How to define connection settings

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

## 3.3.2.1. Row connection settings

The **Basic settings** vertical tab of the **Component** view of the connection displays the schema of the data flow handled by the connection. You can change the schema by clicking the **Edit schema** button. Once you change the schema of the data flow, the schema type of the two components across the connection will become **Built-In**. For more information, see section *How to set a schema*.

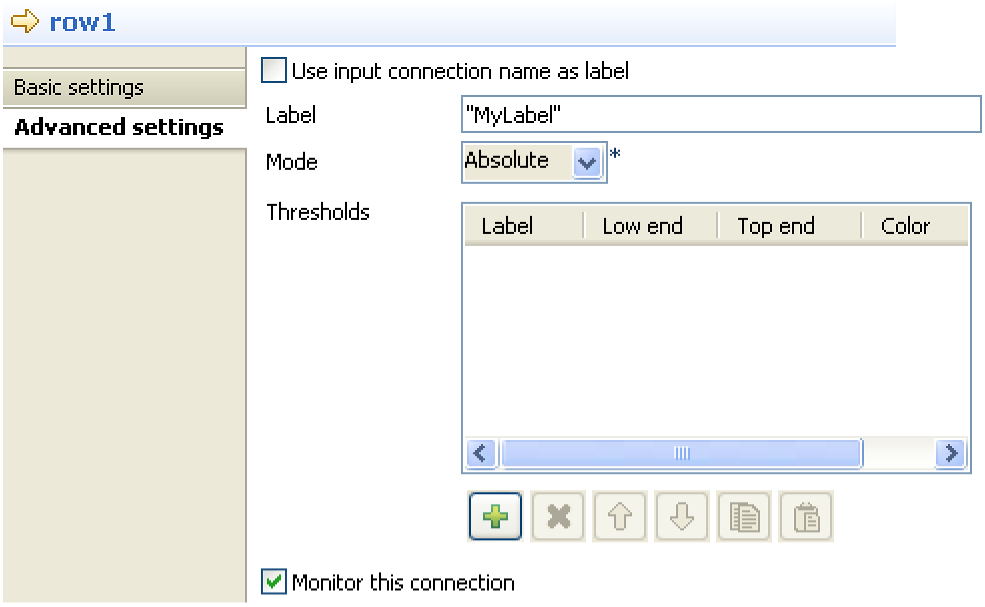


Edit schema ...

Schema from tFileInputDelimited_1 output

| Column | Key | T... | ☑ | N.. | Date ... | L... | P... | D.. | C... |
|---|---|---|---|---|---|---|---|---|---|
| id | ✓ | In... | ✓ | | | 2 | 0 | | |
| Custome... | □ | St... | ✓ | | | 255 | 0 | | |
| Custome... | □ | St... | ✓ | | | 255 | 0 | | |
| idState | □ | In... | ✓ | | | 2 | 0 | | |
| id2 | □ | In... | ✓ | | | 2 | 0 | | |
| RegTime | □ | St... | ✓ | | | 50 | 0 | | |
| Register... | □ | St... | ✓ | | | 50 | 0 | | |

The **Advanced settings** vertical tab lets you monitor the data flow over the connection in a Job without using a separate **tFlowMeter** component. The measured information will be interpreted and displayed in a supervising tool such as *Talend Activity Monitoring Console*. For information about *Talend Activity Monitoring Console*, see *Talend Activity Monitoring Console User Guide*.



□ Use input connection name as label

Label "MyLabel"

Mode Absolute *

Thresholds

| Label | Low end | Top end | Color |
|---|---|---|---|

☑ Monitor this connection

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

1. Select the **Monitor this connection** check box.

2. Select **Use input connection name as label** to use the name of the input flow to label your data to be logged, or enter a label in the **Label** field.

3. 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.

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

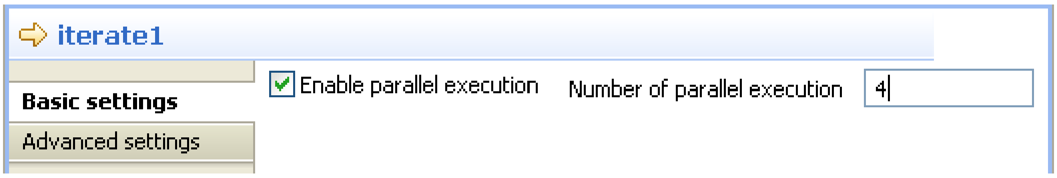
For more information about flow metrics, see the documentation of the **tFlowMeterCatcher** component in *Talend Open Studio for Big Data Components Reference Guide* and see *Talend Activity Monitoring Console User Guide*.

## 3.3.2.2. Iterate connection settings

You can set an **Iterate** link to run parallel iterations:

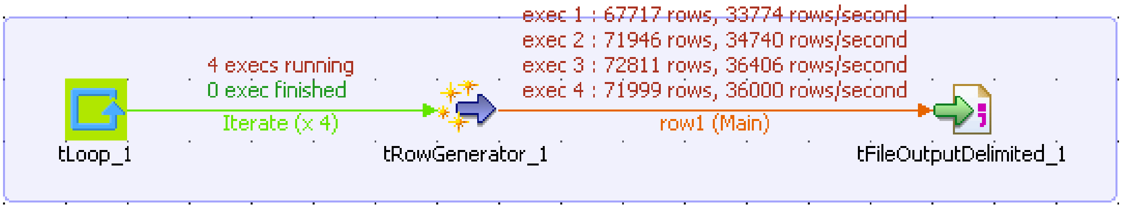
1. Simply select the **Iterate** link of your subjob to display the related **Basic settings** view of the **Components** tab.

2. Select the **Enable parallel execution** check box and set the number of executions to be carried out in parallel.



☑ Enable parallel execution
Number of parallel execution: 4

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



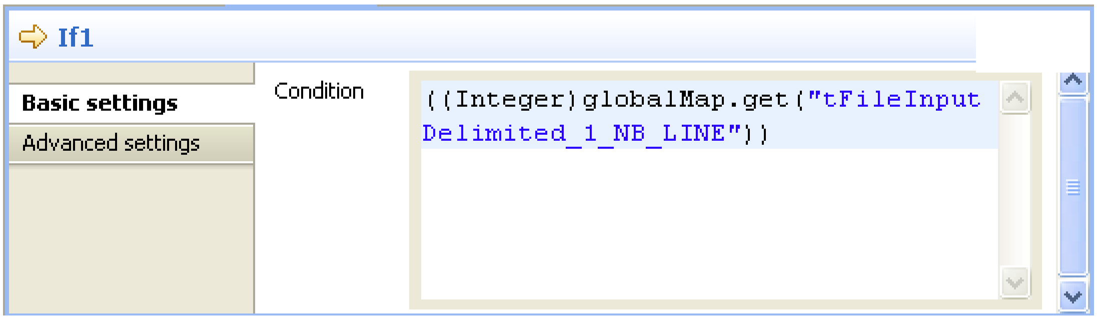
4 execs running
0 exec finished
Iterate (x 4)
exec 1 : 67717 rows, 33774 rows/second
exec 2 : 71946 rows, 34740 rows/second
exec 3 : 72811 rows, 36406 rows/second
exec 4 : 71999 rows, 36000 rows/second
row1 (Main)
tLoop_1
tRowGenerator_1
tFileOutputDelimited_1

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

## 3.3.2.3. Trigger connection settings

### Run if connection settings

In the **Basic settings** view of a **Run if** connection, you can set the condition to the Subjob in Java. Pressing **Ctrl +Space** allows you to access all global and context variables.



```
((Integer)globalMap.get("tFileInputDelimited_1_NB_LINE"))
```

# 3.4. Using the Metadata Manager

*Talend Open Studio for Big Data* is a metadata-driven solution, and can therefore help you ensure the whole Job consistency and quality through a centralized Metadata Manager.

In the integration process, the Metadata Manager consolidates all project information in a repository.

## 3.4.1. How to centralize contexts and variables

Depending on the circumstances the Job is being used in, you might want to manage it differently for various execution types (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.

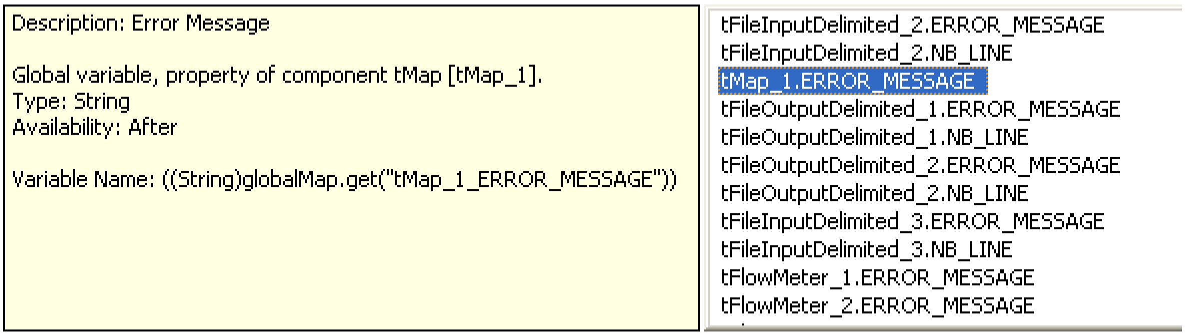
*Talend Open Studio for Big Data* offers you the possibility to create multiple context data sets. Furthermore you can either create context data sets on a one-shot basis, from the context tab of a Job or you can centralize the context data sets in the **Contexts** node of the **Repository** tree view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

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 bar** keystrokes.

### 3.4.1.1. How to use variables in a Job

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

You can use an existing global variable or context variable in any component properties field. Press **Ctrl+Space bar** to display the full list of global and context variables used in various predefined Java functions.


Description: Error Message

Global variable, property of component tMap [tMap_1].
Type: String
Availability: After

Variable Name: ((String)globalMap.get("tMap_1_ERROR_MESSAGE"))

tFileInputDelimited_2.ERROR_MESSAGE
tFileInputDelimited_2.NB_LINE
tMap_1.ERROR_MESSAGE
tFileOutputDelimited_1.ERROR_MESSAGE
tFileOutputDelimited_1.NB_LINE
tFileOutputDelimited_2.ERROR_MESSAGE
tFileOutputDelimited_2.NB_LINE
tFileInputDelimited_3.ERROR_MESSAGE
tFileInputDelimited_3.NB_LINE
tFlowMeter_1.ERROR_MESSAGE
tFlowMeter_2.ERROR_MESSAGE

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

Related topics:

- section *How to define variables from the Component view*
- section *How to use variables in the Contexts view*

## 3.4.1.2. How to use variables in the Contexts view

Various ways are at your disposal to create and define variables. You can manage your variables through the **Contexts** view or directly on the **Component** view.

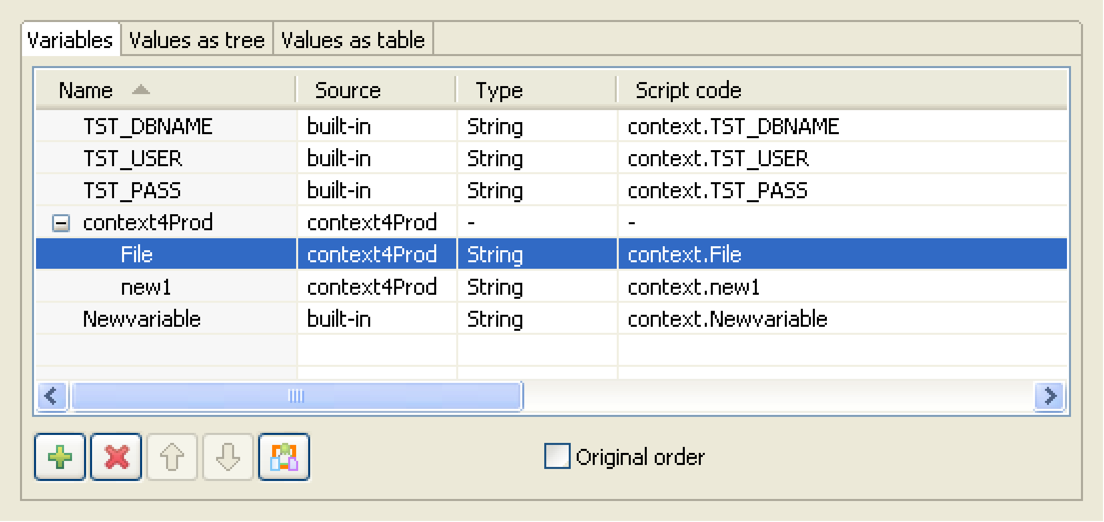
For more information regarding the variable definition directly on the **Component** view, see section *How to define variables from the Component view*.

The **Contexts** view is positioned on the lower part of the design workspace and is made of three tabs: **Variables**, **Values as tree** and **Values as table**.

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

### Variables tab

The **Variables** tab is part of the **Contexts** tab and shows all of the variables that have been defined for each component in the current Job.



| Name | Source | Type | Script code |
|---|---|---|---|
| TST_DBNAME | built-in | String | context.TST_DBNAME |
| TST_USER | built-in | String | context.TST_USER |
| TST_PASS | built-in | String | context.TST_PASS |
| ⊟ context4Prod | context4Prod | - | - |
| File | context4Prod | String | context.File |
| new1 | context4Prod | String | context.new1 |
| Newvariable | built-in | String | context.Newvariable |

□ Original order

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

- Add a parameter line to the table by clicking on **[+]**
- Edit the **Name** of the new variable and type in the <*Newvariable*> name.
- Delete built-in variables. (Reminder: repository variables are read-only.)
- Import variables from a repository context source, using the **Repository variables** button.

- Display the context variables in their original order. They are sorted automatically by the studio upon creation in the tab view or when imported from the **Repository**. To do this, select the **Original order** check box.
- Reorganize the context variables by selecting the variable of interest and then using the ⇧ and ⇩ buttons. To do so, you need select the **Original order** check box to activate the two arrow buttons.

To define the actual value of a newly created variable, click the **Value as tree** tab.

You can add as many entries as you need on the **Variables** tab. By default the variable created is of built-in type.

| **Fields** | **Description** |
|---|---|
| **Name** | Name of the variable. You can edit this field, on the condition that the variable is of Built-in type. Repository variables are read-only. |
| **Source** | **Built-in**: The variable is created in this Job and will be used in this Job only.<br>**<Repository entry name>**: The variable has been defined in a context stored in the repository. The source is thus the actual context group you created in the repository. |
| **Type** | Select the type of data being handled. This is required in Java. |
| **Script code** | Code corresponding to the variable value. Displayed code will be: `context.YourParameterName.` This **Script code** is automatically generated when you define the variable in the **Component** view. |
| **Comment** | Add any useful comment. |

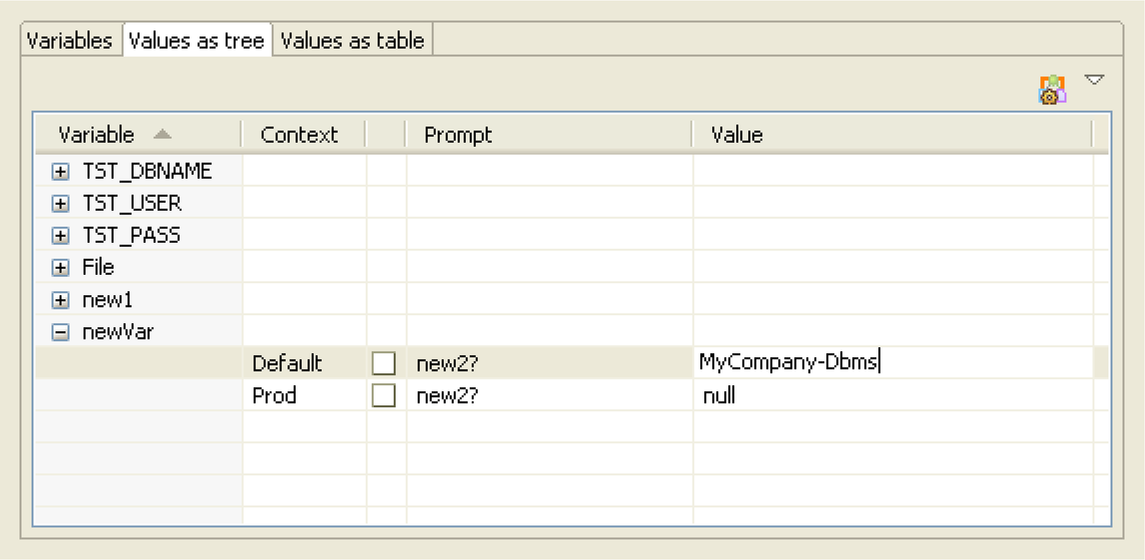
You cannot create contexts from the **Variables** view, but only from the **Values as table** or **as tree** views.

For further information regarding variable definition on the component view, see section *How to define variables from the Component view*.

For more information about the repository variables, see section *How to store contexts in the repository*.

## Values as tree tab

This tab shows the variables as well as their values in a tree view.



| Variable | Context | | Prompt | Value |
|---|---|---|---|---|
| ⊞ TST_DBNAME | | | | |
| ⊞ TST_USER | | | | |
| ⊞ TST_PASS | | | | |
| ⊞ File | | | | |
| ⊞ new1 | | | | |
| ⊟ newVar | | | | |
| | Default | ☐ | new2? | MyCompany-Dbms |
| | Prod | ☐ | new2? | null |

From this view, you can:

- Define the value of a built-in variable directly in the **Value** field. Note that repository variables values are read-only and can only be edited in the relevant repository context.
- Define a question to prompt the user for variable value confirmation at execution time.
- Create or Edit a context name through the top right dedicated button.
- Rearrange the variable/context groupby display.

| **Fields** | **Description** |
|---|---|
| **Variable** | Name of the variables. |
| **Context** | Name of the contexts. |
| **Prompt** | Select this check box, if you want the variable to be editable in the confirmation dialog box at execution time. |
|  | If you asked for a prompt to popup, fill in this field to define the message to show on the dialog box. |
| **Value** | Value for the corresponding variable. Define the value of your built-in variables. Note that repository variables are read-only. |

You can manage your contexts from this tab, through the dedicated button placed on the top right hand side of the **Contexts** view. See section *How to configure contexts* for further information regarding the context management.

On the **Values as tree** tab, you can display the values based on the *contexts* or on the *variables* for more clarity.

To change the way the values are displayed on the tree, click the small down arrow button, then click the **group by** option you want.

For more information regarding variable definition, see section *How to define variables from the Component view* and section *How to store contexts in the repository*.

## Values as table tab

This **Values as table** tab shows the context and variable settings in the form of a table.

| **Fields** | **Description** |
|---|---|
| **Name** | Name of the variable. |
| **<YourContextName>** | Corresponding value for the variable. |

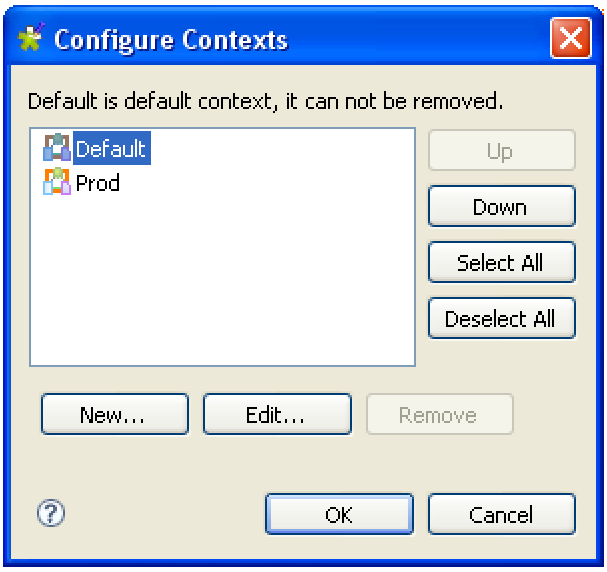
You can manage your contexts from this tab, through the **Configure contexts** button placed on the top right hand side of the **Contexts** panel. See section *How to configure contexts* for further information regarding the context management.

For more information regarding variable definition, see section *How to define variables from the Component view* and section *How to store contexts in the repository*.

# 3.4.1.3. How to configure contexts

You can only manage your contexts from the **Values as table** or **Values as tree** tabs. A dedicated button shows up on the top right hand side of the **Contexts** view.

Click the **Configure Contexts...** icon to open the management dialog box.



Configure Contexts

Default is default context, it can not be removed.

The default context cannot be removed, therefore the **Remove** button is unavailable. To make it editable, select another context on the list.

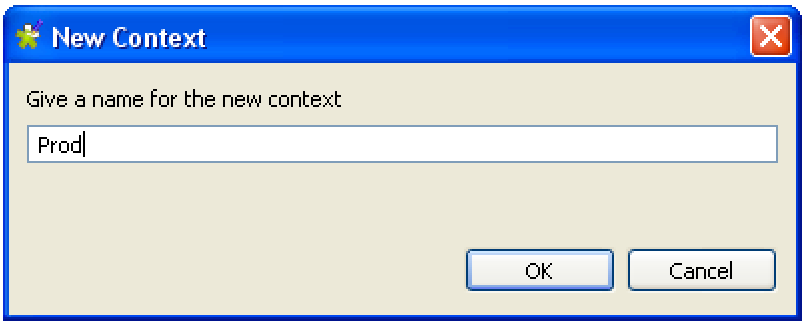
## Creating a context

Based on the default context you set, you can create as many contexts as you need.

To create a new context:

1. Click **New** in the **[Configure Contexts]** dialog box.

2. Type in a name for the new context.



New Context

Give a name for the new context

Prod

3. Click **OK** to validate the creation.

When you create a new context, the entire default context legacy is copied over to the new context. You hence only need to edit the relevant fields on the **Value as tree** tab to customize the context according to your use.

The drop-down list **Default Context** shows all the contexts you created.

You can switch default context by simply selecting the new default context on the **Default Context** list on the **Variables** tab of the **Contexts** view.

Note that the Default (or last) context can never be removed. There should always be a context to run the Job, be this context called Default or any other name.

## Renaming or editing a context

To change the name of an existing context:

1. Click **Edit** on the **[Configure contexts]** dialog box and enter the new context name in the dialog box showing up.

2. Click **OK** to validate the change.

To carry out changes on the actual values of the context variables, go to the **Values as tree** or **Values as table** tabs. For more information about these tabs, see section *How to use variables in the Contexts view*.

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

Various ways are at your disposal to create and define context variables. You can mostly manage your variables from the **Contexts** view, but you can also create them directly on the **Component** view.

For more information related to the variable definition through the **Contexts** view, see section *How to use variables in the Contexts view*.

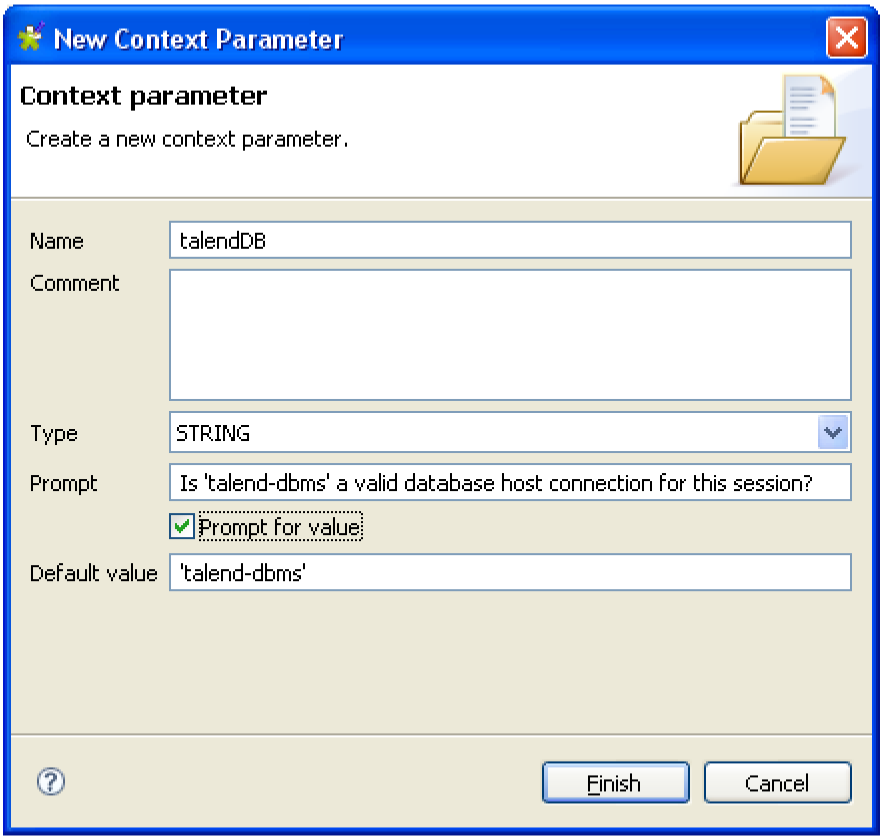
For more information regarding the variable definition in the repository, see section *How to store contexts in the repository*.

### Context variables creation

The quickest way to create context variables on the spot is to use the **F5** key:

1. On the relevant **Component** view, place your cursor on the field that you want to parameterize.

2. Press **F5** to display the context parameter dialog box:



New Context Parameter

Context parameter

Create a new context parameter.

Name: talendDB

Comment:

Type: STRING

Prompt: Is 'talend-dbms' a valid database host connection for this session?

☑ Prompt for value

Default value: 'talend-dbms'

3. Give a **Name** to this new variable, fill in the **Comment** area and choose the **Type**.

4. Enter a **Prompt** to be displayed to confirm the use of this variable in the current Job execution (generally used for test purpose only). And select the **Prompt for value** check box to display the field as editable value.

5. If you filled in a value already in the corresponding properties field, this value is displayed in the **Default value** field. Else, type in the default value you want to use for one context.

6. Click **Finish** to validate.

7. Go to the **Contexts** view tab. Notice that the context variables tab lists the newly created variables.

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

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

The variable created this way is automatically stored in all existing contexts, but you can subsequently change the value independently in each context.

For more information on how to create or edit a context, see section *How to configure contexts*.

## StoreSQLQuery

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

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

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

For further details on **StoreSQLQuery** settings, see *Talend Open Studio for Big Data Components Reference Guide*, and in particular the scenarios of the **tDBInput** component.

# 3.4.1.5. How to store contexts in the repository

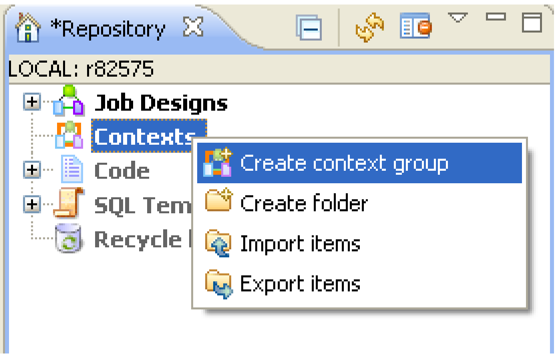
You can store centrally all contexts if you need to reuse them across various Jobs.

## How to create a context group

To create a context group, proceed as follows:

### Create the context group and add required information

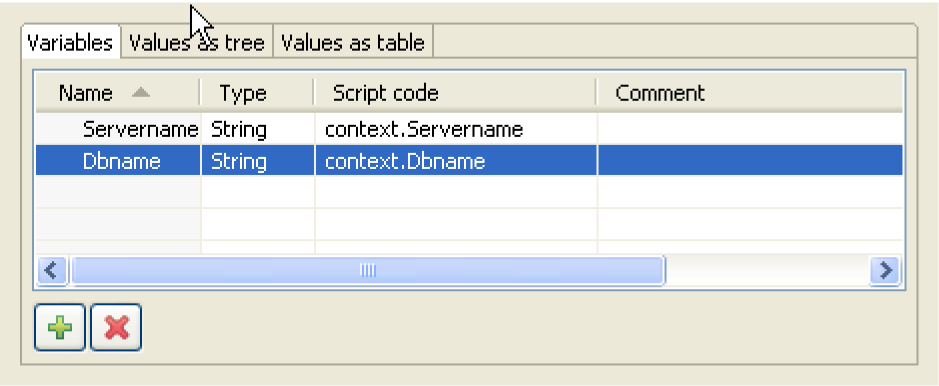
1. Right-click the **Contexts** node in the **Repository** tree view and select **Create new context group** from the contextual menu.



   A 2-step wizard appears to help you define the various contexts and context parameters, which you will be able to select in the **Contexts** view of the design workspace.

2. In Step 1 of 2, type in a name for the context group to be created, and add any general information such as a description if required.

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



| Name | Type | Script code | Comment |
|---|---|---|---|
| Servername | String | context.Servername | |
| Dbname | String | context.Dbname | |

## Define the default context’s variable set to be used as basis for other contexts

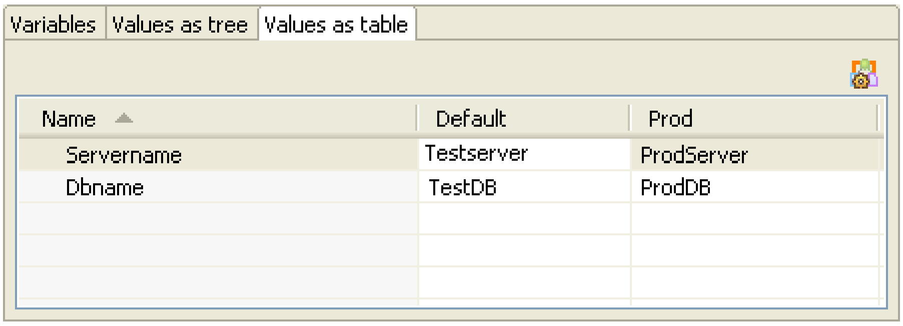
1. On the **Variables** tab, click the **[+]** button to add as many new variable lines as needed and define the name of the variables.

   In this example, we define the variables that can be used in the **Name** field of the **Component** view.

2. Select the type of the variable from the **Type** list.

   The **Script code** varies according to the type of variable you selected, and will be used in the generated code. The screen shot above shows the Java code produced.

3. On the **Tree** or **Table** views, define the various contexts and the values of the variables.

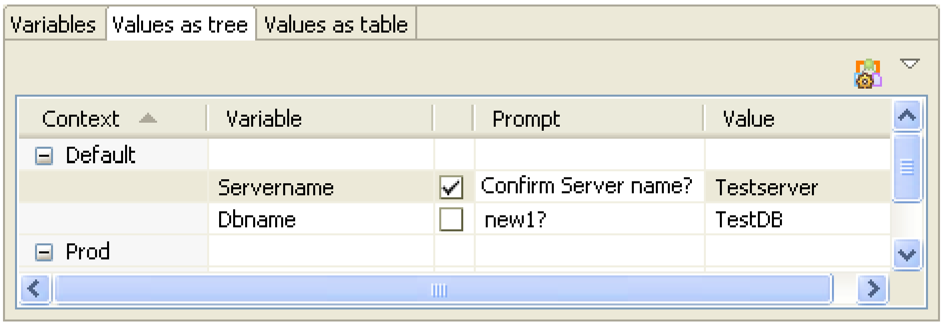


| Name | Default | Prod |
|---|---|---|
| Servername | Testserver | ProdServer |
| Dbname | TestDB | ProdDB |

   First, define the values for the default (first) context variables, then create a new context that will be based on the variables values that you just set.

   For more information about how to create a new context, see section *How to configure contexts*.

4. On the **Values as tree** tab, add a prompt if you want the variable to be editable in a confirmation dialog box at execution time.



| Context | Variable | | Prompt | Value |
|---|---|---|---|---|
| ⊟ Default | | | | |
| | Servername | ☑ | Confirm Server name? | Testserver |
| | Dbname | □ | new1? | TestDB |
| ⊟ Prod | | | | |

To add a prompt message, select the facing check box, then type in the message you want to display at execution time.

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.

## 3.4.1.6. How to apply context variables to a Job from the repository

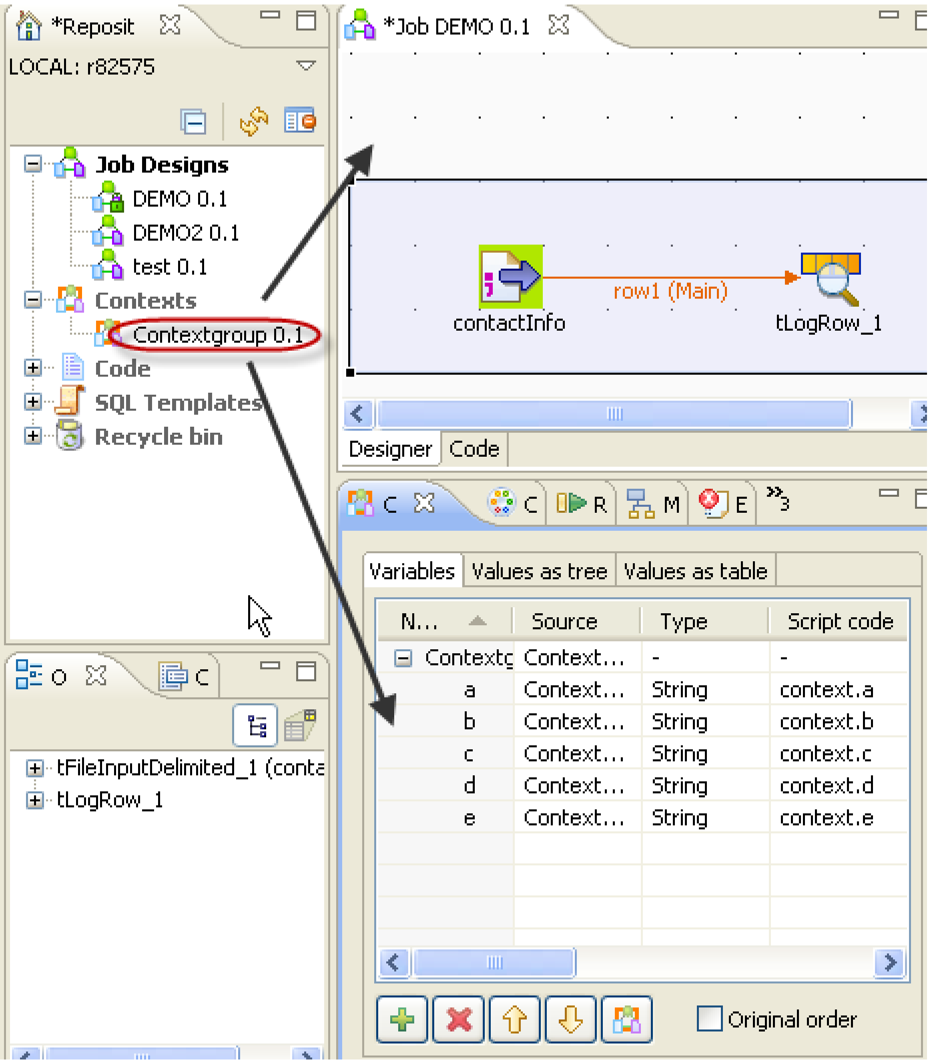
Once a context group is created and stored in the **Repository**, there are two ways of applying it to a Job:

1. Drop a context group. This way, the group is applied as a whole.
2. Use the context icon button . This way, the variables of a context group can be applied separately.

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

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

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



| N... | Source | Type | Script code |
|---|---|---|---|
| Contextg | Context... | - | - |
| a | Context... | String | context.a |
| b | Context... | String | context.b |
| c | Context... | String | context.c |
| d | Context... | String | context.d |
| e | Context... | String | context.e |

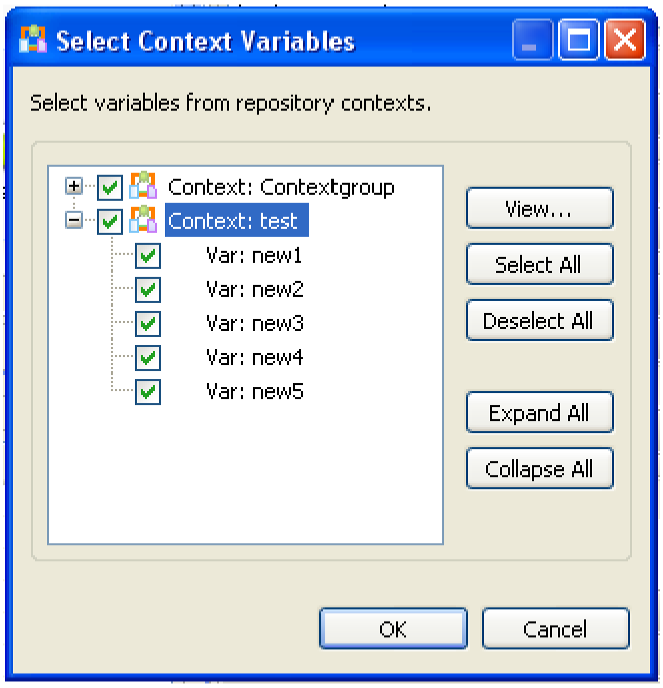
## How to use the context icon button

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

1. Double-click the Job to which a context variable is to be added.

2. Once the Job is opened in the workspace, click the **Contexts** view beneath the workspace to open it.

3. At the bottom of the **Contexts** view, click the button to open the wizard to select the context variables to be applied.



Select Context Variables

Select variables from repository contexts.

- ☑ Context: Contextgroup
- ☑ Context: test
  - ☑ Var: new1
  - ☑ Var: new2
  - ☑ Var: new3
  - ☑ Var: new4
  - ☑ Var: new5

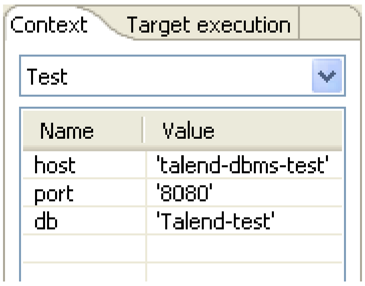
4. In the wizard, select the context variables you need to apply or clear those you do not need to.

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

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

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

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



Test

| Name | Value |
|---|---|
| host | 'talend-dbms-test' |
| port | '8080' |
| db | 'Talend-test' |

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

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

All the context variables you created for the selected context display, along with their respective value, in a table underneath. If you clear the **Prompt** check box next to some variables, you will get a dialog box allowing you to change the variable value for this Job execution only.

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

Related topics:

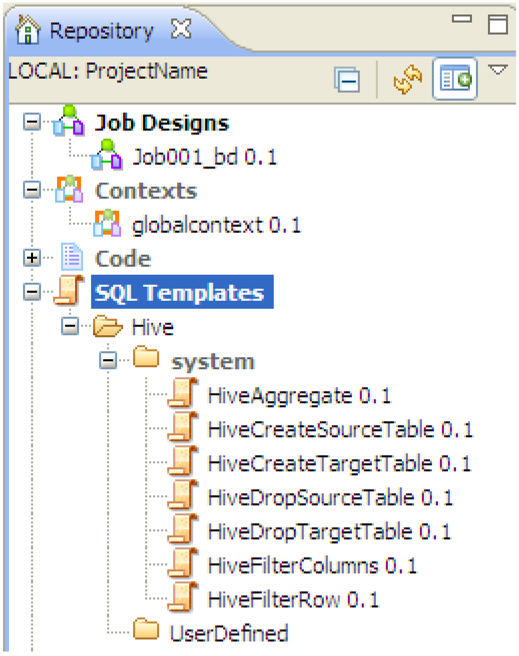
- section *How to use variables in the Contexts view*

- section *How to store contexts in the repository*

## 3.4.2. How to use the SQL Templates

*Talend Open Studio for Big Data* allows you to benefit from using some system SQL templates since many query structures are standardized with common approaches.

*Talend Open Studio for Big Data* lists system SQL templates under the **SQL Templates** node in the **Repository** tree view. There, you can find several standardized SQL templates for Hive.



In each of the above categories, you can create your own user-defined SQL templates using the SQL templates wizard and thus centralize them in the repository for reuse.

For more information about the use of SQL templates in *Talend Open Studio for Big Data*, see chapter *Designing a data integration Job*.

For more information about how to create a user-defined SQL template and use it in a Job context, see the scenario of the **tMysqlTableList** component in *Talend Open Studio for Big Data Components Reference Guide*.

# 3.5. 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.

## 3.5.1. How to map data flows

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

For more information about the principles of using this component, see chapter *Designing a data integration Job*.

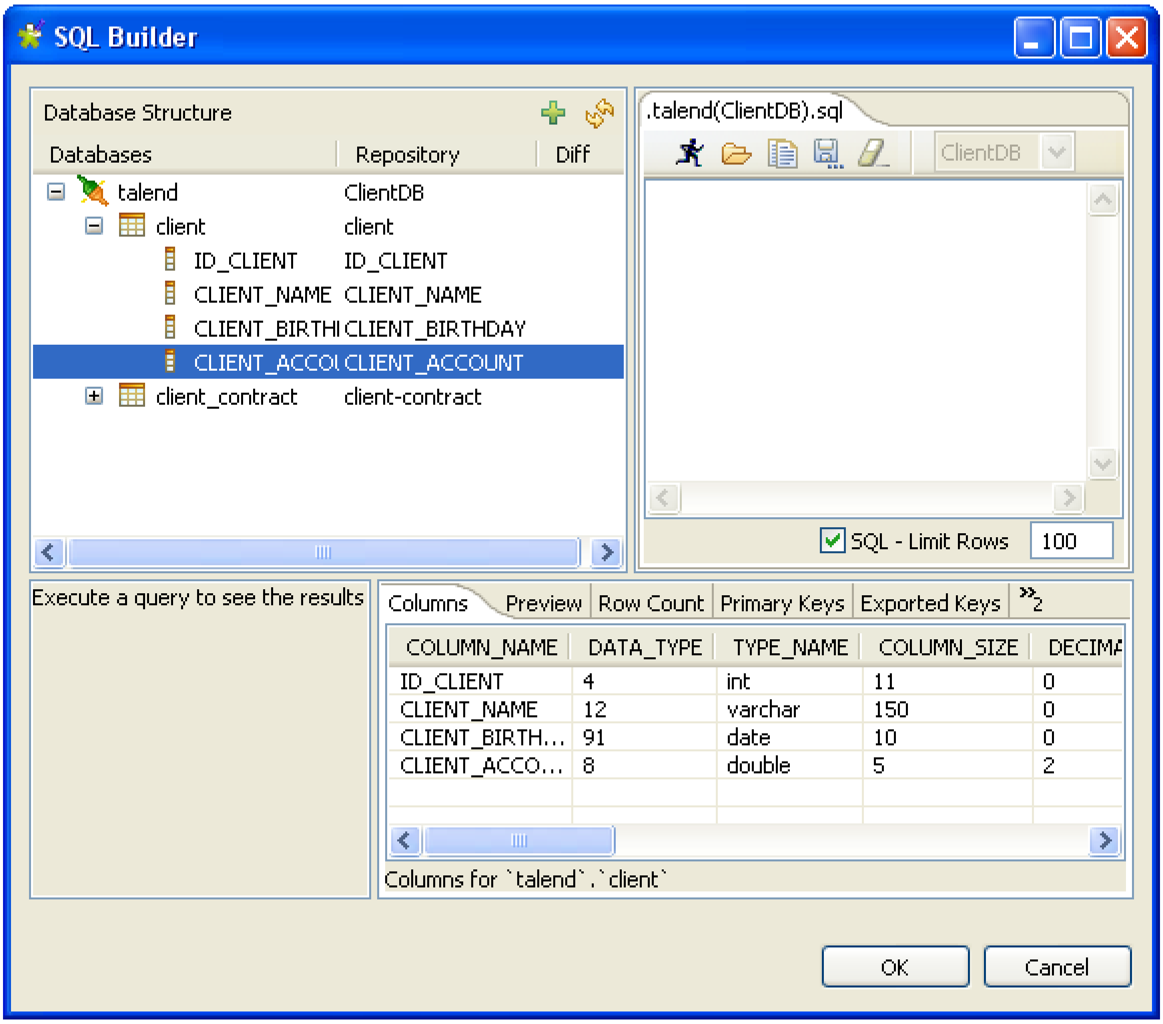
For examples of Jobs using this component, see **tMap** in *Talend Open Studio for Big Data Components Reference Guide*.

## 3.5.2. How to create queries using the SQLBuilder

SQLBuilder helps you build 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).

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.



| Databases | Repository | Diff |
|---|---|---|
| talend | ClientDB | |
| client | client | |
| ID_CLIENT | ID_CLIENT | |
| CLIENT_NAME | CLIENT_NAME | |
| CLIENT_BIRTHI | CLIENT_BIRTHDAY | |
| CLIENT_ACCOI | CLIENT_ACCOUNT | |
| client_contract | client-contract | |

| COLUMN_NAME | DATA_TYPE | TYPE_NAME | COLUMN_SIZE | DECIMA |
|---|---|---|---|---|
| ID_CLIENT | 4 | int | 11 | 0 |
| CLIENT_NAME | 12 | varchar | 150 | 0 |
| CLIENT_BIRTH... | 91 | date | 10 | 0 |
| CLIENT_ACCO... | 8 | double | 5 | 2 |

Columns for `talend`.`client`

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

- 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 in your connection.

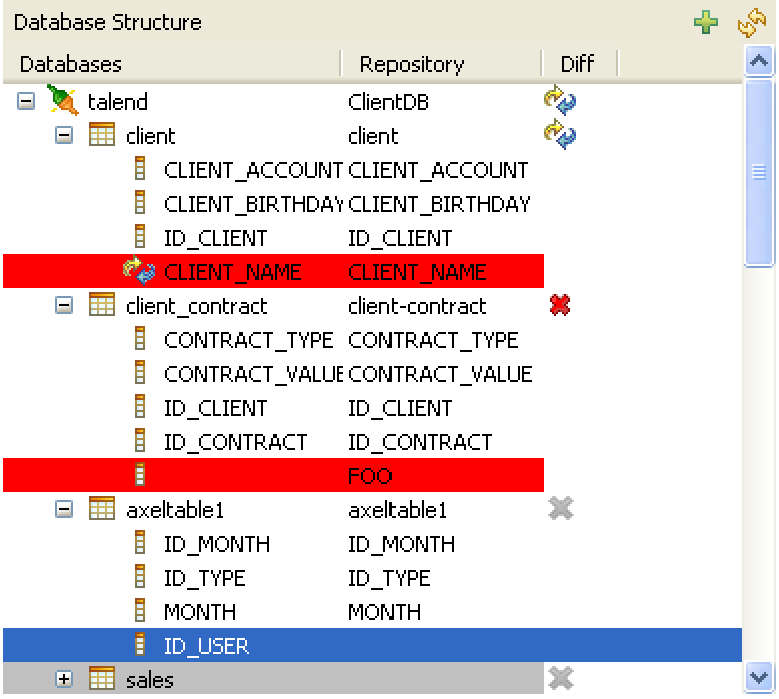
The schema view, in the bottom right corner of the editor, shows the column description.

### 3.5.2.1. How to compare database structures

On the **Database Structure** panel, you can see all the tables of the database.

The connection to the database might take quite some time.

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



Database Structure

| Databases | Repository | Diff |
|---|---|---|
| talend | ClientDB | ⟳ |
| client | client | ⟳ |
| CLIENT_ACCOUNT | CLIENT_ACCOUNT | |
| CLIENT_BIRTHDAY | CLIENT_BIRTHDAY | |
| ID_CLIENT | ID_CLIENT | |
| CLIENT_NAME | CLIENT_NAME | |
| client_contract | client-contract | ✖ |
| CONTRACT_TYPE | CONTRACT_TYPE | |
| CONTRACT_VALUE | CONTRACT_VALUE | |
| ID_CLIENT | ID_CLIENT | |
| ID_CONTRACT | ID_CONTRACT | |
| | FOO | |
| axeltable1 | axeltable1 | ✖ |
| ID_MONTH | ID_MONTH | |
| ID_TYPE | ID_TYPE | |
| MONTH | MONTH | |
| ID_USER | | |
| 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.

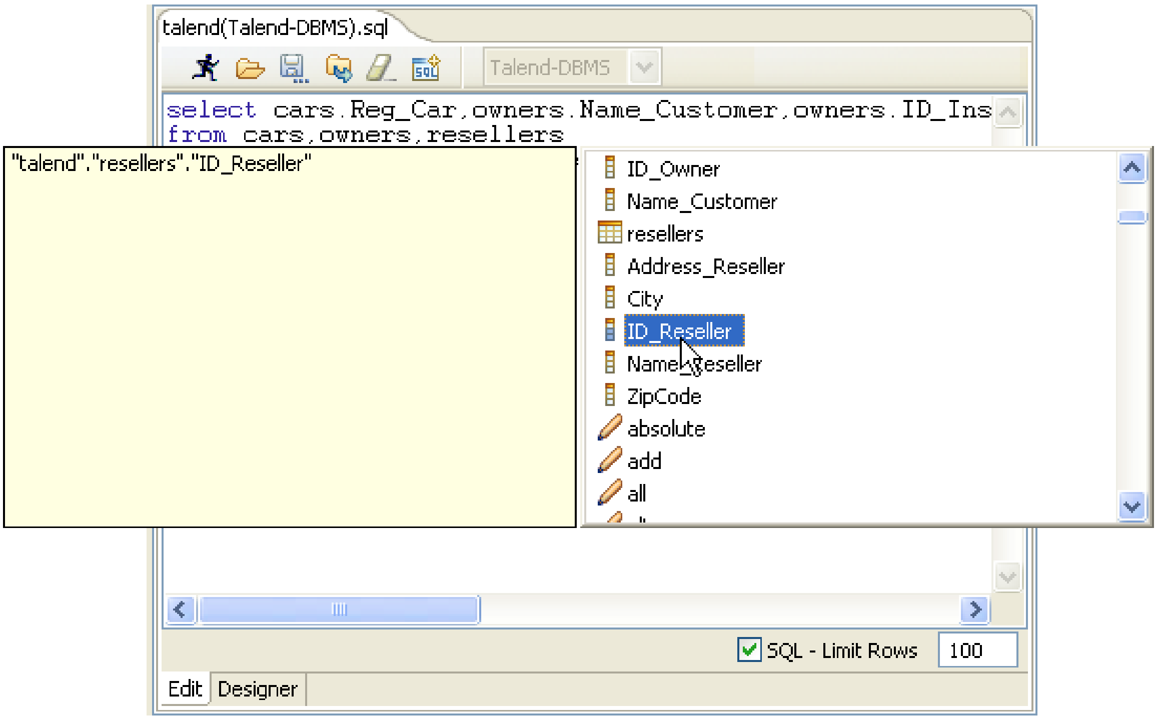
### 3.5.2.2. How to build a query

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

To create a new query, complete the following:

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

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



```
select cars.Reg_Car,owners.Name_Customer,owners.ID_Ins
from cars,owners,resellers
```

"talend"."resellers"."ID_Reseller"

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

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

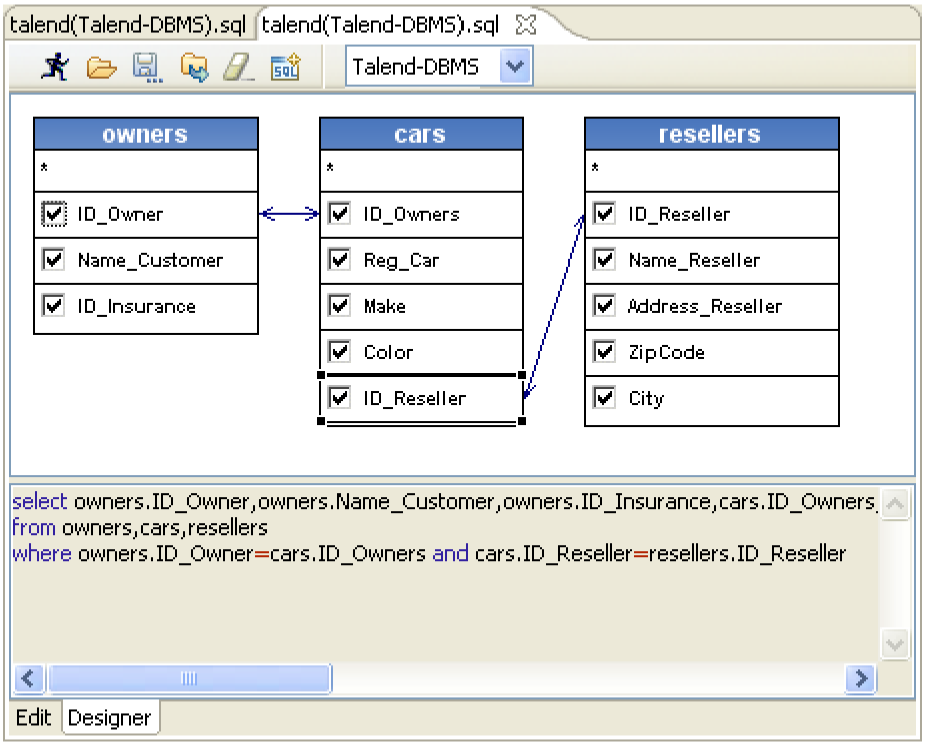
   You may get a message while switching from one view to the other as some SQL statements cannot be interpreted graphically.

4. If you selected a table, all columns are selected by default. Clear the check box facing the relevant columns to exclude them from the selection.

5. Add more tables in a simple right-click. On the **Designer** view, right-click and select **Add tables** in the pop-up list then select the relevant table to be added.

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

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



```
select owners.ID_Owner,owners.Name_Customer,owners.ID_Insurance,cars.ID_Owners,
from owners,cars,resellers
where owners.ID_Owner=cars.ID_Owners and cars.ID_Reseller=resellers.ID_Reseller
```

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

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

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

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

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

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

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

A click on the **Exchange** link on the toolbar of *Talend Open Studio for Big Data* 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 Open Studio for Big Data* but not installed yet in your current Studio, and
- components you have created and uploaded to **Talend Exchange** to share with other **Talend** Community users.

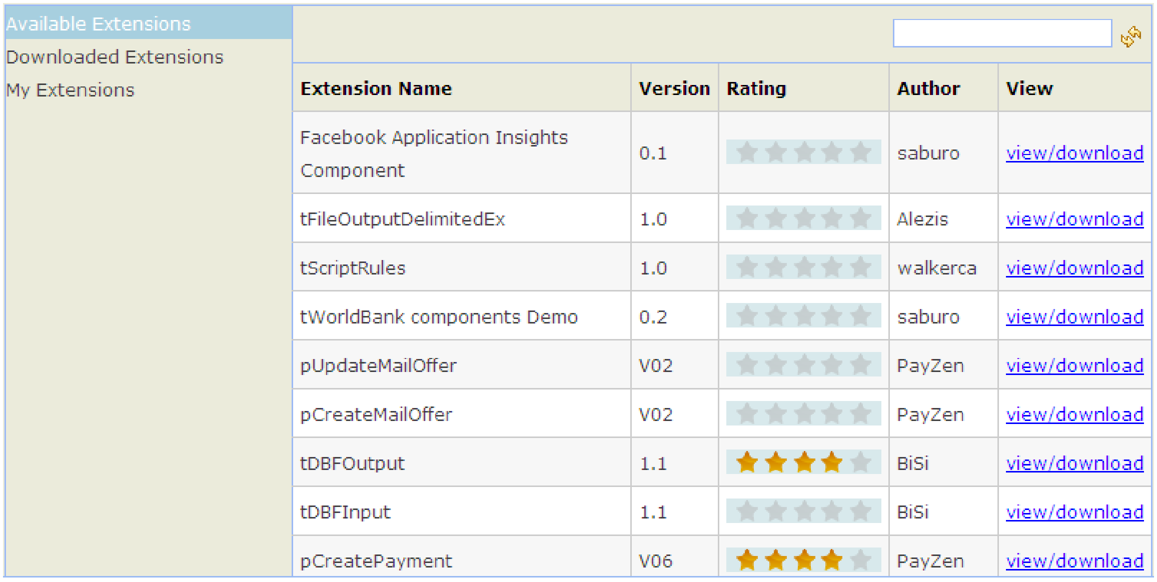
- Before you can download community components or upload your own components to the community, you need to sign in to **Talend Exchange** from your Studio first. If you did not sign in to **Talend Exchange** when launching the Studio, you still have a chance to sign in from the **Talend Exchange** preferences settings page. For more information, see section *Exchange preferences (Talend > Exchange)*.

- The community components available for download are not validated by **Talend**. This explains why you may encounter component loading errors sometimes when trying to install certain community components, why an installed community component may have a different name in the **Palette** than in the **Exchange** tab view, and why you may not be able to find a component in the **Palette** after it is seemingly installed successfully.

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

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

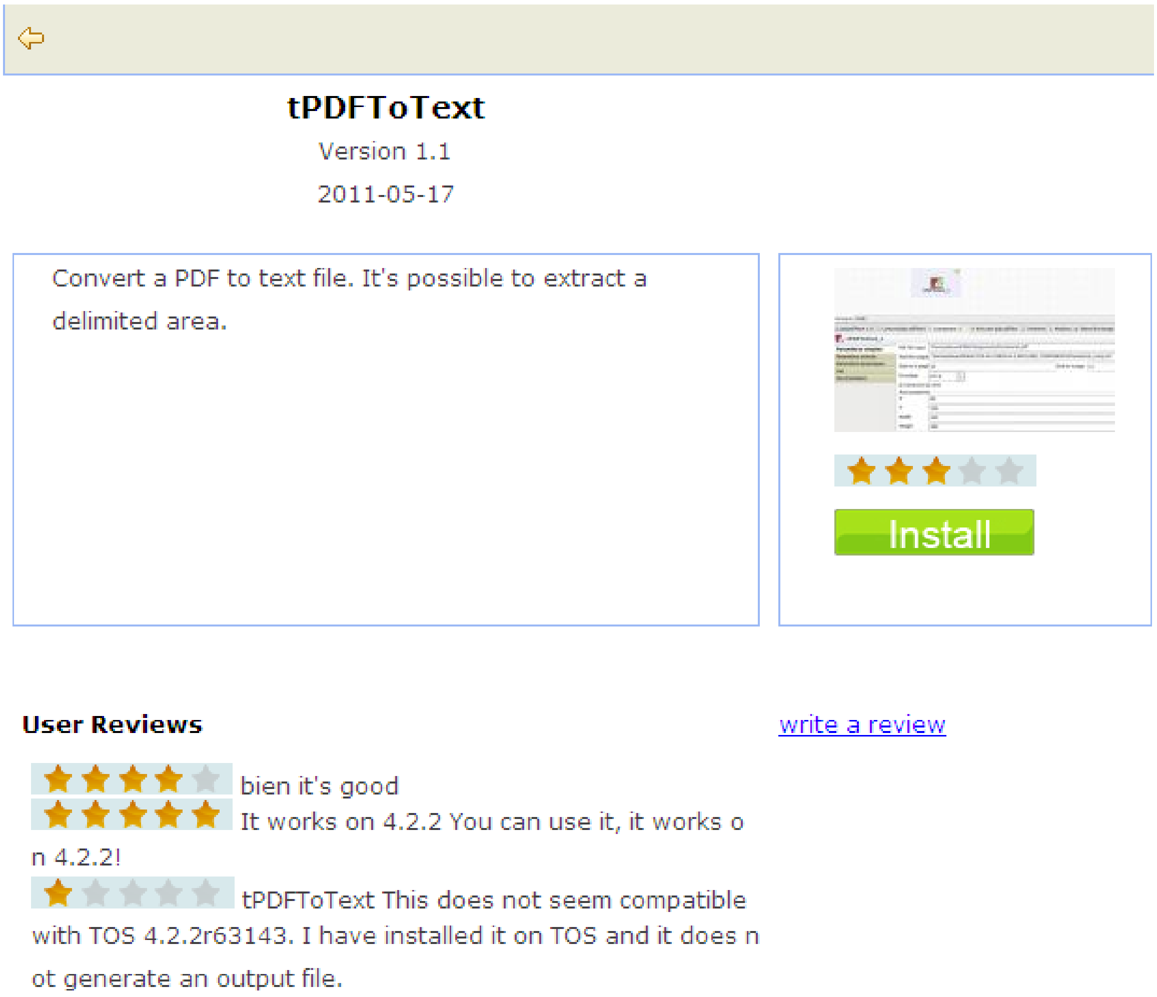
1. Click the **Exchange** link on the toolbar of *Talend Open Studio for Big Data* to open the **Exchange** tab view on the design workspace.



| Extension Name | Version | Rating | Author | View |
|---|---|---|---|---|
| Facebook Application Insights Component | 0.1 | ☆☆☆☆☆ | saburo | view/download |
| tFileOutputDelimitedEx | 1.0 | ☆☆☆☆☆ | Alezis | view/download |
| tScriptRules | 1.0 | ☆☆☆☆☆ | walkerca | view/download |
| tWorldBank components Demo | 0.2 | ☆☆☆☆☆ | saburo | view/download |
| pUpdateMailOffer | V02 | ☆☆☆☆☆ | PayZen | view/download |
| pCreateMailOffer | V02 | ☆☆☆☆☆ | PayZen | view/download |
| tDBFOutput | 1.1 | ★★★★☆ | BiSi | view/download |
| tDBFInput | 1.1 | ☆☆☆☆☆ | BiSi | view/download |
| pCreatePayment | V06 | ★★★★☆ | PayZen | view/download |

2. In the **Available Extensions** view, if needed, enter a full component name or part of it in the text field and click the fresh button to find quickly the component you are interested in.

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



tPDFToText

Version 1.1

2011-05-17

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

★★★☆☆

User Reviews

write a review

★★★★☆ bien it's good

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

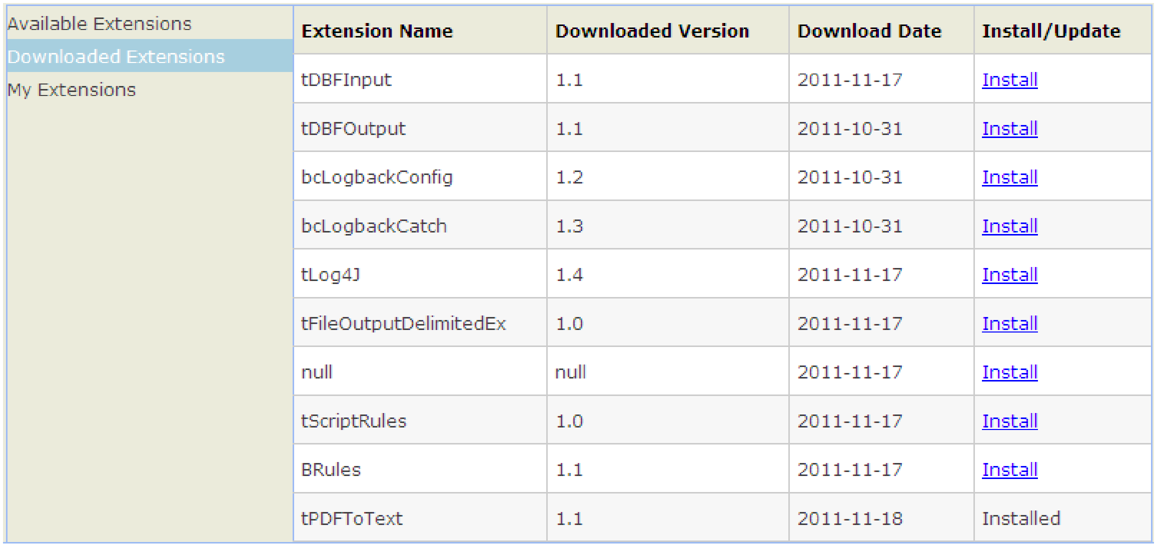
★☆☆☆☆ tPDFToText This does not seem compatible with TOS 4.2.2r63143. I have installed it on TOS and it does not generate an output file.

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

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

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

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



| Extension Name | Downloaded Version | Download Date | Install/Update |
|---|---|---|---|
| tDBFInput | 1.1 | 2011-11-17 | Install |
| tDBFOutput | 1.1 | 2011-10-31 | Install |
| bcLogbackConfig | 1.2 | 2011-10-31 | Install |
| bcLogbackCatch | 1.3 | 2011-10-31 | Install |
| tLog4J | 1.4 | 2011-11-17 | Install |
| tFileOutputDelimitedEx | 1.0 | 2011-11-17 | Install |
| null | null | 2011-11-17 | Install |
| tScriptRules | 1.0 | 2011-11-17 | Install |
| BRules | 1.1 | 2011-11-17 | Install |
| tPDFToText | 1.1 | 2011-11-18 | Installed |

## 3.5.3.2. How to reinstall or update community components

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

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

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

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

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

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

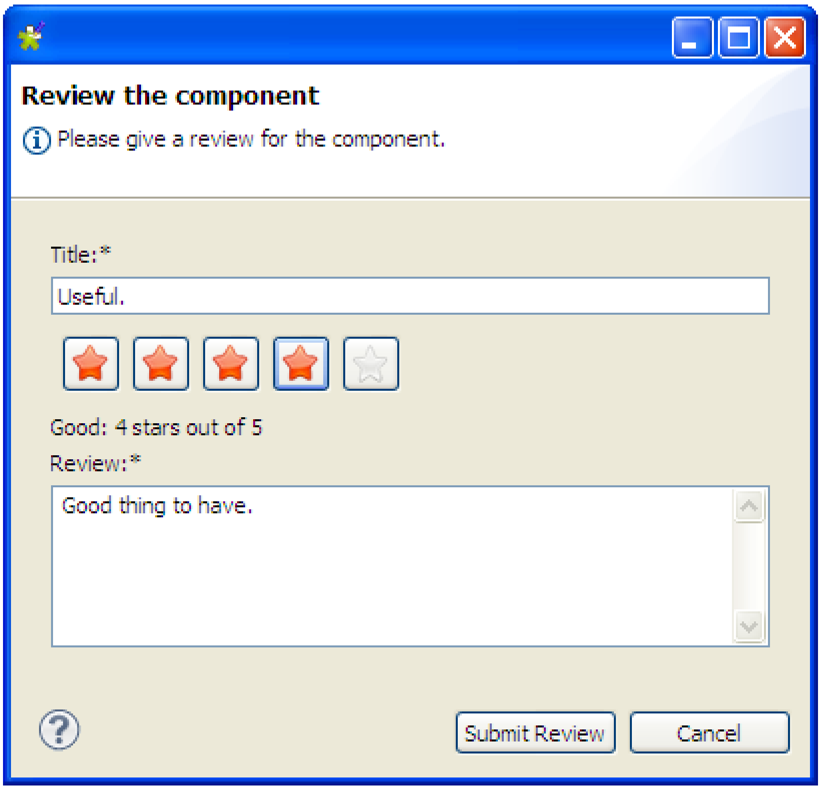
   A progress indicator appears to show the completion percentage of the installation process. Upon successful installation, the **Downloaded Extensions** view displays the status of the component, which is **Installed**.

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

To review and rate a community component:

1. From the **Available Extensions** view, click the **view/download** link for the component you want to review or rate to open the community component download page.

2. On the component download page, click the **write a review** link to open the **[Review the component]** dialog box.



Review the component

Please give a review for the component.

Title:* Useful.

Good: 4 stars out of 5

Review:* Good thing to have.

3. 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 you review to the **Talend Exchange** server.

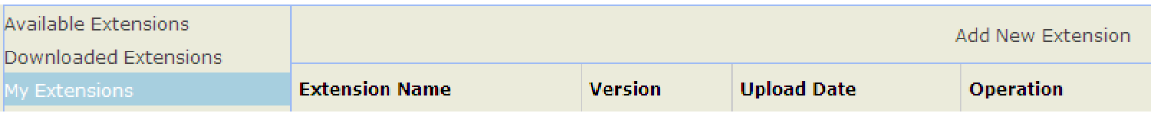
   Upon validation by the **Talend Exchange** moderator, your review is published on **Talend Exchange** and displayed in the **User Review** area of the component download page.

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

You can create your own components for use in your Jobs in *Talend Open Studio for Big Data* 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 Open Studio for Big Data*, see section *External or User components (Talend > Components)*.

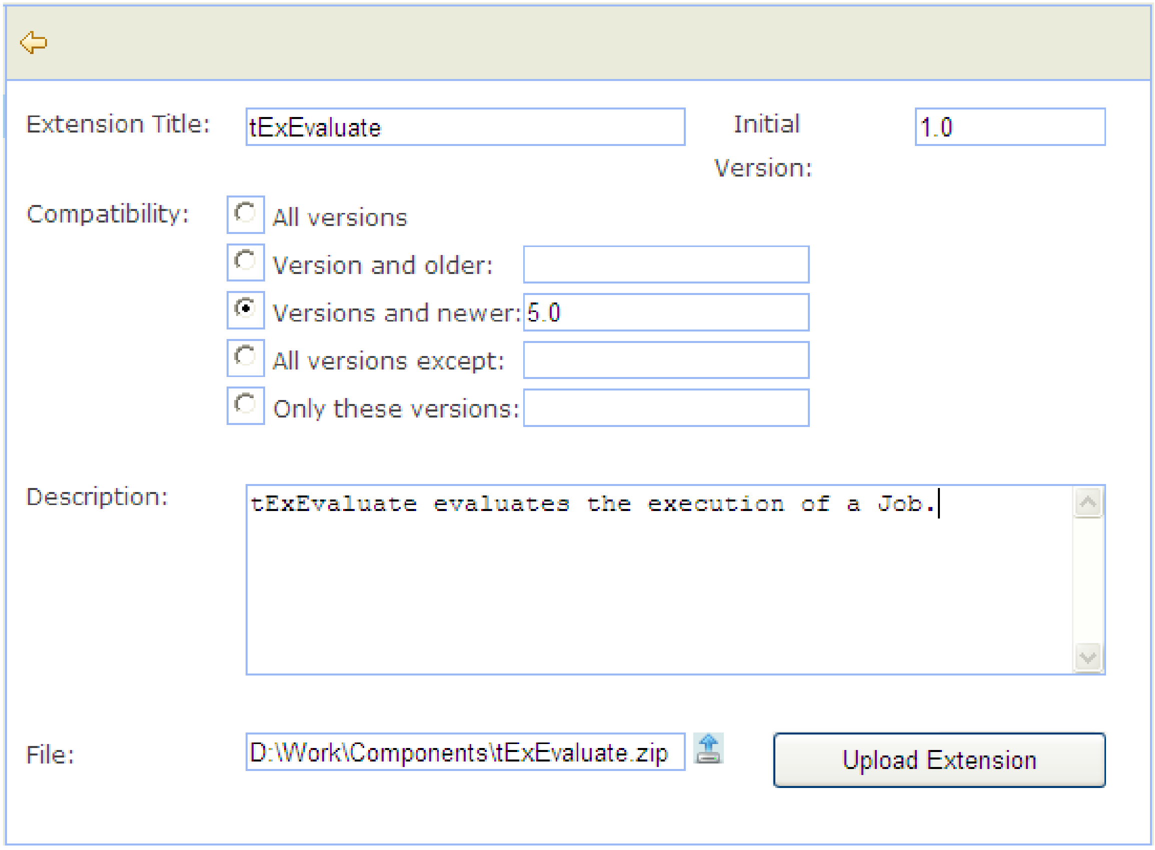
To upload a component you created to **Talend Exchange**, complete the following:

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



| Extension Name | Version | Upload Date | Operation |
|---|---|---|---|

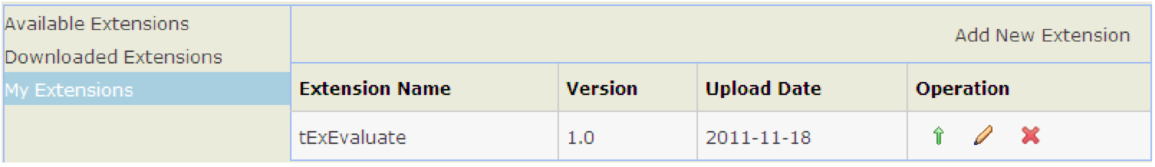
2. Click the **Add New Extension** link in the upper right part of the view to open the component upload page.



|  |  |  |  |
|---|---|---|---|
| Extension Title: | tExEvaluate | Initial Version: | 1.0 |
| Compatibility: | ○ All versions<br>○ Version and older:<br>● Versions and newer: 5.0<br>○ All versions except:<br>○ Only these versions: |  |  |
| Description: | tExEvaluate evaluates the execution of a Job. |  |  |
| File: | D:\Work\Components\tExEvaluate.zip |  |  |

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**.



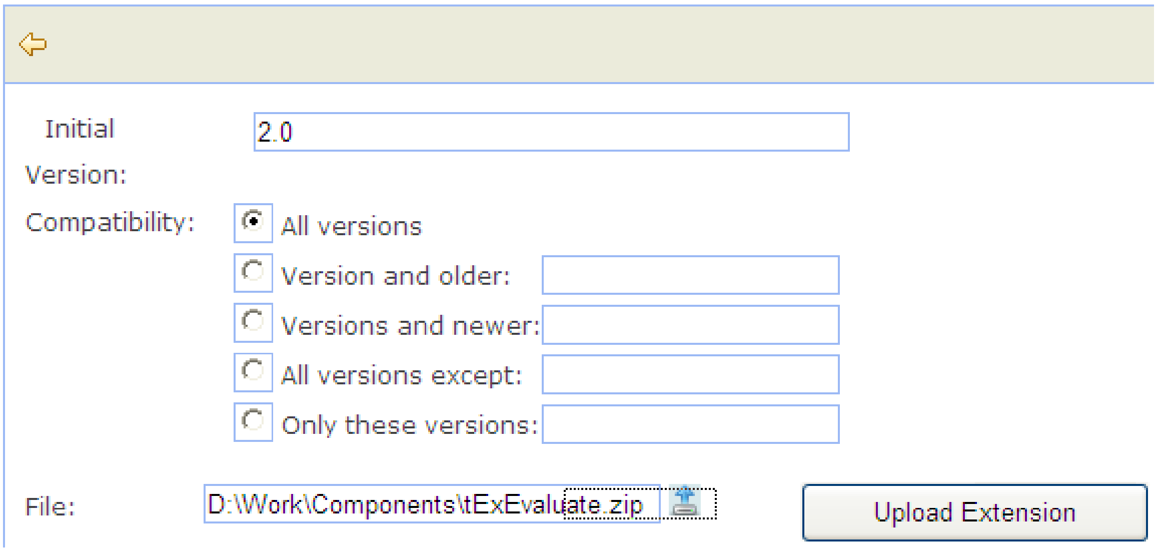
| Extension Name | Version | Upload Date | Operation |
|---|---|---|---|
| tExEvaluate | 1.0 | 2011-11-18 | ⇧ ✎ ✖ |

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

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

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

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



Initial Version: 2.0

Compatibility:
● All versions
○ Version and older:
○ Versions and newer:
○ All versions except:
○ Only these versions:

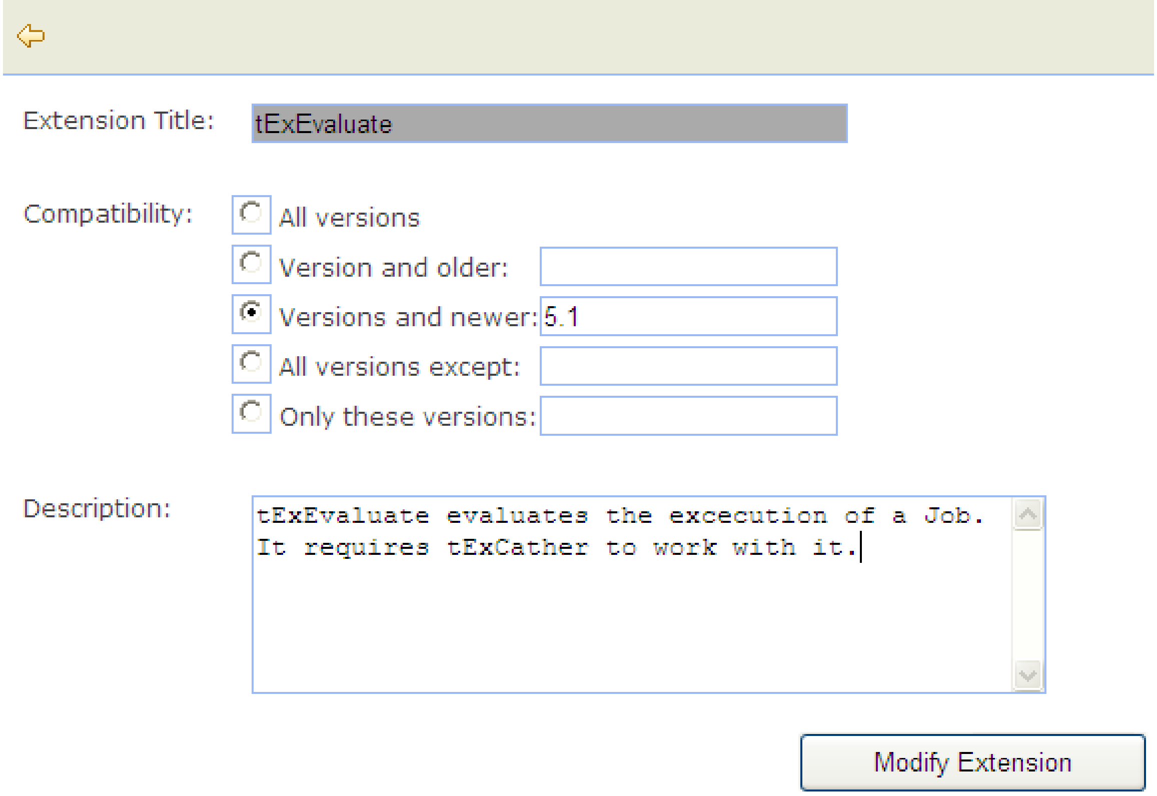
File: D:\Work\Components\tExEvaluate.zip

2. 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:

1. From the **My Extensions** view, click the ✎ icon in the **Operation** column for the component your 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 excecution of a Job. It requires tExCather to work with it.

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.

## 3.5.4. How to install external modules

*Talend Open Studio for Big Data* requires specific third-party Java libraries or database drivers to be installed to connect to sources and targets. Those libraries or drivers, known as external modules, can be required by some of **Talend** components. Due to license restrictions, **Talend** may not be able to ship certain external modules within *Talend Open Studio for Big Data*.

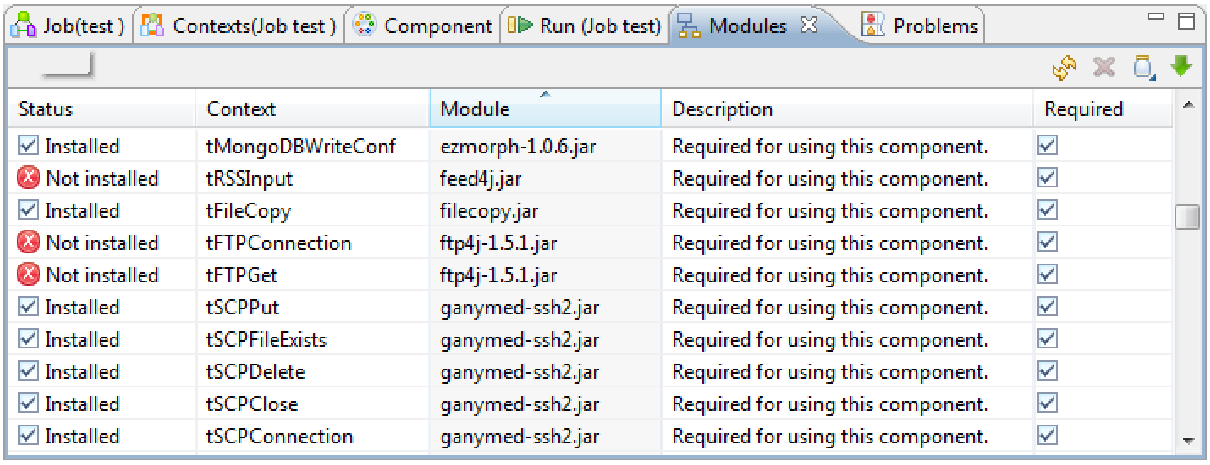
### 3.5.4.1. Identifying required external modules

On your design workspace, if a component requires the installation of an external module before it can work properly, a red error indicator appears on the component. With your mouse pointer over the error indicator, you can see a tooltip message showing which external module is required. For more information on handling error icons, see section *How to handle error icons on components or Jobs*.

The **Modules** view lists all the modules required to use the components embedded in the Studio, including those missing Java libraries and drivers that you must install to get the relevant components working.

If the **Modules** tab does not show on the tab area of your design workspace, go to **Window** > **Show View...** > **Talend** and then select **Modules** from the list.

To access the **Modules** view, click the **Modules** tab in the design workspace.



| Status | Context | Module | Description | Required |
|---|---|---|---|---|
| ☑ Installed | tMongoDBWriteConf | ezmorph-1.0.6.jar | Required for using this component. | ☑ |
| ✖ Not installed | tRSSInput | feed4j.jar | Required for using this component. | ☑ |
| ☑ Installed | tFileCopy | filecopy.jar | Required for using this component. | ☑ |
| ✖ Not installed | tFTPConnection | ftp4j-1.5.1.jar | Required for using this component. | ☑ |
| ✖ Not installed | tFTPGet | ftp4j-1.5.1.jar | Required for using this component. | ☑ |
| ☑ Installed | tSCPPut | ganymed-ssh2.jar | Required for using this component. | ☑ |
| ☑ Installed | tSCPFileExists | ganymed-ssh2.jar | Required for using this component. | ☑ |
| ☑ Installed | tSCPDelete | ganymed-ssh2.jar | Required for using this component. | ☑ |
| ☑ Installed | tSCPClose | ganymed-ssh2.jar | Required for using this component. | ☑ |
| ☑ Installed | tSCPConnection | ganymed-ssh2.jar | Required for using this component. | ☑ |

The table below describes the information presented in the **Modules** view.

| Column | Description |
|---|---|
| **Status** | points out if a module is installed or not installed on your system.<br>The ⚠ icon indicates that the module is not necessarily required for the corresponding component listed in the **Context** column. The ✖ icon indicates that the module is absolutely required for the corresponding component. |
| **Context** | lists the name of **Talend** component using the module. If this column is empty, the module is then required for the general use of *Talend Open Studio for Big Data*.<br>This column lists any external libraries added to the routines you create and save in the Studio library folder. For more information, see section *How to edit user routine libraries*. |

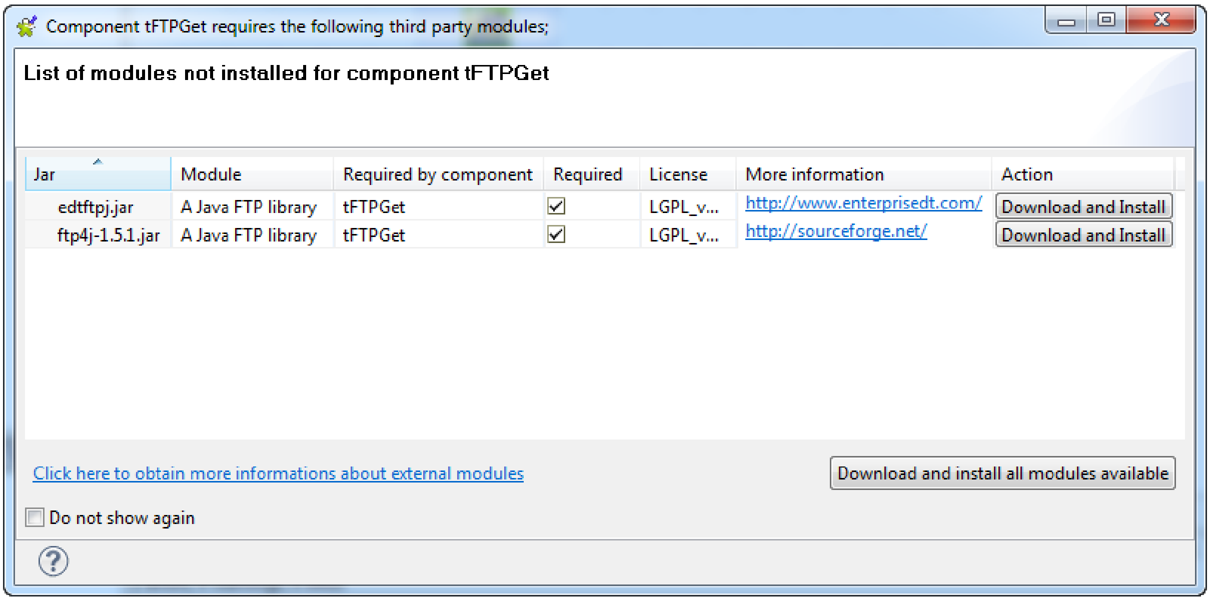
| Column | Description |
|---|---|
| **Module** | lists the module exact name. |
| **Description** | explains why the module/library is required. |
| **Required** | the selected check box indicates that the module is required. |

In addition to the **Modules** view, *Talend Open Studio for Big Data* provides a mechanism that enables you to easily identify, download and install most of the required third-party modules from **Talend** website and directs you to valid websites for the rest.

A Jar installation wizard appears when you:

- drop a component from the **Palette** if one or more external modules required for that component to work are missing in *Talend Open Studio for Big Data*, or
- click the **Guess schema** button in the **Component** view of a component if one or more external modules required for that component to work are missing in *Talend Open Studio for Big Data*, or
- click the ⬇ button in the **Modules** tab view.

When you click this button, the wizard that appears will list all the required external modules that are not integrated in *Talend Open Studio for Big Data*.



Component tFTPGet requires the following third party modules;

List of modules not installed for component tFTPGet

| Jar | Module | Required by component | Required | License | More information | Action |
|---|---|---|---|---|---|---|
| edtftpj.jar | A Java FTP library | tFTPGet | ☑ | LGPL_v... | http://www.enterprisedt.com/ | Download and Install |
| ftp4j-1.5.1.jar | A Java FTP library | tFTPGet | ☑ | LGPL_v... | http://sourceforge.net/ | Download and Install |

Click here to obtain more informations about external modules

Download and install all modules available

□ Do not show again

The table below describes the information presented in the wizard.

| Item | Description |
|---|---|
| **Jar** | The file name of the external module. |
| **Module** | A short description about the nature of the module. |
| **Required by component** | Lists the components that require the external module. |
| **Required** | The selected check box indicates that the module is required. |
| **License** | The license under which the module is provided. |
| **More information** | Provides the URL of the valid website where you can find more information about this module and download the module manually. |
| **Action** | Presents a **Download and Install** button if the module is available on **Talend** website, click to start downloading and installing the module; or a link to direct you to the valid website to download the module manually if the module is not available on **Talend** website. |
| **Download and install all modules available** | Click to download and install all the required modules that are available on **Talend** website. |

| Item | Description |
|---|---|
| **Do not show again** | Select to prevent the wizard from appearing again unless you click the ⬇ button in the **Modules** tab view. This check box shows only when you drop a component, or guess the schema of a database, that requires a missing external module. |
| ? | Click to go to **Talend** online documentation on installing third-party modules. |

When you drop a component, or guess the schema of a database, that requires an external module, if neither the Jar file nor its download URL information is available on **Talend** website, the Jar installation wizard does not appear, but the **Error Log** view will present error information informing you that the download URL for that module is not available, so that you can try to find and download it by yourself.

For more information about the **Error Log** view, see section *Configuration tabs* .

### 3.5.4.2. Installing external modules

To download and install missing modules automatically, do the following:

1. In the Jar installation wizard, click the **Download and Install** button to install a particular module, or click the **Download and install all modules available** button to install all the available missing modules.

2. Click **Accept** in the **[License]** dialog box that appears to continue the installation.

   > The **[License]** dialog box appears for each license under which the relevant modules are provided until that license is accepted.

Upon installation of the chosen external module or modules, a dialog box appears to notify you about the number of modules successfully installed and/or about the modules failed to install, if any.

To install manually a missing external module that you have already downloaded, do the following:

1. In the **Modules** view, click the 📄 icon in the upper right corner of the view to browse your local file system.

2. In the **[Open]** dialog box, browse to the module you want to install.

   > ⚠ *For Oracle9i, the required JDBC driver downloadable from Oracle website is named ojdbc14.jar, the same as that for Oracle 10g. To make the JDBC driver for Oracle9i you have downloaded work in Talend Open Studio for Big Data, you have to change the file name to ojdbc14-9i.jar before installing it into the Studio.*

3. Double-click Jar file, or select it and then click **Open** to install it.

   The dialog box closes and the selected module is installed in the library folder of the current Studio.

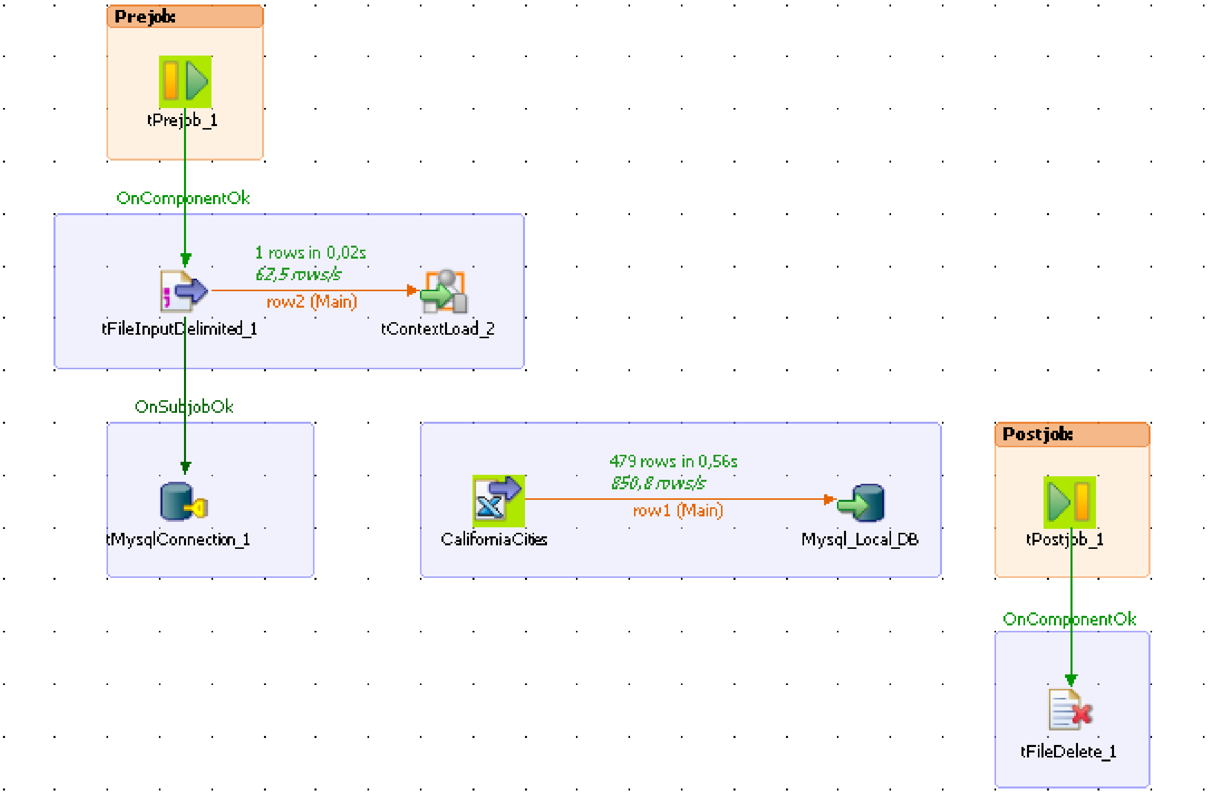
   You can now use the component dependent on this module in any of your Job designs.

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

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

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

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



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

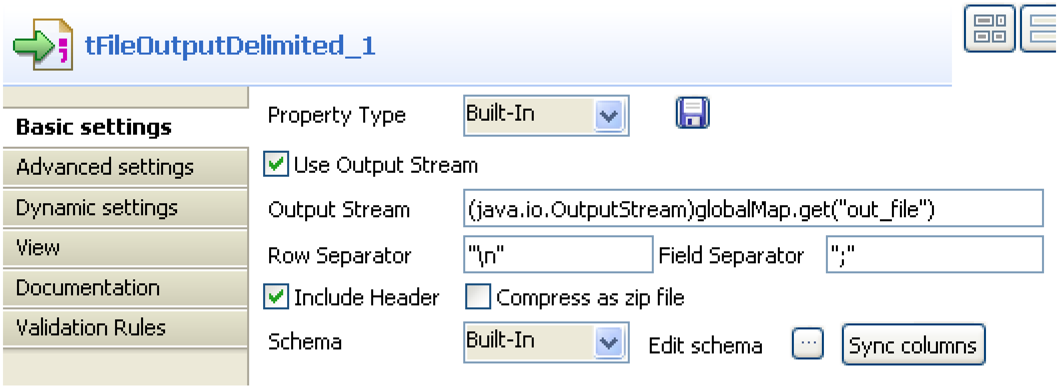
- Loading context information required for the subjob execution.
- Opening a database connection.
- Making sure that a file exists.

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

- Cleaning up temporary files created during the processing of the main data Job.
- Closing a database connection or a connection to an external service.
- Any task required to be executed, even if the preceding Job or subjobs failed.

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

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



Property Type: Built-In

☑ Use Output Stream

Output Stream: (java.io.OutputStream)globalMap.get("out_file")

Row Separator: "\n" Field Separator: ";"

☑ Include Header □ Compress as zip file

Schema: Built-In

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

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

Prior to use 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 section *Using the output stream feature*. For an example of Job using this feature, see the second scenario of **tFileOutputDelimited** in *Talend Open Studio for Big Data Components Reference Guide*.

# 3.6. Handling Jobs: miscellaneous subjects

The sections below give detail information about various subjects related to the management of a data integration Job including defining the start component, handling errors, and searching for jobs that use specific components.

## 3.6.1. How to share a database connection

If you have various Jobs using the same database connection, you can factorize the connection by using the **Use or Register a shared connection** option.

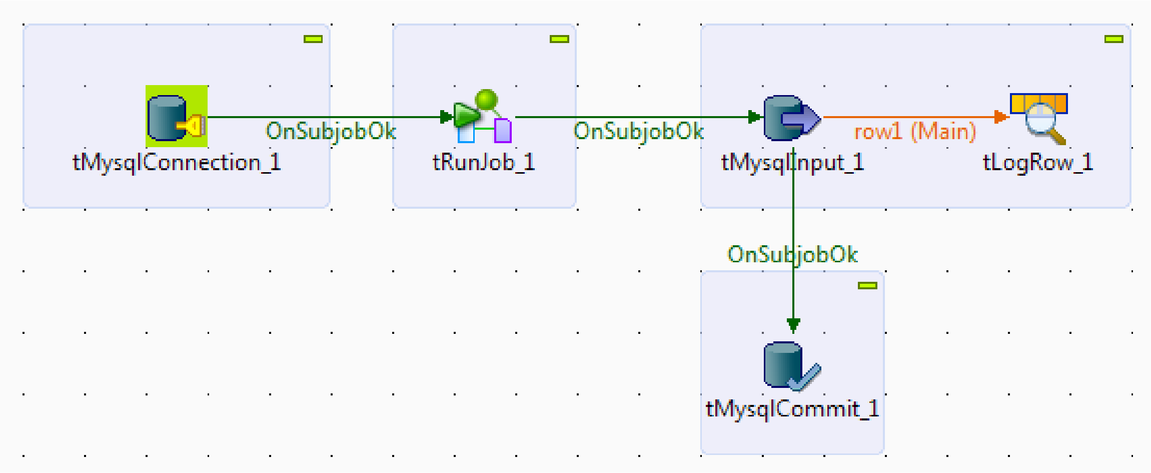
This option has been added to all database connection components in order to reduce the number of connections to open and close.

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

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

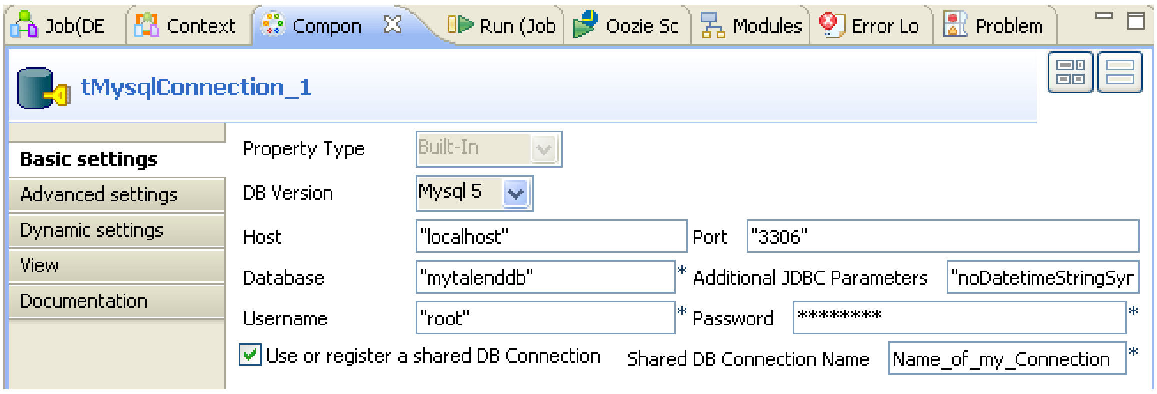
1. Drag and drop a **tMysqlConnection** (assuming that you work with a MySQL database)

2. Connect it to the first component of your parent Job using a **Trigger** > **On Subjob Ok** link.



3. In the **Basic settings** view of the **tMysqlConnection** component, fill in the database connection details..

4. Select the **Use or Register a shared connection** check box, and give a name to the connection in the **Shared DB Connection Name** field.



Property Type: Built-In

DB Version: Mysql 5

Host: "localhost" Port: "3306"

Database: "mytalenddb" * Additional JDBC Parameters: "noDatetimeStringSyr

Username: "root" * Password: ********* *

☑ Use or register a shared DB Connection  Shared DB Connection Name: Name_of_my_Connection *

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

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

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

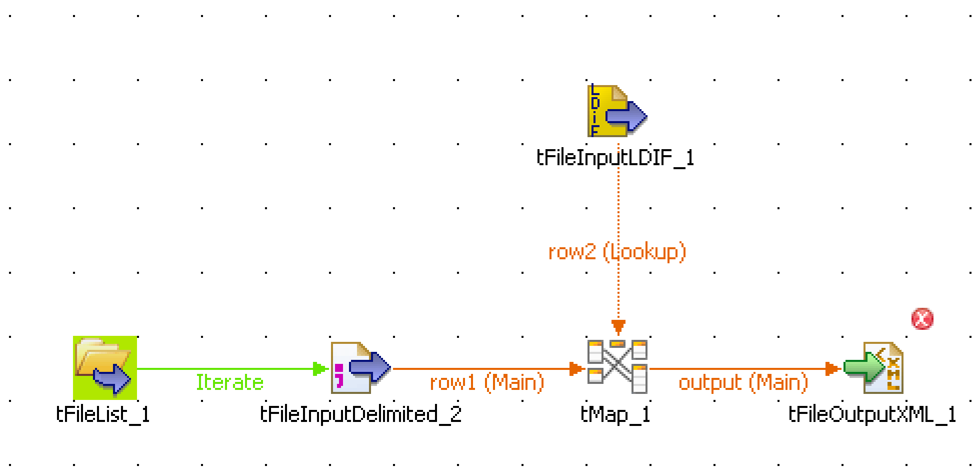
For more information about how to use the Connection components, see *Talend Open Studio for Big Data Components Reference Guide*.

## 3.6.2. How to define the Start component

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

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

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



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

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

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

Related topics:

- section *How to connect components together*
- section *Activating/Deactivating a Job or a sub-job*

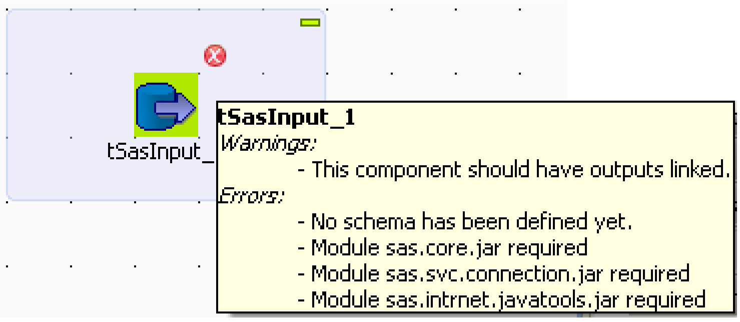
## 3.6.3. How to handle error icons on components or Jobs

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

### 3.6.3.1. Warnings and error icons on components

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

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

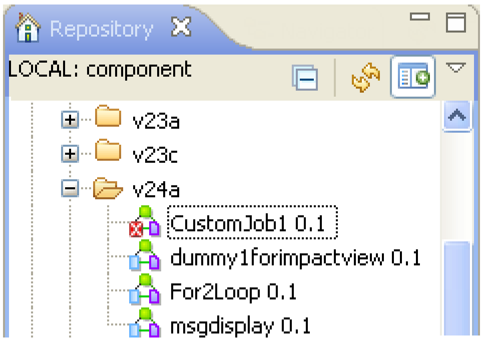


tSasInput_1
*Warnings:*
- This component should have outputs linked.
*Errors:*
- No schema has been defined yet.
- Module sas.core.jar required
- Module sas.svc.connection.jar required
- Module sas.intrnet.javatools.jar required

When the tooltip messages of a component indicate that a module is required, you must install this module for this component using the **Module** view. This view is hiden by default. For further information about how to install external modules using this view, see section *How to install external modules*.

## 3.6.3.2. Error icons on Jobs

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



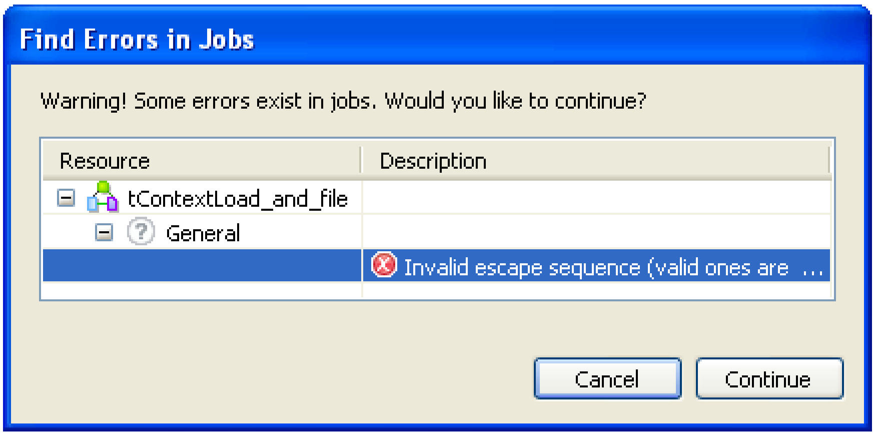
The error icon displays as well on the tab next to the Job name when you open the Job on the design workspace.

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

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

Hence, the red error icon will only show then.

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



Find Errors in Jobs

Warning! Some errors exist in jobs. Would you like to continue?

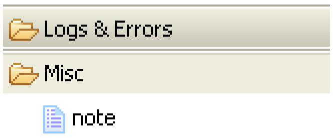
| Resource | Description |
|---|---|
| ⊟ tContextLoad_and_file | |
| ⊟ General | |
| | Invalid escape sequence (valid ones are ... |

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

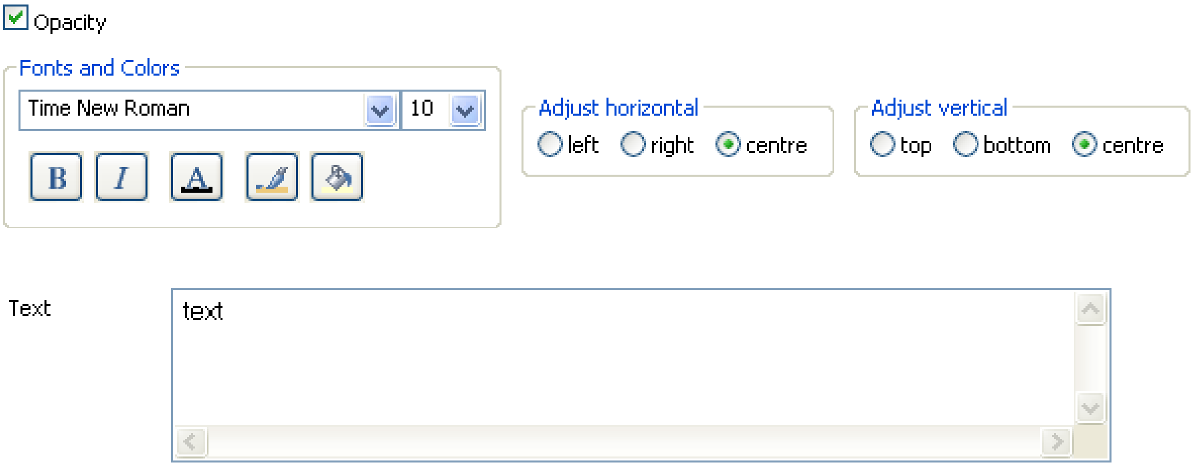
For information on errors on components, see section *Warnings and error icons on components*.

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

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



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



☑ Opacity

Fonts and Colors: Time New Roman, 10

Adjust horizontal: ○ left ○ right ● centre

Adjust vertical: ○ top ○ bottom ● centre

Text: text

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.

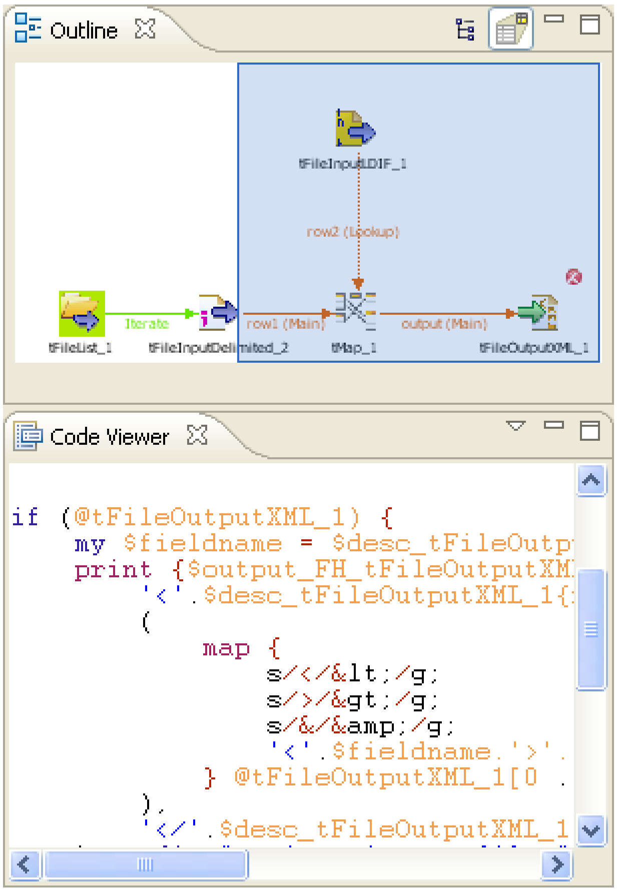
## 3.6.5. How to display the code or the outline of your Job

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

The Information panel is composed of two tabs, **Outline** and **Code Viewer**, which provide information regarding the displayed diagram.

### 3.6.5.1. Outline

The **Outline** tab offers a quick view of the open Job on the design workspace and also a tree view of all used elements in the Job. As the design workspace, like any other window area can be resized upon your needs, the **Outline** view is convenient to check out where about on your design workspace, you are located.



```
if (@tFileOutputXML_1) {
    my $fieldname = $desc_tFileOutp
    print {$output_FH_tFileOutputXM
        '<'.$desc_tFileOutputXML_1{
        (
            map {
                s/</&lt;/g;
                s/>/&gt;/g;
                s/&/&amp;/g;
                '<'.$fieldname.'>'.
            } @tFileOutputXML_1[0 .
        ),
        '</'.$desc_tFileOutputXML_1
```

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.

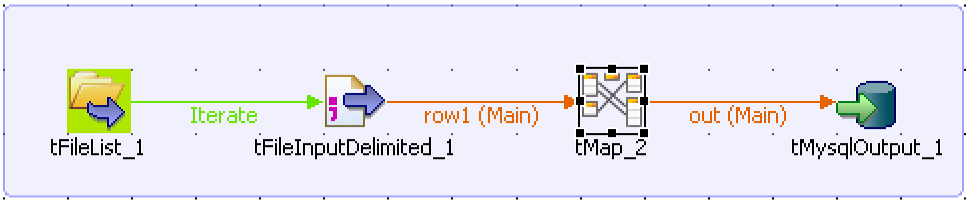
### 3.6.5.2. Code viewer

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

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.

## 3.6.6. How to manage the subjob display

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



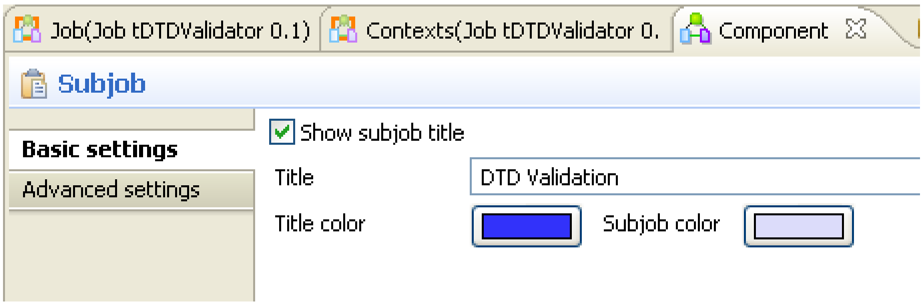
This blue highlight helps you easily distinguish one subjob from another.

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

For more information about prejob and postjob, see section *How to use the tPrejob and tPostjob components*.

### 3.6.6.1. How to format subjobs

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



☑ Show subjob title

Title: DTD Validation

Title color

Subjob color

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

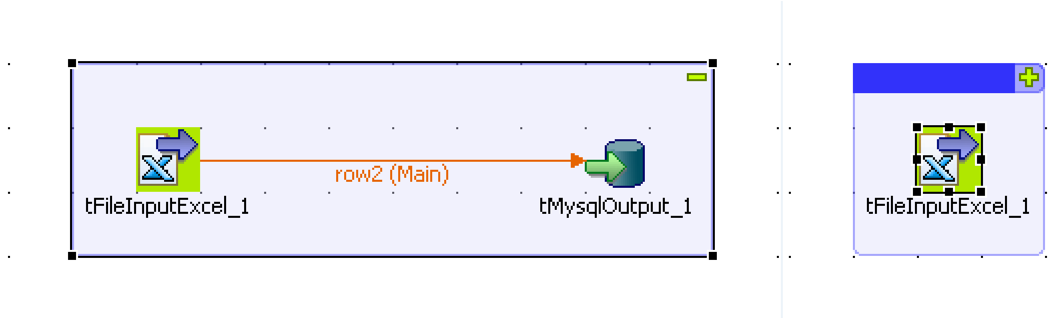
To modify the title color and the subjob color:

1. In the **Basic settings** view, click the **Title color/Subjob color** button to display the **[Colors]** dialog box.

2. Set your colors as desired. By default, the title color is blue and the subjob color is transparent blue.

### 3.6.6.2. How to collapse the subjobs

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



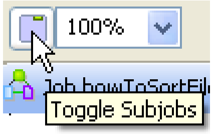
Click the minus sign (-) to collapse the subjob. When reduced, only the first component of the subjob is displayed.

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

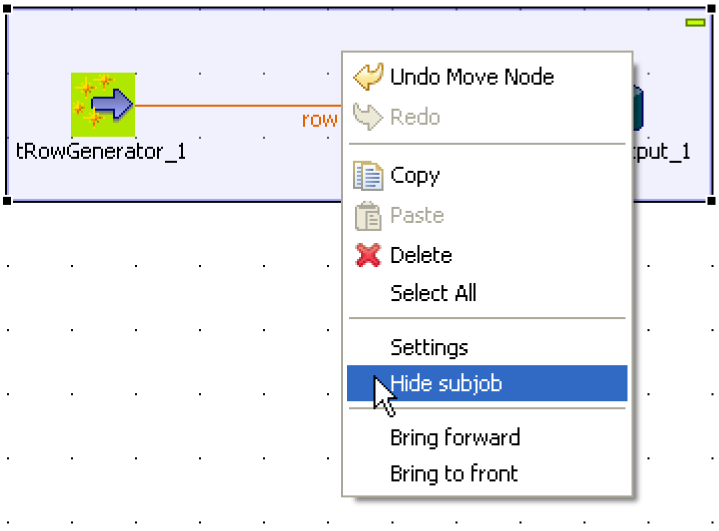
### 3.6.6.3. How to remove the subjob background color

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

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



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



# 3.6.7. How to define options on the Job view

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

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

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

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

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

For more information regarding the Log component, see *Talend Open Studio for Big Data Components Reference Guide*.

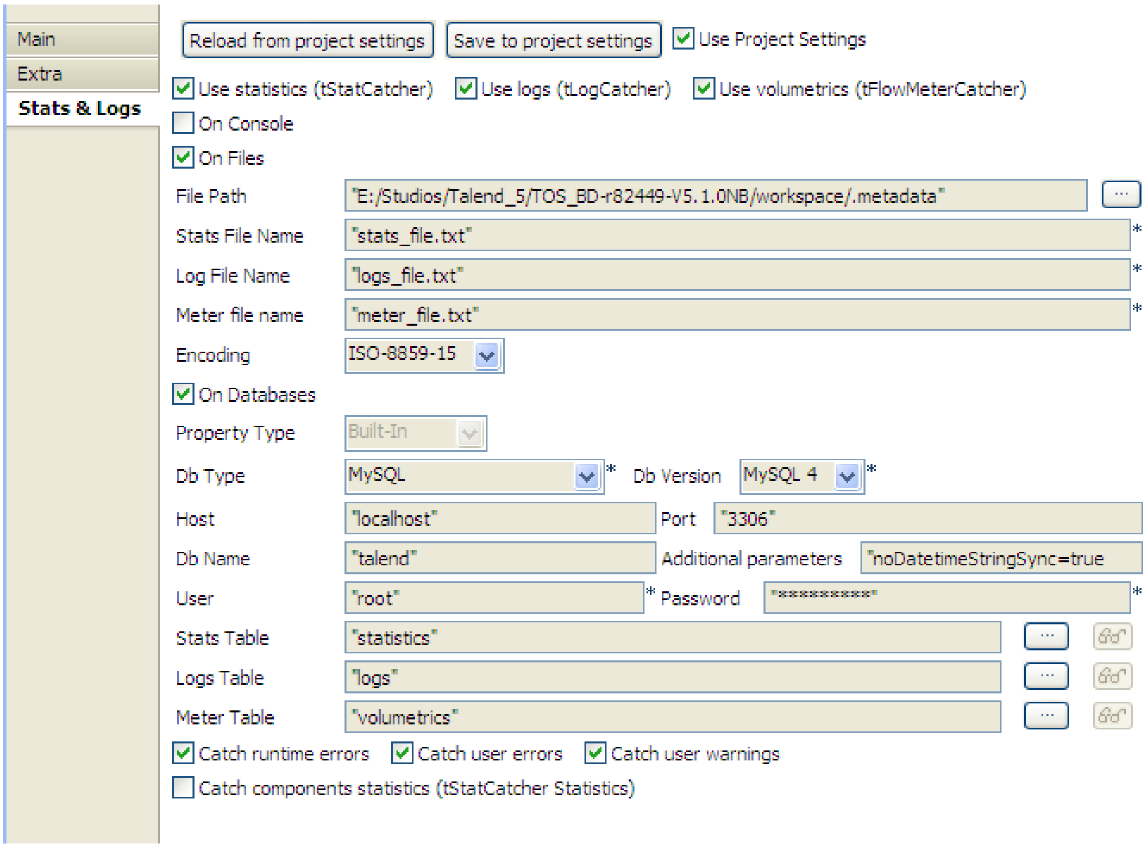
The **Stats & Logs** panel is located on the **Job** tab underneath the design workspace and prevents your Jobs Designs to be overloaded by components.

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

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

1. Click the **Job** tab.

2. Select the **Stats & Logs** panel to display the configuration view.



☑ Use Project Settings

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

□ On Console

☑ On Files

File Path "E:/Studios/Talend_5/TOS_BD-r82449-V5.1.0NB/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 Built-In

Db Type MySQL * Db Version MySQL 4 *

Host "localhost" Port "3306"

Db Name "talend" Additional parameters "noDatetimeStringSync=true

User "root" * Password "**********" *

Stats Table "statistics"

Logs Table "logs"

Meter Table "volumetrics"

☑ Catch runtime errors ☑ Catch user errors ☑ Catch user warnings

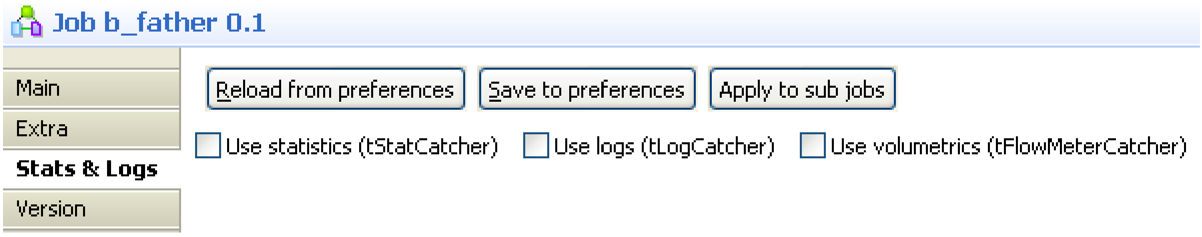
□ Catch components statistics (tStatCatcher Statistics)

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.

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 b_father 0.1

Main<br>Extra<br>**Stats & Logs**<br>Version

Reload from preferences | Save to preferences | Apply to sub jobs

□ 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.

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

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

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

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

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

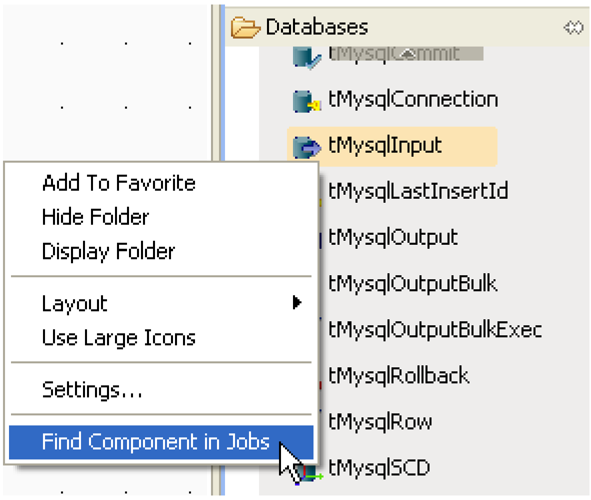
  For more information about context parameters, see section *Context settings*.
- Click **Reload from project settings** to update the context parameters list with the latest context parameters from the project settings.

## 3.6.8. How to find components in Jobs

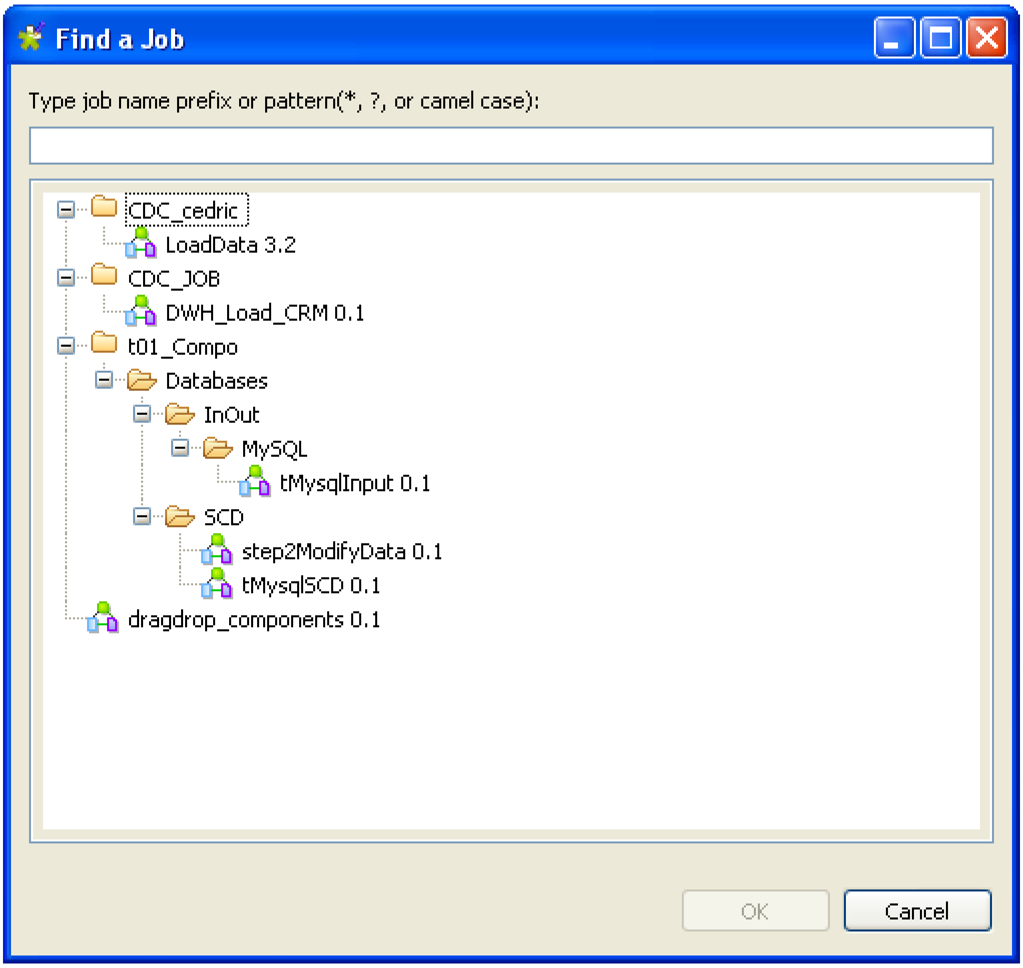
You should open one Job at least in the Studio to display the **Palette** to the right of the design workspace and thus start the search.

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

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



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



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

# 3.6.9. How to set default values in the schema of an component

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

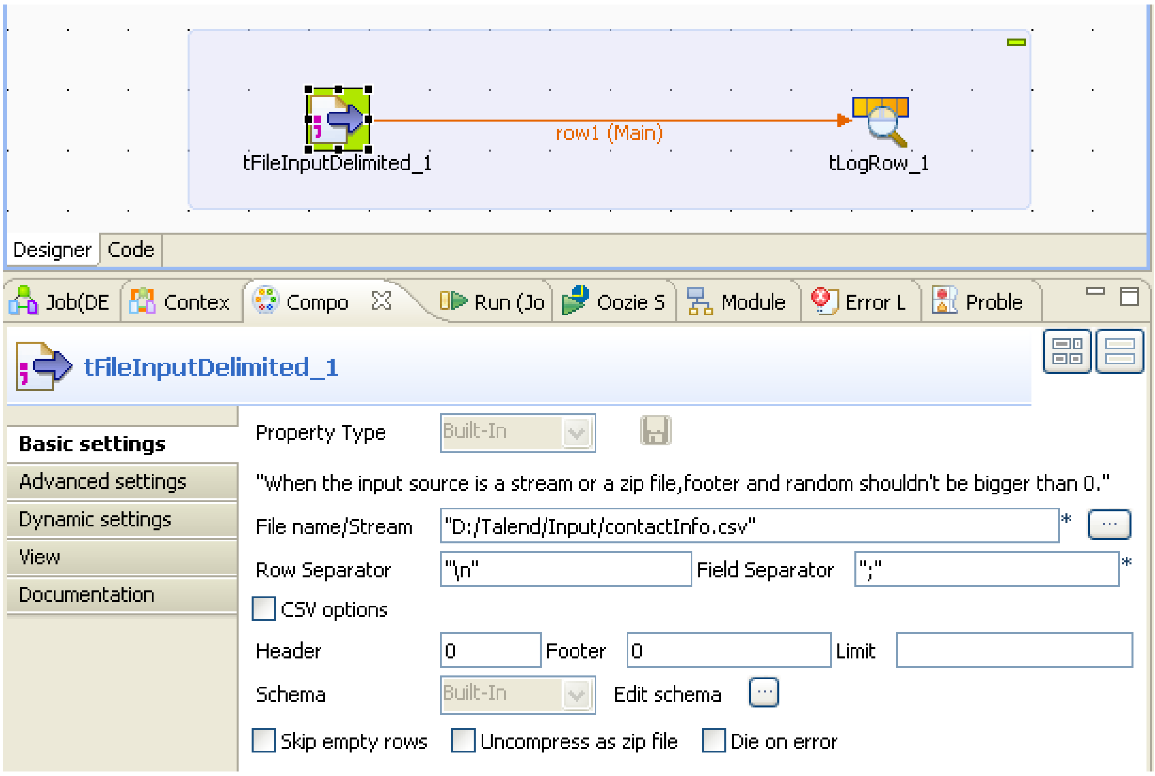
At present, only **tFileInputDelimited**, **tFileInputExcel**, and **tFixedFlowInput** support default values in the schema.

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

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

To set default values:

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



Property Type: Built-In

"When the input source is a stream or a zip file,footer and random shouldn't be bigger than 0."

File name/Stream: "D:/Talend/Input/contactInfo.csv"

Row Separator: "\n"   Field Separator: ";"

□ CSV options

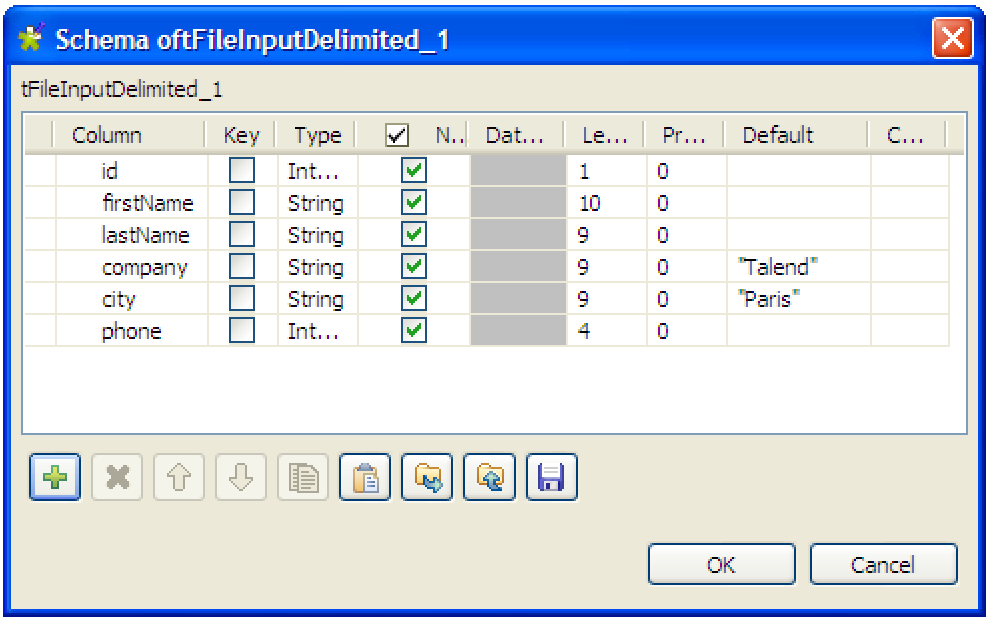
Header: 0   Footer: 0   Limit:

Schema: Built-In   Edit schema ...

□ Skip empty rows   □ Uncompress as zip file   □ Die on error

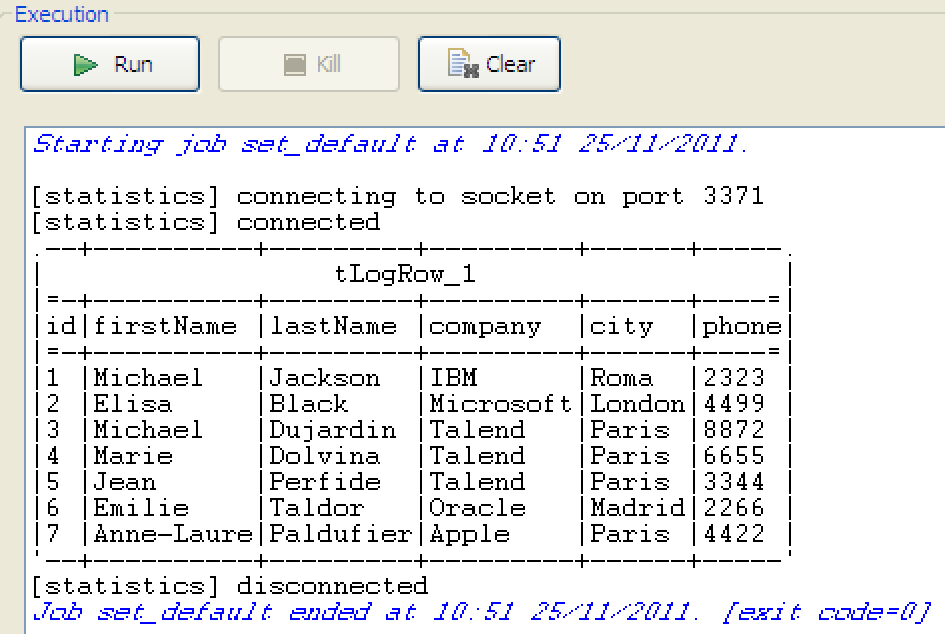
2. Click the **[...]** button next to **Edit schema**, and select the **Change to built-in property** option from the pop-up dialog box to open the schema editor.

3. Enter *Talend* between quotation marks in the **Default** field for the *company* column, enter *Paris* between quotation marks in the **Default** field for the *city* column, and click **OK** to close the schema editor.



| Column | Key | Type | ☑ N.. | Dat... | Le... | Pr... | Default | C... |
|---|---|---|---|---|---|---|---|---|
| id | □ | Int... | ☑ | | 1 | 0 | | |
| firstName | □ | String | ☑ | | 10 | 0 | | |
| lastName | □ | String | ☑ | | 9 | 0 | | |
| company | □ | String | ☑ | | 9 | 0 | "Talend" | |
| city | □ | String | ☑ | | 9 | 0 | "Paris" | |
| phone | □ | Int... | ☑ | | 4 | 0 | | |

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



```
Starting job set_default at 10:51 25/11/2011.

[statistics] connecting to socket on port 3371
[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 set_default ended at 10:51 25/11/2011. [exit code=0]
```

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

# Chapter 4. Managing data integration Jobs

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

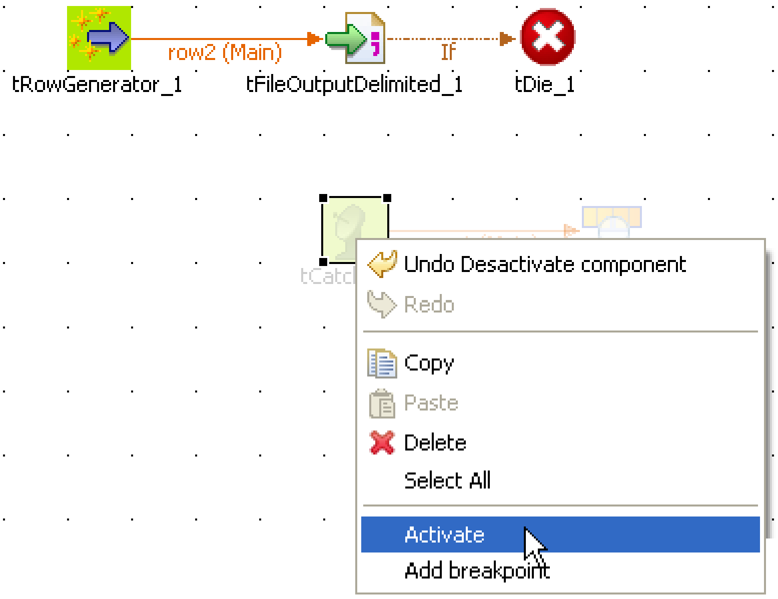
These management procedures include importing and exporting Jobs and items between different projects or machines, scheduling Job execution, etc.

Before starting any data integration processes, you need to be familiar with *Talend Open Studio for Big Data* Graphical User Interface (GUI). For more information, see appendix *GUI*.

# 4.1. Activating/Deactivating a Job or a sub-job

You can enable or disable the whole Job, sub-job directly connected to the selected component. By default, a component is activated.

In the **Main** properties of the selected component, select or clear the **Activate** check box.



Alternatively, right-click the component and select the relevant **Activate/Deactivate** command according to the current component status.

If you disable a component, no code will be generated, you will not be able to add or modify links from the disabled component to active or new components.

Related topic: section *How to define the Start component*.

## 4.1.1. How to disable a Start component

In the case the component you deactivated is a **Start** component, components of all types and links of all nature connected directly and indirectly to it will get disabled too.

## 4.1.2. How to disable a non-Start component

When you clear the **Activate** check box of a regular (non Start) component, are deactivated only the selected component itself along with all direct links.

If a direct link to the disabled component is a **Main Row** connection to a sub-job, all components of this subjob will also get disabled.

# 4.2. Importing/exporting items or Jobs

*Talend Open Studio for Big Data* 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 export Jobs and thus deploy and execute those created in the Studio on any server.

## 4.2.1. How to import items

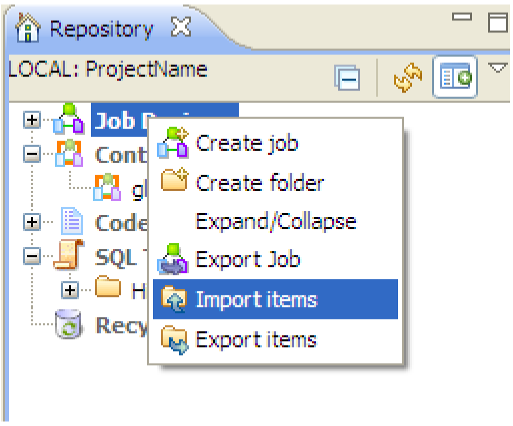
You can import items from previous versions of *Talend Open Studio for Big Data* or from a different project of your current version.

The items you can possibly import are multiple:

- Jobs Designs
- Routines

Follow the steps below to import them to the repository:

1. In the **Repository** tree view, right-click any entry such as **Job Designs**, and select **Import Items** from the contextual menu to open the **[Import items]** dialog box.

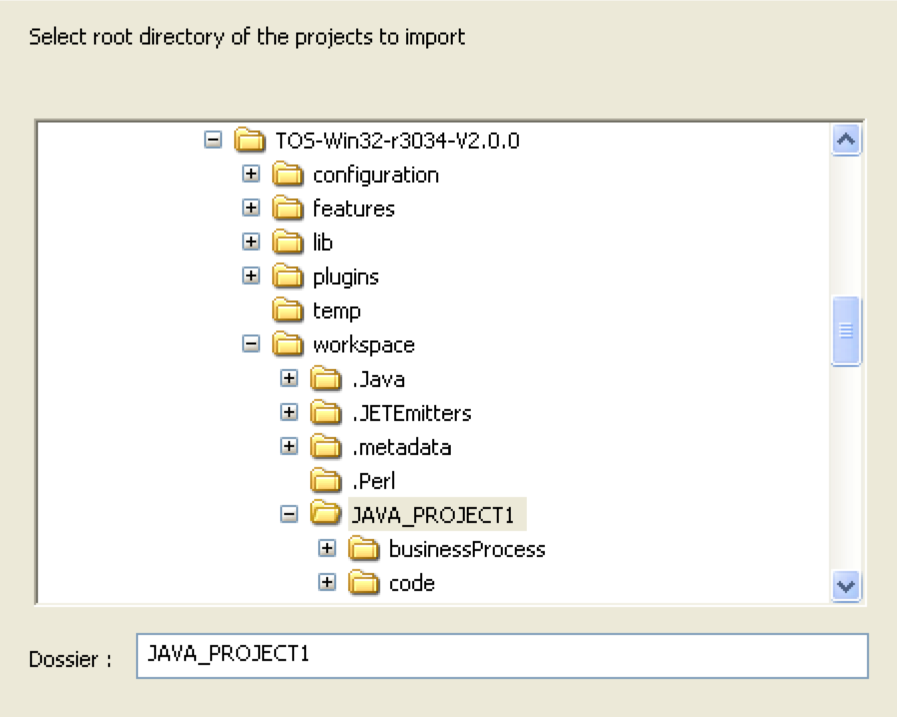


   Alternatively, you can directly click the icon on the toolbar.

2. In the dialog box that appears, select the root directory or the archive file to import the items from.

   - If the items to import are still stored on a local repository, use the **Select root directory** option and browse to the relevant project directory on your system, and then proceed to the next step.
   - If you exported the items from your local repository into an archive file (including source files and scripts), use the **Select archive file** option, browse to the file and then click **Open** to go to Step 6.

3. Browse down to the relevant **Project** folder within the workspace directory. It should correspond to the project name you picked up.

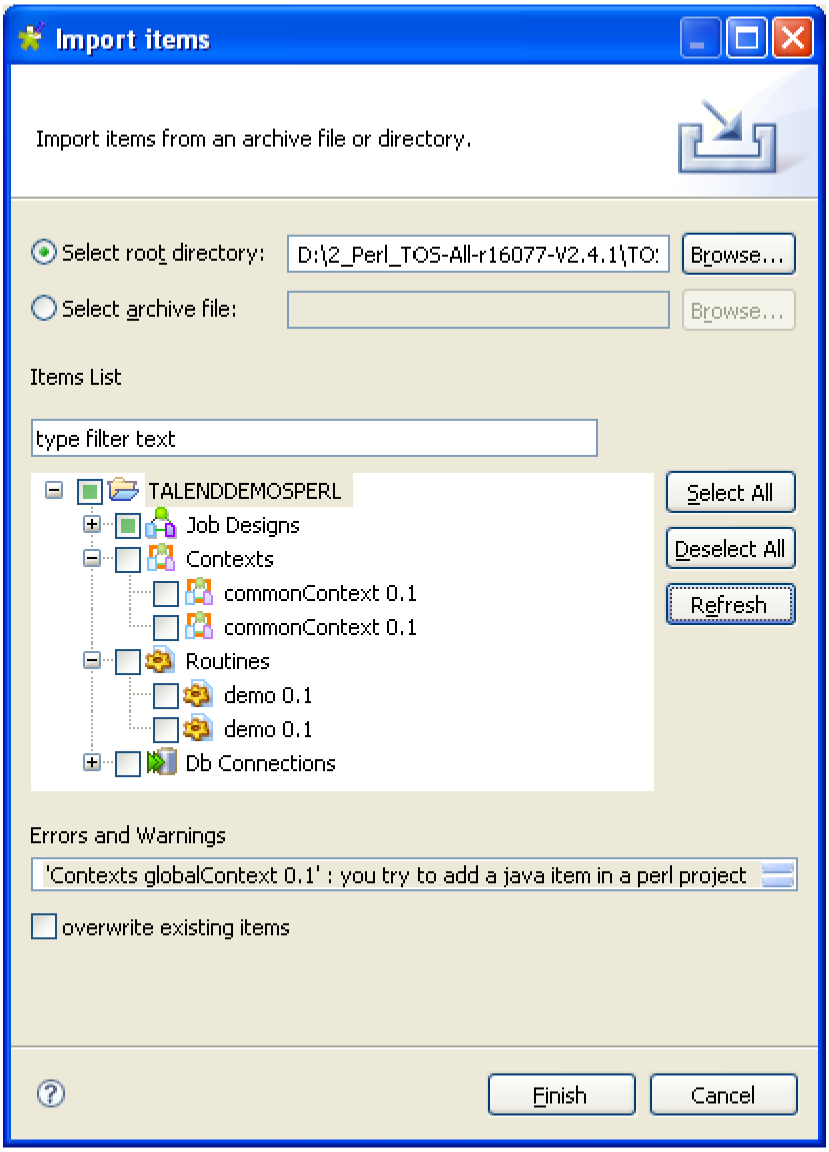


Select root directory of the projects to import

Dossier : JAVA_PROJECT1

4. 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.

5. But if your project gather various types of items (Jobs Designs, Routines...), we recommend you to select the **Project** folder to import all items in one go, and click **OK** to continue.



Import items

Import items from an archive file or directory.

● Select root directory: D:\2_Perl_TOS-All-r16077-V2.4.1\TO:

○ Select archive file:

Items List

type filter text

- ☑ TALENDDEMOSPERL
  - ☑ Job Designs
  - □ Contexts
    - □ commonContext 0.1
    - □ commonContext 0.1
  - □ Routines
    - □ demo 0.1
    - □ demo 0.1
  - □ Db Connections

Errors and Warnings

'Contexts globalContext 0.1' : you try to add a java item in a perl project

□ overwrite existing items

6. Select the **overwirte existing items** check box if you want to overwrite existing items with those having the same names to be imported. This will refresh the **Items List**.

7. From the **Items List** whih displays all valid items that can be imported, select the items that you want to import by selecting the corresponding check boxes.

8. Click **Finish** to validate the import.

   The imported items are displayed in the repository in the relevant folder respective to their nature.

## 4.2.2. How to export Jobs

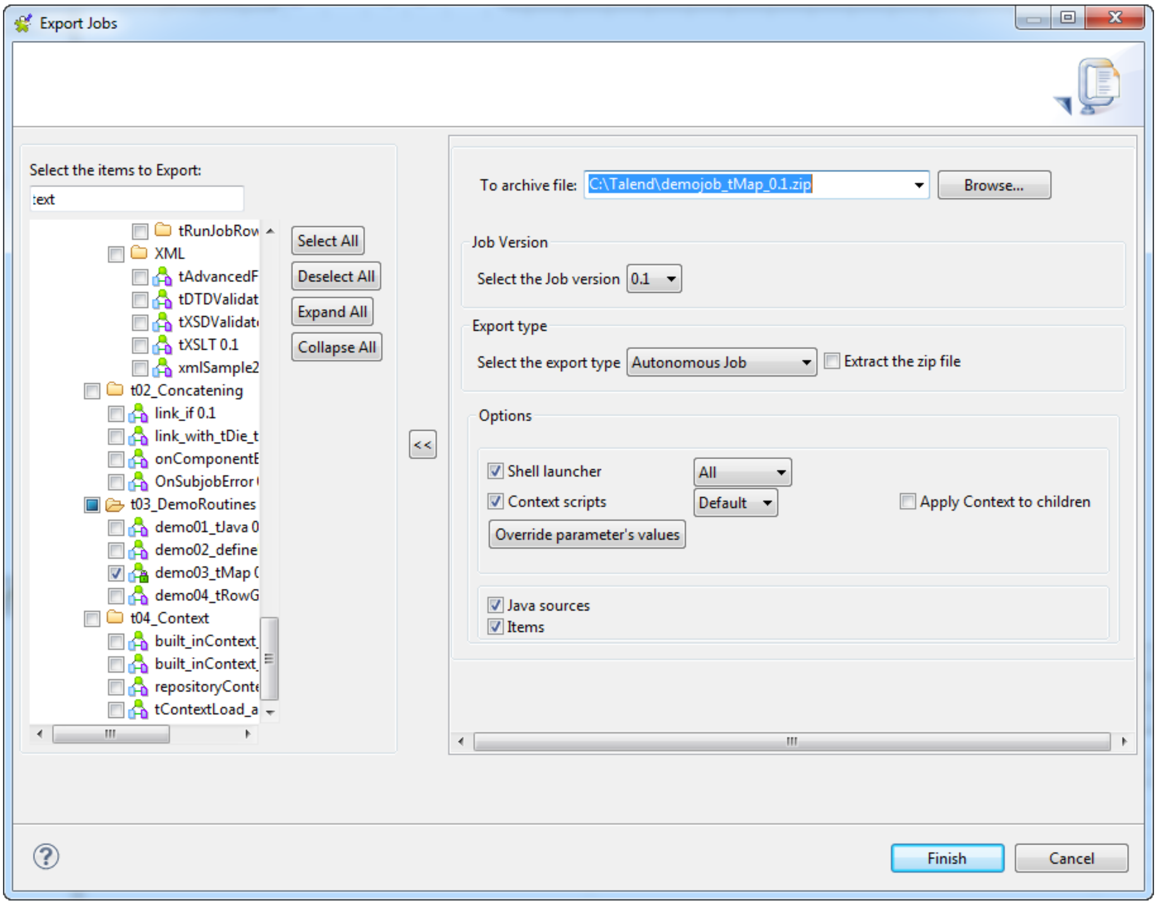
The **Export Job** feature allows you to deploy and execute a Job on any server, independent of *Talend Open Studio for Big Data*.

The export 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.

To export Jobs, complete the following:

1. In the **Repository** tree view, right-click the Job you want to export, and select **Export Job** to open the **[Export Jobs]** dialog box.

   You can show/hide a tree view of all created Jobs in *Talend Open Studio for Big Data* directly from the **[Export Jobs]** 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 Open Studio for Big Data* to do that.



To archive file: C:\Talend\demojob_tMap_0.1.zip

Job Version

Select the Job version 0.1

Export type

Select the export type Autonomous Job □ Extract the zip file

Options

☑ Shell launcher All

☑ Context scripts Default □ Apply Context to children

Override parameter's values

☑ Java sources

☑ Items

2. In the **To archive file** field, browse to the directory where you want to save your exported Job.

3. Select the **Export Type** from the list between **Autonomous Job**, **Axis Webservice (WAR)**, **Axis Webservice (Zip)**, **JBoss ESB**, **Petals ESB** and **OSGI Bundle For ESB**.

4. Select the **Extract the zip file** check box if you want the archive file to be automatically extracted in the target directory.

5. 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 export.

| Option | Description |
|---|---|
| **Shell launcher** | Select this check box to export the *.bat* and/or *.sh* files necessary to launch the exported Job.<br>• **All**: exports the *.bat* and *.sh* files.<br>• **Unix** exports the *.sh* file.<br>• **Windows** exports the *.bat* file. |
| **Context scripts** | Select this check box to export ALL context parameters files and not just those you select in the corresponding list. |

| Option | Description |
|---|---|
| | To export only one context, select the context that fits your needs from the **Context scripts** 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. |
| **Apply to children** | Select this check box if you want to apply the context selected from the list to all child Jobs. |
| **Java sources** | Select this check box to export the *.java* file holding Java classes generated by the Job when designing it. |
| **Items / Source files** | Select this check box to export the sources used by the Job during its execution including the *.item* and *.properties* files, Java and **Talend** sources.<br>If you select the **Items** or **Source files** check box, you can reuse the exported Job in a *Talend Open Studio for Big Data* installed on another machine. These source files are only used in *Talend Open Studio for Big Data*. |

6. 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.

7. Click **Finish** to validate your changes, complete the export operation and close the dialog box.

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

If the Job to be exported 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 section *Managing user routines*. For more information about classes and inner classes, see relevant Java manuals.

### 4.2.2.1. How to export Jobs as Autonomous Job

In the case of a Plain Old Java Object export, if you want to reuse the Job in *Talend Open Studio for Big Data* installed on another machine, make sure you selected the **Items** check box. These source files (*.item* and *.properties*) are only needed within *Talend Open Studio for Big Data*.

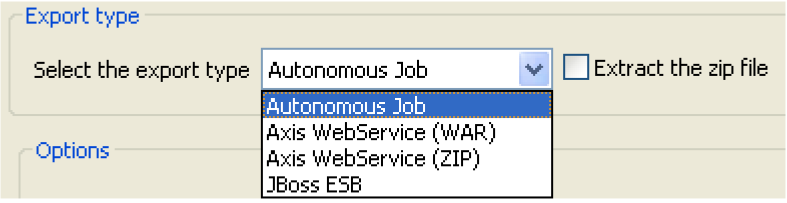
Select a context from the list when offered. Then once you click the **Override parameters' values** button below the **Context scripts** check box, the opened window will list all of the parameters of the selected context. In this window, you can configure the selected context as needs.

All contexts parameter files are exported along in addition to the one selected in the list.

After being exported, the context selection information is stored in the *.bat* or *.sh* file and the context settings are stored in the context *.properties* file.

### 4.2.2.2. How to export Jobs as Webservice

In the **[Export Jobs]** dialog box, you can change the type of export in order to export the Job selection as Webservice archive.



Select the export type: Autonomous Job

☐ Extract the zip file

Select the type of archive you want to use in your Web application.

| Archive type | Description |
|---|---|
| WAR | The options are read-only. Indeed, the WAR archive generated includes all configuration files necessary for the execution or deployment from the Web application. |
| ZIP | All options are available. In the case the files of your Web application config are all set, you have the possibility to only set the Context parameters if relevant and export only the Classes into the archive. |

Once the archive is produced, place the WAR or the relevant Class from the ZIP (or unzipped files) into the relevant location, of your Web application server.

The URL to be used to deploy the Job, typically reads as follow:

```
http://localhost:8080/Webappname/services/JobName?method=runJob&args=null
```

where the parameters stand as follow:

| URL parameters | Description |
|---|---|
| http://localhost:8080/ | Type in the Webapp host and port. |
| /Webappname/ | Type in the actual name of your web application. |
| /services/ | Type in “services” as the standard call term for web services. |
| /JobName | Type in the exact name of the Job you want to execute. |
| ?method=runJob&args=null | The method is RunJob to execute the Job. |

The call return from the Web application is 0 when there is no error and different from 0 in case of error. For a real-life example of creating and exporting a Job as a Webservice and calling the exported Job from a browser, see section *An example of exporting a Job as a Web service*.

The **tBufferOutput** component was especially designed for this type of deployment. For more information regarding this component, see *Talend Open Studio for Big Data Components Reference Guide*.

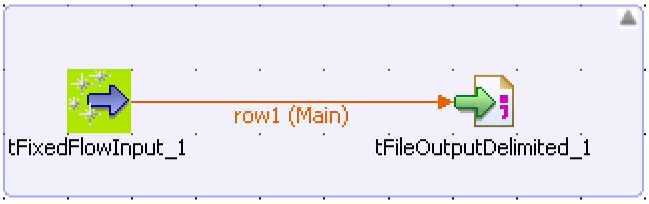
### 4.2.2.3. An example of exporting a Job as a Web service

This scenario describes first a simple Job that creates a *.txt* file and writes in it the current date along with first and last names. Second, it shows how to export this Job as a Webservice. And finally, it calls the Job exported as a Webservice from a browser. The exported Job as a Webservice will simply return the “return code” given by the operating system.

#### Creating the Job:

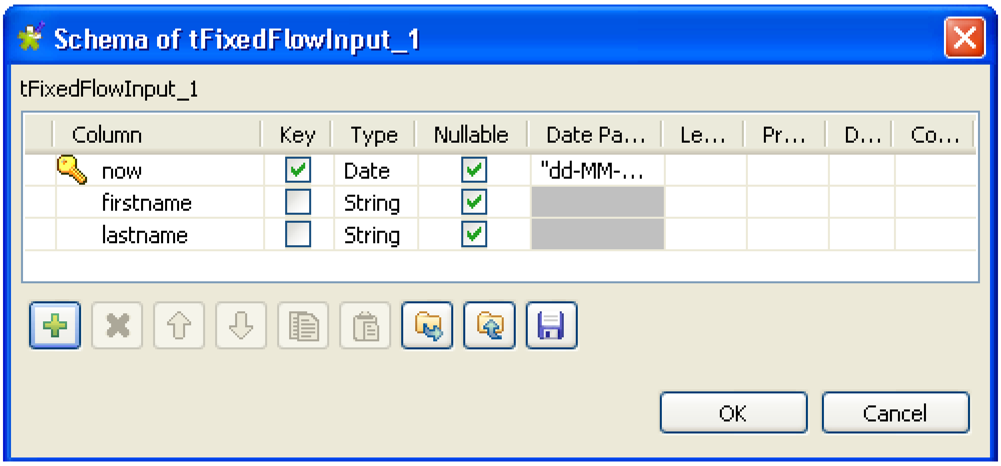
1. Drop the following components from the **Palette** onto the design workspace: **tFixedFlowInput** and **tFileOutputDelimited**.

2. Connect **tFixedFlowInput** to **tFileOutputDelimited** using a **Row** > **Main** link.



3. In the design workspace, select **tFixedFlowInput**, and click the **Component** tab to define the basic settings for **tFixedFlowInput**.

4. Set the **Schema** to **Built-In** and click the **[...]** button next to **Edit Schema** to describe the data structure you want to create from internal variables. In this scenario, the schema is made of three columns, *now*, *firstname*, and *lastname*.



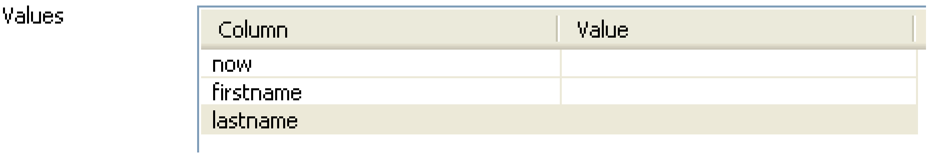
Schema of tFixedFlowInput_1

tFixedFlowInput_1

| Column | Key | Type | Nullable | Date Pa... | Le... | Pr... | D... | Co... |
|---|---|---|---|---|---|---|---|---|
| now | ☑ | Date | ☑ | "dd-MM-... | | | | |
| firstname | □ | String | ☑ | | | | | |
| lastname | □ | String | ☑ | | | | | |

5. Click the **[+]** button to add the three parameter lines and define your variables, and then click **OK** to close the dialog box and accept propagating the changes when prompted by the system.

The three defined columns display in the **Values** table of the **Basic settings** view of **tFixedFlowInput**.

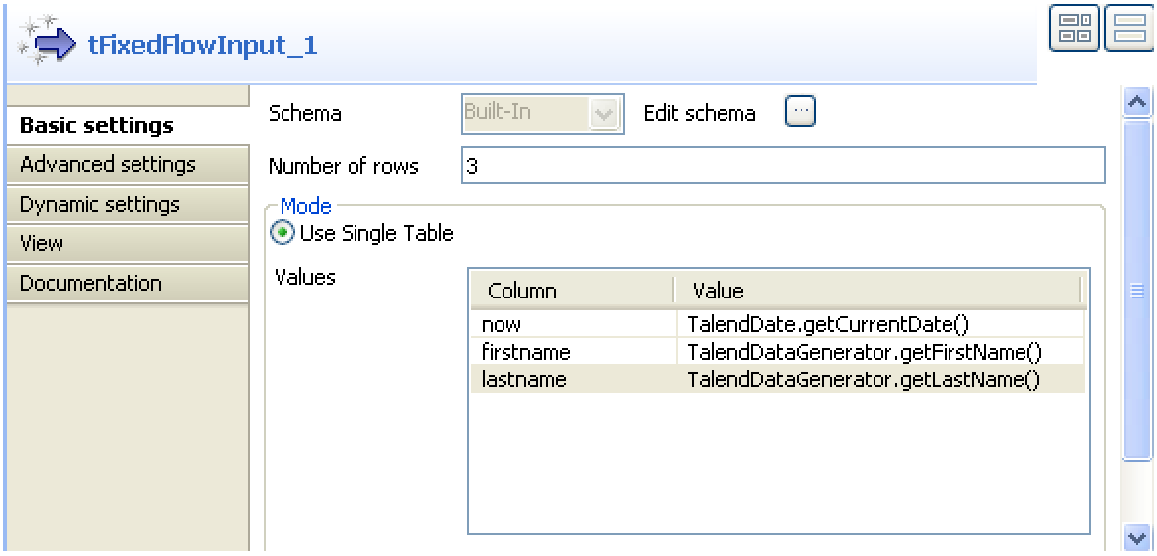


Values

| Column | Value |
|---|---|
| now | |
| firstname | |
| lastname | |

6. In the **Value** cell of each of the three defined columns, press **Ctrl+Space** to access the global variable list, and select *TalendDate.getCurrentDate()*, *talendDatagenerator.getFirstName*, and *talendDataGenerator.getLastName* for the *now*, *firstname*, and *lastname* columns respectively.

7. In the **Number of rows** field, enter the number of lines to be generated.



tFixedFlowInput_1

Schema: Built-In

Edit schema ...

Number of rows: 3

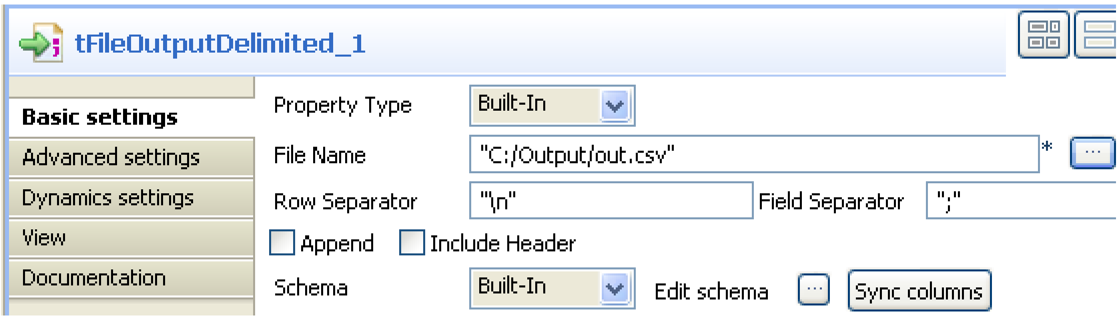
Mode

● Use Single Table

Values

| Column | Value |
|---|---|
| now | TalendDate.getCurrentDate() |
| firstname | TalendDataGenerator.getFirstName() |
| lastname | TalendDataGenerator.getLastName() |

8. In the design workspace, select **tFileOutputDelimited**, click the **Component** tab for **tFileOutputDelimited**, and browse to the output file to set its path in the **File name** field. Define other properties as needed.



tFileOutputDelimited_1

Property Type: Built-In

File Name: "C:/Output/out.csv"

Row Separator: "\n"  Field Separator: ";"

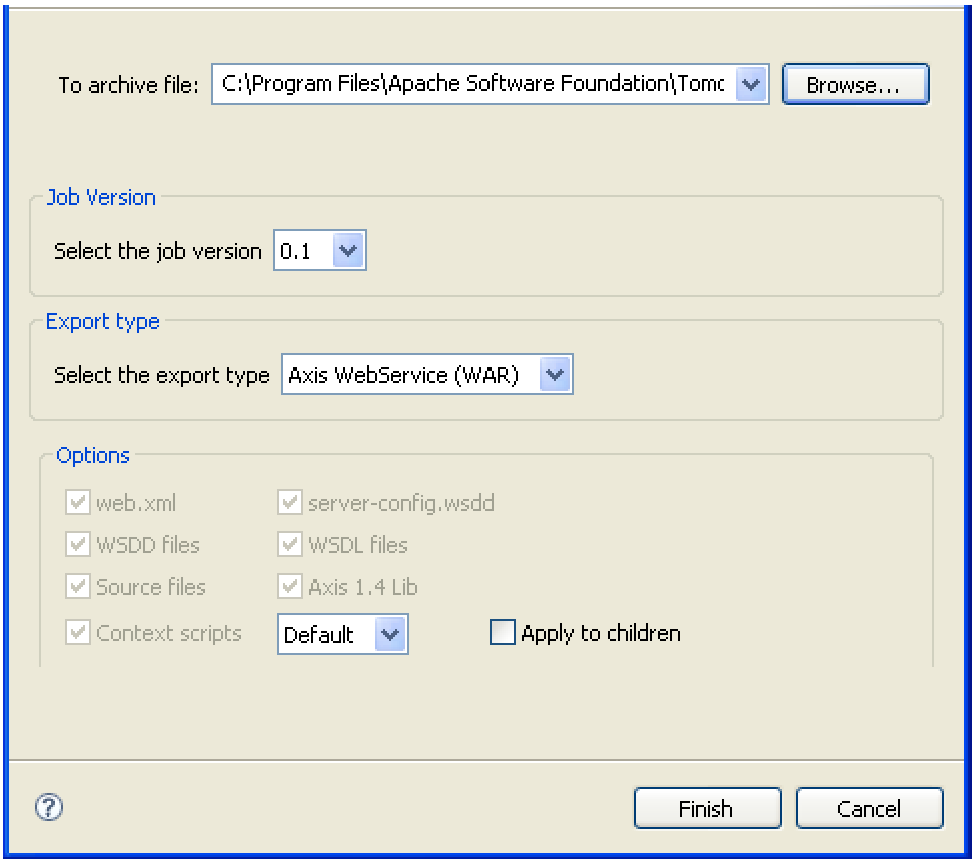
□ Append □ Include Header

Schema: Built-In  Edit schema

If you press **F6** to execute the Job, three rows holding the current date and first and last names will be written to the set output file.

## Exporting the Job as a Webservice:

1. In the **Repository** tree view, right-click the above created Job and select **Export Job**. The **[Export Jobs]** dialog box appears.



To archive file: C:\Program Files\Apache Software Foundation\Tomc

Job Version

Select the job version: 0.1

Export type

Select the export type: Axis WebService (WAR)

Options

☑ web.xml ☑ server-config.wsdd

☑ WSDD files ☑ WSDL files

☑ Source files ☑ Axis 1.4 Lib

☑ Context scripts  Default  □ Apply to children

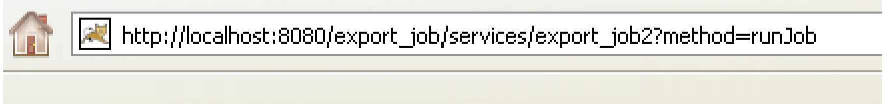
2. Click the **Browse...** button to select a directory to archive your Job in.

3. In the **Export type** area, select the export type you want to use in your Web application (WAR in this example) and click **Finish**. The **[Export Jobs]** dialog box disappears.

4. Copy the War folder and paste it in the Tomcat webapp directory.

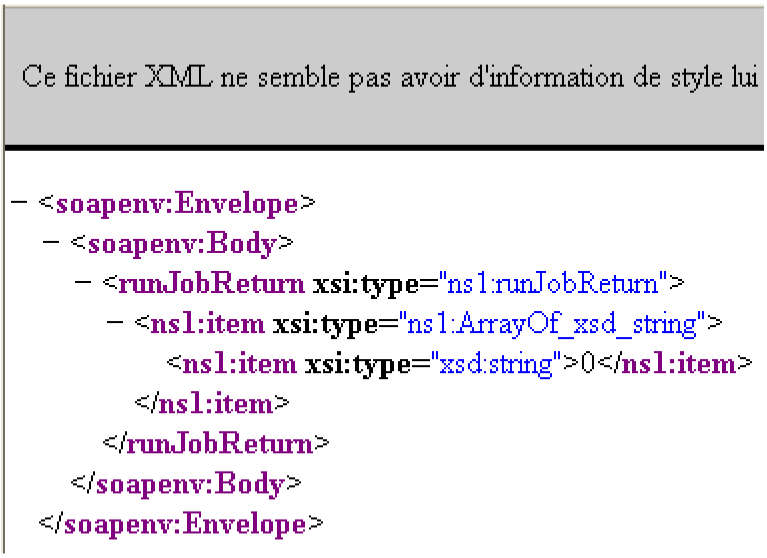
## Calling the Job from a browser:

1. Type the following URL into your browser: *http://localhost:8080//export_job/services/export_job2?method=runJob* where “export_job” is the name of the webapp directory deployed in Tomcat and “export_job2” is the name of the Job.



```
http://localhost:8080/export_job/services/export_job2?method=runJob
```

2. Click **Enter** to execute the Job from your browser.



Ce fichier XML ne semble pas avoir d'information de style lui

```
- <soapenv:Envelope>
  - <soapenv:Body>
    - <runJobReturn xsi:type="ns1:runJobReturn">
      - <ns1:item xsi:type="ns1:ArrayOf_xsd_string">
          <ns1:item xsi:type="xsd:string">0</ns1:item>
        </ns1:item>
      </runJobReturn>
    </soapenv:Body>
  </soapenv:Envelope>
```

The return code from the Web application is 0 when there is no error and 1 if an error occurs.

For a real-life example of creating and exporting a Job as a Webservices using the **tBufferOutput** component, see the **tBufferOutput** component in *Talend Open Studio for Big Data Components Reference Guide*.

### 4.2.2.4. How to export Jobs as JBoss ESB

*Talend Open Studio for Big Data* provides the possibility to expose **Talend** Jobs as services into JBoss ESB (Enterprise Service Bus) in order to execute these Jobs on the messaging engine (the bus).

*In order to be able to export a Job to be deployed on a JBoss ESB server, make sure that the jar specific to JBoss ESB is installed in the Java library and that it displays in the Modules view of* Talend Open Studio for Big Data . *For more information about the Modules view, see section How to install external modules.*

In the **[Export Jobs]** dialog box, you can change the type of export in order to export the selected Job as an ESB archive. You can then deploy this exported Job on a JBoss ESB server.

To export a Job on ESB:

1. In the **Job Version** area, select the version of the Job you want to execute on a JBoss ESB server.

2. From the **Select export type** list in the **Export type** area, select **JBoss ESB**.

3. In the **Options** area, select the file type you want to add to the archive. When the **Context scripts** list displays more than one context, select the one you need, and select the **Apply to children** check box if you want to apply the context selected from the list to all child Jobs.

4. To export the sources used by the Job during its execution including the files *.item*, *.properties* and Java sources of Jobs and routines, select the **Source files** check box.

   If you select the **Source files** check box, you can reuse the exported Job in a *Talend Open Studio for Big Data* installed on another machine. These source files are only used in *Talend Open Studio for Big Data*.

5. In the **ESB Export type** list, select between *JBoss MQ* or *JBoss Messaging*.

6. In the **Service Name** field, type in the name of the service on which you will deploy your Job.
7. In the **Category** field, type in the category of the service on which the Job will be deployed.
8. In the **Message Queue Name** field, type in the name of the queue that is used to deploy the Job.
9. Click the **Browse...** button next to the **To archive file** field and browse to set the path to the archive file in which you want to export the Job. Then click **Finish**.

   The dialog box closes. A progress indicator displays to show the progress percentage of the export operation. The Job is exported in the selected archive.

When you copy the ESB archive in the deployment directory and launch the server, the Job is automatically deployed and will be ready to be executed on the ESB server.

## 4.2.2.5. How to export Jobs as Petals ESB

*Talend Open Studio for Big Data* provides the possibility to expose **Talend** Jobs as services into petals ESB (Enterprise Service Bus) in order to execute these Jobs on the messaging engine (the bus).

### Integrating Talend with petals ESB

**Talend** provides a smooth approach to expose services on petals ESB and thus facilitates:

- application integration on the bus: This will enable the integration of systems and applications across the enterprise.
- service interactions: The ESB provides connectivity between services. i.e. allows services with varying interfaces to communicate.

The Java Business Integration (JBI) is the approach used to implement a service-oriented architecture (SOA) and export **Talend** Jobs on petals ESB.

Petals ESB is complemented with Binding Components (BC) and **Talend** Service Engine (SE) in order to provide: first the access methods necessary for different types of services including FileTransfer, WebService, MOM, and second the engine to deploy the service. For more information about interaction between Petals and **Talend** Jobs, check http://doc.petalslink.com/display/petalsesb/A+Simple+Talend+Job.

Then, with the integration of **Talend** and petals ESB, you can execute the Jobs designed in *Talend Open Studio for Big Data* on petals ESB. For more information, see section *How to export Jobs as Petals ESB*. Several mechanisms are provided to pass information and data to a Job and to retrieve information and data from a Job.

Using *Talend Open Studio for Big Data* and petals ESB, you can execute a Job which has no specific interaction with Petals. You can:

- expose a context as a parameter into the service's WSDL,
- pass attachment files to a Job,
- pass native parameters and options to a Job,
- get the Job's execution result.

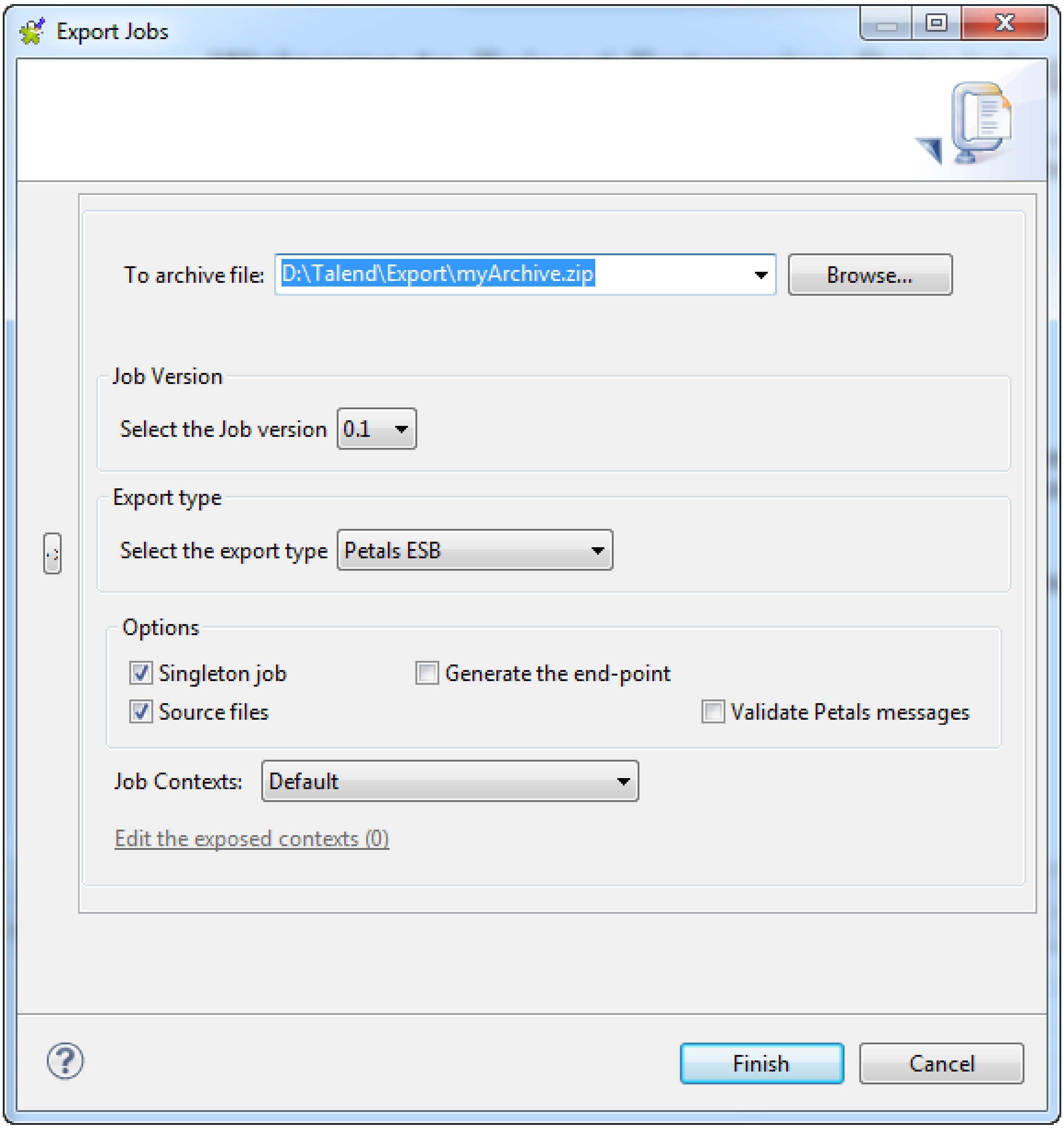
### How to export Jobs to petals ESB

From the **[Export Jobs]** dialog box, you can export a selected Job as a petals ESB archive. You can then execute the exported Job on the bus (the messaging engine).

To export a Job as a petals ESB archive, complete the following:

1. In the **Repository** tree view, right-click the Job you want to export and then select **Export Job** from the contextual menu.

   The **[Export Jobs]** dialog box appears.



To archive file: D:\Talend\Export\myArchive.zip

Job Version

Select the Job version 0.1

Export type

Select the export type Petals ESB

Options

☑ Singleton job ☐ Generate the end-point

☑ Source files ☐ Validate Petals messages

Job Contexts: Default

2. In the **To archive file** field, browse to set the path to the archive file in which you want to export the Job.

3. From the **Select the job version** list, select the Job version you want to export.

4. From the **Select export type** list in the **Export type** area, select **Petals ESB**.

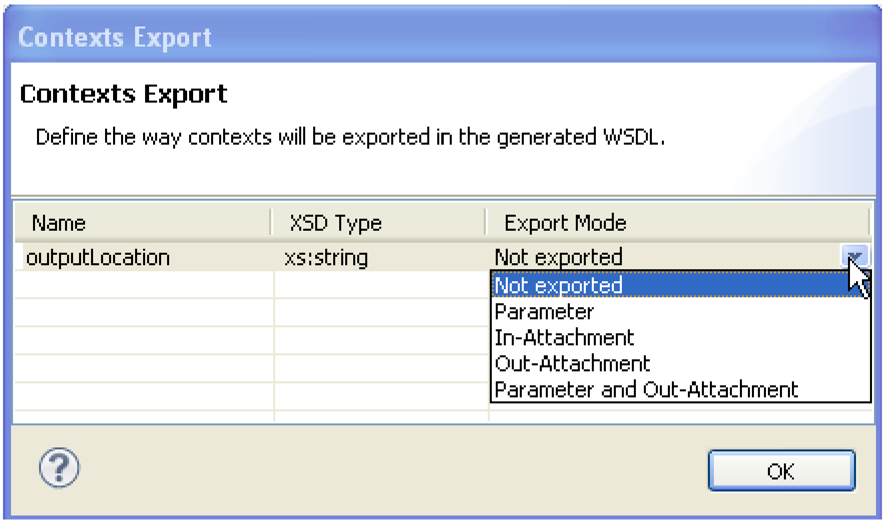
   The three following options in the **Options** area are selected by default: **Singleton job** and **Source file**. You can select any of the other options as needed.

   The table below explains the export options:

| Option | Description |
|---|---|
| Singleton job | Exports the Job as singleton: A singleton Job can have only one instance running at a time on a given **Talend** Service Engine in petals ESB. |

| Option | Description |
|---|---|
| Generate the end-point | Generates the end-point at deployment time. If this option is not selected, the end-point name is the Job name with the suffix `Endpoint`. |
| Validate Petals messages | Validates all the messages / requests against the WSDL.<br>💡 Selecting this option reduces system performance (disk access). |
| Source files | Embeds the source files in the generated service-unit. |
| Jobs contexts | A list from which to select the context that will be used by default by the Job. |

5. In the **[Export Jobs]** dialog box, click the **Edit the exposed contexts** link to open the **[Context Export]** dialog box.



Contexts Export

Contexts Export

Define the way contexts will be exported in the generated WSDL.

| Name | XSD Type | Export Mode |
|---|---|---|
| outputLocation | xs:string | Not exported |

This dialog box will display a list of all the context variables that are used in the exported Job. Here you can specify how each context variable should be exported in the generated WSDL file.

6. Click in the **Export Mode field** and select from the list the context export mode for each of the listed context variables.

The table below explains the export mode options:

| Export Mode | Description |
|---|---|
| Not exported | The context is not exported (not visible as a parameter). But the context can still be overridden using the native parameters (options) of the Job. |
| Parameter | The context is exported as a parameter in the WSDL. |
| In-Attachment | The context will pass the path of a temporary file which content is attached in the input message. |
| Out-Attachment | The context will be read after the Job execution.<br>-This context must point to a file,<br>-The file content will be read by the service engine and attached to the response,<br>-The context name will be used as the attachment name,<br>-The file will be deleted by the service engine right after its content was loaded. |
| Parameter and Out-Attachment | A mix between the **Parameter** and the **Out-Attachment** modes.<br>-The context is exposed as a parameter,<br>-It will also be read after the Job execution,<br>-The file will be deleted anyway,<br>The advantage of this export mode is to define the output file destination dynamically. |

7. Click **OK** to validate your choice and close the **[Context Export]** dialog box.

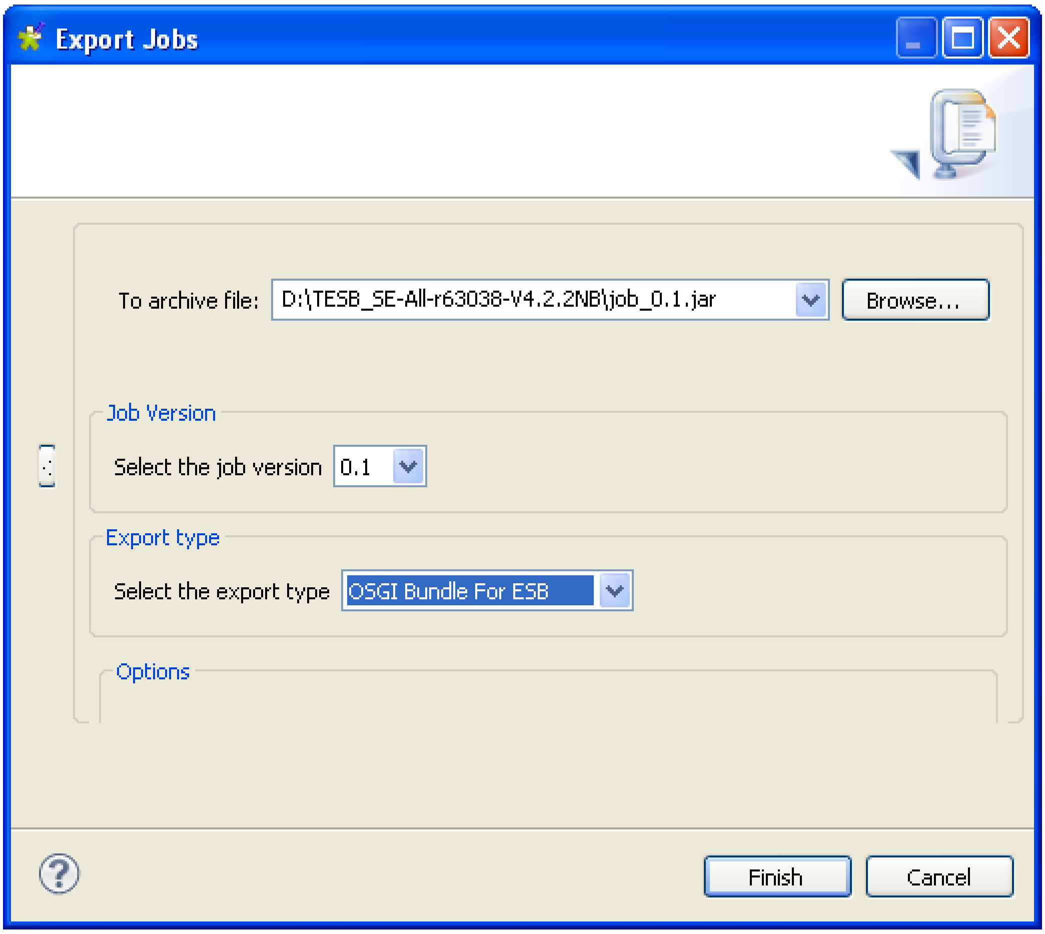
8. In the **[Export Jobs]** dialog box, click **Finish**.

   The dialog box closes. A progress indicator displays to show the progress percentage of the export operation. The Job is exported in the selected archive.

   The **Talend** Job is now exposed as a service into petals ESB and can be executed inside the bus.

### 4.2.2.6. How to export a Job as an OSGI Bundle For ESB

In the **[Export Jobs]** dialog box, you can change the type of export in order to export the Job selection as an OSGI Bundle in order to deploy your Job in **Talend ESB Container**.



To archive file: D:\TESB_SE-All-r63038-V4.2.2NB\job_0.1.jar

Job Version

Select the job version 0.1

Export type

Select the export type OSGI Bundle For ESB

1. In the **Export type** area, select **OSGI Bundle For ESB** to export your Job as an OSGI Bundle.

   The extension of your export automatically change to *.jar* as it is what **Talend ESB Container** is expecting.

2. Click the **Browse...** button to specify the folder in which exporting your Job.

3. Click **Finish** to export it.

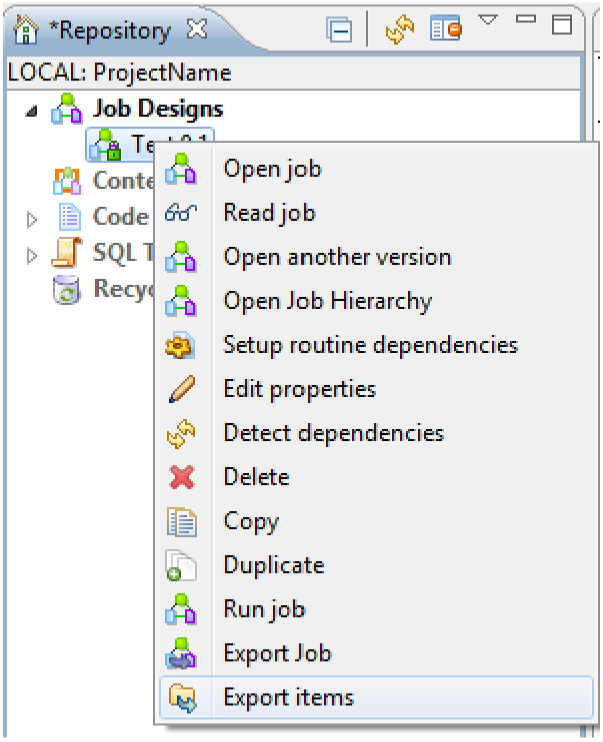
## 4.2.3. How to export items

You can export multiple items from the repository onto a directory or an archive file.

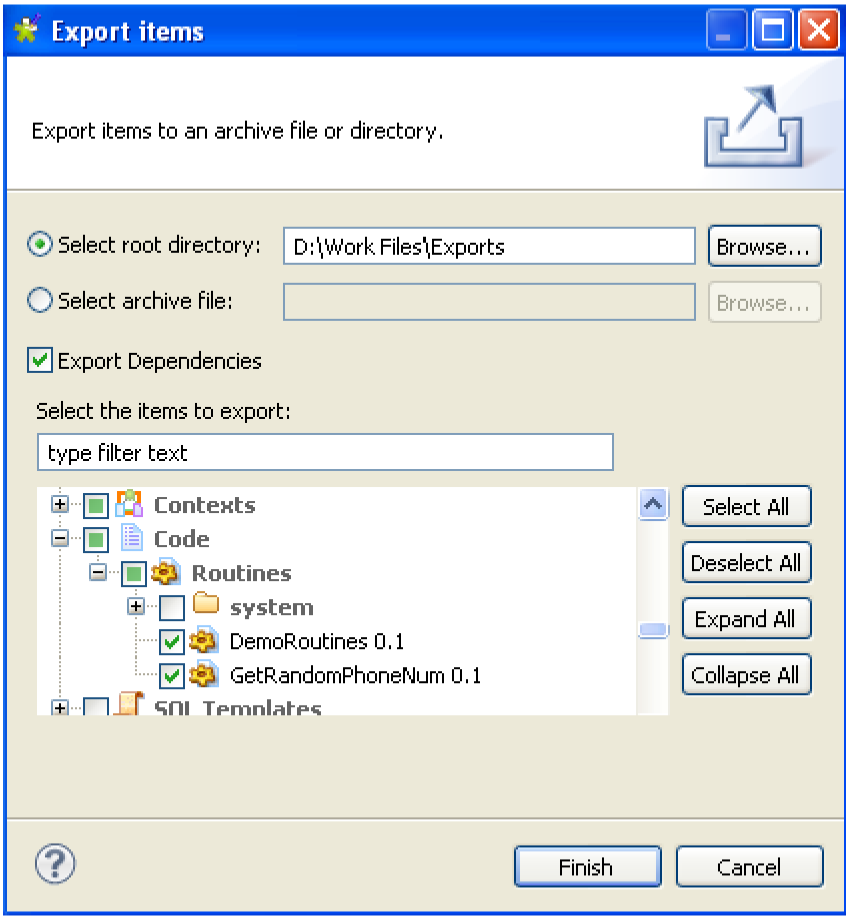
To do so:

1. In the **Repository** tree view, select the items you want to export.

2. To select several items at a time, press the **Ctrl** key and select the relevant items.



3. Right-click while maintaining the **Ctrl** key down and select **Export items** on the pop-up menu:



Export items to an archive file or directory.

● Select root directory: D:\Work Files\Exports Browse...

○ Select archive file: Browse...

☑ Export Dependencies

Select the items to export:

type filter text

You can select additional items on the tree for exportation if required.

4. Click **Browse** to browse to where you want to store the exported items. Alternatively, define the archive file where to compress the files for all selected items.

   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 section *What are routines*.

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

## 4.2.4. How to change context parameters in Jobs

As explained in section *How to export Jobs*, you can edit the context parameters:

If you want to change the context selection, simply edit the .bat/.sh file and change the following setting: `--context=Prod` to the relevant context.

If you want to change individual parameters in the context selection, edit the .bat/.sh file and add the following setting according to your need:

| **Operation** | **Setting** |
|---|---|
| To change *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”` |

# 4.3. Managing repository items

*Talend Open Studio for Big Data* enables you to edit the items centralized in the repository and to update the Jobs that use these items accordingly.

## 4.3.1. How to handle updates in repository items

You can update the context parameters that are centralized in the **Repository** tree view any time in order to update the context group details, for example.

When you modify any of the parameters of an entry in the **Repository** tree view, all Jobs using this repository entry will be impacted by the modification. This is why the system will prompt you to propagate these modifications to all the Jobs that use the repository entry.

The following sections explain how to modify the parameters of a repository entry and how to propagate the modifications to all or some of the Jobs that use the entry in question.

### 4.3.1.1. How to modify a repository item

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

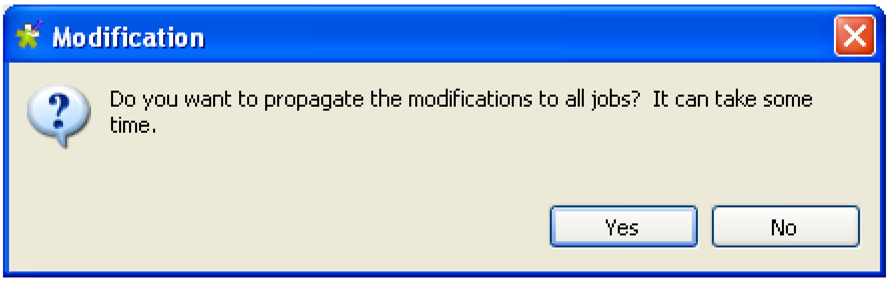
1. Expand the **Contexts** node in the **Repository** tree view and browse to the relevant entry that you need to update.

2. Right-click this entry and select the corresponding edit option in the contextual menu.

   A respective wizard displays where you can edit each of the definition steps for the entry parameters.

   When updating the entry parameters, you need to propagate the changes throughout numerous Jobs or all your Jobs that use this entry.

   A prompt message pops up automatically at the end of your update/modification process when you click the **Finish** button in the wizard.



Do you want to propagate the modifications to all jobs? It can take some time.

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 section *How to update impacted Jobs automatically*.

   Click **No** if you want to close the message without propagating the changes. This will allow you to propagate your changes on the impacted Jobs manually on one by one basis. For more information on another way of propagating changes, see section *How to update impacted Jobs manually*.

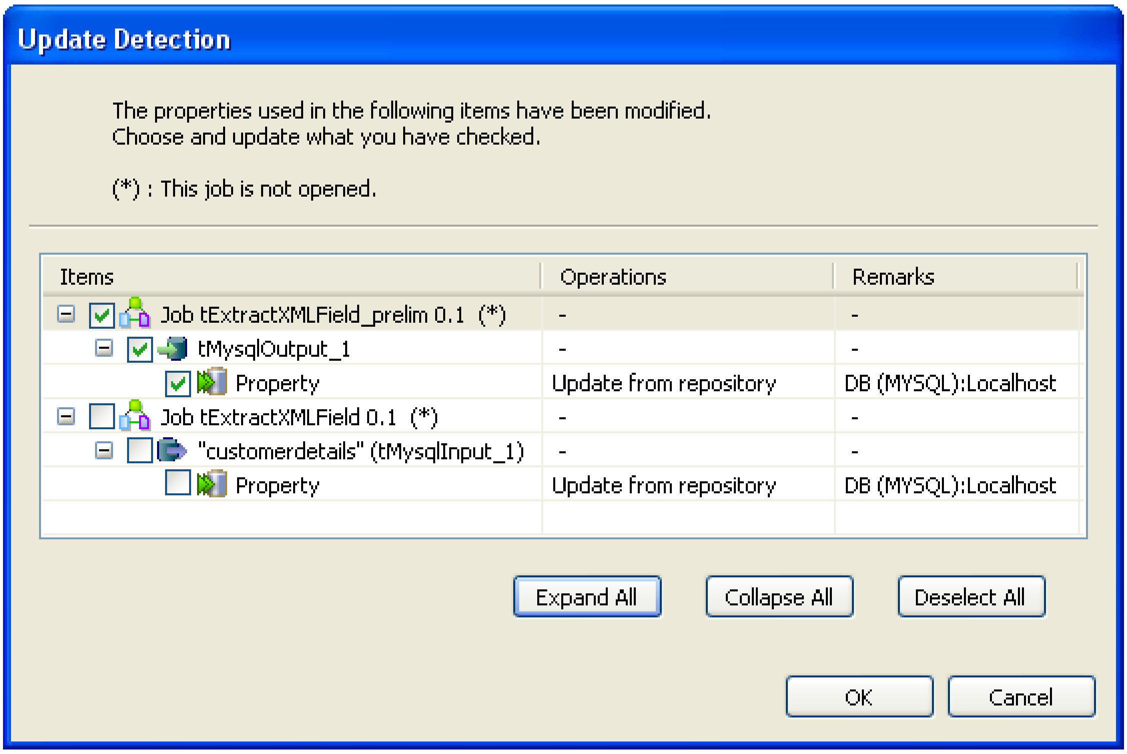
### 4.3.1.2. How to update impacted Jobs automatically

After you update the parameters of any item already centralized in the **Repository** tree view and used in different Jobs, a message will prompt you to propagate the modifications you did to all Jobs that use these parameters.

To update impacted Jobs, complete the following:

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.



Update Detection

The properties used in the following items have been modified.
Choose and update what you have checked.

(*) : This job is not opened.

| Items | Operations | Remarks |
|---|---|---|
| ☑ Job tExtractXMLField_prelim 0.1 (*) | - | - |
| ☑ tMysqlOutput_1 | - | - |
| ☑ Property | Update from repository | DB (MYSQL):Localhost |
| □ Job tExtractXMLField 0.1 (*) | - | - |
| □ "customerdetails" (tMysqlInput_1) | - | - |
| □ Property | Update from repository | DB (MYSQL):Localhost |

You can open the **[Update Detection]** dialog box any time if you right-click the item centralized in the **Repository** tree view and select **Manage Dependencies** from the contextual menu. For more information, see section *How to update impacted Jobs manually*.

2. If needed, clear the check boxes that correspond to the Jobs you do not wish to update. You can update them any time later through the **Detect Dependencies** menu. For more information, see section *How to update impacted Jobs manually*.

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

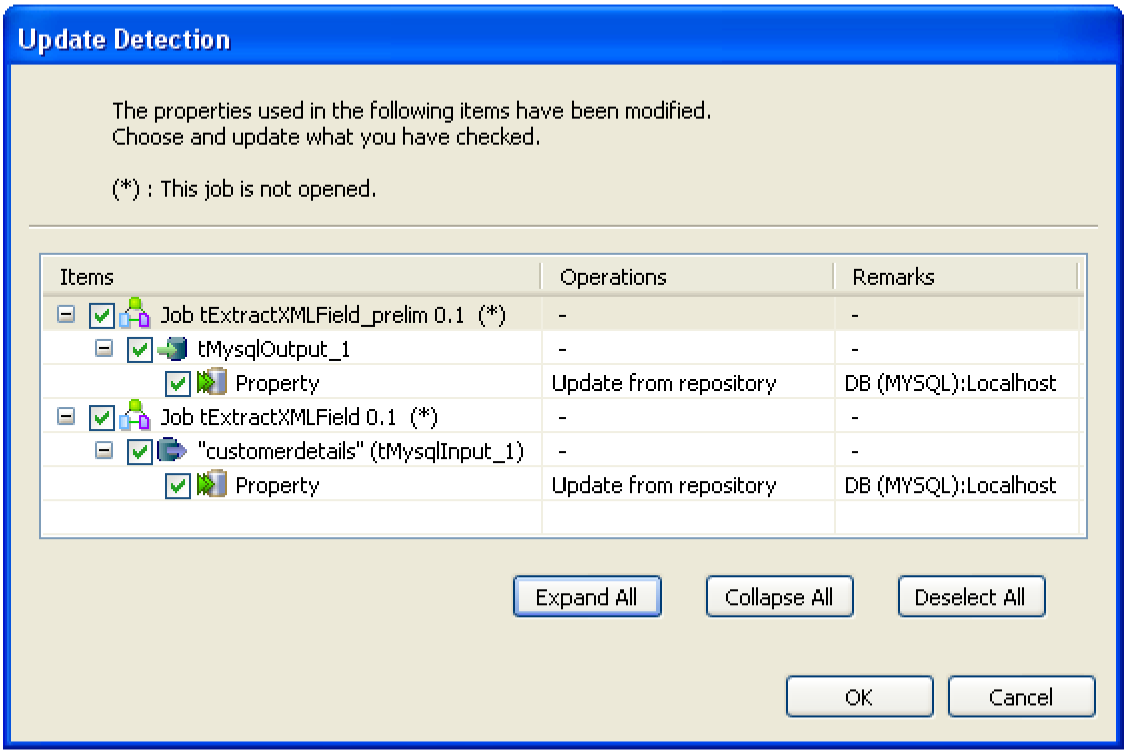
### 4.3.1.3. How to update impacted Jobs manually

Before propagating changes in the parameters of an item centralized in the tree view throughout the Jobs using this entry, you might want to view all Jobs that are impacted by the changes. To do that, complete the following:

1. In the **Repository** tree view, expand the node holding the entry you want to check what Jobs use it.

2. Right-click the entry and select **Detect Dependencies**.

   A progress bar indicates the process of checking for all Jobs that use the modified context parameter. Then a dialog box displays to list all Jobs that use the modified item.



Update Detection

The properties used in the following items have been modified.
Choose and update what you have checked.

(*) : This job is not opened.

| Items | Operations | Remarks |
|---|---|---|
| ⊟ ☑ Job tExtractXMLField_prelim 0.1 (*) | - | - |
| ⊟ ☑ tMysqlOutput_1 | - | - |
| ☑ Property | Update from repository | DB (MYSQL):Localhost |
| ⊟ ☑ Job tExtractXMLField 0.1 (*) | - | - |
| ⊟ ☑ "customerdetails" (tMysqlInput_1) | - | - |
| ☑ Property | Update from repository | DB (MYSQL):Localhost |

3. Select the check boxes corresponding to the Jobs you want to update with the context parameter and clear those corresponding to the Jobs you do not want to update.

4. Click **OK** to validate and close the dialog box.

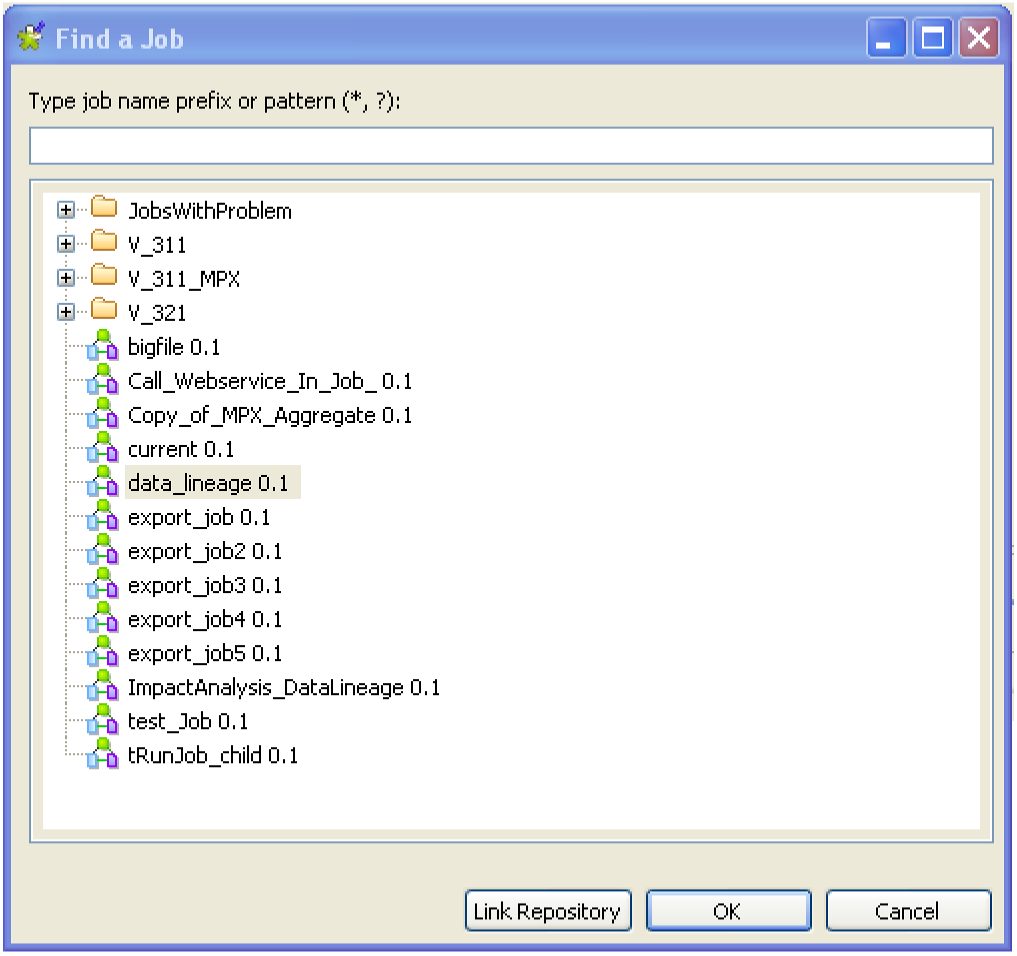
The Jobs that you choose not to update will be switched back to **Built-in**, as the link to the Repository cannot be maintained. It will thus keep their setting as it was before the change.

# 4.4. Searching a Job in the repository

If you want to open a specific Job in the **Repository** tree view of the current *Talend Open Studio for Big Data* and you can not find it for one reason or another, you can simply click 🔍 on the quick access toolbar.

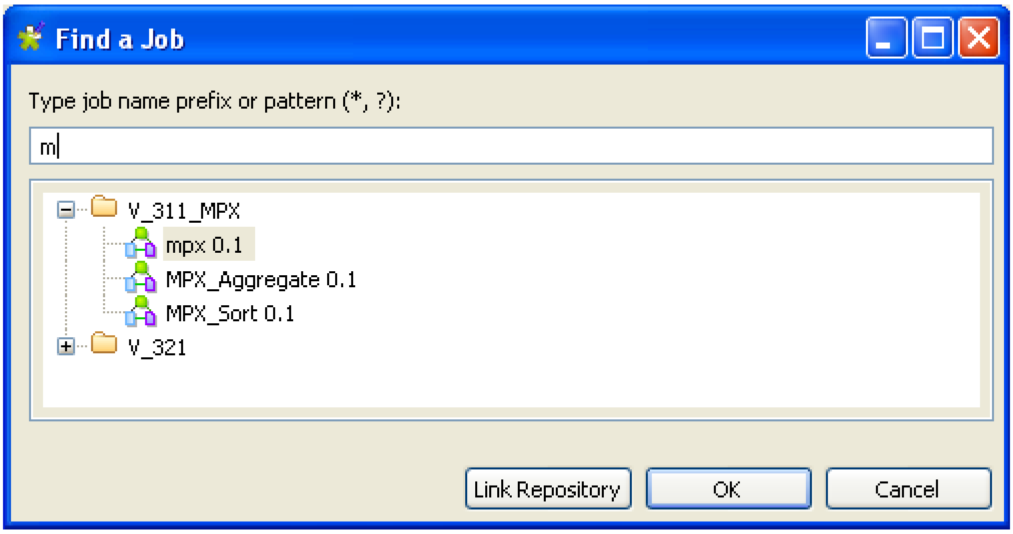
To find a Job in the **Repository** tree view, complete the following:

1. On *Talend Open Studio for Big Data* 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.

# Chapter 5. Mapping data flows

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

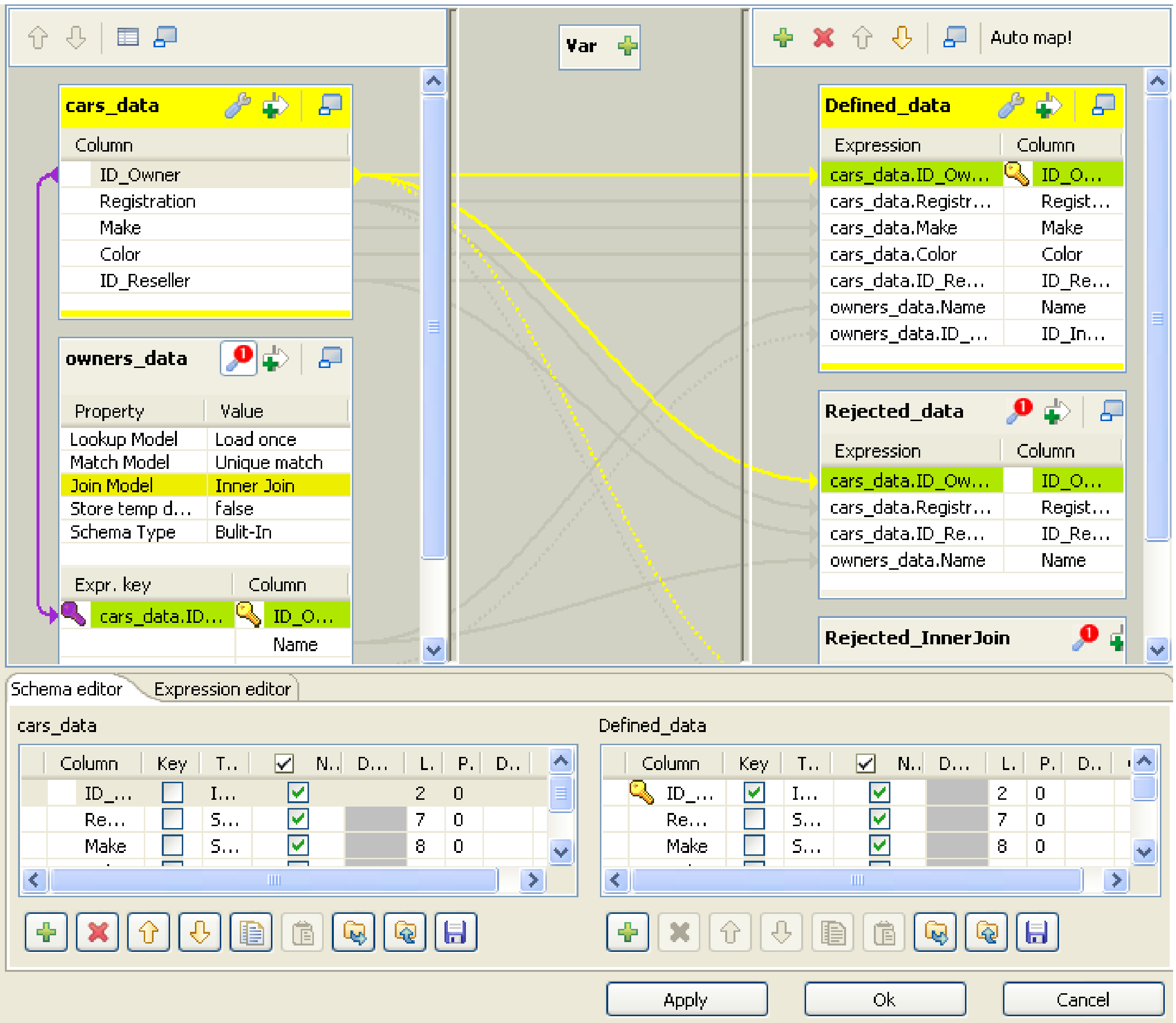
This chapter gives details separately about the usage principles of these two components, for further information or scenario and use cases, see *Talend Open Studio for Big Data Components Reference Guide*.

Before starting any data integration processes, you need to be familiar with *Talend Open Studio for Big Data* Graphical User Interface (GUI). For more information, see appendix *GUI*.

# 5.1. tMap and tXMLMap interfaces

**tMap** and **tXMLMap** 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.



cars_data

| Column |
|---|
| ID_Owner |
| Registration |
| Make |
| Color |
| ID_Reseller |

owners_data

| Property | Value |
|---|---|
| Lookup Model | Load once |
| Match Model | Unique match |
| Join Model | Inner Join |
| Store temp d... | false |
| Schema Type | Bulit-In |

| Expr. key | Column |
|---|---|
| cars_data.ID... | ID_O... |
| | Name |

Defined_data

| Expression | Column |
|---|---|
| cars_data.ID_Ow... | ID_O... |
| cars_data.Registr... | Regist... |
| cars_data.Make | Make |
| cars_data.Color | Color |
| cars_data.ID_Re... | ID_Re... |
| owners_data.Name | Name |
| owners_data.ID_... | ID_In... |

Rejected_data

| Expression | Column |
|---|---|
| cars_data.ID_Ow... | ID_O... |
| cars_data.Registr... | Regist... |
| cars_data.ID_Re... | ID_Re... |
| owners_data.Name | Name |

Rejected_InnerJoin

Schema editor | Expression editor

cars_data

| Column | Key | T.. | ☑ | N.. | D... | L. | P. | D.. |
|---|---|---|---|---|---|---|---|---|
| ID_... | □ | I... | ☑ | | | 2 | 0 | |
| Re... | □ | S... | ☑ | | | 7 | 0 | |
| Make | □ | S... | ☑ | | | 8 | 0 | |

Defined_data

| Column | Key | T.. | ☑ | N.. | D... | L. | P. | D.. |
|---|---|---|---|---|---|---|---|---|
| ID_... | ☑ | I... | ☑ | | | 2 | 0 | |
| Re... | □ | S... | ☑ | | | 7 | 0 | |
| Make | □ | S... | ☑ | | | 8 | 0 | |

Apply | Ok | Cancel

This figure presents the interface of **tMap**. That of **tXMLMap** differs 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 section *tXMLMap operation*.

The **Map Editor** is made of several panels:

- The **Input panel** is the top left panel on the editor. It offers a graphical representation of all (main and lookup) incoming data flows. The data are gathered in various columns of input tables. Note that the table name reflects the main or lookup row from the Job design on the design workspace.
- The **Variable panel** is the central panel in the **Map Editor**. It allows the centralization of redundant information through the mapping to variable and allows you to carry out transformations.
- The **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).

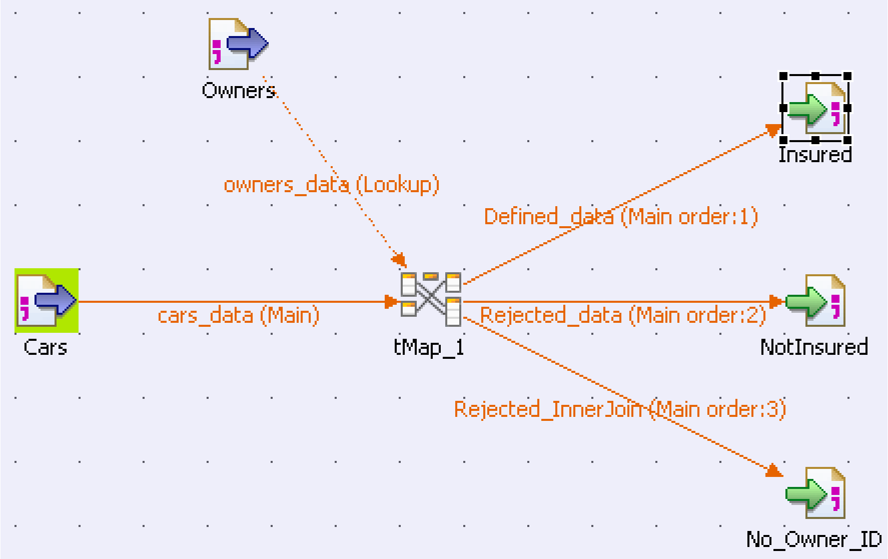
The following sections present separately **tMap** and **tXMLMap**.

# 5.2. tMap operation

**tMap** allows the following types of operations:

- data multiplexing and demultiplexing,
- data transformation on any type of fields,
- fields concatenation and interchange,
- field filtering using constraints,
- data rejecting.

As all these operations of transformation and/or routing are carried out by **tMap**, this component cannot be a start or end component in the Job design.


Owners
owners_data (Lookup)
Insured
Defined_data (Main order:1)
Cars
cars_data (Main)
tMap_1
Rejected_data (Main order:2)
NotInsured
Rejected_InnerJoin (Main order:3)
No_Owner_ID

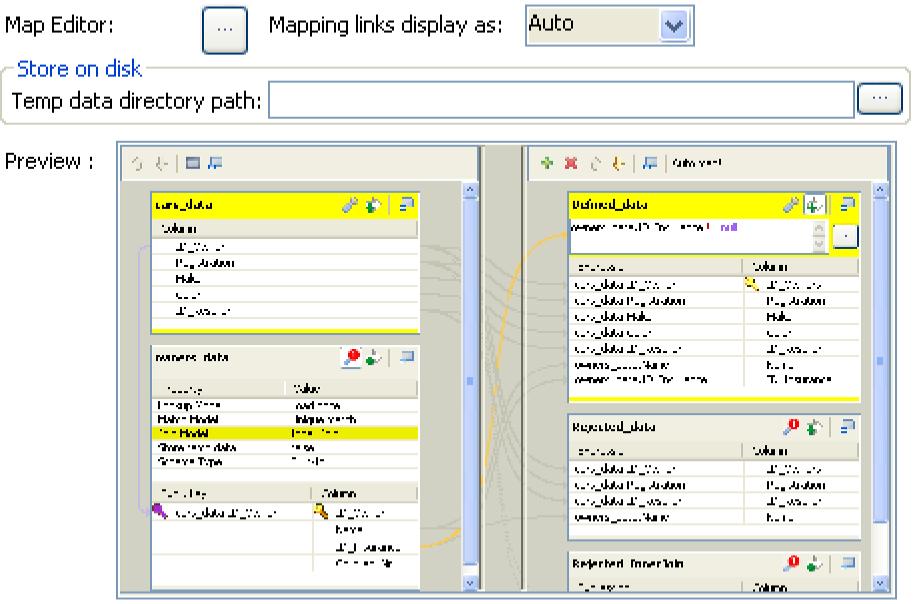
**tMap** uses incoming connections to pre-fill input schemas with data in the **Map Editor**. Therefore, you cannot create new input schemas directly in the **Map Editor**. Instead, you need to implement as many **Row** connections incoming to **tMap** component as required, in order to create as many 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: section *Row connection*.

Lookup rows are incoming connections from secondary (or reference) flows of data. These reference data might depend directly or indirectly on the primary flow. This dependency relationship is translated with a graphical mapping and the creation of an expression key.

The **Map Editor** requires the connections to be implemented in your Job in order to be able to define the input and output flows in the **Map Editor**. You also need to create the actual mapping in your Job in order to display the **Map Editor** in the **Preview** area of the **Basic settings** view of the **tMap** component.



Map Editor: ...

Mapping links display as: Auto

Store on disk

Temp data directory path: ...

Preview :

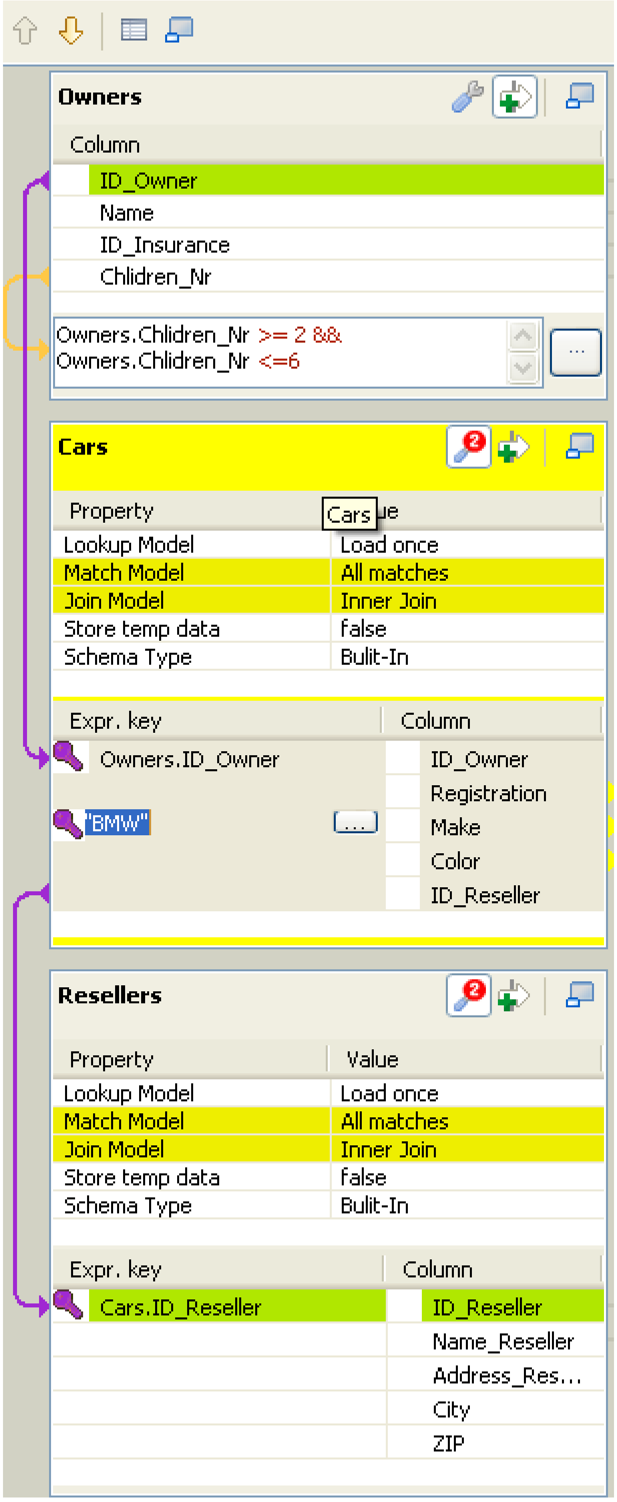
To open the **Map Editor** in a new window, double-click the **tMap** icon in the design workspace or click the three-dot button next to the **Map Editor** in the **Basic settings** view of the **tMap** component.

The following sections give the information necessary to use the **tMap** component in any of your Job designs.

## 5.2.1. Setting the input flow in the Map Editor

The order of the **Input** tables is essential. The top table reflects the **Main** flow connection, and for this reason, is given priority for reading and processing through the **tMap** component.

For this priority reason, you are not allowed to move up or down the **Main** flow table. This ensures that no Join can be lost.



**Owners**

| Column |
|---|
| ID_Owner |
| Name |
| ID_Insurance |
| Chlidren_Nr |

```
Owners.Chlidren_Nr >= 2 &&
Owners.Chlidren_Nr <=6
```

**Cars**

| Property | Cars ue |
|---|---|
| Lookup Model | Load once |
| Match Model | All matches |
| Join Model | Inner Join |
| Store temp data | false |
| Schema Type | Bulit-In |

| 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 |
| Schema Type | Bulit-In |

| Expr. key | Column |
|---|---|
| Cars.ID_Reseller | ID_Reseller |
| | Name_Reseller |
| | 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: section *How to use Explicit Join*.

### 5.2.1.1. How to fill in Input tables with a schema

To fill in the input tables, you need to define either the schemas of the input components connected to the **tMap** component on your design workspace, or the input schemas within the **Map Editor**.

For more information about setting a component schema, see section *How to define component properties*.

For more information about setting an input schema in the **Map Editor**, see section *Setting schemas in the Map Editor*.

#### Main and Lookup table content

The order of the **Input** tables is essential.

The **Main Row** connection determines the **Main** flow table content. This input flow is reflected in the first table of the **Map Editor's** Input panel.

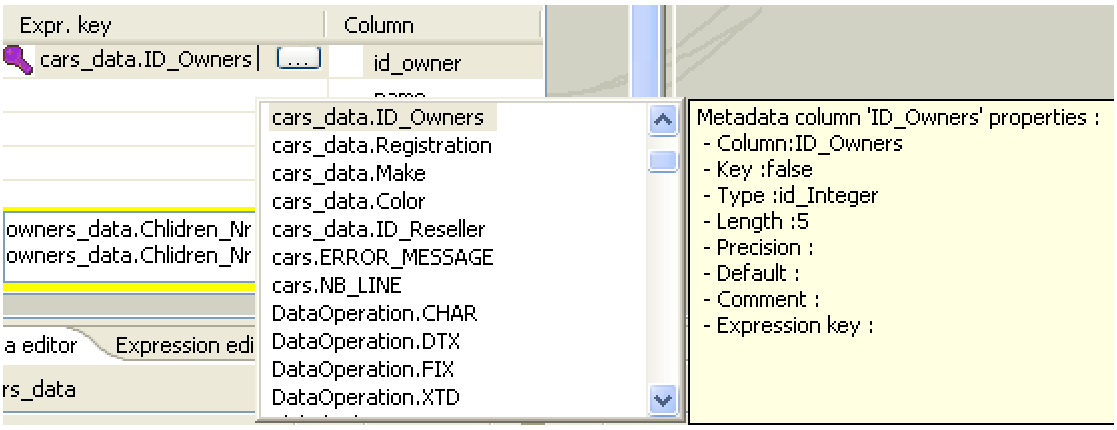
The **Lookup** connections' content fills in all other (secondary or subordinate) tables which displays below the **Main** flow table. If you have not define the schema of an input component yet, the input table displays as empty in the Input area.

The key is also retrieved from the schema defined in the Input component. This **Key** corresponds to the key defined in the input schema where relevant. It has to be distinguished from the hash key that is internally used in the **Map Editor**, which displays in a different color.

#### Variables

You can use global or context variables or reuse the variable defined in the **Variables** area. Press **Ctrl+Space bar** to access the list of variables. This list gathers together global, context and mapping variables.

The list of variables changes according to the context and grows along new variable creation. Only valid mappable variables in the context show on the list.



Metadata column 'ID_Owners' properties :
- Column:ID_Owners
- Key :false
- Type :id_Integer
- Length :5
- Precision :
- Default :
- Comment :
- Expression key :

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

Related topic: section *Mapping variables*

### 5.2.1.2. How to use Explicit Join

In fact, **Joins** let you select data from a table depending upon the data from another table. In the **Map Editor** context, the data of a **Main** table and of a **Lookup** table can be bound together on **expression keys**. In this case, the order of table does fully make sense.

Simply drop column names from one table to a subordinate one, to create a **Join** relationship between the two tables. This way, you can retrieve and process data from multiple inputs.

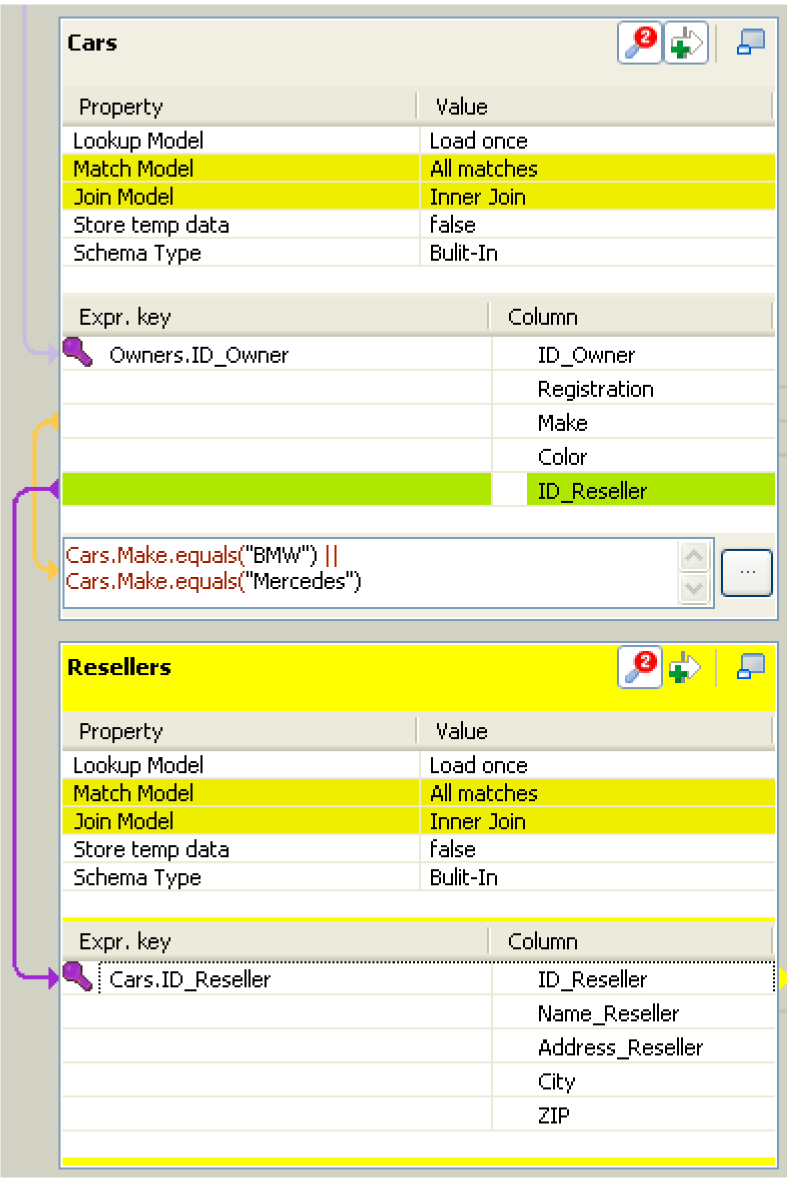
The join displays graphically as a purple link and creates automatically a key that will be used as a hash key to speed up the match search.

You can create direct joins between the main table and lookup tables. But you can also create indirect joins from the main table to a lookup table, via another lookup table. This requires a direct join between one of the **Lookup** table to the **Main** one.

You cannot create a **Join** from a subordinate table towards a superior table in the **Input** area.

The **Expression key** field which is filled in with the dragged and dropped data is editable in the input schema, whereas the column name can only be changed from the **Schema editor** panel.

You can either insert the dragged data into a new entry or replace the existing entries or else concatenate all selected data into one cell.



Cars

| Property | Value |
|---|---|
| Lookup Model | Load once |
| Match Model | All matches |
| Join Model | Inner Join |
| Store temp data | false |
| Schema Type | Bulit-In |

| Expr. key | Column |
|---|---|
| Owners.ID_Owner | ID_Owner |
| | Registration |
| | Make |
| | Color |
| | ID_Reseller |

```
Cars.Make.equals("BMW") ||
Cars.Make.equals("Mercedes")
```

Resellers

| Property | Value |
|---|---|
| Lookup Model | Load once |
| Match Model | All matches |
| Join Model | Inner Join |
| Store temp data | false |
| Schema Type | Bulit-In |

| Expr. key | Column |
|---|---|
| Cars.ID_Reseller | ID_Reseller |
| | Name_Reseller |
| | Address_Reseller |
| | City |
| | ZIP |

For further information about possible types of drag and drops, see section *Mapping the Output setting* .

If you have a big number of input tables, you can use the minimize/maximize icon to reduce or restore the table size in the **Input** area. The Join binding two tables remains visible even though the table is minimized.

Creating a Join automatically assigns a hash key onto the joined field name. The key symbol displays in violet on the input table itself and is removed when the Join between the two tables is removed.

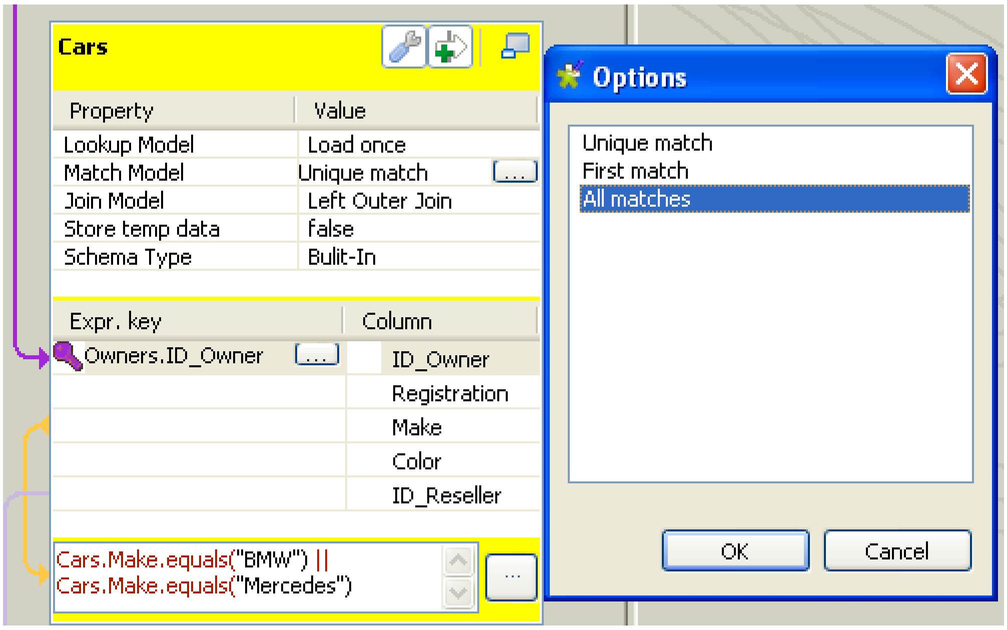
Related topics:

- section *Setting schemas in the Map Editor*
- section *How to use Inner Join*

Along with the explicit Join you can select whether you want to filter down to a unique match or if you allow several matches to be taken into account. In this last case, you can choose to consider only the first or the last match or all of them.

To define the match model for an explicit Join:

1. Click the **tMap settings** button at the top of the table to which the Join links to display the table properties.
2. Click in the **Value** field corresponding to **Match Model** and then click the three-dot button that appears to open the **[Options]** dialog box.
3. In the **[Options]** dialog box, double-click the wanted match model, or select it and click **OK** to validate the setting and close the dialog box.



Cars

| Property | Value |
|---|---|
| Lookup Model | Load once |
| Match Model | Unique match |
| Join Model | Left Outer Join |
| Store temp data | false |
| Schema Type | Bulit-In |

| Expr. key | Column |
|---|---|
| Owners.ID_Owner | ID_Owner |
| | Registration |
| | Make |
| | Color |
| | ID_Reseller |

```
Cars.Make.equals("BMW") ||
Cars.Make.equals("Mercedes")
```

## 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.

## 5.2.1.3. How to use Inner Join

The **Inner join** is a particular type of Join that distinguishes itself by the way the rejection is performed.

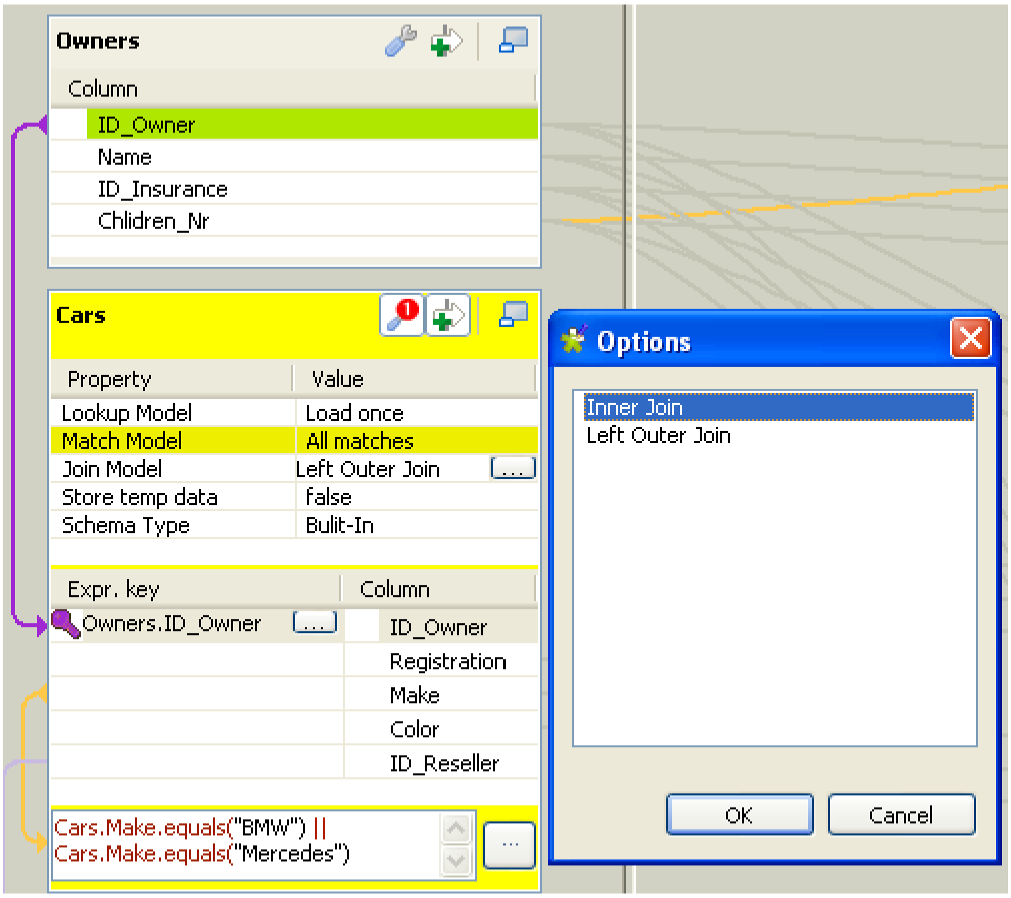
This option avoids that null values are passed on to the main output flow. It allows also to pass on the rejected data to a specific table called **Inner Join Reject** table.

If the data searched cannot be retrieved through the explicit Join or the filter Join, in other words, the Inner Join cannot be established for any reason, then the requested data will be rejected to the Output table defined as **Inner Join Reject** table if any.

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

To define the type of an explicit Join:

1. Click the **tMap settings** button at the top of the table to which the Join links to display the table properties.
2. Click in the **Value** field corresponding to **Join Model** and then click the three-dot button that appears to open the **[Options]** dialog box.
3. In the **[Options]** dialog box, double-click the wanted Join type, or select it and click **OK** to validate the setting and close the dialog box.



Owners

| Column |
|---|
| ID_Owner |
| Name |
| ID_Insurance |
| Chlidren_Nr |

Cars

| Property | Value |
|---|---|
| Lookup Model | Load once |
| Match Model | All matches |
| Join Model | Left Outer Join |
| Store temp data | false |
| Schema Type | Bulit-In |

| Expr. key | Column |
|---|---|
| Owners.ID_Owner | ID_Owner |
| | Registration |
| | Make |
| | Color |
| | ID_Reseller |

```
Cars.Make.equals("BMW") ||
Cars.Make.equals("Mercedes")
```

Options

- Inner Join
- Left Outer Join

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 section *Lookup Inner Join rejection*.

You can also use the filter button to decrease the number of rows to be searched and improve the performance (in Java).

Related topics:

- section *Lookup Inner Join rejection*
- section *How to filter an input flow*

## 5.2.1.4. How to use the All Rows option

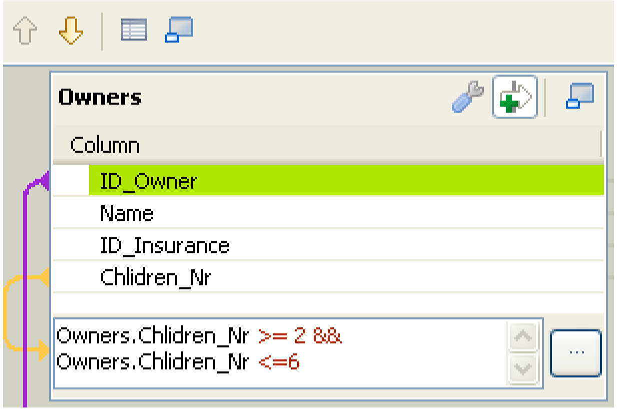
By default, without a Join set up, in each input table of the input area of the **Map Editor**, the **All rows** match model option is selected. This **All rows** option means that all the rows are loaded from the **Lookup** flow and searched against the **Main** flow.

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

If you create an explicit or an inner Join between two tables, the **All rows** option is no longer available. You then have to select **Unique match**, **First match** or **All matches**. For more information, see section *How to use Explicit Join* and section *How to use Inner Join*.

## 5.2.1.5. How to filter an input flow

Click the **Filter** button next to the **tMap settings** button to add a **Filter** field.



```
Owners.Chlidren_Nr >= 2 &&
Owners.Chlidren_Nr <=6
```

In the **Filter** field, type in the condition to be applied. This allows to reduce the number of rows parsed against the main flow, enhancing the performance on long and heterogeneous flows.

You can use the Auto-completion tool via the **Ctrl+Space bar** keystrokes in order to reuse schema columns in the condition statement.

## 5.2.1.6. How to remove input entries from table

To remove input entries, click the red cross sign on the Schema Editor of the selected table. Press **Ctrl** or **Shift** and click fields for multiple selection to be removed.

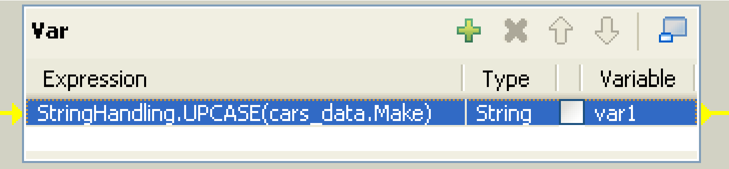
If you remove Input entries from the **Map Editor** schema, this removal also occurs in your component schema definition.

## 5.2.2. Mapping variables

The **Var** table (variable table) regroups all mapping variables which are used numerous times in various places.

You can also use the **Expression** field of the **Var** table to carry out any transformation you want to, using Java Code.

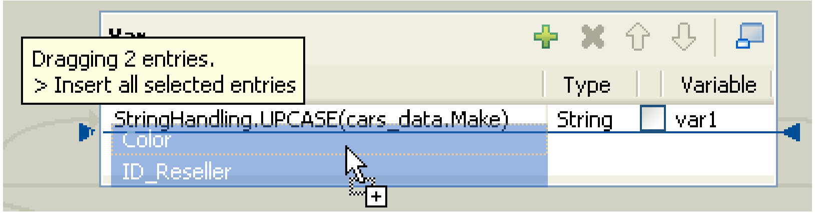
Variables help you save processing time and avoid you to retype many times the same data.



| Expression | Type | Variable |
|---|---|---|
| StringHandling.UPCASE(cars_data.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.


Dragging 2 entries.
> Insert all selected entries

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 **Ctrl** 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 **Shift** 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. |

### 5.2.2.1. How to access global or context variables

Press **Ctrl+Space** to access the global and context variable list.

Appended to the variable list, a metadata list provides information about the selected column.

### 5.2.2.2. How to remove variables

To remove a selected **Var** entry, click the red cross sign. This removes the whole line as well as the link.

Press **Ctrl** or **Shift** and click fields for multiple selection then click the red cross sign.

## 5.2.3. Using the expression editor

All expressions (**Input**, **Var** or **Output**) and constraint statements can be viewed and edited from the expression editor. This editor provides visual comfort to write any function or transformation in a handy dedicated view.

### 5.2.3.1. How to access the expression editor

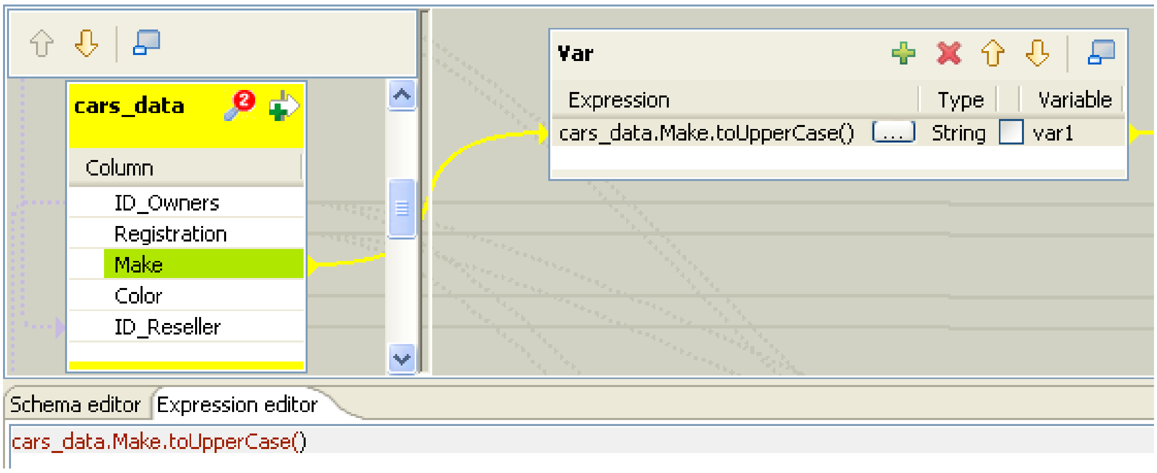
You can write the expressions necessary for the data transformation directly in the **Expression editor** view located in the lower half of the expression editor, or you can open the **[Expression Builder]** dialog box where you can write the data transformation expressions.

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

1. Double-click the **tMap** component in your job design to open the **Map Editor**.

2. In the lower half of the editor, click the **Expression editor** tab to open the corresponding view.

To edit an expression, select it in the **Input** panel and then click the **Expression editor** tab and modify the expression as required.



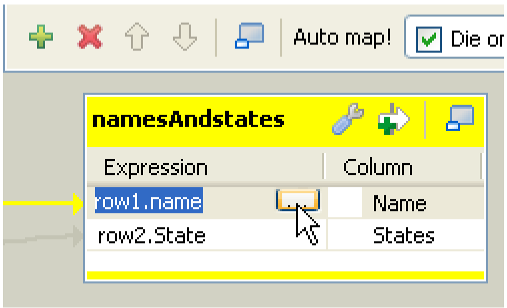
| Expression | Type | Variable |
|---|---|---|
| cars_data.Make.toUpperCase() [...] | String | var1 |

```
cars_data.Make.toUpperCase()
```

3. Enter the Java code according to your needs. The corresponding expression in the output panel is synchronized.

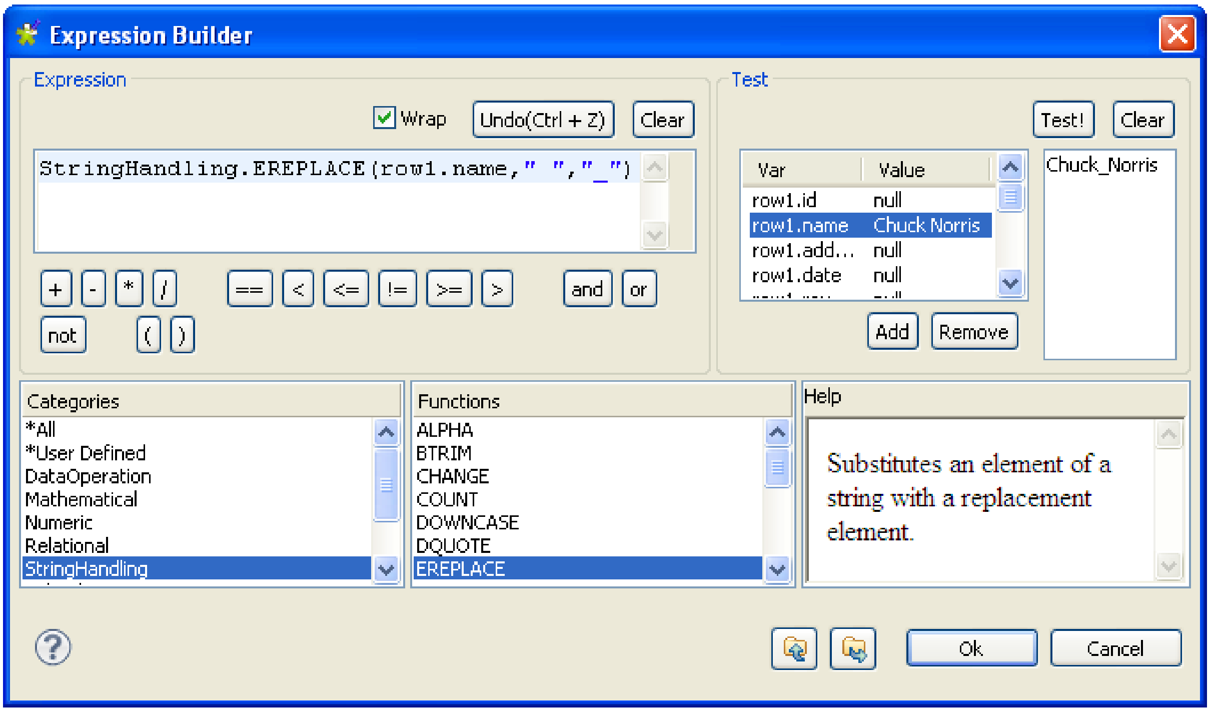
Refer to the Java documentation for more information regarding functions and operations.

To open the **[Expression Builder]** dialog box, click the three-dot button next to the expression you want to open in the **Var** or **Output** panel of the **Map Editor**.



| Expression | Column |
|---|---|
| row1.name | Name |
| row2.State | States |

The **[Expression Builder]** dialog box opens on the selected expression.



```
StringHandling.EREPLACE(row1.name," ","_")
```

| Var | Value |
|---|---|
| row1.id | null |
| row1.name | Chuck Norris |
| row1.add... | null |
| row1.date | null |

```
Chuck_Norris
```

Categories: *All, *User Defined, DataOperation, Mathematical, Numeric, Relational, StringHandling

Functions: ALPHA, BTRIM, CHANGE, COUNT, DOWNCASE, DQUOTE, EREPLACE

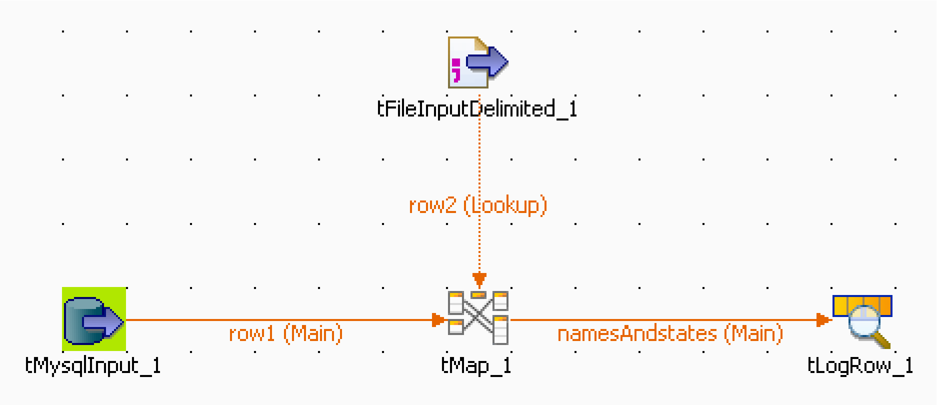
Help: Substitutes an element of a string with a replacement element.

For a use case showing the usage of the expression editor, see the following section.

## 5.2.3.2. How to write code using the Expression Builder

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

The following example shows the use of **Expression Builder** in a **tMap** component.



Two input flows are connected to the **tMap** component.

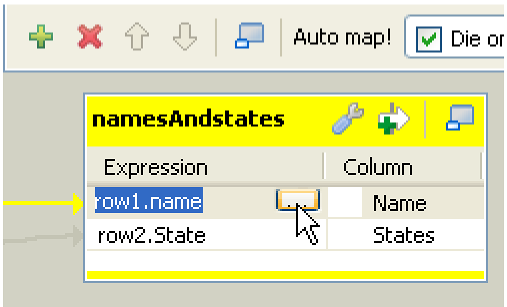
- From the DB input, comes a list of names made of a first name and a last name separated by a space char.
- From the File input, comes a list of US states, in lower case.

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

1. In the **tMap**, set the relevant inner join to set the reference mapping. For more information regarding **tMap**, see section *tMap operation* and section *tMap and tXMLMap interfaces*.

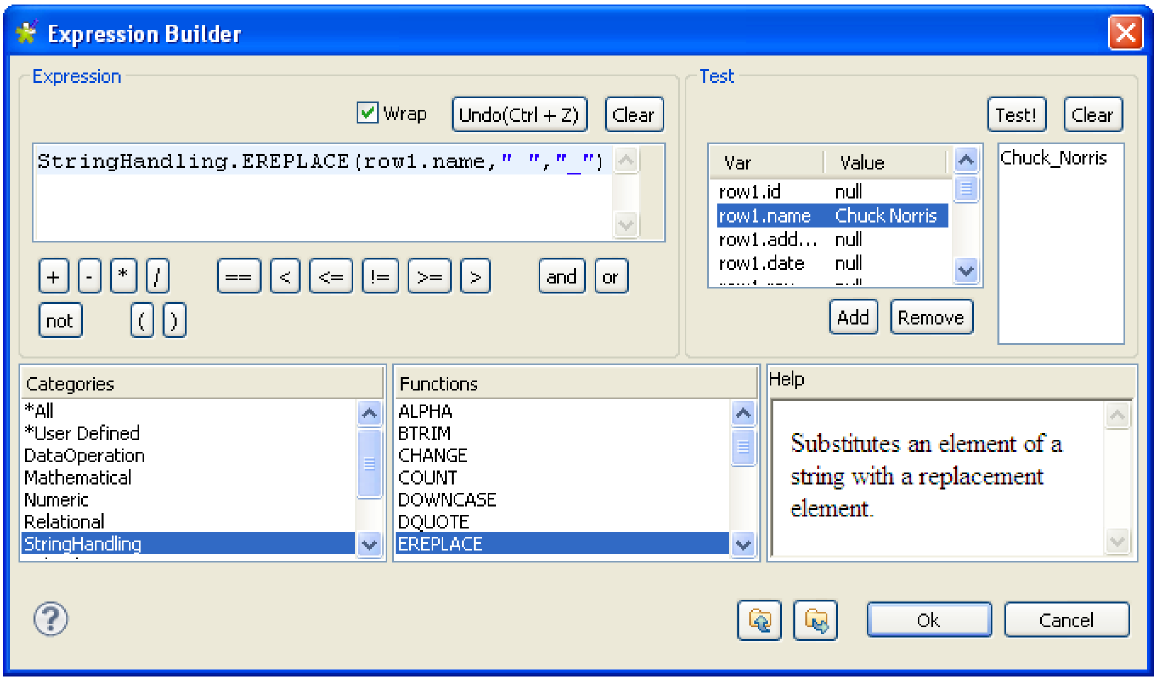
2. From the main (*row1*) input, drop the *Names* column to the output area, and the *State* column from the lookup (*row2*) input towards the same output area.

3. Then click in the first **Expression** field (*row1.Name*) to display the three-dot button.



| Expression | Column |
|---|---|
| row1.name | Name |
| row2.State | States |

The **[Expression Builder]** dialog box opens up.



```
StringHandling.EREPLACE(row1.name," ","_")
```

| Var | Value |
|---|---|
| row1.id | null |
| row1.name | Chuck Norris |
| row1.add... | null |
| row1.date | null |

Chuck_Norris

Substitutes an element of a string with a replacement element.

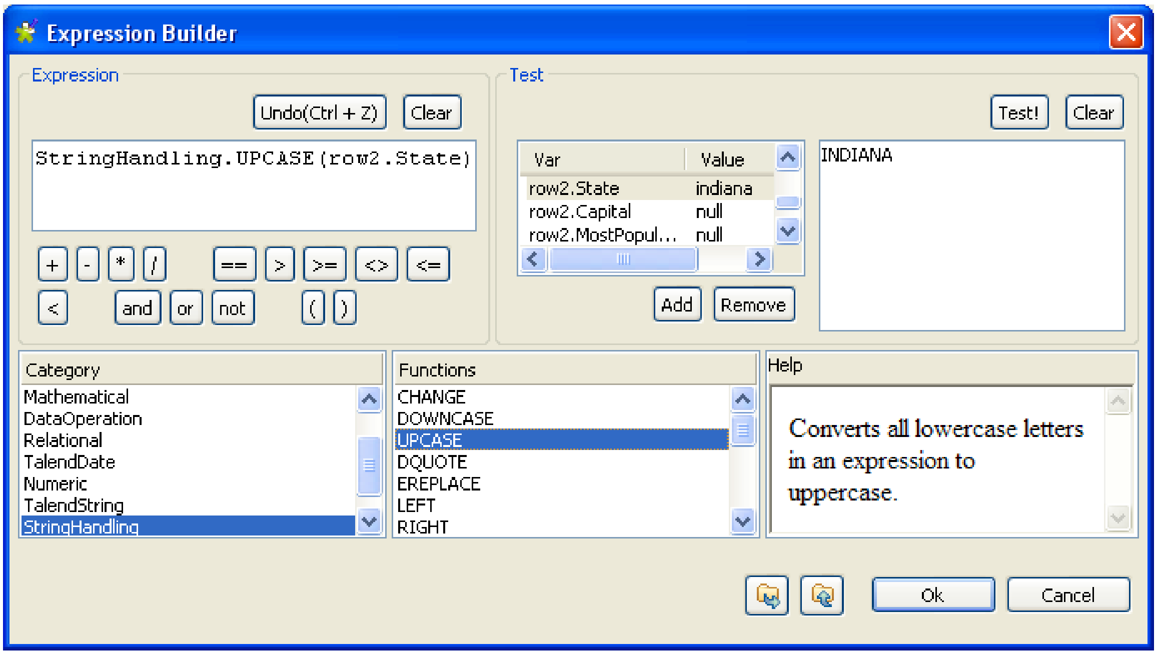
4. In the **Category** area, select the relevant action you want to perform. In this example, select `StringHandling` and select the `EREPLACE` function.

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.

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.



StringHandling.UPCASE(row2.State)

| Var | Value |
|---|---|
| row2.State | indiana |
| row2.Capital | null |
| row2.MostPopul... | null |

INDIANA

Category: Mathematical, DataOperation, Relational, TalendDate, Numeric, TalendString, StringHandling

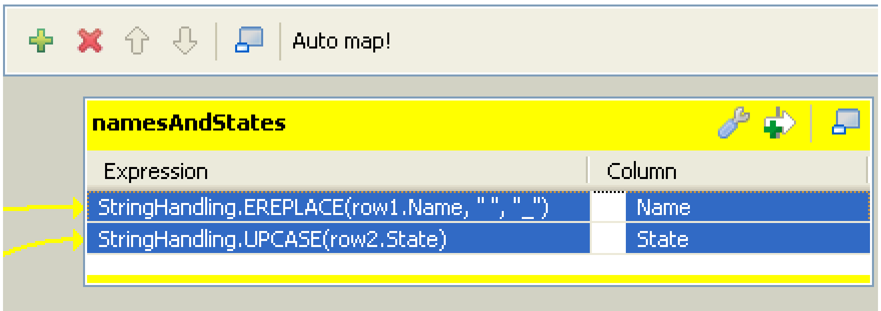
Functions: CHANGE, DOWNCASE, UPCASE, DQUOTE, EREPLACE, LEFT, RIGHT

Help: Converts all lowercase letters in an expression to uppercase.

This time, the `StringHandling` function to be used is `UPCASE`. The complete expression says: `StringHandling.UPCASE(row2.State)`.

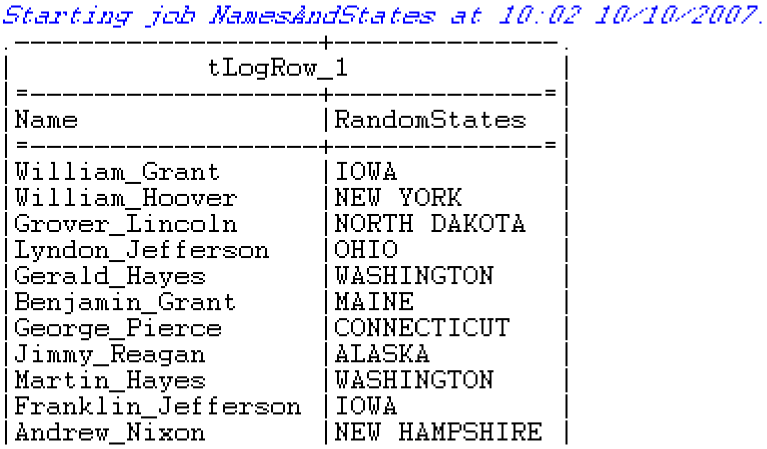
9. Once again, check that the expression syntax is correct using a dummy **Value** in the **Test** area, for example *indiana*. The **Test!** result should display *INDIANA* for this example. Then, click **OK** to validate the changes.

   Both expressions are now displayed in the **tMap Expression** field.



| Expression | Column |
|---|---|
| StringHandling.EREPLACE(row1.Name, " ", "_") | Name |
| StringHandling.UPCASE(row2.State) | State |

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 |
```

# 5.2.4. Mapping the Output setting

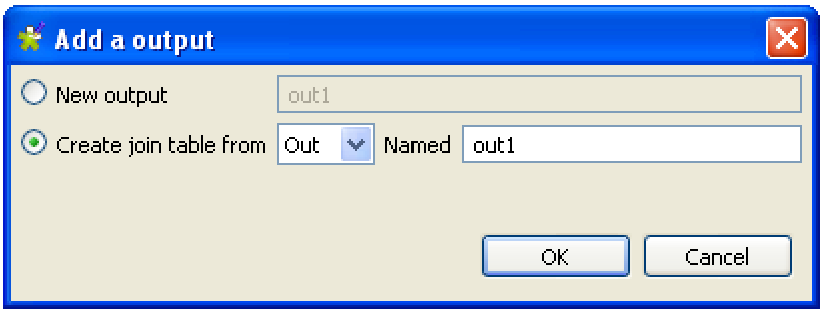
On the design workspace, the creation of a **Row** connection from the **tMap** component to the output components adds Output schema tables in the **Map Editor**.

You can also add an Output schema in your **Map Editor**, using the plus sign from the tool bar of the Output area.

You have as well the possibility to create a join between your output tables. The join on the tables enables you to process several flows separately and unite them in a single output. For more information about the output join tables feature, see *Talend Open Studio for Big Data Components Reference Guide*.

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.



Add a output

○ New output: out1

● Create join table from: Out  Named: out1

| 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 **Named** 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.

### 5.2.4.1. Building complex expressions

If you have complex expressions to build, or advanced changes to be carried out on the output flow, then the Expression Builder interface can help in this task.

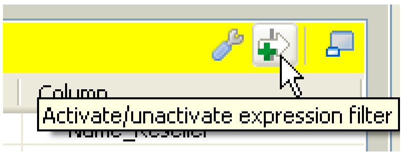
Click the **Expression** field of your input or output table to display the **[...]** button. Then click this three-dot button to open the **Expression Builder**.

For more information regarding the Expression Builder, see section *How to write code using the Expression Builder*.

### 5.2.4.2. Filters

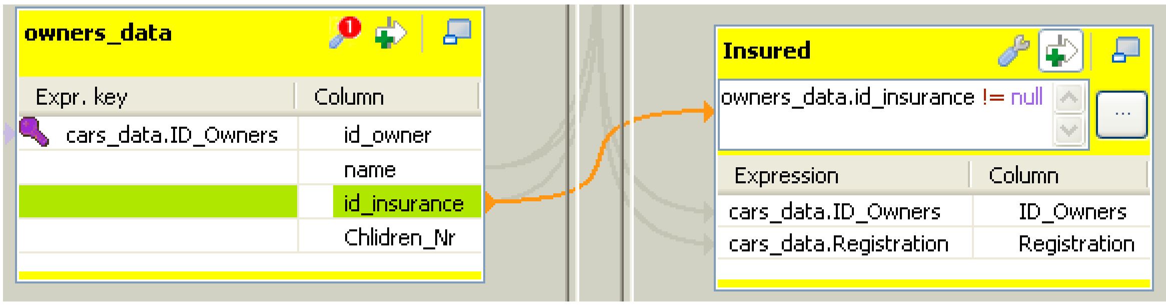
Filters allow you to make a selection among the input fields, and send only the selected fields to various outputs.

Click the **[+]** button at the top of the table to add a filter line.



You can enter freely your filter statements using Java operators and functions.

Drop expressions from the **Input** area or from the **Var** area to the Filter row entry of the relevant Output table.



| Expr. key | Column |
|---|---|
| cars_data.ID_Owners | id_owner |
| | name |
| | id_insurance |
| | Chlidren_Nr |

owners_data.id_insurance != null

| Expression | Column |
|---|---|
| cars_data.ID_Owners | ID_Owners |
| cars_data.Registration | Registration |

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.

### 5.2.4.3. Output rejection

Reject options define the nature of an output table.

It groups data which do not satisfy one or more filters defined in the standard output tables. Note that as standard output tables, are meant all non-reject tables.

This way, data rejected from other output tables, are gathered in one or more dedicated tables, allowing you to spot any error or unpredicted case.

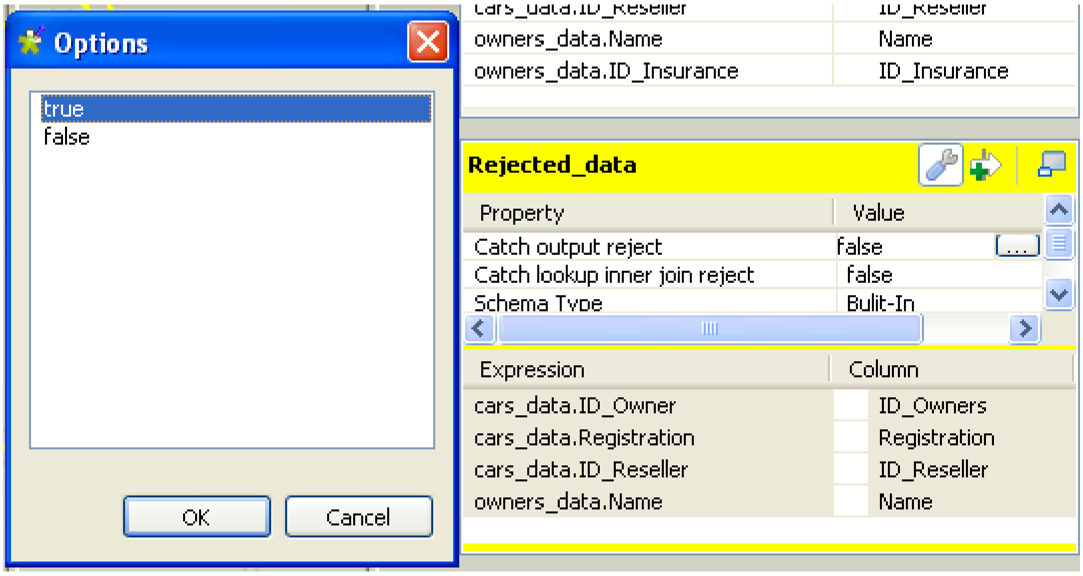
The Reject principle concatenates all non Reject tables filters and defines them as an ELSE statement.

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

1. Click the **tMap settings** button at the top of the output table to display the table properties.

2. Click in the **Value** field corresponding to **Catch output reject** and then click the **[...]** button that appears to display the **[Options]** dialog box.

3. In the **[Options]** dialog box, double-click **true**, or select it and click **OK** to validate the setting and close the dialog box.



| | |
|---|---|
| cars_data.ID_Reseller | ID_Reseller |
| owners_data.Name | Name |
| owners_data.ID_Insurance | ID_Insurance |

| Property | Value |
|---|---|
| Catch output reject | false |
| Catch lookup inner join reject | false |
| Schema Type | Bulit-In |

| Expression | Column |
|---|---|
| cars_data.ID_Owner | ID_Owners |
| cars_data.Registration | Registration |
| cars_data.ID_Reseller | ID_Reseller |
| owners_data.Name | Name |

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.

## 5.2.4.4. Lookup Inner Join rejection

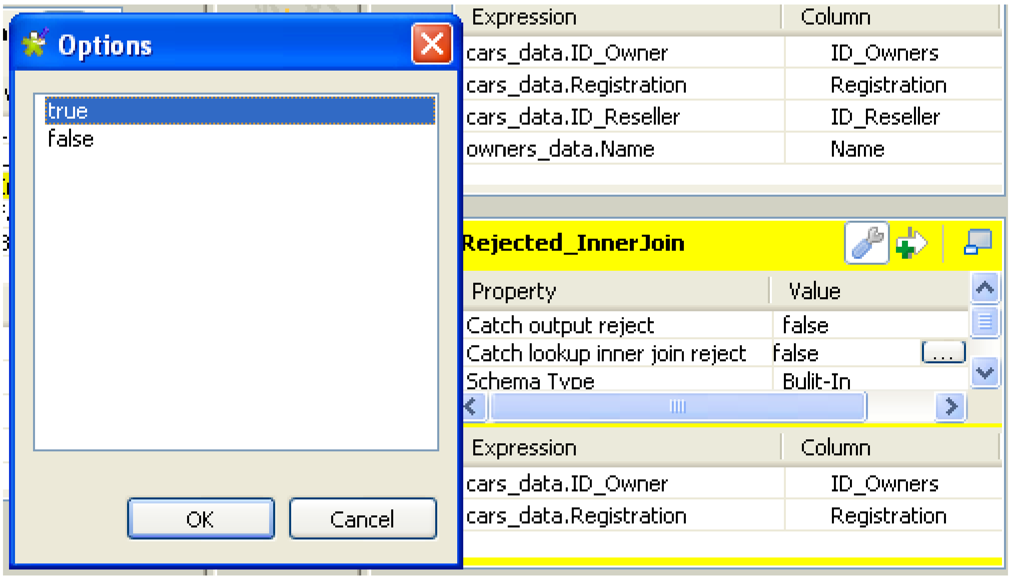
The Inner Join is a Lookup Join. The Inner Join Reject table is a particular type of Rejection output. It gathers rejected data from the main row table after an Inner Join could not be established.

To define an Output flow as container for rejected Inner Join data, create a new output component on your Job that you connect to the **Map Editor**. Then in the **Map Editor**, follow the steps below:

1. Click the **tMap settings** button at the top of the output table to display the table properties.

2. Click in the **Value** field corresponding to **Catch lookup inner join reject** and then click the **[...]** button that appears to display the **[Options]** dialog box.

3. In the **[Options]** dialog box, double-click **true**, or select it and click **OK** to validate the setting and close the dialog box.



| Expression | Column |
|---|---|
| cars_data.ID_Owner | ID_Owners |
| cars_data.Registration | Registration |
| cars_data.ID_Reseller | ID_Reseller |
| owners_data.Name | Name |

| Property | Value |
|---|---|
| Catch output reject | false |
| Catch lookup inner join reject | false |
| Schema Type | Bulit-In |

| Expression | Column |
|---|---|
| cars_data.ID_Owner | ID_Owners |
| cars_data.Registration | Registration |

## 5.2.4.5. Removing Output entries

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

## 5.2.4.6. Handling errors

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

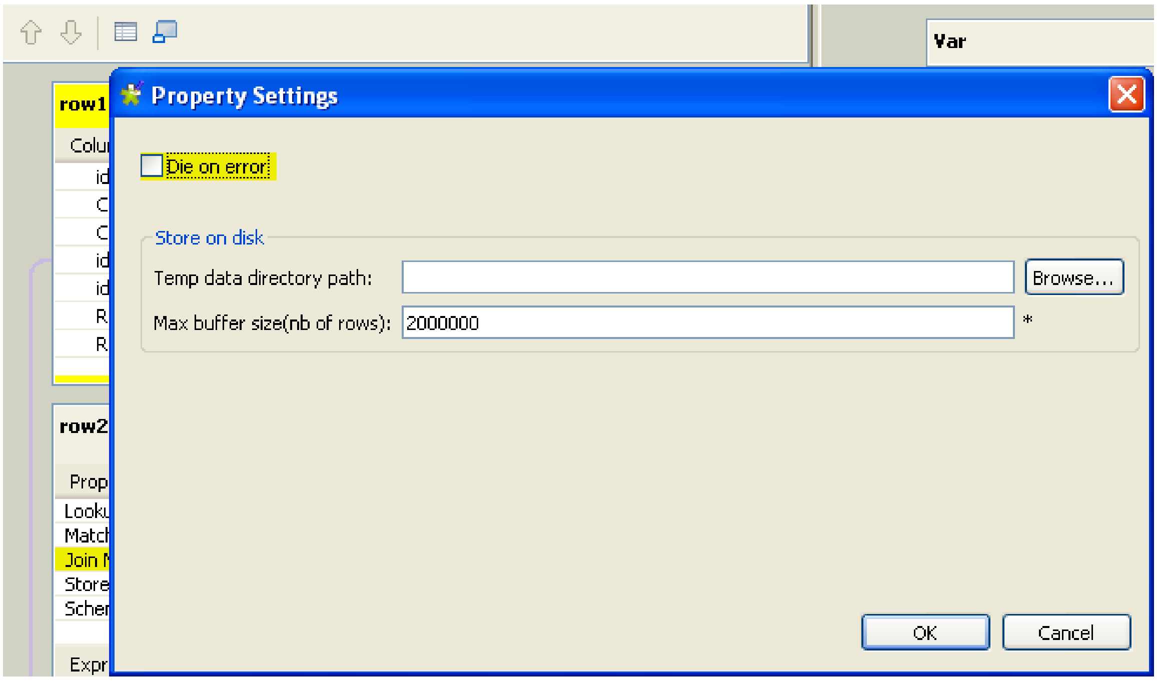
Deactivating the **Die on error** option will allow you to skip the rows on error and complete the process for error-free rows on one hand, and to retrieve the rows on error and manage them if needed.

To deactivate the **Die on error** option:

1. Double-click the **tMap** component on the design workspace to open the **Map Editor**.

2. Click the **Property Settings** button at the top of the input area to display the **[Property Settings]** dialog box.

3. In **[Property Settings]** dialog box, clear the **Die on error** check box and click **OK**.



Property Settings

□ Die on error

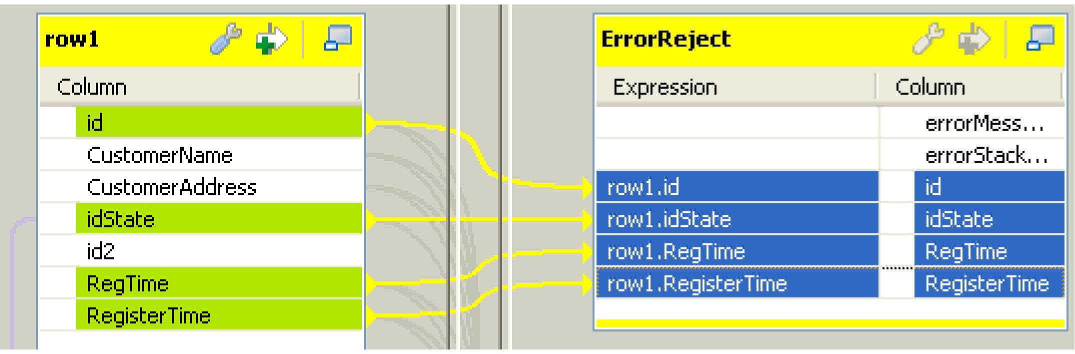
Store on disk

Temp data directory path:

Max buffer size(nb of rows): 2000000 *

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.



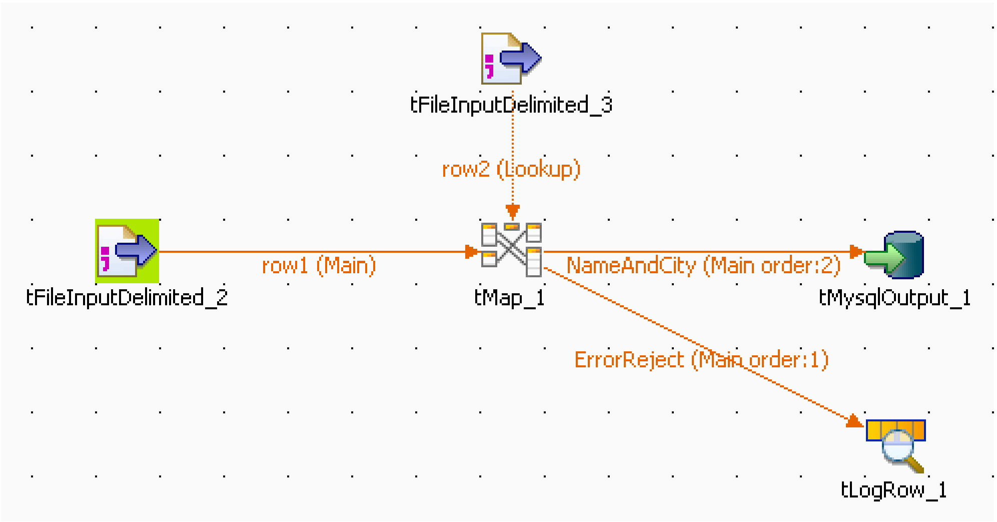
row1

| Column |
|---|
| id |
| CustomerName |
| CustomerAddress |
| idState |
| id2 |
| RegTime |
| RegisterTime |

ErrorReject

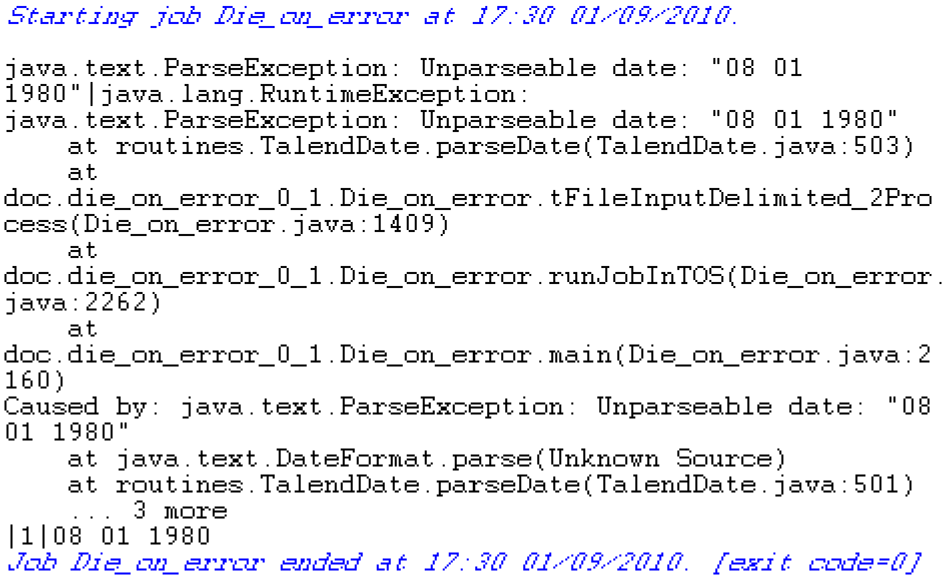
| Expression | Column |
|---|---|
| | errorMess... |
| | errorStack... |
| row1.id | id |
| row1.idState | idState |
| row1.RegTime | RegTime |
| row1.RegisterTime | RegisterTime |

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]
```

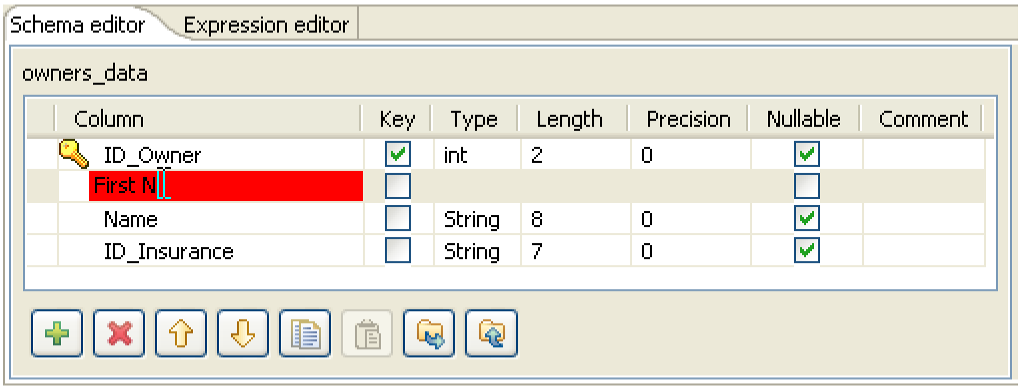
The result contains the error message, its stack trace, and the two columns, *id* and *date*, dragged and dropped to the **ErrorReject** table, separated by a pipe “|”.

## 5.2.5. Setting schemas in the Map Editor

In the **Map Editor**, you can define the type of a table schema as **Built-In** so that you can modify the data structure in the **Schema editor** panel.

### 5.2.5.1. 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.



owners_data

| Column | Key | Type | Length | Precision | Nullable | Comment |
|---|---|---|---|---|---|---|
| ID_Owner | ☑ | int | 2 | 0 | ☑ | |
| First N | ☐ | | | | ☐ | |
| Name | ☐ | String | 8 | 0 | ☑ | |
| ID_Insurance | ☐ | String | 7 | 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 |
|---|---|
| **Column** | Column name as defined on the **Map Editor** schemas and on the Input or Output component schemas. |
| **Key** | 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. |
| **Type** | Type of data: String, Integer, Date, etc.<br>This column should always be defined in a Java version. |
| **Length** | -1 shows that no length value has been defined in the schema. |
| **Precision** | Defines the number of digits to the right of the decimal point. |
| **Nullable** | Clear this check box if the field value should not be null. |
| **Default** | Shows any default value that may be defined for this field. |
| **Comment** | Free text field. Enter any useful comment. |

Input metadata and output metadata are independent from each other. You can, for instance, change the label of a column on the output side without the column label of the input schema being changed.

However, any change made to the metadata are immediately reflected in the corresponding schema on the **tMap** relevant (Input or Output) area, but also on the schema defined for the component itself on the design workspace.

A Red colored background shows that an invalid character has been entered. Most special characters are prohibited in order for the Job to be able to interpret and use the text entered in the code. Authorized characters include lower-case, upper-case, figures except as start character.

## 5.2.6. 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:

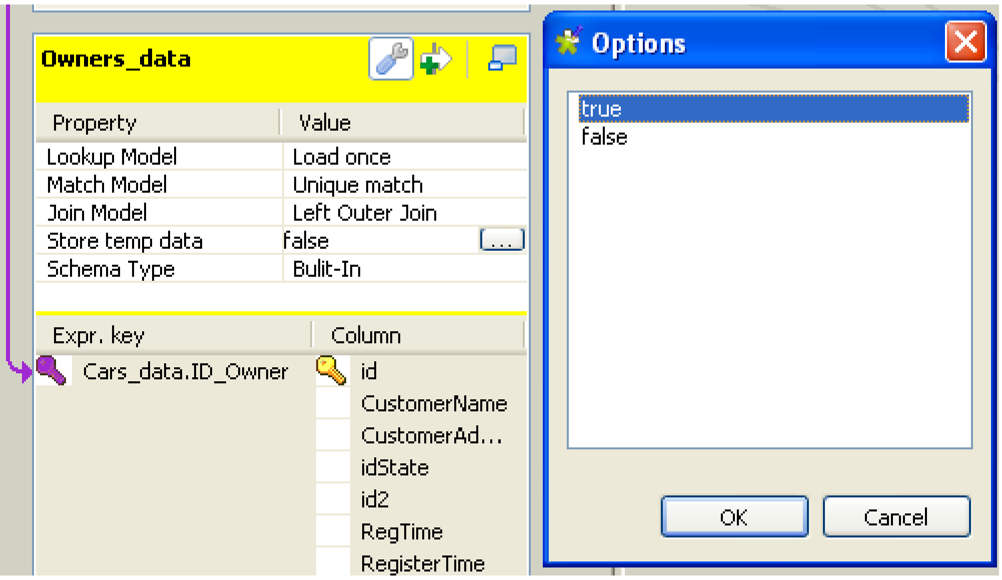
1. Double-click the **tMap** component in your Job to launch the **Map Editor**.

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.



| Property | Value |
|---|---|
| Lookup Model | Load once |
| Match Model | Unique match |
| Join Model | Left Outer Join |
| Store temp data | false ... |
| Schema Type | Bulit-In |

| Expr. key | Column |
|---|---|
| Cars_data.ID_Owner | id |
| | CustomerName |
| | CustomerAd... |
| | idState |
| | id2 |
| | RegTime |
| | RegisterTime |

true
false

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.

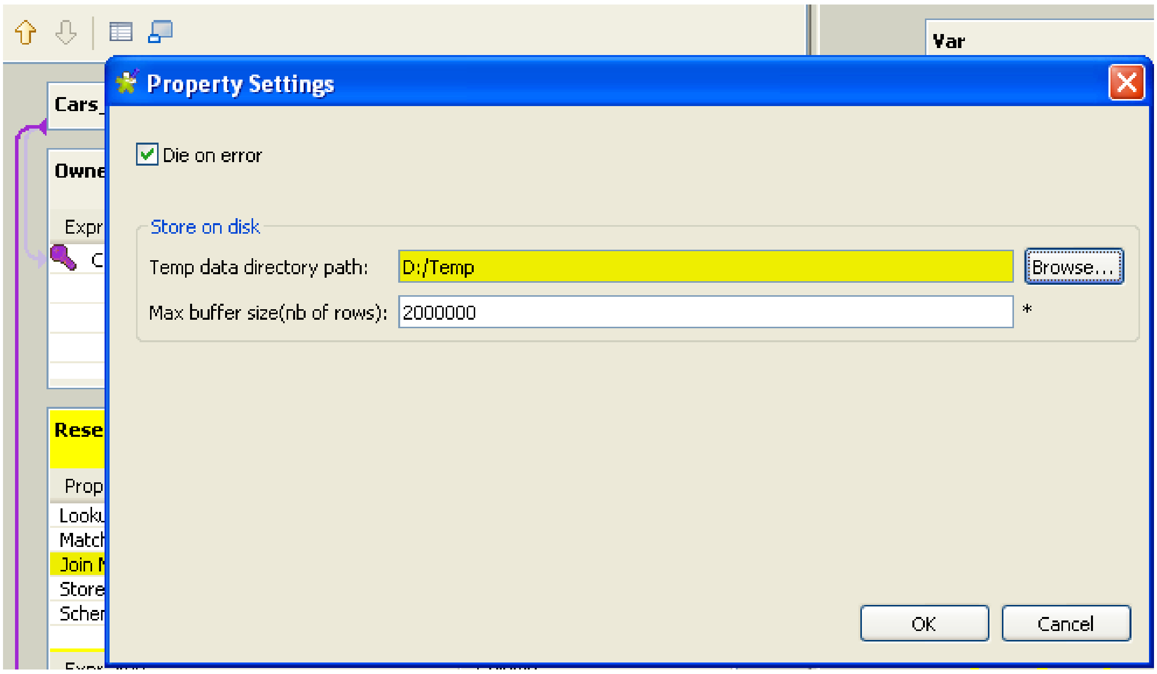
To set the temporary storage directory and the buffer size in the **Map Editor**:

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.



Property Settings

☑ Die on error

Store on disk

Temp data directory path: D:/Temp

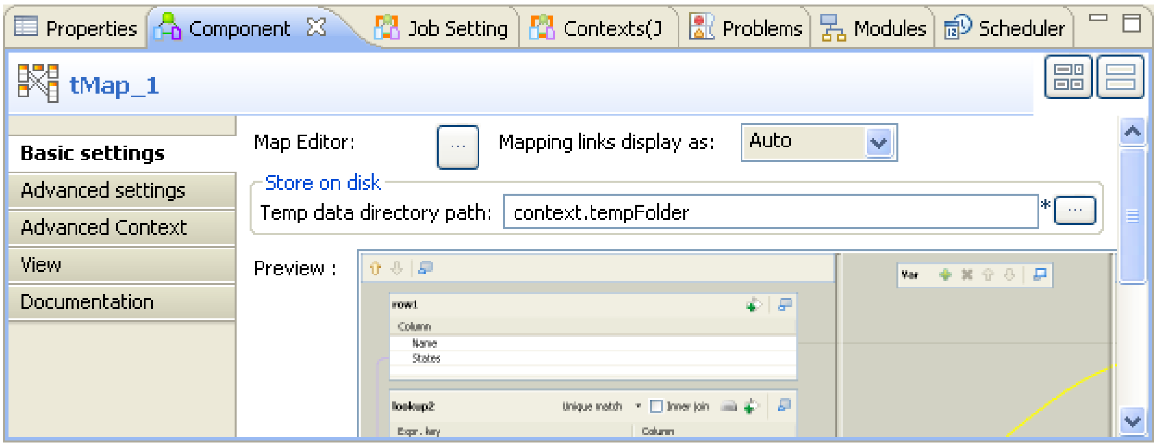
Max buffer size(nb of rows): 2000000 *

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 section *How to centralize contexts and variables*.



tMap_1

Basic settings

Map Editor: ...

Mapping links display as: Auto

Store on disk

Temp data directory path: context.tempFolder *

Preview :

At the end of the subjob, the temporary files are cleared.

This way, you will limit the use of allocated memory per reference data to be written onto temporary files stored on the disk.

As writing the main flow onto the disk requires the data to be sorted, note that the order of the output rows cannot be guaranteed.

On the **Advanced settings** view, you can also set a buffer size if needed. Simply fill out the field **Max buffer size (nb of rows)** in order for the data stored on the disk to be split into as many files as needed.

## 5.2.7. Handling Lookups

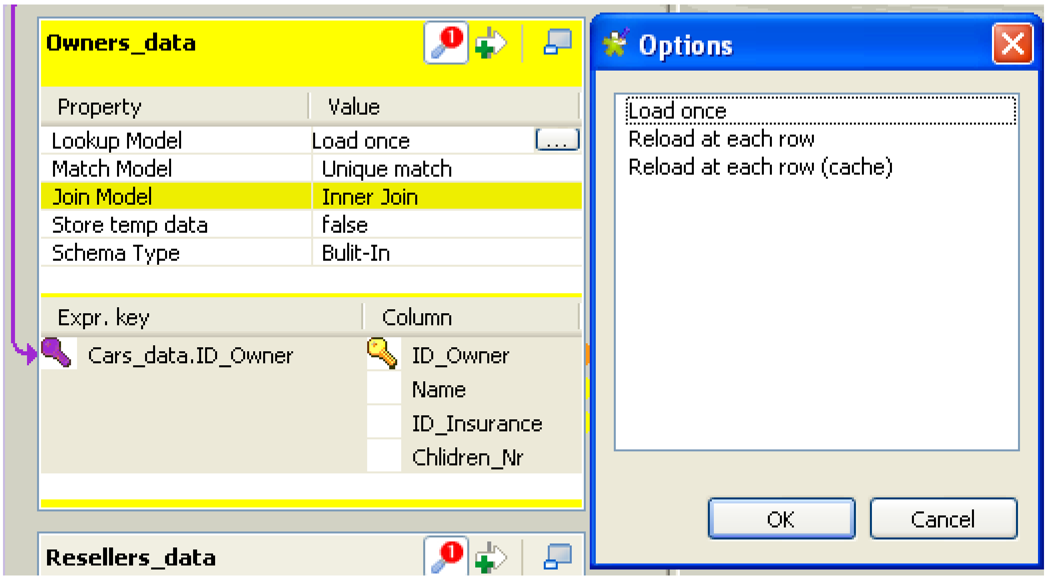
In order to adapt to the multiple processing types as well as to address performance issues, the **tMap** component supports different lookup loading modes.

- **Load once**: Default setting. Select this option to load the entire lookup flow before processing the main flow. This is the preferred option if you have a great number of data from your main flow that needs to be requested in your lookup, or if your reference (or lookup) data comes from a file that can be easily loaded.
- **Reload at each row**: At each row, the lookup gets loaded again. This is mainly interesting in Jobs where the lookup volume is large, while the main flow is pretty small. Note that this option allows you to use dynamic variable settings such as where clause, to change/update the lookup flow on the fly as it gets loaded, before the main flow join is processed. This option could be considered as the counter-part of the **Store temp data** option that is available for file lookups.
- **Reload at each row (cache)**: Expressions (in the Lookup table) are assessed and looked up in the cache first. The results of joins that have already been solved, are stored in the cache, in order to avoid loading the same results twice. This option optimizes the processing time and helps improve processing performance of the **tMap** component.

Note that for the time being, you cannot use **Reload at each row (cache)** and **Store temp data** at the same time.

To set the loading mode of a lookup flow:

1. Click the **tMap settings** button at the top 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.



Owners_data

| Property | Value |
|---|---|
| Lookup Model | Load once |
| Match Model | Unique match |
| Join Model | Inner Join |
| Store temp data | false |
| Schema Type | Bulit-In |

| Expr. key | Column |
|---|---|
| Cars_data.ID_Owner | ID_Owner |
| | Name |
| | ID_Insurance |
| | Chlidren_Nr |

Resellers_data

Options

Load once
Reload at each row
Reload at each row (cache)

For use cases using these options, see the **tMap** section of *Talend Open Studio for Big Data Components Reference Guide*.

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. For a use case using this option, see **tMap** in *Talend Open Studio for Big Data Components Reference Guide*.

# 5.3. tXMLMap operation

Before starting this section, we recommend reading the previous **tMap** sections for the basic knowledge of a **Talend** mapping component.

**tXMLMap** is fine-tuned to leverage the **Document** data type for processing XML data, a case of transformation that often mixes hierarchical data (XML) and flat data together. This **Document** type carries a complete user-specific XML flow. In using **tXMLMap**, you are able to add as many input or output flows as required into a visual map editor to perform, on these flows, the operations as follows:

- data multiplexing and demultiplexing,
- data transformation on any type of fields, particularly on the **Document** type,
- data matching via different models, for example, the **Unique match** mode (related topic: section *How to use Explicit Join*),
- Automated XML tree construction on both of the input and the output sides,
- inner join and left outer join (related topic: section *How to use Inner Join*)
- lookup between data sources whatever they are flat or XML data using models like **Load once** (related topic: section *Handling Lookups*),
- fields concatenation and interchange,
- field filtering using constraints,
- data rejecting.

Like **tMap**, a map editor is required to configure these operations. To open this map editor, you can double-click the **tXMLMap** icon in the design workspace, or alternatively, click the three-dot button next to the **Map Editor** in the **Basic settings** view of the **tXMLMap** component.

**tXMLMap** and **tMap** use the common approaches to accomplish most of these operations. Therefore, the following sections explain only the particular operations to which **tXMLMap** is dedicated for processing the hierarchical XML data.

The operations focusing on hierarchical data are:

- using the **Document** type to create the XML tree;
- managing the output XML data;
- editing the XML tree schema.

The following sections present more relevant details.

Different from **tMap**, **tXMLMap** does not provide the **Store temp data** option for storing temporary data onto the directory of your disk. For further information about this option of **tMap**, see section *Solving memory limitation issues in tMap use*.

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

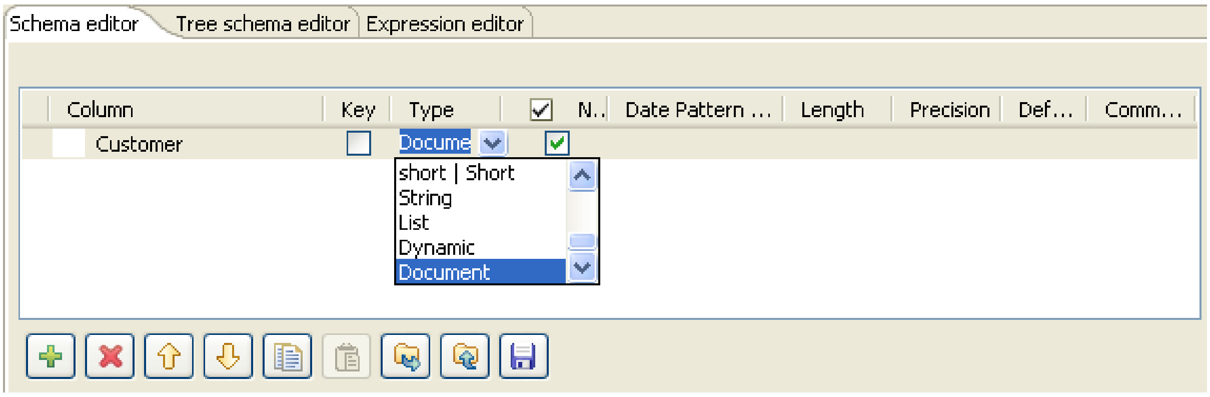
The **Document** data type fits perfectly the conception of defining XML structure as easily as possible. When you need the XML tree structure to map the input or output flow or both, use this type. Then you can import the

XML tree structure from various XML sources and edit the tree directly in the mapping editor, thus saving the manual efforts.

#### 5.3.1.1. How to set up the Document type

The **Document** data type is one of the data types provided by **Talend**. This **Document** type is set up when you edit the schema for the corresponding data in the **Schema editor**. For further information about the schema editor, see section *Using the Schema Editor*.

The following figure presents an example in which the input flow, *Customer*, is set up as the **Document** type. To replicate it, in the Map editor, you can simply click the **[+]** button to add one row on the input side of the **Schema editor**, rename it and select **Document** from the drop-down list of the given data types.



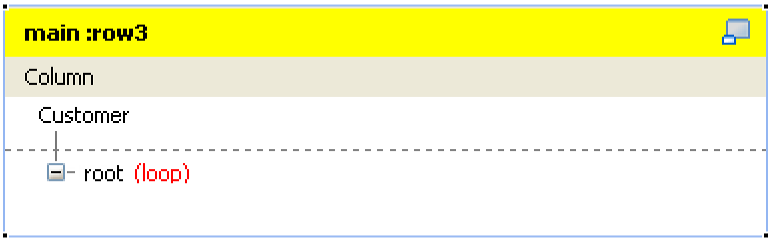
In practice for most cases, **tXMLMap** retrieves the schema of its preceding or succeeding components, for example, from a **tFileInputXML** component or in the ESB use case, from a **tESBProviderRequest** component. This avoids many manual efforts to set up the **Document** type for the XML flow to be processed. However, to continue to modify the XML structure as the content of a Document row, you need still to use the given Map editor.

Be aware that a **Document** flow carries a user-defined XML tree and is no more than one single field of a schema, which, same as the other schemas, may contain different data types between each field. For further information about how to set a schema, see section *Basic Settings tab*.

Once the **Document** type is set up for a row of data, in the corresponding data flow table in the map editor, a basic XML tree structure is created automatically to reflect the details of this structure. This basic structure represents the minimum element required by a valid XML tree in using **tXMLMap**:

- The root element: it is the minimum element required by an XML tree to be processed and when needs be, the foundation to develop a sophisticated XML tree.

- The loop element: it determines the element over which the iteration takes place to read the hierarchical data of an XML tree. By default, the root element is set as loop element.



This figure gives an example with the input flow, *Customer*. Based on this generated XML root tagged as **root** by default, you can develop the XML tree structure of interest.

To do this, you need to:

1. Import the custom XML tree structure from one of the following types of source:

   - XML or XSD files (related topic: section *How to import the XML tree structure from XML and XSD files*)

     When you import an XSD file, you will create the XML structure this XSD file describes.

   If needs be, you can develop the XML tree of interest manually using the options provided on the contextual menu.

2. 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 situation:

   - If you have to create several XML trees, you need to define the loop element for each of them.

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.

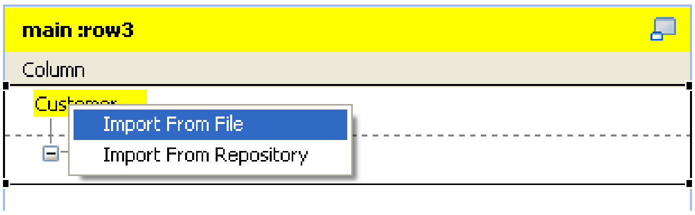
| Options | Operations |
|---|---|
| **Create Sub-element** and **Create Attribute** | Add elements or attributes to develop an XML tree. Related topic: section *How to add a sub-element or an attribute to an XML tree structure* |
| **Set a namespace** | Add and manage given namespaces on the imported XML tree. Related topic: section *How to manage a namespace* |
| **Delete** | Delete an element or an attribute. Related topic: section *How to delete an element or an attribute from the XML tree structure* |
| **Rename** | Rename an element or an attribute. |
| **As loop element** | Set or reset an element as loop element. Multiple loop elements and optional loop element are supported. |
| **As optional loop** | This option is not available unless to the loop element you have defined.<br>When the corresponding element exists in the source file, an optional loop element works the same way as a normal loop element; otherwise, it resets automatically its parent element as loop element or in absence of parent element in the source file, it takes the element of the higher level until the root element. But in the real-world practice, with such differences between the XML tree and the source file structure, we recommend adapting the XML tree to the source file for better performance. |
| **As group element** | On the XML tree of the output side, set an element as group element. Related topic: section *How to group the output data* |
| **As aggregate element** | On the XML tree of the output side, set an element as aggregate element. Related topic: section *How to aggregate the output data* |
| **Add Choice** | 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 **tXMLMap** to perform the function of the XSD Choice element to read or write a Document flow.<br>When **tXMLMap** processes a choice element, the elements contained in its declaration will not be outputted unless their mapping expressions are appropriately defined.<br>The **tXMLMap** component declares automatically any Choice element set in the XSD file it imports. |
| **Set as Substitution** | Set the Substitution element to specify the element substitutable for a given head element defined in the corresponding XSD. The Substitution element enables **tXMLMap** to perform the function of the XSD Substitution element to read or write a Document flow<br>When **tXMLMap** processes a substitution element, the elements contained in its declaration will not be outputted unless their mapping expressions are appropriately defined.<br>The **tXMLMap** component declares automatically any Substitution element set in the XSD file it imports. |

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

### 5.3.1.2. How to import the XML tree structure from XML and XSD files

To import the XML tree structure from an XML file, proceed as follows:

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

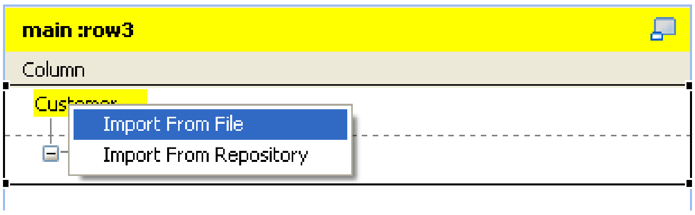


2. From this menu, select **Import From File**.

3. In the pop-up dialog box, browse to the XML file you need to use to provide the XML tree structure of interest and double-click the file.

To import the XML tree structure from an XSD file, proceed as follows:

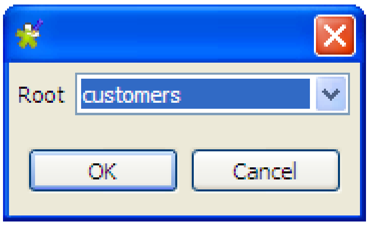
1. In the input flow table of interest, right-click the column name to open the contextual menu. In this example, it is *Customer*.



2. From this menu, select **Import From File**.

3. In the pop-up dialog box, browse to the XSD file you need to use to provide the XML tree structure of interest and double-click the file.

4. In the dialog box that appears, select an element from the **Root** list as the root of your XML tree, and click **OK**. Then the XML tree described by the XSD file imported is established.


Root customers

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 section *Editing the XML tree schema*.

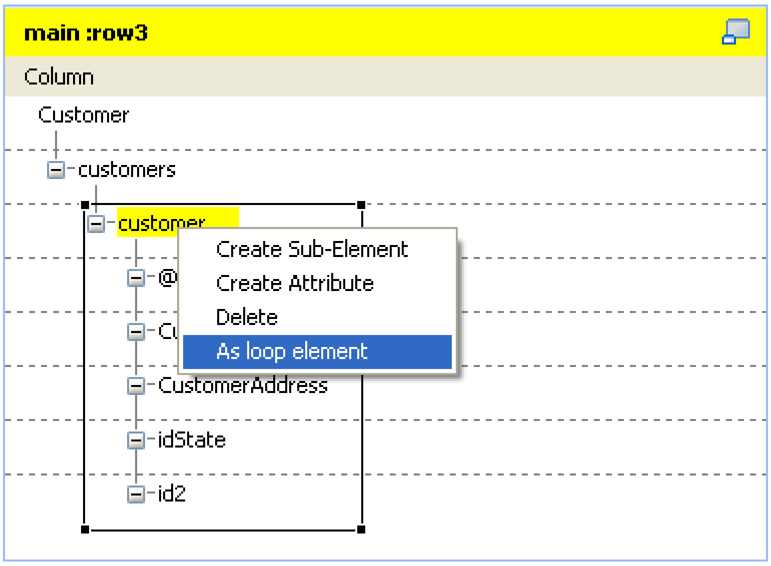
Then, you need to define the loop element in this XML tree structure. For further information about how to define a loop element, see section *How to set or reset a loop element for an imported XML structure*.

## 5.3.1.3. How to set or reset a loop element for an imported XML structure

You need to set at least one loop element for each XML tree if it does not have any. If it does, you may have to reset the existing loop element when needs be.

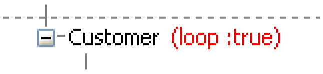
Whatever you need to set or reset a loop element, proceed as follows:

1. In the created XML tree structure, right-click the element you need to define as loop. For example, you need to define the *Customer* element as loop in the following figure.



2. From the pop-up contextual menu, select **As loop element** to define the selected element as loop.

   Once done, this selected element is marked with the text: **loop:true**.



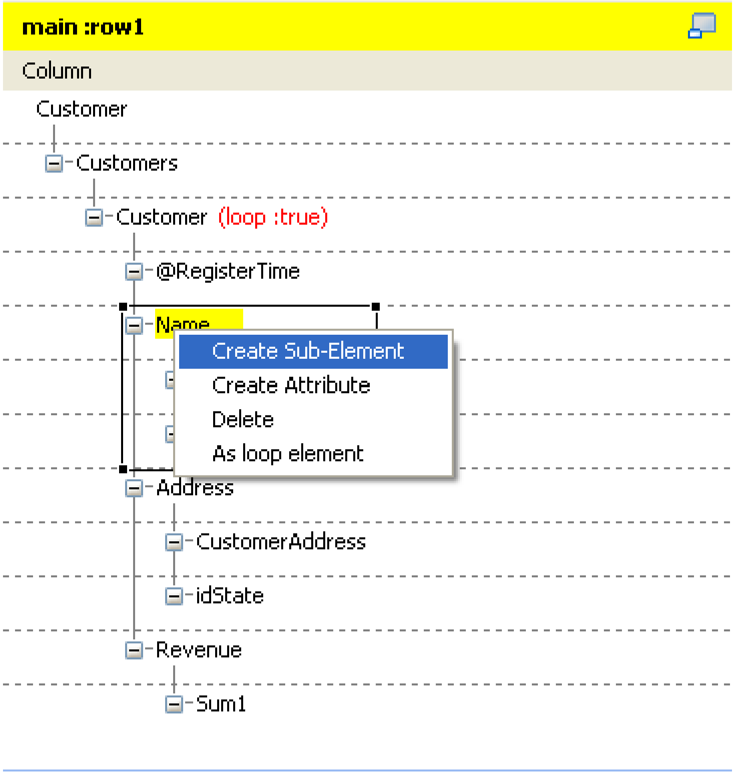
If you close the **Map Editor** without having set the required loop element for a given XML tree, its root element will be set automatically as loop element.

## 5.3.1.4. How to add a sub-element or an attribute to an XML tree structure

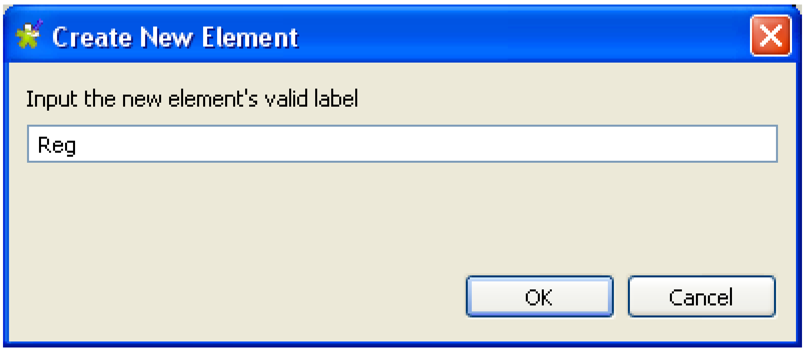
In the XML tree structure view, you are able to manually add a sub-element or an attribute to the root or to any of the existing elements when needs be.

To do either of these operations, proceed as follows:

1. In the XML tree you need to edit, right-click the element to which you need to add a sub-element or an attribute underneath and select **Create Sub-Element** or **Create Attribute** according to your purpose.



2. In the pop-up **[Create New Element]** wizard, type in the name you need to use for the added sub-element or attribute.



Input the new element's valid label

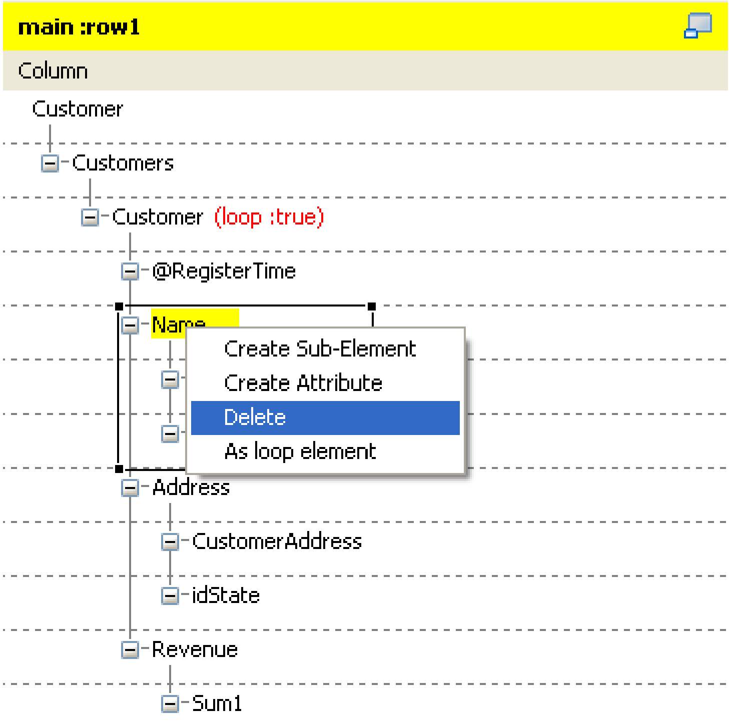
Reg

3. Click **OK** to validate this creation. The new sub-element or attribute displays in the XML tree structure you are editing.

### 5.3.1.5. How to delete an element or an attribute from the XML tree structure

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

1. In the XML tree you need to edit, right-click the element or the attribute you need to delete.



2. In the pop-up contextual menu, select **Delete**.

   Then the selected element or attribute is deleted, including all of the sub-elements or the attributes attached to it underneath.

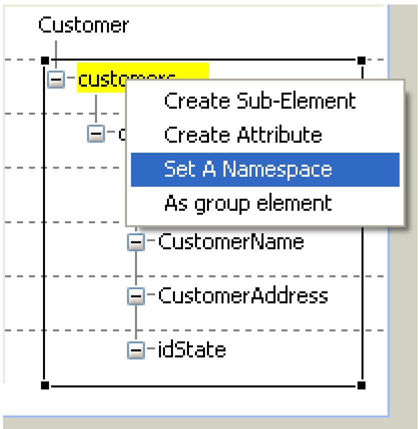
## 5.3.1.6. How to manage a namespace

When necessary, you are able to set and edit namespace for each of the element in the a created XML tree of the input or the output data flow.

### Defining a namespace

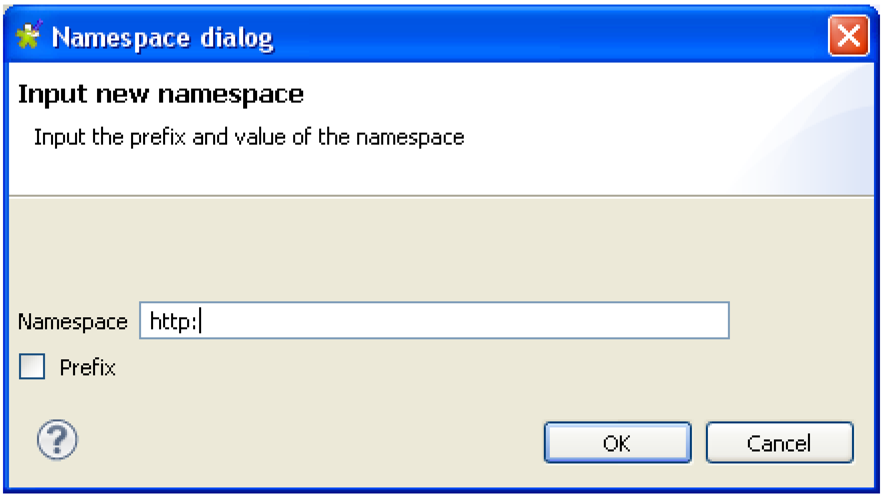
To do this, proceed as follows:

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



2. In the pop-up contextual menu, select **Set a namespace**. Then the **[Namespace dialog]** wizard displays.

3. In this wizard, type in the URI you need to use.



Namespace dialog

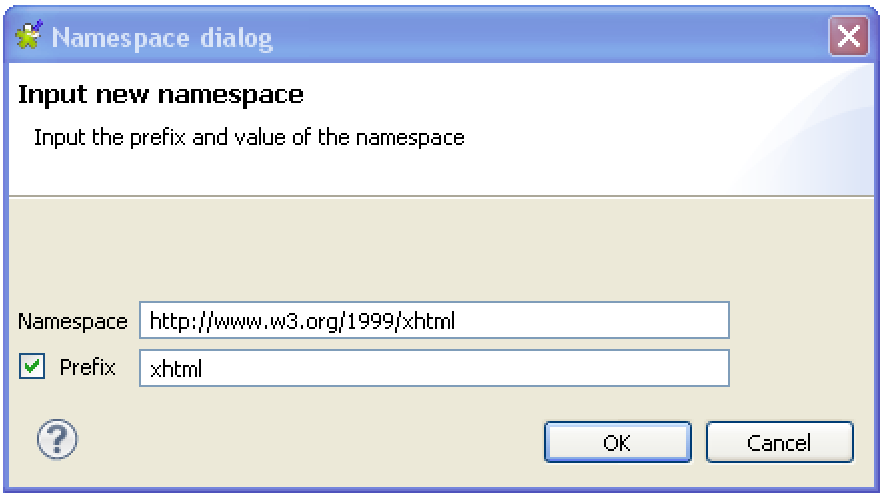
Input new namespace

Input the prefix and value of the namespace

Namespace: http:

□ Prefix

4. 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*.



Namespace dialog

Input new namespace

Input the prefix and value of the namespace

Namespace: http://www.w3.org/1999/xhtml

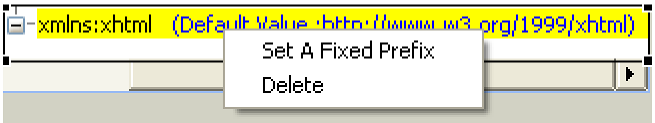
☑ Prefix: xhtml

5. Click **OK** to validate this declaration.

### Modifying the default value of a namespace

To do this, proceed as follows:

1. In the XML tree that the namespace you need to edit belongs to, right-click this namespace to open the contextual menu.



2. In this menu, select **Set A Fixed Prefix** to open the corresponding wizard.

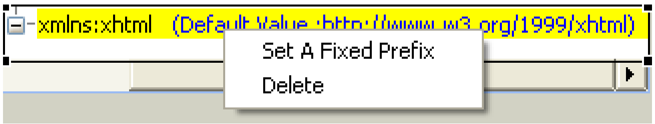
3. Type in the new default value you need in this wizard.

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

### Deleting a namespace

To do this, proceed as follows:

1. In the XML tree that the namespace you need to edit belongs to, right-click this namespace to open the contextual menu.



2. In this menu, click **Delete** to validate this deletion

## 5.3.1.7. How to group the output data

The **tXMLMap** component uses a group element to group the output data according to a given grouping condition. This allows you to wrap elements matching the same condition with this group element.

To set a group element, two restrictions must be respected:

1. the root node cannot be set as group element;

2. the group element must be the parent of the loop element.

The option of setting group element is not visible until you have set the loop element; this option is also invisible if an element is not allowed to be set as group element.

Once the group element is set, all of its sub-elements except the loop one are used as conditions to group the output data.

You have to carefully design the XML tree view for the optimized usage of a given group element. For further information about how to use a group element, see **tXMLMap** in *Talend Open Studio for Big Data Components Reference Guide*.

**tXMLMap** provides group element and aggregate element to classify data in the XML tree structure. When handling a row of XML data flow, the behavioral difference between them is:

- The group element processes the data always within one single flow.
- The aggregate element splits this flow into separate and complete XML flows.

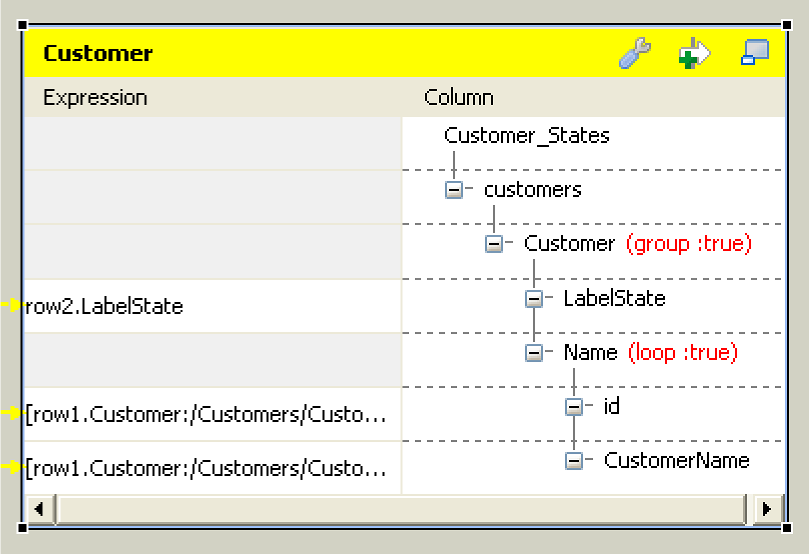
### Setting a group element

To set a group element, proceed as follows:

1. In the XML tree view on the output side of the **Map editor**, right-click the element you need to set as group element.

2. From the opened contextual menu, select **As group element**.

   Then this element of selection becomes the group element. The following figure presents an example of an XML tree with the group element.



Customer

| Expression | Column |
|---|---|
| | Customer_States |
| | customers |
| | Customer (group :true) |
| row2.LabelState | LabelState |
| | Name (loop :true) |
| [row1.Customer:/Customers/Custo... | id |
| [row1.Customer:/Customers/Custo... | CustomerName |

### Revoking a defined group element

To revoke a defined group element, proceed as follows:

1. In the XML tree view on the output side of the **Map editor**, right-click the element you have defined as **group element**.

2. From the opened contextual menu, select **Remove group element**.

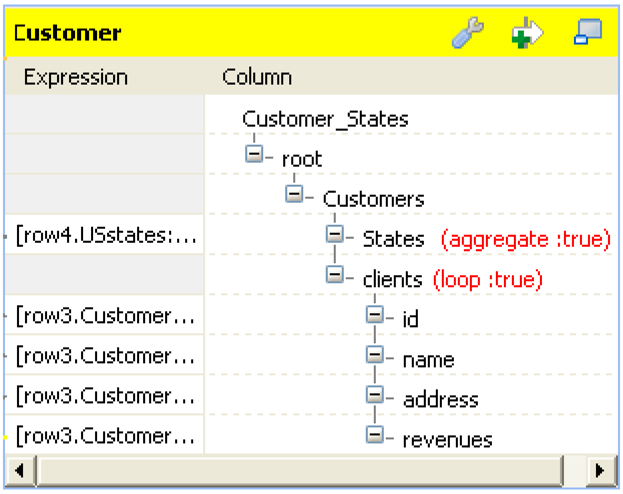
   Then the defined group element is revoked.

## 5.3.1.8. How to aggregate the output data

With **tXMLMap**, you can define as many aggregate elements as required in the output XML tree to class the XML data accordingly. Then this component outputs these classes, each as one complete XML flow.

1. To define an element as aggregate element, simply right-click this element of interest in the XML tree view on the output side of the **Map editor** and from the contextual menu, select **As aggregate element**.

   Then this element becomes the aggregate element. Texts in red are added to it, reading **aggregate : true**. The following figure presents an example.



| Expression | Column |
|---|---|
| | Customer_States |
| | root |
| | Customers |
| [row4.USstates:... | States (aggregate :true) |
| | clients (loop :true) |
| [row3.Customer... | id |
| [row3.Customer... | name |
| [row3.Customer... | address |
| [row3.Customer... | revenues |

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

To define an element as aggregate element, ensure that this element has no child element and the **All in one** feature is being disabled. The **As aggregate element** option is not available in the contextual menu until both of the conditions are respected. For further information about the **All in one** feature, see section *How to output elements into one document*.

For an example about how to use the aggregate element with **tXMLMap**, see *Talend Open Studio for Big Data Components Reference Guide*.

**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.

## 5.3.2. Defining the output mode

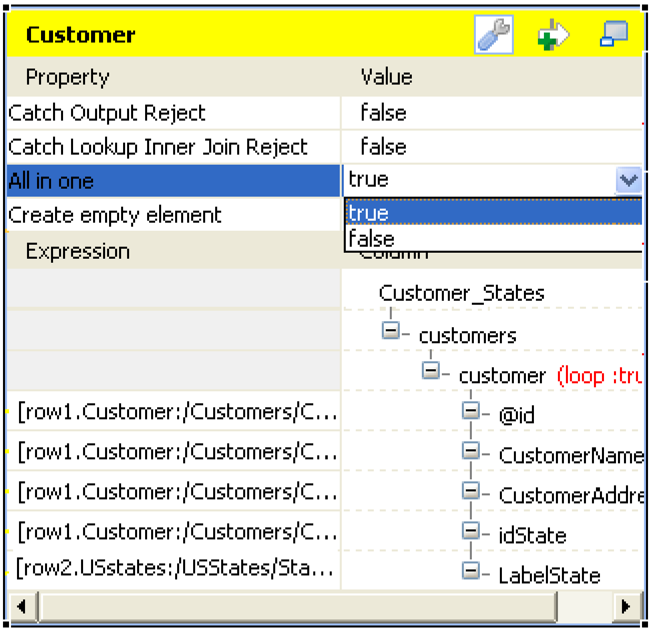
To define the output mode of the document-type data, you are defining whether to put all of the XML elements into one single XML flow and when empty element exist, whether to output them. By doing this, you do not change the structure of the XML tree you have created.

### 5.3.2.1. How to output elements into one document

Unless you are using the aggregate element which always classifies the output elements and splits an output XML flow, you are able to determine whether an XML flow is output as one single flow or as separate flows, using the **All in one** feature in the **tXMLMap** editor.

To do this, on the output side of the **Map editor**, proceed as follows:

1. Click the pincer icon to open the map setting panel. The following figure presents an example.



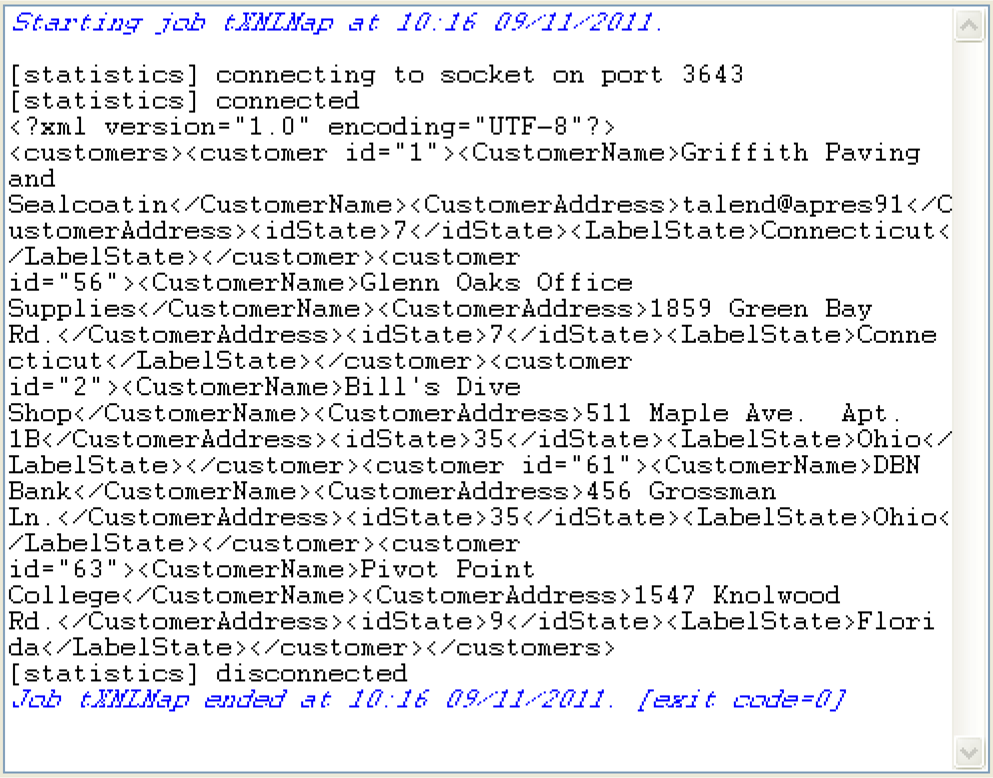
**Customer**

| Property | Value |
|---|---|
| Catch Output Reject | false |
| Catch Lookup Inner Join Reject | false |
| All in one | true<br>true<br>false |
| Create empty element | |

| Expression | Column |
|---|---|
| | Customer_States |
| | customers |
| | customer (loop :tru |
| [row1.Customer:/Customers/C... | @id |
| [row1.Customer:/Customers/C... | CustomerName |
| [row1.Customer:/Customers/C... | CustomerAddre |
| [row1.Customer:/Customers/C... | idState |
| [row2.USstates:/USStates/Sta... | LabelState |

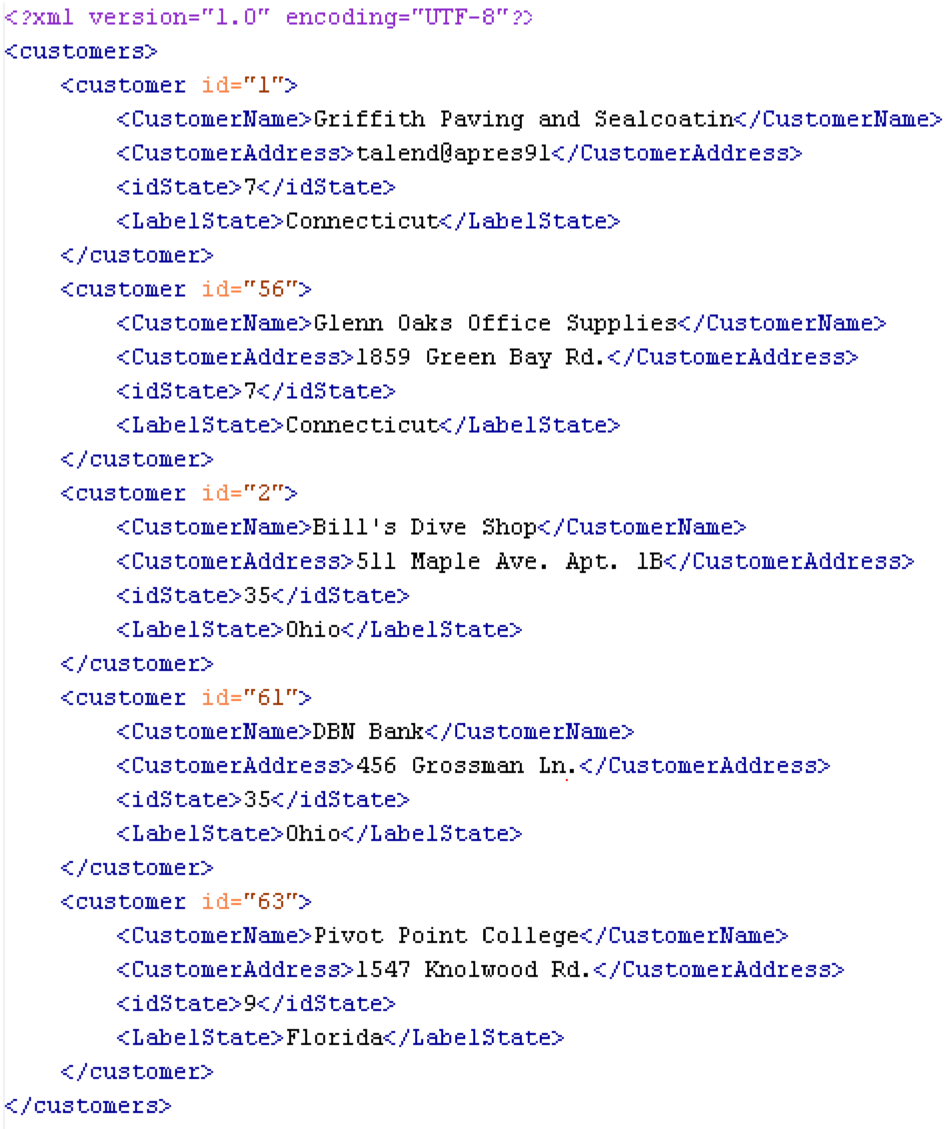
2. Click the **All in one** field and from the drop-down list, select **true** or **false** to decide whether the output XML flow should be one single flow.

- If you select **true**, the XML data is output all in one single flow. In this example, the single flow reads as follows:



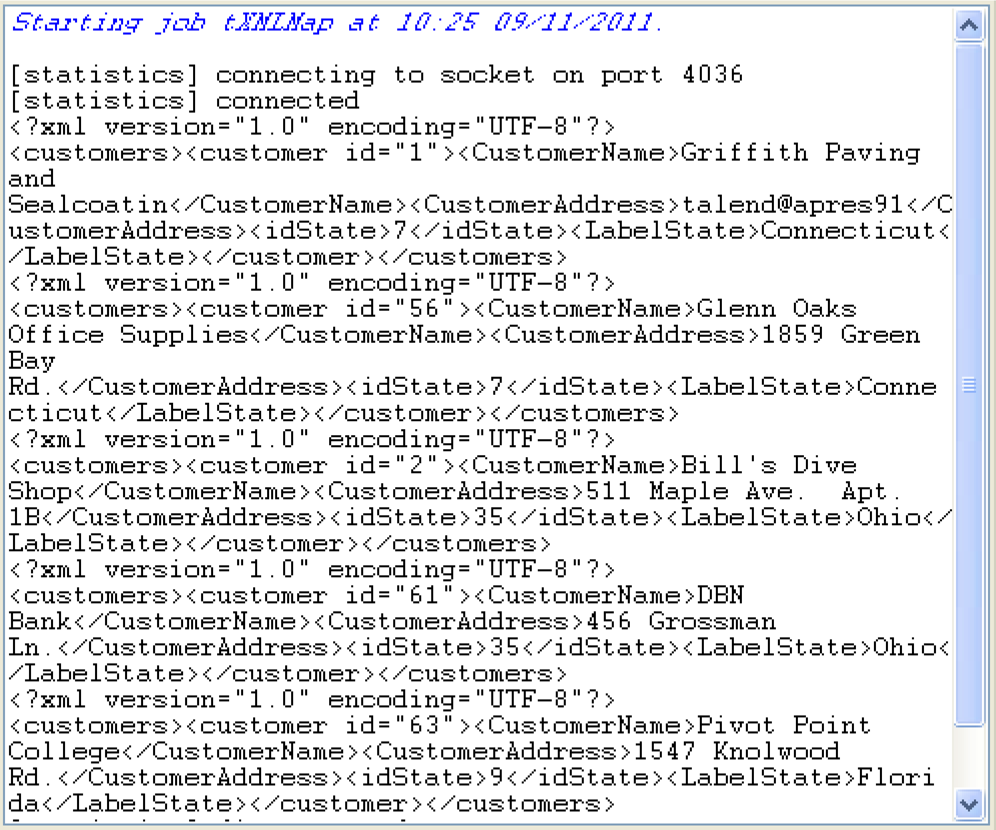
```
Starting job tXMLMap at 10:16 09/11/2011.

[statistics] connecting to socket on port 3643
[statistics] connected
<?xml version="1.0" encoding="UTF-8"?>
<customers><customer id="1"><CustomerName>Griffith Paving
and
Sealcoatin</CustomerName><CustomerAddress>talend@apres91</C
ustomerAddress><idState>7</idState><LabelState>Connecticut<
/LabelState></customer><customer
id="56"><CustomerName>Glenn Oaks Office
Supplies</CustomerName><CustomerAddress>1859 Green Bay
Rd.</CustomerAddress><idState>7</idState><LabelState>Conne
cticut</LabelState></customer><customer
id="2"><CustomerName>Bill's Dive
Shop</CustomerName><CustomerAddress>511 Maple Ave.  Apt.
1B</CustomerAddress><idState>35</idState><LabelState>Ohio</
LabelState></customer><customer id="61"><CustomerName>DBN
Bank</CustomerName><CustomerAddress>456 Grossman
Ln.</CustomerAddress><idState>35</idState><LabelState>Ohio<
/LabelState></customer><customer
id="63"><CustomerName>Pivot Point
College</CustomerName><CustomerAddress>1547 Knolwood
Rd.</CustomerAddress><idState>9</idState><LabelState>Flori
da</LabelState></customer></customers>
[statistics] disconnected
Job tXMLMap ended at 10:16 09/11/2011. [exit code=0]
```

The structure of this flow reads:



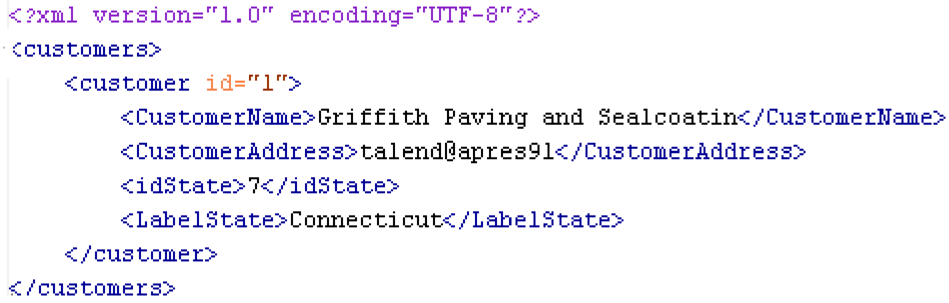
```
<?xml version="1.0" encoding="UTF-8"?>
<customers>
    <customer id="1">
        <CustomerName>Griffith Paving and Sealcoatin</CustomerName>
        <CustomerAddress>talend@apres91</CustomerAddress>
        <idState>7</idState>
        <LabelState>Connecticut</LabelState>
    </customer>
    <customer id="56">
        <CustomerName>Glenn Oaks Office Supplies</CustomerName>
        <CustomerAddress>1859 Green Bay Rd.</CustomerAddress>
        <idState>7</idState>
        <LabelState>Connecticut</LabelState>
    </customer>
    <customer id="2">
        <CustomerName>Bill's Dive Shop</CustomerName>
        <CustomerAddress>511 Maple Ave. Apt. 1B</CustomerAddress>
        <idState>35</idState>
        <LabelState>Ohio</LabelState>
    </customer>
    <customer id="61">
        <CustomerName>DBN Bank</CustomerName>
        <CustomerAddress>456 Grossman Ln.</CustomerAddress>
        <idState>35</idState>
        <LabelState>Ohio</LabelState>
    </customer>
    <customer id="63">
        <CustomerName>Pivot Point College</CustomerName>
        <CustomerAddress>1547 Knolwood Rd.</CustomerAddress>
        <idState>9</idState>
        <LabelState>Florida</LabelState>
    </customer>
</customers>
```

- If you select **false**, the XML data is output in separate flows, each loop being one flow, neither grouped nor aggregated. In this example, these flows read as follows:



```
Starting job tXMLMap at 10:25 09/11/2011.

[statistics] connecting to socket on port 4036
[statistics] connected
<?xml version="1.0" encoding="UTF-8"?>
<customers><customer id="1"><CustomerName>Griffith Paving
and
Sealcoatin</CustomerName><CustomerAddress>talend@apres91</C
ustomerAddress><idState>7</idState><LabelState>Connecticut<
/LabelState></customer></customers>
<?xml version="1.0" encoding="UTF-8"?>
<customers><customer id="56"><CustomerName>Glenn Oaks
Office Supplies</CustomerName><CustomerAddress>1859 Green
Bay
Rd.</CustomerAddress><idState>7</idState><LabelState>Conne
cticut</LabelState></customer></customers>
<?xml version="1.0" encoding="UTF-8"?>
<customers><customer id="2"><CustomerName>Bill's Dive
Shop</CustomerName><CustomerAddress>511 Maple Ave.  Apt.
1B</CustomerAddress><idState>35</idState><LabelState>Ohio</
LabelState></customer></customers>
<?xml version="1.0" encoding="UTF-8"?>
<customers><customer id="61"><CustomerName>DBN
Bank</CustomerName><CustomerAddress>456 Grossman
Ln.</CustomerAddress><idState>35</idState><LabelState>Ohio<
/LabelState></customer></customers>
<?xml version="1.0" encoding="UTF-8"?>
<customers><customer id="63"><CustomerName>Pivot Point
College</CustomerName><CustomerAddress>1547 Knolwood
Rd.</CustomerAddress><idState>9</idState><LabelState>Flori
da</LabelState></customer></customers>
```

Each flow contains one complete XML structure. To take the first flow as example, its structure reads:



```
<?xml version="1.0" encoding="UTF-8"?>
<customers>
    <customer id="1">
        <CustomerName>Griffith Paving and Sealcoatin</CustomerName>
        <CustomerAddress>talend@apres91</CustomerAddress>
        <idState>7</idState>
        <LabelState>Connecticut</LabelState>
    </customer>
</customers>
```

The **All in one** feature is disabled if you are using the aggregate element. For further information about the aggregate element, see section *How to aggregate the output data*

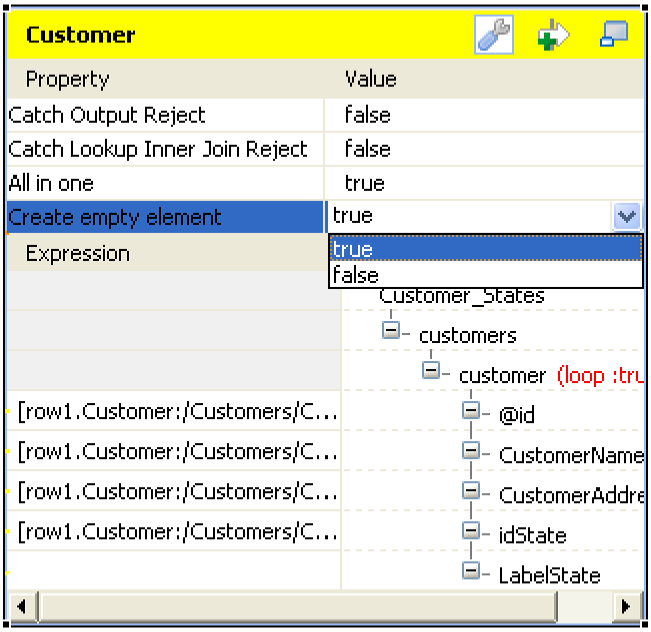
## 5.3.2.2. How to manage empty element in Map editor

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

By contrast, in some scenarios, you do not need to output the empty element while you have to keep them in the output XML tree for some reasons.

**tXMLMap** allows you to set the boolean for the creation of empty element. To do this, on the output side of the **Map editor**, perform the following operations:

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



| Property | Value |
|---|---|
| Catch Output Reject | false |
| Catch Lookup Inner Join Reject | false |
| All in one | true |
| Create empty element | true |

| Expression |
|---|
| [row1.Customer:/Customers/C... |
| [row1.Customer:/Customers/C... |
| [row1.Customer:/Customers/C... |
| [row1.Customer:/Customers/C... |

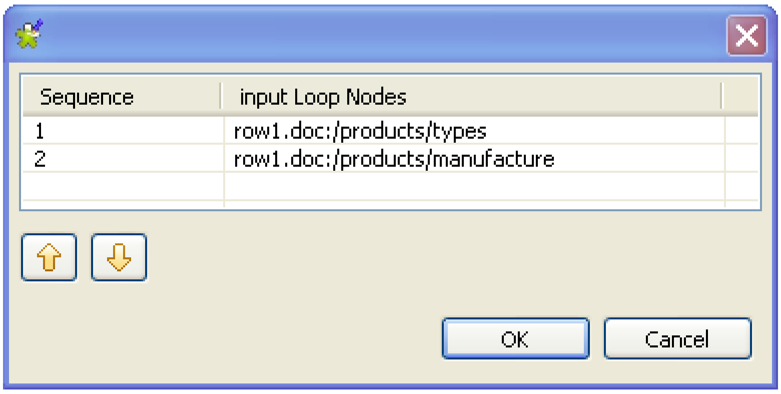
2. In the panel, click the **Create empty element** field and from the drop-down list, select **true** or **false** to decide whether to output the empty element.

   - If you select **true**, the empty element is created in the output XML flow and output, for example, `<customer><LabelState/></customer>`.

   - If you select **false**, the empty element is not output.

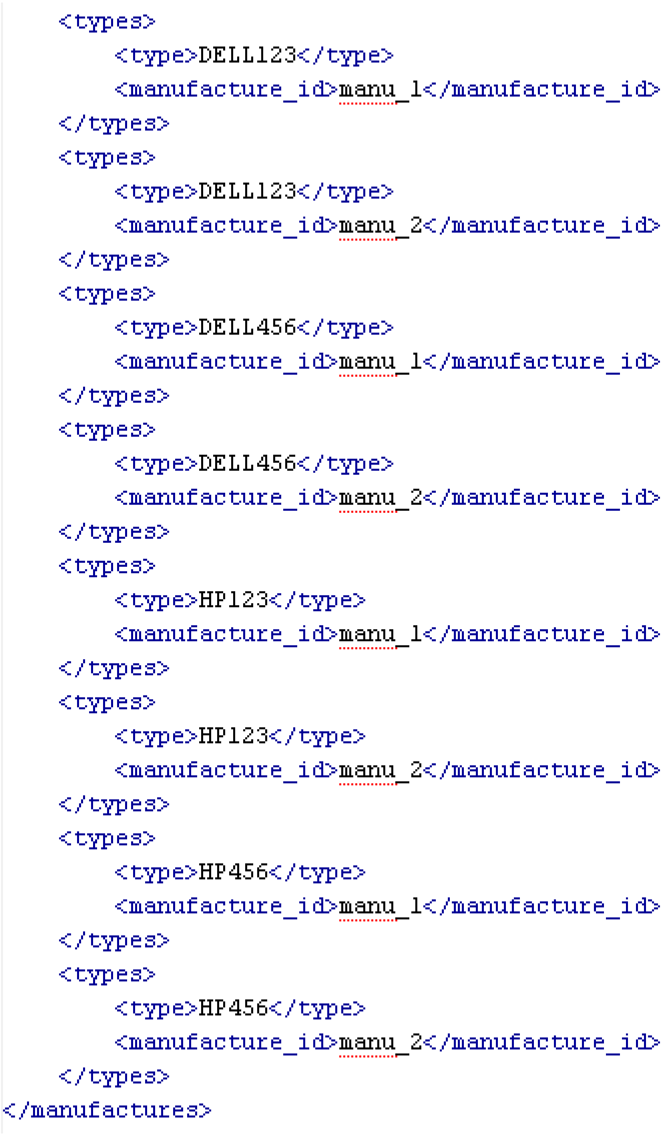
## 5.3.2.3. How to define the sequence of multiple input loops

If a loop element, or the flat data flow, receives mappings from more than one loop element of the input flow, you need to define the sequence of the input loops. The first loop element of this sequence will be the primary loop, so the transformation process related to this sequence will first loop over this element such that the data outputted will be sorted with regard to its element values.



| Sequence | input Loop Nodes |
|---|---|
| 1 | row1.doc:/products/types |
| 2 | row1.doc:/products/manufacture |

For example, in this figure, the *types* element is the primary loop and the outputted data will be sorted by the values of this element.



```
    <types>
        <type>DELL123</type>
        <manufacture_id>manu_1</manufacture_id>
    </types>
    <types>
        <type>DELL123</type>
        <manufacture_id>manu_2</manufacture_id>
    </types>
    <types>
        <type>DELL456</type>
        <manufacture_id>manu_1</manufacture_id>
    </types>
    <types>
        <type>DELL456</type>
        <manufacture_id>manu_2</manufacture_id>
    </types>
    <types>
        <type>HP123</type>
        <manufacture_id>manu_1</manufacture_id>
    </types>
    <types>
        <type>HP123</type>
        <manufacture_id>manu_2</manufacture_id>
    </types>
    <types>
        <type>HP456</type>
        <manufacture_id>manu_1</manufacture_id>
    </types>
    <types>
        <type>HP456</type>
        <manufacture_id>manu_2</manufacture_id>
    </types>
</manufactures>
```

In this case in which one output loop element receives several input loop elements, a **[...]** button appears next to this receiving loop element or for the flat data, appears on the head of the table representing the flat data flow. To define the loop sequence, do the following:

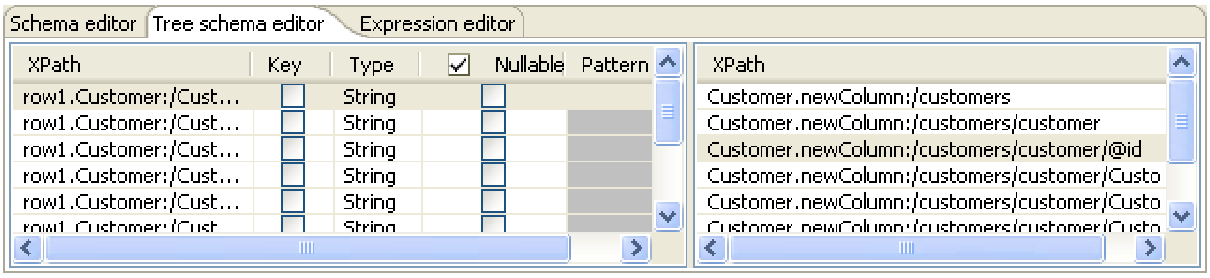
1. Click this **[...]** button to open the sequence arrangement window as presented by the figure used earlier in this section.

2. Use the up or down flash button to arrange this sequence.

## 5.3.3. Editing the XML tree schema

In addition to the **Schema editor** and the **Expression editor** views that **tMap** is also equipped with, a **Tree schema editor** view is provided in the map editor of **tXMLMap** for you to edit the XML tree schema of an input or output data flow.

To access this schema editor, click the **Tree schema editor** tab on the lower part of the map editor.



| XPath | Key | Type | ☑ Nullable | Pattern |
|---|---|---|---|---|
| row1.Customer:/Cust... | □ | String | □ | |
| row1.Customer:/Cust... | □ | String | □ | |
| row1.Customer:/Cust... | □ | String | □ | |
| row1.Customer:/Cust... | □ | String | □ | |
| row1.Customer:/Cust... | □ | String | □ | |
| row1.Customer:/Cust | □ | String | □ | |

| XPath |
|---|
| Customer.newColumn:/customers |
| Customer.newColumn:/customers/customer |
| Customer.newColumn:/customers/customer/@id |
| Customer.newColumn:/customers/customer/Custo |
| Customer.newColumn:/customers/customer/Custo |
| Customer.newColumn:/customers/customer/Custo |

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 |
|---|---|
| **XPath** | 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. |
| **Key** | 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. |
| **Type** | Type of data: String, Integer, Document, etc.<br>This column should always be defined in a Java version. |
| **Nullable** | Select this check box if the field value could be null. |
| **Pattern** | Define the pattern for the Date data type. |

Input metadata and output metadata are independent from each other. You can, for instance, change the label of a column on the output side without the column label of the input schema being changed.

However, any change made to the metadata are immediately reflected in the corresponding schema on the **tXMLMap** relevant (Input or Output) area, but also on the schema defined for the component itself on the design workspace.

For detailed use cases about the multiple operations that you can perform using **tXMLMap**, see *Talend Open Studio for Big Data Components Reference Guide*.

# Chapter 6. Managing routines

This chapter defines the routines, along with user scenarios, and explains how to create your own routines or how to customize the system routines. The chapter also gives an overview of the main routines and use cases of them. To have an overview of the most commonly used routines and other use cases, see appendix *System routines*.

Before starting any data integration processes, you need to be familiar with *Talend Open Studio for Big Data* Graphical User Interface (GUI). For more information, see appendix *GUI*.

# 6.1. What are routines

Routines are fairly complex Java functions, generally used to factorize code. They therefore optimize data processing and improve Job capacities.

You can also use the **Repository** tree view to store frequently used parts of code or extract parts of existing company functions, by calling them via the routines. This factorization makes it easier to resolve any problems which may arise and allows you to update the code used in multiple Jobs quickly and easily.

On top of this, certain system routines adopt the most common Java methods, using the **Talend** syntax. This allows you to escalate Java errors in the studio directly, thereby facilitating the identification and resolution of problems which may arise as your integration processes evolve with **Talend**.

There are two types of routines:

- System routines: a number of system routines are provided. They are classed according to the type of data which they process: numerical, string, date...
- User routines: these are routines which you have created or adapted from existing routines.

You do not need any knowledge of the Java language to create and use **Talend** routines.

All of the routines are stored under **Code > Routines** in the **Repository** tree view.

For further information concerning the system routines, see section *Accessing the System Routines*.

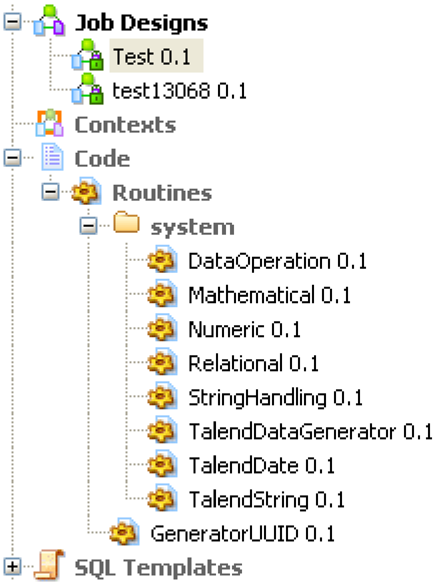
For further information about how to create user routines, see section *How to create user routines*.

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

# 6.2. Accessing the System Routines

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

The **system** folder and its content are read only.



Each class or category in the system folder contains several routines or functions. Double-click the class that you want to open.

All of the routines or functions within a class are composed of some descriptive text, followed by the corresponding Java code. In the Routines view, you can use the scrollbar to browse the different routines. Or alternatively:

1. Press **Ctrl+O** in the routines view.

   A dialog box displays a list of the different routines in the category.

2. Click the routine of interest.

   The view jumps to the section comprising the routine's descriptive text and corresponding code.

The syntax of routine call statements is case sensitive.

For more information about the most common Java routines, see appendix *System routines*.

# 6.3. Customizing the system routines

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

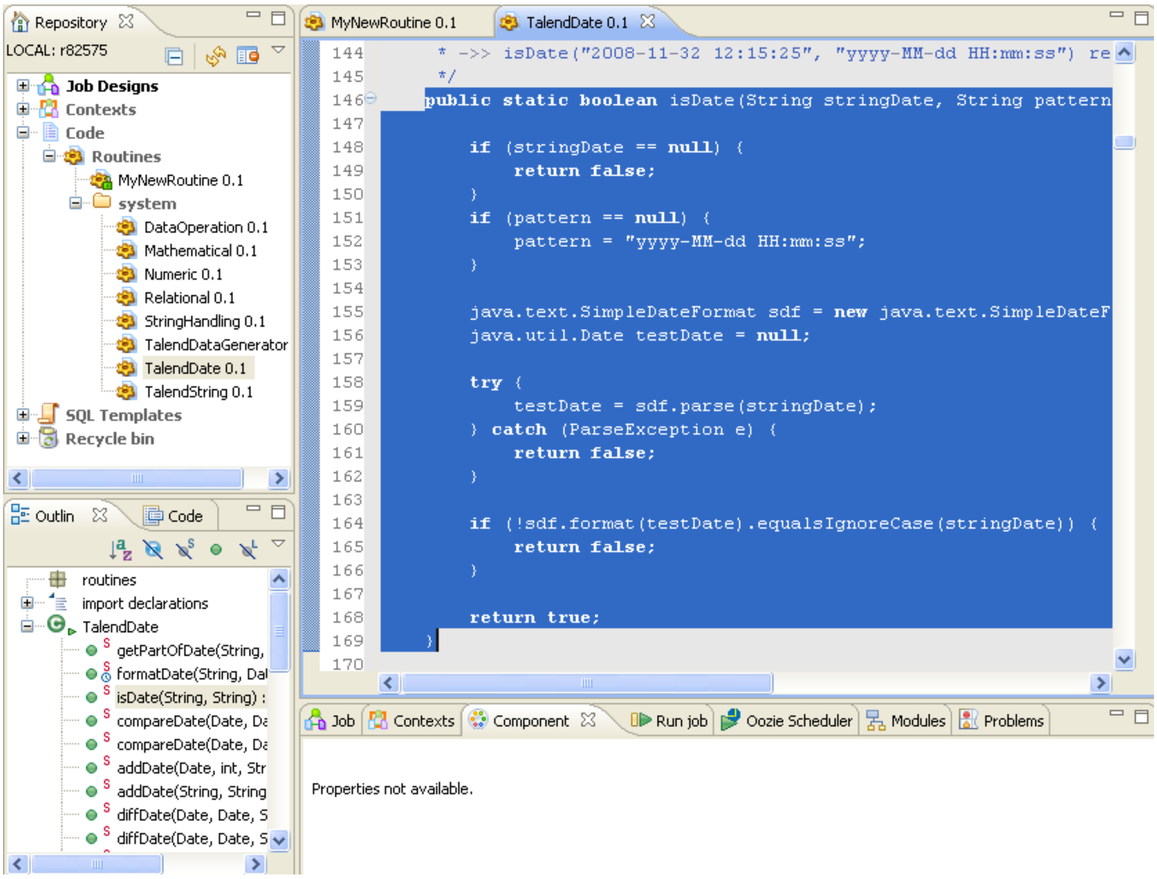
To customize a system routine:

1. First of all, create a user routine by following the steps outlined in the section *How to create user routines*. The routine opens in the workspace, where you shall find a basic example of a routine.

2. Then, under **Code** > **Routines** > **system**, select the class of routines which contains the routine(s) you want to customize.

3. Double-click the class which contains the relevant routine to open it in the workspace.

4. Use the **Outline** panel on the bottom left of the studio to locate the routine from which you want to copy all or part of the content.



```
     * ->> isDate("2008-11-32 12:15:25", "yyyy-MM-dd HH:mm:ss") re
     */
    public static boolean isDate(String stringDate, String pattern

        if (stringDate == null) {
            return false;
        }
        if (pattern == null) {
            pattern = "yyyy-MM-dd HH:mm:ss";
        }

        java.text.SimpleDateFormat sdf = new java.text.SimpleDateF
        java.util.Date testDate = null;

        try {
            testDate = sdf.parse(stringDate);
        } catch (ParseException e) {
            return false;
        }

        if (!sdf.format(testDate).equalsIgnoreCase(stringDate)) {
            return false;
        }

        return true;
    }
```

Properties not available.

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.

# 6.4. Managing user routines

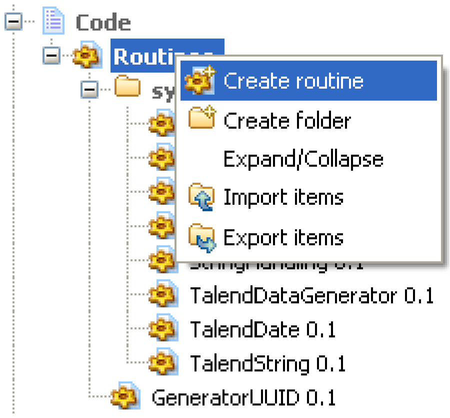
*Talend Open Studio for Big Data* allows you to create user routines, to modify them or to modify system routines, in order to fill your specific needs.

## 6.4.1. How to create user routines

You can create your own routines according to your particular factorization needs. Like the system routines, the user routines are stored in the **Repository** tree view under **Code** > **Routines**. You can add folders to help organize your routines and call them easily in any of your Jobs.

To create a new user routine, complete the following:

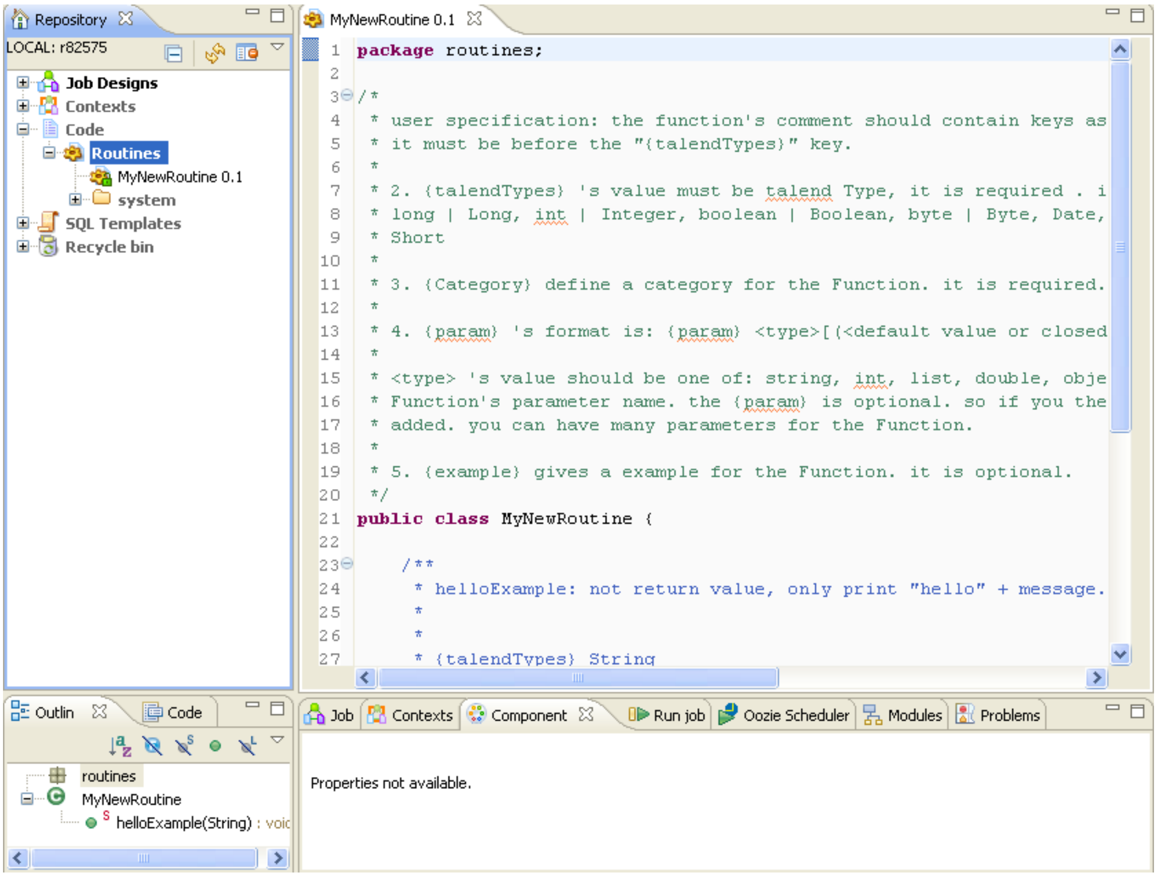
1. In the **Repository** tree view, expand **Code** to display the **Routines** folder.



2. Right-click **Routines** and select **Create routine**.

3. The **[New routine]** dialog box opens. Enter the information required to create the routine, ie., its name, description...

4. Click **Finish** to proceed to the next step.



```
package routines;

/*
 * user specification: the function's comment should contain keys as
 * it must be before the "{talendTypes}" key.
 *
 * 2. {talendTypes} 's value must be talend Type, it is required . i
 * long | Long, int | Integer, boolean | Boolean, byte | Byte, Date,
 * Short
 *
 * 3. {Category} define a category for the Function. it is required.
 *
 * 4. {param} 's format is: {param} <type>[(<default value or closed
 *
 * <type> 's value should be one of: string, int, list, double, obje
 * Function's parameter name. the {param} is optional. so if you the
 * added. you can have many parameters for the Function.
 *
 * 5. {example} gives a example for the Function. it is optional.
 */
public class MyNewRoutine {

    /**
     * helloExample: not return value, only print "hello" + message.
     *
     *
     * {talendTypes} String
```

Properties not available.

The newly created routine appears in the **Repository** tree view, directly below the **Routines** node. The routine editor opens to reveal a model routine which contains a simple example, by default, comprising descriptive text in blue, followed by the corresponding code.

We advise you to add a very detailed description of the routine. The description should generally include the input and output parameters you would expect to use in the routine, as well as the results returned along with an example. This information tends to be useful for collaborative work and the maintenance of the routines.

The following example of code is provided by default:

```
public static void helloExample(String message) {
        if (message == null) {
            message = "World"; //$NON-NLS-1$
        }
        System.out.println("Hello " + message + " !");
```

5. Modify or replace the model with your own code and press **Ctrl+S** to save the routine. Otherwise, the routine is saved automatically when you close it.

You can copy all or part of a system routine or class and use it in a user routine by using the **Ctrl+C** and **Ctrl+V** commands, then adapt the code according to your needs. For further information about how to customize routines, see section *Customizing the system routines*.

## 6.4.2. How to edit user routines

You can modify the user routines whenever you like.

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

To edit your user routines:

1. Right click the routine you want to edit and select **Edit Routine**.

2. The routine opens in the workspace, where you can modify it.

3. Once you have adapted the routine to suit your needs, press **Ctrl+S** to save it.

If you want to reuse a system routine for your own specific needs, see section *Customizing the system routines*.

## 6.4.3. How to edit user routine libraries

You can edit the library of any of the user routines by importing external .jar files for the selected routine. These external library files will be listed, like modules, in the **Modules** view in your current Studio. For more information on the **Modules** view, see section *How to install external modules*.

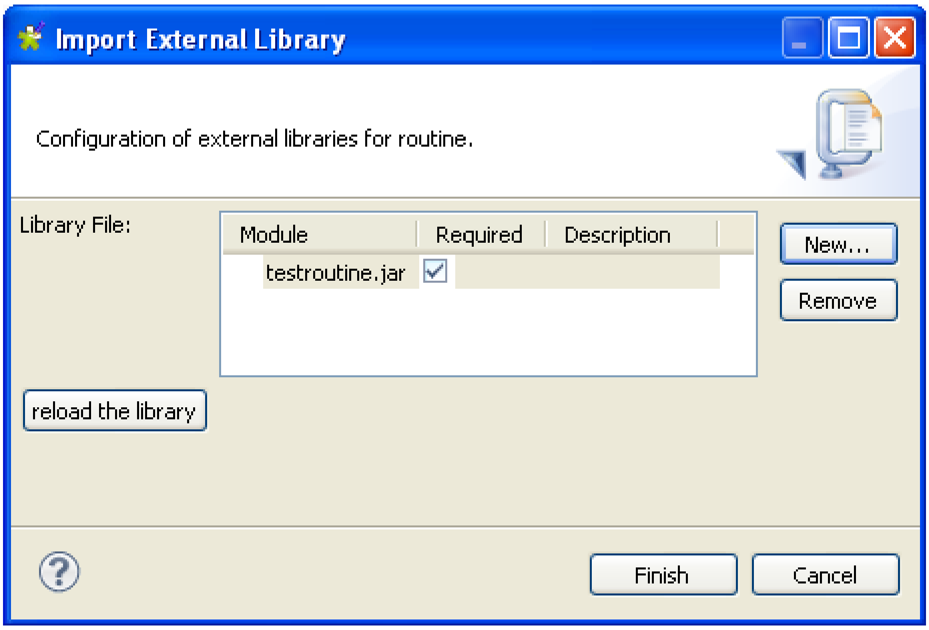
The .jar file of the imported library will be also listed in the library file of your current Studio.

To edit a user routine library, complete the following:

1. In the **Repository** tree view, expand **Code** > **Routines**.

2. Right-click the user routine you want to edit its library and then select **Edit Routine Library**.

   The **[Import External Library]** dialog box displays.



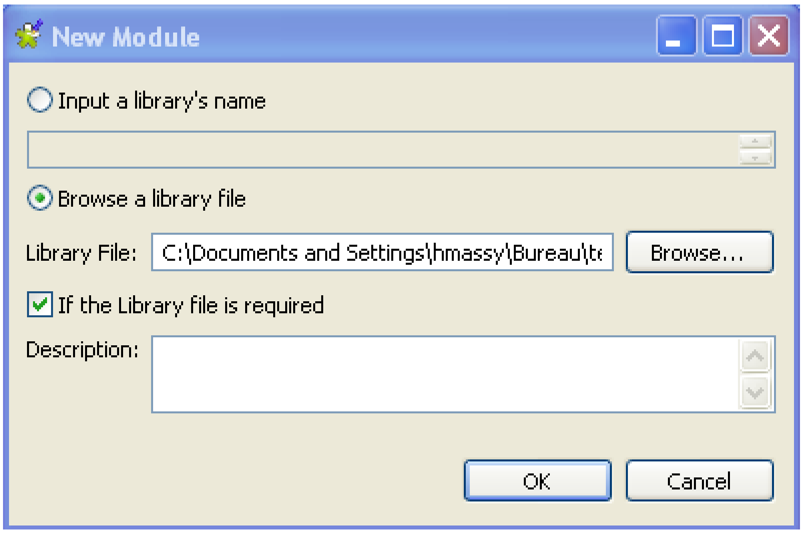
Configuration of external libraries for routine.

Library File:

| Module | Required | Description |
|---|---|---|
| testroutine.jar | ☑ | |

3. Click **New** to open a new dialog box where you can import the external library.

You can delete any of the already imported routine files if you select the file in the **Library File** list and click the **Remove** button.



○ Input a library's name

● Browse a library file

Library File: C:\Documents and Settings\hmassy\Bureau\te

☑ If the Library file is required

Description:

4. Enter the name of the library file in the **Input a library’s name** field followed by the file format (.jar), or

5. Select the **Browse a library file** option and click browse to set the file path in the corresponding field.

6. If required, enter a description in the **Description** field and then click **OK** to confirm your changes.

   The imported library file is listed in the **Library File** list in the **[Import External Library]** dialog box.

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

   The library file is imported into the library folder of your current Studio and also listed in the **Module** view of the same Studio.

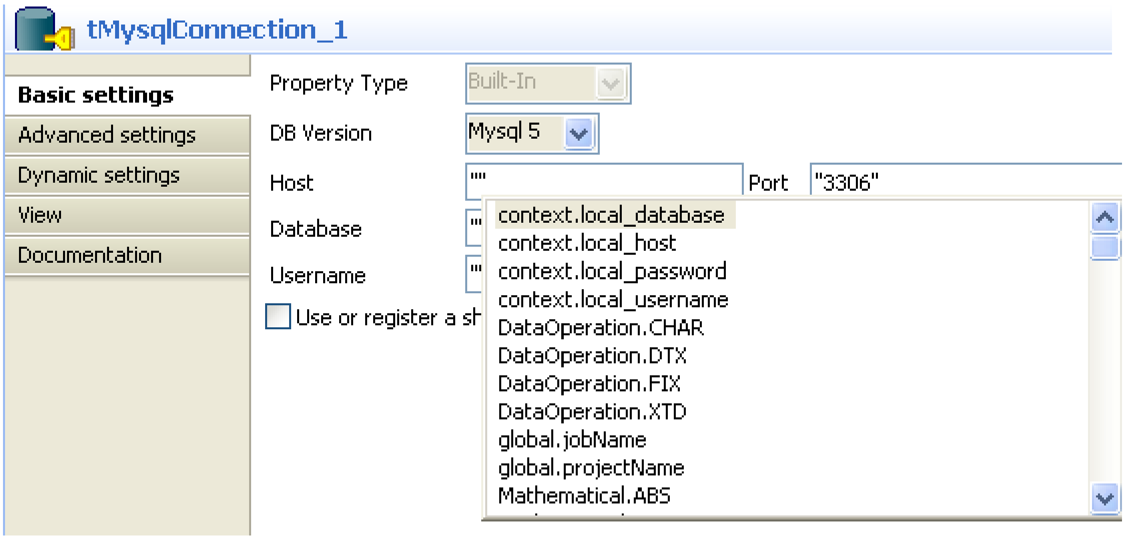
   For more information about the **Modules** view, see section *How to install external modules*.

# 6.5. Calling a routine from a Job

Pre-requisite: You must have at least one Job created, in order to run a routine. For further information regarding how to create a Job, see section *How to create a Job* .

You can call any of your user and system routines from your Job components in order to run them at the same time as your Job.

To access all the routines saved in the **Routines** folder in the **Repository** tree view, press **Ctrl+Space** in any of the fields in the **Basic settings** view of any of the **Talend** components used in your Job and select the one you want to run.

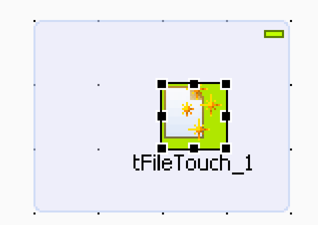


Alternatively, you can call any of these routines by indicating the relevant class name and the name of the routine, followed by the expected settings, in any of the **Basic settings** fields in the following way:

```
<ClassName>.<RoutineName>
```

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

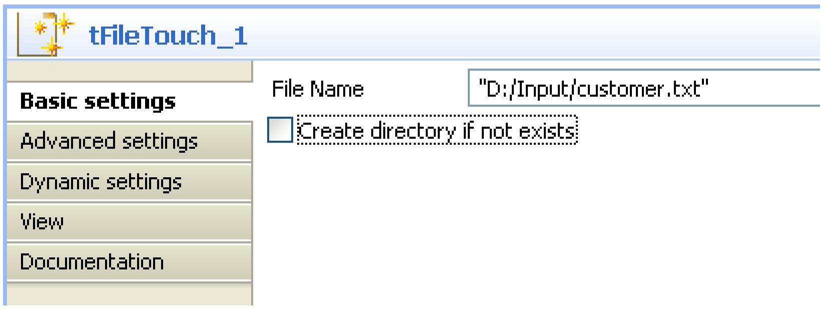
This scenario describes how to use a routine. The Job uses just one component, which calls a system routine.



1. In the **Palette**, click **File** > **Management**, then drop a **tFileTouch** component onto the workspace. This component allows you to create an empty file.

2. Double-click the component to open its **Basic settings** view in the **Component** tab.

3. In the **FileName** field, enter the path to access your file, or click **[...]** and browse the directory to locate the file.



File Name "D:/Input/customer.txt"

□ Create directory if not exists

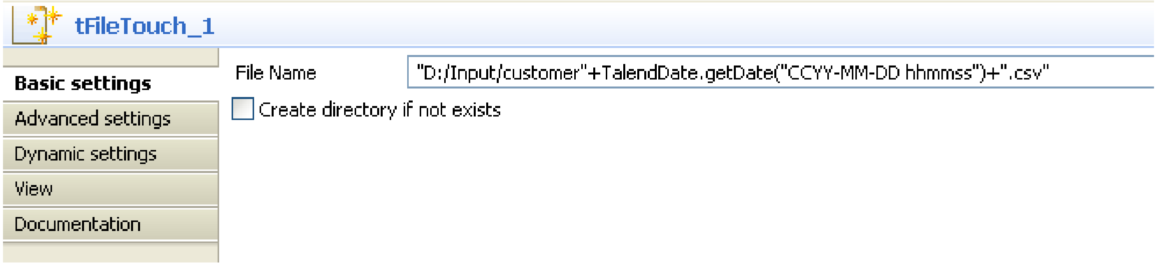
4. Close the double inverted commas around your file extension as follows: *"D:/Input/customer".txt*.

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.



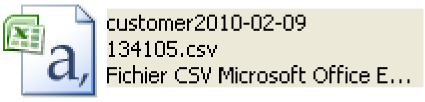
File Name "D:/Input/customer"+TalendDate.getDate("CCYY-MM-DD hhmmss")+".csv"

□ Create directory if not exists

*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.



customer2010-02-09 134105.csv
Fichier CSV Microsoft Office E...

# Chapter 7. Using SQL templates

SQL templates are groups of pre-defined query arguments that run in the ELT mode. This chapter explains the ELT mode, defines the SQL templates and provides user scenarios to explain how to use the SQL templates or how to create your own ones.

Before starting any data integration processes, you need to be familiar with *Talend Open Studio for Big Data* Graphical User Interface (GUI). For more information, see appendix *GUI*.

# 7.1. What is ELT

Extract, Load and Transform (ELT) is a data manipulation process in database usage, especially in data warehousing. Different from the traditional ETL (Extract, Transform, Load) mode, in ELT, data is extracted, loaded into the database and then is transformed where it sits in the database, prior to use. This data is migrated in bulk according to the data set and the transformation process occurs after the data has been loaded into the targeted DBMS in its raw format. This way, less stress is placed on the network and larger throughput is gained.

However, the ELT mode is certainly not optimal for all situations, for example,

- As SQL is less powerful than Java, the scope of available data transformations is limited.
- ELT requires users that have high proficiency in SQL tuning and DBMS tuning.
- Using ELT with *Talend Open Studio for Big Data*, you cannot pass or reject one single row of data as you can do in ETL. For more information about row rejection, see section *Row connection*.

Based on the advantages and disadvantages of ELT, the SQL templates are designed as the ELT facilitation requires.

# 7.2. Introducing Talend SQL templates

SQL is a standardized query language used to access and manage information in databases. Its scope includes data query and update, schema creation and modification, and data access control. *Talend Open Studio for Big Data* 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 Open Studio for Big Data* 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.

For further information concerning the components from the ELT component family, see *Talend Open Studio for Big Data Components Reference Guide*.

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.

# 7.3. Managing Talend SQL templates

*Talend Open Studio for Big Data* 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.

## 7.3.1. Types of system SQL templates

This section gives detail information related to the different types of the pre-defined SQL templates.

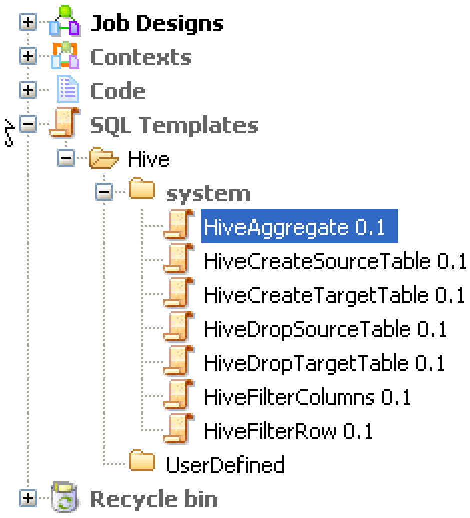
Even though the statements of each group of templates vary from database to database, according to the operations they are intended to accomplish, they are also grouped on the basis of their types in each folder.

The below table provides these types and their related information.

| Name | Function | Associated components | Required component parameters |
|---|---|---|---|
| *Aggregate* | Realizes aggregation (sum, average, count, etc.) over a set of data. | **tSQLTemplateAggregate** | Database name<br>Source table name<br>Target table name |
| *Commit* | Sends a Commit instruction to RDBMS | **tSQLTemplate tSQLTemplateAggregate tSQLTemplateCommit tSQLTemplateFilterColumns tSQLTemplateFilterRows tSQLTemplateMerge tSQLTemplateRollback** | Null |
| *Rollback* | Sends a Rollback instruction to RDBMS. | **tSQLTemplate tSQLTemplateAggregate tSQLTemplateCommit tSQLTemplateFilterColumns tSQLTemplateFilterRows tSQLTemplateMerge tSQLTemplateRollback** | Null |
| *DropSourceTable* | Removes a source table. | **tSQLTemplate tSQLTemplateAggregate tSQLTemplateFilterColumns tSQLTemplateFilterRows** | Table name (when use **tSQLTemplate**)<br>Source table name |
| *DropTargetTable* | Removes a target table | **tSQLTemplateAggregate tSQLTemplateFilterColumns tSQLTemplateFilterRows** | Target table name |
| *FilterColumns* | Selects and extracts a set of data from given columns in RDBMS. | **tSQLTemplateAggregate tSQLTemplateFilterColumns tSQLTemplateFilterRows** | Target table name (and schema)<br>Source table name (and schema) |
| *FilterRow* | Selects and extracts a set of data from given rows in RDBMS. | **tSQLTemplateAggregate tSQLTemplateFilterColumns tSQLTemplateFilterRows** | Target table name (and schema)<br>Source table name (and schema)<br>Conditions |
| *MergeInsert* | Inserts records from the source table to the target table. | **tSQLTemplateMerge tSQLTemplateCommit** | Target table name (and schema)<br>Source table name (and schema)<br>Conditions |
| *MergeUpdate* | Updates the target table with records from the source table. | **tSQLTemplateMerge tSQLTemplateCommit** | Target table name (and schema)<br>Source table name (and schema)<br>Conditions |

## 7.3.2. How to access a system SQL template

To access a system SQL template, expand the **SQL Templates** node in the **Repository** tree view.



Each folder contains **system** sub-folders 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 Open Studio for Big Data*.

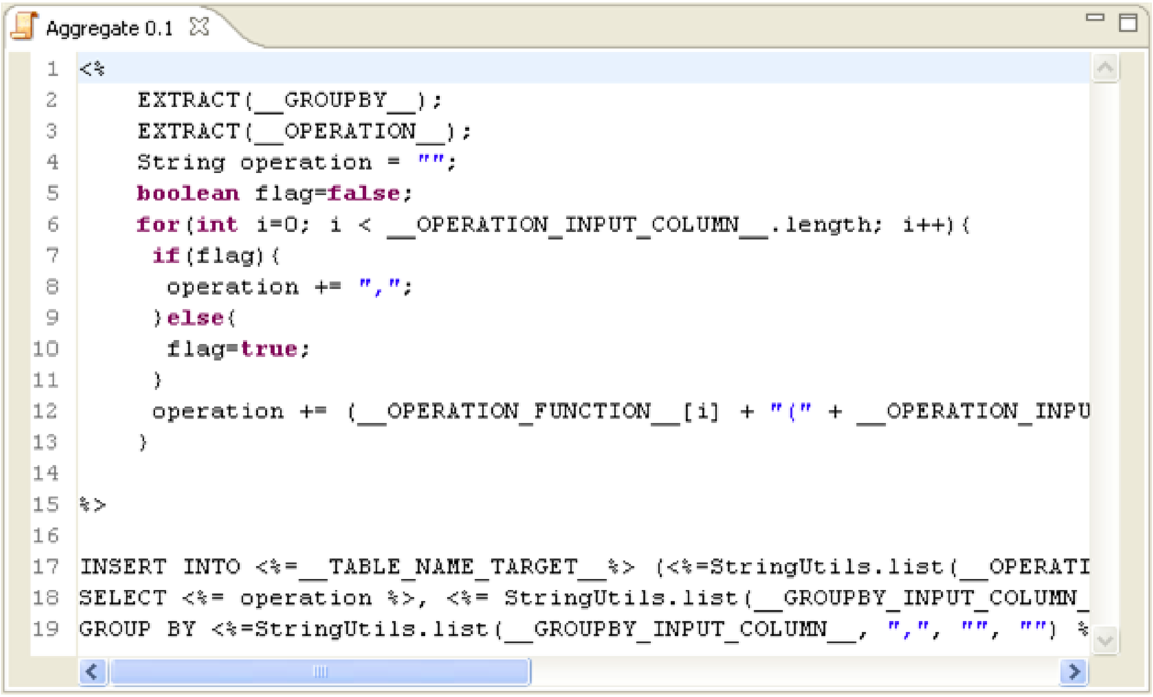
The **system** folders and their content are read only.

From the **Repository** tree view, proceed as follows to open an SQL template:

1. In the **Repository** tree view, expand **SQL Templates** and browse to the template you want to open.

2. Double click the class that you want to open, for example, *aggregate* in the **Generic** folder.

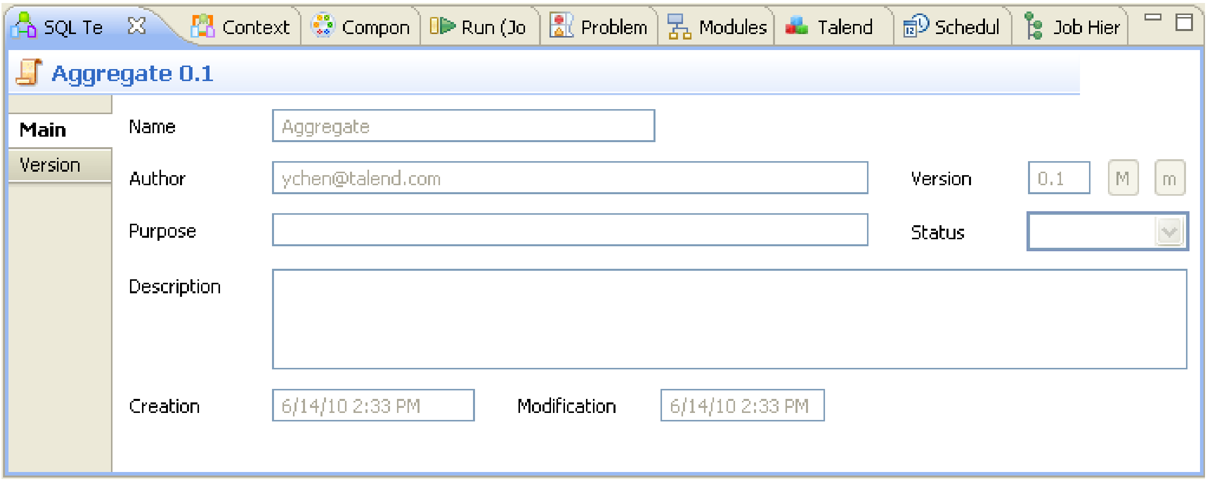
   The *aggregate* template view displays in the workspace.



```
<%
    EXTRACT(__GROUPBY__);
    EXTRACT(__OPERATION__);
    String operation = "";
    boolean flag=false;
    for(int i=0; i < __OPERATION_INPUT_COLUMN__.length; i++){
     if(flag){
      operation += ",";
     }else{
      flag=true;
     }
     operation += (__OPERATION_FUNCTION__[i] + "(" + __OPERATION_INPU
    }

%>

INSERT INTO <%=__TABLE_NAME_TARGET__%> (<%=StringUtils.list(__OPERATI
SELECT <%= operation %>, <%= StringUtils.list(__GROUPBY_INPUT_COLUMN_
GROUP BY <%=StringUtils.list(__GROUPBY_INPUT_COLUMN__, ",", "", "") %
```

You can read the predefined *aggregate* statements in the template view. The parameters, such as `TABLE_NAME_TARGET`, `operation`, are to be defined when you design related Jobs. Then the parameters can be easily set in the associated components, as mentioned in the previous section.

Everytime you click or open an SQL template, its corresponding property view displays at the bottom of the studio. Click the *aggregate* template, for example, to view its properties as presented below:



Name: Aggregate

Author: ychen@talend.com

Version: 0.1

Purpose:

Status:

Description:

Creation: 6/14/10 2:33 PM

Modification: 6/14/10 2:33 PM

For further information regarding the different types of SQL templates, see section *Types of system SQL templates*

For further information about how to use the SQL templates with the associated components, see section *How to use the SQL Templates*.

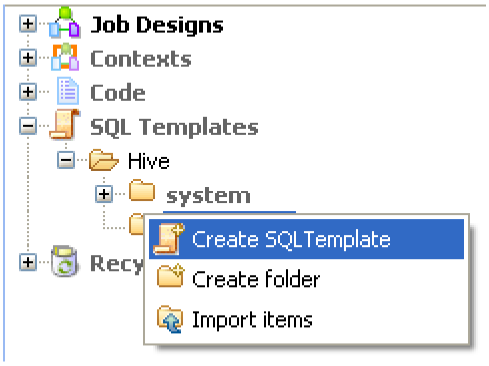
## 7.3.3. How to create user-defined SQL templates

As the transformation you need to accomplish in ELT may exceed the scope of what the given SQL templates can achieve, *Talend Open Studio for Big Data* allows you to develop your own SQL templates according to some writing rules. These SQL templates are stored in the **User-defined** folders grouped according to the database type in which they will be used.

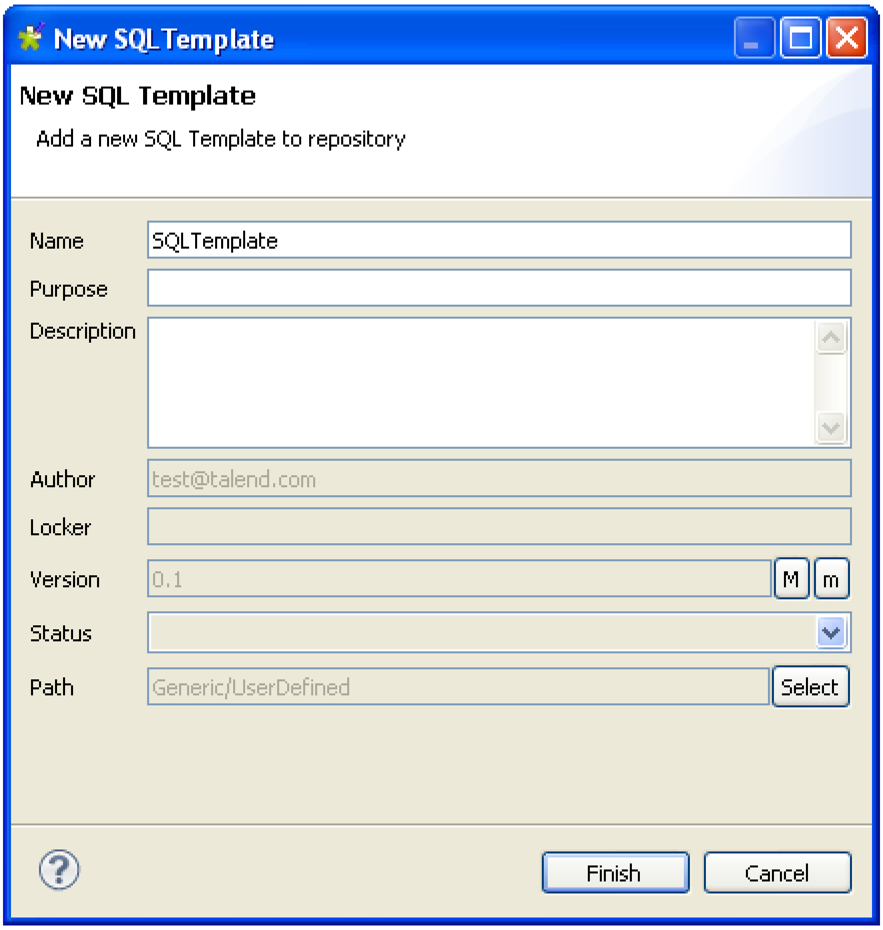
For more information on the SQL template writing rules, see appendix *SQL template writing rules*.

To create a user-defined SQL template:

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 SQL Template** to open the **[SQL Templates]** wizard.



New SQL Template

Add a new SQL Template to repository

Name: SQLTemplate

Purpose:

Description:

Author: test@talend.com

Locker:

Version: 0.1

Status:

Path: Generic/UserDefined

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.

# Appendix A. GUI

This appendix describes the Graphical User Interfaces (GUI) of *Talend Open Studio for Big Data*.

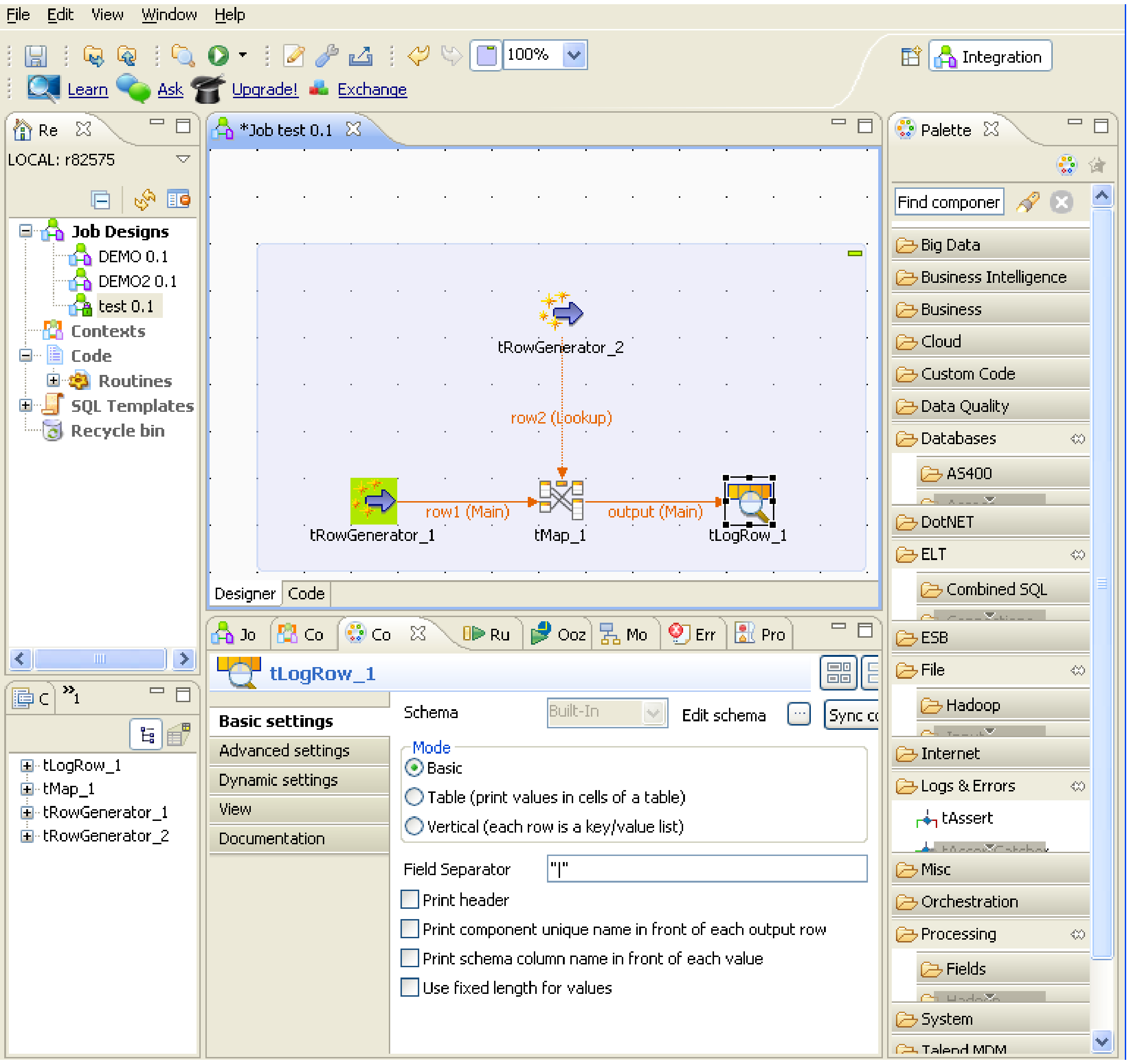
# A.1. Main window

*Talend Open Studio for Big Data* main window is the interface from which you manage all types of data integration processes.

The *Talend Open Studio for Big Data* multi-panel window is divided into:

- menu bar,
- toolbar,
- **Repository** tree view,
- design workspace,
- Palette,
- various configuration views in a tab system, for any of the elements in the data integration Job designed in the workspace,
- **Outline view** and **Code Viewer**.

The figure below illustrates *Talend Open Studio for Big Data* main window and its panels and views.



tLogRow_1

Schema: Built-In

Mode
- ● Basic
- ○ Table (print values in cells of a table)
- ○ Vertical (each row is a key/value list)

Field Separator: "|"

- □ Print header
- □ Print component unique name in front of each output row
- □ Print schema column name in front of each value
- □ Use fixed length for values

The various panels and their respective features are detailed hereafter.

All the panels, tabs, and views described in this documentation are specific to *Talend Open Studio for Big Data*. Some views listed in the **[Show View]** dialog box are Eclipse specific and are not subjects of this documentation. For information on such views, check Eclipse online documentation at http://www.eclipse.org/documentation/.

# A.2. Menu bar and Toolbar

At the top of the *Talend Open Studio for Big Data* main window, various menus and a quick access toolbar gather **Talend** commonly features along with some Eclipse functions.

## A.2.1. Menu bar of *Talend Open Studio for Big Data*

*Talend Open Studio for Big Data*'s menus include:

- some standard functions, such as **Save**, **Print**, **Exit**, which are to be used at the application level.
- some Eclipse native features to be used mainly at the design workspace level as well as specific *Talend Open Studio for Big Data* functions.

The table below describes menus and menu items available to you on the menu bar of *Talend Open Studio for Big Data*.

| Menu | Menu item | Description |
|---|---|---|
| **File** | **Close** | Closes the current open view on the Studio design workspace. |
| | **Close All** | Closes all open views on the Studio design workspace. |
| | **Save** | Saves any changes done in the current open view. |
| | **Save as** | Saves any changes done without changing the current open view. |
| | **Save All** | Saves any changes done in all open views. |
| | **Print** | Unavailable option. |
| | **Switch project** | Closes the current session and launches another one to enable you to open a different project in the Studio. |
| | **Edit project properties** | Opens a dialog box where you can customize the settings of the current project.<br>For more information, see section *Customizing project settings* |
| | **Import** | Opens a wizard that helps you to import different types of resources (files, items, preferences, XML catalogs, etc.) from different sources. |
| | **Export** | Opens a wizard that helps you to export different types of resources (files, items, preferences, breakpoints, XML catalogs, etc.) to different destinations. |
| | **Exit** | Closes The Studio main window. |
| | **Open File** | Opens a file from within the Studio. |
| **Edit** | **Undo** | Undoes the last action done in the Studio design workspace. |
| | **Redo** | Redoes the last action done in the Studio design workspace. |
| | **Cut** | Cuts selected object in the Studio design workspace. |
| | **Copy** | Copies the selected object in the Studio design workspace. |
| | **Paste** | Pastes the previously copied object in the Studio design workspace. |
| | **Delete** | Deletes the selected object in the Studio design workspace. |
| | **Select All** | Selects all components present in the Studio design workspace. |
| **View** | **Zoom In** | Obtains a larger image of the open Job. |
| | **Zoom Out** | Obtains a smaller image of the open Job. |
| | **Grid** | Displays grid in the design workspace. All items in the open Job are snapped to it. |

| Menu | Menu item | Description |
|---|---|---|
| | **Snap to Geometry** | Enables the Snap to Geometry feature. |
| | | |
| **Window** | **Perspective** | Opens different perspectives corresponding to the different items in the list. |
| | **Show View...** | Opens the **[Show View]** dialog box which enables you to display different views on the Studio. |
| | **Maximize Active View or Editor...** | Maximizes the current perspective. |
| | **Preferences** | Opens the **[Preferences]** dialog box which enables you to set your preferences.<br>For more information about preferences, see section *Setting Talend Open Studio for Big Data preferences* |
| | | |
| **Help** | **Welcome** | Opens a welcoming page which has links to *Talend Open Studio for Big Data* documentation and **Talend** practical sites. |
| | **Help Contents** | Opens the Eclipse help system documentation. |
| | **About** *Talend Open Studio for Big Data* | Displays:<br>-the software version you are using,<br>-detailed information on your software configuration that may be useful if there is a problem,<br>-detailed information about plug-in(s),<br>-detailed information about *Talend Open Studio for Big Data* features. |
| | **Export logs** | Opens a wizard that helps you to export all logs generated in the Studio and system configuration information to an archived file. |
| | **Software Updates** | **Find and Install...:** Opens the **[Install/Update]** wizard that helps searching for updates for the currently installed features, or searching for new features to install. |
| | | **Manage Configuration...:** Opens the **[Product Configuration]** dialog box where you can manage *Talend Open Studio for Big Data* configuration. |

## A.2.2. Toolbar of *Talend Open Studio for Big Data*

The toolbar contains icons that provide you with quick access to the commonly used operations you can perform from *Talend Open Studio for Big Data* main window.

The table below describes the toolbar icons and their functions.

| Name | Icon | Description |
|---|---|---|
| **Save** | 💾 | Saves current job design. |
| **Save as** | 💾 | Saves as another new Job. |
| **Export items** | 📁 | Exports repository items to an archive file, for deploying outside *Talend Open Studio for Big Data*. Instead if you intend to import the exported element into a newer version of *Talend Open Studio for Big Data* or of another workstation, make sure the source files are included in the archive. |
| **Import items** | 📁 | Imports repository items from an archive file into your current *Talend Open Studio for Big Data*. For more information regarding the import/export items feature, see section *How to import items*. |
| **Find a specific job** | 🔍 | Displays the relevant dialog box that enables you to open any Job listed in the **Repository** tree view. |
| **Run job** | ▶ ▾ | Executes the Job currently shown on the design space. For more information about job execution, see section *How to run a Job*. |

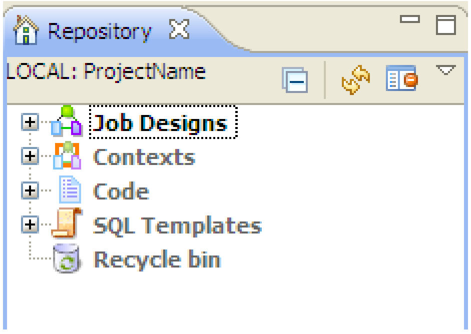
| Name | Icon | Description |
|---|---|---|
| **Create** | | Launches the relevant creation wizard. Through this menu, you can create any repository item including Job Designs, contexts, and routines. |
| **Project settings** | | Launches the **[Project Settings]** dialog box. From this dialog box, you can add a description to the current Project and customize the **Palette** display. For more information, see section *Customizing project settings*. |
| **Detect and update all jobs** | | Searches for all updates available for your Jobs. |
| **Export Talend projects** | | Launches the **[Export Talend projects]** wizard. For more information about project export, see section *How to export a project*. |

# A.3. Repository tree view

The **Repository** tree view gathers all the technical items that can be used to design Jobs. It gives access to any item including , **JobDesigns**, as well as reusable routines.

The **Repository** centralizes and stores all necessary elements for any Job design contained in a project.

The figure below illustrates the elements stored in the **Repository**.



The **Refresh** button allows you to update the tree view with the last changes made.

The **Activate filter** button allows you to open the filter settings view so as to configure the display of the **Repository** view.

The **Repository** tree view stores all your data (Jobs).

The table below describes the nodes in the **Repository** tree view.

| Node | Description |
|---|---|
| **Job Designs** | The **Job Designs** folder shows the tree view of the designed Jobs for the current project. Double-click the name of the Job to open it on the design workspace. For more information, see chapter *Designing a data integration Job*. |
| **Contexts** | The **Context** folder groups files holding the contextual variables that you want to reuse in various Jobs, such as filepaths or DB connection details. For more information, see section *How to centralize contexts and variables*. |
| **Code** | The **Code** folder is a library that groups the routines available for this project and other pieces of code that could be reused in the project. Click the relevant tree entry to expand the appropriate code piece.<br>For more information, see chapter *Designing a data integration Job*. |
| **SQL Templates** | The **SQL Templates** folder groups all system SQL templates and gives the possibility to create user-defined SQL templates. For more information, see section *How to use the SQL Templates*. |
| **Recycle bin** | The **Recycle bin** groups all elements deleted from any folder in the **Repository** tree view. |

| Node | Description |
|---|---|
| | The deleted elements are still present on your file system, in the recycle bin, until you right-click the recycle bin icon and select **Empty Recycle bin**.<br>Expand the recycle bin to view any folders, subfolders or elements held within. You can action an element directly from the recycle bin, restore it or delete it forever by clicking right and selecting the desired action from the list. |

# A.4. Design workspace

In the *Talend Open Studio for Big Data*'s design workspace, Job Designs can be laid out.

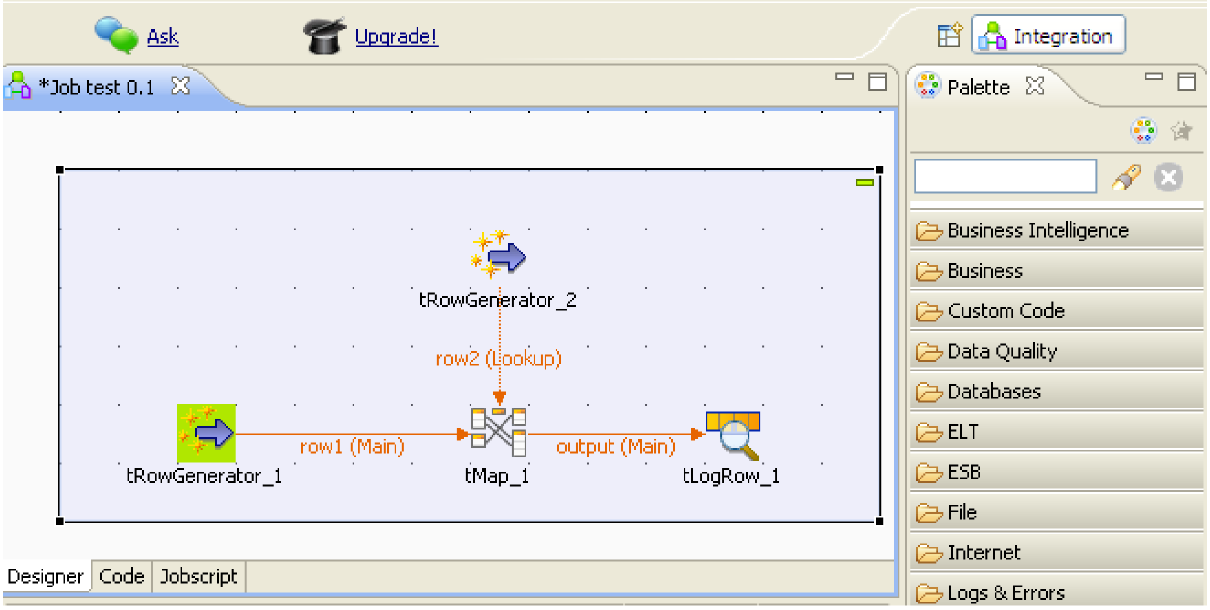
For more information, see section *How to create a Job* .

For Job Designs: active designs display in a easily accessible tab system above this workspace.

Under this workspace, you can access several other tabs:

- the **Designer** tab: opens by default when creating a Job. It displays the Job in a graphical mode.
- the **Code** tab: enables you to visualize the code and highlights the possible language errors.

Warnings are indicated in yellow whereas errors are indicated in red.



A**Palette** is docked at the top of the design workspace to help you draw the model corresponding to your workflow needs.

# A.5. Palette

From the **Palette**, you can drop technical components and notes to the design workspace. You can define and format them using the various tools offered in the **Component** view for the Job.

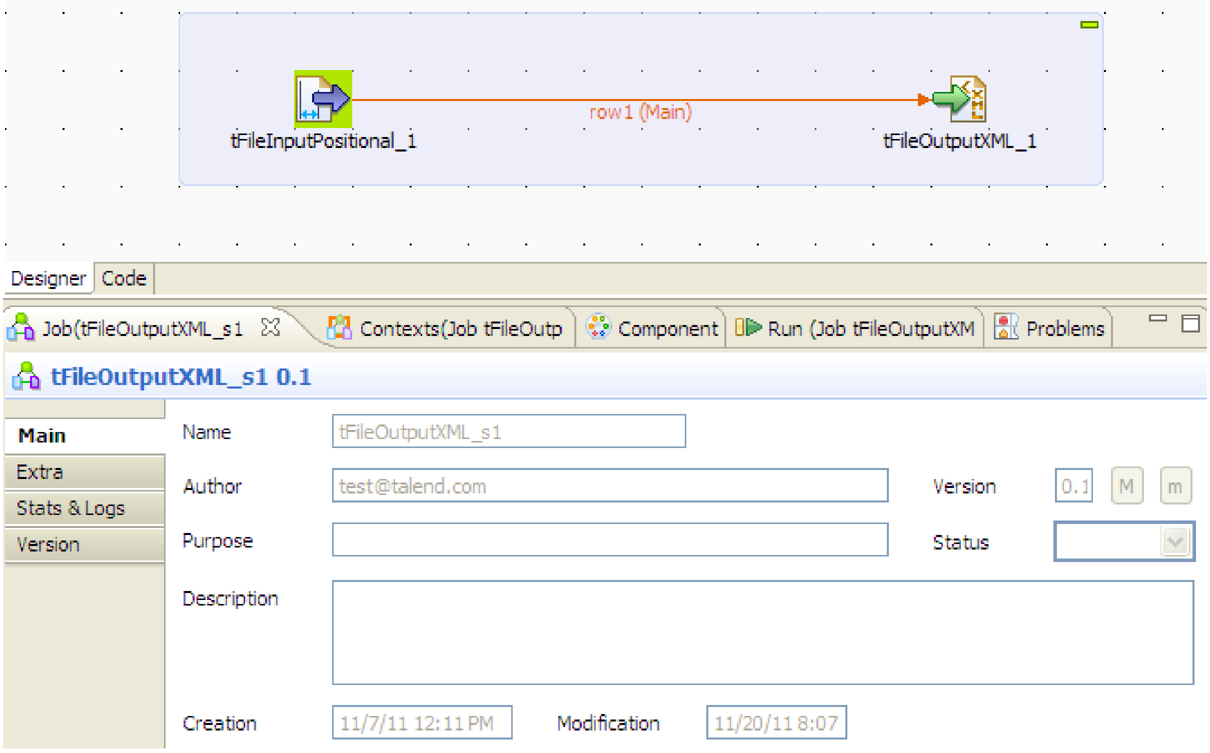
Related topics:

- chapter *Designing a data integration Job*.

- section *How to change the Palette layout and settings*.

# A.6. Configuration tabs

The configuration tabs are located in the lower half of the design workspace. Each tab opens a view that displays the properties of the selected element in the design workspace. These properties can be edited to change or set the parameters related to a particular component or to the Job as a whole.



Name: tFileOutputXML_s1
Author: test@talend.com
Version: 0.1
Purpose:
Status:
Description:
Creation: 11/7/11 12:11 PM
Modification: 11/20/11 8:07

The **Component**, **Run Jobs**, **Problems** and **Error Log** views gather all information relative to the graphical elements selected in the design workspace or the actual execution of the open Job.

The **Modules** and **Scheduler** tabs are located in the same tab system as the **Component**, **Logs** and **Run Job** tabs. Both views are independent from the active or inactive Jobs open on the design workspace.

You can show more tabs in this tab system and directly open the corresponding view if you select **Window** > **Show view** and then, in the open dialog box, expand any node and select the element you want to display.

The sections below describe the view of each of the configuration tabs.

| View | Description |
|---|---|
| **Component** | This view details the parameters specific to each component of the **Palette**. To build a Job that will function, you are required to fill out the necessary fields of this **Component** view for each component forming your Job.<br>For more information about the **Component** view, see section *How to define component properties*. |
| **Run Job** | This view obviously shows the current job execution. It becomes a log console at the end of an execution.<br>For details about job execution, see section *How to run a Job*. |
| **Oozie scheduler** | This enables you to run the current Job, or schedule it so that it will launch periodically, on a remote HDFS server.<br>For more information, see section *How to run a Job on a remote HDFS server*. |
| **Error Log** | This view is mainly used for Job execution errors. It shows the history of warnings or errors occurring during job executions. |

| View | Description |
|---|---|
| | The log tab has also an informative function for a Java component operating progress, for example.<br>**Error Log** tab is hidden by default. As for any other view, go to **Window** > **Show views**, then expand **General** node and select **Error Log** to display it on the tab system. |
| **Modules** | This view shows if a module is necessary and required for the use of a referenced component. Checking the **Modules** view helps to verify what modules you have or should have to run smoothly your Jobs.<br>For more information, see section *How to install external modules*. |
| **Job view** | The **Job** view displays various information related to the open Job on the design workspace. This view has the following tabs: |
| | **Main tab**<br>This tab displays basic information about the Job opened on the design workspace, i.e. its name, author, version number, etc. The information is read-only. To edit it you have to close your Job, right-click its label on the **Repository** tree view and click **Edit properties** on the drop-down list. |
| | **Extra tab**<br>This tab displays extra parameters including multi thread and implicit context loading features. For more information, see section *How to use the features in the Extra tab* |
| | **Stats/Log tab**<br>This tab allows you to enable/disable the statistics and logs for the whole Job.<br>You can already enable these features for every single component of your Job by simply using and setting the relevant components: **tFlowMeterCatcher**, **tStatCatcher**, **tLogCatcher**.<br>For more information about these components, see *Talend Open Studio for Big Data Components Reference Guide*.<br>In addition, you can now set these features for the whole active Job (i.e. all components of your Job) in one go, without using the Catcher components mentioned above. This way, all components get tracked and logged in the File or Database table according to your setting.<br>You can also save the current setting to Project Settings by clicking the Save to project settings button.<br>For more details about the Stats & Logs automation, see section *How to automate the use of statistics & logs*. |
| | **Version tab**<br>This tab displays the different versions of the Job opened on the design workspace and their creation and modification dates. |
| **Problems** | This view displays the messages linked to the icons docked at a components in case of problem, for example when part of its setting is missing. Three types of icons/messages exist: **Error**, **Warning** and **Infos**.<br>For more information, see section *Warnings and error icons on components*. |
| **Job Hierarchy** | This view displays a tree folder showing the child Job(s) of the parent Job selected. To show this view, right-click the parent Job in the **Repository** tree view and select **Open Job Hierarchy** on the drop-down list.<br>You can also show this view in the **Window** > **Show view...** combination where you can select **Talend** > **Job Hierarchy**.<br>You can see **Job Hierarchy** only if you create a parent Job and one or more child Job(s) via the **tRunJob** component. For more information about **tRunJob**, see *Talend Open Studio for Big Data Components Reference Guide*. |

# A.7. Outline and code summary panel

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

The Information panel is composed of two tabs, **Outline** and **Code Viewer**, which provide information regarding the displayed diagram and also the generated code.

For more information, see section *How to display the code or the outline of your Job*.

# A.8. Shortcuts and aliases

Below is a table gathering all keyboard shortcuts currently in use:

| **Shortcut** | **Operation** | **Context** |
|---|---|---|
| F2 | Shows **Component** settings view. | Global application |
| F4 | Shows **Run Job** view. | Global application |
| F6 | Runs current Job or shows **Run Job** view if no Job is open. | Global application |
| Ctrl + F2 | Shows **Module** view. | Global application |
| Ctrl + F3 | Shows **Problems** view. | Global application |
| Ctrl + H | Shows the **Designer** view of the current Job. | Global application |
| Ctrl + G | Shows the **Code** view of the current Job. | Global application |
| Ctrl + R | Restores the initial **Repository** view. | From **Repository** view |
| Ctrl + Shift + F3 | Synchronizes components javajet components. | Global application |
| Ctrl + Shift + J | Opens a Job. | Global application (In Windows) |
| F7 | Switches to **Debug** mode. | From **Run Job** view |
| F5 | Refreshes the **Repository** view. | From **Repository** view |
| F8 | Kills current Job. | From **Run Job** view |
| F5 | Refreshes **Modules** install status. | From **Modules** view |
| Ctrl+L | Execute SQL queries. | **Talend** commands (in Windows) |
| Ctrl+Space bar | Access global and user-defined variables. It can be error messages or line number for example, depending on the component selected. | From any component field in **Job** or **Component** views |

# Appendix B. Theory into practice: Job examples

This chapter aims at users of *Talend Open Studio for Big Data* who seek a real-life use case to help them take full control over the product. This chapter comes as a complement of *Talend Open Studio for Big Data Components Reference Guide*.

# B.1. tMap Job example

## B.1.1. Introducing the scenario

To illustrate the way *Talend Open Studio for Big Data* operates, find below a real-life example scenario. In this scenario, we will load a MySQL table with a file, that gets transformed on the fly. Then in a further step, we will select the data to be loaded using a dynamic filter.

Before actually starting the Job, let’s inspect the input data and the expected output data.

### B.1.1.1. Input data

Our input file, the data of which will be loaded into the database table, lists clients from all over the State of California.

The file structure usually called **Schema** in *Talend Open Studio for Big Data* includes the following columns:

- First name
- Last name
- Address
- City

### B.1.1.2. Output data

We want to load into the database, California clients living in a couple of Counties only: Orange and Los Angeles counties.

The table structure is slightly different, therefore the data expected to be loaded into the DB table should have the following structure:

- `Key` (key, Type: Integer)
- `Name` (Type: String, max. length: 40)
- `Address` (Type: String, max.length: 40)
- `County` (Type: String, max. length:40)

In order to load this table, we will need to use the following mapping process:

The `Key` column is fed with an auto-incremented integer.

The `Name` column is filled out with a concatenation of first and last names.

The `Address` column data comes from the equivalent Address column of the input file, but supports a upper-case transformation before the loading.

The `County` column is fed with the name of the County where the city is located using a reference file which will help filtering Orange and Los Angeles counties' cities.

### B.1.1.3. Reference data

As only Orange and Los Angeles counties data should be loaded into the database, we need to map cities of California with their respective county, in order to filter only Orange and Los Angeles ones.

To do so, we will use a reference file, listing cities that are located in Orange and Los Angeles counties such as:

| City | County |
|---|---|
| Agoura Hills | Los Angeles |
| Alhambra | Los Angeles |
| Aliso Viejo | Orange |
| Anaheim | Orange |
| Arcadia | Los Angeles |

The reference file in this Job is named *LosAngelesandOrangeCounties.txt*.

## B.1.2. Translating the scenario into a Job

In order to implement this scenario, let's break down the Job into four steps:

1. Creation of the Job, configuration of the input file parameters, and reading of the input file,
2. Mapping of data and transformations,
3. Definition of the reference file parameters, relevant mapping using the **tMap** component, and selection of inner join mode,
4. Redirection of the output into a MySQL table.

### B.1.2.1. Step 1: Job creation, input definition, file reading

Launch *Talend Open Studio for Big Data*, and create a local project or import the demo project if you are launching *Talend Open Studio for Big Data* for the first time. For more information, see section *Launching Talend Open Studio for Big Data* and section *Working with projects*.

The main window of *Talend Open Studio for Big Data* is divided into several areas:

- To the left: the **Repository** tree view that holds Jobs, shared Code, and so on.
- In the center: the **Editor** (main Design area)
- At the bottom: **Component** and **Job** tabs
- To the right: the **Palette** of technical components.

To the left of the Studio, the **Repository** tree view that gives an access to:

- The **Job Designer**: For details about this part, see section *Getting started with a basic Job design*.
- Contexts and routines: For details, see section *Using the Metadata Manager*

To create the Job, right-click **Job Designs** in the **Repository** tree view and select **Create Job**.

In the dialog box displaying then, only the first field (**Name**) is required. Type in **California1** and click **Finish**.

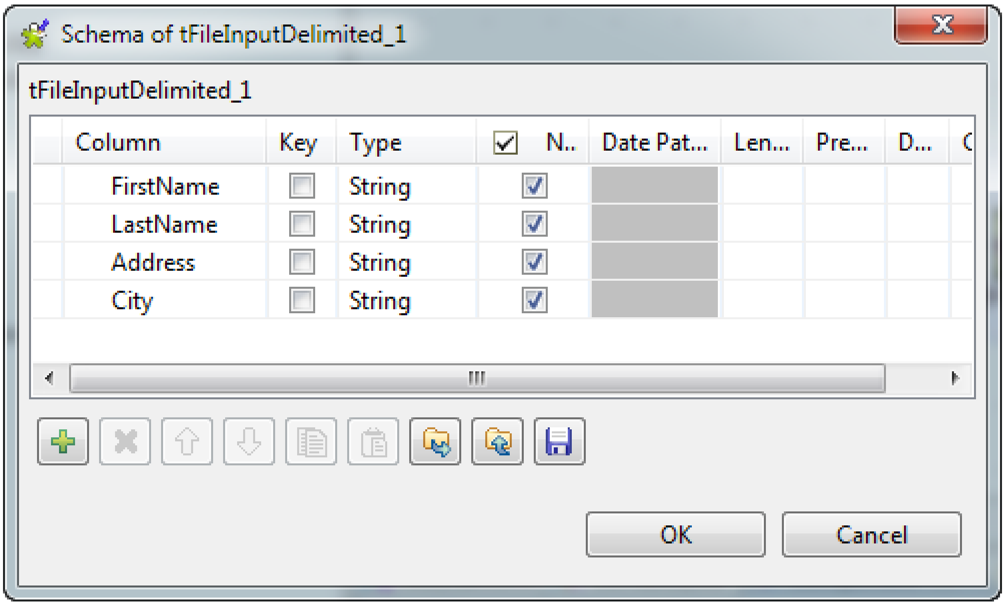
An empty Job then opens on the main window and the **Palette** of technical components (by default, to the right of the Studio) comes up showing a dozen of component families such as: **Databases**, **Files**, **Internet**, **Data Quality** and so on, hundreds of components are already available.

To read the file *California_Clients*, let's use the **tFileInputDelimited** component. This component can be found in the **File/Input** group of the **Palette**. Click this component then click to the left of the design workspace to place it on the design area.

Let's define now the reading properties for this component: File path, column delimiter, encoding...

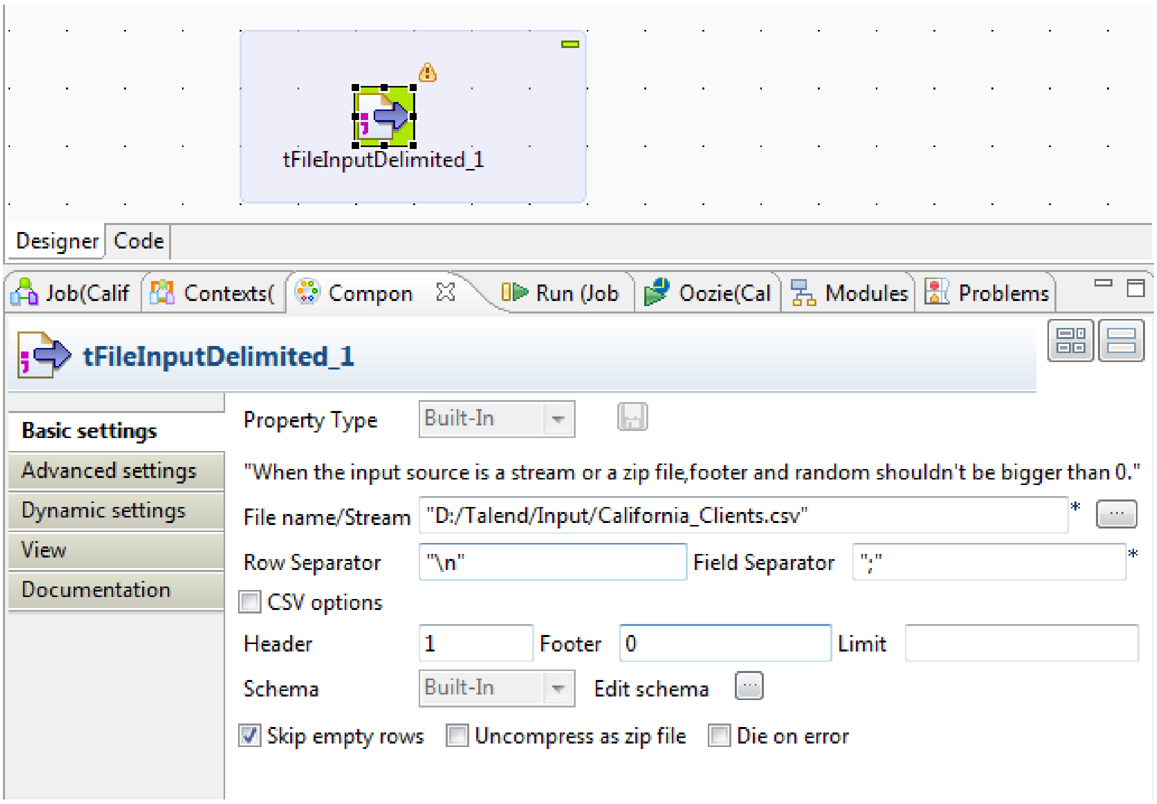
To do so:

1. Drop the **tFileInputDelimited** component from the **File** group of the **Palette** to the design workspace.
2. Double-click the **tFileInputDelimited** component to open its **Basic settings** view in the **Component** tab.
3. In the **File name/Stream** field, define the path to the input file.
4. In the **Header** field, define the number of rows to be skipped when reading the input file.
5. Click the **[...]** button next to the **Edit schema** to open the **[Schema]** dialog box, and define the data structure of the input file. When done, click **OK** to close the**[Schema]** dialog box.



| Column | Key | Type | ☑ N.. | Date Pat... | Len... | Pre... | D... | ( |
|---|---|---|---|---|---|---|---|---|
| FirstName | □ | String | ☑ | | | | | |
| LastName | □ | String | ☑ | | | | | |
| Address | □ | String | ☑ | | | | | |
| City | □ | String | ☑ | | | | | |

Now the basic property settings of the input component are complete.



Property Type: Built-In

"When the input source is a stream or a zip file,footer and random shouldn't be bigger than 0."

File name/Stream: "D:/Talend/Input/California_Clients.csv"

Row Separator: "\n" Field Separator: ";"

□ CSV options

Header 1 Footer 0 Limit

Schema: Built-In Edit schema

☑ Skip empty rows □ Uncompress as zip file □ Die on error

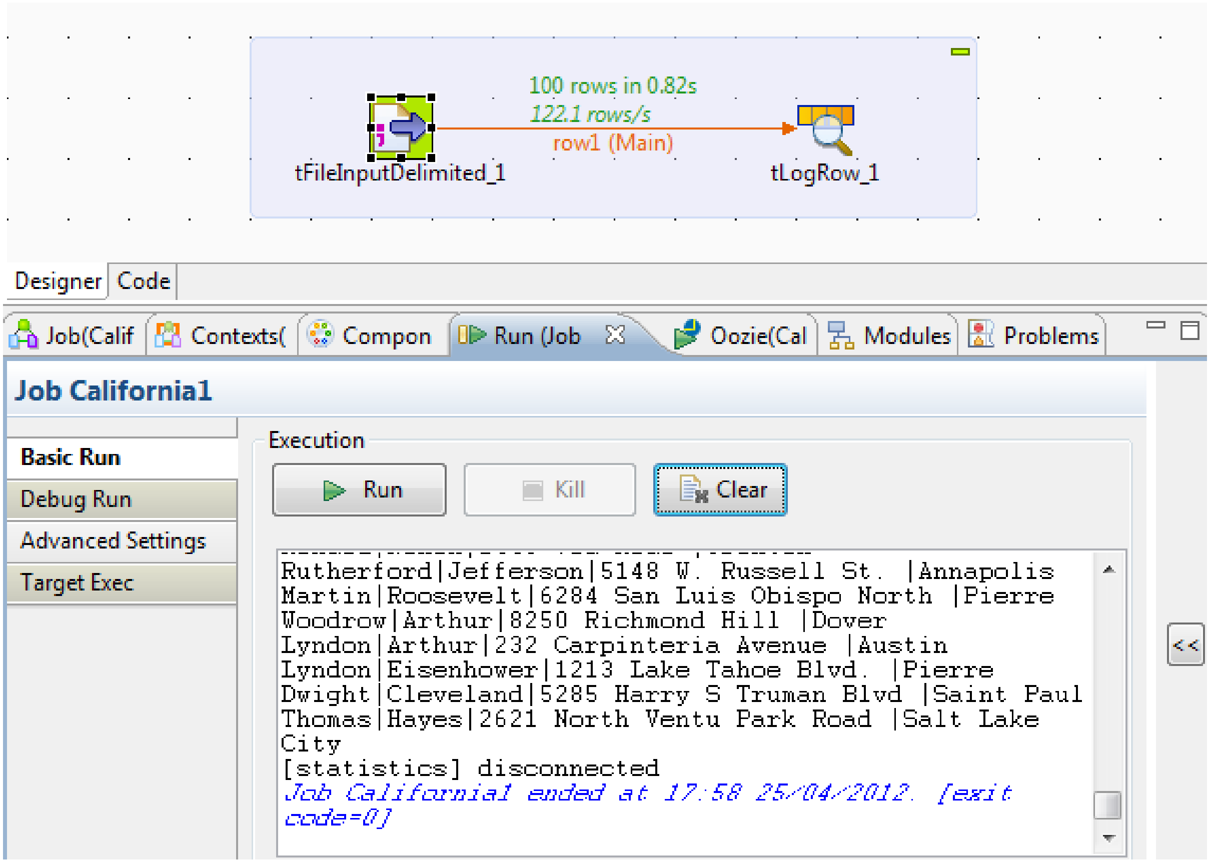
At this stage, we will terminate our flow by simply sending the data read from this input file onto the standard output (StdOut).

To do so, add a **tLogRow** component (from the **Logs & Errors** group).

To link both components, right-click the input component and select **Row/Main**. Then click the output component: **tLogRow**.

This Job is now ready to be executed. To run it, select the **Run** tab on the bottom panel.

Enable the statistics by selecting the **Statistics** check box in the **Advanced Settings** vertical tab of the **Run** view, then run the Job by clicking **Run** in the **Basic Run** tab.



```
Rutherford|Jefferson|5148 W. Russell St. |Annapolis
Martin|Roosevelt|6284 San Luis Obispo North |Pierre
Woodrow|Arthur|8250 Richmond Hill |Dover
Lyndon|Arthur|232 Carpinteria Avenue |Austin
Lyndon|Eisenhower|1213 Lake Tahoe Blvd. |Pierre
Dwight|Cleveland|5285 Harry S Truman Blvd |Saint Paul
Thomas|Hayes|2621 North Ventu Park Road |Salt Lake
City
[statistics] disconnected
Job California1 ended at 17:58 25/04/2012. [exit
code=0]
```

The content of the input file display thus onto the console.

## B.1.2.2. Step 2: Mapping and transformations

We will now enrich our Job to include on-the-fly transformations. To implement these transformation, we need to add a **tMap** component to our Job. This component is multiple and can handle:

- multiple inputs and outputs
- search for reference (simple, cartesian product, first, last match...)
- join (inner, outer)
- transformations
- rejections
- and more...

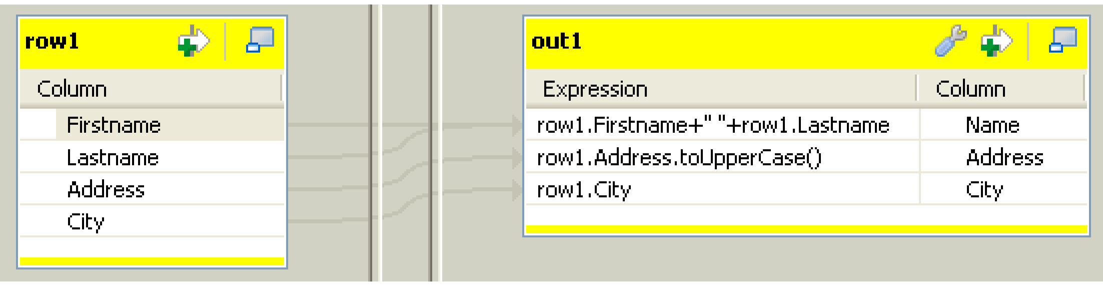
Remove the link that binds together the job’s two components via a right-click the link, then **Delete** option. Then place the **tMap** of the **Processing** component group in between before linking the input component to the **tMap** as we did it previously.

Eventually to link the **tMap** to the standard output, right-click the **tMap** component, select **Row/*New Output* (Main) and click the tLogRow component**. Type in *out1* in the dialog box to implement the link. Logically, a message box shows up (for the back-propagation of schemas), ignore it by clicking on **No**.

Now, double-click the **tMap** to access its interface.

To the left, you can see the schema (description) of your input file (*row1*). To the right, your output is for the time being still empty (*out1*).

Drop the *Firstname* and *Lastname* columns to the right, onto the *Name* column as shown on the screen below. Then drop the other columns *Address* and *City* to their respective line.



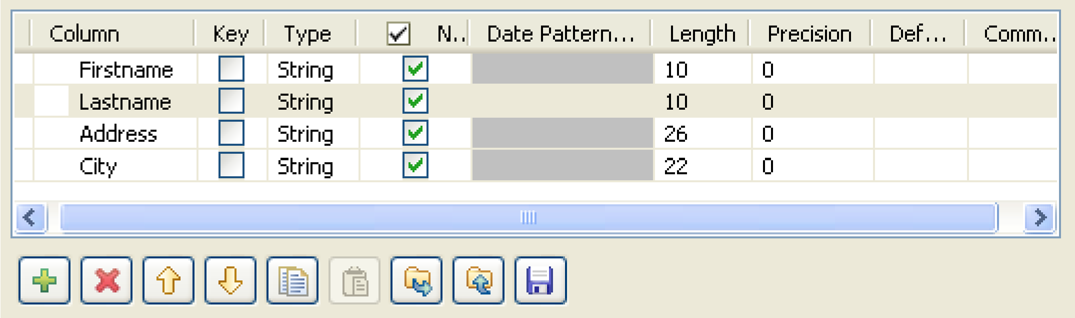
| row1 |
|---|
| Column |
| Firstname |
| Lastname |
| Address |
| City |

| out1 | |
|---|---|
| Expression | Column |
| row1.Firstname+" "+row1.Lastname | Name |
| row1.Address.toUpperCase() | Address |
| row1.City | City |

Then carry out the following transformations on each column:

- Change the Expression of the *Name* column to `row1.Firstname + " " + row1.LastName`. Concatenate the *Firstname* column with the *Lastname* column following strictly this syntax (in Java), in order for the columns to display together in one column.
- Change the Expression of the *Address* column to `row1.Address.toUpperCase()` which will thus change the address case to upper case.

Then remove the *Lastname* column from the *out1* table and increase the length of the remaining columns. To do so, go to the **Schema Editor** located at the bottom of the tMap editor and proceed as follows:



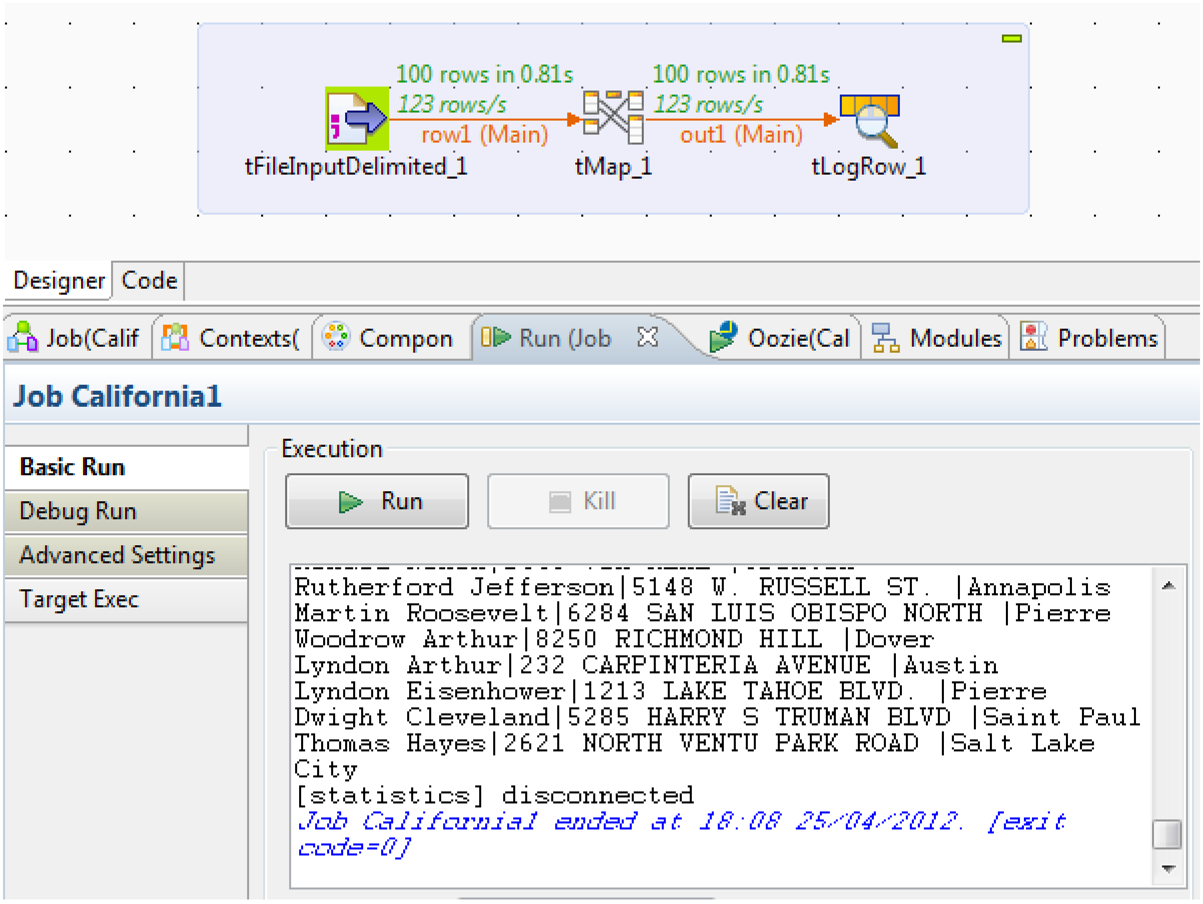
| Column | Key | Type | ☑ N.. | Date Pattern... | Length | Precision | Def... | Comm.. |
|---|---|---|---|---|---|---|---|---|
| Firstname | □ | String | ☑ | | 10 | 0 | | |
| Lastname | □ | String | ☑ | | 10 | 0 | | |
| Address | □ | String | ☑ | | 26 | 0 | | |
| City | □ | String | ☑ | | 22 | 0 | | |

1. Select the column to be removed from the schema, and click the cross icon.
2. Select the column of which you need increase the length size.
3. Type in the length size you intend in the length column. In this example, change the length of every remaining column to 40.

As the first name and the last name of a client is concatenated, it is necessary to increase the length of the name column in order to match the full name size.

No transformation is made onto the *City* column. Click **OK** to validate the changes and close the Map editor interface.

If you run your Job at this stage (via the **Run** view as we did it before), you’ll notice the changes that you defined are implemented.



```
Rutherford Jefferson|5148 W. RUSSELL ST. |Annapolis
Martin Roosevelt|6284 SAN LUIS OBISPO NORTH |Pierre
Woodrow Arthur|8250 RICHMOND HILL |Dover
Lyndon Arthur|232 CARPINTERIA AVENUE |Austin
Lyndon Eisenhower|1213 LAKE TAHOE BLVD. |Pierre
Dwight Cleveland|5285 HARRY S TRUMAN BLVD |Saint Paul
Thomas Hayes|2621 NORTH VENTU PARK ROAD |Salt Lake
City
[statistics] disconnected
Job California1 ended at 18:08 25/04/2012. [exit
code=0]
```

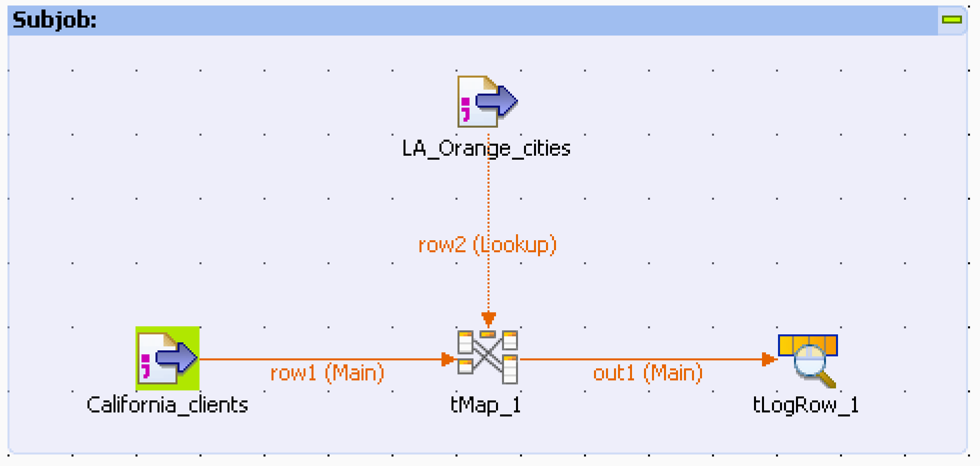
For example, the addresses are displayed in upper case and the first names and last names are gathered together in the same column.

## B.1.2.3. Step 3: Reference file definition, re-mapping, inner join mode selection

Define the Metadata corresponding to the *LosAngelesandOrangeCounties.txt* file just the way we did it previously for *California_clients* file.

First drop another **tFileInputDelimited** component onto the design workspace, and define its basic properties: the path to the reference input file, the number of rows to be skipped, and the schema.

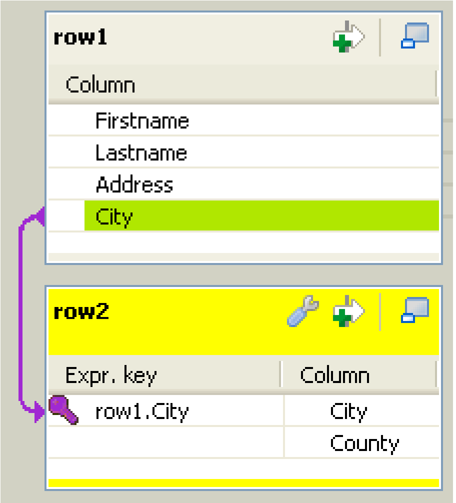
Then link this component to the **tMap** component.



Double-click again on the **tMap** component to open its interface. Note that the reference input table (*row2*) corresponding to the LA and Orange county file, shows to the left of the window, right under your main input (*row1*).

Now let's define the join between the main flow and the reference flow. In this use case, the join is pretty basic to define as the *City* column is present in both files and the data match perfectly. But even though this was not the case, we could have carried out operations directly at this level to establish a link among the data (padding, case change...)

To implement the join, drop the *City* column from your first input table onto the *City* column of your reference table. A violet link then displays, to materialize this join.

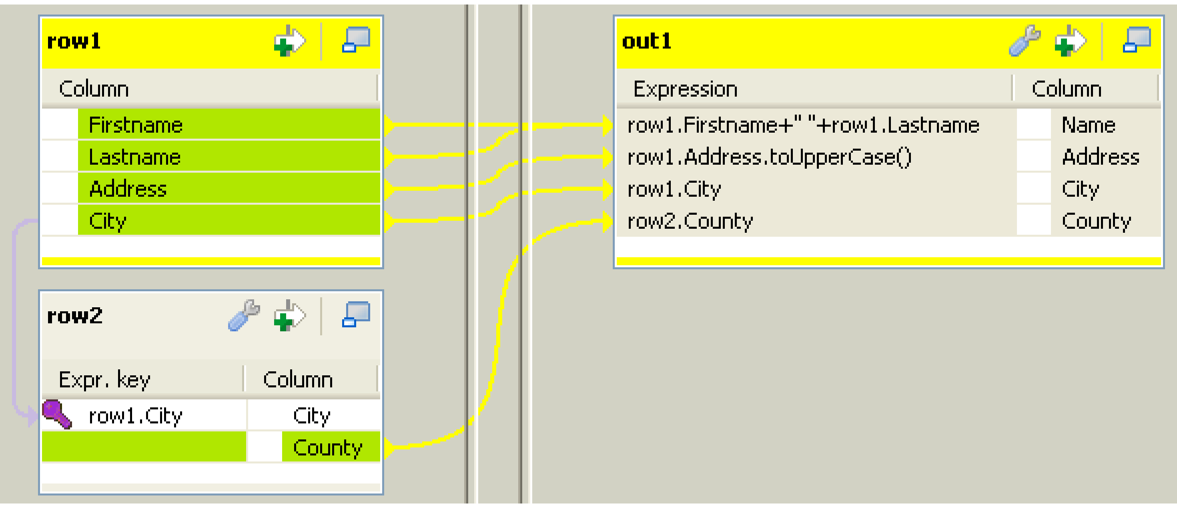

row1

| Column |
|---|
| Firstname |
| Lastname |
| Address |
| City |

row2

| Expr. key | Column |
|---|---|
| row1.City | City |
| | County |

Now, we are able to use the *County* column from the reference table in the output table (out1).



row1

| Column |
|---|
| Firstname |
| Lastname |
| Address |
| City |

row2

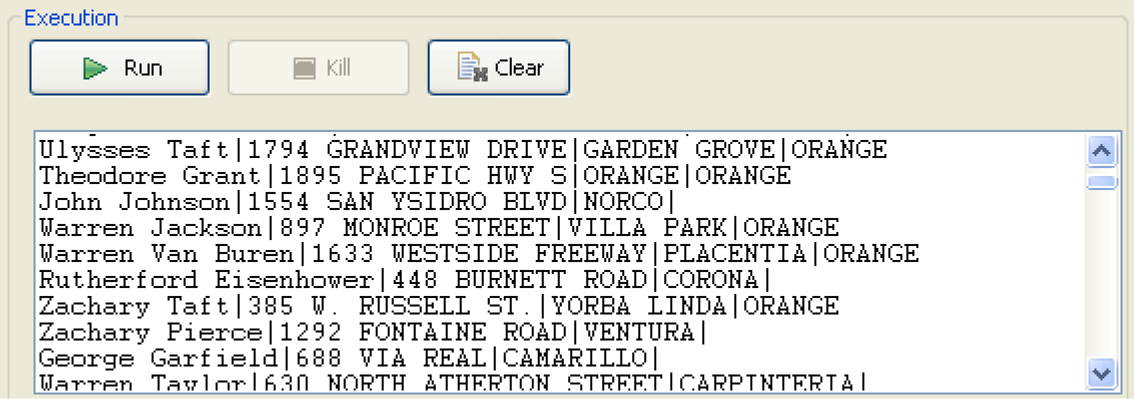
| Expr. key | Column |
|---|---|
| row1.City | City |
|  | County |

out1

| Expression | Column |
|---|---|
| row1.Firstname+" "+row1.Lastname | Name |
| row1.Address.toUpperCase() | Address |
| row1.City | City |
| row2.County | County |

Eventually, click the **OK** button to validate your changes, and run the new Job.

The following output should display on the console.



```
Ulysses Taft|1794 GRANDVIEW DRIVE|GARDEN GROVE|ORANGE
Theodore Grant|1895 PACIFIC HWY S|ORANGE|ORANGE
John Johnson|1554 SAN YSIDRO BLVD|NORCO|
Warren Jackson|897 MONROE STREET|VILLA PARK|ORANGE
Warren Van Buren|1633 WESTSIDE FREEWAY|PLACENTIA|ORANGE
Rutherford Eisenhower|448 BURNETT ROAD|CORONA|
Zachary Taft|385 W. RUSSELL ST.|YORBA LINDA|ORANGE
Zachary Pierce|1292 FONTAINE ROAD|VENTURA|
George Garfield|688 VIA REAL|CAMARILLO|
Warren Taylor|630 NORTH ATHERTON STREET|CARPINTERIA|
```

As you can notice, the last column is only filled out for *Los Angeles* and *Orange* counties' cities. For all other lines, this column is empty. The reason for this is that by default, the **tMap** implements a left outer join mode. If you want to filter your data to only display lines for which a match is found by the **tMap**, then open again the **tMap**, click the **tMap settings** button and select the **Inner Join** in the **Join Model** list on the reference table (*row2*).

## B.1.2.4. Step 4: Output to a MySQL table

Our Job works perfectly! To finalize it, let's direct the output flow to a MySQL table.

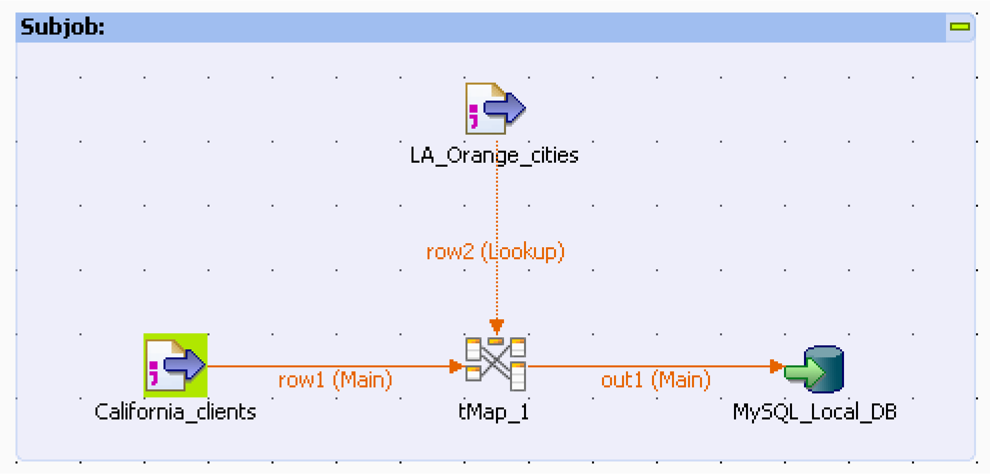
To do so, let's first create the Metadata describing the connection to the MySQL database. Double-click Metadata/MySQL/DemoMySQL in the referential (on the condition that you imported the Demo project properly). This opens the Metadata wizard.

On Step2 of the wizard, type in the relevant connection parameters. Check the validity of this connection by clicking on the **Check** button. Eventually, validate your changes, by clicking on **Finish**.

Drop this metadata to the right of the design workspace, while maintaining the **Ctrl** key down, in order to create automatically a **tMysqlOutput** component.

Remove the **tLogRow** component from your Job.

Reconnect the out1 output flow from the **tMap** to the new component **tMysqlOutput** (Right-click/Row/out1):



On the **Basic Settings** tab of this component:

1. Type in *LA_Orange_Clients* in the **Table** field, in order to name your target table which will get created on the fly.
2. Select the **Drop table if exists and create** option or on the **Action on table** field.
3. Click **Edit Schema** and click the **Reset DB type** button (DB button on the tool bar) in order to fill out automatically the DB type if need be.

Run again the Job. The target table should be automatically created and filled with data in less a second!

In this scenario, we did use only four different components out of hundreds of components available in the **Palette** and grouped according to different categories (databases, Web service, FTP and so on)!

And more components, this time created by the community, are also available on the community site (talendforge.org).

For more information regarding the components, see *Talend Open Studio for Big Data Components Reference Guide*.

# B.2. Using the output stream feature

## B.2.1. Introducing the scenario

The following use case aims to show how to use the output stream feature in a number of components in order to greatly improve the output performance.

In this scenario, a pre-defined csv file containing customer information is loaded in a database table. Then the loaded data is selected using a **tMap**, and output to a local file and to the console using the output stream feature.

### B.2.1.1. Input data

The input file, the data of which will be loaded into the database table, contains customer information of various aspects.

The file structure usually called **Schema** in *Talend Open Studio for Big Data* includes the following columns:

- *id* (Type: Integer)
- *CustomerName* (Type: String)
- *CustomerAge* (Type: Integer)
- *CustomerAddress* (Type: String)
- *CustomerCity* (Type: String)
- *RegisterTime* (Type: Date)

### B.2.1.2. Output data

The **tMap** component is used to select *id*, *CustomerName* and *CustomerAge* columns from the input data. Then the selected data is output using the output stream feature.

Thus the expected output data should have the following structure:

- *id* (Type: Integer)
- *CustomerName* (Type: String)
- *CustomerAge* (Type: Integer)

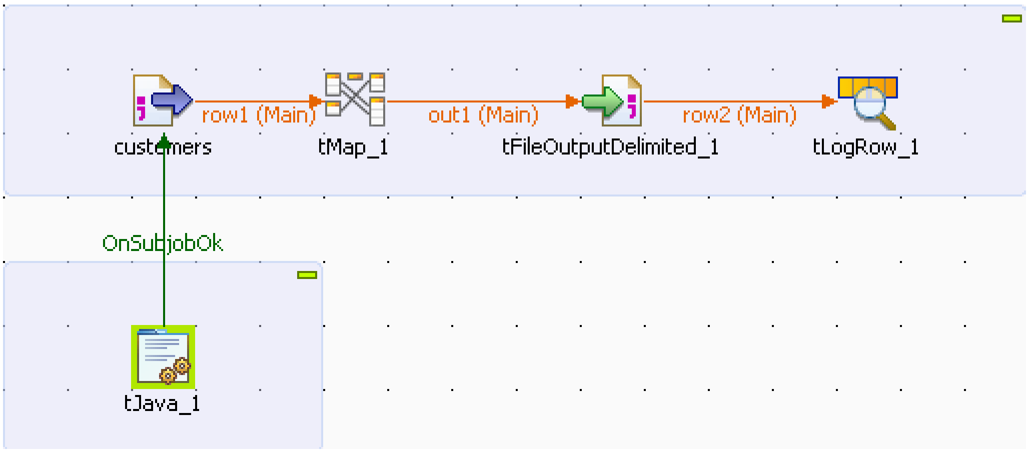
All the three columns above come from the respective columns in the input data.

## B.2.2. Translating the scenario into a Job

In order to implement this scenario, break down the Job into four steps:

1. Create the Job, define the schema for the input data, and read the input file according to the defined schema.
2. Set the command to enable the output stream feature.
3. Map the data using the **tMap** component.
4. Output the selected data stream.

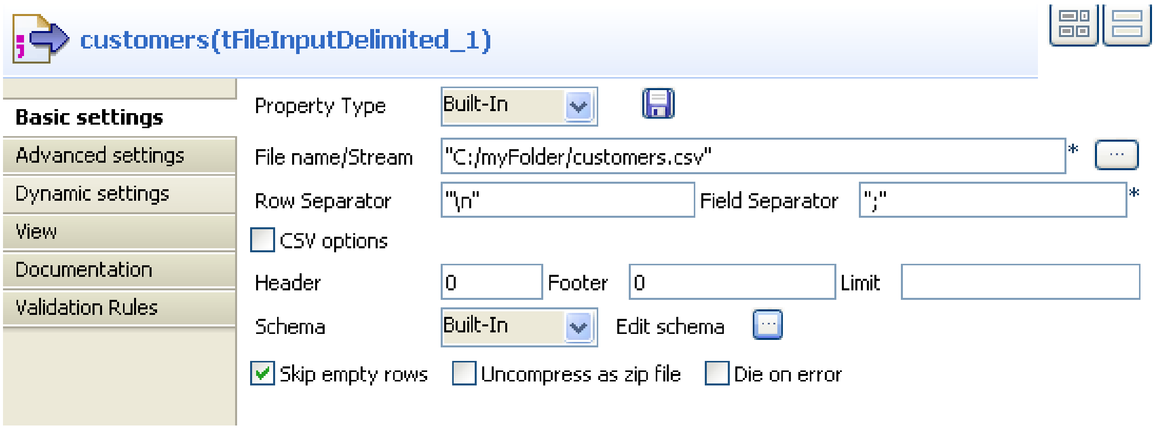
A complete Job looks as what it displays in the following image. For the detailed instruction for designing the Job, read the following sections.



### B.2.2.1. Step 1: Reading input data from a local file

We will use the **tFileInputDelimited** component to read the file *customers.csv* for the input data. This component can be found in the **File/Input** group of the **Palette**.

1. Drop a **tFileInputDelimited** component onto the design workspace, and double-click the to open the **Basic settings** view to set its properties.

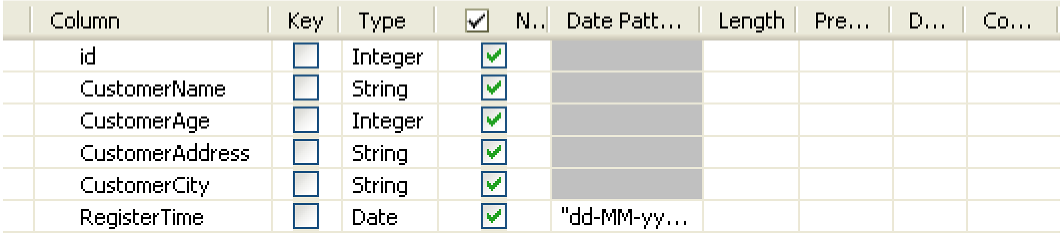


Property Type: Built-In
File name/Stream: "C:/myFolder/customers.csv"
Row Separator: "\n" Field Separator: ";"
□ CSV options
Header: 0 Footer: 0 Limit:
Schema: Built-In Edit schema
☑ Skip empty rows □ Uncompress as zip file □ Die on error

2. Click the three-dot button next to the **File name/Stream** field to browse to the path of the input data file. You can also type in the path of the input data file manually.

3. Click **Edit schema** to open a dialog box to configure the file structure of the input file.

4. Click the plus button to add six columns and set the **Type** and columns names to what we listed in the following:



| Column | Key | Type | ☑ N.. | Date Patt... | Length | Pre... | D... | Co... |
|---|---|---|---|---|---|---|---|---|
| id | □ | Integer | ☑ | | | | | |
| CustomerName | □ | String | ☑ | | | | | |
| CustomerAge | □ | Integer | ☑ | | | | | |
| CustomerAddress | □ | String | ☑ | | | | | |
| CustomerCity | □ | String | ☑ | | | | | |
| RegisterTime | □ | Date | ☑ | "dd-MM-yy... | | | | |

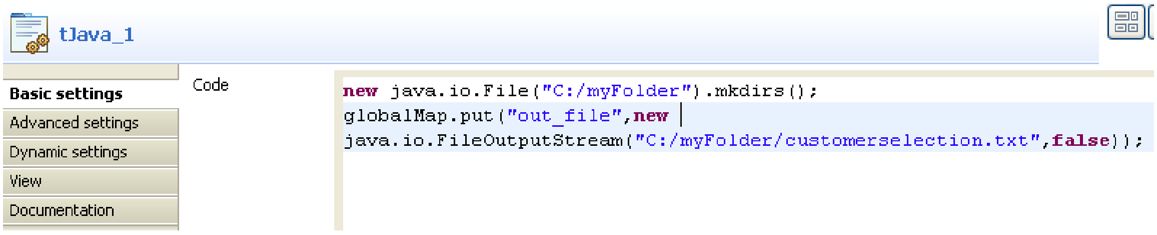
5. Click **OK** to close the dialog box.

### B.2.2.2. Step2: Setting the command to enable the output stream feature

Now we will make use of **tJava** to set the command for creating an output file and a directory that contains the output file.

To do so:

1. Drop a **tJava** component onto the design workspace, and double-click it to open the **Basic settings** view to set its properties.



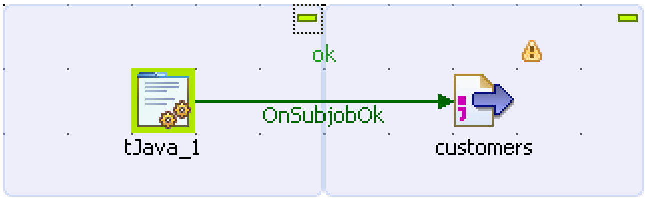
```
new java.io.File("C:/myFolder").mkdirs();
globalMap.put("out_file",new
java.io.FileOutputStream("C:/myFolder/customerselection.txt",false));
```

2. Fill in the **Code** area with the following command:

```
new java.io.File("C:/myFolder").mkdirs();
globalMap.put("out_file",new java.io.FileOutputStream("C:/myFolder/
customerselection.txt",false));
```

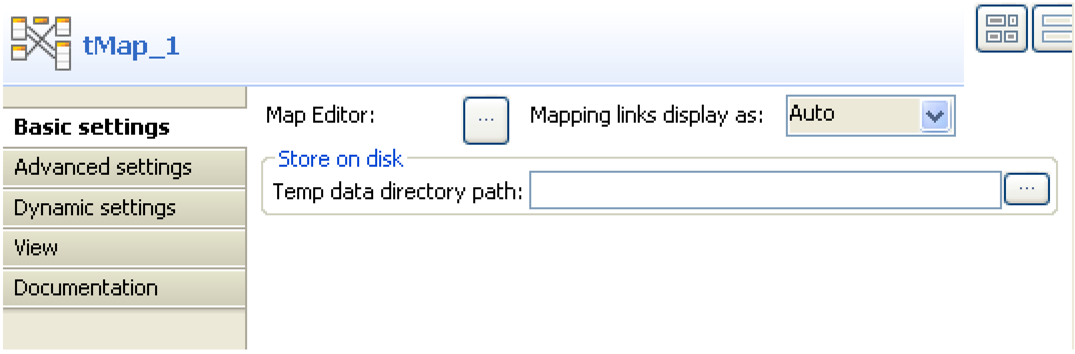
The command we typed in this step will create a new directory *C:/myFolder* for saving the output file *customerselection.txt* which is defined followingly. You can customize the command in accordance with actual practice.

3. Connect **tJava** to **tFileInputDelimited** using a **Trigger** > **On Subjob Ok** connection. This will trigger **tJava** when subjob that starts with **tFileInputDelimited** succeeds in running.



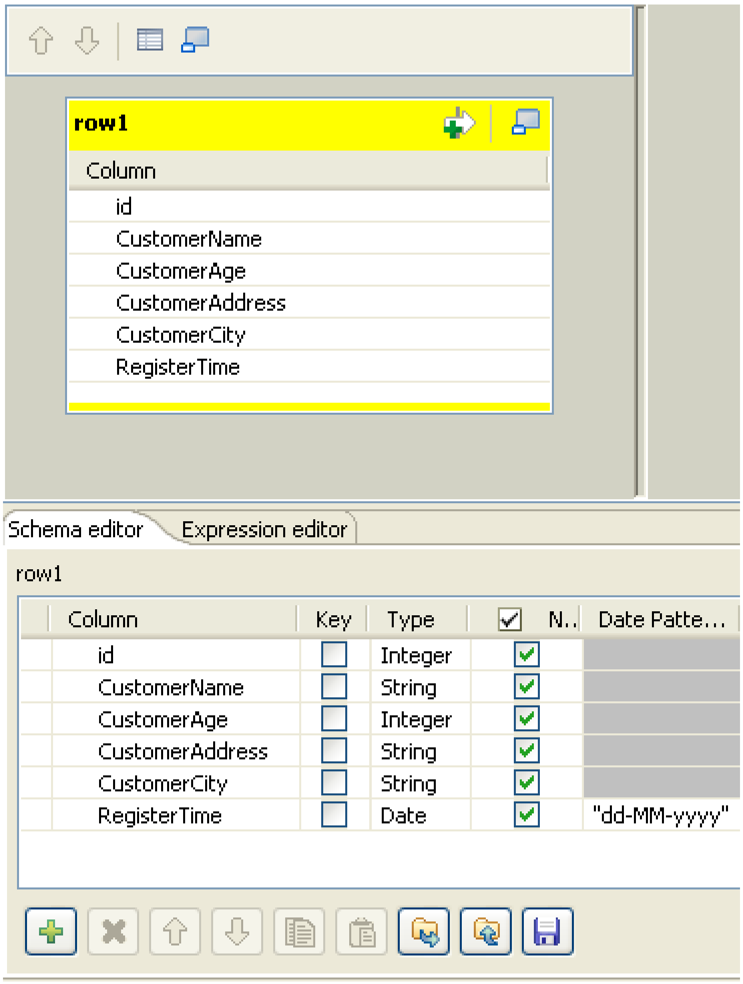
## B.2.2.3. Step3: Mapping the data using the tMap component

1. Drop a **tMap** component onto the design workspace, and double-click it to open the **Basic settings** view to set its properties.



2. Click the three-dot button next to **Map Editor** to open a dialog box to set the mapping.

3. Click the plus button on the left to add six columns for the schema of the incoming data, these columns should be the same as the following:



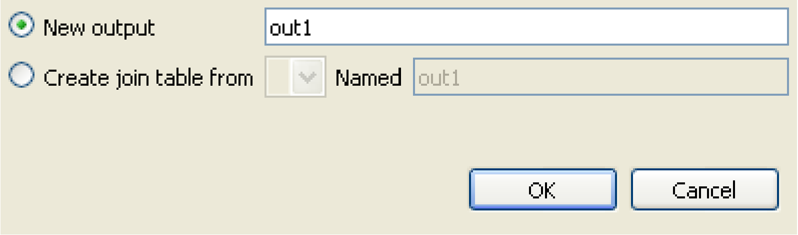
**row1**

| Column |
|---|
| id |
| CustomerName |
| CustomerAge |
| CustomerAddress |
| CustomerCity |
| RegisterTime |

row1

| Column | Key | Type | ☑ N.. | Date Patte... |
|---|---|---|---|---|
| id | □ | Integer | ☑ | |
| CustomerName | □ | String | ☑ | |
| CustomerAge | □ | Integer | ☑ | |
| CustomerAddress | □ | String | ☑ | |
| CustomerCity | □ | String | ☑ | |
| RegisterTime | □ | Date | ☑ | "dd-MM-yyyy" |

4. Click the plus button on the right to add a schema of the outgoing data flow.

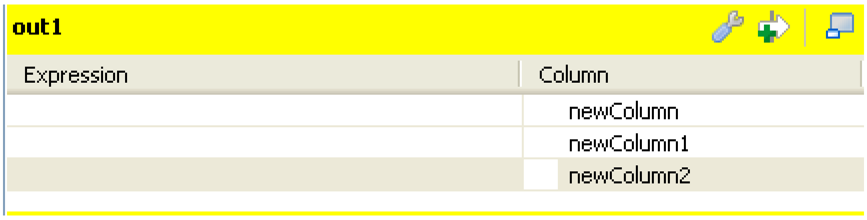


● New output: out1

○ Create join table from Named: out1

5. Select **New output** and Click **OK** to save the output schema. For the time being, the output schema is still empty.

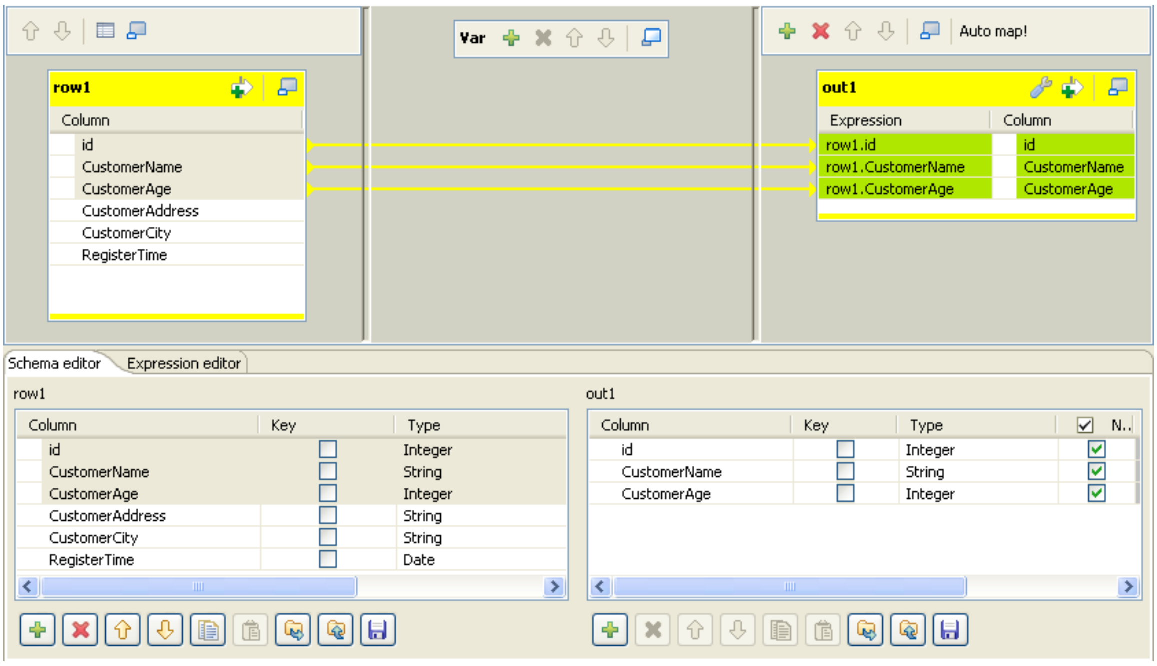
6. Click the plus button beneath the *out1* table to add three columns for the output data.



**out1**

| Expression | Column |
|---|---|
| | newColumn |
| | newColumn1 |
| | newColumn2 |

7. Drop the **id**, **CustomerName** and **CustomerAge** columns onto their respective line on the right.



row1

| Column |
|---|
| id |
| CustomerName |
| CustomerAge |
| CustomerAddress |
| CustomerCity |
| RegisterTime |

out1

| Expression | Column |
|---|---|
| row1.id | id |
| row1.CustomerName | CustomerName |
| row1.CustomerAge | CustomerAge |

row1

| Column | Key | Type |
|---|---|---|
| id | □ | Integer |
| CustomerName | □ | String |
| CustomerAge | □ | Integer |
| CustomerAddress | □ | String |
| CustomerCity | □ | String |
| RegisterTime | □ | Date |

out1

| Column | Key | Type | ☑ N.. |
|---|---|---|---|
| id | □ | Integer | ☑ |
| CustomerName | □ | String | ☑ |
| CustomerAge | □ | Integer | ☑ |

8. Click **OK** to save the settings.

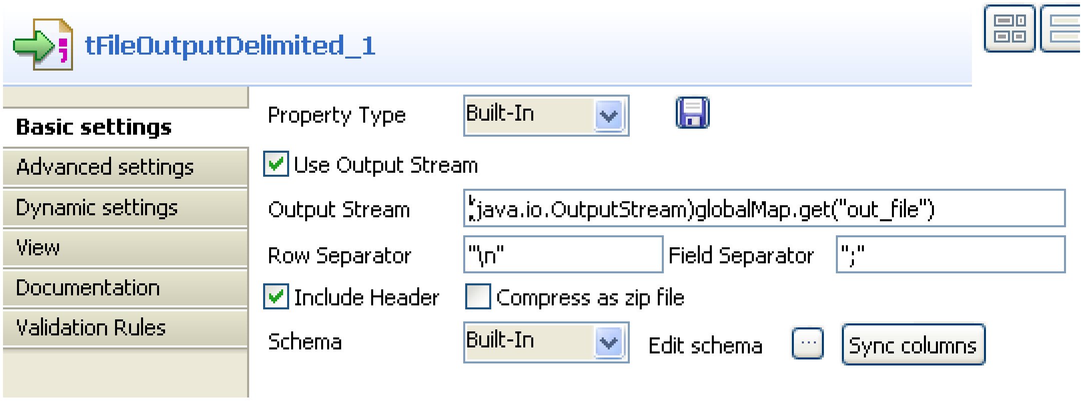
## B.2.2.4. Step4: Outputing the selected data stream

1. Drop a **tFileOutputDelimited** component onto the design workspace, and double-click it to open the **Basic settings** view to set its component properties.

2. Select the **Use Output Stream** check box to enable the **Output Stream** field and fill the **Output Stream** field with the following command:

```
(java.io.OutputStream)globalMap.get("out_file")
```

You can customize the command in the **Output Stream** field by pressing **CTRL**+**SPACE** to select built-in command from the list or type in the command into the field manually in accordance with actual practice. In this scenario, the command we use in the **Output Stream** field will call the `java.io.OutputStream` class to output the filtered data stream to a local file which is defined in the **Code** area of **tJava** in this scenario.



Property Type: Built-In

☑ Use Output Stream

Output Stream: (java.io.OutputStream)globalMap.get("out_file")

Row Separator: "\n" Field Separator: ";"

☑ Include Header □ Compress as zip file

Schema: Built-In

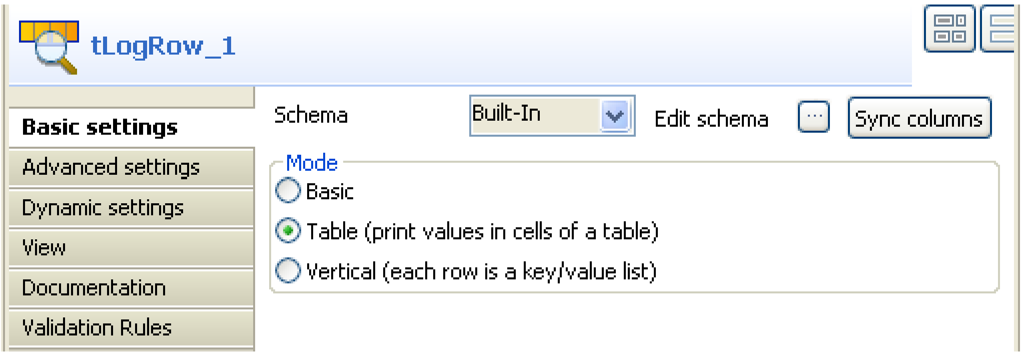
3. Connect **tFileInputDelimited** to **tMap** using a **Row** > **Main** connection and connect **tMap** to **tFileOutputDelimited** using a **Row** > **out1** connection which is defined in the **Map Editor** of **tMap**.

4. Click **Sync columns** to retrieve the schema defined in the preceding component.

To output the selected data to the console:

1. Drop a **tLogRow** component onto the design workspace, and double-click it to open its **Basic settings** view.

2. Select the **Table** radio button in the **Mode** area.



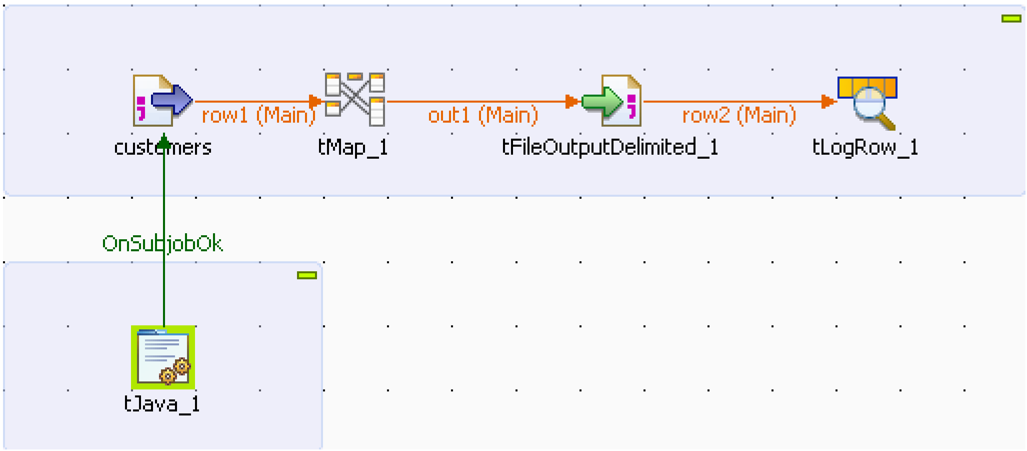
Schema Built-In

Mode
○ Basic
● Table (print values in cells of a table)
○ Vertical (each row is a key/value list)

3. Connect **tFileOutputDelimited** to **tLogRow** using a **Row** > **Main** connection.

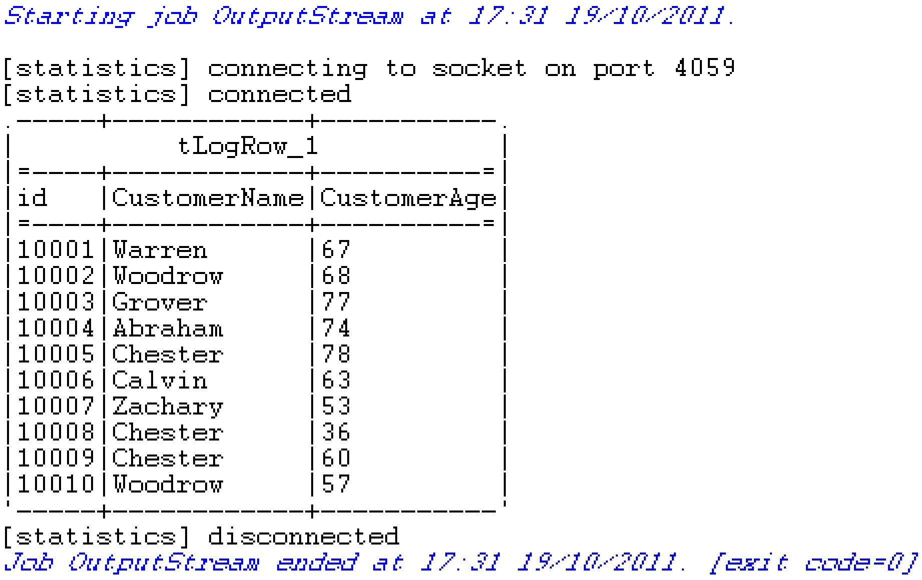
4. Click **Sync columns** to retrieve the schema defined in the preceding component.

This Job is now ready to be executed.



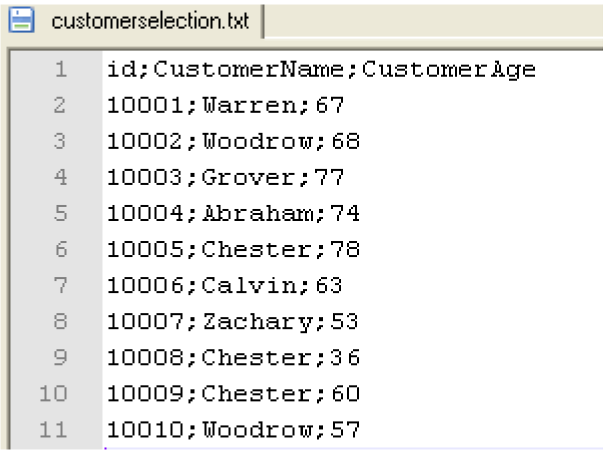
5. Press **CTRL+S** to save your Job and press **F6** to execute it.

The content of the selected data is displayed on the console.



```
Starting job OutputStream at 17:31 19/10/2011.

[statistics] connecting to socket on port 4059
[statistics] connected
.-----+------------+-----------.
|          tLogRow_1           |
|=----+------------+----------=|
|id   |CustomerName|CustomerAge|
|=----+------------+----------=|
|10001|Warren      |67         |
|10002|Woodrow     |68         |
|10003|Grover      |77         |
|10004|Abraham     |74         |
|10005|Chester     |78         |
|10006|Calvin      |63         |
|10007|Zachary     |53         |
|10008|Chester     |36         |
|10009|Chester     |60         |
|10010|Woodrow     |57         |
'-----+------------+-----------'
[statistics] disconnected
Job OutputStream ended at 17:31 19/10/2011. [exit code=0]
```

The selected data is also output to the specified local file *customerselection.txt*.



```
id;CustomerName;CustomerAge
10001;Warren;67
10002;Woodrow;68
10003;Grover;77
10004;Abraham;74
10005;Chester;78
10006;Calvin;63
10007;Zachary;53
10008;Chester;36
10009;Chester;60
10010;Woodrow;57
```

For an example of Job using this feature, see Scenario: *Utilizing Output Stream in saving filtered data to a local file* of **tFileOutputDelimited** in *Talend Open Studio for Big Data Components Reference Guide*.

For the principle of the **Use Output Stream** feature, see section *How to use the Use Output Stream feature*.

# B.3. Finding out who visit your website most often

To drive a focused marketing campaign based on habits or profiles of your customers or users, you need to be able to fetch data based on their habits or behavior on your website to be able to create user profiles and send them the right advertisements, for example.

This section provides an example of finding out users having visited a website most often by sorting out their IP addresses from a huge number of records in the access-log file for an Apache HTTP server to enable further analysis on user behavior on the website.

## B.3.1. Discovering the scenario

In this example, certain **Talend** Big Data components are used to leverage the advantage of the Hadoop open source platform for handling big data. In this scenario we use four Jobs:

- The first Job sets up an HCatalog database, table and partition in HDFS
- The second Job uploads the access-log file to be analyzed to the HDFS file system.
- The third Job parses the uploaded access-log file, including filtering any records with an "404" error, counting the number of successful service calls to the website, sorting the result data and saving it in the HDFS file system.
- The last Job reads the result data from HDFS and displays the IP addresses with successful service calls and their number of visits to the website on the standard system console.

# B.3.2. Translating the scenario into Jobs

## B.3.2.1. Setting up an HCatalog database

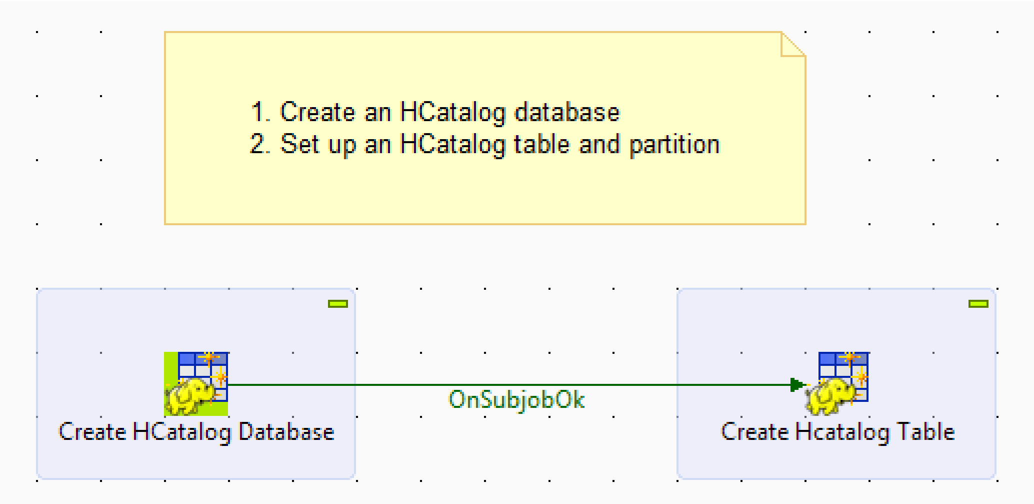
In the first step, we will set up an HCatalog environment to manage the access-log file to be analyzed.

### Choose the right components and build the first Job

1. Drop two **tHCatalogOperation** components from the **Palette** onto the design workspace.

2. Connect the two **tHCatalogOperation** components using a **Trigger** > **On Subjob Ok** connection. These two subjobs will create an HCatalog database and set up an HCatalog table and partition in the created HCatalog table, respectively.

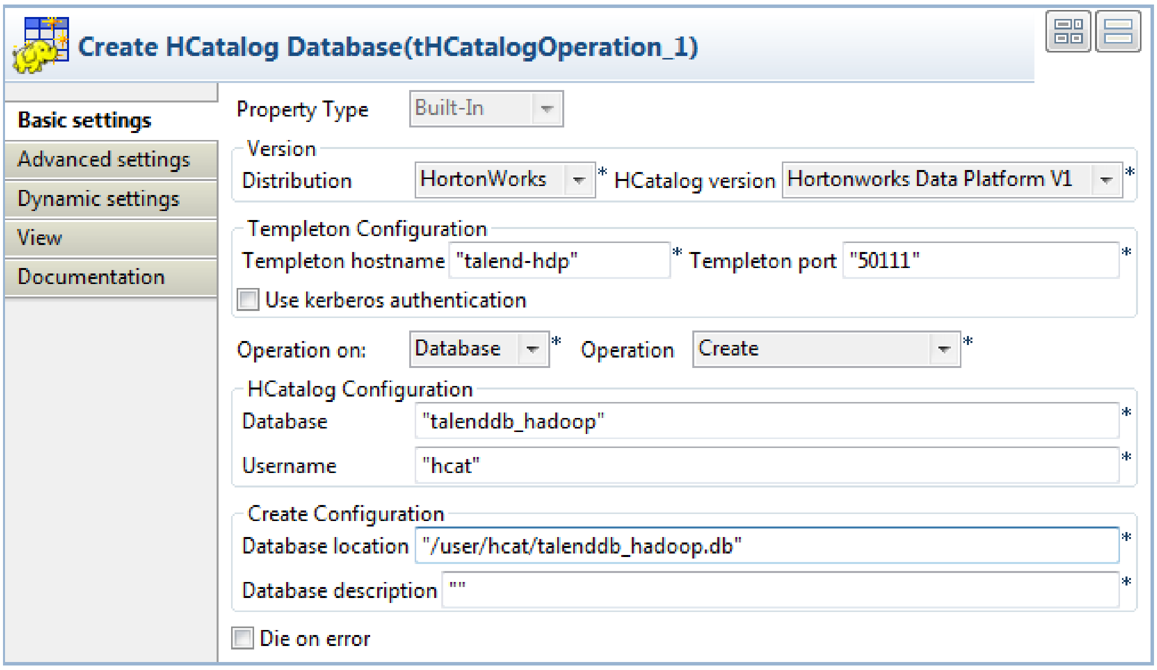
3. Label these components to better identify their functionality.



1. Create an HCatalog database
2. Set up an HCatalog table and partition

### Create an HCatalog database

1. Double-click the first **tHCatalogOperation** component to open its **Basic settings** view on the **Component** tab.



Create HCatalog Database(tHCatalogOperation_1)

Property Type: Built-In

Version

Distribution: HortonWorks * HCatalog version: Hortonworks Data Platform V1 *

Templeton Configuration

Templeton hostname: "talend-hdp" * Templeton port: "50111" *

□ Use kerberos authentication

Operation on: Database * Operation: Create *

HCatalog Configuration

Database: "talenddb_hadoop" *

Username: "hcat" *

Create Configuration

Database location: "/user/hcat/talenddb_hadoop.db" *

Database description: "" *

□ Die on error

2. From the relevant lists, select the Hadoop distribution and version. In this example we use the default settings: *HortonWorks* distribution with the version of *HortonWorks Data Platform V1*.

3. Provide either the host name or the IP address of your Templeton server and the Templeton port in the relevant fields, both in double quotation marks.

4. From the **Operation on** list, select **Database**; from the **Operation** list, select **Create**.

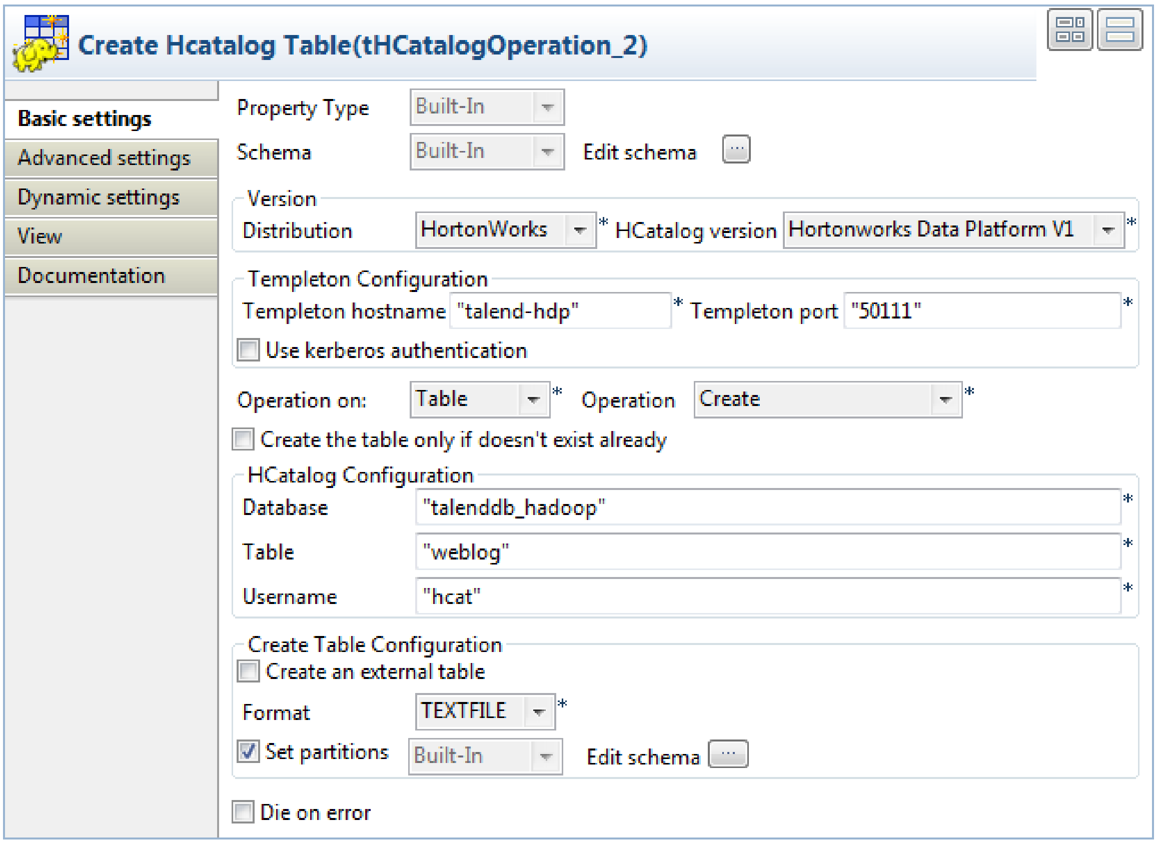
5. In the **Database** field, enter a name for the database you're going to create, *talenddb_hadoop* in this example.

6. In the **Username** field, enter the user name for database authentication.

7. In the **Database location** field, enter the location for the database file is to be created in HDFS.

## Set up an HCatalog table and partition

1. Double-click the second **tHCatalogOperation** component to open its **Basic settings** view on the **Component** tab.



Create Hcatalog Table(tHCatalogOperation_2)

Property Type: Built-In

Schema: Built-In — Edit schema […]

Version
Distribution: HortonWorks * HCatalog version: Hortonworks Data Platform V1 *

Templeton Configuration
Templeton hostname: "talend-hdp" * Templeton port: "50111" *
□ Use kerberos authentication

Operation on: Table * Operation: Create *
□ Create the table only if doesn't exist already

HCatalog Configuration
Database: "talenddb_hadoop" *
Table: "weblog" *
Username: "hcat" *

Create Table Configuration
□ Create an external table
Format: TEXTFILE *
☑ Set partitions Built-In Edit schema […]

□ Die on error

2. Specify the same HCatalog distribution and version, Templeton host name/IP address, and Templeton port as in the first **tHCatalogOperation** component.

3. From the **Operation on** list, select **Table**; from the **Operation** list, select **Create**.

   When you work on a table, HCatalog requires you to define a schema for it. This schema, however, will not take part in our subsequent operations, so simply click the **[...]** button and add a column to the schema, and give it any name that's different from the column name of the partition schema you're going to set later on.

4. Specify the same database and user name as in the first **tHCatalogOperation** component.

5. In the **Table** field, enter a name for the table to be created, *weblog* in this example.

6. Select the **Set partitions** check box and click the **[...]** button next to **Edit schema** to set a partition and partition schema. Note that the partition schema must not contain any column name defined in the table schema. In this example, the partition schema column is named *ipaddresses*.

## B.3.2.2. Uploading the access-log file to the Hadoop system

In the second step, we will build and configure the second Job to upload the access-log file to the Hadoop system, and then check the uploaded file.

### Choose the right components and build the second Job

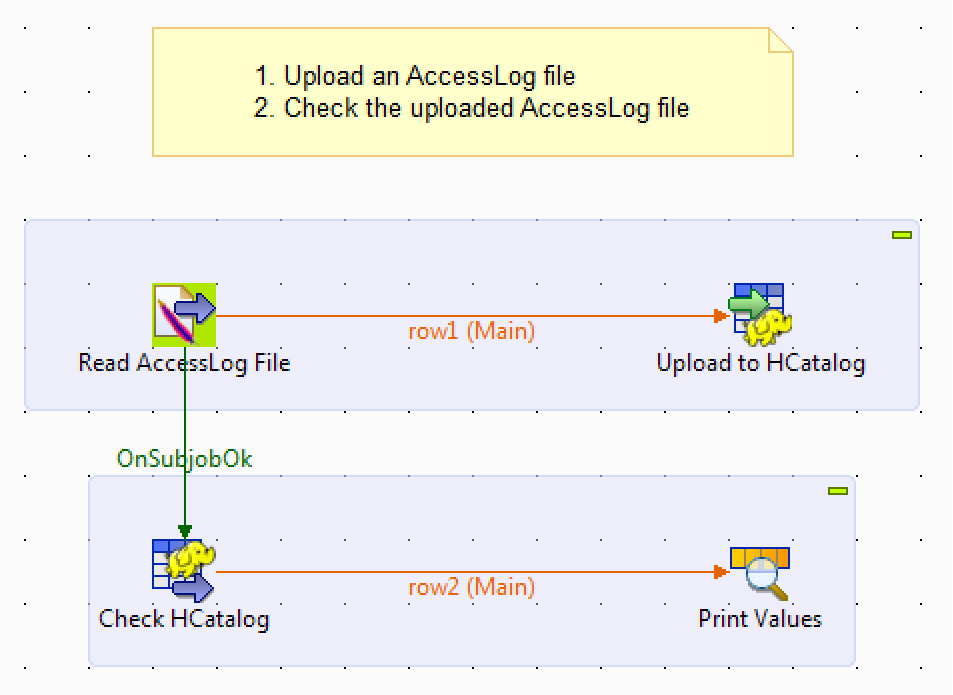
1. From the **Palette**, drop a **tApacheLogInput**, a **tHCatalogOutput**, a **tHCatalogInput**, and a **tLogRow** component onto the design workspace.

2. Connect the **tApacheLogInput** component to the **tHCatalogOutput** component using a **Row** > **Main** connection. This subjob will read the access-log file to be analyzed and upload it to the established HCatalog database.

3. Connect the **tHCatalogInput** component to the **tLogRow** component using a **Row** > **Main** connection. This subjob will verify the file upload by reading the access-log file from the HCatalog system and displaying the content on the console.

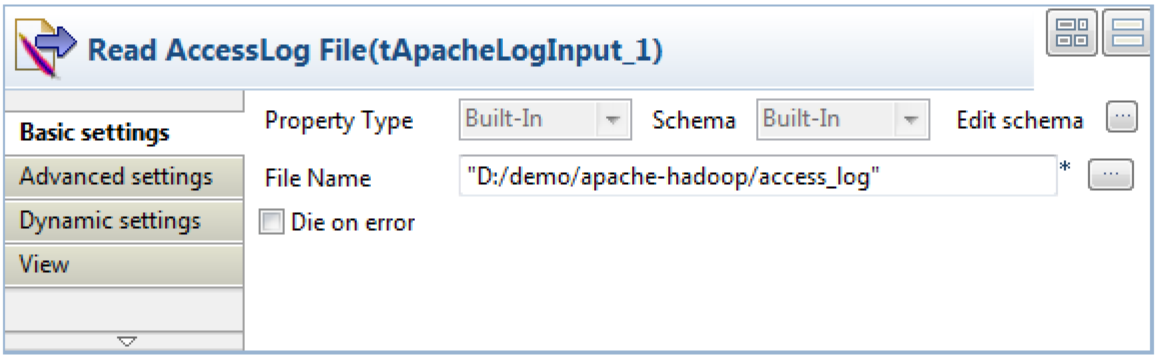
4. Connect the **tApacheLogInput** component to the **tHCatalogInput** component using a **Trigger** > **On Subjob Ok** connection.

5. Label these components to better identify their functionality.



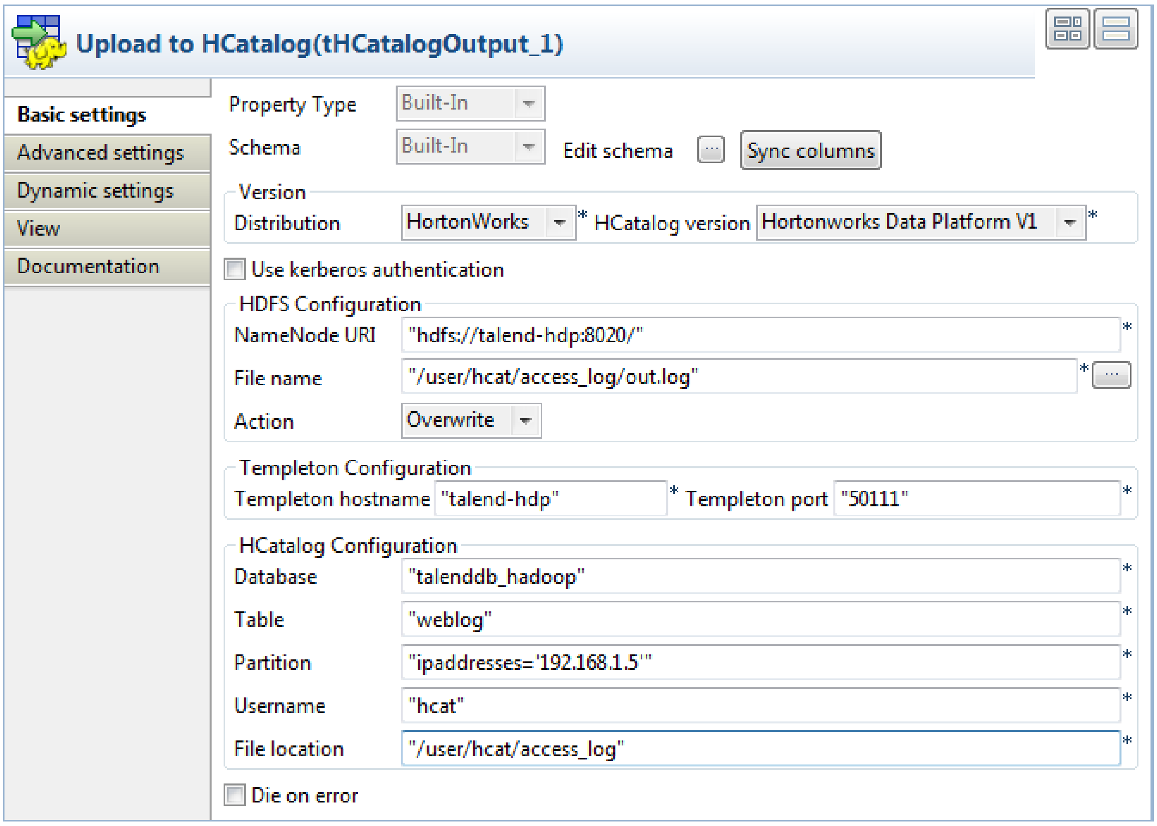
## Upload the access-log file to HDFS

1. Double-click the **tApacheLogInput** component to open its **Basic settings** view, and specify the path to the access-log file to be uploaded in the **File Name** field.


Read AccessLog File(tApacheLogInput_1)

Property Type: Built-In
Schema: Built-In
Edit schema
File Name: "D:/demo/apache-hadoop/access_log"
☐ Die on error

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



Upload to HCatalog(tHCatalogOutput_1)

Property Type: Built-In

Schema: Built-In

Version

Distribution: HortonWorks *

HCatalog version: Hortonworks Data Platform V1 *

□ Use kerberos authentication

HDFS Configuration

NameNode URI: "hdfs://talend-hdp:8020/" *

File name: "/user/hcat/access_log/out.log" *

Action: Overwrite

Templeton Configuration

Templeton hostname: "talend-hdp" *

Templeton port: "50111" *

HCatalog Configuration

Database: "talenddb_hadoop" *

Table: "weblog" *

Partition: "ipaddresses='192.168.1.5'" *

Username: "hcat" *

File location: "/user/hcat/access_log" *

□ Die on error

3. Click the **[...]** button to verify that the schema has been properly propagated from the preceding component. If needed, click **Sync columns** to retrieve the schema.

4. For the following items, use the same settings as defined in the first Job:

   - Hadoop distribution and version

   - Templeton host name/IP address and Templeton port number

   - HCatalog database, table, and user name

5. In the **NameNode URI** field, enter the URI of the HDFS NameNode.

6. In the **File name** field, specify the path and name of the output file in the HDFS file system.

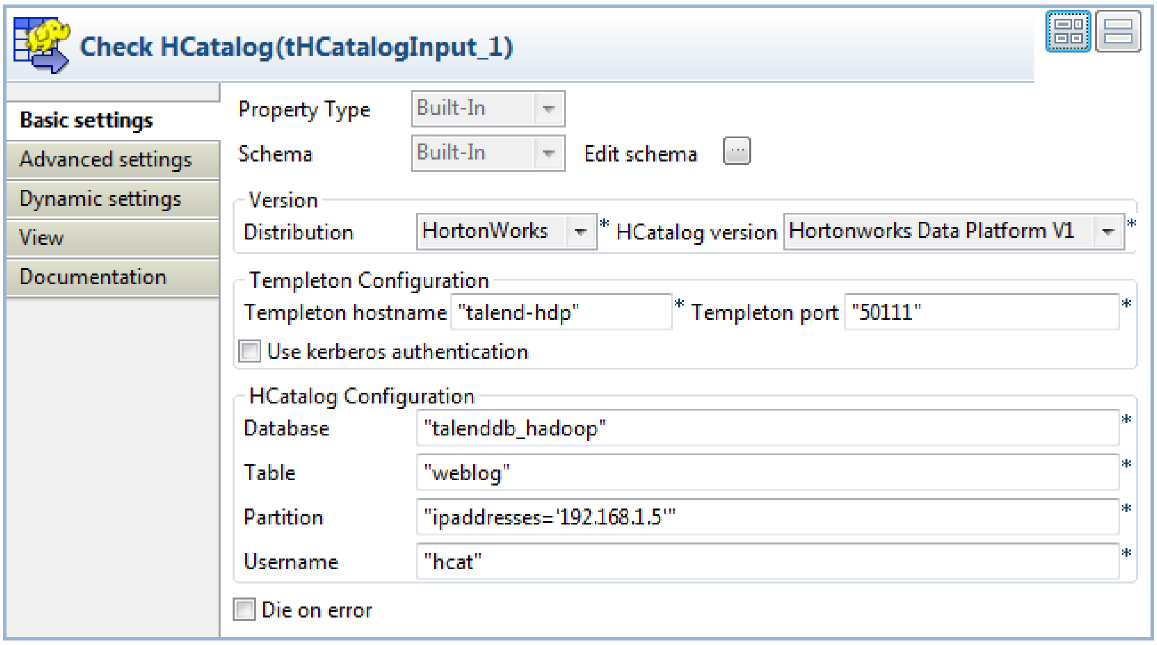
7. From the **Action** list, select **Create** to create the file or **Overwrite** if the file already exists.

8. In the **Partition** field, enter the partition name-value pair, *ipaddresses='192.168.1.15'* in this example.

9. In the **File location** field, enter the path where the data will be save, */user/hcat/access_log* in this example.

## Check the uploaded access-log file

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



Check HCatalog(tHCatalogInput_1)

Property Type: Built-In

Schema: Built-In    Edit schema ...

Version

Distribution: HortonWorks *    HCatalog version: Hortonworks Data Platform V1 *

Templeton Configuration

Templeton hostname: "talend-hdp" *    Templeton port: "50111" *

□ Use kerberos authentication

HCatalog Configuration

Database: "talenddb_hadoop" *

Table: "weblog" *

Partition: "ipaddresses='192.168.1.5'" *

Username: "hcat" *

□ Die on error

2. Click the **[...]** button to open the **[Schema]** dialog box and define the input schema. In this example, we can simply copy the schema from that of the **tApacheLogInput** or **tHCatalogOutput** component.

3. For all the other items, use the same settings as defined in the **tHCatalogOutput** component.

4. In the **Basic settings** view of the **tLogRow** component, select the **Vertical** mode to display the each row in a key-value manner when the Job is executed.

## B.3.2.3. Analyzing the access-log file on the Hadoop platform

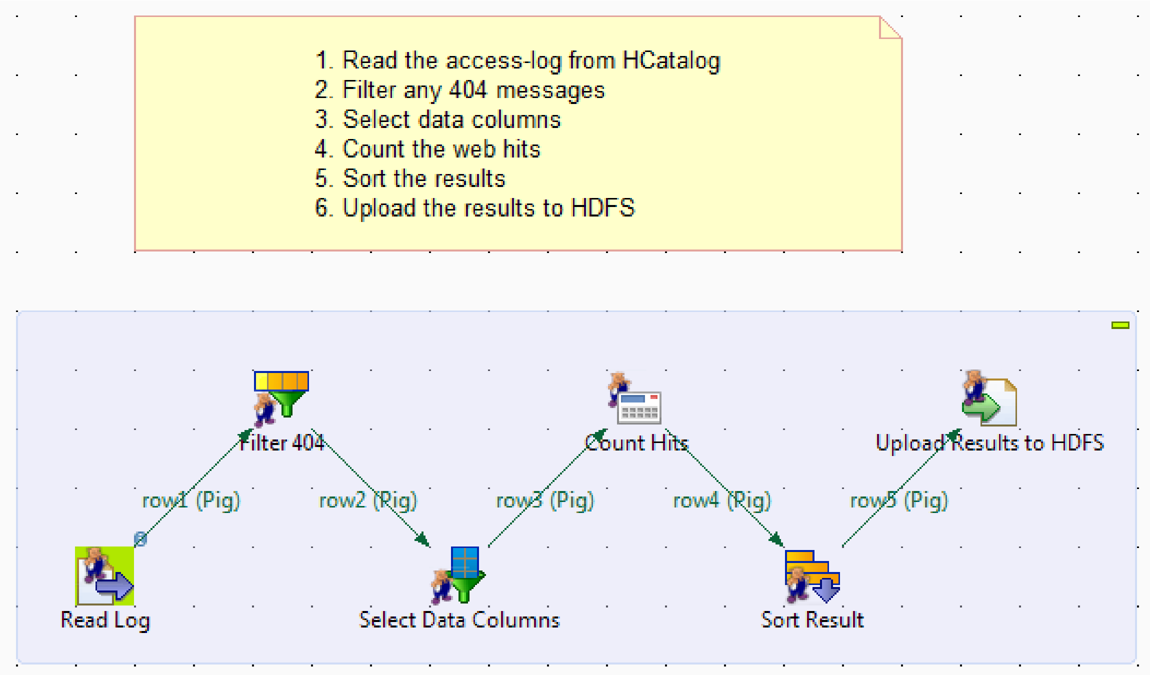
In this step, we will build and configure the third Job, which uses several Pig components to analyze the uploaded access-log file in a Pig chain to get the IP addresses with successful service calls and their number of visits to the website.

### Choose the right components and build the third Job

1. Drop the following components from the **Palette** to the design workspace:

   - a **tPigLoad**, to load the data to be analyzed,
   - a **tPigFilterRow**, to remove records with the '404' error from the input flow,
   - a **tPigFilterColumns**, to select the columns you want to include in the result data,
   - a **tPigAggregate**, to count the number of visits to the website from each host,
   - a **tPigSort**, to sort the result data, and
   - a **tPigStoreResult**, to save the result to HDFS.

2. Connect these components using **Row** > **Pig Combine** connections to form a Pig chain, and label them to better identify their functionality.

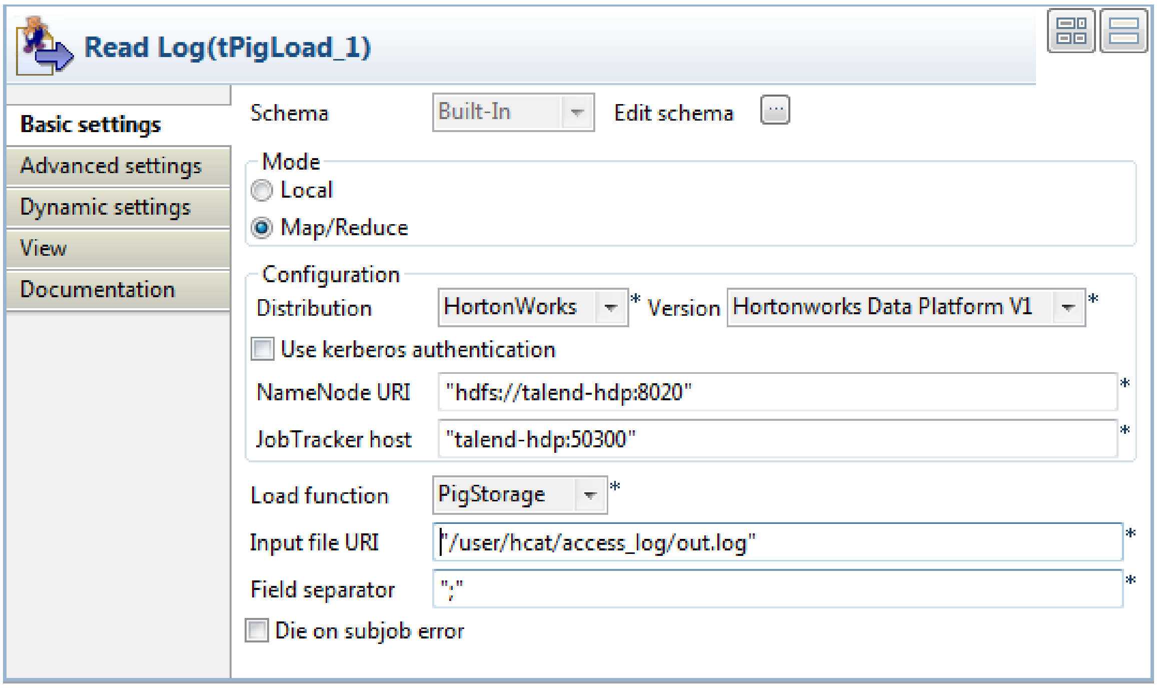


1. Read the access-log from HCatalog
2. Filter any 404 messages
3. Select data columns
4. Count the web hits
5. Sort the results
6. Upload the results to HDFS

## Configure the Pig chain

1. Double-click the **tPigLoad** component to open its **Basic settings** view, and configure the following items to load the file to be analyzed in the Pig chain:

   - Schema: copy from the previous Job, and permit the schema to be propagated to the next component.
   - Pig mode: select **Map/Reduce**.
   - Hadoop distribution and version: the same as in the previous Job, *HortonWorks* and *HortonWorks Data Platform V1* in this example.
   - NameNode URI: the same as in the previous Job, *hdfs://talend-hdp:8020* in this example.
   - JobTracker host: *talend-hdp:50300* in this example.
   - Load function: select **PigStorage**.
   - Input file URI: the output file name defined in the previous Job, */user/hcat/access_log/out.log* in this example.



Read Log(tPigLoad_1)

Schema: Built-In — Edit schema [...]

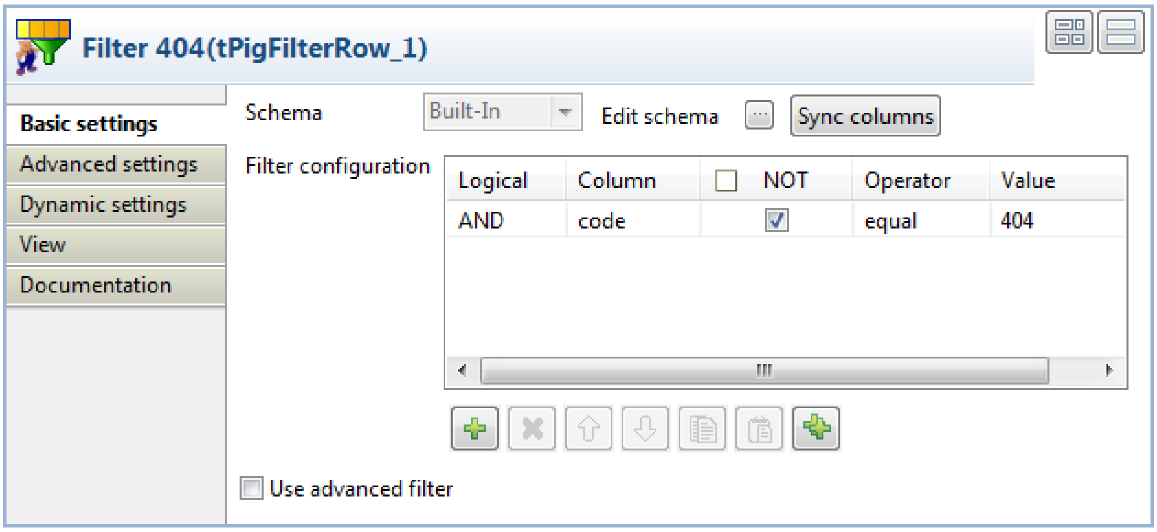
Mode
○ Local
● Map/Reduce

Configuration
Distribution: HortonWorks * Version: Hortonworks Data Platform V1 *
□ Use kerberos authentication
NameNode URI: "hdfs://talend-hdp:8020" *
JobTracker host: "talend-hdp:50300" *

Load function: PigStorage *
Input file URI: "/user/hcat/access_log/out.log" *
Field separator: ";" *
□ Die on subjob error

2. In the **Basic settings** view of the **tPigFilterRow** component, click the **[+]** button to add a line in the **Filter configuration** table, and set filter parameters to remove records that contain the code of 404 and pass the rest records on to the output flow:

- In the **Logical** field, select **AND**.
- In the **Column** field, select the *code* column of the schema.
- Select the **NOT** check box.
- In the **Operator** field, select **equal**.
- In the **Value** field, enter *404*.



Filter 404(tPigFilterRow_1)

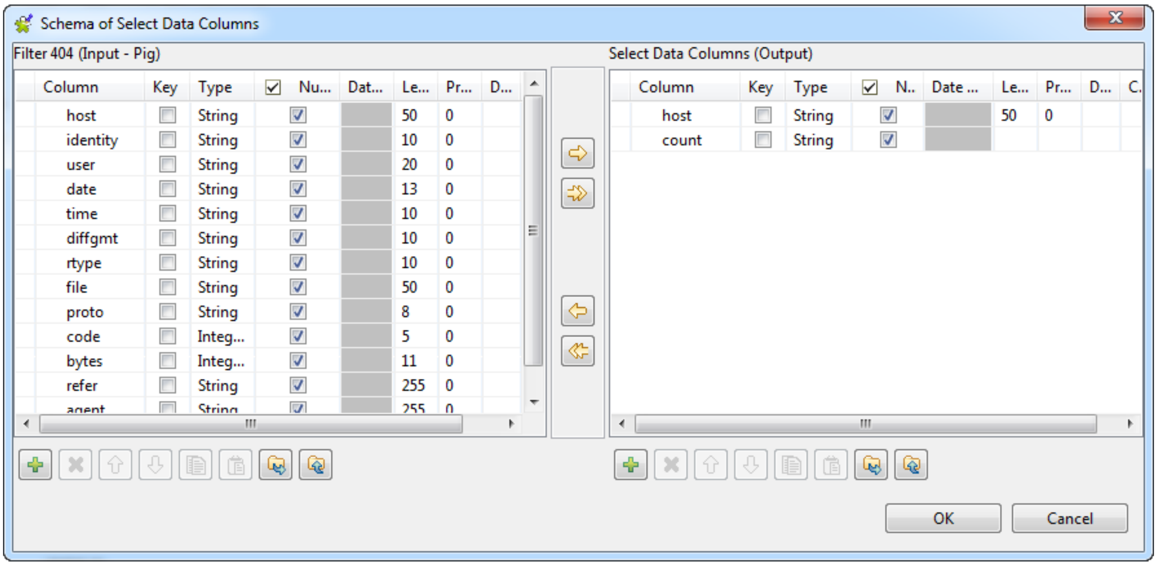
Schema: Built-In — Edit schema [...]

Filter configuration

| Logical | Column | □ NOT | Operator | Value |
|---|---|---|---|---|
| AND | code | ☑ | equal | 404 |

□ Use advanced filter

3. In the **Basic settings** view of the **tPigFilterColumns** component, click the **[...]** button to open the **[Schema]** dialog box. In the **Output** panel, set two columns, *host* and *count*, which will store the information of IP addresses and their number of visits to the website, respectively.



Schema of Select Data Columns

Filter 404 (Input - Pig)

| Column | Key | Type | ☑ Nu... | Dat... | Le... | Pr... | D... |
|---|---|---|---|---|---|---|---|
| host | □ | String | ☑ | | 50 | 0 | |
| identity | □ | String | ☑ | | 10 | 0 | |
| user | □ | String | ☑ | | 20 | 0 | |
| date | □ | String | ☑ | | 13 | 0 | |
| time | □ | String | ☑ | | 10 | 0 | |
| diffgmt | □ | String | ☑ | | 10 | 0 | |
| rtype | □ | String | ☑ | | 10 | 0 | |
| file | □ | String | ☑ | | 50 | 0 | |
| proto | □ | String | ☑ | | 8 | 0 | |
| code | □ | Integ... | ☑ | | 5 | 0 | |
| bytes | □ | Integ... | ☑ | | 11 | 0 | |
| refer | □ | String | ☑ | | 255 | 0 | |
| agent | □ | String | ☑ | | 255 | 0 | |

Select Data Columns (Output)

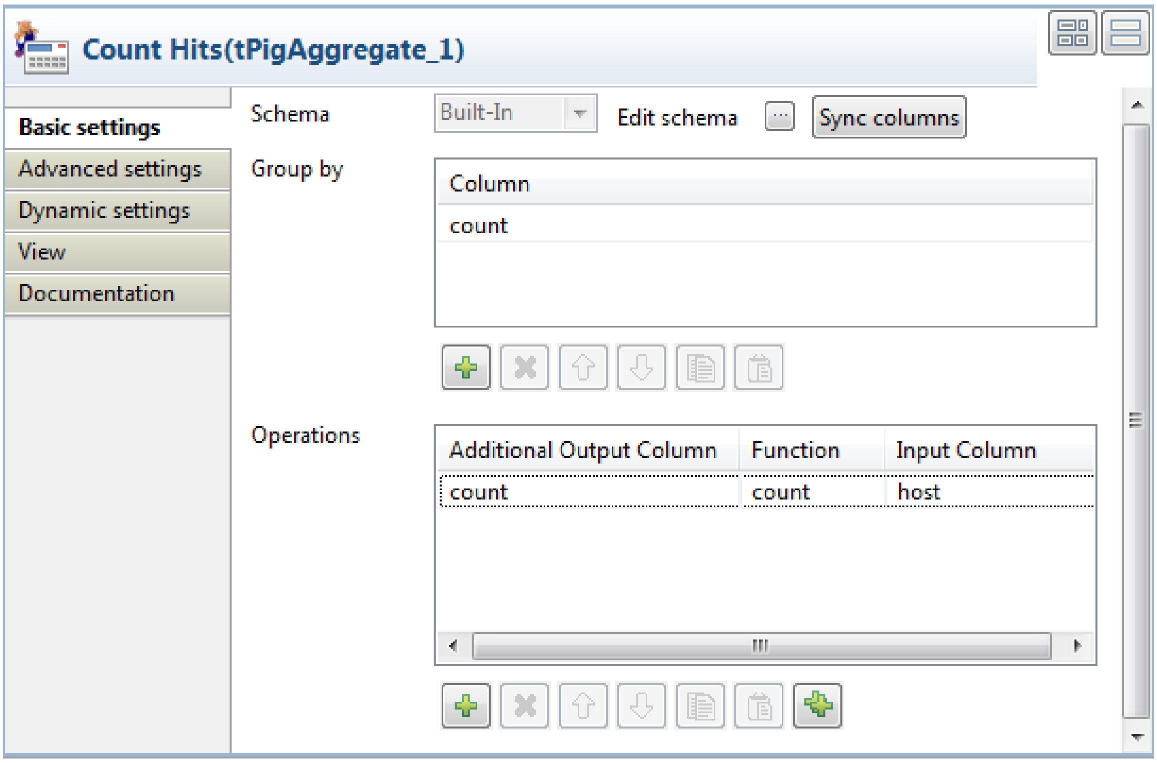
| Column | Key | Type | ☑ N.. | Date ... | Le... | Pr... | D... | C.. |
|---|---|---|---|---|---|---|---|---|
| host | □ | String | ☑ | | 50 | 0 | | |
| count | □ | String | ☑ | | | | | |

4. In the **Basic settings** view of the **tPigAggregate** component, click **Sync columns** to retrieve the schema from the preceding component, and permit the schema to be propagated to the next component.

5. Configure the following parameters to count the number of occurrences of each IP address:

   - In the **Group by** area, click the **[+]** button to add a line in the table, and select the column *count* in the **Column** field.

   - In the **Operations** area, click the **[+]** button to add a line in the table, and select the column *count* in the **Additional Output Column** field, select **count** in the **Function** field, and select the column *host* in the **Input Column** field.



Count Hits(tPigAggregate_1)

Schema: Built-In

Group by

| Column |
|---|
| count |

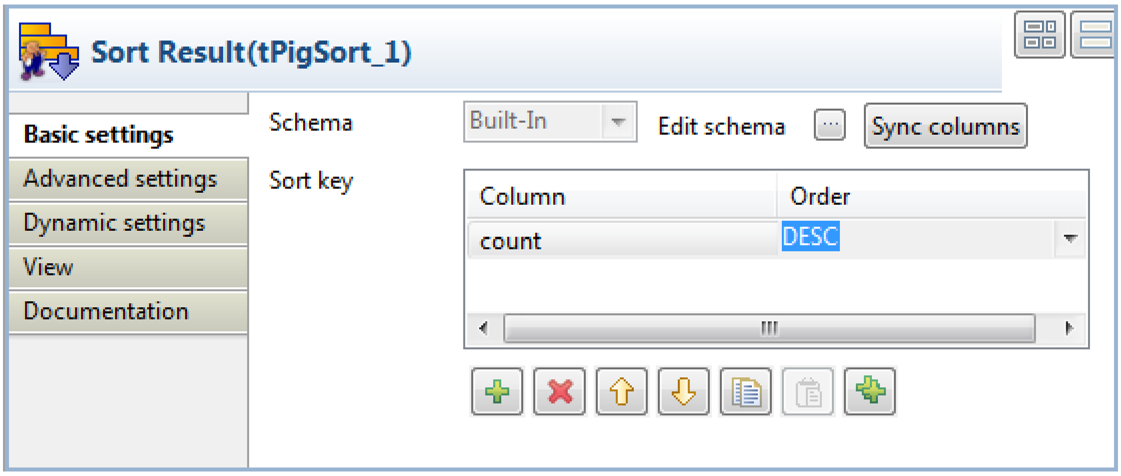
Operations

| Additional Output Column | Function | Input Column |
|---|---|---|
| count | count | host |

6. In the **Basic settings** view of the **tPigSort** component, configure the sorting parameters to sort the data to be passed on:

   - Click the **[+]** button to add a line in the **Sort key** table.

- In the **Column** field, select **count** to set the column *count* as the key.
- In the **Order** field, select **DESC** to sort data in the descendent order.


Sort Result(tPigSort_1)

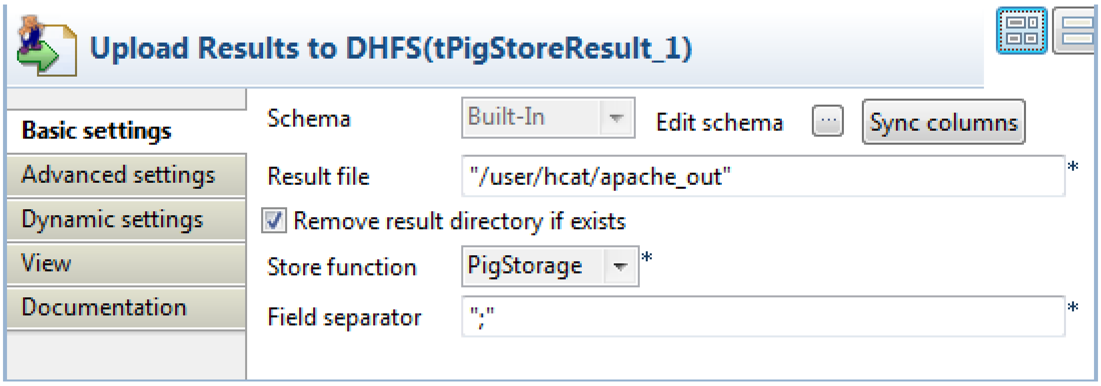
Schema: Built-In

Sort key:

| Column | Order |
|---|---|
| count | DESC |

7. In the **Basic settings** view of the **tPigStoreResult** component, configure the component properties to upload the result data to the specified location on the Hadoop system:

- Check the schema; retrieve the schema from the preceding component if needed.
- In the **Result file** field, enter the path to the result file.
- From the **Store function** list, select **PigStorage**.
- If needed, select the **Remove result directory if exists** check box.


Upload Results to DHFS(tPigStoreResult_1)

Schema: Built-In

Result file: "/user/hcat/apache_out"*

☑ Remove result directory if exists

Store function: PigStorage*

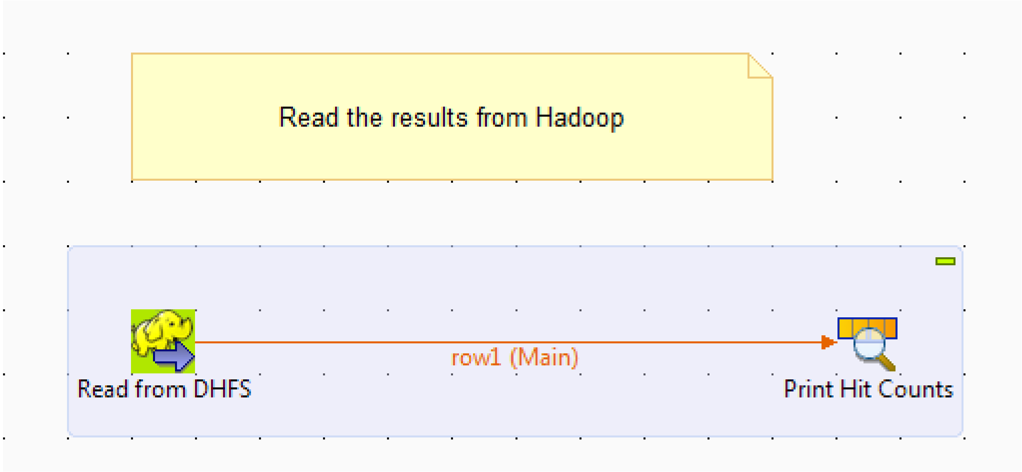
Field separator: ";"*

## B.3.2.4. Checking the analysis result

In this step, we will build the last Job, which is a two-component Job that will read the result data from Hadoop and display it on the standard system console. Then, we will execute all the Jobs one by one and check the result on the console.

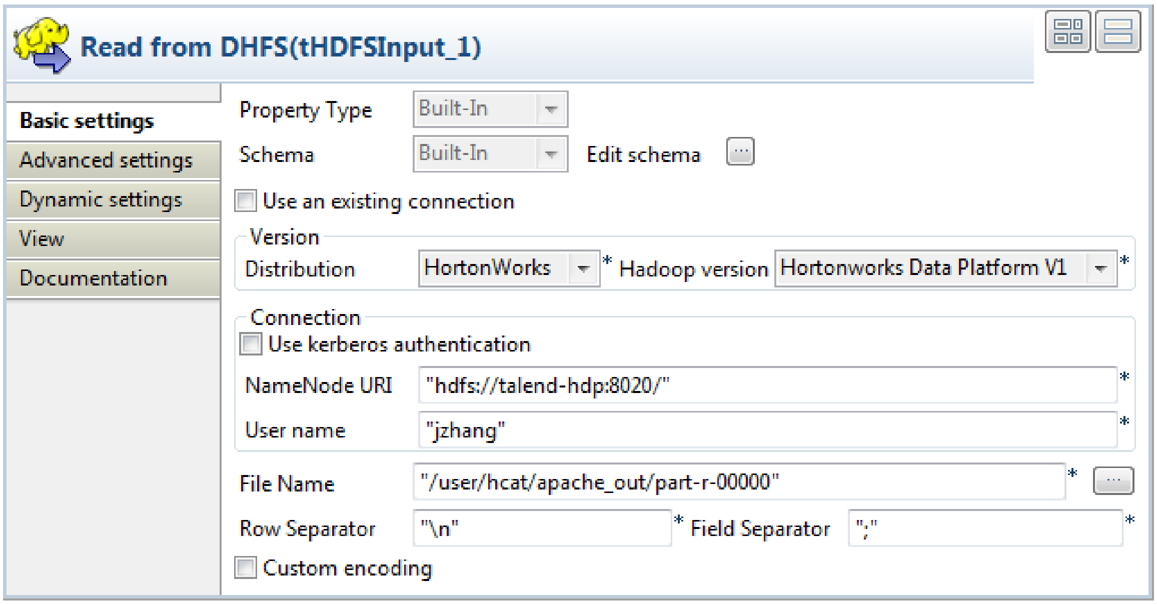
### Choose the right components to build the last Job

1. From the **Palette**, drop a **tHDFSInput** and a **tLogRow** component onto the design workspace.
2. Connect the components using a **Row** > **Main** connection, and label them to better identify their functionality.



## Configure the last Job

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



Property Type: Built-In
Schema: Built-In
□ Use an existing connection
Distribution: HortonWorks * Hadoop version: Hortonworks Data Platform V1 *
□ Use kerberos authentication
NameNode URI: "hdfs://talend-hdp:8020/" *
User name: "jzhang" *
File Name: "/user/hcat/apache_out/part-r-00000" *
Row Separator: "\n" * Field Separator: ";" *
□ Custom encoding

2. For the following items, use the same settings as in the previous Job:

   - Schema, which should contain two columns, *host* and *count*, according to the structure of the file uploaded to HDFS through the Pig chain in the previous Job.
   - Hadoop distribution and version, *HortonWorks* and *HortonWorks Data Platform V1* in this example.
   - NameNode URI, *hdfs://talend-hdp:8020/* in this example.

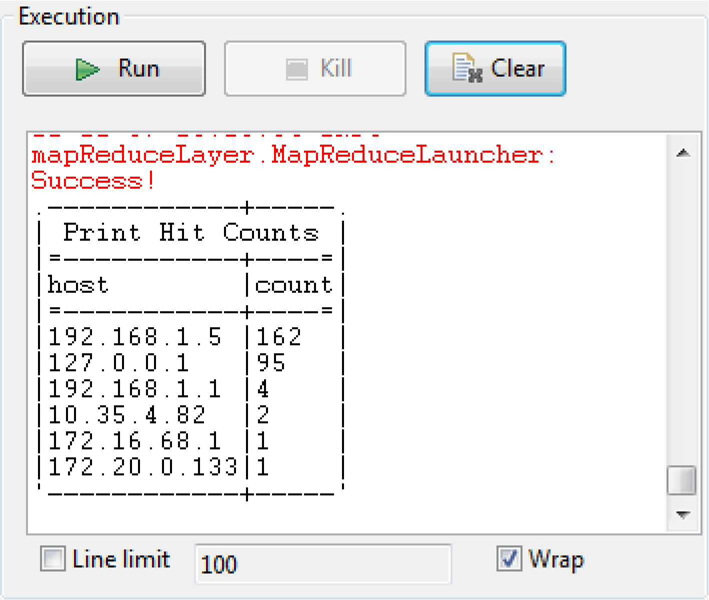
3. In the **User name** field, enter a user name permitted to access the file in HDFS.

4. In the **File Name** field, enter the path and file name in HDFS.

5. In the **Basic settings** view of the **tLogRow** component, select the **Table** option.

After the four Jobs are properly set up and configured, run them one by one.

Upon successful execution of the last Job, the system console displays IP addresses with successful service calls and their number of visits to the website.



```
mapReduceLayer.MapReduceLauncher:
Success!
.------------+-----.
| Print Hit Counts |
|=-----------+----=|
|host        |count|
|=-----------+----=|
|192.168.1.5 |162  |
|127.0.0.1   |95   |
|192.168.1.1 |4    |
|10.35.4.82  |2    |
|172.16.68.1 |1    |
|172.20.0.133|1    |
'------------+-----'
```

# Appendix C. System routines

This appendix gives you an overview of the most commonly used routines, along with use cases. In this Appendix, routines follow the order in which they display in the **Repository**. They are grouped according to their types. Each type is detailed in a different section.

For more information on how to define routines, to access to system routines or to manage system or user routines, see chapter *Managing routines*.

Before starting any data integration processes, you need to be familiar with *Talend Open Studio for Big Data* Graphical User Interface (GUI). For more information, see appendix *GUI*.

# C.1. Numeric Routines

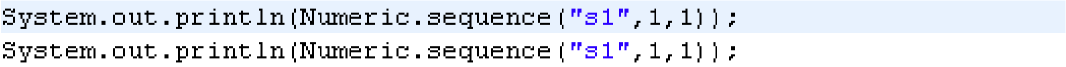
Numeric routines allow you to return whole or decimal numbers in order to use them as settings in one or more Job components. To add numeric IDs, for instance.

To access these routines, double click on the **Numeric** category, in the **system** folder. The Numeric category contains several routines, notably sequence, random and decimal (convertImpliedDecimalFormat):

| **Routine** | **Description** | **Syntax** |
|---|---|---|
| *sequence* | Returns an incremental numeric ID. | `Numeric.sequence("Parameter name", start value, increment value)` |
| *resetSequence* | Creates a sequence if it doesn’t exist and attributes a new start value. | `Numeric.resetSequence (Sequence Identifier, start value)` |
| *removeSequence* | Removes a sequence. | `Numeric.RemoveSequence (Sequence Identifier)` |
| *random* | Returns a random whole number between the maximum and minimum values. | `Numeric.random(minimum start value, maximum end value)` |
| *convertImpliedDecimalFormat* | Returns a decimal with the help of an implicit decimal model. | `Numeric. convertImpliedDecimalFormat ("Target Format", value to be converted)` |

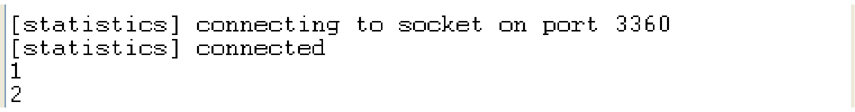
## C.1.1. How to create a Sequence

The **sequence** routine allows you to create automatically incremented IDs, using a **tJava** component:



```
System.out.println(Numeric.sequence("s1",1,1));
System.out.println(Numeric.sequence("s1",1,1));
```

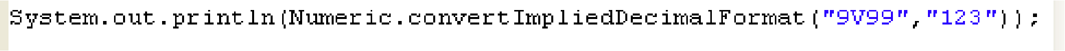
The routine generates and increments the ID automatically:



```
[statistics] connecting to socket on port 3360
[statistics] connected
1
2
```

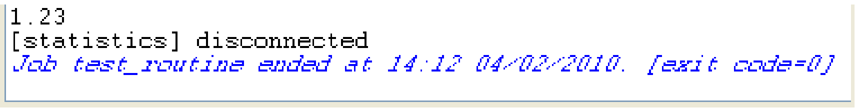
## C.1.2. How to convert an Implied Decimal

It is easy to use the **convertImpliedDecimalFormat** routine, along with a **tJava** component, for example:



```
System.out.println(Numeric.convertImpliedDecimalFormat("9V99","123"));
```

The routine automatically converts the value entered as a parameter according to the format of the implied decimal provided:



```
1.23
[statistics] disconnected
Job test_routine ended at 14:12 04/02/2010. [exit code=0]
```

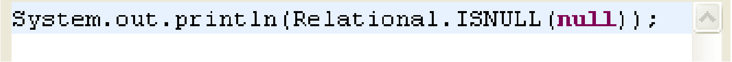
# C.2. Relational Routines

Relational routines allow you to check affirmations based on booleans.

To access these routines, double click on the **Relational** class under the **system** folder. The Relational class contains several routines, notably:

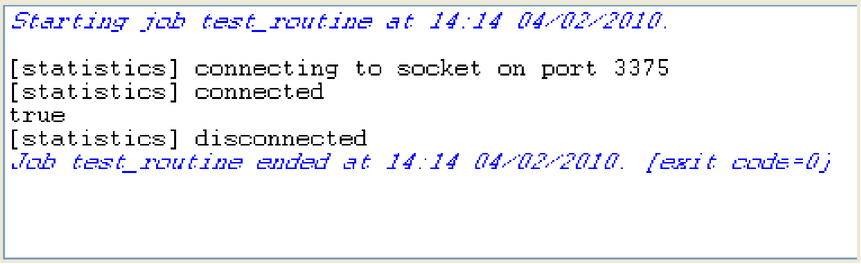
| Routine | Description | Syntax |
|---|---|---|
| *ISNULL* | Checks if the variable provided is a null value. | `Relational.ISNULL(variable to be checked)` |

To check a Relational Routine, you can use the **ISNULL** routine, along with a **tJava** component, for example:



```
System.out.println(Relational.ISNULL(null));
```

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]
```

# C.3. StringHandling Routines

The **StringHandling** routines allow you to carry out various kinds of operations and tests on alphanumeric expressions, based on Java methods.

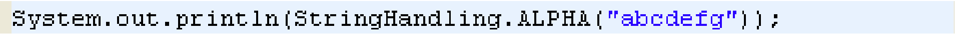
To access these routines, doubleclick on **StringHandling** under the system folder. The **StringHandling** class includes the following routines:

| 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")` |
| *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")` |
| *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("chaîne à vérifier", 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)` |

| Routine | Description | Syntax |
|---|---|---|
| *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")` |

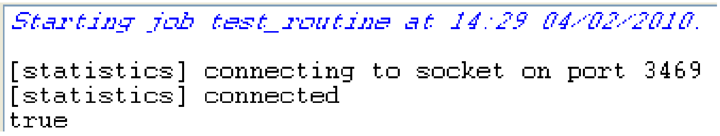
# C.3.1. How to store a string in alphabetical order

It is easy to use the ALPHA routine along with a **tJava** component, to check whether a string is in alphabetical order:



```
System.out.println(StringHandling.ALPHA("abcdefg"));
```

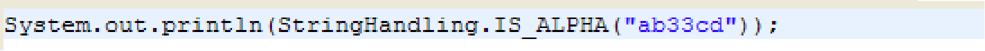
The check returns a boolean value.



```
Starting job test_routine at 14:29 04/02/2010.

[statistics] connecting to socket on port 3469
[statistics] connected
true
```

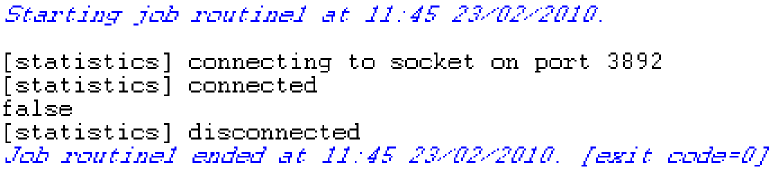
# C.3.2. How to check whether a string is alphabetical

It is easy to use the **IS_ALPHA** routine along with a **tJava** component, to check whether the string is alphabetical:



```
System.out.println(StringHandling.IS_ALPHA("ab33cd"));
```

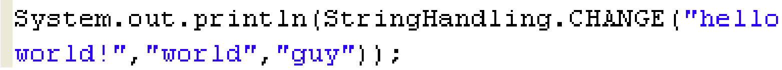
The check returns a boolean value.



```
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]
```

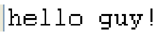
# C.3.3. How to replace an element in a string

It is easy 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"));
```

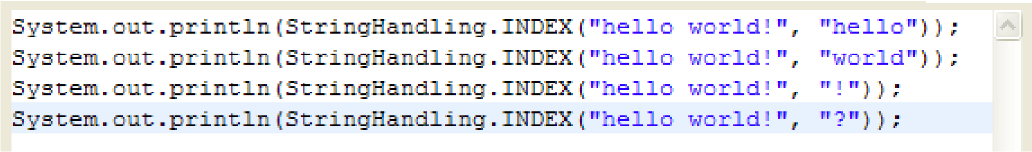
The routine replaces the old element with the new element specified.



```
hello guy!
```

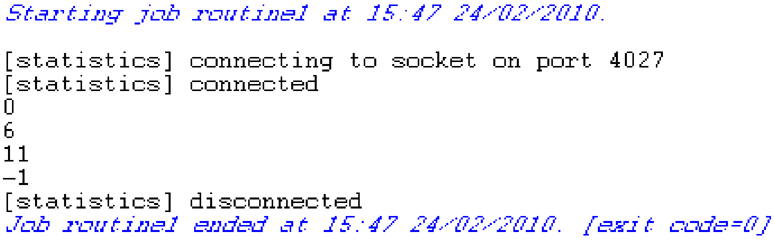
## C.3.4. How to check the position of a specific character or substring, within a string

The **INDEX** routine is easy to use along with a **tJava** component, to check whether a string contains a specified character or substring:



```
System.out.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!", "?"));
```

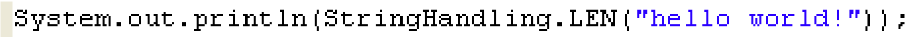
The routine returns a whole number which indicates the position of the first character specified, or indeed the first character of the substring specified. Otherwise, - 1 is returned if no occurrences are found.



```
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]
```

## C.3.5. How to calculate the length of a string

The **LEN** routine is easy to use, along with a **tJava** component, to check the length of a string:



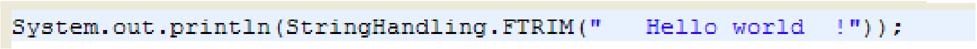
```
System.out.println(StringHandling.LEN("hello world!"));
```

The check returns a whole number which indicates the length of the chain, including spaces and blank characters.

```
12
```

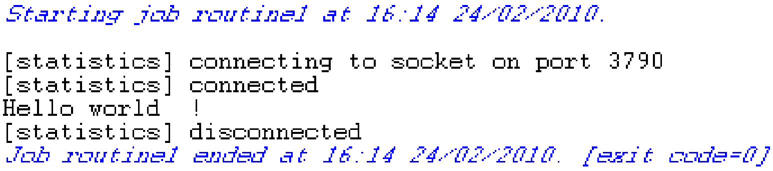
## C.3.6. How to delete blank characters

The **FTRIM** routine is easy to use, along with a **tJava** component, to delete blank characters from the start of a chain:



```
System.out.println(StringHandling.FTRIM("   Hello world   !"));
```

The routine returns the string with the blank characters removed from the beginning.



```
Starting job routine1 at 16:14 24/02/2010.

[statistics] connecting to socket on port 3790
[statistics] connected
Hello world   !
[statistics] disconnected
Job routine1 ended at 16:14 24/02/2010. [exit code=0]
```

# C.4. TalendDataGenerator Routines

The **TalendDataGenerator** routines are functions which allow you to generate sets of test data. They are based on fictitious lists of first names, second names, addresses, towns and States provided by **Talend**. These routines are generally used when developing Jobs, using a **tRowGenerator**, for example, to avoid using production or company data.

To access the routines, double click on **TalendDataGenerator** under the **system** folder:

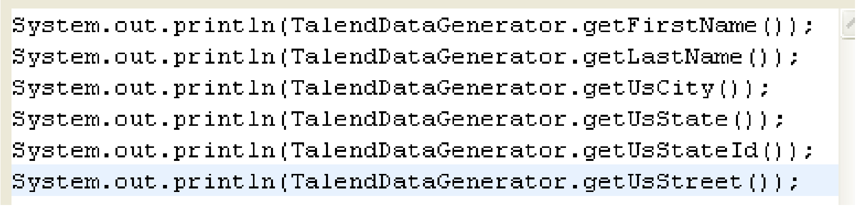
| Routine | Description | Syntax |
|---|---|---|
| *getFirstName* | returns a first name taken randomly from a fictitious list. | `TalendDataGenerator.getFirstName()` |
| *getLastName* | returns a random surname from a fictitious list. | `TalendDataGenerator.getLastName()` |
| *getUsStreet* | returns 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()` |

No entry parameter is required as **Talend** provides the list of fictitious data.

You can customize the fictitious data by modifying the **TalendGeneratorRoutines**. For further information on how to customize routines, see section *Customizing the system routines*.

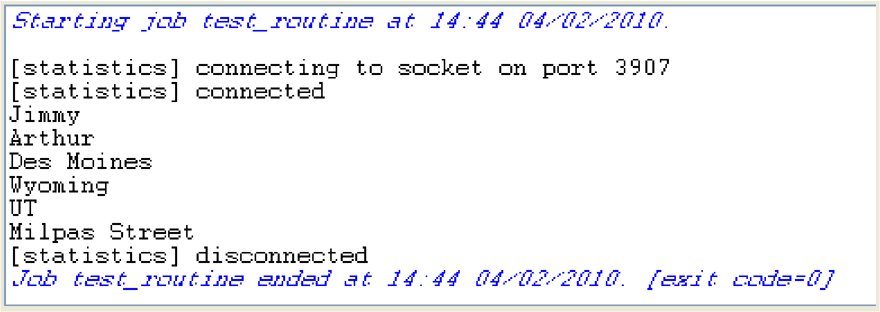
## C.4.1. How to generate fictitious data

It is easy to use the different functions to generate data randomly. Using a **tJava** component, you can, for example, create a list of fictitious client data using functions such as **getFirstName**, **getLastName**, **getUSCity**:



```
System.out.println(TalendDataGenerator.getFirstName());
System.out.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]
```

# C.5. TalendDate Routines

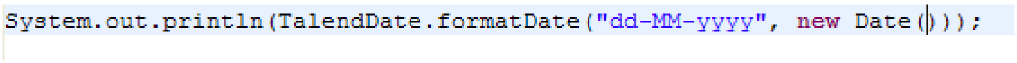
The TalendDate routines allow you to carry out different kinds of operations and checks concerning the format of Date expressions.

To access these routines, double click on 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.<br>The Date format is: "yyyy", "MM", "dd", "HH", "mm", "ss" or "SSS". | `TalendDate.addDate("String date initiale", "format Date - eg.: yyyy/MM/dd", whole n,"format of the part of the date to which n is to be added - eg.:yyyy").` |
| *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 older than the second and -1 if it is more recent than the second. | `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")` |
| *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, 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)` |
| *getPartOfDate* | returns part of a date according to the format specified. This string can contain fixed character strings or variables linked to the date. | `TalendDate.getPartOfDate("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")` |
| *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")` |
| *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")` |

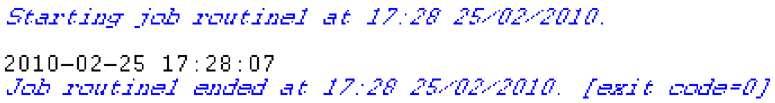
# C.5.1. How to format a Date

The **formatDate** routine is easy to use, along with a **tJava** component:



```
System.out.println(TalendDate.formatDate("dd-MM-yyyy", new Date()));
```

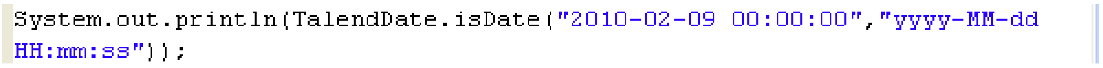
The current date is initialized according to the pattern specified by the `new date()` Java function and is displayed in the **Run** view:



```
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]
```

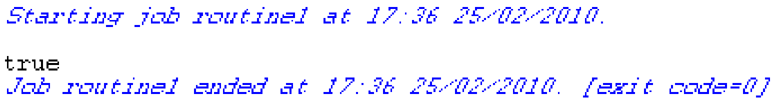
## C.5.2. How to check a Date

It is easy to use the **isDate** routine, along with a **tJava** component to check if a date expression is in the format specified:



```
System.out.println(TalendDate.isDate("2010-02-09 00:00:00","yyyy-MM-dd
HH:mm:ss"));
```

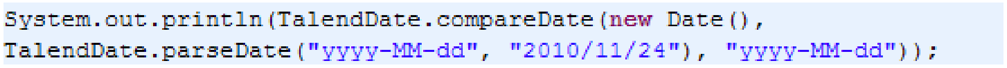
A boolean is returned in the **Run** view:



```
Starting job routine1 at 17:36 25/02/2010.

true
Job routine1 ended at 17:36 25/02/2010. [exit code=0]
```

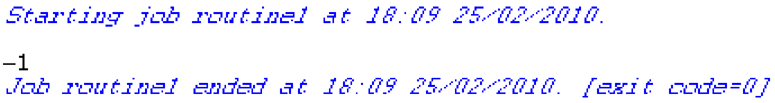
## C.5.3. How to compare Dates

It is easy to use the **formatDate** routine, along with a **tJava** component to check if the current date is more recent than a specific date, according to the format specified.



```
System.out.println(TalendDate.compareDate(new Date(),
TalendDate.parseDate("yyyy-MM-dd", "2010/11/24"), "yyyy-MM-dd"));
```

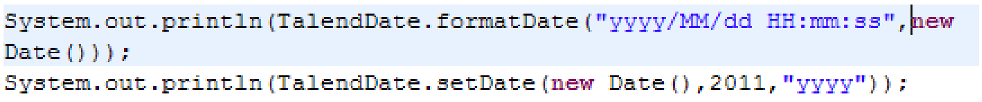
The current date is initialized by the Java function `new date()`and the value -1 is displayed in the **Run** view to indicate that the current date precedes the reference date.



```
Starting job routine1 at 18:09 25/02/2010.

-1
Job routine1 ended at 18:09 25/02/2010. [exit code=0]
```

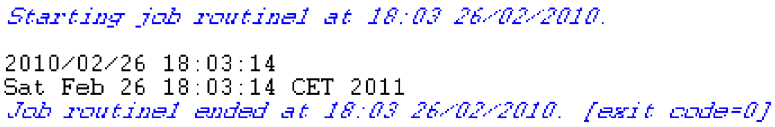
## C.5.4. How to configure a Date

It is easy to use the **setDate** routine, along with a **tJava** component to change the year of the current date, for example:



```
System.out.println(TalendDate.formatDate("yyyy/MM/dd HH:mm:ss",new
Date()));
System.out.println(TalendDate.setDate(new Date(),2011,"yyyy"));
```

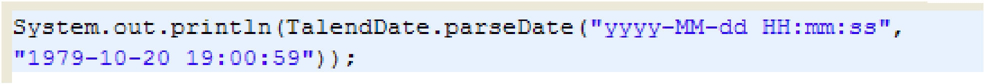
The current date, followed by the new date are displayed in the **Run** view:



```
Starting job routine1 at 18:03 26/02/2010.

2010/02/26 18:03:14
Sat Feb 26 18:03:14 CET 2011
Job routine1 ended at 18:03 26/02/2010. [exit code=0]
```

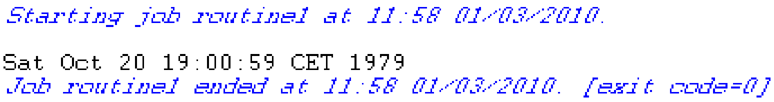
## C.5.5. How to parse a Date

It is easy to use the **parseDate** routine, along with a **tJava** component to change a date string from one format into another Date format, for example:



```
System.out.println(TalendDate.parseDate("yyyy-MM-dd HH:mm:ss",
"1979-10-20 19:00:59"));
```

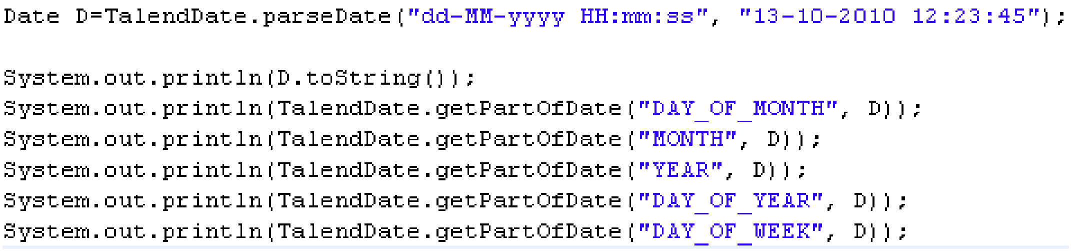
The string is changed and returned in the Date format:



```
Starting job routine1 at 11:58 01/03/2010.

Sat Oct 20 19:00:59 CET 1979
Job routine1 ended at 11:58 01/03/2010. [exit code=0]
```

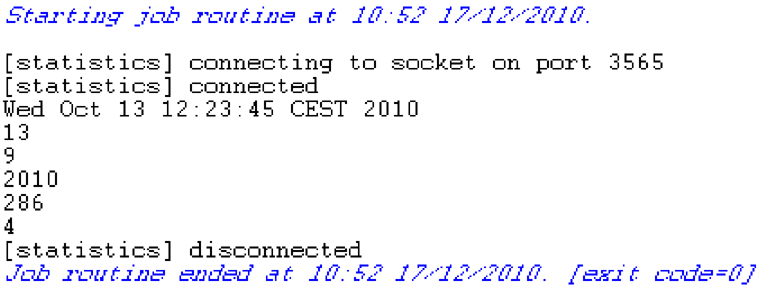
## C.5.6. How to retrieve part of a Date

It is easy to use the **getPartOfDate** routine, along with a **tJava** component to retrieve part of a date, for example:



```
Date D=TalendDate.parseDate("dd-MM-yyyy HH:mm:ss", "13-10-2010 12:23:45");

System.out.println(D.toString());
System.out.println(TalendDate.getPartOfDate("DAY_OF_MONTH", D));
System.out.println(TalendDate.getPartOfDate("MONTH", D));
System.out.println(TalendDate.getPartOfDate("YEAR", D));
System.out.println(TalendDate.getPartOfDate("DAY_OF_YEAR", D));
System.out.println(TalendDate.getPartOfDate("DAY_OF_WEEK", D));
```

In this example, the day of month (DAY_OF_MONTH), the month (MONTH), the year (YEAR), the day number of the year (DAY_OF_YEAR) and the day number of the week (DAY_OF_WEEK) are returned in the **Run** view. All the returned data are numeric data types.

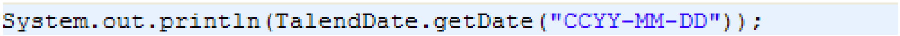


```
Starting job routine at 10:52 17/12/2010.

[statistics] connecting to socket on port 3565
[statistics] connected
Wed Oct 13 12:23:45 CEST 2010
13
9
2010
286
4
[statistics] disconnected
Job routine ended at 10:52 17/12/2010. [exit code=0]
```

In the **Run** view, the date string referring to the months (MONTH) starts with 0 and ends with 11: 0 corresponds to January, 11 corresponds to December.

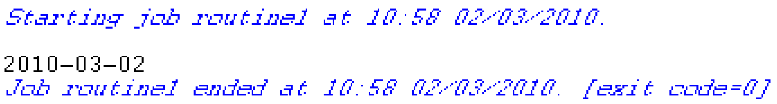
## C.5.7. How to format the Current Date

It is easy to use the **getDate** routine, along with a **tJava** component, to retrieve and format the current date according to a specified format, for example:



```
System.out.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]
```

# C.6. TalendString Routines

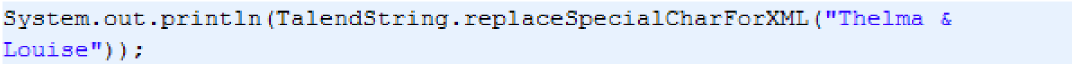
The **TalendString** routines allow you to carry out various operations on alphanumerical expressions.

To access these routines, double click on **TalendString** under the **system** folder. The **TalendString** class contains the following routines:

| Routine | Description | Syntax |
|---|---|---|
| *replaceSpecialCharForXML* | returns a string from which the special characters (eg.:: <, >, &...) have been replaced by equivalent XML characters. | `TalendString.replaceSpecialCharForXML ("string containing the special characters - eg.: Thelma & Louise")` |
| *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. | `TalendString.checkCDATAForXML("string to be parsed")` |
| *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)` |

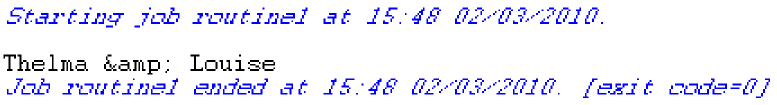
## C.6.1. How to format an XML string

It is easy to run the **replaceSpecialCharForXML** routine along with a **tJava** component, to format a string for XML:



```
System.out.println(TalendString.replaceSpecialCharForXML("Thelma &
Louise"));
```

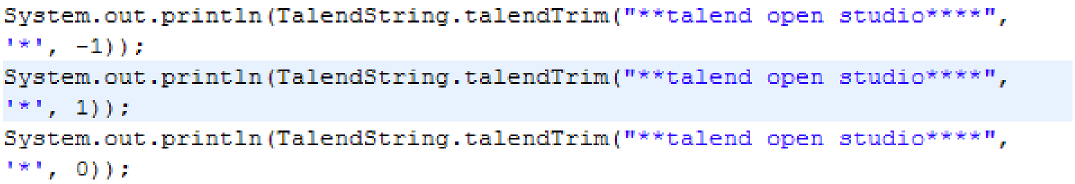
In this example, the "&" character is replaced in order to make the string XML compatible:



```
Starting job routine1 at 15:48 02/03/2010.

Thelma &amp; Louise
Job routine1 ended at 15:48 02/03/2010. [exit code=0]
```

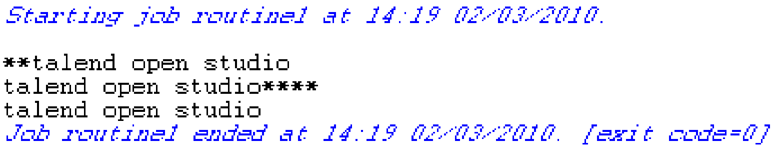
## C.6.2. How to trim a string

It is easy to use the **talendTrim** routine, along with a **tJava** component to remove the string padding characters from the start and end of the string:



```
System.out.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));
```

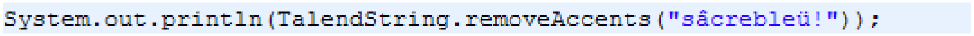
The star characters are removed from the start, then the end of the string and then finally from both ends:



```
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]
```

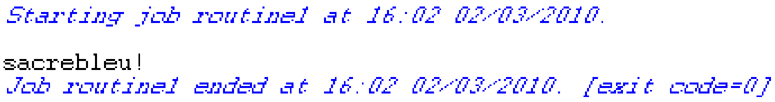
## C.6.3. How to remove accents from a string

It is easy to use the **removeAccents** routine, along with a **tJava** component, to replace the accented characters, for example:



```
System.out.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]
```

# Appendix D. SQL template writing rules

This chapter describes the rules applied for the creation of SQL templates. It aims to help users of SQL templates in *Talend Open Studio for Big Data* to understand and develop the SQL templates for more customized usage.

These rules provide details that you have to respect when writing the template statement, a comment line or the different relevant syntaxes.

These rules helps to use the SQL code in specific use cases, such as to access the various parameters defined in components.

# D.1. SQL statements

An SQL statement can be any valid SQL statement that the related JDBC is able to execute. The SQL template code is a group of SQL statements. The basic rules to write an SQL statement in the SQL template editor are:

- An SQL statement must end with `;`.
- An SQL statement can span lines. In this case, no line should be ended with `;` except the last one.

# D.2. Comment lines

A comment line starts with `#` or `--`. Any line that starts with `#` or `--` will be ignored in code generating.

There is no exception to the lines in the middle part of a SQL statement or within the `<%... %>` syntax.

# D.3. The `<%...%>` syntax

This syntax can span lines. The following list points out what you can do with this syntax and what you should pay attention to.

- You can define new variables, use Java logical code like `if`, `for` and `while`, and also get parameter values.

  For example, if you want to get the *FILE_Name* parameter, use the code as follows:

```
<%
String filename = __FILE_NAME__;
%>
```

- This syntax cannot be used within an SQL statement. In other words, it should be used between two separated SQL statements.

  For example, the syntax in the following code is valid.

```
#sql sentence
DROP TABLE temp_0;
<%
#loop
for(int i=1; i<10; i++){
%>
#sql sentence
DROP TABLE temp_<%=i %>;
<%
}
%>
#sql sentence
DROP TABLE temp_10;
```

In this example, the syntax is used between two separated SQL templates: `DROP TABLE temp_0;` and `DROP TABLE temp_<%=i%>;`.

The SQL statements are intended to remove several tables beginning from *temp_0*. The code between `<%` and `%>` generate a sequence of number in loop to identify tables to be removed and close the loop after the number generation.

- Within this syntax, the `<%=...%>` or `</.../>` syntax should not be used.

`<%=...%>` and `</.../>` are also syntax intended for the SQL templates. The below sections describe related information.

Parameters that the SQL templates can access with this syntax are simple. They are often used for connection purpose and can be easily defined in components, such as *TABLE_NAME*, *DB_VERSION*, *SCHEMA_TYPE*, etc.

## D.4. The `<%=...%>` syntax

This syntax cannot span lines and is used for SQL statement. The following list points out what you can do with this syntax and what you should pay attention to.

- This syntax can be used to generate any variable value, and also the value of any existing parameter.
- No space char is allowed after `<%=`.
- Inside this syntax, the <%…%> or </…/> syntax should not be used.

The statement written in the below example is a valid one.

```
#sql sentence
DROP TABLE temp_<%=__TABLE_NAME__ %>;
```

The code is used to remove the table defined through an associated component.

For more information about what components are associated with the SQL templates, see chapter *Designing a data integration Job*.

For more information on the `<%...%>` syntax, see section *The <%...%> syntax*.

For more information on the `</.../>` syntax, see the following section.

Parameters that the SQL templates can access with this syntax are simple. They are often used for connection purpose and can be easily defined in components, such as *TABLE_NAME*, *DB_VERSION*, *SCHEMA_TYPE*, etc.

## D.5. The `</.../>` syntax

This syntax cannot span lines. The following list points out what you can do with this syntax and what you should pay attention to.

- It can be used to generate the value of any existing parameter. The generated value should not be enclosed by quotation marks.
- No space char is allowed after `</` or before `/>`.
- Inside this syntax, the <%…%> or <%=…%> syntax should not be used.

The statement written in the below example is a valid one.

```
#sql sentence
DROP TABLE temp_</TABLE_NAME/>;
```

The statement identifies the *TABLE_NAME* parameter and then removes the corresponding table.

For more information on the <%…%> syntax, see section *The <%...%> syntax*.

For more information on the <%=…%> syntax, see section *The <%=...%> syntax*.

The following sections present more specific code used to access more complicated parameters.

Parameters that the SQL templates can access with this syntax are simple. They are often used for connection purpose and can be easily defined in components, such as *TABLE_NAME*, *DB_VERSION*, *SCHEMA_TYPE*, etc.

# D.6. Code to access the component schema elements

Component schema elements are presented on a schema column name list (delimited by a dot “.”). These elements are created and defined in components by users.

The below code composes an example to access some elements included in a component schema. In the following example, the *ELT_METADATA_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.

*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 section *Basic Settings tab*.

# D.7. Code to access the component matrix properties

The component matrix properties are created and changed by users according to various data transformation purposes. These properties are defined by tabular parameters, for example, the *operation* parameters or *groupby* parameters that users can define through the **tSQLTemplateAggregate** component.

To access these tabular parameters that are naturally more flexible and complicated, two approaches are available:

- The `</.../>` approach:

`</.../>` is one of the syntax used by the SQL templates. This approach often needs hard coding for every parameter to be extracted.

For example, a new parameter is created by user and is given the name *NEW_PROPERTY*. If you want to access it by using `</NEW_PROPERTY/>`, the below code is needed.

```
else if (paramName.equals("NEW_PROPERTY")) {
List<Map<String, String>> newPropertyTableValue = (List<Map<String, String>>)
ElementParameterParser.getObjectValue(node, "__NEW_PROPERTY__");
for (int ii = 0; ii <newPropertyTableValue.size(); ii++) {
Map<String, String> newPropertyMap =newPropertyTableValue.get(ii);
realValue += ...;//append generated codes
……
}
}
```

- The `EXTRACT(__GROUPBY__);` approach:

The below code shows the second way to access the tabular parameter *(GROUPBY)*.

```
<%
String query = "insert into " + __TABLE_NAME__ + "(id, name, date_birth) select sum(id), name, date_birth from cust_teradata group by";
EXTRACT(__GROUPBY__);
for (int i=0; i < __GROUPBY_LENGTH__ ; i++) {
query += (__GROUPBY_INPUT_COLUMN__[i] + " ");
}
%>
<%=query %>;
```

When coding the statements, respect the rules as follows:

- The extract statement must use `EXTRACT(__GROUPBY__);`. Upcase should be used and no space char is allowed. This statement should be used between `<%` and `%>`.
- Use `__GROUPBY_LENGTH__`, in which the parameter name is followed by `_LENGTH`, to get the line number of the tabular *GROUPBY* parameters you define in the **Groupby** area on a **Component** view. It can be used between `<%` and `%>` or `<%=` and `%>`.
- Use code like `__GROUPBY_INPUT_COLUMN__[i]` to extract the parameter values. This can be used between `<%` and `%>` or between `<%=` and `%>`.
- In order to access the parameter correctly, do not use the identical name prefix for several parameters. For example in the component, avoid to define two parameters with the names `PARAMETER_NAME` and `PARAMETER_NAME_2`, as the same prefix in the names causes erroneous code generation.