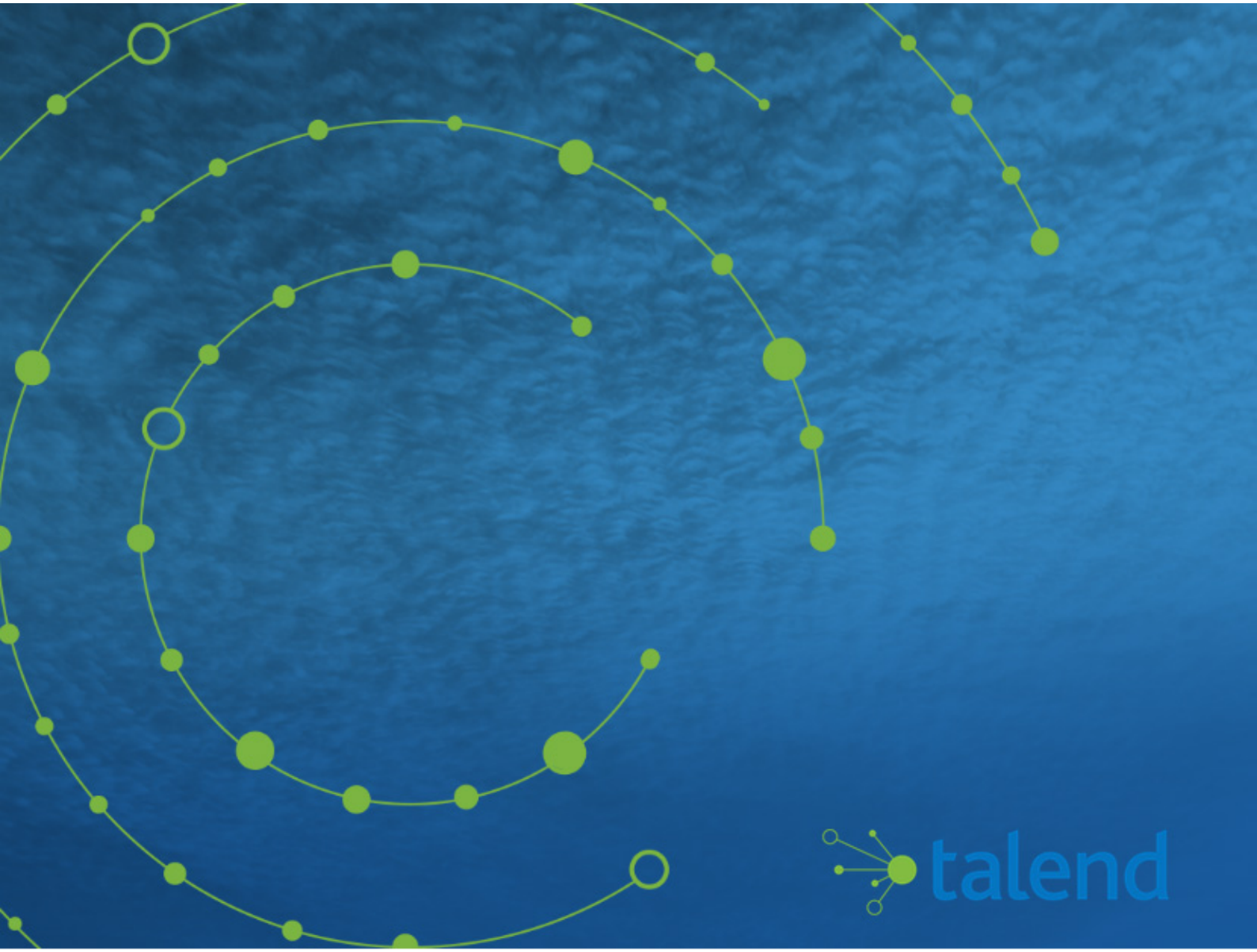


Data Integration Job Examples

7.0.1

Contents

Copyleft

Adapted for 7.0.1. Supersedes previous releases.

Publication date: April 13, 2018

Notices

License Agreement

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

tMap Job example

To illustrate the way Talend Studio operates, find below a real-life example scenario. In this scenario, we will load a MySQL table with a file, that gets transformed on the fly. Then in a further step, we will select the data to be loaded using a dynamic filter.

Before actually starting the Job, let's inspect the input data and the expected output data.

Input data

Our input file, the data of which will be loaded into the database table, lists clients from all over the State of California.

The file structure usually called **Schema** in Talend Studio includes the following columns:

- First name
- Last name
- Address
- City

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.

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`.

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.

Step 1: Job creation, input definition, file reading

Procedure

1. Launch Talend Studio, and create a local project or import the demo project if you are launching Talend Studio for the first time.
2. To create the Job, right-click **Job Designs** in the **Repository** tree view and select **Create Job**.
3. 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.
4. 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.
5. Let's define now the reading properties for this component: File path, column delimiter, encoding... To do so, let's use the **Metadata Manager**. This tool offers numerous wizards that will help us to configure parameters and allow us to store these properties for a one-click re-use in all future Jobs we may need.
6. As our input file is a delimited flat file, let's select **File Delimited** on the right-click list of the **Metadata** folder in the **Repository** tree view. Then select **Create file delimited**.

A wizard dedicated to delimited file thus displays:

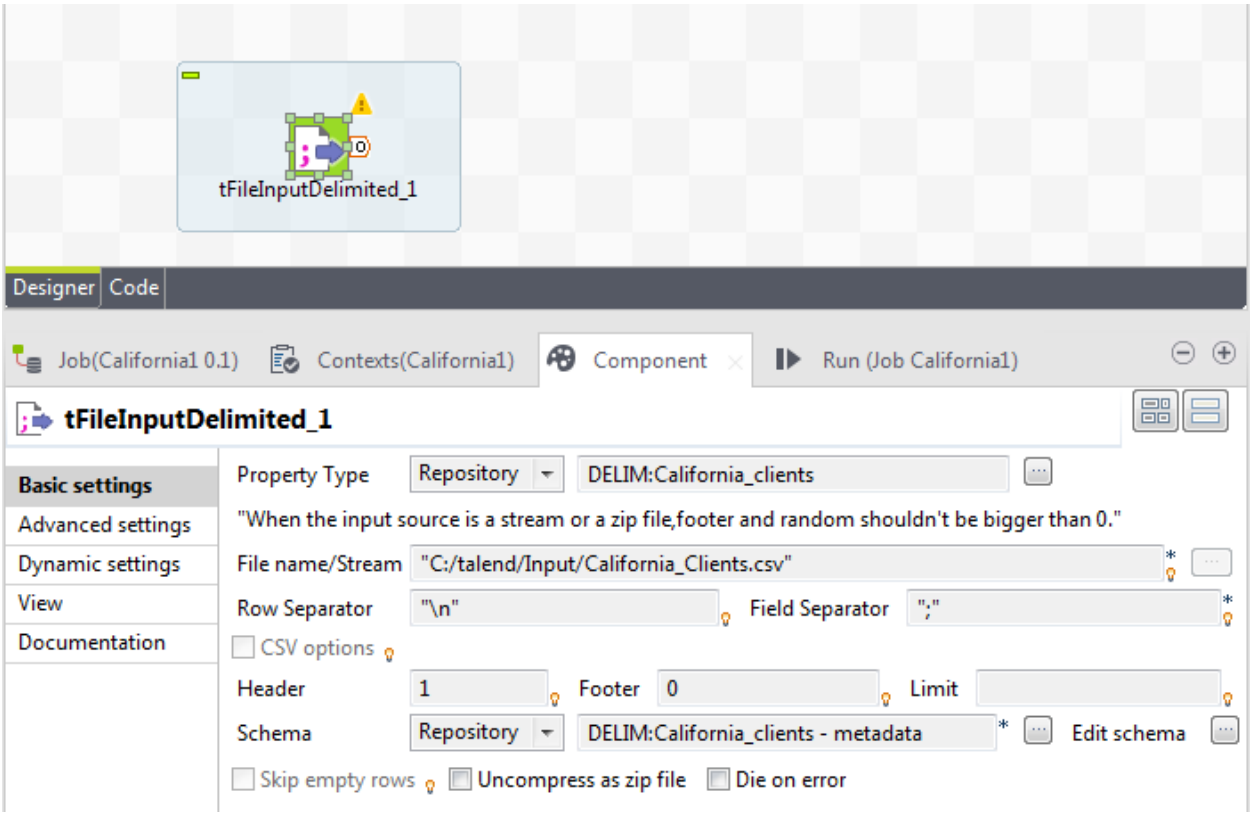
- At Step 1, only the **Name** field is required: simply type in **California_clients** and go to the next Step.
- At Step 2, select the input file (`California_Clients.csv`) via the **Browse...** button. Immediately an extract of the file shows on the Preview, at the bottom of the screen so that you can check its content. Click **Next**.
- At Step 3, we will define the file parameters: file encoding, line and column delimiters... As our input file is pretty standard, most default values are fine. The first line of our file is a header containing column names. To retrieve automatically these names, click **Set heading row as column names** then click **Refresh Preview**. And click **Next** to the last step.
- At Step 4, each column of the file is to be set. The wizard includes algorithms which guess types and length of the column based on the file first data rows. The suggested data description (called schema in Talend Studio) can be modified at any time. In this particular scenario, they can be used as is.

There you go, the **California_clients** metadata is complete!

We can now use it in our input component. Select the **tFileInputDelimited** you had dropped on the design workspace earlier, and select the **Component** view at the bottom of the window.

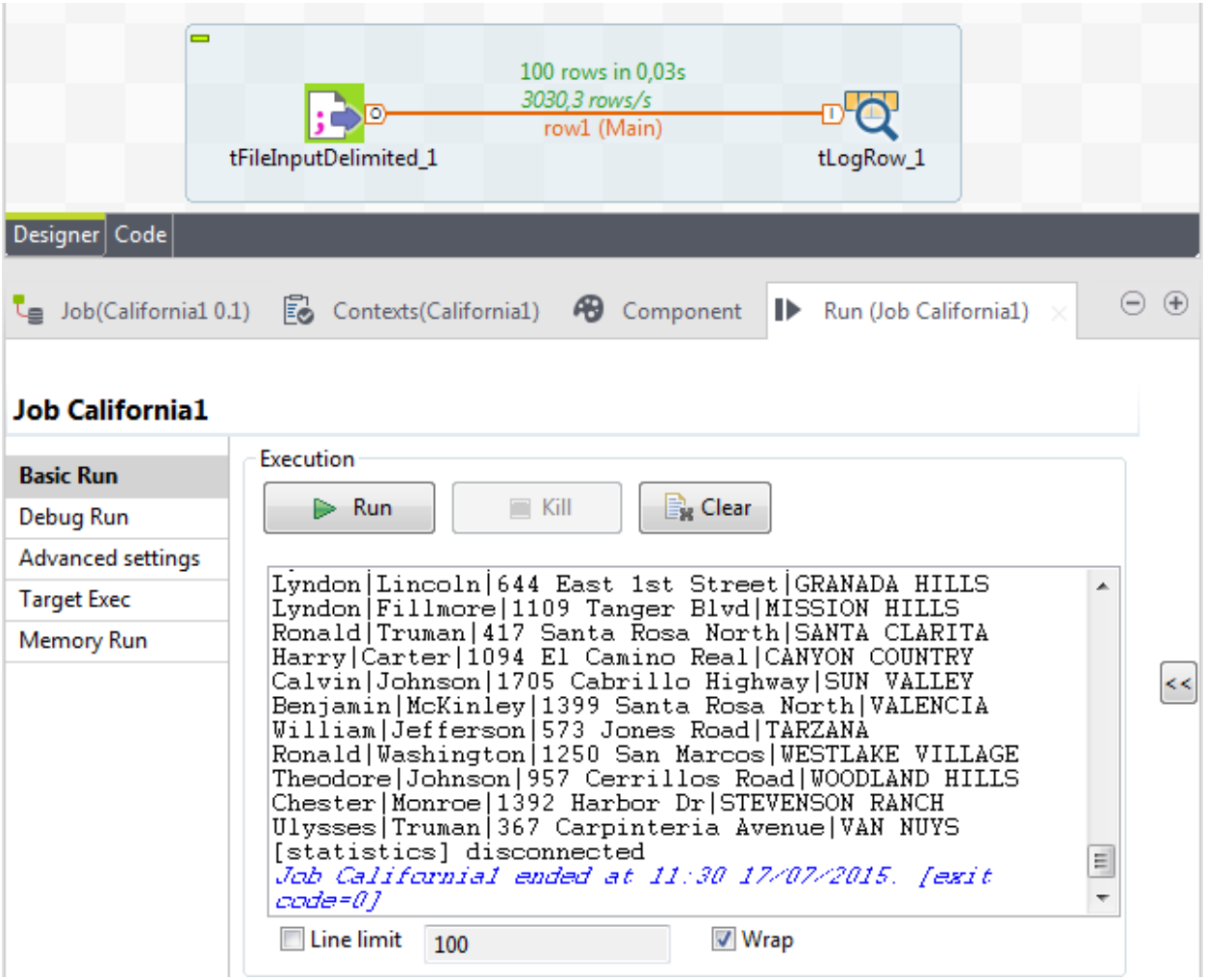
7. Select the vertical tab **Basic settings**. In this tab, you'll find all technical properties required to let the component work. Rather than setting each one of these properties, let's use the Metadata entry we just defined.
8. Select **Repository** as **Property type** in the list. A new field shows: **Repository**, click "**...**" button and select the relevant Metadata entry on the list: **California_clients**.

   You can notice now that all parameters get automatically filled out.



At this stage, we will terminate our flow by simply sending the data read from this input file onto the standard output (StdOut).

9. 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**.
10. This Job is now ready to be executed. To run it, select the **Run** tab on the bottom panel.
11. Enable the statistics by selecting the **Statistics** check box in the **Advanced Settings** vertical tab of the **Run** view, then run the Job by clicking **Run** in the **Basic Run** tab.



The content of the input file display thus onto the console.

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:

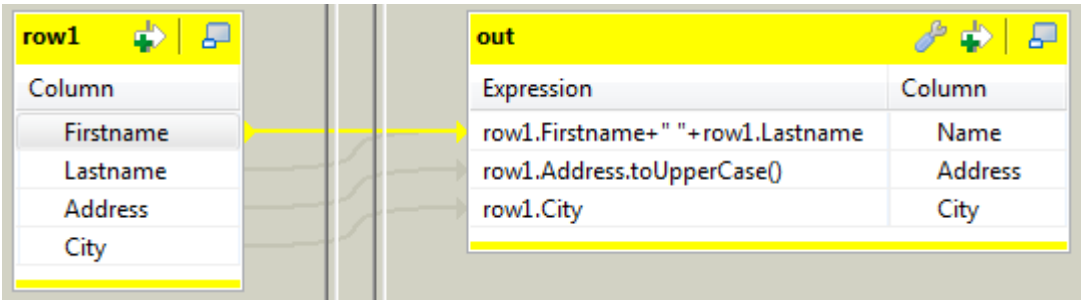
- multiple inputs and outputs
- search for reference (simple, cartesian product, first, last match...)
- join (inner, outer)
- transformations
- rejections

- and more...

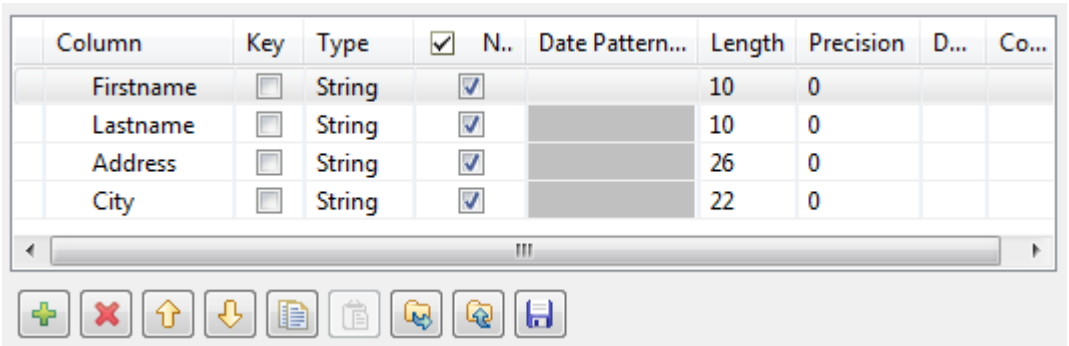
Procedure

1. 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.
2. 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**.
3. 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**).
4. 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.



5. Then carry out the following transformations on each column:
   - Change the Expression of the **Name** column to `row1.Firstname + " " + row1.LastName`. This concatenates the **Firstname** column with the **Lastname** column, 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.
6. 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 Map editor and proceed as follows:



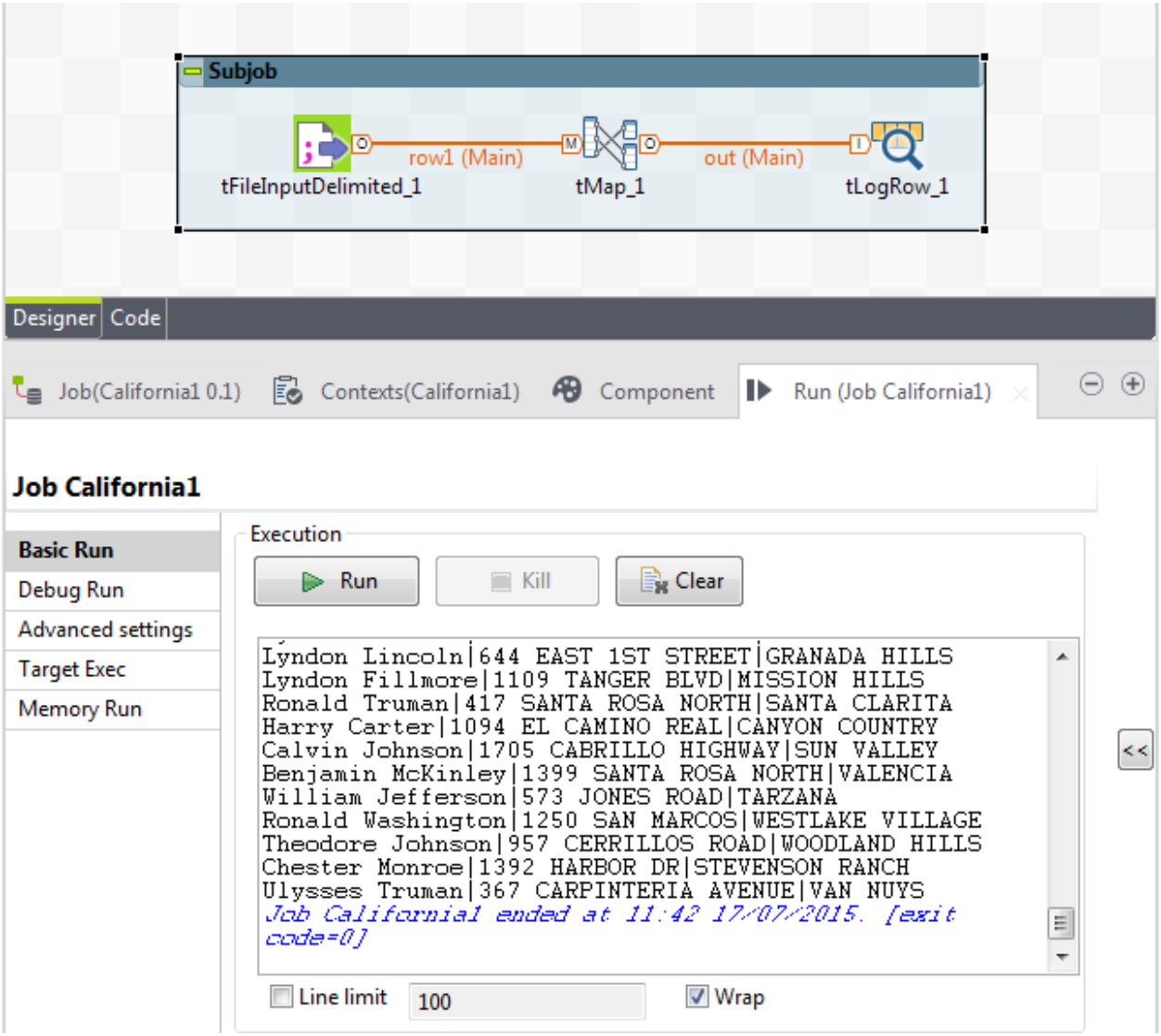
   1. Select the column to be removed from the schema, and click the cross icon.
   2. Select the column of which you need increase the length size.
   3. Type in the length size you intend in the length column. In this example, change the length of every remaining column to 40.

Note:

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.

7. Click **OK** to validate the changes and close the Map editor interface.
8. If you run your Job at this stage (via the **Run** view as we did it before), you'll notice the changes that you defined are implemented.



For example, the addresses are displayed in upper case and the first names and last names are gathered together in the same column.

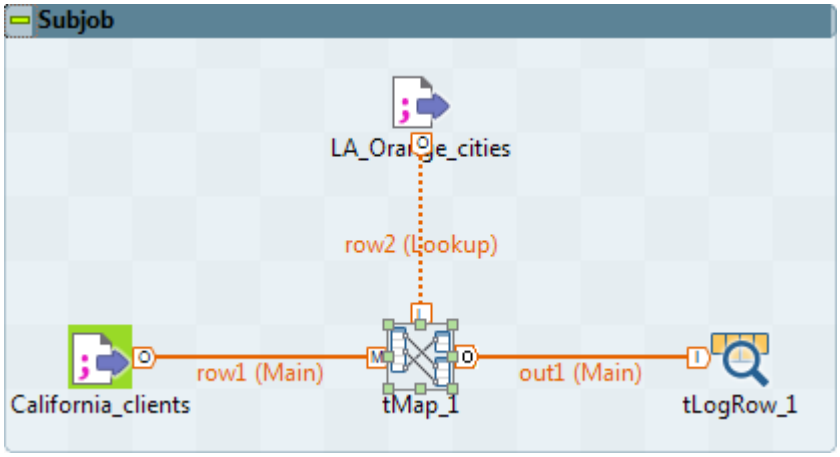
Step 3: Reference file definition, remapping, inner join mode selection

Procedure

1. Define the Metadata corresponding to the `LosAngelesandOrangeCounties.txt` file just the way we did it previously for `California_clients` file, using the wizard.

   At Step1 of the wizard, name this metadata entry: `LA_Orange_cities`.

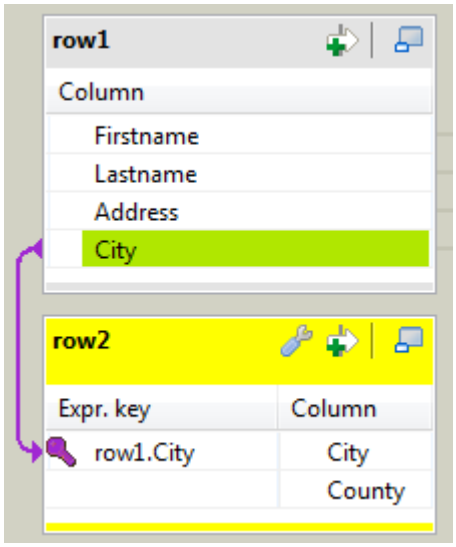
2. Then drop this newly created metadata to the top of the design area to create automatically a reading component pointing to this metadata.
3. Then link this component to the **tMap** component.



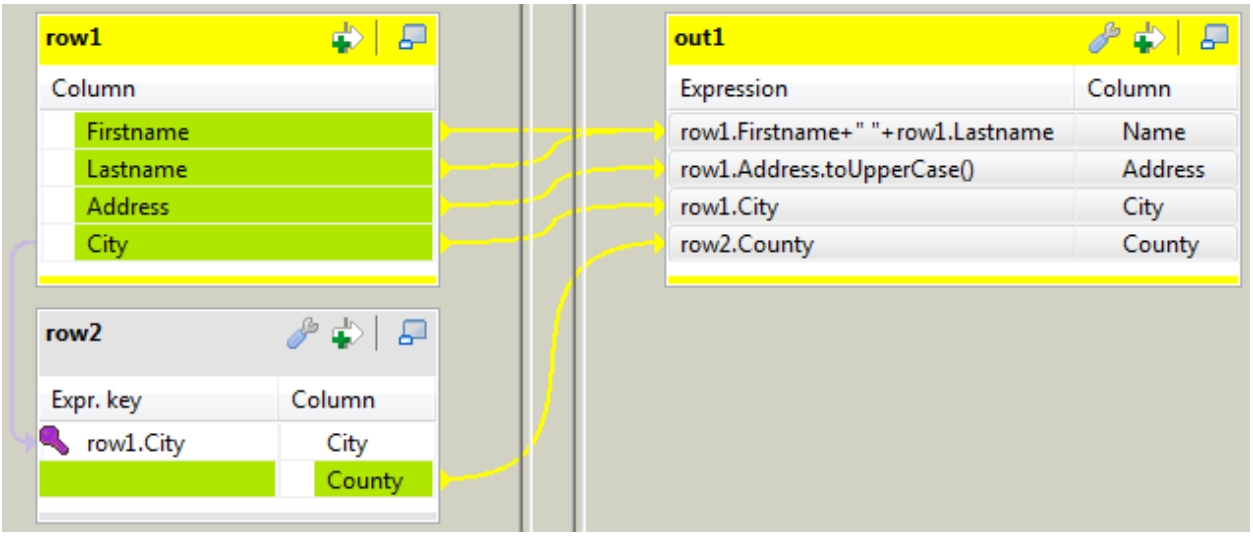
4. Double-click again on the **tMap** component to open its interface. Note that the reference input table (**row2**) corresponding to the LA and Orange county file, shows to the left of the window, right under your main input (**row1**).
5. Now let's define the join between the main flow and the reference flow.

   In this use case, the join is pretty basic to define as the **City** column is present in both files and the data match perfectly. But even though this was not the case, we could have carried out operations directly at this level to establish a link among the data (padding, case change...)

   To implement the join, drop the **City** column from your first input table onto the **City** column of your reference table. A violet link then displays, to materialize this join.

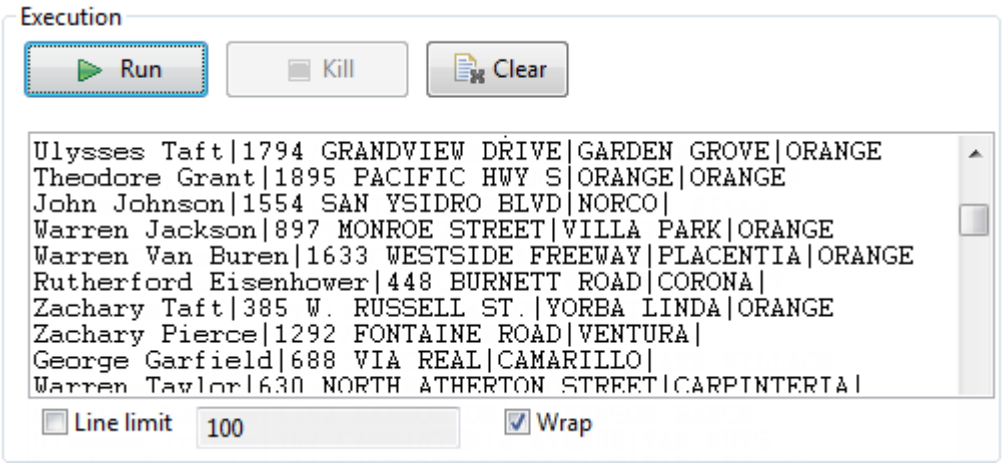


   Now, we are able to use the **County** column from the reference table in the output table (**out1**).



6. Eventually, click the **OK** button to validate your changes, and run the new Job.

   The following output should display on the console.



   As you can notice, the last column is only filled out for Los Angeles and Orange counties' cities. For all other lines, this column is empty. The reason for this is that by default, the **tMap** implements a left outer join mode. If you want to filter your data to only display lines for which a match is found by the **tMap**, then open again the **tMap**, click the **tMap settings** button and select the **Inner Join** in the **Join Model** list on the reference table (**row2**).

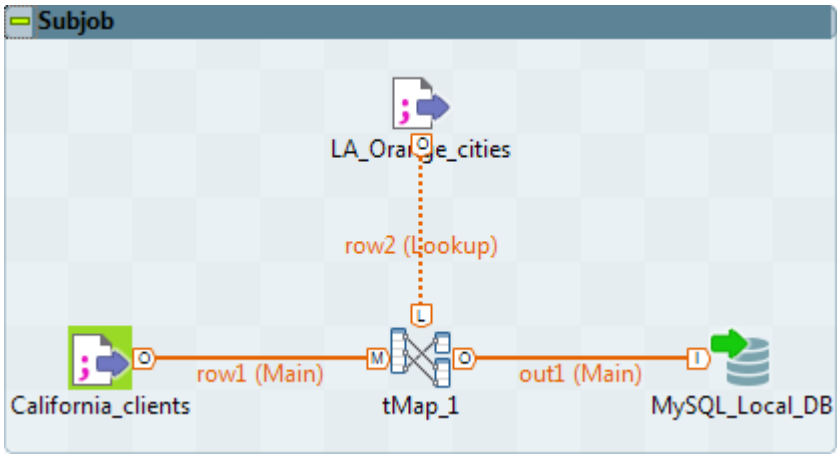
Step 4: Output to a MySQL table

Our Job works perfectly! To finalize it, let's direct the output flow to a MySQL table.

Procedure

1. To do so, let's first create the Metadata describing the connection to the MySQL database. Expand the **Metadata** > **MySQL** nodes in the Repository and double-click **DemoMySQL** (on the condition that you imported the Demo project properly) to open the Metadata wizard.
2. 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**.

3. Drop this metadata to the right of the design workspace, while maintaining the **Ctrl** key down, in order to create automatically a **tMysqlOutput** component.
4. Remove the **tLogRow** component from your Job.
5. Reconnect the **out1** output flow from the **tMap** to the new component **tMysqlOutput**.



6. On the **Basic Settings** tab of this component:
   a) Type in `LA_Orange_Clients` in the **Table** field, in order to name your target table which will get created on the fly.
   b) Select the **Drop table if exists and create** option or on the **Action on table** field.
   c) 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.
7. Run again the Job.

Results

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).

Using the output stream feature

The following use case aims to show how to use the output stream feature in a number of components in order to greatly improve the output performance.

In this scenario, a pre-defined csv file containing customer information is loaded in a database table. Then the loaded data is selected using a **tMap**, and output to a local file and to the console using the output stream feature.

For the principle of the **Use Output Stream** feature, see the *How to use the Use Output Stream feature* section of the Talend Studio User Guide at https://help.talend.com.

Input data

The input file, the data of which will be loaded into the database table, contains customer information of various aspects.

The file structure usually called **Schema** in Talend Studio includes the following columns:

- id (Type: Integer)
- CustomerName (Type: String)
- CustomerAge (Type: Integer)
- CustomerAddress (Type: String)
- CustomerCity (Type: String)
- RegisterTime (Type: Date)

Output data

The **tMap** component is used to select the **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.

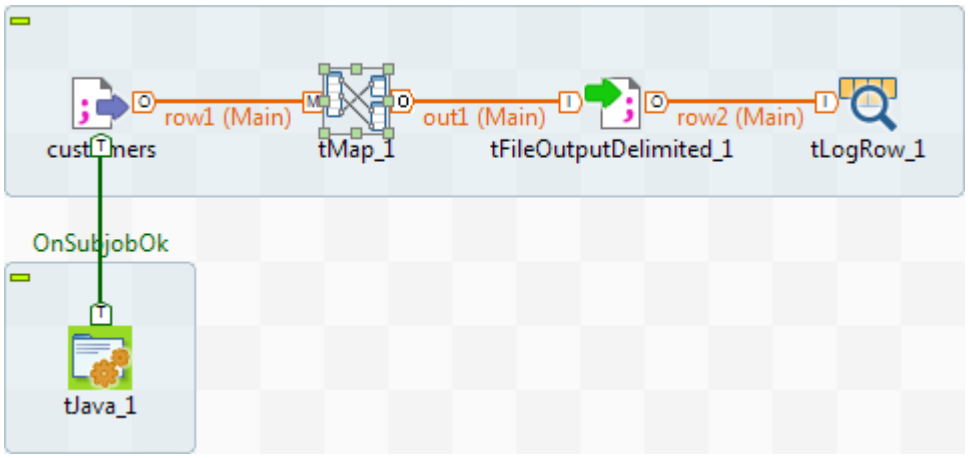
Translating the scenario into a Job

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.

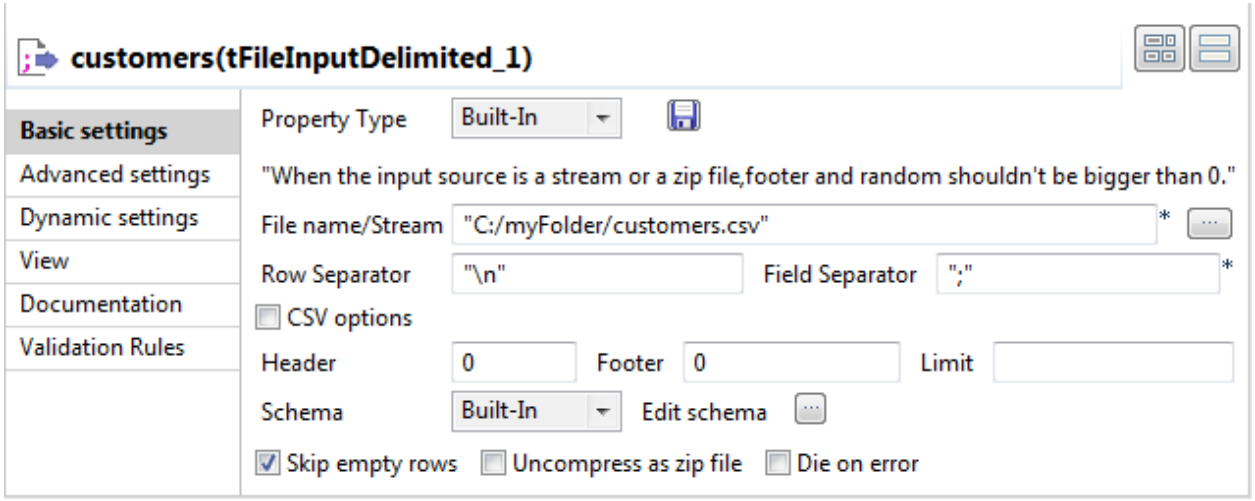


Step 1: Reading input data from a local file

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**.

Procedure

1. Drop a **tFileInputDelimited** component onto the design workspace, and double-click the to open the **Basic settings** view to set its properties.



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...				

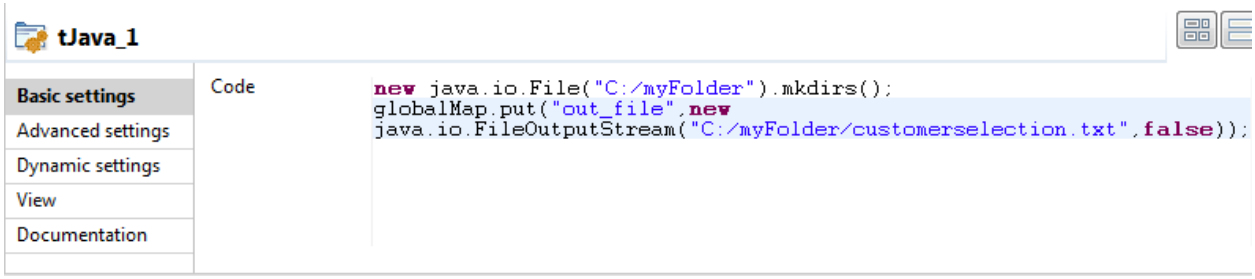
5. Click **OK** to close the dialog box.

Step 2: 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.

Procedure

1. Drop a **tJava** component onto the design workspace, and double-click it to open the **Basic settings** view to set its properties.



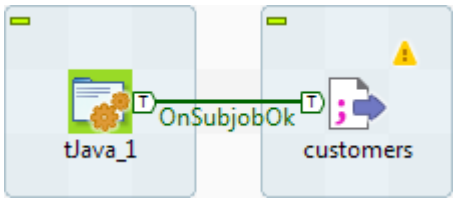
2. Fill in the **Code** area with the following command:

```
new java.io.File("C:/myFolder").mkdirs();
globalMap.put("out_file",new java.io.FileOutputStream("C:/myFolder/cust
omerselection.txt",false));
```

Note:

The command we typed in this step will create a new directory `C:/myFolder` for saving the output file `customerselection.txt`. You can customize the command in accordance with actual practice.

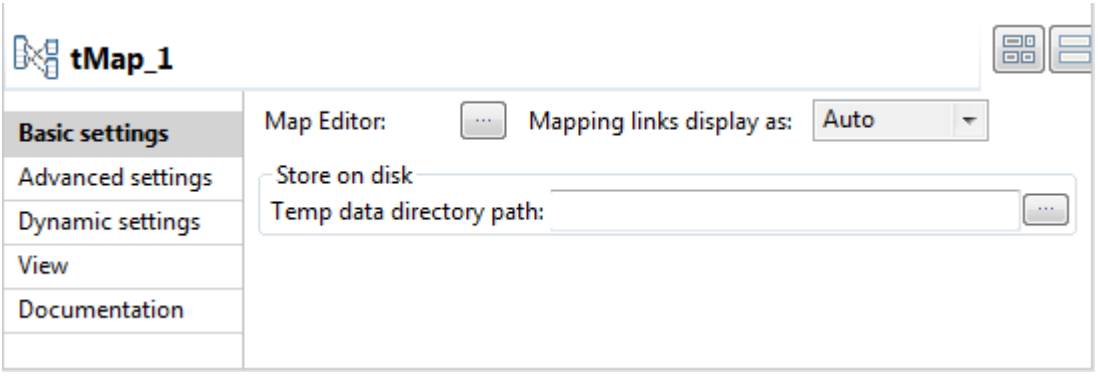
3. Connect **tJava** to **tFileInputDelimited** using a **Trigger** > **On Subjob Ok** connection. This will trigger **tJava** when subjob that starts with **tFileInputDelimited** succeeds in running.



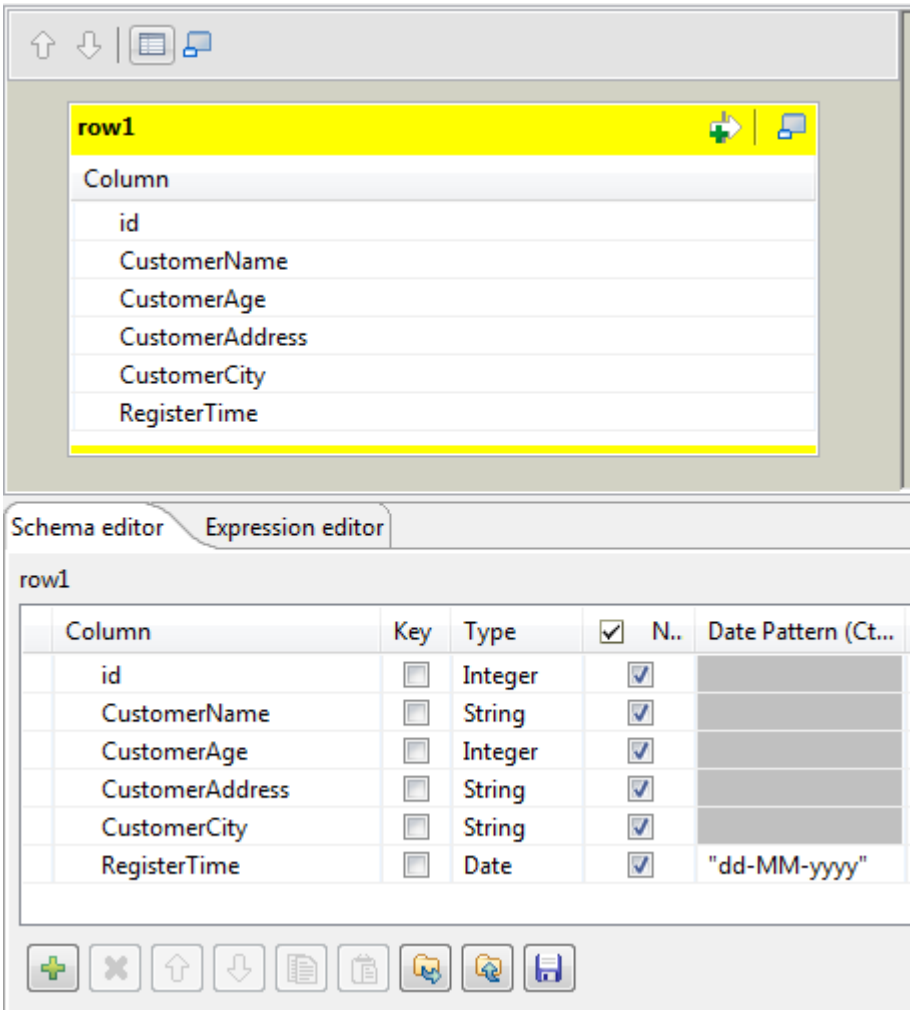
Step 3: Mapping the data using the tMap component

Procedure

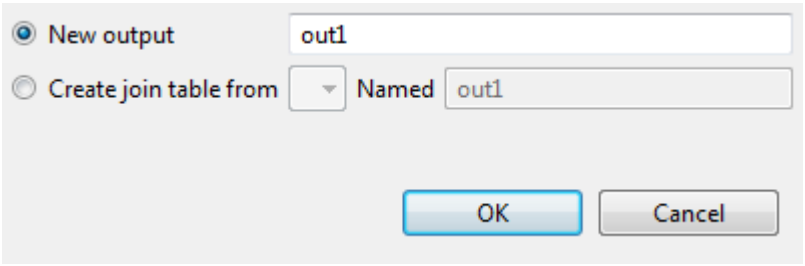
1. Drop a **tMap** component onto the design workspace, and double-click it to open the **Basic settings** view to set its properties.



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:

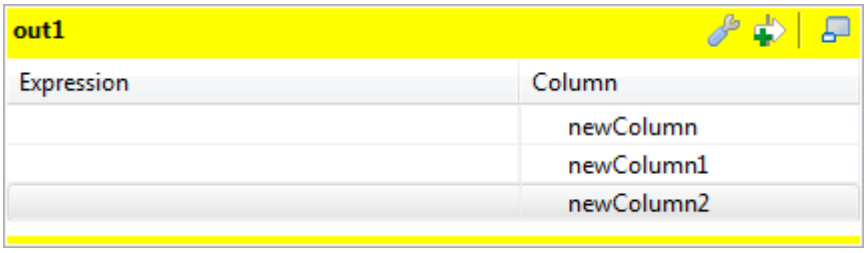


4. Click the plus button on the right to add a schema of the outgoing data flow.

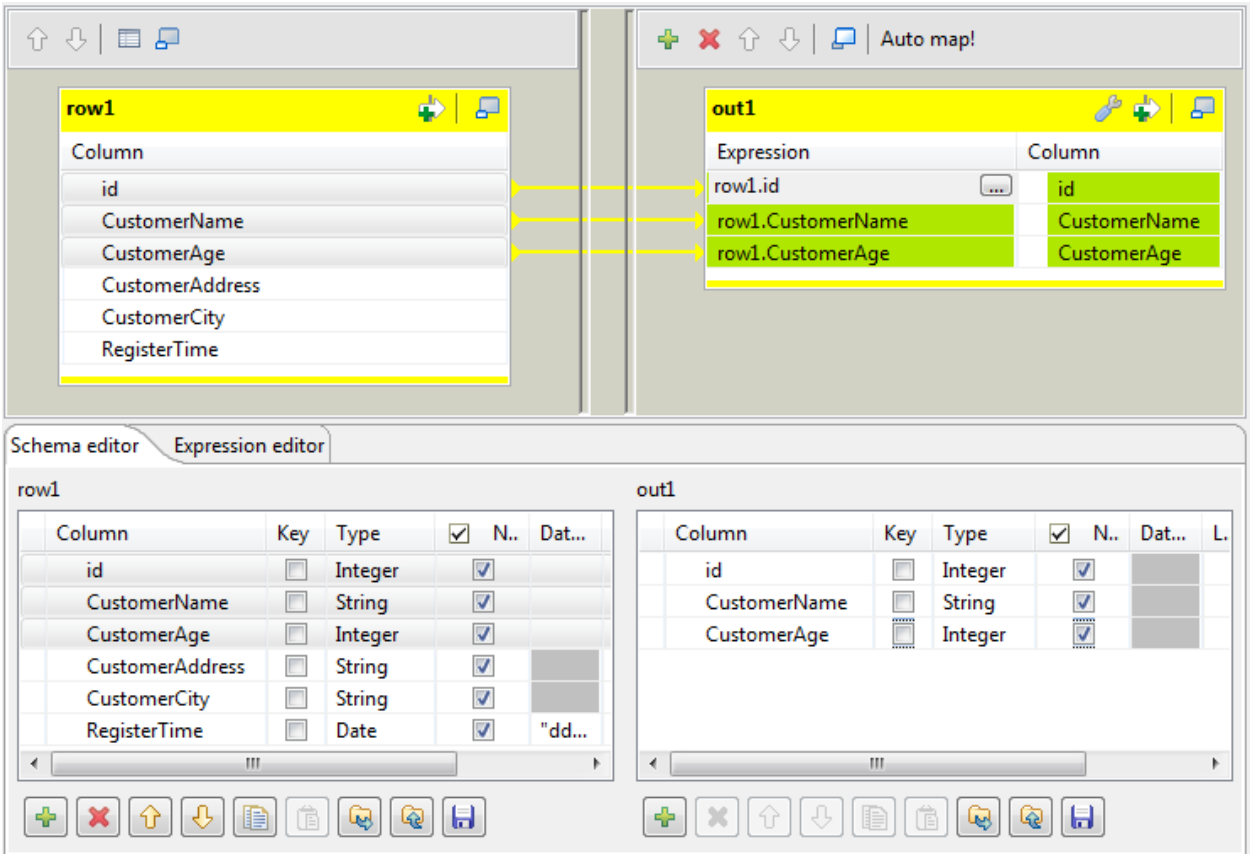


5. Select **New output** and click **OK** to save the output schema.

   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.



7. Drop the **id**, **CustomerName** and **CustomerAge** columns onto their respective line on the right.



8. Click **OK** to save the settings.

Step 4: Outputing the selected data stream

Procedure

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:

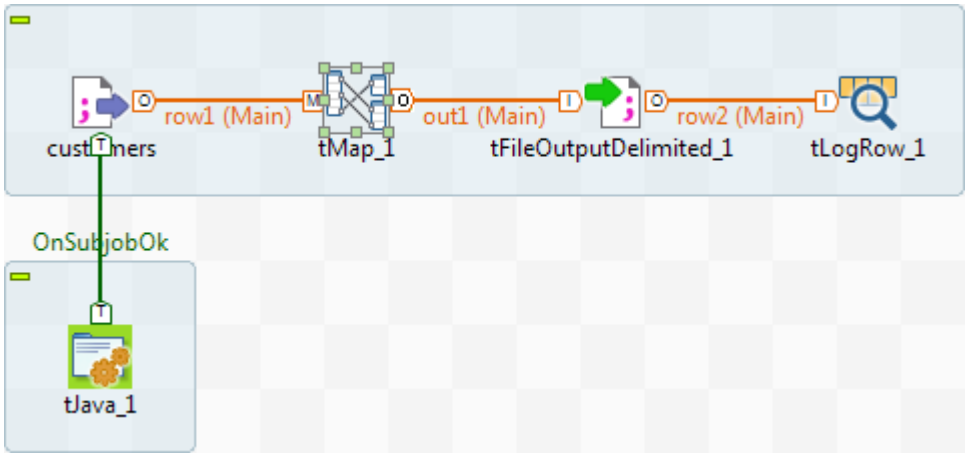
```
(java.io.OutputStream)globalMap.get("out_file")
```

Note:

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.

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.
5. Drop a **tLogRow** component onto the design workspace, and double-click it to open its **Basic settings** view.
6. Select the **Table** radio button in the **Mode** area.
7. Connect **tFileOutputDelimited** to **tLogRow** using a **Row** > **Main** connection.
8. Click **Sync columns** to retrieve the schema defined in the preceding component.

This Job is now ready to be executed.



9. Press **Ctrl+S** to save your Job and press **F6** to execute it.

The content of the selected data is displayed on the console.

```
Starting job OutputStream at 17:31 19/10/2011.

[statistics] connecting to socket on port 4059
[statistics] connected
.-----+------------+-----------.
|           tLogRow_1            |
|=====+============+===========|
|id   |CustomerName|CustomerAge|
|=====+============+===========|
|10001|Warren      |67         |
|10002|Woodrow     |68         |
|10003|Grover      |77         |
|10004|Abraham     |74         |
|10005|Chester     |78         |
|10006|Calvin      |63         |
|10007|Zachary     |53         |
|10008|Chester     |36         |
|10009|Chester     |60         |
|10010|Woodrow     |57         |
'-----+------------+-----------'
[statistics] disconnected
Job OutputStream ended at 17:31 19/10/2011. [exit code=0]
```

The selected data is also output to the specified local file `customerselection.txt`.

customerselection.txt

```
1  id;CustomerName;CustomerAge
2  10001;Warren;67
3  10002;Woodrow;68
4  10003;Grover;77
5  10004;Abraham;74
6  10005;Chester;78
7  10006;Calvin;63
8  10007;Zachary;53
9  10008;Chester;36
10 10009;Chester;60
11 10010;Woodrow;57
```

Using the Implicit Context Load feature

Job parameterization based on context variables enables you to orchestrate and execute your Jobs in different contexts or environments. You can define the values of your context variables when creating them, or load your context parameters dynamically, either explicitly or implicitly, when your Jobs are executed.

The use case below describes how to use the Implicit Context Load feature of your Talend Studio to load context parameters dynamically at the time of Job execution.

The Job in this use case is composed of only two components. It will read employees data stored in two MySQL databases, one for testing and the other for production purposes. The connection parameters for accessing these two databases are stored in another MySQL database. When executed, the Job loads these connection parameters dynamically to connect to the two databases.

Creating the Job and defining context variables

Before you begin

Create two tables named `db_testing` and `db_production` respectively in a MySQL database named `db_connections`, to hold the connection parameters for accessing the above mentioned databases, `testing` and `production`. Each table should contain only two columns: `key` and `value`, both of type VARCHAR. Below is an example of the content of the database tables:

db_testing:

key	**value**
host	localhost
port	3306
username	root
password	talend
database	testing

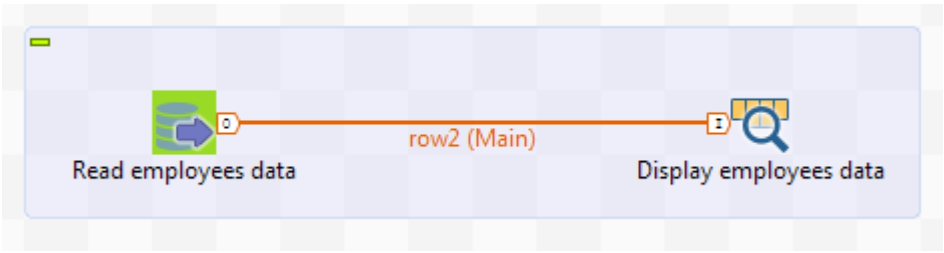
db_production:

key	**value**
host	localhost
port	3306
username	root
password	talend
database	production

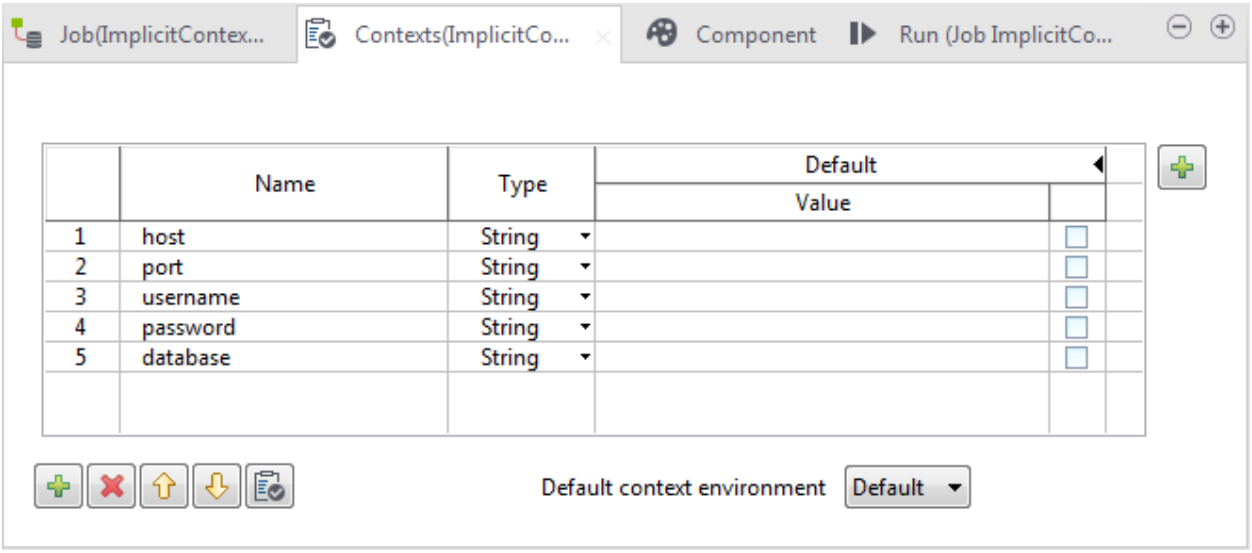
You can create these database tables using another Talend Job that contains **tFixedFlowInput** and **tMysqlOutput** components.

Procedure

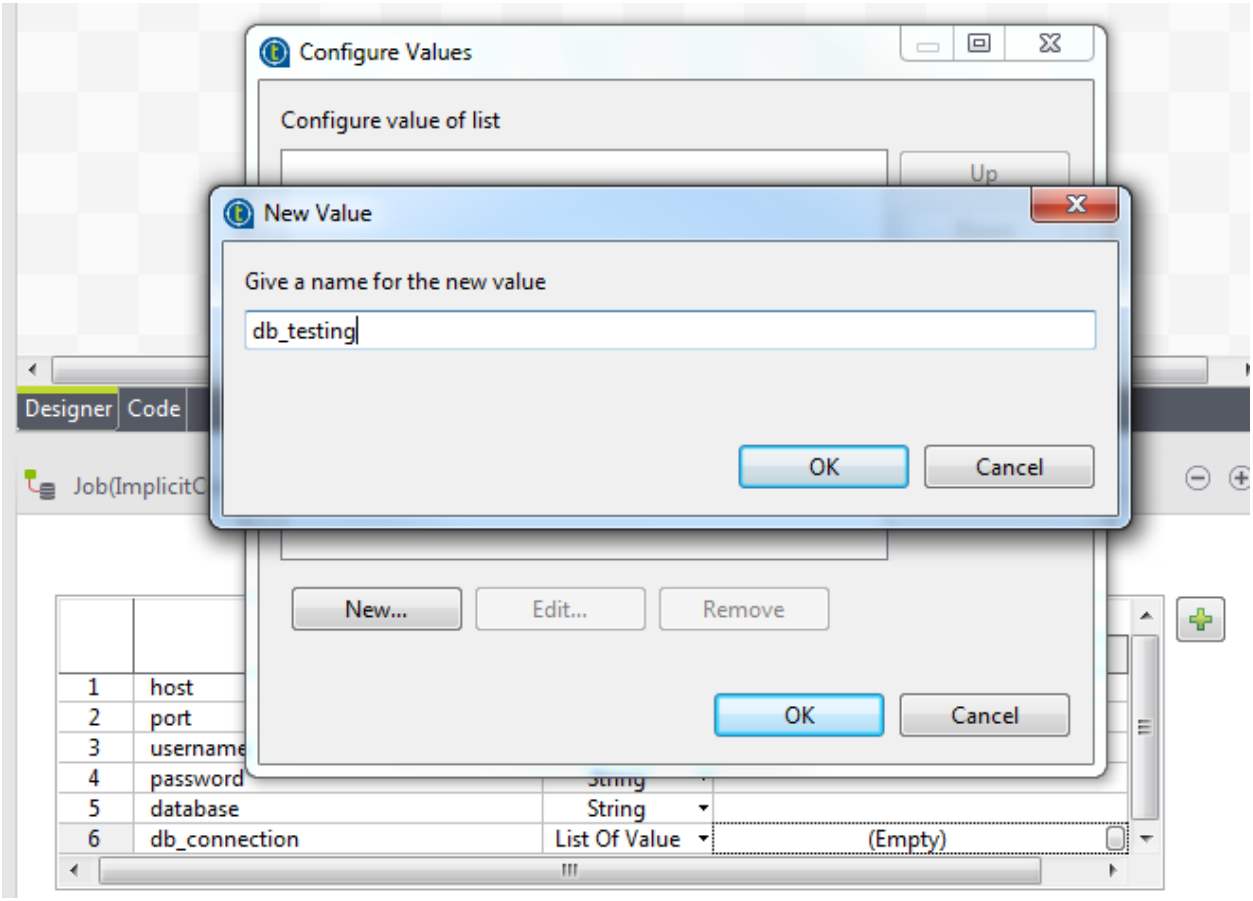
1. Create a Job and add a **tMysqlInput** component and a **tLogRow** component onto the design workspace, and link them using a **Row** > **Main** connection.



2. Select the **Contexts** view of the Job, and click the **[+]** button at the bottom of the view to add five rows in the table to define the following variables, all of type **String**, without defining their values, which will be loaded dynamically at Job execution: `host`, `port`, `username`, `password`, and `database`.



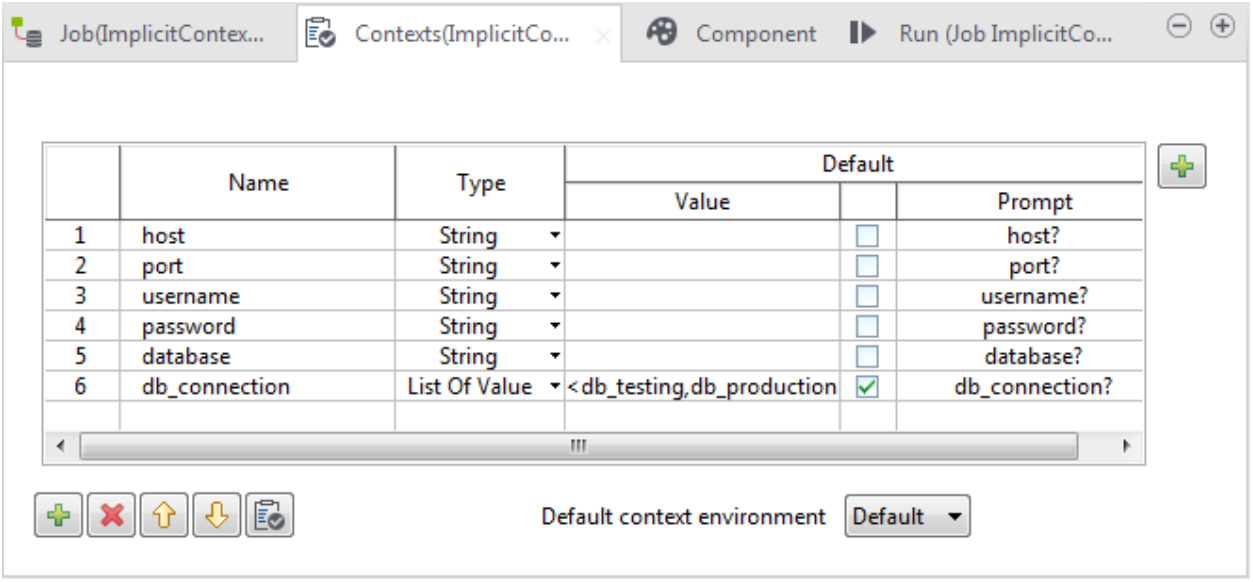
3. Now create another variable named `db_connection` of type **List Of Value**.
4. Click in the **Value** field of the newly created variable and click the button that appears to open the **Configure Values** dialog box, and click **New...** to open the **New Value** dialog box. Enter the name of one of the database tables holding the database connection details and click **OK**.



5. Click **New...** again to define the other table holding the database connection details. When done, click **OK** to close the **Configure Values** dialog box.

   Now the variable `db_connection` has a list of values `db_testing` and `db_production`, which are the database tables to load the connection parameters from.

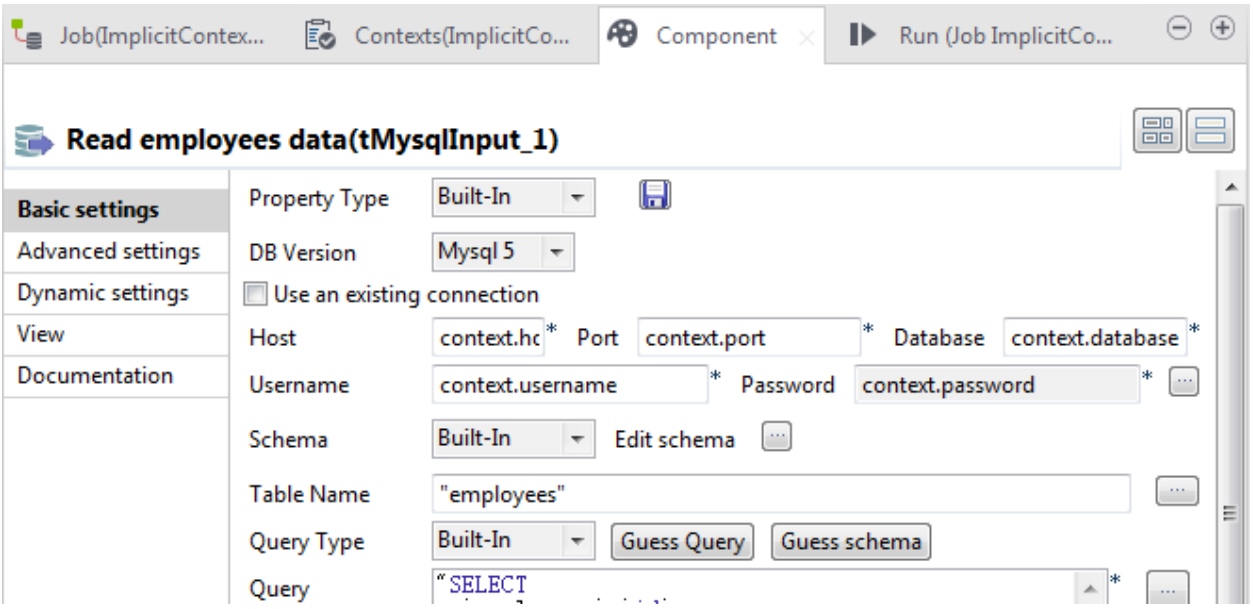
6. Select the **Prompt** check box next to the **Value** field of the `db_connection` variable to show the **Prompt** fields and enter the prompt message to be displayed at the execution time.



Configuring the components

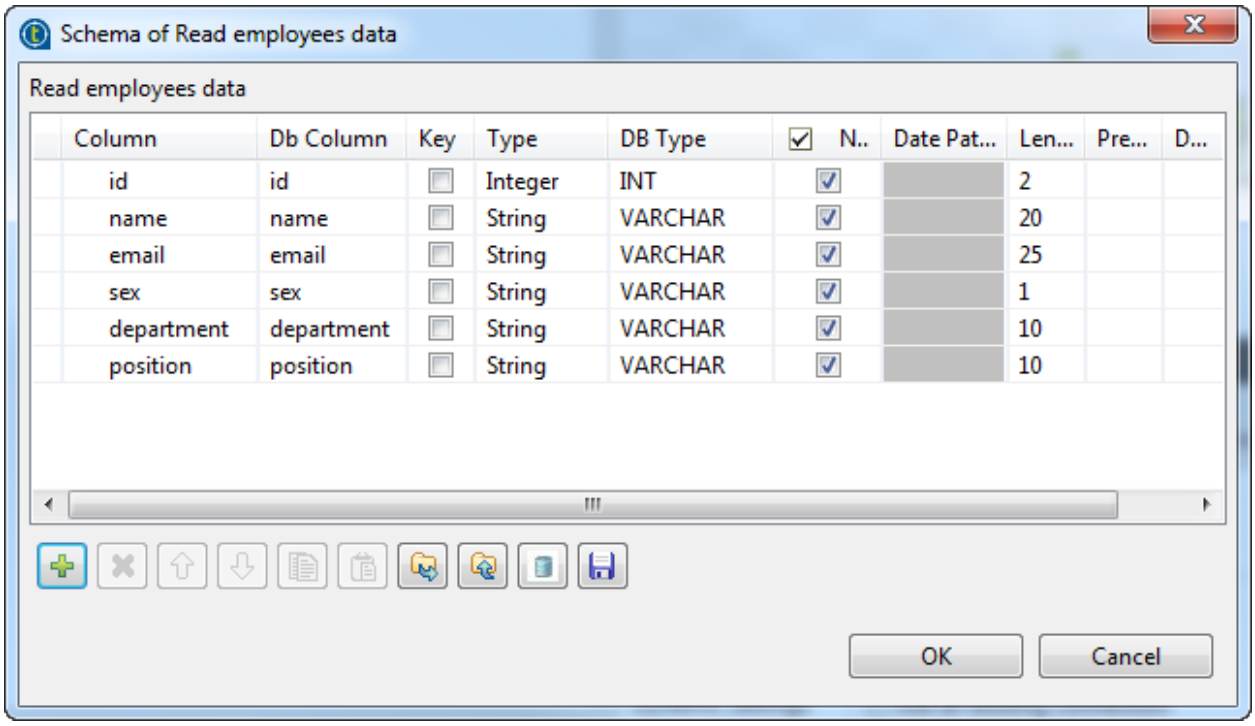
Procedure

1. Then double-click to open the **tMysqlInput** component **Basic settings** view.
2. Fill the **Host**, **Port**, **Database**, **Username**, **Password**, and **Table Name** fields with the relevant variables defined in the **Contexts** tab view: `context.host`, `context.port`, `context.database`, `context.username`, and `context.password` respectively in this example.



3. Fill the **Table Name** field with `employees`, which is the name of the table holding employees information in both databases in our example.
4. Then fill in the **Schema** information. If you stored the schema in the **Repository**, then you can retrieve it by selecting **Repository** and the relevant entry in the list.

   In this example, the schema of both the database tables to read is made of six columns: `id` (INT, 2 characters long), `name` (VARCHAR, 20 characters long), `email` (VARCHAR, 25 characters long), `sex` (VARCHAR, 1 characters long), `department` (VARCHAR, 10 characters long), and `position` (VARCHAR, 10 characters long).



5. Click **Guess Query** to retrieve all the table columns, which will be displayed on the **Run** tab, through the **tLogRow** component.
6. In the **Basic settings** view of the **tLogRow** component, select the **Table** option to display data records in the form of a table.

Configuring the Implicit Context Load feature

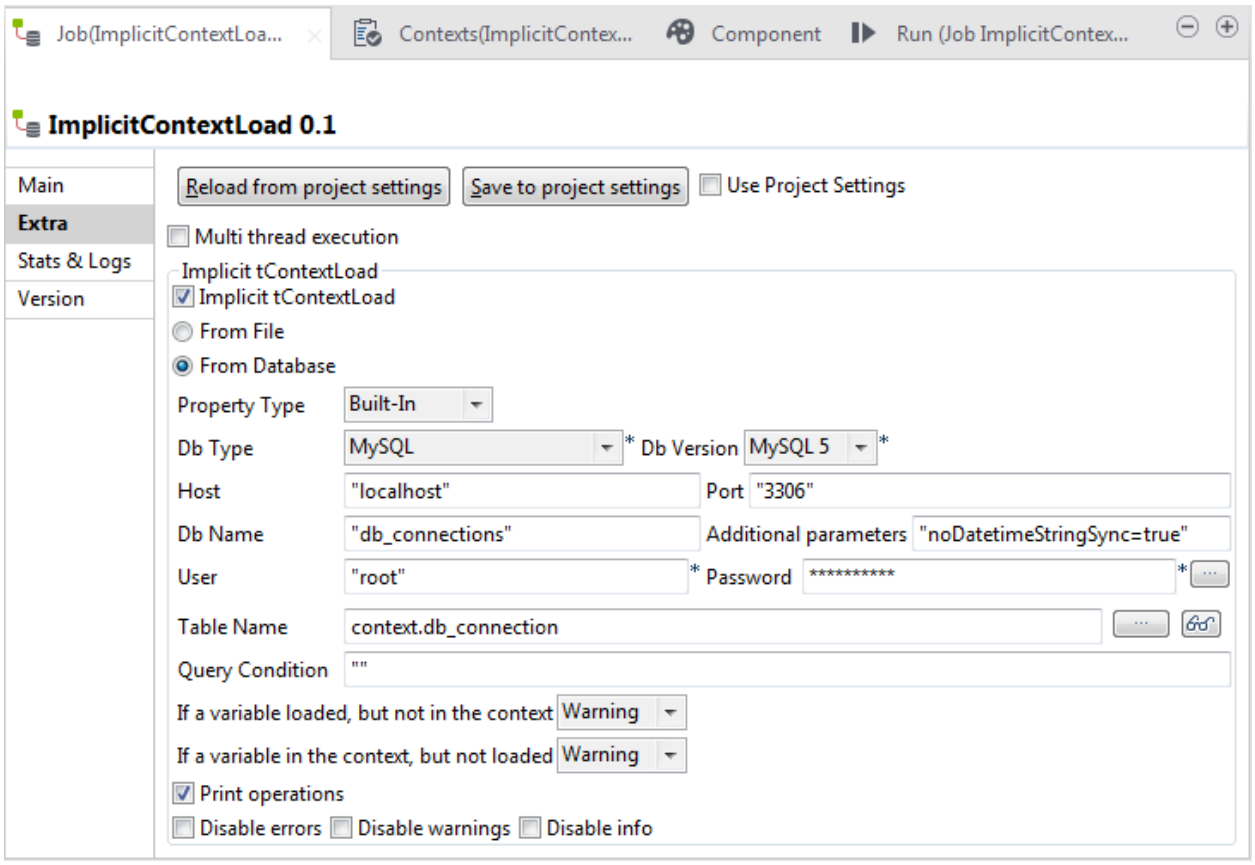
You can configure the Implicit Context Load feature either in Project Settings so that it can be used across Jobs within the project, or in the Job view for a particular Job.

The following example shows how to configure the Implicit Context Load feature in the **Job** view for this particular Job. If you want to configure the feature to be reused across different Jobs, select **File** > **Edit Project properties** from the menu bar to open the **Project Settings** dialog box, go to **Job Settings** > **Implicit context load**, select the **Implicit tContextLoad** check box, and set the parameters following steps 2 through 6 below. Then in the **Job** view, select the **Use Project Settings** check box to apply the settings to the Job.

Procedure

1. From the **Job** view, select the **Extra** vertical tab, and select the **Implicit tContextLoad** check box to enable context loading without using the **tContextLoad** component explicitly in the Job.
2. Select the source to load context parameters from. A context source can be a two-column flat file or a two-column database table. In this use case the database connection details are stored in database tables, so select the **From Database** option.
3. Define the database connection details just like defining the basic settings of a database input component.

   In this example, all the connection parameters are used just for this particular Job, so select **Built-In** from the **Property Type** list and fill in the connection details manually.



4. Fill the **Table Name** field with the context variable named `db_connection` defined in the **Contexts** view of the Job so that we will be able to choose the database table to load context parameters from dynamically at Job execution.
5. As we will fetch all the connection details from the database tables unconditionally, leave the **Query Condition** field blank.
6. Select the **Print operations** check box to list the context parameters loaded at Job execution.

Executing the Job

Procedure

1. Press **Ctrl+S** to save the Job, and press **F6** to run the Job.
2. A dialog box pops up asking you to select a database. Select a database and click **OK**.

   The loaded context parameters and the content of the "employees" table of the selected database are displayed on the **Run** console.
3. Now press **F6** to launch the Job again and select the other database when prompted.

   The loaded context parameters and the content of the "employees" table of the other database are displayed on the **Run** console.

Using the Multi-thread Execution feature to run Jobs in parallel

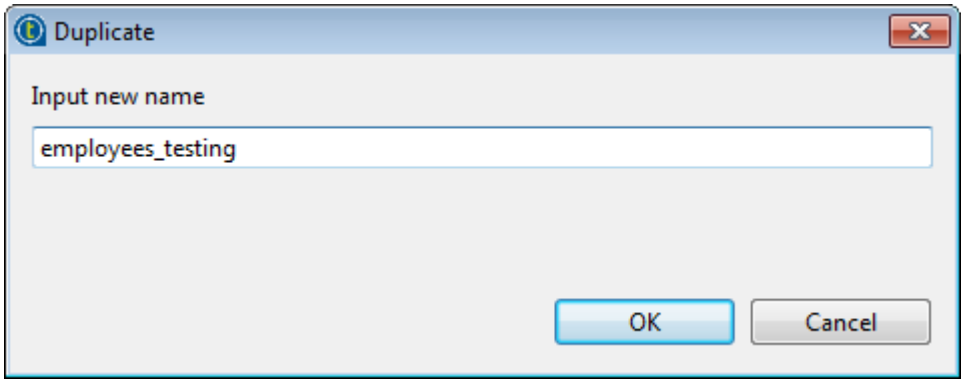
Based on the previous use case Using the output stream feature on page 13, this use case give an example of how to use the Multi-thread Execution feature to run two Jobs in parallel to display the employees information in both the testing and production environments at the same time. When handling large data volumes, this feature can significantly optimize the Job execution performance of the Talend Studio.

For more information on the multi-thread execution feature, see the *How to execute multiple Subjobs in parallel* section of the Talend Studio User Guide at https://help.talend.com.

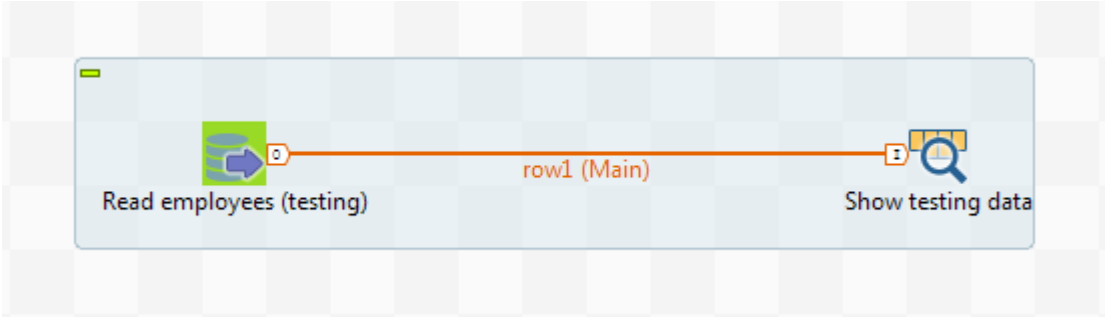
Preparing Jobs to read employees data in different contexts

Procedure

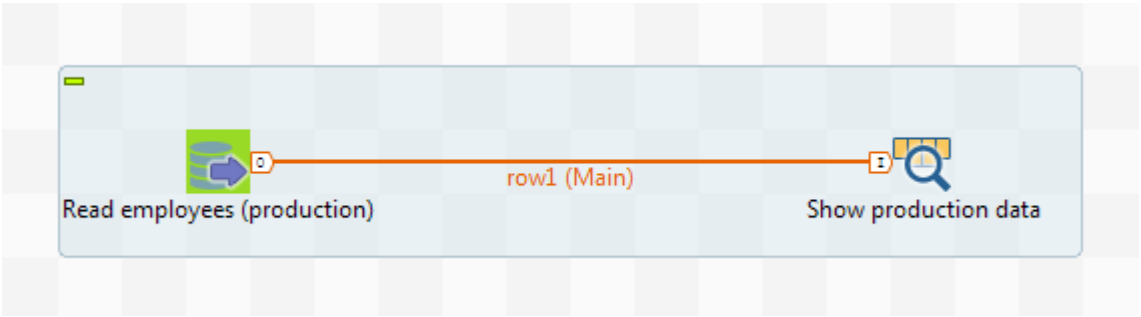
1. In the **Repository** tree view, right-click the Job created in the use case Using the output stream feature on page 13 and select **Duplicate** from the context menu. Then, in the **[Duplicate]** dialog box enter a new name for the Job, `employees_testing` in this example, and click **OK**.



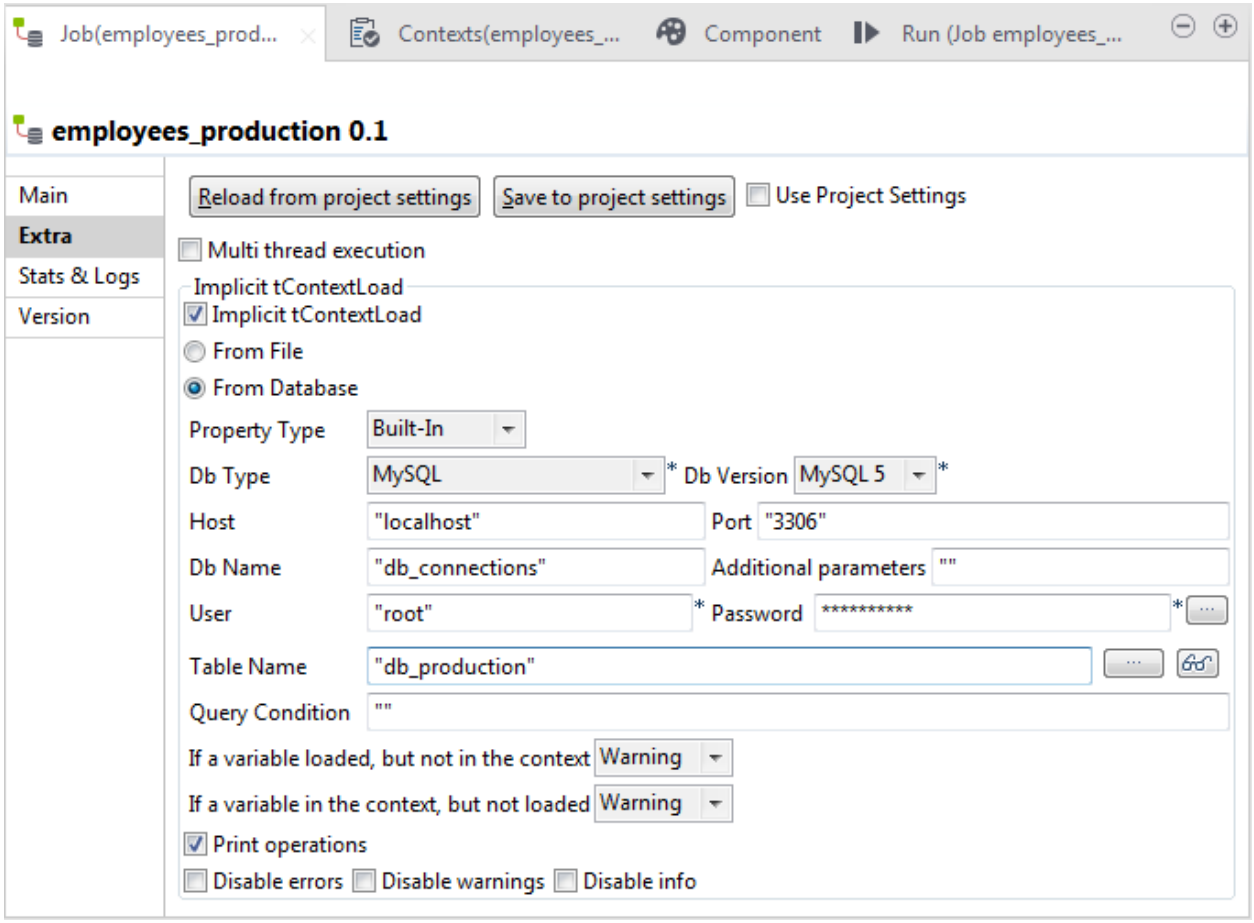
2. Open the new Job, and label the components to better reflect their roles.



3. Create another Job named `employees_production` by repeating the steps above.



4. In the **Contexts** view of both Jobs, remove the **db_connection** variable.
5. On **Extra** tab of the **Job** view of the Job **employees_testing**, fill the **Table Name** field of database settings with `db_testing`; on the **Extra** tab of the **Job** view of the Job **employees_production**, fill the **Table Name** field with `db_production`.

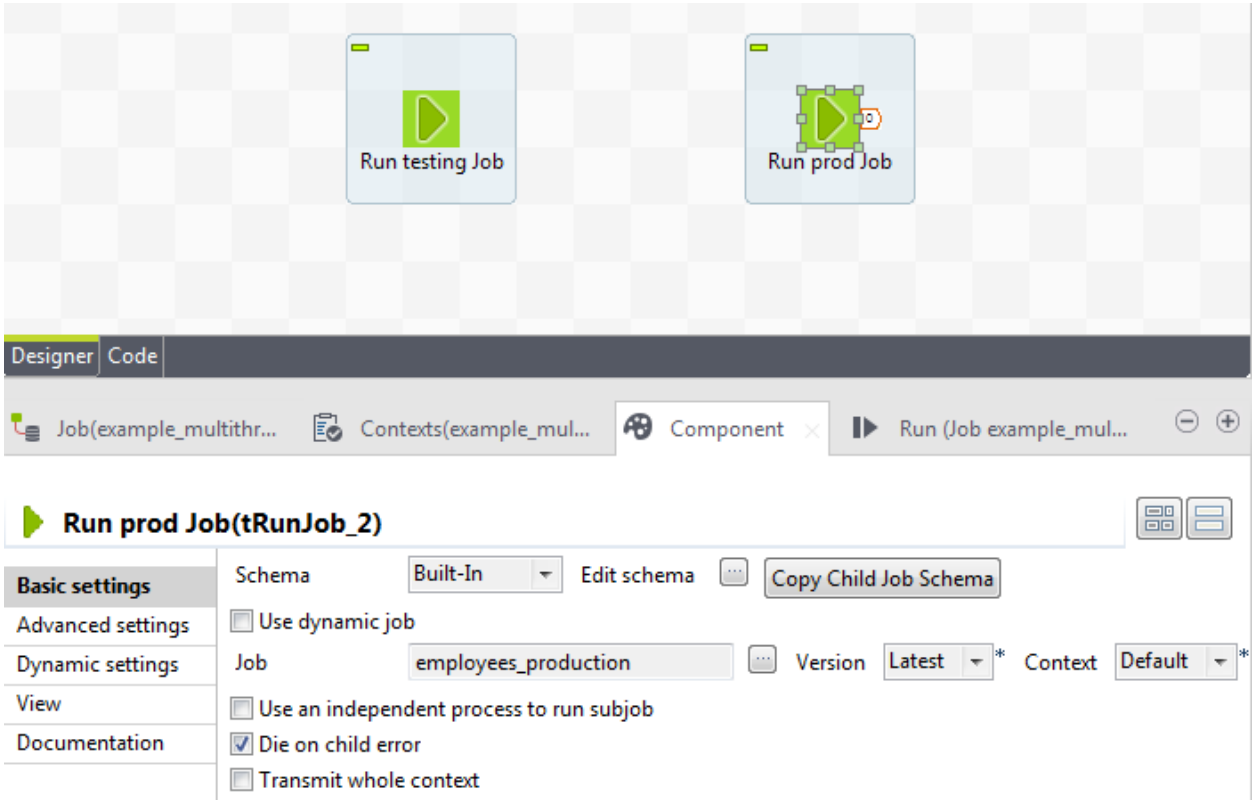


Set up a parent Job to run the Jobs in parallel

Procedure

1. Create a new Job and add two **tRunJob** components on the design workspace, and label the components to better reflect their roles.

2. In the **Component** view of the first **tRunJob** component, click the **[...]** button next to the **Job** field and specify the Job it will run, `employees_testing` in this example.
3. Configure the other **tRunJob** component to run the other Job, `employees_production`.



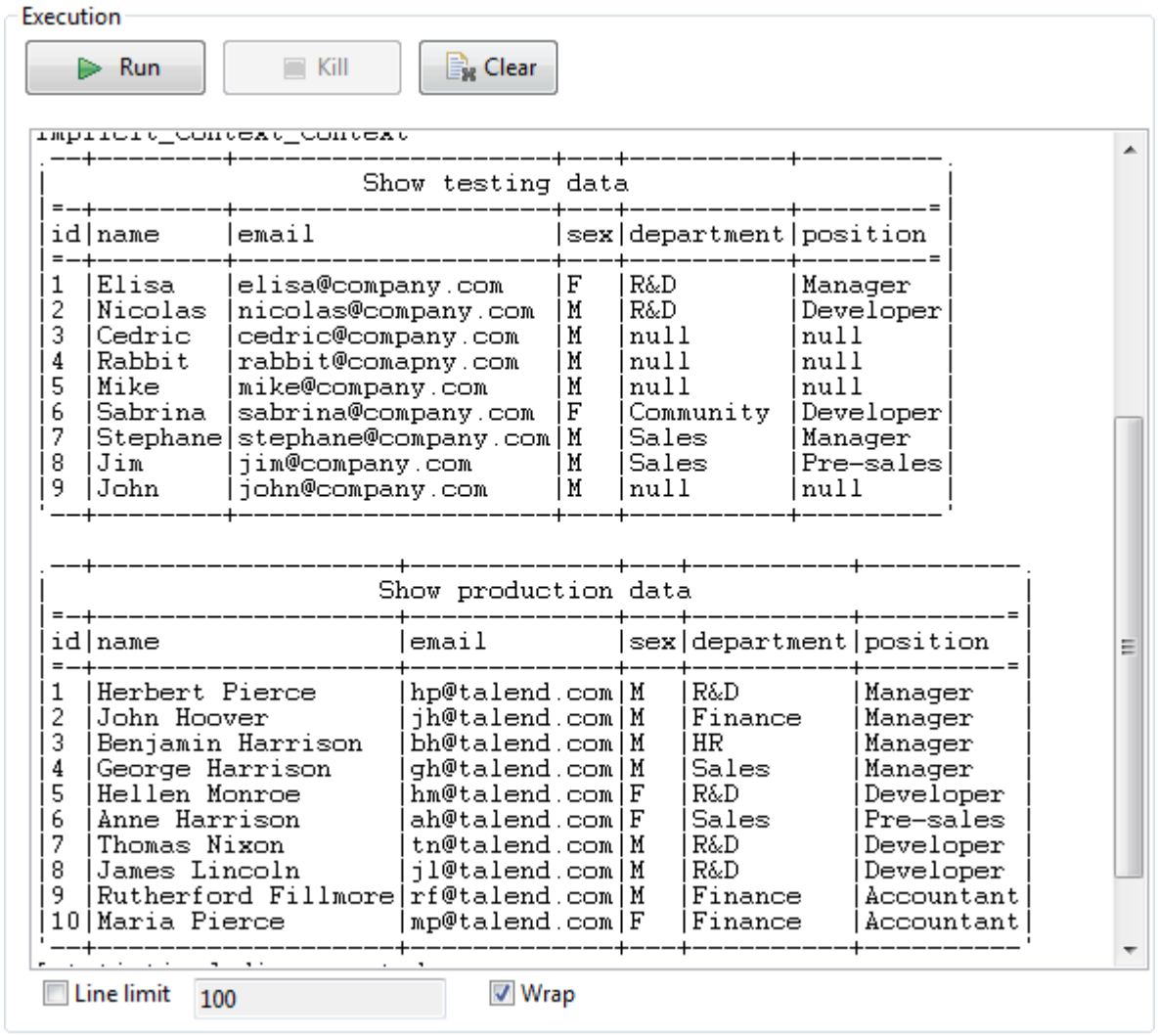
4. On the **Extra** tab of the **Job** view, select the **Multi thread execution** check box to activate the Multi-thread Execution feature.

Executing the Jobs

Procedure

1. Save each Job by pressing **Ctrl+S**.
2. In the parent Job, press **F6** of click **Run** on the **Run** view to start execution of the child Jobs.

   The child Jobs are executed in parallel, reading employees data from both databases and displaying the data on the console.


Execution
Run
Kill
Clear
Show testing data
id|name|email|sex|department|position
1|Elisa|elisa@company.com|F|R&D|Manager
2|Nicolas|nicolas@company.com|M|R&D|Developer
3|Cedric|cedric@company.com|M|null|null
4|Rabbit|rabbit@comapny.com|M|null|null
5|Mike|mike@company.com|M|null|null
6|Sabrina|sabrina@company.com|F|Community|Developer
7|Stephane|stephane@company.com|M|Sales|Manager
8|Jim|jim@company.com|M|Sales|Pre-sales
9|John|john@company.com|M|null|null
Show production data
id|name|email|sex|department|position
1|Herbert Pierce|hp@talend.com|M|R&D|Manager
2|John Hoover|jh@talend.com|M|Finance|Manager
3|Benjamin Harrison|bh@talend.com|M|HR|Manager
4|George Harrison|gh@talend.com|M|Sales|Manager
5|Hellen Monroe|hm@talend.com|F|R&D|Developer
6|Anne Harrison|ah@talend.com|F|Sales|Pre-sales
7|Thomas Nixon|tn@talend.com|M|R&D|Developer
8|James Lincoln|jl@talend.com|M|R&D|Developer
9|Rutherford Fillmore|rf@talend.com|M|Finance|Accountant
10|Maria Pierce|mp@talend.com|F|Finance|Accountant
Line limit
100
Wrap