talend*
*open data solutions

Talend Open Studio Components 3.X

Reference Guide

Version 3.2_a

Adapted for **Talend Open Studio** v3.2.x. Supersedes previous Reference Guide releases.

Copyright

Talend Open Studio Components Reference Guide i

Preface

Purpose

This Reference Guide explains in detail the major components you can find in each of the different groups in the **Palette** of **Talend Open Studio**.

Information presented in this document applies to **Talend Open Studio** releases beginning with **3.2.x**.

Audience

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

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

Typographical conventions

This guide uses the following typographical conventions:

- text in **bold:** window and dialog box buttons and fields, keyboard keys, menus, and menu options,
- text in **[bold]:** window, wizard, and dialog box titles,
- text in `courier`: system parameters typed in by the user,
- text in *italics*: file, schema, column, row, and variable names referred to in all use cases, and also names of the fields in the Basic and Advanced setting views referred to in the property table for each component,
- In the properties section for every component, the  or  icon indicates whether the component is available in Java and/or in Perl.
- 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 need to be aware of or pay special attention to.

History of changes

The below table lists the changes made in the 3.x release of the **Talend Open Studio** Reference Guide.

Version	Date	History of Change
v3.0_c	17/12/2008	Updates in **Talend Open Studio** Reference Guide include: -Updates in the DB input and output component properties regarding the Advanced settings view. -Updates in the DB SCD components regarding the SCD editor.
v3.1_a	30/04/2009	Updates in **Talend Open Studio** Reference Guide include: -Updates in the *Row component properties regarding the Advanced settings view -Updates in the DB components properties to adjust them to the basic settings view -New components: **tAlfrescoOutput**, **tMysqlTableList**, **tFileInputEBCDIC, tFileOutputEBCDIC**, **tFileInputMSDelimited**, **tFileInputMSXML**, **tFileInputPositional**, **tFileOutputPositional**, **tFileOutputMSXML**. -Changes in **tMap** interface and new **tMap** scenario
v3.1_b	13/05/2009	No updates in **Talend Open Studio** Reference Guide.
v3.1_c	18/06/2009	Updates in **Talend Open Studio** Reference Guide include: -New components in the Business, File, Internet, and XML families.
v3.2_a	20/10/2009	Updates in **Talend Open Studio** Reference Guide include: -New components in the File, Internet, Processing, Database, Business intelligence and Data quality chapters. -Modifications in the settings and scenarios of many components to match the modifications in the GUI. -Modifications in tSAPInput + new scenario. -Modifications in tRunJob and its scenario.

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

Business Intelligence components

This chapter details the major components that you can find in **Business Intelligence** group of the **Palette** of **Talend Open Studio**.

The BI family groups connectors that cover needs such as reading or writing multidimensional or OLAP databases, outputting Jasper reports, tracking DB changes in slow changing dimension tables and so on.

tDB2SCD

tDB2SCD properties

Component family	Databases/DB2	
Function	**tDB2SCD** reflects and tracks changes in a dedicated DB2 SCD table.	
Purpose	**tDB2SCD** addresses Slowly Changing Dimension needs, reading regularly a source of data and logging the changes into a dedicated SCD table	
Basic settings	*Use an existing connection*	Select this check box and click the relevant DB connection component on the **Component list** to reuse the connection details you already defined.
	Property type	Either Built-in or Repository.
		Built-in: No property data stored centrally.
		Repository: Select the Repository file where properties are stored. The following fields are pre-filled in using fetched data.
	Host	Database server IP address.
	Port	Listening port number of DB server.
	Database	Name of the database.
	Table Schema	Name of the DB schema.
	Username and *Password*	DB user authentication data.
	Table	Name of the table to be written. Note that only one table can be written at a time.
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either built-in or remotely stored in the Repository.
		Built-in: The schema is created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	SCD Editor	The SCD editor helps to build and configure the data flow for slowly changing dimension outputs. For more information, see *SCD management methodologies on page 13*.
	Use memory saving Mode	Select this check box to maximize system performance.
Usage	This component is used as Output component. It requires an Input component and Row main link as input.	

Related scenarios

For related topics, see *Scenario: Tracking changes using Slowly Changing Dimensions (type 0 through type 3) on page 15*.

tIngresSCD

tIngresSCD Properties

Component family	Databases/Ingress	
Function	**tIngresSCD** reflects and tracks changes in a dedicated Ingres SCD table.	
Purpose	**tIngresSCD** addresses Slowly Changing Dimension needs, reading regularly a source of data and logging the changes into a dedicated SCD table	
Basic settings	*Property type*	Either Built-in or Repository.
		Built-in: No property data stored centrally.
		Repository: Select the Repository file where properties are stored. The fields to follow are pre-filled in using fetched data.
	Server	Database server IP address.
	Port	Listening port number of DB server.
	Database	Name of the database.
	Username and *Password*	DB user authentication data.
	Table	Name of the table to be written. Note that only one table can be written at a time.
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either built-in or remotely stored in the Repository.
		Built-in: The schema is created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	SCD Editor	The SCD editor helps to build and configure the data flow for slowly changing dimension outputs. For more information, see *SCD management methodologies on page 13*.
	Use memory saving Mode	Select this check box to maximize system performance.
Usage	This component is used as Output component. It requires an Input component and Row main link as input.	

Related scenario

For related scenarios, see **tMysqlSCD** *Scenario: Tracking changes using Slowly Changing Dimensions (type 0 through type 3) on page 15*.

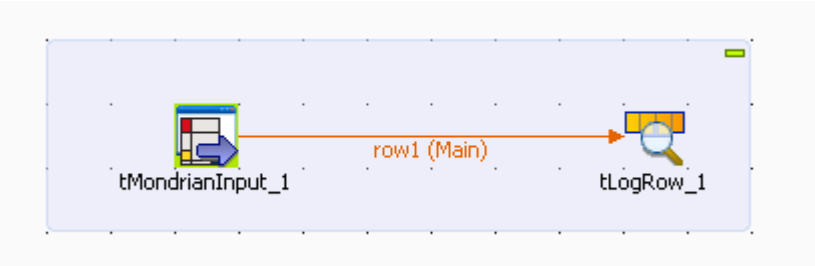
tMondrianInput

tMondrianInput Properties

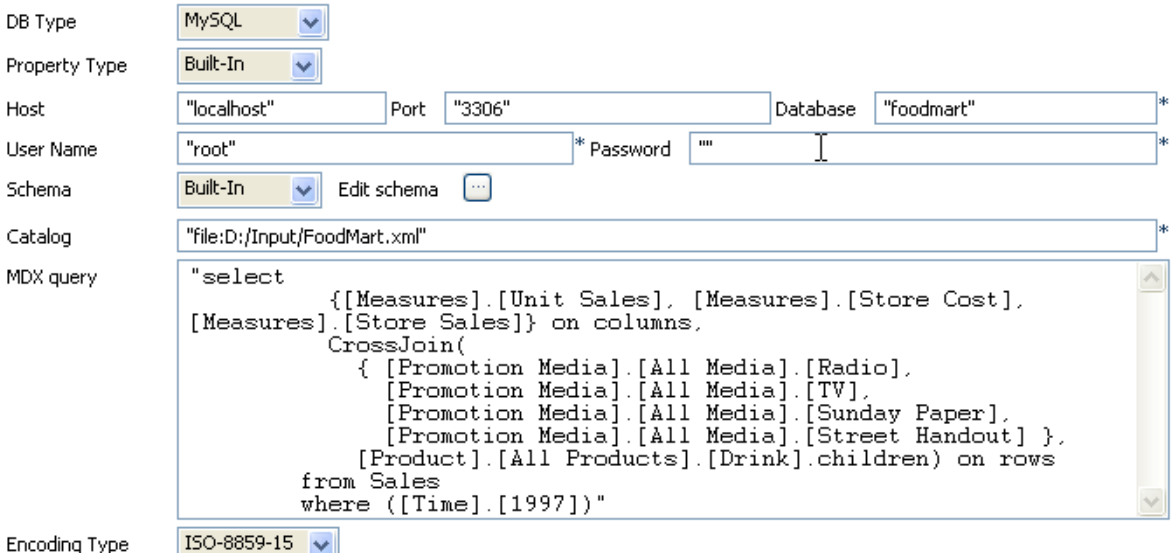
Component family	Business Intelligence/OLAP Cube	
Function	**tMondrianInput** reads data from relational databases and produces multidimensional data sets based on an MDX query.	
Purpose	**tMondrianInput** executes a multi-dimensional expression (MDX) query corresponding to the dataset structure and schema definition. Then it passes on the multidimensional dataset obtained to the next component via a Main row link.	
Basic settings	*DB type*	Select the relevant type of relational database
	Property type	Either Built-in or Repository.
		Built-in: No property data stored centrally.
		Repository: Select the Repository file where Properties are stored. The following fields are pre-filled in using fetched data.
	Host	Database server IP address.
	Port	Listening port number of DB server.
	Database	Name of the database.
	Username and *Password*	DB user authentication data.
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either built-in or remotely stored in the Repository.
		Built-in: The schema is created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	Catalog	Path to the catalog (structure of the data warehouse).
	MDX Query	Type in the MDX query paying particularly attention to properly sequence the fields in order to match the schema definition and the data warehouse structure.
	Encoding	Select the encoding from the list or select Custom and define it manually. This field is compulsory for DB data handling.
Usage	This component covers MDX queries for multi-dimensional datasets.	

Scenario: Cross-join tables

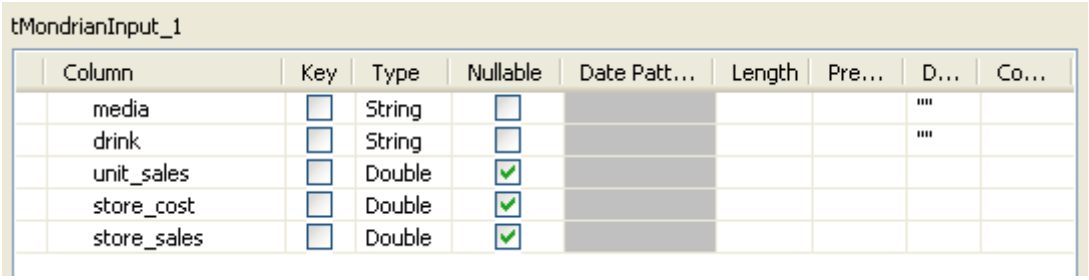
This Job extracts multi-dimensional datasets from relational database tables stored in a MySql base. The data are retrieved using a multidimensional expression (MDX query). Obviously you need to have to know the structure of your data, or at least have a structure description (catalog) as a reference for the dataset to be retrieved in the various dimensions.



- Drop **tMondrianInput** and **tLogRow** from the **Palette** to the design workspace.
- Link the Mondrian connector to the output component using a **Row Main** connection.
- Select the **tMondrianInput** component and select the **Component** view.



- In **DB type** field, select the relational database you are using with Mondrian.
- Select the relevant Repository entry as **Property type**, if you store your DB connection details centrally. In this example the properties are built-in.
- Fill out the details of connection to your DB: **Host**, **Port**, **Database** name, **User Name** and **Password**.
- Select the relevant **Schema** in the Repository if you store it centrally. In this example, the schema is to be set (built-in).



- The relational database we want to query contains five columns: *media*, *drink*, *unit_sales*, *store_cost* and *store_sales*.
- The query aims at retrieving the *unit_sales*, *store_cost* and *store_sales* figures for various *media* / *drink* using an MDX query such as in the example below:

CrossJoin Example 1

The current slicer is **1997**.

		Unit Sales	Store Cost	Store Sales
Radio	Alcoholic Beverages	75	70.40	168.62
	Beverages	97	75.70	186.03
	Dairy	54	36.75	89.03
TV	Alcoholic Beverages	76	70.99	182.38
	Beverages	188	167.00	419.14
	Dairy	68	45.19	119.55
Sunday Paper	Alcoholic Beverages	148	128.97	316.88
	Beverages	197	161.81	399.58
	Dairy	85	54.75	140.27
Street Handout	Alcoholic Beverages	158	121.14	294.55
	Beverages	270	201.28	520.55
	Dairy	84	50.26	128.32

- Back on the **Basic settings** tab of the **tMondrianInput** component, set the **Catalog** path to the data warehouse. This catalog describes the structure of the warehouse.
- Then type in the MDX query such as:

```
"select
  {[Measures].[Unit Sales], [Measures].[Store Cost],
[Measures].[Store Sales]} on columns,
  CrossJoin(
    { [Promotion Media].[All Media].[Radio],
      [Promotion Media].[All Media].[TV],
      [Promotion Media].[All Media].[Sunday Paper],
      [Promotion Media].[All Media].[Street Handout] },
    [Product].[All Products].[Drink].children) on rows
from Sales
where ([Time].[1997])"
```

- Eventually, select the **Encoding** type on the list.
- Select the **tLogRow** component and select the **Print header** check box to display the column names on the console.
- Then press **F6** to run the Job.

```
Starting job Mondrian at 15:31 08/01/2008.
media   drink   unit sales  store cost  store sales
Promotion Media].[All Media].[Radio]|[Product].[All Products].[Drink].[Alcoholic
everages]|75.0|70.3977|168.62
Promotion Media].[All Media].[Radio]|[Product].[All Products].[Drink].[Beverages]|97.0|75.7016|186.03
Promotion Media].[All Media].[Radio]|[Product].[All Products].[Drink].[Dairy]|54.0|36.7474|89.03
Promotion Media].[All Media].[TV]|[Product].[All Products].[Drink].[Alcoholic
everages]|76.0|70.9895|182.38
Promotion Media].[All Media].[TV]|[Product].[All Products].[Drink].[Beverages]|188.0|167.0025|419.14
Promotion Media].[All Media].[TV]|[Product].[All Products].[Drink].[Dairy]|68.0|45.1909|119.55
Promotion Media].[All Media].[Sunday Paper]|[Product].[All Products].[Drink].[Alcoholic
everages]|148.0|128.9736|316.88
Promotion Media].[All Media].[Sunday Paper]|[Product].[All Products].[Drink].
Beverages]|197.0|161.8067|399.58
Promotion Media].[All Media].[Sunday Paper]|[Product].[All Products].[Drink].
Dairy]|85.0|54.746|140.27
Promotion Media].[All Media].[Street Handout]|[Product].[All Products].[Drink].[Alcoholic
everages]|158.0|121.1394|294.55
Promotion Media].[All Media].[Street Handout]|[Product].[All Products].[Drink].
Beverages]|270.0|201.2816|520.55
Promotion Media].[All Media].[Street Handout]|[Product].[All Products].[Drink].
Dairy]|84.0|50.2604|128.32
Job Mondrian ended at 15:31 08/01/2008. [exit code=0]
```

The console shows the result of the *unit_sales*, *store_cost* and *store_sales* for each type of *Drink* (*Beverages*, *Dairy*, *Alcoholic beverages*) crossed with each media (*TV*, *Sunday Paper*, *Street handout*) as shown previously in a table form.

tMSSqlSCD

tMSSqlSCD Properties

Component family	Databases/MSSQL Server	
Function	**tMSSqlSCD** reflects and tracks changes in a dedicated MSSQL SCD table.	
Purpose	**tMSqlSCD** addresses Slowly Changing Dimension needs, reading regularly a source of data and logging the changes into a dedicated SCD table	
Basic settings	*Use an existing connection*	Select this check box and click the relevant DB connection component on the **Component list** to reuse the connection details you already defined.
	Property type	Either Built-in or Repository.
		Built-in: No property data stored centrally.
		Repository: Select the Repository file where Properties are stored. The following fields are pre-filled in using fetched data.
	Server	Database server IP address.
	Port	Listening port number of DB server.
	Database	Name of the database.
	Schema	Name of the DB schema.
	Username and *Password*	DB user authentication data.
	Table	Name of the table to be written. Note that only one table can be written at a time.
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either built-in or remotely stored in the Repository.
		Built-in: The schema is created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	SCD Editor	The SCD editor helps to build and configure the data flow for slowly changing dimension outputs. For more information, see *SCD management methodologies on page 13*.
	Use memory saving Mode	Select this check box to maximize system performance.
Usage	This component is used as Output component. It requires an Input component and Row main link as input.	

Related scenario

For related topics, see *Scenario: Tracking changes using Slowly Changing Dimensions (type 0 through type 3) on page 15*.

tMysqlSCD

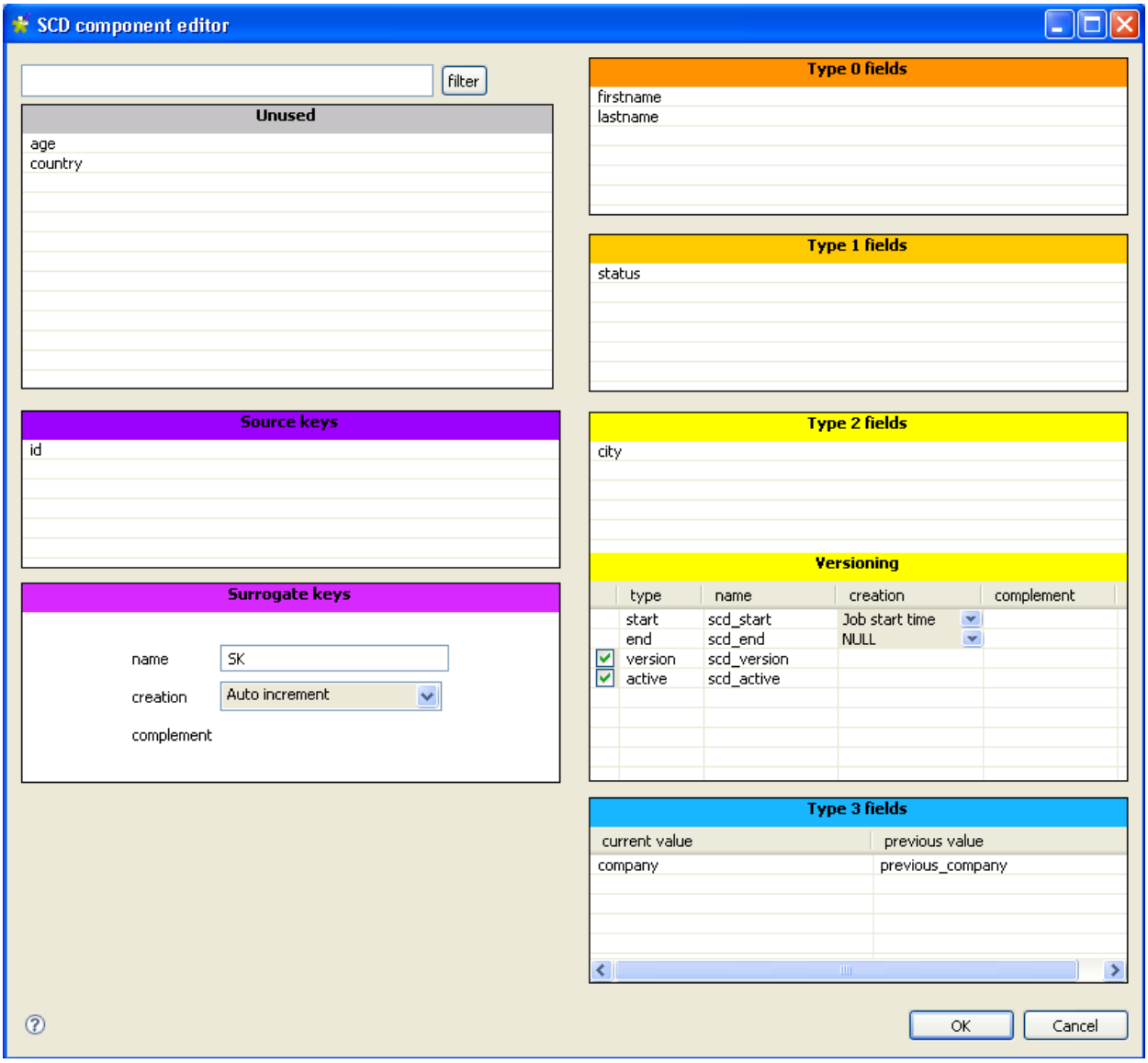
tMysqlSCD Properties

Component family	Databases/MySQL	
Function	**tMysqlSCD** reflects and tracks changes in a dedicated MySQL SCD table.	
Purpose	**tMysqlSCD** addresses Slowly Changing Dimension needs, reading regularly a source of data and logging the changes into a dedicated SCD table	
Basic settings	*Use an existing connection*	Select this check box and click the relevant DB connection component on the **Component list** to reuse the connection details you already defined.
	Property type	Either Built-in or Repository.
		Built-in: No property data stored centrally.
		Repository: Select the Repository file where properties are stored. The following fields are pre-filled in using fetched data.
	Host	Database server IP address.
	Port	Listening port number of DB server.
	Database	Name of the database.
	Username and *Password*	DB user authentication data.
	Table	Name of the table to be written. Note that only one table can be written at a time.
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either built-in or remotely stored in the Repository.
		Built-in: The schema is created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	SCD Editor	The SCD editor helps to build and configure the data flow for slowly changing dimension outputs. For more information, see *SCD management methodologies on page 13*.
	Use memory saving Mode	Select this check box to maximize system performance.
Usage	This component is used as Output component. It requires an Input component and Row main link as input.	

SCD management methodologies

Slowly Changing Dimensions (SCD)s are dimensions that have data that slowly changes. The SCD editor offers the simplest method of building the data flow for the SCD outputs. In the SCD editor, you can map columns, select surrogate key columns, and set column change attributes through combining SCD types.

The below figure illustrates an example of the SCD editor.



SCD keys

You must choose one or more source key columns from the incoming data to ensure its unicity.
You must set one surrogate key column in the dimension table and map it to an input column in the source table. The value of the surrogate key links a record in the source to a record in the dimension table. The editor uses this mapping to locate the record in the dimension table and to determine whether a record is new or changing. The surrogate key is typically the primary key in the source, but it can be an alternate key as long as it uniquely identifies a record and its value does not change.

Source keys: Drag one or more columns from the **Unused** panel to the **Source keys** panel to be used as the key (s) that ensure the unicity of the incoming data.

Surrogate keys: Set the column where the generated surrogate key will be stored. A surrogate key can be generated based on a method selected on the **Creation** list.

Creation: Select any of the below methods to be used for the key generation:

- **Auto increment**: auto-incremental key.
- **Input field**: key is provided in an input field.
- **Routine**: you can access the basic functions through Ctrl+ Space bar combination.
- **Table max +1**: the maximum value from the SCD table is incremented to create a surrogate key.

Combining SCD types

The Slowly Changing Dimensions support four types of changes: **Type 0** through **Type 3**. You can apply any of the SCD types to any column in a source table by a simple drag-and-drop operation.

- **Type 0**: is not used frequently. Some dimension data may be overwritten and other may stay unchanged over time. This is most appropriate when no effort has been made to deal with the changing dimension issues.
- **Type 1**: no history is kept in the database. New data overwrites old data. Use this type if tracking changes is not necessary. this is most appropriate when correcting certain typos, for example the spelling of a name.

- **Type2**: the whole history is stored in the database. This type tracks historical data by inserting a new record in the dimensional table with a separate key each time a change is made. This is most appropriate to track updates, for example.

  SCD **Type 2** principle lies in the fact that a new record is added to the SCD table when changes are detected on the columns defined. Note that although several changes may be made to the same record on various columns defined as SCD **Type 2**, only one additional line tracks these changes in the SCD table.

  The SCD schema in this type should include SCD-specific extra columns that hold standard log information such as:
  -**start**: adds a column to your SCD schema to hold the start date. You can select one of the input schema columns as a start date in the SCD table.
  -**end**: adds a column to your SCD schema to hold the end date value for a record. When the record is currently active, the end date is **NULL** or you can select **Fixed Year Value** and fill in a fictive year to avoid having a null value in the end date field.
  -**version**: adds a column to your SCD schema to hold the version number of the record.
  -**active**: adds a column to your SCD schema to hold the **true** or **false** status value. this column helps to easily spot the active record.

- **Type 3**: only the information about a previous value of a dimension is written into the database. This type tracks changes using separate columns. This is most appropriate to track only the previous value of a changing column.

Scenario: Tracking changes using Slowly Changing Dimensions (type 0 through type 3)

This five-component Java scenario describes a Job that tracks changes in four of the columns in a source delimited file and writes changes and the history of changes in an SCD table.

The source delimited file contains various personal details including *firstname, lastname,address, city, company, age*, and *status*. An *id* column helps ensuring the unicity of the data.

	A	B	C	D	E	F
1	id;firstname;lastname;address;city;company;age;status					
2	1;Janet;Anderson;511 Maple Ave. Apt. 1B;Agoura Hills;Adecco;30;married					
3	2;Harold;Smith;662 Lyons Circle;Alameda;Adidas;33;married					
4	3;Adam;Brown;220 Vine Ave.;Albany;Bridgestone;28;single					
5	4;Martin;Clinton;770 Exmoor Rd.;Alhambra;Cutco;25;single					
6	5;Sue;White;1486 Oakwood;Covina;Black&White;35;married					

We want any change in the marital status to overwrite the existing old status record. This type of change is equivalent to an SCD **Type 1**.

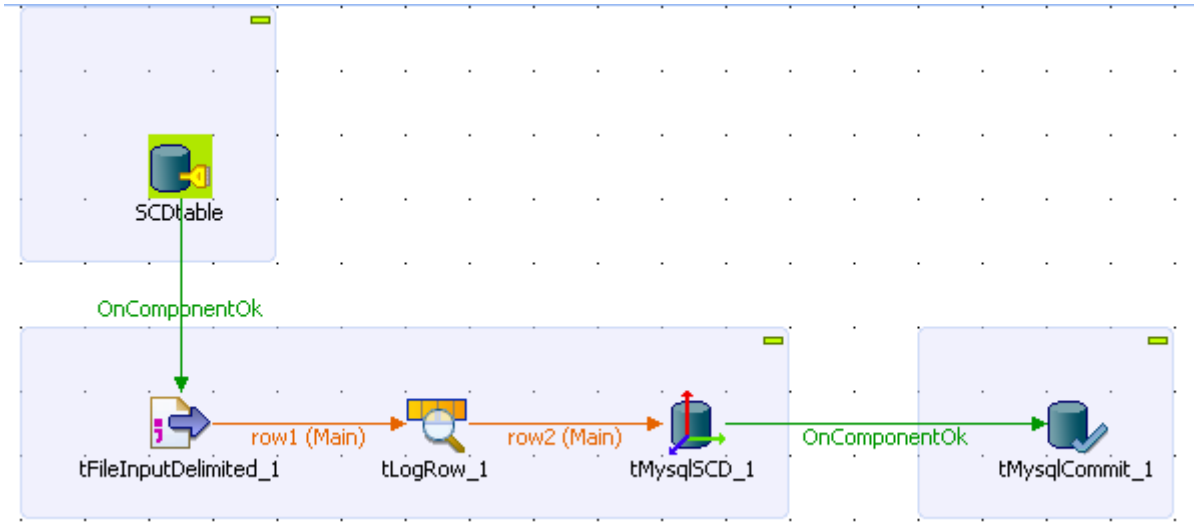
We want to insert a new record in the dimensional table with a separate key each time a person changes his/her company. This type of change is equivalent to an SCD **Type 2**.

We want to track only the previous city and previous address of a person. This type of change is equivalent to an SCD **Type 3**.

To realize this kind of scenario, it is better to divide it into three main steps: defining the main flow of the Job, setting up the SCD editor, and finally creating the relevant SCD table in the database.

Step 1: Defining the main flow of the job:

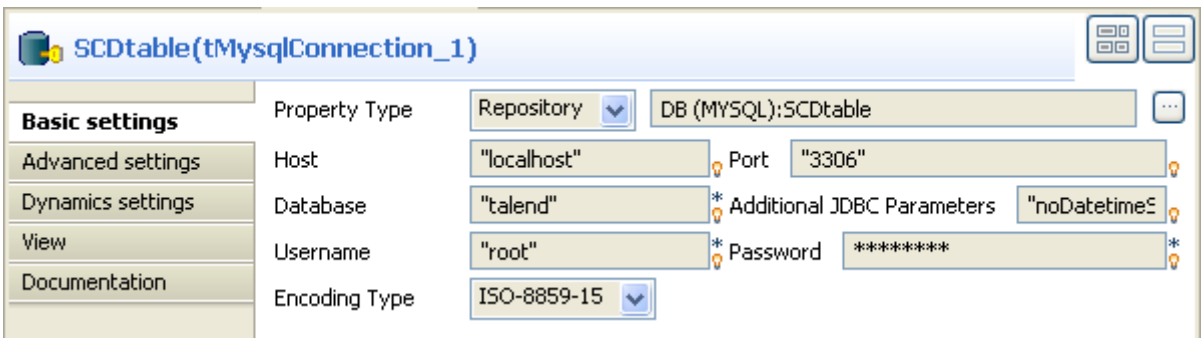
- Drop the following components from the **Palette** onto the design workspace: **tMysqlConnection**, **tFileInputDelimited**, **tLogRow**, **tMysqlSCD**, **tMysqlCommit**.
- Connect **tFileInputDelimited**, **tLogRow**, and **tMysqlSCD** using the **Row Main** link. This is the main flow of your Job.
- Connect **tMysqlConnection** to **tFileInputDelimited** and **tMysqlSCD** to **tMysqlCommit** using the **OnComponntOk** trigger.



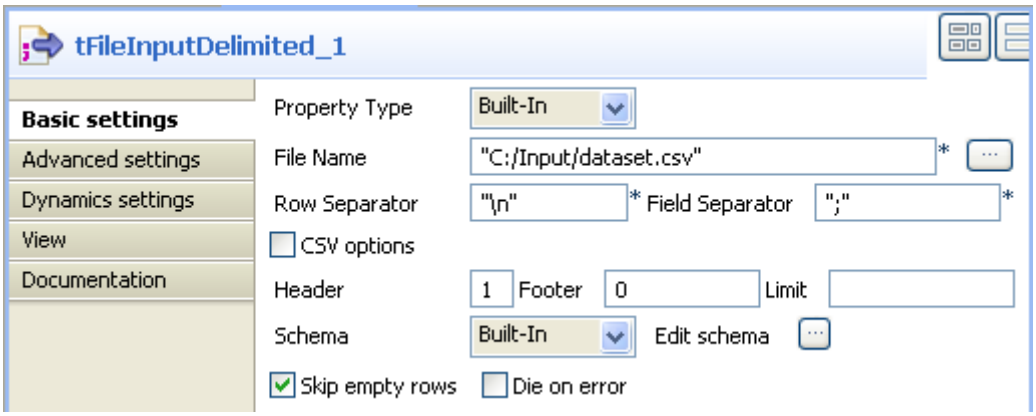
- In the design workspace, double-click **tMysqlConnection** to display its **Basic settings** view and set the database connection details manually. The **tMysqlConnection** component should be used to avoid setting several times the same DB connection when multiple DB components are used.

If you have already stored the connection details locally in the **Repository**, drop the needed metadata item to the design workspace and the database connection detail will automatically display in the relevant fields. For more information about Metadata, see *Defining Metadata items* of **Talend Open Studio** User Guide.

In this scenario, we want to connect to the SCD table where all changes in the source delimited file will be tracked down.



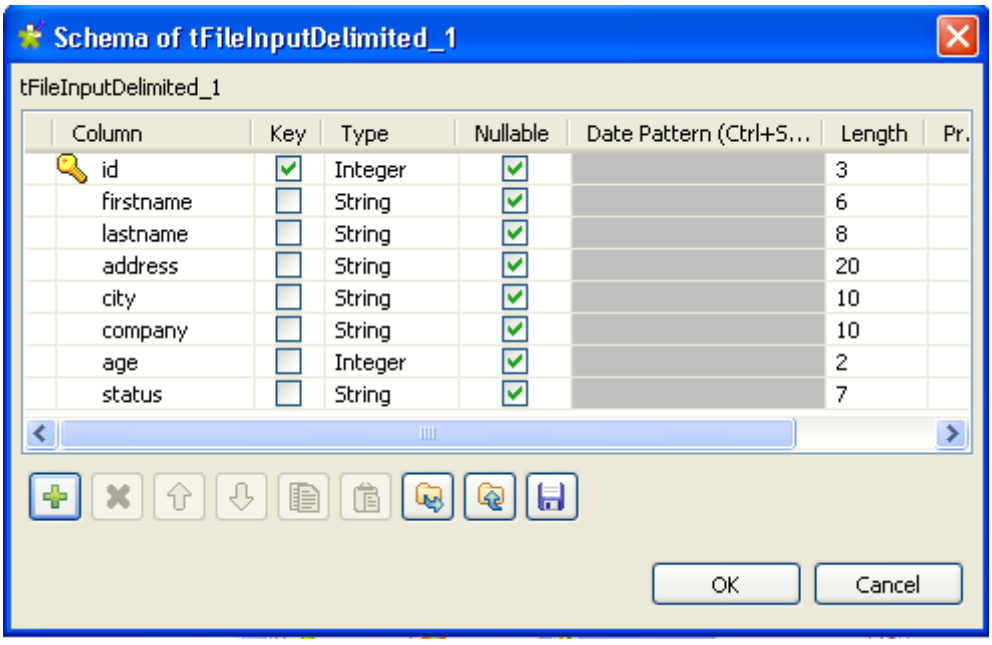
- In the design workspace, double-click **tFileInputDelimited** to display its **Basic settings** view.



- Click the three-dot [...] button next to the **File Name** field to select the path to the source delimited file, *dataset*, that contains the personal details.
- Define the row and field separators used in the source file.

The **File Name**, **Row separator**, and **Field separators** are mandatory.

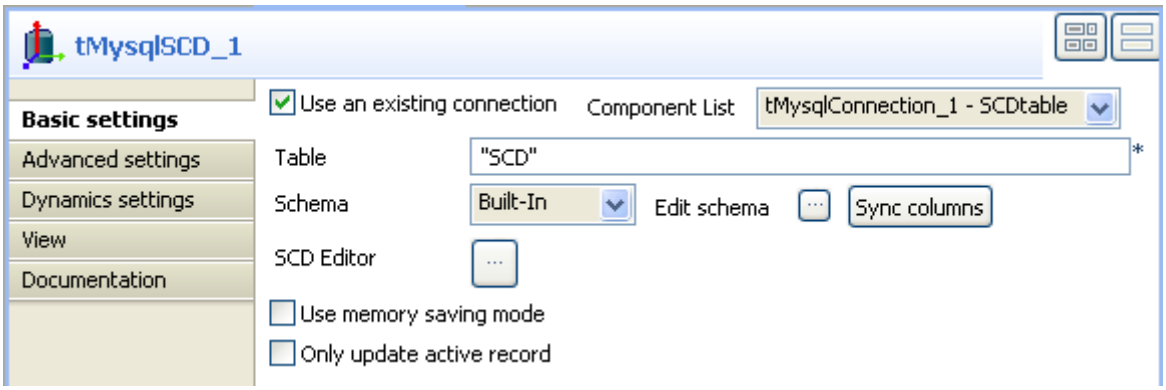
- If needed, set **Header**, **Footer**, and **Limit**.
  In this scenario, set **Header** to 1. Footer and limit for the number of processed rows are not set.
- Click **Edit schema** to describe the data structure of the source delimited file.



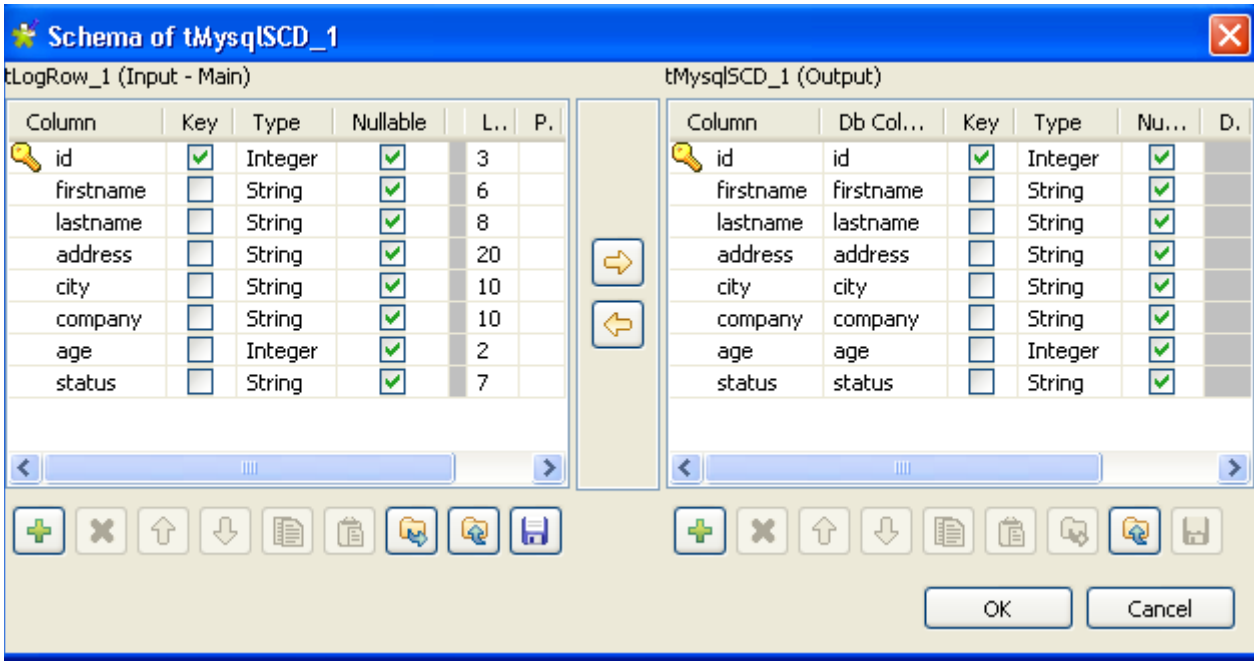
In this scenario, the source schema is made of eight columns: *firstname, lastname,address, city, company, age, status* and *id.*

- Set the basic settings for **tLogRow** in order for the content of the source file with varying attributes to display in cells of a table on the console before being processed through the SCD component.

- In the design workspace, double-click **tMysqlSCD** to define its basic settings.



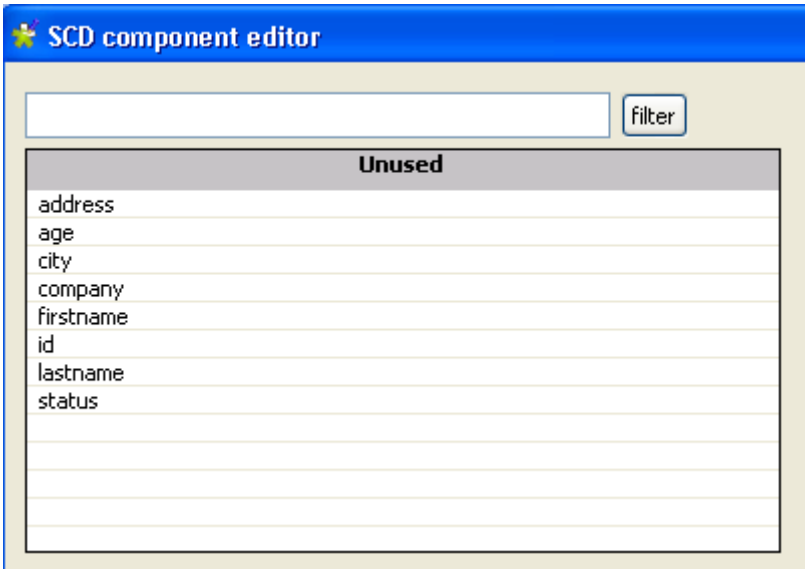
- In the **Basic settings** view, select the **Use an existing connection** check box to reuse the connection details defined on the **tMysqlConnection** properties.
- In the **Table** field, enter the table name to be used to track changes.
- Click **Sync columns** to auto propagate all the columns of the delimited file.
- If needed, click **Edit schema** to view the output data structure of **tMysqlSCD**.



- In the design workspace, double-click **tMysqlCommit** to define its basic settings.
- Select the relevant connection on the **Component list** if more than one connection exists.

Step 2: Setting up the SCD editor:

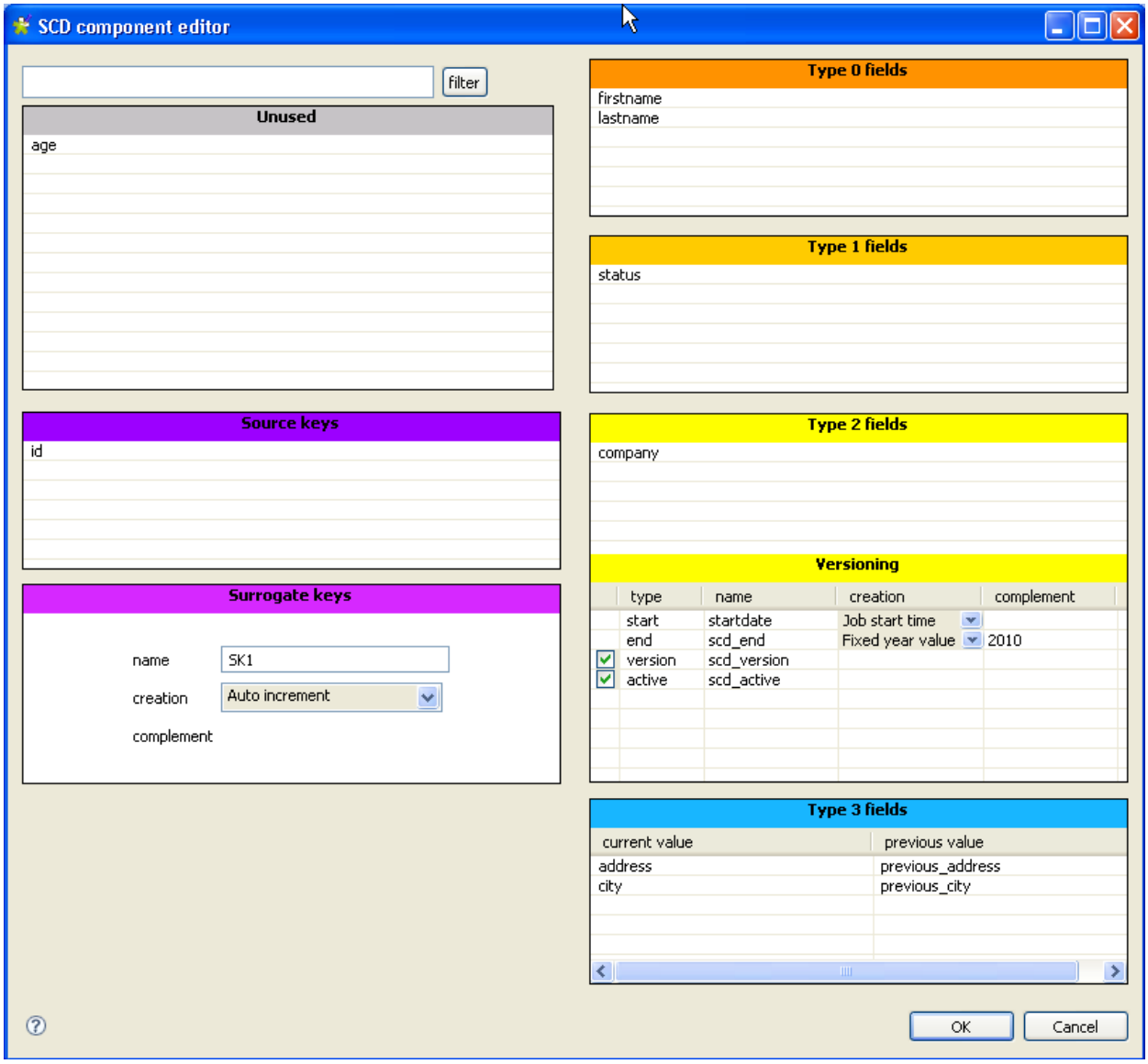
- In **tMysqlSCD** basic settings, click the three-dot [...] button next to the **SCD Editor** to open the **SCD editor** and build the data flow for the SCD outputs.



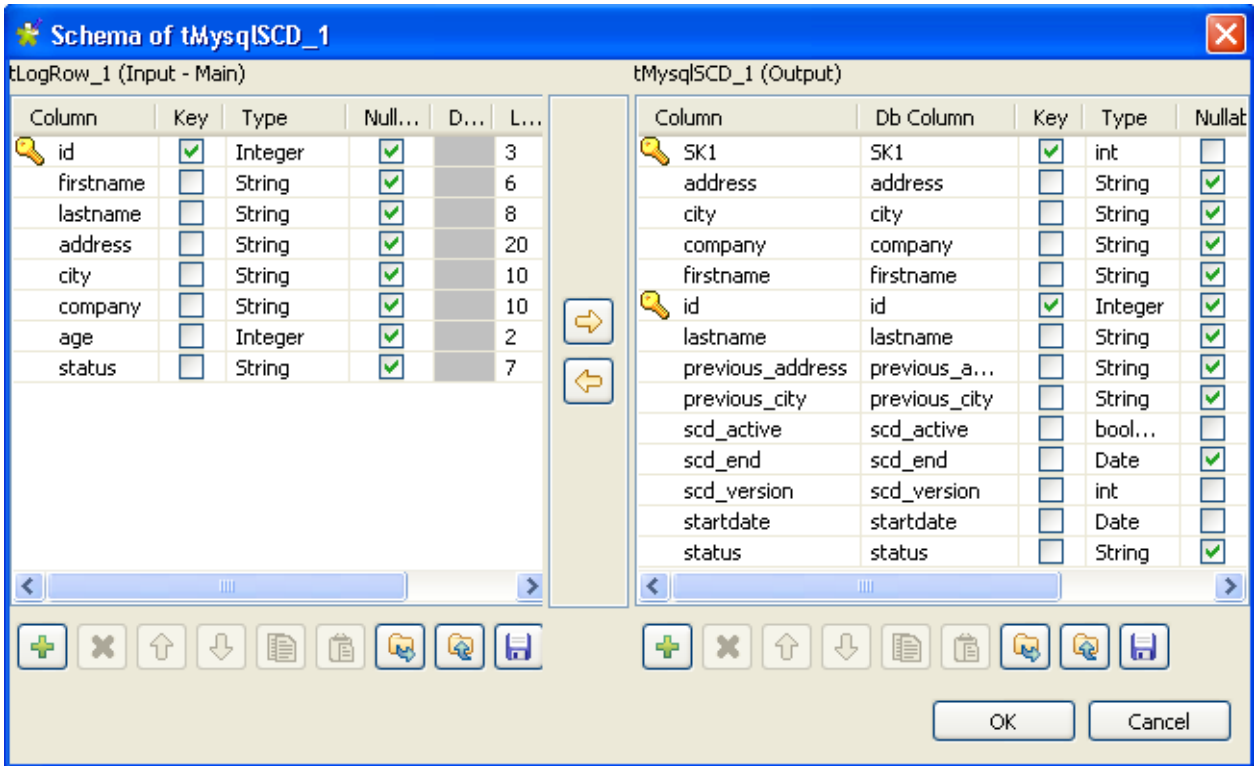
All columns of the source delimited file, coming from the previous component, display in the **Unused** panel of the **SCD editor**. All other panels in the **SCD editor** display empty.

- From the **Unused** list, drop the *id* column to the **Source keys** panel to use it as the key to ensure the unicity of the incoming data.
- In the **Surrogate keys** panel, enter a name for the surrogate key in the **Name** field, **SK1** in this scenario.
- From the **Creation** list, select the method to be used for the surrogate key generation, **Auto-increment** in this scenario.
- From the **Unused** list, drop the *firstname* and *lastname* columns to the **Type 0** panel, changes in these two columns do not interest us.
- Drop the *status* column to the **Type 1** panel. The new value will overwrite the old value.
- Drop the *company* column to the **Type 2** panel. Each time a person changes his/her company, a new record will be inserted in the dimensional table with a separate key.
  In the **Versioning** area:
  -Define the **start** and **end** columns of your SCD table that will hold the start and end date values. The end date is null for current records until a change is detected. Then the end date gets filled in and a new record is added with no end date.
  In this scenario, we select **Fixed Year Value** for the **end** column and fill in a fictive year to avoid having a null value in the end date field.
  -select the **version** check box to hold the version number of the record.
  -select the **active** check box to spot the column that will hold the **True** or **False** status. **True** for the current active record and **False** for the modified record.
- Drop the *address* and *city* columns to the **Type 3** panel to track only the information about the previous value of the address and city.

For more information about SCD types, see *SCD management methodologies on page 13*.



- Click **Ok** to validate your configuration and close the **SCD editor**.
- Click **Edit schema** to view the input and output data flow. The SCD output schema should include the SCD-specific columns that hold standard log information and that were defined in the **SCD editor**.



Step 3: Creating the SCD table:

You must create in the database the SCD table you defined in the **tMysqlSCD** basic settings to be the table that will track changes. The SCD table must include the delimited file schema along with the SCD output schema.

In this scenario, the SCD table has the following schema: *firstname, lastname,address, city, company, age status, id, SK1, previous_address, previuos city, scd_active, scd_end, scd version, ,* and *scd_date*.

To write changes and the history of changes in the defined SCD table, save your Job and press **F6** to execute it.

```
Starting job SCD1 at 00:07 15/12/2008.
.--+---------+--------+----------------------+------------+-----------+---+-------.
|                                    tLogRow_1                                     |
|=-+---------+--------+----------------------+------------+-----------+---+------=|
|id|firstname|lastname|address               |city        |company    |age|status |
|=-+---------+--------+----------------------+------------+-----------+---+------=|
|1 |Janet    |Anderson|511 Maple Ave.  Apt. 1B|Agoura Hills|Adecco     |30 |married|
|2 |Harold   |Smith   |662 Lyons Circle      |Alameda     |Adidas     |33 |married|
|3 |Adam     |Brown   |220 Vine Ave.         |Albany      |Bridgestone|28 |single |
|4 |Martin   |Clinton |770 Exmoor Rd.        |Alhambra    |Cutco      |25 |single |
|5 |Sue      |White   |1486 Oakwood          |Covina      |Black&White|35 |married|
'--+---------+--------+----------------------+------------+-----------+---+-------'

Job SCD1 ended at 00:07 15/12/2008. [exit code=0]
```

The console shows the content of the input delimited file.

Janet gets divorced and moves to *Adelanto* at *355 Golf Rd.* She works at *Greenwood.*

Adam gets married and moves to *Belmont* at *2505 Alisson ct.*He works at *Scoop.*

Update the delimited file with the above information and press **F6** to run your Job.

```
Starting job SCD1 at 00:32 15/12/2008.
.--+---------+--------+----------------+--------+-----------+---+-------.
|                                    tLogRow_1                             |
|=-+---------+--------+----------------+--------+-----------+---+------=|
|id|firstname|lastname|address         |city    |company    |age|status  |
|=-+---------+--------+----------------+--------+-----------+---+------=|
|1 |Janet    |Anderson|355 Golf Rd     |Adelanto|Greenwood  |30 |single  |
|2 |Harold   |Smith   |662 Lyons Circle|Alameda |Adidas     |33 |married|
|3 |Adam     |Brown   |2505 Alisson ct |Belmont |Scoop      |28 |married|
|4 |Martin   |Clinton |770 Exmoor Rd.  |Alhambra|Cutco      |25 |single  |
|5 |Sue      |White   |1486 Oakwood    |Covina  |Black&White|35 |married|
'--+---------+--------+----------------+--------+-----------+---+-------'

Job SCD1 ended at 00:32 15/12/2008. [exit code=0]
```

The console shows the changes in the personal information and the SCD table shows the history of changes made to the input file along with the status and version number.

SK1	id	firstname	lastname	address	previous_address	city	previous_city	company	status	scd_version	scd_activ
1	1	Janet	Anderson	511 Maple Ave. Apt. 1B	511 Maple Ave. Apt. 1B	Agoura Hills	Adelanto	Adecco	single	1	
2	2	Harold	Smith	662 Lyons Circle	NULL	Alameda	NULL	Adidas	married	1	
3	3	Adam	Brown	220 Vine Ave.	220 Vine Ave.	Albany	Belmont	Bridgestone	married	1	
4	4	Martin	Clinton	770 Exmoor Rd.	NULL	Alhambra	NULL	Cutco	single	1	
5	5	Sue	White	1486 Oakwood	NULL	Covina	NULL	Black&Wh...	married	1	
6	1	Janet	Anderson	355 Golf Rd	511 Maple Ave. Apt. 1B	Adelanto	Agoura Hills	Greenwood	single	2	
7	3	Adam	Brown	2505 Alisson ct	220 Vine Ave.	Belmont	Albany	Scoop	married	2	

tOracleSCD

tOracleSCD Properties

Component family	Databases/Oracle	
Function	**tOracleSCD** reflects and tracks changes in a dedicated Oracle SCD table.	
Purpose	**tOracleSCD** addresses Slowly Changing Dimension needs, reading regularly a source of data and logging the changes into a dedicated SCD table	
Basic settings	*Property type*	Either Built-in or Repository.
		Built-in: No property data stored centrally.
		Repository: Select the Repository file where properties are stored. The following fields are pre-filled in using fetched data.
	Use an existing connection	Select this check box and click the relevant DB connection component on the **Component list** to reuse the connection details you already defined
	Connection type	Select the relevant driver on the list.
	Host	Database server IP address.
	Port	Listening port number of DB server.
	Database	Name of the database.
	Schema	Name of the DB schema.
	Username and *Password*	DB user authentication data.
	Table	Name of the table to be written. Note that only one table can be written at a time.
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either built-in or remotely stored in the Repository.
		Built-in: The schema is created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	SCD Editor	The SCD editor helps to build and configure the data flow for slowly changing dimension outputs. For more information, see *SCD management methodologies on page 13*.
	Use memory saving Mode	Select this check box to maximize system performance.

Usage	This component is used as Output component. It requires an Input component and Row main link as input.

Related scenario

For related scenarios, see **tMysqlSCD** *Scenario: Tracking changes using Slowly Changing Dimensions (type 0 through type 3) on page 15*.

tSybaseSCD

tSybaseSCD properties

Component family	Databases/Sybase	
Function	**tSybaseSCD** reflects and tracks changes in a dedicated Sybase SCD table.	
Purpose	**tSybaseSCD** addresses Slowly Changing Dimension needs, reading regularly a source of data and logging the changes into a dedicated SCD table	
Basic settings	*Property type*	Either Built-in or Repository.
		Built-in: No property data stored centrally.
		Repository: Select the Repository file where Properties are stored. The following fields are pre-filled in using fetched data.
	Host	Database server IP address.
	Port	Listening port number of DB server.
	Database	Name of the database.
	Username and *Password*	DB user authentication data.
	Table	Name of the table to be written. Note that only one table can be written at a time.
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either built-in or remotely stored in the Repository.
		Built-in: The schema is created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	SCD Editor	The SCD editor helps to build and configure the data flow for slowly changing dimension outputs. For more information, see *SCD management methodologies on page 13*.
	Use memory saving Mode	Select this check box to maximize system performance.
Usage	This component is used as Output component. It requires an Input component and Row main link as input.	

Related scenarios

For related topics, see *Scenario: Tracking changes using Slowly Changing Dimensions (type 0 through type 3) on page 15*.

tSPSSInput

Before being able to benefit from all functional objectives of the SPSS components, make sure to do the following:
-If you have already installed SPSS, add the path to the SPSS directory as the following: SET PATH=%PATH%;<DR>:\program\SPSS, or
-If you have not installed SPSS, you must copy the SPSS IO "spssio32.dll" lib from the SPSS installation CD and paste it in your "system32" directory.

tSPSSInput properties

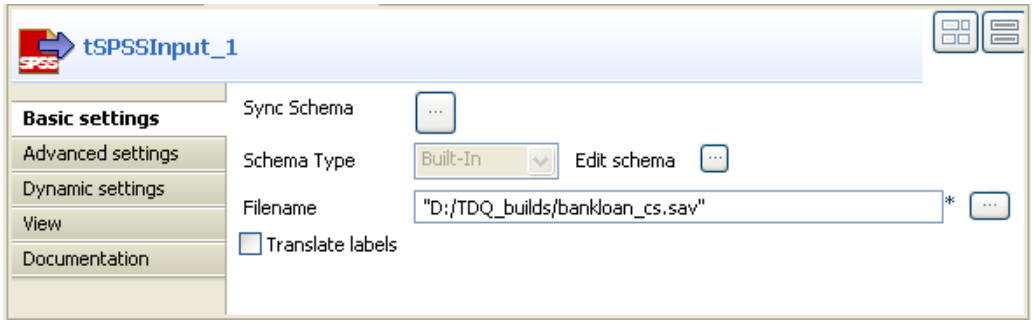
Component family	Business Intelligence	
Function	**tSPSSInput** reads data from SPSS .sav files.	
Purpose	**tSPSSInput** addresses SPSS .sav data to write it for example in another file.	
Basic settings	*Schema type* and *Edit Schema*	The schema metadata in this component is retrieved directly from the input SPSS .sav file and thus is read-only. You can click **Edit schema** to view the retrieved metadata.
	Sync schema	Click this button to synchronize with the columns of the input SPSS .sav file.
	File name	Name or path of the SPSS .sav file to be read.
	Translate labels	Select this check box to translate the labels of the stored values. If you select this check box, you need to retrieve the metadata again.
Usage	This component is used as a start component. It requires an output flow.	

Scenario: Displaying the content of an SPSS .sav file

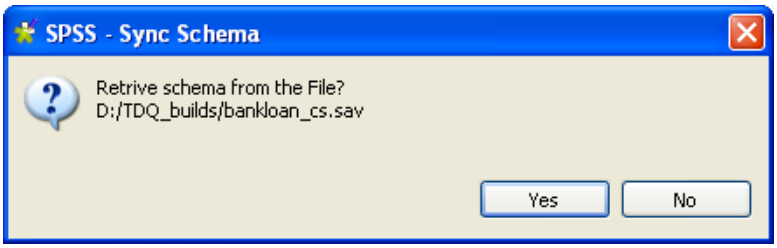
The following scenario creates a two-component Job, which aims at reading each row of a .sav file and displaying the output on the log console.

- Drop a **tSPSSInput** component and **tLogRow** component from the **Palette** onto the design workspace.

- Right-click on **tPSSInput** and connect it to **tLogRow** using a **Main Row** link.
- Click **tSPSSInput** to display its **Basic settings** view and define its properties.



- Click the three-dot button next to the **Filename** field and browse to the SPSS .sav file you want to read.
- Click the three-dot button next to **Sync schema**. A message opens up prompting you to accept retrieving the schema from the defined SPSS file.



- Click **Yes** to close the message.
- If needed, click the three-dot button next to **Edit schema** to view the pre-defined data structure of the source SPSS file.
- Click **OK** to close the dialog box.
- In the **Basic settings** view, select the **Translate label** check box if you want to carry out translation on the stored labels.

  If you select this check box, you have to click **Sync Schema** a second time to retrieve the schema from the defined file.

- Save the Job and press **F6** to execute it.

The SPSS file is read row by row and the extracted fields are displayed on the log console.

The Log sums up all parameters in a header followed by the result of the Job.

tSPSSOutput

Before being able to benefit from all functional objectives of the SPSS components, make sure to do the following:
-If you have already installed SPSS, add the path to the SPSS directory as the following: SET PATH=%PATH%;<DR>:\program\SPSS, or
-If you have not installed SPSS, you must copy the SPSS IO "spssio32.dll" lib from the SPSS installation CD and paste it in your "system32" directory.

tSPSSOutput properties

Component family	Business Intelligence	
Function	**tSPSSOutput** writes data entries in a .sav file.	
Purpose	**tSPSSOutput** writes or appends data to an SPSS .sav file. It creates SPSS files on the fly and overwrites existing ones.	
Basic settings	*Schema type* and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either built-in or remotely stored in the Repository.
		Built-in: The schema is created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	File name	Name or path of the SPSS .sav file to be written.
	Write Type	Select the writing type from the list: **Write**: simply writes the new data. **Append**: writes the new data at the end of the existing data.
Usage	This component can not be used as start component. It requires an input flow	

Related scenarios

For related topics, see:

- *Scenario: Displaying the content of an SPSS .sav file on page 27.*

tSPSSProperties

tSPSSProperties properties

Component family	Business Intelligence	
Function	**tSPSSProperties** describes the properties of a defined SPSS .sav file.	
Purpose	**tSPSSProperties** allows to have information about the main properties of a defined SPSS .sav file.	
Basic settings	*Schema type* and *Edit Schema*	The schema metadata in this component is predefined and thus read-only. You can click **Edit schema** to view the predefined metadata. A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either built-in or remotely stored in the Repository.
		Built-in: The schema is created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	File name	Name or path of the .sav file to be processed.
Usage	Use this component as a start component. It needs an output flow.	

Related scenarios

For related topics, see:

- *Scenario: Displaying the content of an SPSS .sav file on page 27*.

tSPSSStructure

tSPSSStructure properties

Component family	Business Intelligence	
Function	**tSPSSStructure** retrieves information about the variables inside .sav files.	
Purpose	**tSPSSStructure** addresses variables inside .sav files. You can use this component in combination with **tFileList** to gather information about existing *.sav files to further analyze or check the findings.	
Basic settings	*Schema type* and *Edit Schema*	The schema metadata in this component is predefined and thus read-only. It is based on the internal SPSS convention. You can click **Edit schema** to view the predefined metadata. A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either built-in or remotely stored in the Repository.
		Built-in: The schema is created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	File name	Name or path of the .sav file to be processed.
Usage	Use this component as a start component. It needs an output flow.	

Related scenarios

For related topics, see:

- *Scenario: Displaying the content of an SPSS .sav file on page 27.*

Business components

This chapter details the major components that you can find in **Business** group of the **Palette** of **Talend Open Studio**.

The Business component family groups connectors that covers specific Business needs, such as reading and writing CRM, or ERP types of database and reading from or writing to an SAP system.

tAlfrescoOutput

tAlfresco Properties

Component family	Business	
Function	Creates dematerialized documents in an Alfresco server where they are indexed under meaningful models.	
Purpose	Allows to create and manage documents in an Alfresco server.	
Basic settings	*URL*	Type in the URL to connect to the Alfresco Web application.
	Login and Password	Type in the user authentication data to the Alfresco server.
	Base	Type in the base path where to put the document, or Select the **Map...** check box and then in the **Column** list, select the target location column. **Note:** When you type in the base name, make sure to use the double backslash (\\) escape character.
	Document Mode	Select in the list the mode you want to use for the created document. **Create only**: creates a document if it does not exist. Note that an error message will display if you try to create a document that already exists **Create or update**: creates a document if it does not exist or updates the document if it exists.
	Container Mode	Select in the list the mode you want to use for the destination folder in Alfresco. **Update only**: updates a destination folder if the folder exists. Note that an error message will display if you try to update a document that does not exist **Create or update**: creates a destination folder if it does not exist or updates the destination folder if it exists.
	Define Document Type	Click the three-dot button to display the **tAlfrescoOutput** editor. This editor enables you to: - select the file where you defined the metadata according to which you want to save the document in Alfresco -define the type f the document -select any of the aspects in the **available aspects** list of the model file and click the plus button to add it in the list to the left.
	Property Mapping	Displays the parameters you set in the **tAlfrescoOutput** editor and according to which the document will be created in the Alfresco server. Note that in the **Property Mapping** area, you can modify any of the input schemas.

	Schema and Edit schema	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the Repository. Click **Edit Schema** to make changes to the schema. Note that if you make changes, the schema automatically becomes built-in.
	Result Log File Name	Browse to the file where you want to save any logs related to the job execution.
Advanced settings	*Configure Target Location Container*	Allows to configure the (by default) type of containers (folders) Select this check box to display new fields where you can modify the container type to use your own created types based on the father/child model.
	Configure Permissions	When selected, allows to manually configure access rights to containers and documents. Select the **Inherit Permissions** check box to synchronize access rights between containers and documents. Click the **Plus** button to add new lines to the **Permissions** list, then you can assign roles to user or group columns.
	Encoding Type	Select the encoding type from the list or select **Custom** and define it manually. This field is compulsory.
	Association Target Mapping	Allows to create new documents in Alfresco with associated links towards other documents already existing in Alfresco, to facilitate the navigation process for example. To create associations: -Open the **tAlfresco** editor. -Click the **Add** button and select a model where you have already defined aspects that contain associations. -Click the drop-down arrow at the top of the editor and select the corresponding document type. -Click **OK** to close the editor and display the created association in the **Association Target Mapping** list.
	tStatCatcher Statistics	Select this check box to gather the job processing metadata at a job level as well as at each component level.
Usage	Usually used as an output component. An input component is required.	
Limitation/Prerequisites	To be able to use the **tAlfrescoOutput** component, few relevant resources need to be installed: check the *Installation procedure* sub section for more information.	

Installation procedure

To be able to use **tAlfrescoOutput** in **Talend Open Studio**, you need first to install the Alfresco server with few relevant resources.

The below sub sections detail the prerequisite and the installation procedure.

Prerequisites

Start with the below operations:

- Download the file `alfresco-community-tomcat-2.1.0.zip`
- Unzip the file in an installation folder, for example: `C:\Program Files\Java\jdk1.50_16`
- Install JDK 1.5.0+
- Update the environment variable `JAVA_HOME (JAVA_HOME= C:\alfresco)`
- From the installation folder (`C:\alfresco`), launch the alfresco server using the script `alf_start.bat`

Make sure that the Alfresco server is launched correctly before start using the **tAlfrescoOutput** *component.*

Installing the Talend Alfresco module

Note that the `talendalfresco_20081014.zip` is provided with the **tAlfrescoOutput** component in **Talend Open Studio**.

To install the `talendalfresco` module:

- From `talendalfresco_20081014.zip` and in the `talendalfresco_20081014\alfresco` folder, look for the following jars: `stax-api-1.0.1.jar, wstx-lgpl-3.2.7.jar`, `talendalfresco-client_1.0.jar`, and `talendalfresco-alfresco_1.0.jar` and move them to `C:\alfredsco\tomcat\webapps\alfresco\WEB-INF\lib`
- Add the authentification filter of the commands to the `web.xml` file located in the path `C:\alfredsco\tomcat\webapps\alfresco\WEB-INF son WEB-INF/` following the model of the example provided in `talendalfresco_20081014/alfresco` folder of the zipped file `talendalfresco_20081014.zip`

The below figures show the portion of lines (in blue) to add in the file `web.xml alfresco.`

```
<filter>
   <filter-name>Admin Authen
   <filter-class>org.alfresc
</filter>

<filter-mapping>
   <filter-name>Authenticati
   <url-pattern>/faces/*</ur
</filter-mapping>
```

```
<filter>
   <filter-name>Admin Authentication Filter</filter-name>
   <filter-class>org.alfresco.web.app.servlet.AdminAuthenticationFilter</filter-class>
</filter>

<!-- [talendalfresco] command servlet auth filter def -->
<filter>
   <filter-name>Command Authentication Filter</filter-name>
   <filter-class>fr.openwide.talendalfresco.rest.server.CommandAuthenticationFilter</filter-class>
</filter>

<filter-mapping>
   <filter-name>Authentication Filter</filter-name>
   <url-pattern>/faces/*</url-pattern>
</filter-mapping>
```

Useful information for advanced use

Installing new types for Alfresco:

From the `package_jeu_test.zip` and in the `package_jeu_test/fichiers_conf_alfresco2.1` folder, look for the following files: `xml H76ModelCustom.xml` (description of the model), `web-client-config-custom.xml` (web interface of the model), and `custom-model-context.xml` (registration of the new model) and paste them in the following folder:
`C:/alfredsco/tomcat/shared/classes/alfresco/extension`

Dates:

- The dates must be of the **Talend** date type `java.util.Date`
- Columns without either mapping or default values, for example of the type `Date`, are written as empty strings.
  Solution: delete all columns without mapping or default values. Note that any modification of the type Alfresco will put them back.

Content:

- Do not mix up between the file path which content you want to create in Alfresco and its target location in Alfresco.
- Provide a URL! It can target various protocols, among which are file, HTTP and so on.
- For URLs referring to files on the file system, precede them by "file:" for Windows used locally, and by "file://" for Windows on a network (which accepts as well "file: \ \") or for Linux.
- Do not double the backslash in the target base path (automatic escape), unless you type in the path in the basic settings of the **tAlfrescoOutput** component, or doing concatenation in the **tMap** editor for example.

Multiple properties or associations:

- It is possible to create only one association by document if it is mapped to a string value, or one or more associations by document if it is mapped to a list value (object).
- You can empty an association by mapping it to an empty list, which you can create, for example, by using `new java.util.ArrayList()` in the **tMap** component.

- However, it is impossible to delete an association.
- Building `List(object)` with **tAggregate**:
  -define the table of the relation n-n in a file, containing a `name` line for example (included in the input rows), and a `category` line (that can be defined with its mapping in a third file).
  -group by: input name, output name.
  -operation: output `categoryList`, function `list(object)`, input `category`. ATTENTION list (object) and non simple list!

References (documents and folders):

- References are created by mapping one or more existing reference nodes (xpath or namepath) using `String` type or `List(object).`
- An error in the association or the property of the reference type does not prevent the creation of the node that holds the reference.
- Properties of the reference type are created in the **Basic Settings** view.
- Associations are created in the **Advanced Settings** view.

Dematerialization, tAlfrescoOutput, and Enterprise Content Management

Dematerialization is the process that convert documents held in physical form into electronic form, and thus helps to move away from the use of physical documentation to the use of electronic Enterprise Content Management (ECM) systems. The range of documents that can be managed with an Enterprise Content Management system include just about everything from basic documents to stock certificates, for example.

Enterprises dematerialize their content via a manual document handling, done by man, or an automatic document handling, machine-based.

Considering the varied nature of the content to be dematerialized, enterprises have to use varied technologies to do it. Scanning paper documents, creating interfaces to capture electronic documents from other applications, converting document images into machine-readable/editable text documents, and so on are examples of the technologies available.

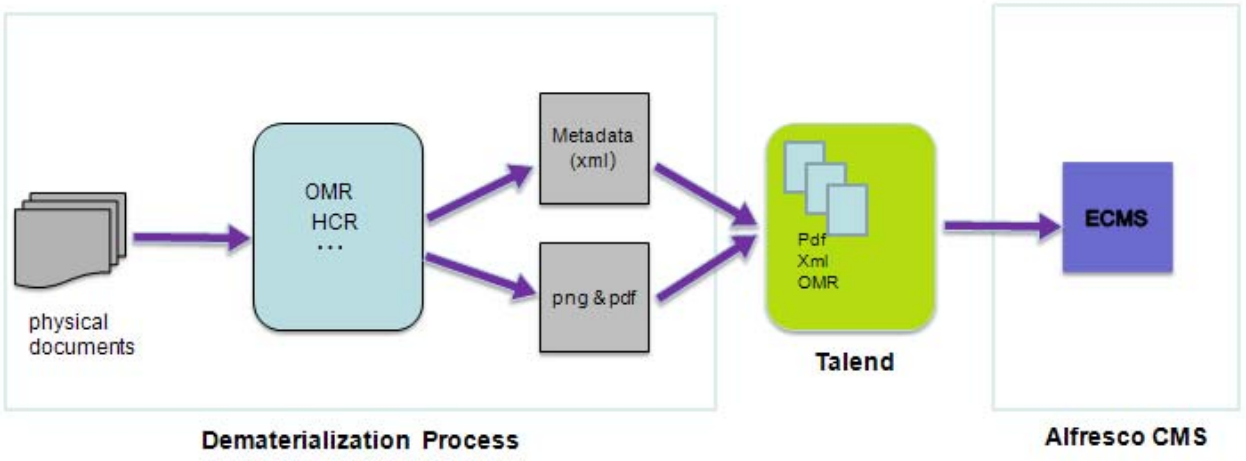
Furthermore, scanned documents and digital faxes are not readable texts. To convert them into machine-readable characters, different character recognition technologies are used. Handwritten Character Recognition (HCR) and Optical Mark Recognition (OMR) are two examples of such technologies.

Equally important as the content that is captured in various formats from numerous sources in the dematerialization process is the supporting metadata that allows efficient identification of the content via specific queries.

Now how can this document content along with the related metadata be aggregated and indexed in an Enterprise Content Management system so that it can be retrieved and managed in meaningful ways? **Talend** provides the answer through the **tAlfrescoOutput** component.

The **tAlfrescoOutput** component allows you to stock and manage your electronic documents and the related metadata on the Alfresco server, the leading open source enterprise content management system.

The below figure illustrates **Talend**'s role between the dematerialization process and the Enterprise Content Management system (Alfresco).

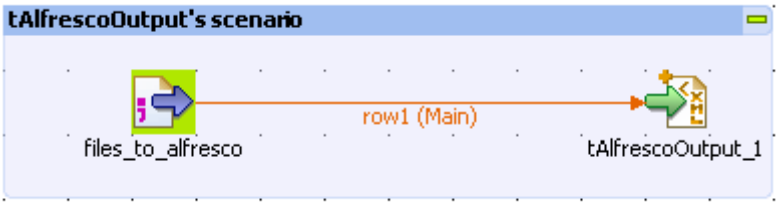


ECMS: Enterprise Content Management System
HCR: Handwritten Character Recognition
OMR: Optical Mark Recognition

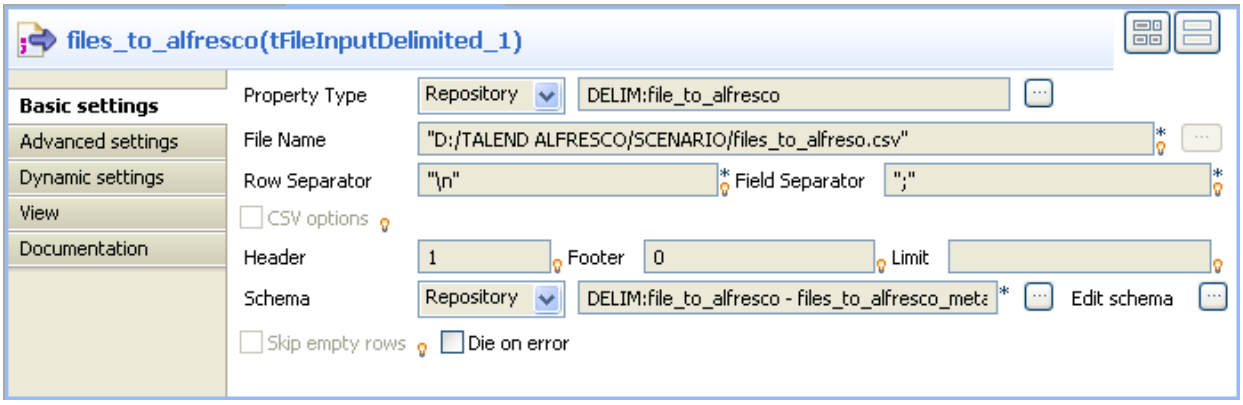
Scenario: Creating documents on an Alfresco server

This Java scenario describes a two-component Job which aims at creating two document files with the related metadata in an Alfresco server, the java-based Enterprise Control Management system.

- Drop the **tFileInputDelimited** and **tAlfrescoOutput** components from the **Palette** onto the design workspace.
- Connect the two components together using a **Row Main** link.



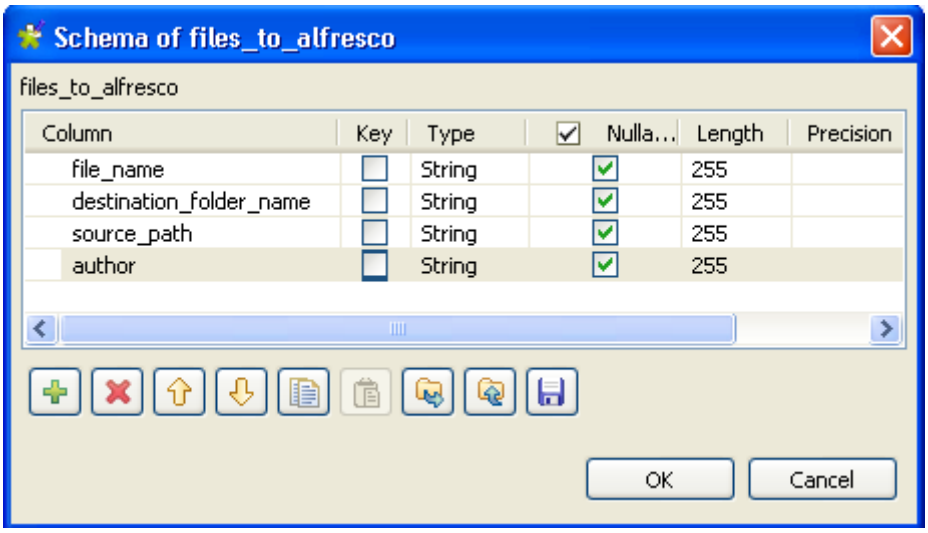
- In the design workspace, double-click **tFileInputDelimited** to display its basic settings**.**
- Set the **File Name** path and all related properties. Note that if you have already stored your input schemas locally in the **Repository**, you can simply drop the relevant file item from the **Metadata** folder onto the design workspace and the delimited file settings will automatically display in the relevant fields in the component **Basic settings** view.
  For more information about metadata, see *Setting up a File Delimited schema* in **Talend Open Studio** User Guide.



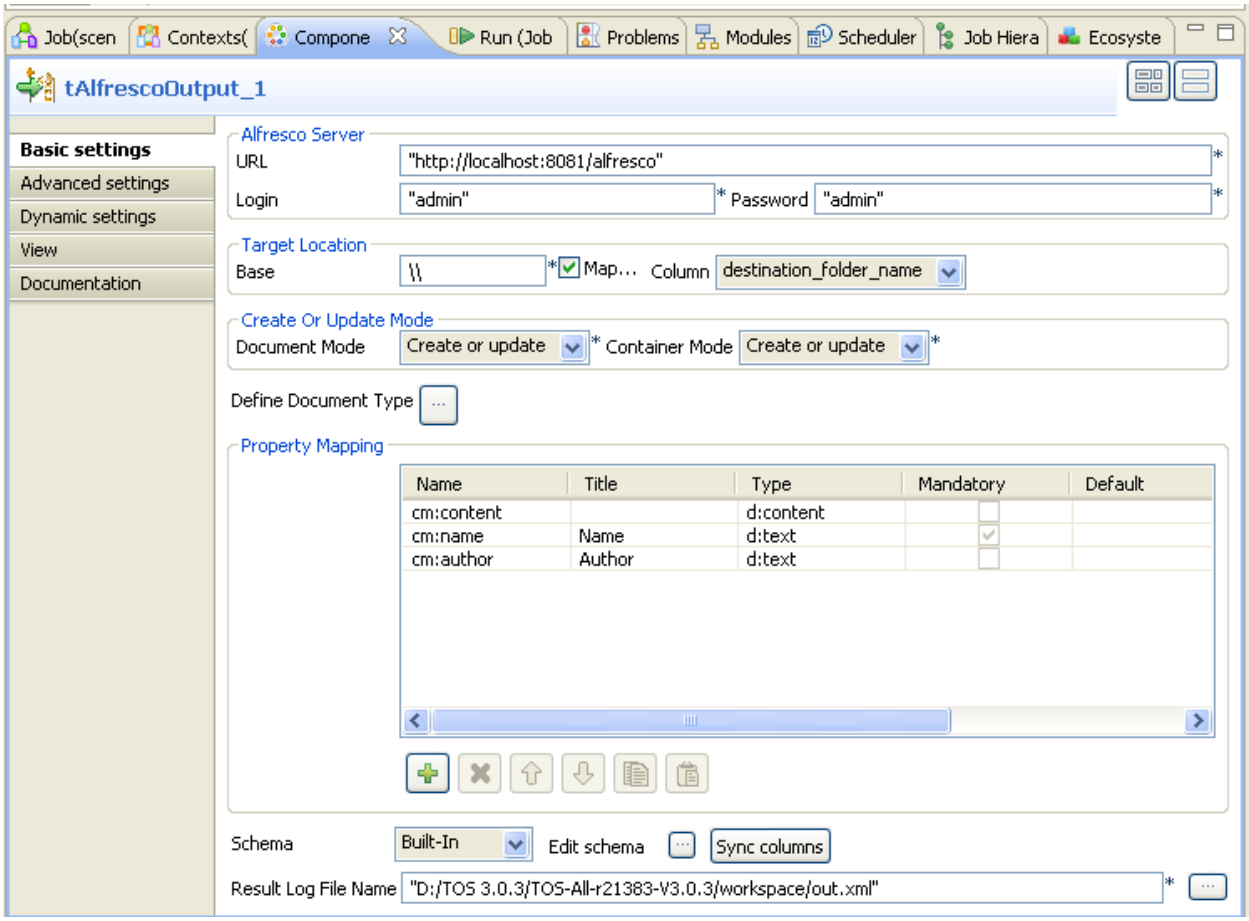
In this scenario, the delimited file provides the metadata and path of two documents we want to create in the Alfresco server. The input schema for the documents consists of four columns: `file_name, destination_folder name, source_path,` and `author`.

	A	B	C	D	E	F
1	file_name;destination_folder_name;source_path;author					
2	PO_43278.pdf;FINANCE\PO;file:D:/PO/customer1_2009-03-24.pdf;talend					
3	PO_43279.pdf;FINANCE\PO;file:D:/PO/customer2_2009-03-24.pdf;talend					

And therefore the input schema of the delimited file will be as the following:

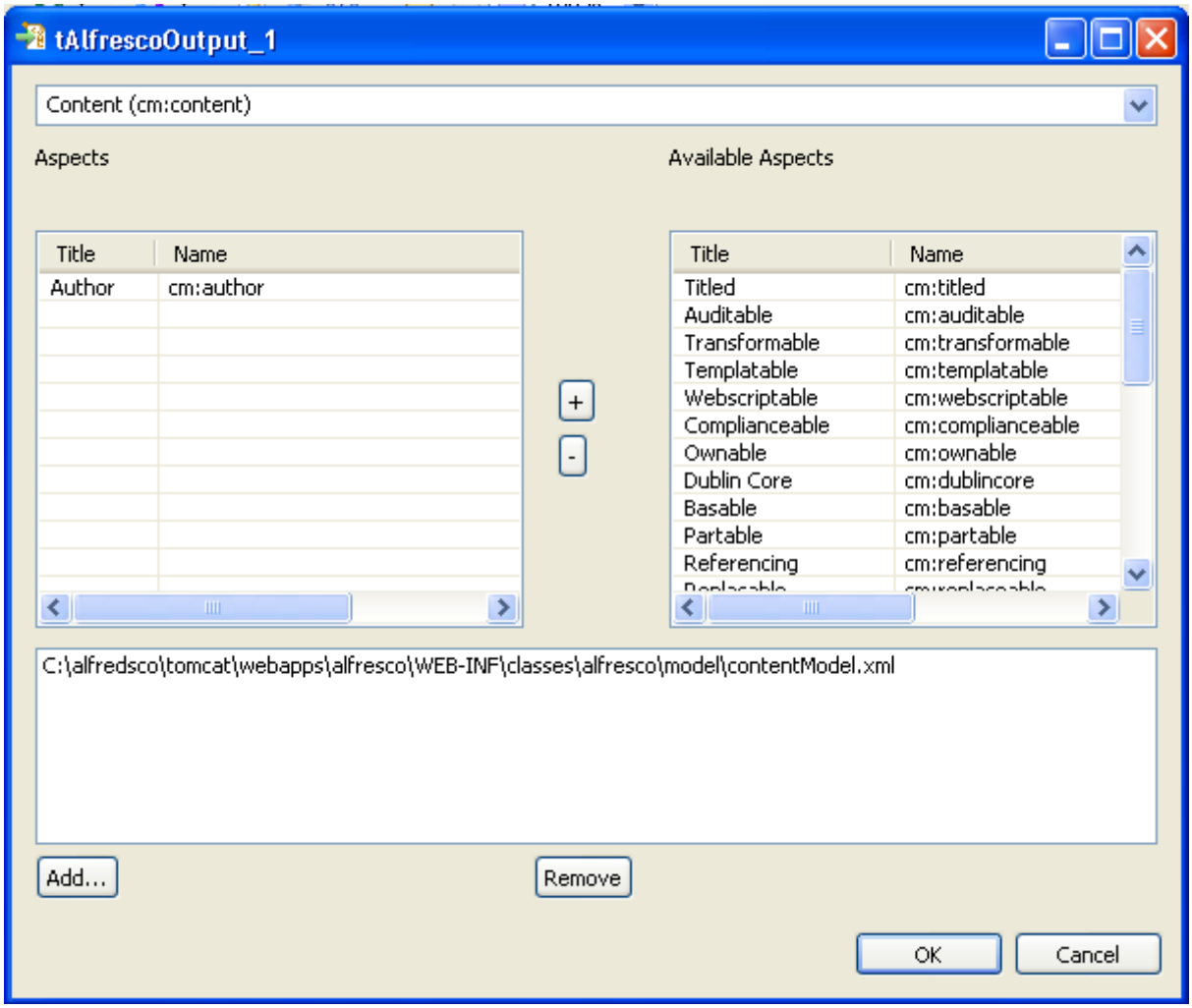


- In the design workspace, double-click **tAlfrescoOutput** to display its basic settings**.**



- In the **Alfresco Server** area, enter the Alfresco server URL and user authentication information in the corresponding fields.
- In the **TargetLocation** area, either type in the base name where to put the document in the server, or Select the **Map...** check box and then in the **Column** list, select the target location column, `destination_folder_name` in this scenario.

  When you type in the base name, make sure to use the double backslash (\\) escape character.
- In the **Document Mode** list, select the mode you want to use for the created documents.
- In the **Container Mode** list, select the mode you want to use for the destination folder in Alfresco.
- Click the **Define Document Type** three-dot button to open the **tAlfrescoOutput** editor.



- Click the **Add** button to browse and select the xml file that holds the metadata according to which you want to save the documents in Alfresco.
  All available aspects in the selected model file display in the **Available Aspects** list.

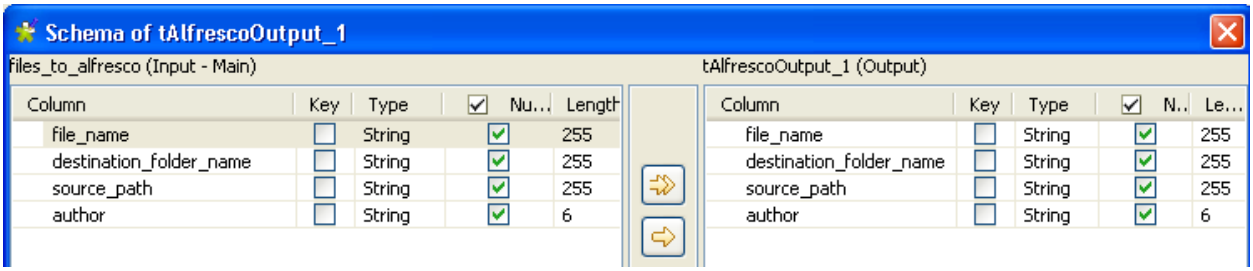
  You can browse for this model folder locally or on the network. After defining the aspects to use for the document to be created in Alfresco, this model folder is not needed any more.

- If needed, select in the **Available Aspects** list the aspect(s) to be included in the metadata to write in the Alfresco server. In this scenario we want the author name to be part of the metadata registered in Alfresco.

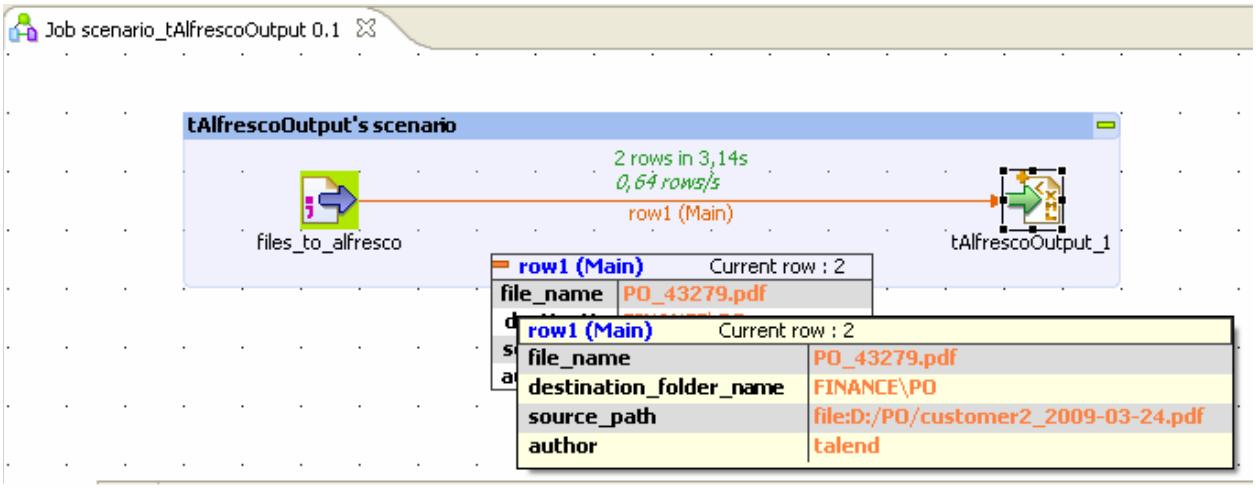
- Click the drop-down arrow at the top of the editor to select from the list the type to give to the created document in Alfresco, `Content` in this scenario.
  All the defined aspects used to select the metadata to write in the Alfresco server display in the **Property Mapping** list in the **Basic Settings** view of **tAlfrescoOutput**, three aspects in this scenario, two basic for the `Content` type (`content` and `name`) and an additional one (`author`).

- Click **Sync columns** to auto propagate all the columns of the delimited file.

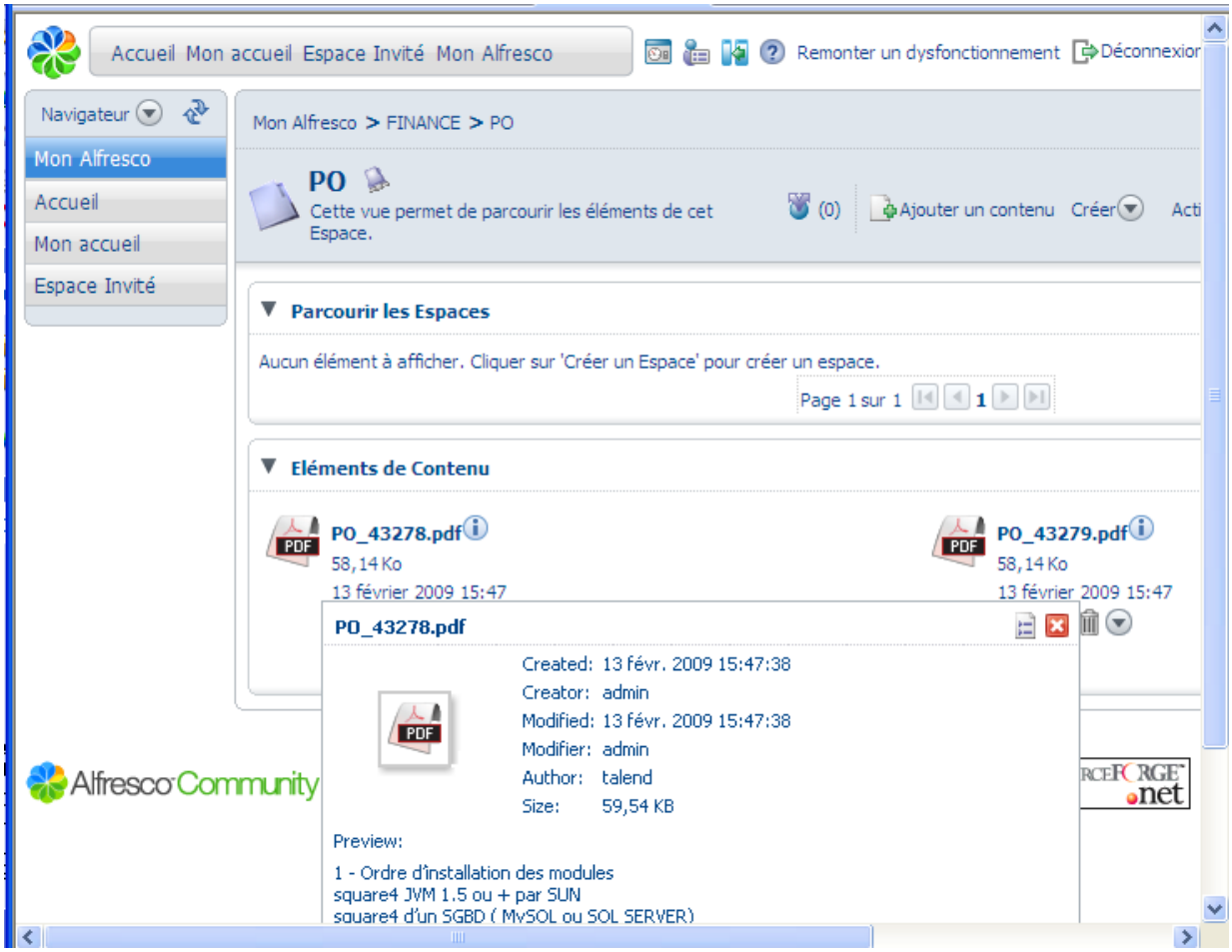
- If needed, click **Edit schema** to view the output data structure of **tAlfrescoOutput**.



- Click the three-dot button next to the **Result Log File Name** field and browse to the file where you want to save any logs after job execution.
- Save your Job, and press **F6** to execute it.



The two documents are created in Alfresco using the metadata provided in the input schemas.


Accueil Mon accueil Espace Invité Mon Alfresco
Remonter un dysfonctionnement
Déconnexion
Navigateur
Mon Alfresco
Accueil
Mon accueil
Espace Invité
Mon Alfresco > FINANCE > PO
PO
Cette vue permet de parcourir les éléments de cet Espace.
(0)
Ajouter un contenu
Créer
Parcourir les Espaces
Aucun élément à afficher. Cliquer sur 'Créer un Espace' pour créer un espace.
Page 1 sur 1
1
Eléments de Contenu
PO_43278.pdf
58,14 Ko
13 février 2009 15:47
PO_43279.pdf
58,14 Ko
13 février 2009 15:47
PO_43278.pdf
Created: 13 févr. 2009 15:47:38
Creator: admin
Modified: 13 févr. 2009 15:47:38
Modifier: admin
Author: talend
Size: 59,54 KB
Preview:
1 - Ordre d'installation des modules
square4 JVM 1.5 ou + par SUN
square4 d'un SGBD (MySQL ou SQL SERVER)
Alfresco Community

tCentricCRMInput

tCentricCRMInput Properties

Component family	Business/CentricCRM	
Function	Connects to a module of a Centric CRM database via the relevant webservice.	
Purpose	Allows to extract data from a Centric CRM DB based on a query.	
Basic settings	*CentricCRM URL*	Type in the webservice URL to connect to the CentricCRM DB.
	Module	Select the relevant module in the list
	Server	Type in the IP address of the DB server.
	UserID and *Password*	Type in the Webservice user authentication data.
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the Repository. Click **Edit Schema** to make changes to the schema. Note that if you make changes, the schema automatically becomes built-in. In this component the schema is related to the Module selected.
	Query condition	Type in the query to select the data to be extracted.
Usage	Usually used as a Start component. An output component is required.	
Limitation	n/a	

Related Scenario

No scenario is available for this component yet.

tCentricCRMOutput

tCentricCRMOutput Properties

Component family	Business/CentricCRM	
Function	Writes data in a module of a CentricCRM database via the relevant webservice.	
Purpose	Allows to write data into a CentricCRM DB.	
Basic settings	*CentricCRM URL*	Type in the webservice URL to connect to the CentricCRM DB.
	Module	Select the relevant module in the list
	Server	IP address of the DB server
	Username and *Password*	Type in the Webservice user authentication data.
	Action	**Insert**, **Update** or **Delete** the data in the CentricCRM module.
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the Repository. Click **Edit Schema** to make changes to the schema. Note that if you make changes, the schema automatically becomes built-in. Click **Sync columns** to retrieve the schema from the previous component connected in the Job.
Usage	Used as an output component. An Input component is required.	
Limitation	n/a	

Related Scenario

No scenario is available for this component yet.

tMicrosoftCRMInput

tMicrosoftCRMInput Properties

Component family	Business	
Function	Connects to an entity of Microsoft CRM database via the relevant webservice.	
Purpose	Allows to extract data from a MicrosoftCRM DB based on conditions set on specific columns.	
Basic settings	*Property type*	Either **Built-in** or **Repository**:
		Built-in: No property data stored centrally.
		Repository: Select the Repository file where properties are stored. The fields that come after are pre-filled in using the fetched data.
	Microsoft Webservice URL	Type in the webservice URL to connect to the MicrosoftCRM DB.
	Organizename	Enter the name of the user or organization, set by an administrator, that needs to access the MicrosoftCRM database.
	Username and *Password*	Type in the Webservice user authentication data.
	Domain	Type in the domain name of the server on which MicrosoftCRM is hosted.
	Host	Type in the IP address of Microsoft CRM database server.
	Port	Listening port number of Microsoft CRM database server.
	Time out (seconds)	Number of seconds for the port to listen before closing.
	Entity	Select the relevant entity in the list.
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the Repository. Click **Edit Schema** to make changes to the schema.  if you make changes, the schema automatically becomes built-in.  In this component the schema is related to the selected entity.
	Logical operators used to combine conditions	In the case you want to combine the conditions you set on columns, select the combine mode you want to use.

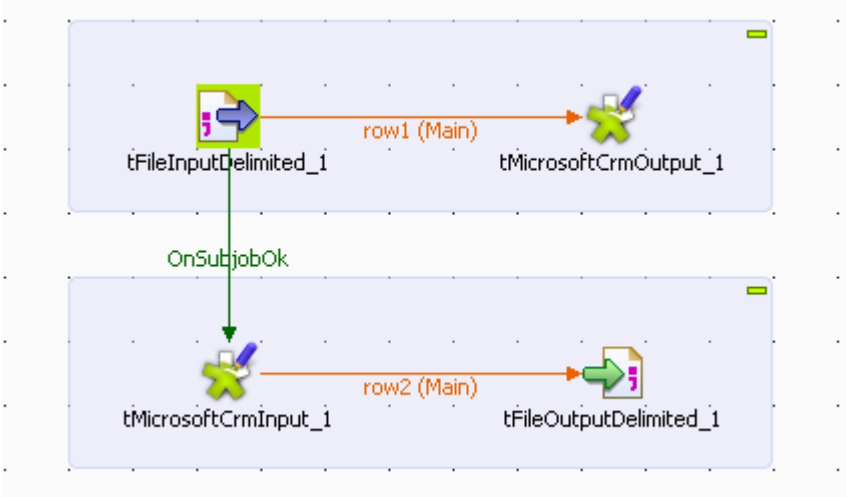
	Conditions	Click the plus button to add as many conditions as needed. The conditions are performed one after the other for each row. **Input column**: Click in the cell and select the column of the input schema the condition is to be set on. **Operator**: Click in the cell and select the operator to bind the input column with the value. **Value**: Type in the column value, between quotes if need be.
Advanced settings	*tStatCatcher Statistics*	Select this check box to gather the job processing metadata at a job level as well as at each component level.
Usage	Usually used as a Start component. An output component is required.	
Limitation	n/a	

Scenario: Writing data in a Microsoft CRM database and putting conditions on columns to extract specified rows

This scenario describes a four-component Job which aims at writing the data included in a delimited input file in a custom entity in a MicrosoftCRM database. It then extracts specified rows to an output file using the conditions set on certain input columns.

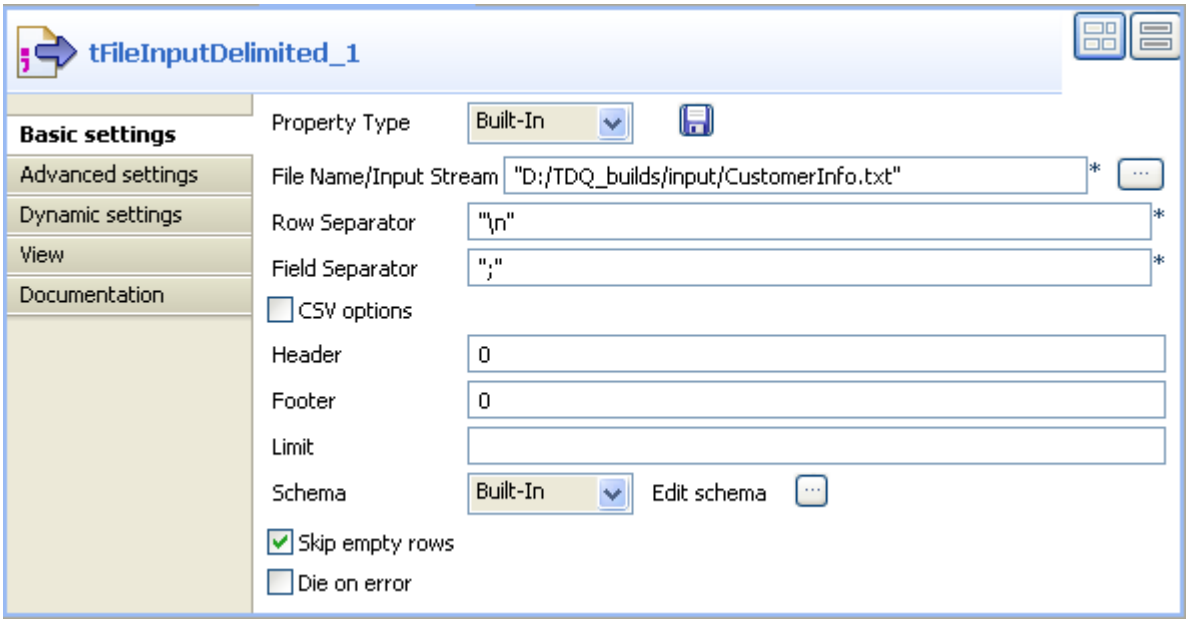
If you want to write in a CustomEntity in Microsoft CRM database, make sure to name the columns in accordance with the naming rule set by Microsoft, that is "name_columnname" all in lower case.

- Drop the following components from the **Palette** to the design workspace: **tFileInputdelimited**, **tFileOutputDelimited**, **tMicrosoftCRMInput**, and **tMicrosoftCRMOutput**.



- Connect **tFileInputDelimited** to **tMicrosoftCRMOutput** using a **Row Main** connection.
- Connect **tMicrosoftCRMIntput** to **tFileOutputDelimited** using a **Row Main** connection.
- Connect **tFileInputDelimited** to **tMicrosoftCRMInput** using **OnSubjobOk** connection.

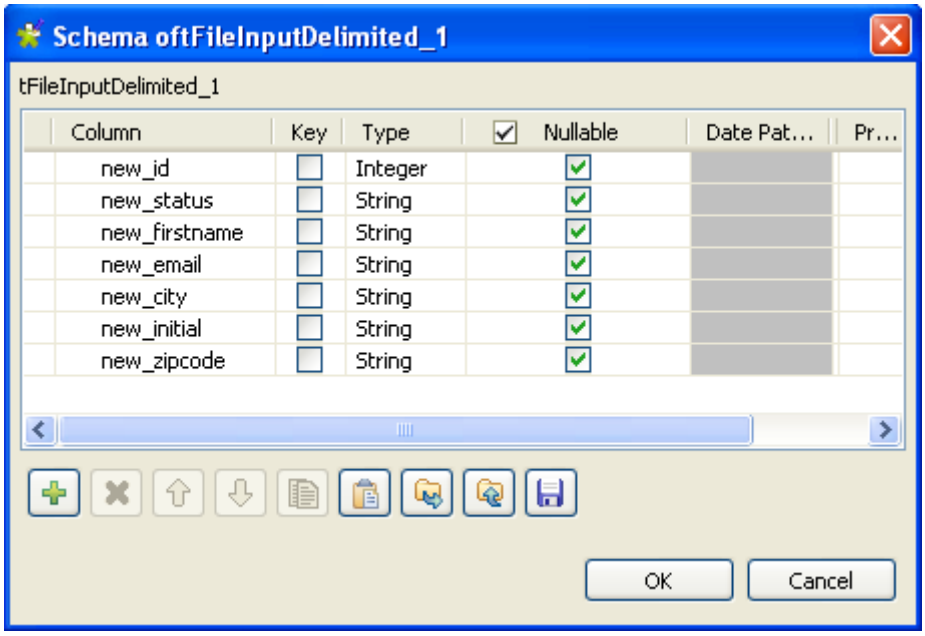
- Double-click **tFileInputDelimited** to display its **Basic settings** view and define its properties



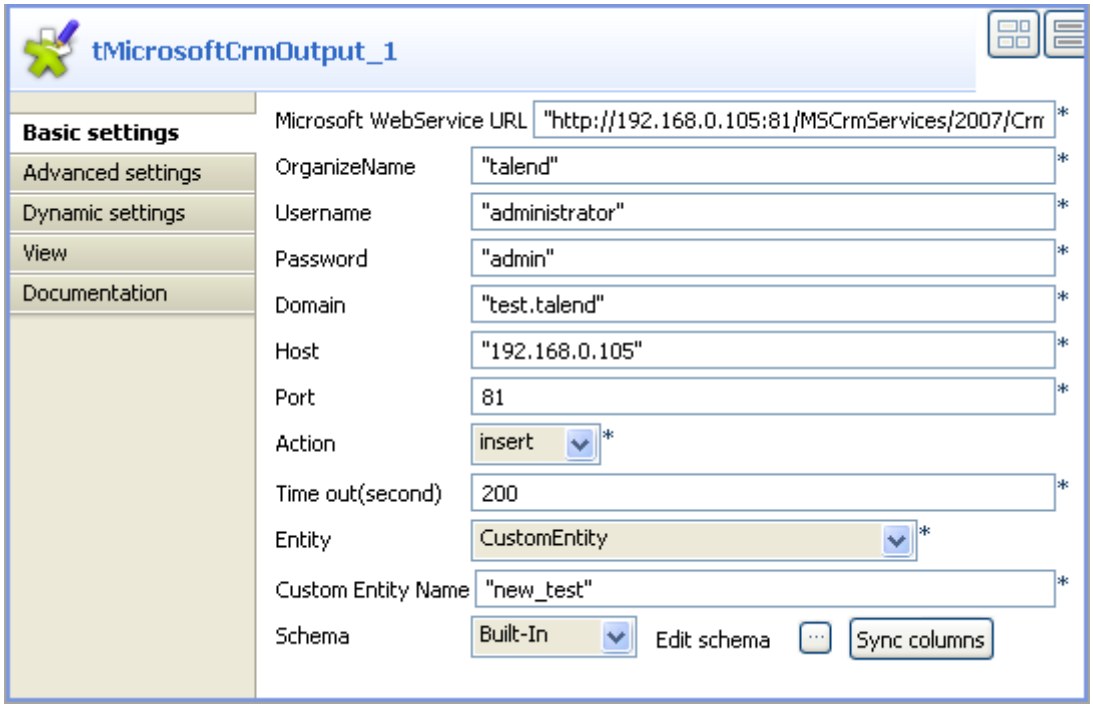
- Set the **Property Type** to **Repository** if you have stored the input file properties centrally in the **Metadata** node in the **Repository** tree view. Otherwise, select **Built-In** and fill the fields that follow manually. In this example, property is set to **Built-In**.
- Click the three-dot button next to the **File Name/Input Stream** field and browse to the delimited file that holds the input data. The input file in this example contains the following columns: *new_id*, *new_status*, *new_firstname*, *new_email*, *new_city*, *new_initial* and *new_zipcode*.

```
new_id;new_status;new_firstname;new_email;new_city;new_initial;new_zipcode
1;married;Paul;pnewman@comp.com;New York;P.M;55677
2;single;Raul;pnewman@comp.com;New York;R.L;55677
3;single;Mary;mnewman@comp.com;Chicago;M.B;66898
4;married;John;Jnewman@comp.com;Chicago,J.M;66898
5;single;Martin;mnewman@comp.com;Sunnyvale,M.P;33662
6;married;Janet;jnewman@comp.com;Sunnyvale,J.P;33662
7;married;Harry;hnewman@comp.com;New York;H.M;55677
8;married;Jerry;jnewman@comp.com;New York;J.M;55677
9;married;Alice;anewman@comp.com;New York;A.M;55677
10;single;Jack;jnewman@comp.com;New York;J.M;55677
```

- In the **Basic settings** view, define the **Row Separator** allowing to identify the end of a row. Then define the **Field Separator** used to delimit fields in a row.
- If needed, define the header, footer and limit number of processed rows in the corresponding fields. In this example, the header, footer and limits are not set.
- Click **Edit schema** to open a dialog box where you can define the input schema you want to write in Microsoft CRM database.



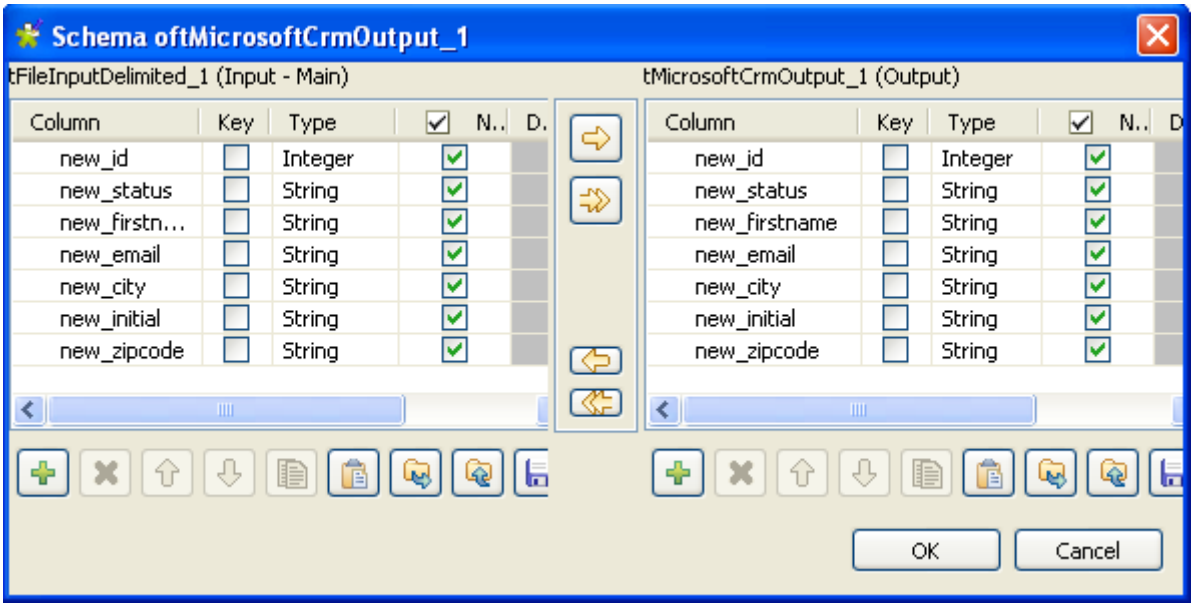
- Click **OK** to close the dialog box.
- Double-click **tMicrosoftCRMOutput** to display the component **Basic settings** view and define its properties.



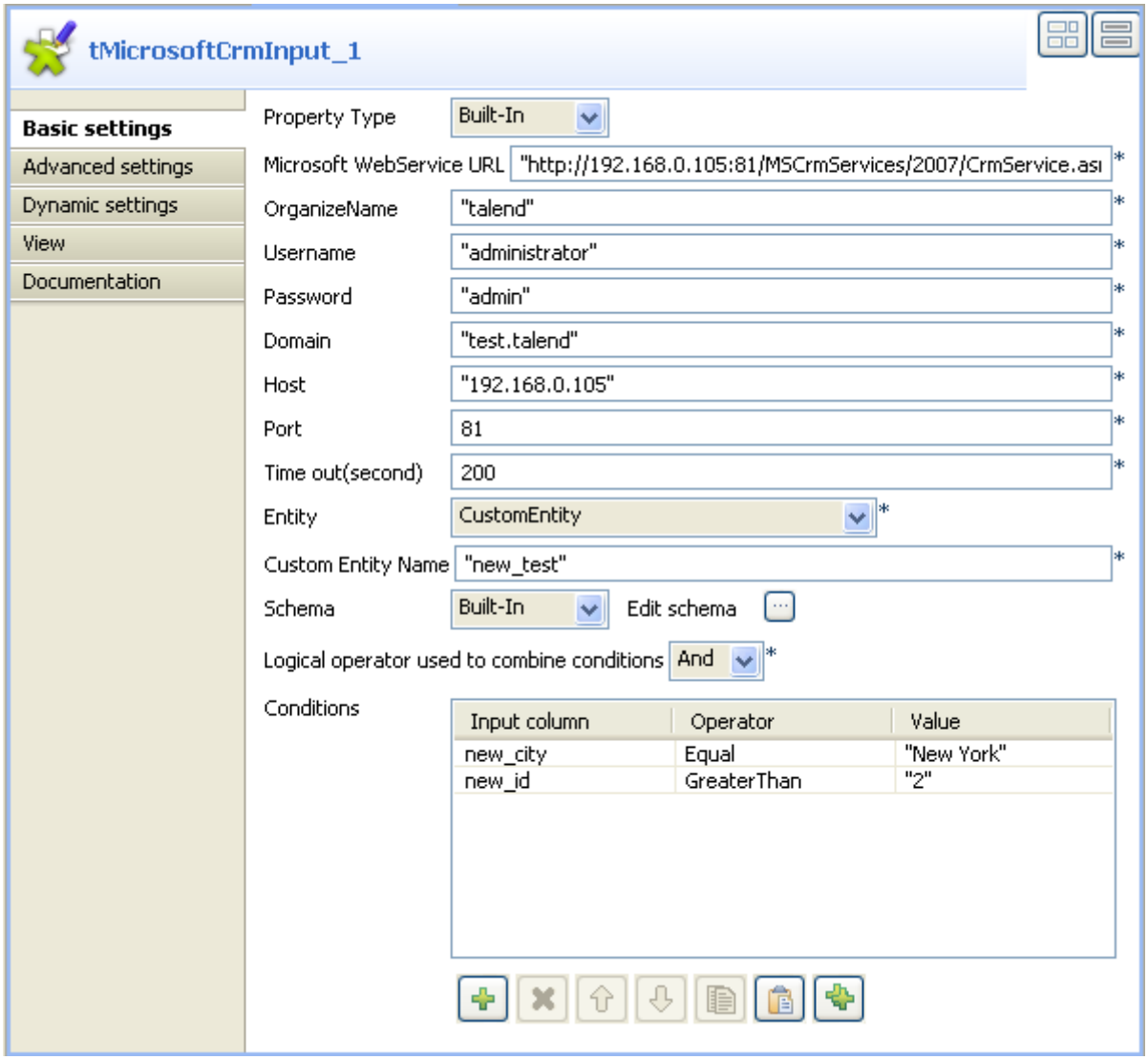
- Enter the Microsoft Web Service URL as well as the user name and password in the corresponding fields.
- In the **OrganizeName** field, enter the name that is given the right to access the Microsoft CRM database.
- In the **Domain field**, enter the domain name of the server on which Microsoft CRM is hosted, and then enter the host IP address and the listening port number in the corresponding fields.

- In the **Action** list, select the operation you want to carry on. In this example, we want to insert data in a custom entity in Microsoft CRM.
- In the **Time out** field, set the amount of time (in seconds) after which the Job will time out.
- In the **Entity** list, select one among those offered. In this example, *CustomEntity* is selected.

  If *CustomEntity* is selected, a **Custom Entity Name** field displays where you need to enter a name for the custom entity.
- The **Schema** is then automatically set according to the entity selected. If needed, click **Edit schema** to display a dialog box where you can modify this schema and remove the columns that you do not need in the output.
- Click **Sync columns** to retrieve the schema from the preceding component.

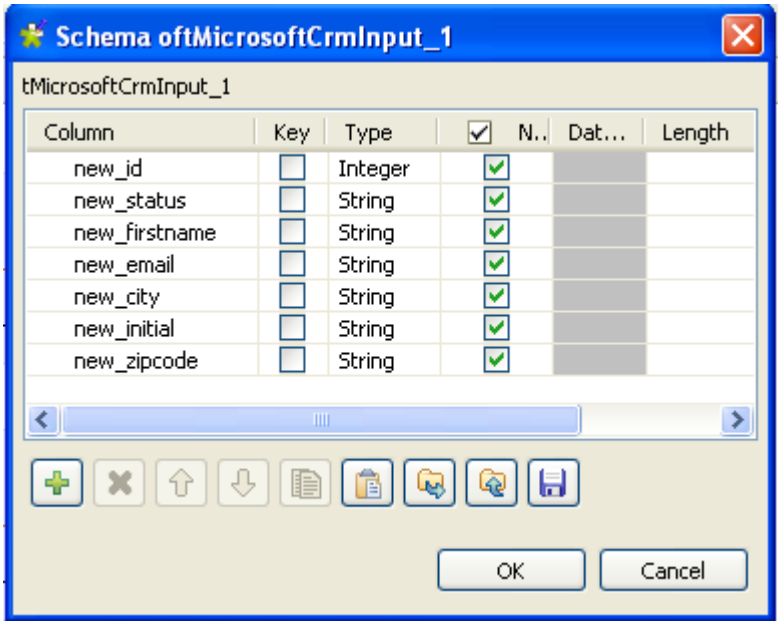


- Double-click **tMicrosoftCRMInput** to display the component **Basic settings** view and define its properties.

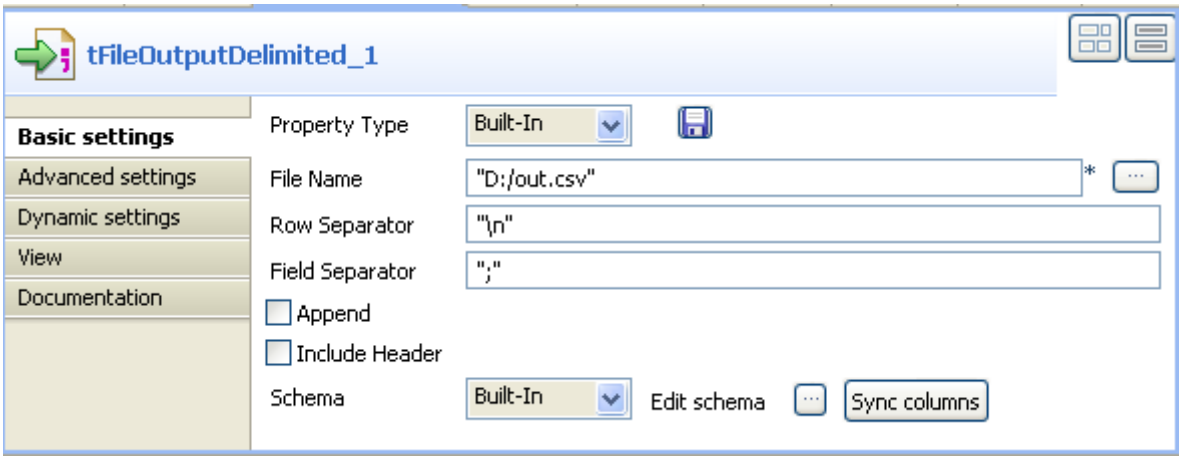


- Set the **Property Type** to **Repository** if you have stored the input file properties centrally in the **Metadata** node in the **Repository** tree view. Otherwise, select **Built-In** and fill the fields that follow manually. In this example, property is set to **Built-In**.
- Enter the Microsoft Web Service URL as well as the user name and password in the corresponding fields.
- In the **OrganizeName** field, enter the name that is given the right to access the Microsoft CRM database.
- In the **Domain field**, enter the domain name of the server on which Microsoft CRM is hosted, and then enter the host IP address and the listening port number in the corresponding fields.
- In the **Time out** field, set the amount of time (in seconds) after which the Job will time out.
- In the **Entity** list, select the one among those offered you want to connect to. In this example, *CustomEntity* is selected.

- The **Schema** is then automatically set according to the entity selected. But you can modify it according to your needs. In this example, you should set the schema manually since you want to access a custom entity. Copy the seven-column schema from **tMicrosoftCRMOutput** and paste it in the schema dialog box in **tMicrosoftCRMInput**.



- Click **OK** to close the dialog box. You will be prompted to propagate changes. Click **Yes** in the popup message.
- In the **Basic settings** view, select **And** or **Or** as the logical operator you want to use to combine the conditions you set on the input columns. In this example, we want to set two conditions on two different input columns and we use **And** as the logical operator.
- In the **Condition** area, click the plus button to add as many lines as needed and then click in each line in the **Input column** list and select the column you want to set condition on. In this example, we want to set conditions on two columns, *new-city* and *new_id*. We want to extract all customer rows whose city is equal to “New York” and whose id is greater than 2.
- Click in each line in the **Operator** list and select the operator to bind the input column with its value, in this example **Equal** is selected for *new_city* and **Greater Than** for *new_id*.
- Click in each line in the **Value** list and set the column value, `New York` for *new_city* and `2` for *new_id* in this example. You can use a fixed or a context value in this filed.
- Double-click **tFileOutputdelimited** to display the component **Basic settings** view and define its properties.



- Set **Property Type** to **Built-In** and then click the three-dot button next to the **File Name** field and browse to the output file.
- Set row and field separators in the corresponding fields.
- Select the **Append** check box if you want to add the new rows at the end of the records.
- Select the **Include Header** check box if the output file includes a header.
- Click **Sync columns** to retrieve the schema from the preceding component.
- Save the Job and press **F6** to execute it.

	A	B	C	D	E
1	7;married;Harry;hnewman@comp.com;New York;H.M;55677				
2	8;married;Jerry;jnewman@comp.com;New York;J.M;55677				
3	9;married;Alice;anewman@comp.com;New York;A.M;55677				
4	10;single;Jack;jnewman@comp.com;New York;J.M;55677				
5					
6					

Only customers who live in New York city and those whose “id” is greater than 2 are listed in the output file you stored locally.

tMicrosoftCRMOutput

tMicrosoftCRMOutput Properties

Component family	Business	
Function	Writes in an entity of a Microsoft CRM database via the relevant webservice.	
Purpose	Allows to write data into a MicrosoftCRM DB.	
Basic settings	*Microsoft Webservice URL*	Type in the webservice URL to connect to the Microsoft CRM DB.
	Organizename	Enter the name of the organization that needs to access the MicrosoftCRM database
	Username and *Password*	Type in the Webservice user authentication data.
	Domain	Type in the domain name of the server that installs MicrosoftCRM.
	Host	Type in the IP address of Microsoft CRM database server.
	Port	Listening port number of Microsoft CRM database server.
	Action	Select in the list the action you want to do on the CRM data. Available actions are: **insert**, **update**, and **delete**.
	Time out (seconds)	Number of seconds for the port to listen before closing.
	Entity	Select the relevant entity in the list.
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the Repository. Click **Edit Schema** to make changes to the schema. Note that if you make changes, the schema automatically becomes built-in. Click **Sync columns** to retrieve the schema from the previous component connected in the Job.
Advanced settings	*tStatCatcher Statistics*	Select this check box to gather the job processing metadata at a job level as well as at each component level.
Usage	Used as an output component. An Input component is required.	
Limitation	n/a	

Related Scenario

For a related use case, see *Scenario: Writing data in a Microsoft CRM database and putting conditions on columns to extract specified rows on page 48*.

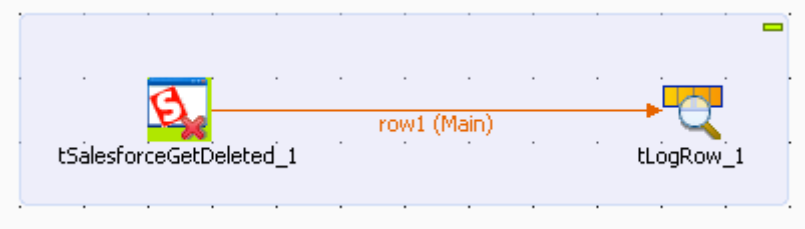
tSalesforceGetDeleted

tSalesforceGetDeleted properties

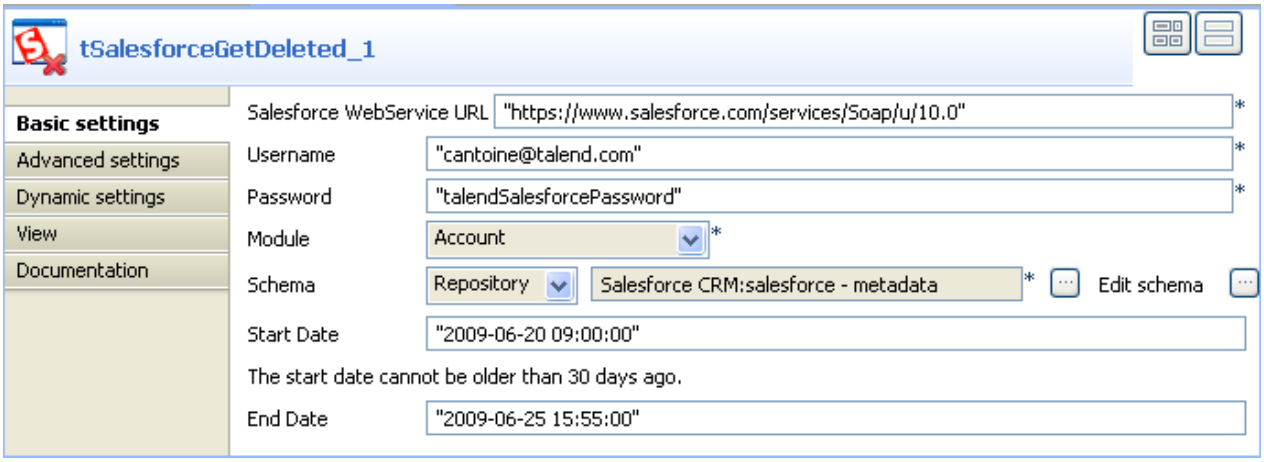
Component family	Business	
Function	**tSalesforceGetDeleted** recovers deleted data from a Salesforce object over a given period of time.	
Purpose	This component can collect the deleted data from a Salesforce object during a specific period of time.	
Basic settings	*Salesforce Webservice URL*	Type in the webservice URL to connect to the Salesforce DB.
	Username and *Password*	Type in the Webservice user authentication data.
	Module	Select the relevant module in the list.  if you select the **Use Custom module** option, you display the **Custom Module Name** field where you can enter the name of the module you want to connect to.
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the Repository. Click **Edit Schema** to make changes to the schema. Note that if you make changes, the schema automatically becomes built-in. Click **Sync columns** to retrieve the schema from the previous component connected in the Job.
	Start Date	Type in between double quotes the date at which you want to start the search. Use the following date format: “yyy-MM-dd HH:mm:ss”.  You can do the search only on the past 30 days.
	End Date	Type in between double quotes the date at which you want to end the search. Use the following date format:“yyy-MM-dd HH:mm:ss”.
Advanced settings	*Use Soap Compression*	Select this check box to activate the SOAP compression.  The compression of SOAP messages optimizes system performance.
	tStatCatcher Statistics	Select this check box to gather the job processing metadata at a job level as well as at each component level.
Usage	You can use this component as an output component. **tSalesforceGetDeleted** requires an input component.	
Limitation	n/a	

Scenario: Recovering deleted data from the Salesforce server

This scenario describes a two-component Job that collects the deleted data over the past 5 days from the Salesforce server.



- Drop **tSalesforceGetDeleted** and **tLogRow** from the **Palette** onto the design workspace.
- Connect the two components together using a **Row** > **Main** link.
- Double-click **tSalesforceGetDeleted** to display its **Basic settings** view and define the component properties.



- In the **Salesforce WebService URL** filed, use the by-default URL of the Salesforce Web service or enter the URL you want to access.
- In the **Username** and **Password** fields, enter your login and password for the Web service.
- From the **Module** list, select the object you want to access, **Account** in this example.
- From the **Schema** list, select **Repository** and then click the three-dot button to open a dialog box where you can select the repository schema you want to use for this component. If you have not defined your schema locally in the metadata, select **Built-in** from the **Schema** list and then click the three-dot button next to the **Edit schema** field to open the dialog box where you can set the schema manually.
- In the **Start Date** and **End Date** fields, enter respectively the start and end dates for collecting the deleted data using the following date format: "*yyyy-MM-dd HH:mm:ss*". You can collect deleted data over the past 30 days. In this example, we want to recover deleted data over the past 5 days.
- Double-click **tLogRow** to display its **Basic settings** view and define the component properties.
- Click **Sync columns** to retrieve the schema from the preceding component.

- In the **Mode** area, select **Vertical** to display the results in a tabular form on the console.
- Save your Job and press **F6** to execute it.

```
Starting job tSalesforceGetDeleted_scenario at 17:17 25/06/2009.

.-------------------------------------------------------.
|                    #1. tLogRow_1                      |
+----------------------+--------------------------------+
| key                  | value                          |
+----------------------+--------------------------------+
| Id                   | 001700000003smNAAR             |
| IsDeleted            | true                           |
| MasterRecordId       | null                           |
| Name                 | sForce                         |
| Type                 | null                           |
| ParentId             | null                           |
| BillingStreet        | The Landmark at One Market     |
| BillingCity          | San Francisco                  |
| BillingState         | CA                             |
| BillingPostalCode    | 94087                          |
| BillingCountry       | US                             |
| ShippingStreet       | null                           |
| ShippingCity         | null                           |
| ShippingState        | null                           |
| ShippingPostalCode   | null                           |
| ShippingCountry      | null                           |
```

Deleted data collected by the **tSalesforceGetDeleted** component is displayed in a tabular form on the console.

tSalesforceGetServerTimestamp

tSalesforceGetServerTimestamp properties

Component family	Business	
Function	**tSalesforceGetServerTimestamp** retrieves the current date of the Salesforce server.	
Purpose	This component retrieves the current date of the Salesforce server presented in a timestamp format.	
Basic settings	*Salesforce Webservice URL*	Type in the webservice URL to connect to the Salesforce DB.
	Username and *Password*	Type in the Webservice user authentication data.
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the Repository. Click **Edit Schema** to make changes to the schema. Note that if you make changes, the schema automatically becomes built-in. Click **Sync columns** to retrieve the schema from the previous component connected in the Job.
Advanced settings	*Use Socks Proxy*	Select this check box if you want to use a proxy server.
	Use Soap Compression	Select this check box to activate the SOAP compression. The compression of the SOAP messages optimizes system performance.
	tStatCatcher Statistics	Select this check box to gather the job processing metadata at a job level as well as at each component level.
Usage	You can use this component as an output component. **tSalesforceGetServerTimestamp** requires an input component.	
Limitation	n/a	

Related scenarios

No scenario is available for this component yet.

tSalesforceGetUpdated

tSalesforceGetUpdated properties

Component family	Business	
Function	**tSalesforceGetUpdated** recovers updated data from a Salesforce object over a given period of time.	
Purpose	This component can collect all updated data from a given Salesforce object during a specific period of time.	
Basic settings	*Salesforce Webservice URL*	Type in the webservice URL to connect to the Salesforce DB.
	Username and *Password*	Type in the Webservice user authentication data.
	Module	Select the relevant module in the list.  if you select the **Use Custom module** option, you display the **Custom Module Name** field where you can enter the name of the module you want to connect to.
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the Repository. Click **Edit Schema** to make changes to the schema. Note that if you make changes, the schema automatically becomes built-in. Click **Sync columns** to retrieve the schema from the previous component connected in the Job.
	Start Date	Type in between double quotes the date at which you want to start the search. Use the following date format: “yyy-MM-dd HH:mm:ss”.  You can do the search only on the past 30 days.
	End Date	Type in between double quotes the date at which you want to end the search. Use the following date format:“yyy-MM-dd HH:mm:ss”.
Advanced settings	*Use Soap Compression*	Select this check box to activate the SOAP compression.  The compression of SOAP messages optimizes system performance.
	tStatCatcher Statistics	Select this check box to gather the job processing metadata at a job level as well as at each component level.
Usage	You can use this component as an output component. **tSalesforceGetUpdate** requires an input component.	
Limitation	n/a	

Related scenarios

No scenario is available for this component yet.

tSalesforceInput

tSalesforceInput Properties

Component family	Business	
Function	**tSalesforceInput** connects to an object of a Salesforce database via the relevant webservice.	
Purpose	Allows to extract data from a Salesforce DB based on a query.	
Basic settings	*Property type*	Either **Built-in** or **Repository**.
		Built-in: No property data stored centrally.
		Repository: Select the repository file where properties are stored. The fields that come after are pre-filled in using the fetched data
	Salesforce Webservice URL	Type in the webservice URL to connect to the Salesforce DB.
	Username and *Password*	Type in the Webservice user authentication data.
	Module	Select the relevant module in the list. if you select the **Use Custom module** option, you display the **Custom Module Name** field where you can enter the name of the module you want to connect to.
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the Repository. Click **Edit Schema** to make changes to the schema. Note that if you make changes, the schema automatically becomes built-in. In this component the schema is related to the Module selected.
	Query condition	Type in the query to select the data to be extracted. Example: account_name= ‘Talend’
	Maunal input of SOQL query	Select this check box to display the **Query** field where you can manually enter the desired query.
Advanced settings	*Batch Size*	Number of lines in each processed batch.
	Use Socks Proxy	Select this check box if you want to use a proxy server.
	Normalize delimited (for child relationship)	Characters, strings or regular expressions used to normaliz the data that is collected by queries set on different hierarchical Salsforce objects.
	Column name delimiter (for child relationship)	XXX

	Use Soap Compression	Select this check box to activate the SOAP compression.  The compression of SOAP messages optimizes system performance, in particular for the batch operations.
	tStatCatcher Statistics	Select this check box to gather the job processing metadata at a job level as well as at each component level.
Usage	Usually used as a Start component. An output component is required.	
Limitation	n/a	

Related scenario

For a related scenario, see *Scenario: Deleting data from the Account object on page 66*.

The operation is similar to the connection to SugarCRM, therefore see scenario of *tSugarCRMInput on page 83* for more information.

tSalesforceOutput

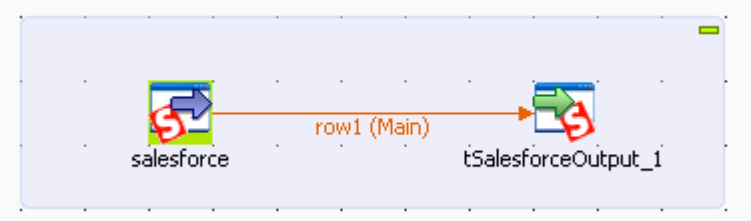
tSalesforceOutput Properties

Component family	Business	
Function	**tSalesforceoutput** writes in an object of a Salesforce database via the relevant webservice.	
Purpose	Allows to write data into a Salesforce DB.	
Basic settings	*Salesforce Webservice URL*	Type in the webservice URL to connect to the Salesforce DB.
	Username and *Password*	Type in the Webservice user authentication data.
	Action	You can do any of the following operations on the data of the Salesforce object: **Insert**: insert data. **Update**: update data. **Delete**: delete data. **Upsert**: update and insert data.
	Module	Select the relevant module in the list.  if you select the **Use Custom module** option, you display the **Custom Module Name** field where you can enter the name of the module you want to connect to.
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the Repository. Click **Edit Schema** to make changes to the schema. Note that if you make changes, the schema automatically becomes built-in. Click **Sync columns** to retrieve the schema from the previous component connected in the Job.
Advanced settings	*Extended Output*	This check box is selected by default. It allows to transfer output data in batches. You can specify the number of lines per batch in the **Rows to commit** field.
	Die on error	Clear this check box to skip the row on error and complete the process for error-free rows.
	Error logging file	If you want to create a file that holds all error logs, click the three-dot button next to this field and browse to the specified file to set its access path and its name.
	Use Socks Proxy	Select this check box if you want to use a proxy server.

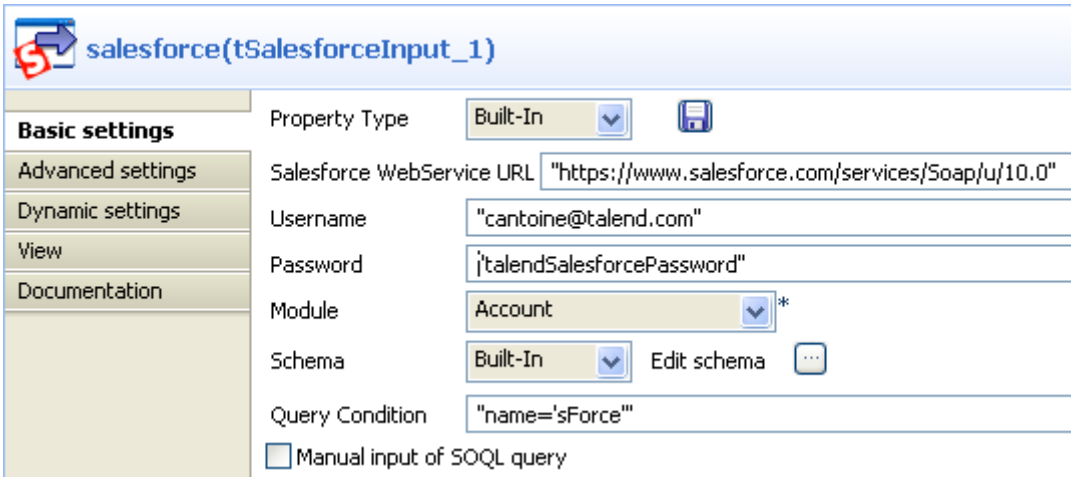
	Use Soap Compression	Select this check box to activate the SOAP compression. The compression of SOAP messages optimizes system performance.
	tStatCatcher Statistics	Select this check box to gather the job processing metadata at a job level as well as at each component level.
Usage	Used as an output component. An Input component is required.	
Limitation	n/a	

Scenario: Deleting data from the Account object

This scenario describes a two-component Job that removes an entry from the Account object.

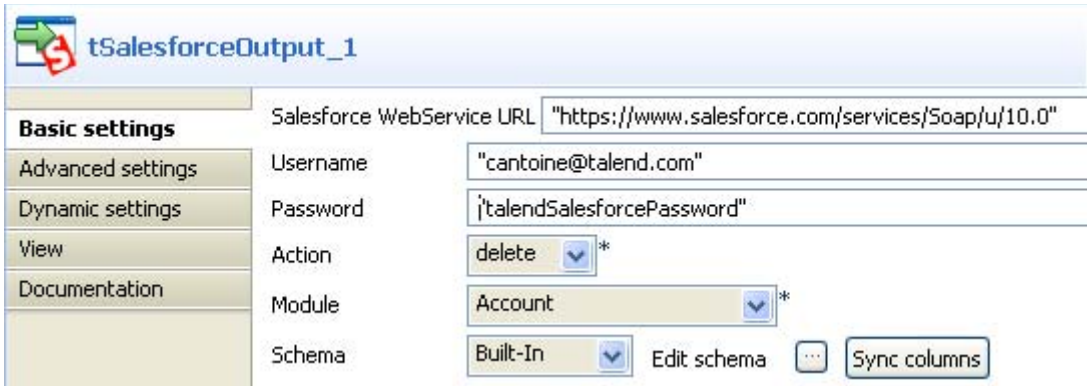


- Drop **tSalesforceInput** and **tSalesforceOutput** from the **Palette** onto the design workspace.
- Connect the two components together using a **Row** > **Main** link.
- Double-click **tSalesforceInput** to display its **Basic settings** view and define the component properties.



- From the **Property Type** list, select **Repository** if you have already stored the connection to the salesforce server in the **Metadata** node of the **Repository** tree view. The property fields that follow are automatically filled in. If you have not defined the server connection locally in the Repository, fill in the details manually after selecting **Built-in** from the **Property Type** list.
  For more information about metadata, see *Managing Metadata*.
- In the **Salesforce WebService URL** field, use the by-default URL of the Salesforce Web service or enter the URL you want to access.

- In the **Username** and **Password** fields, enter your login and password for the Web service.
- From the **Module** list, select the object you want to access, **Account** in this example.
- From the **Schema** list, select **Repository** and then click the three-dot button to open a dialog box where you can select the repository schema you want to use for this component. If you have not defined your schema locally in the metadata, select **Built-in** from the **Schema** list and then click the three-dot button next to the **Edit schema** field to open the dialog box where you can set the schema manually.
- In the **Query Condition** field, enter the query you want to apply. In this example, we want to retrieve the clients whose names are *sForce*. To do this, we use the query: `“name='sForce'”`.
- For a more advanced query, select the Manual input of SOQL query and enter the query manually.
- Double-click **tSalesforceOutput** to display its **Basic settings** view and define the component properties.



- In the **Salesforce WebService URL** field, use the by-default URL of the Salesforce Web service or enter the URL you want to access.
- In the **Username** and **Password** fields, enter your login and password for the Web service.
- From the **Action** list, select the operation you want to carry out. In this example we select **Delete** to delete the *sForce* account selected in the previous component.
- From the **Module** list, select the object you want to access, **Account** in this example.
- Click **Sync columns** to retrieve the schema of the preceding component.
- save your Job and press **F6** to execute it.

Check the content of the **Account** object and verify that the *sForce* account(s) is/are deleted from the server.

tSAPConnection

tSAPConnection properties

Component family	Business	
Function	**tSAPConnection** opens a connection to the SAP system for the current transaction.	
Purpose	**tSAPConnection** allows to commit a whole job data in one go to the SAP system as one transaction.	
Basic settings	*Property type*	Either **Built-in** or **Repository**.
		Built-in: No property data is stored centrally.
		Repository: Select the Repository file where Properties are stored. The fields that follow are pre-filled in using fetched data.
	Connection configuration	**Client type**: enter your usual SAP connection. **Userid :** enter user login. **Password**: enter password. **Language**: specify the language. **Host name**: enter the IP address of the SAP system. **System number**: enter the system number.
Usage	This component is to be used along with other SAP components.	
Limitation	n/a	

Related scenarios

For a related scenarios, see *Scenario 1: Retrieving metadata from the SAP system on page 70* and *Scenario 2: Reading data in the different schemas of the RFC_READ_TABLE function on page 76*.

tSAPInput

tSAPInput Properties

Component family	Business	
Function	**tSAPInput** connects to the SAP system using the system IP address.	
Purpose	**tSAPInput** allows to extract data from an SAP system at any level through calling RFC or BAPI functions.	
Basic settings	*Property type*	Either **Built-in** or **Repository**:
		Built-in: No property data stored centrally.
		Repository: Select the Repository file where Properties are stored. The fields that come after are pre-filled in using the fetched data.
	Use an existing connection	Select this check box and click the relevant DB connection component on the **Component** list to reuse the connection details you already defined.
	Connection configuration	**Client type**: Enter your SAP usual connection code **Userid:** Enter the user connection Id. **Password**: Enter the password. **Language**: Specify a language. **Host name** Enter the SAP system IP address. **System number** Enter the system number.
	Function name	Enter the name of the function you want to use to retrieve data.
	Initialize input	Set input parameters. **Parameter Value**: Enter between inverted commas the value that corresponds to the parameter you set in the **Parameter Name** column. **Type**: Select the type of the input entity to retrieve. **Table Name (Structure Name)**: Enter between inverted commas the table name. **Parameter Name**: Enter between in,verted commas the name of the field that corresponds to the table set in the **Table Name** column.
	Outputs	Configure the parameters of the output schema to select the data to be extracted: **Schema**: Enter the output schema name. **Type (for iterate)**: Select the type of the output entity you want to have. **Table Name (Structure Name)**: Enter between inverted commas the table name. **Mapping**: Enter between inverted commas the name of the field you want to retrieve data from.
Usage	Usually used as a Start component. An output component is required.	
Limitation	n/a	

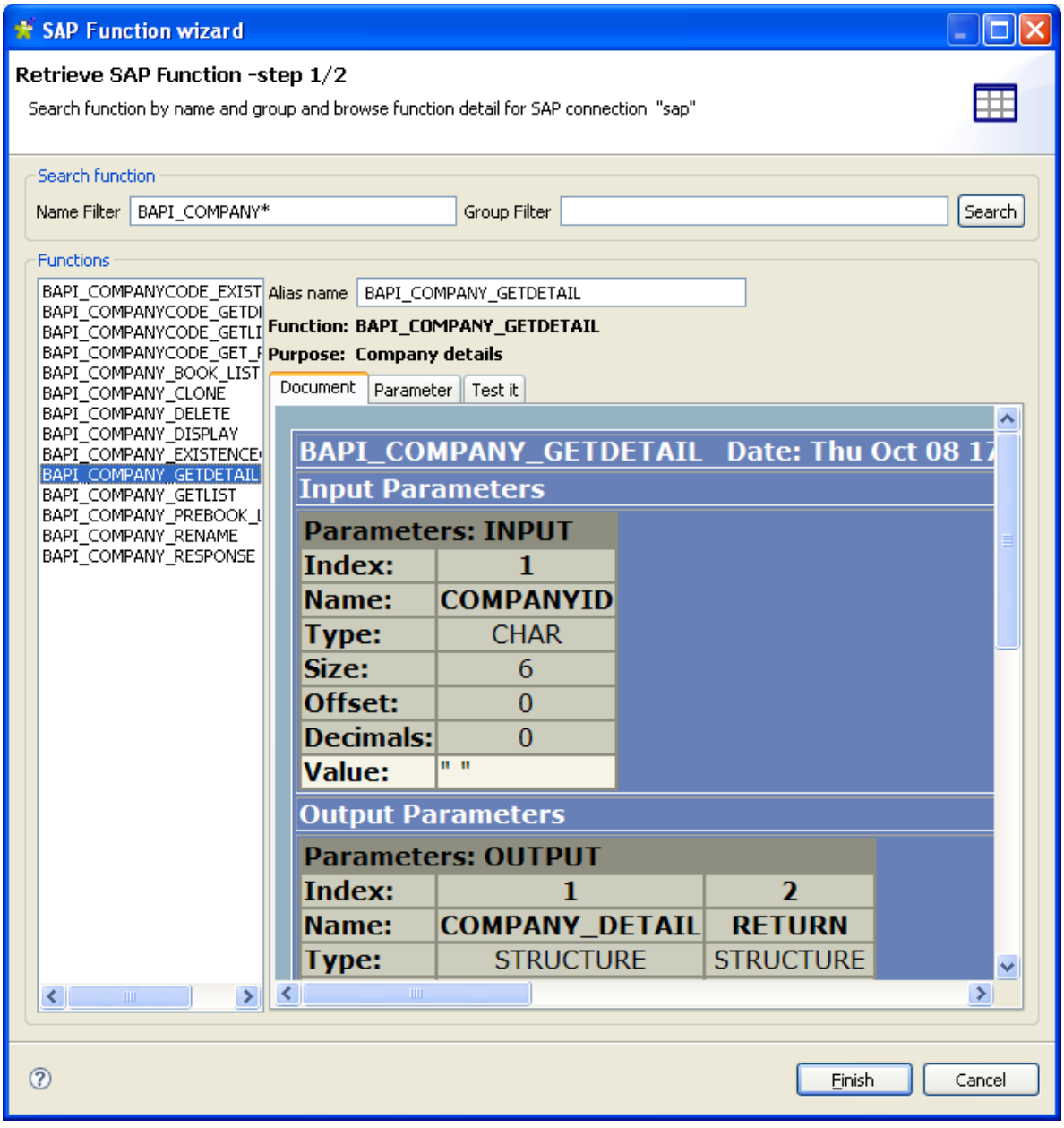
Scenario 1: Retrieving metadata from the SAP system

Talend SAP components (**tSAPInput** and **tSAPOutput**) as well as the SAP wizard are based on a library validated and provided by SAP (JCO) that allows the user to call functions and retrieve data from the SAP system at Table, RFC or BAPI, levels.

This scenario uses the SAP wizard that leads a user through dialog steps to create SAP connection and call RFC and BAPI functions. This SAP wizard is available only for **Talend Integration Suite** users. If you are a user of **Talend Open Studio** or **Talend On Demand**, you need to set the basic settings for the **tSAPInput** component manually.

This java scenario uses the SAP wizard to first create a connection to the SAP system, and then call a BAPI function to retrieve the details of a company from the SAP system. It finally displays in **Talend Open Studio** the company details stored in the SAP system.

The below figure shows the company detail parameters stored in the SAP system and that we want to read in **Talend Open Studio** using the **tSAPInput** component.



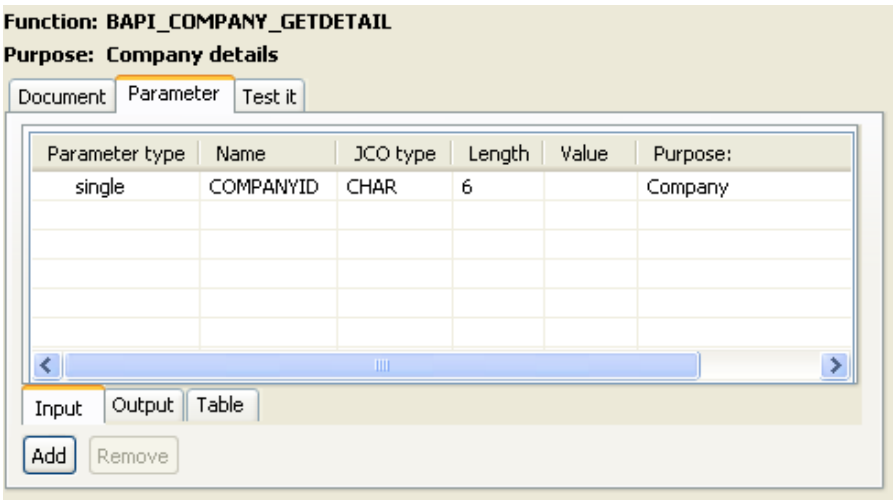
- Create a connection to the SAP system using the SAP connection wizard, in this scenario the SAP connection is called *sap* and is saved in the **Metadata** node.
- Call the BAPI function *BAPI_COMPANY_GETDETAIL* using the SAP wizard to access the BAPI HTML document stored in the SAP system and see the company details.
- In the **Name filter** field, type in *BAPI** and click the **Search** button to display all available BAPI functions.
- Select *BAPI_COMPANY_GETDETAIL* to display the schema that describe the company details.

The three-tab view to the right of the wizard displays the metadata of the *BAPI_COMPANY_GETDETAIL* function and allows you to set the necessary parameters.

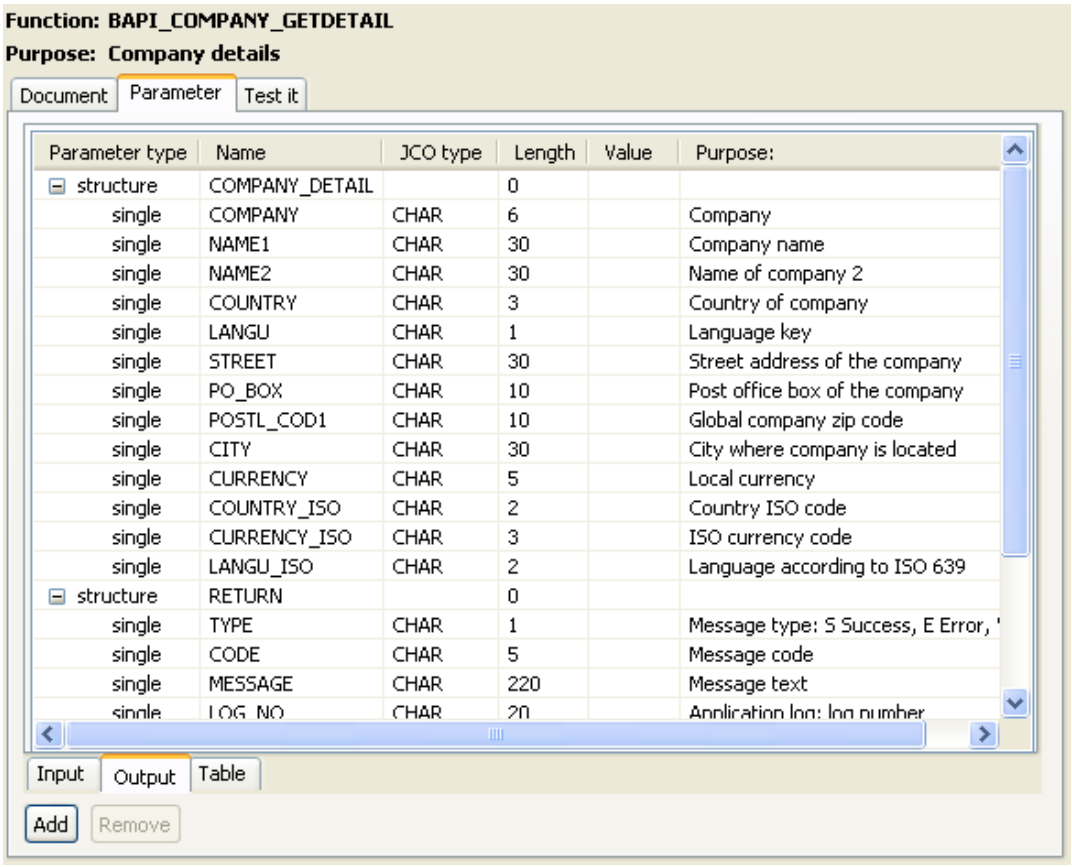
The **Document** view displays the SAP html document about the *BAPI_COMPANY_GETDETAIL* function.

The **Parameter** view provides information about the input and output parameters required by the *BAPI_COMPANY_GETDETAIL* function to return values.

- In the **Parameter** view, click the **Input** tab to list the input parameter(s). In this scenario, there is only one input parameter required by *BAPI_COMPANY_GETDETAIL* and it is called *COMPANYID*.

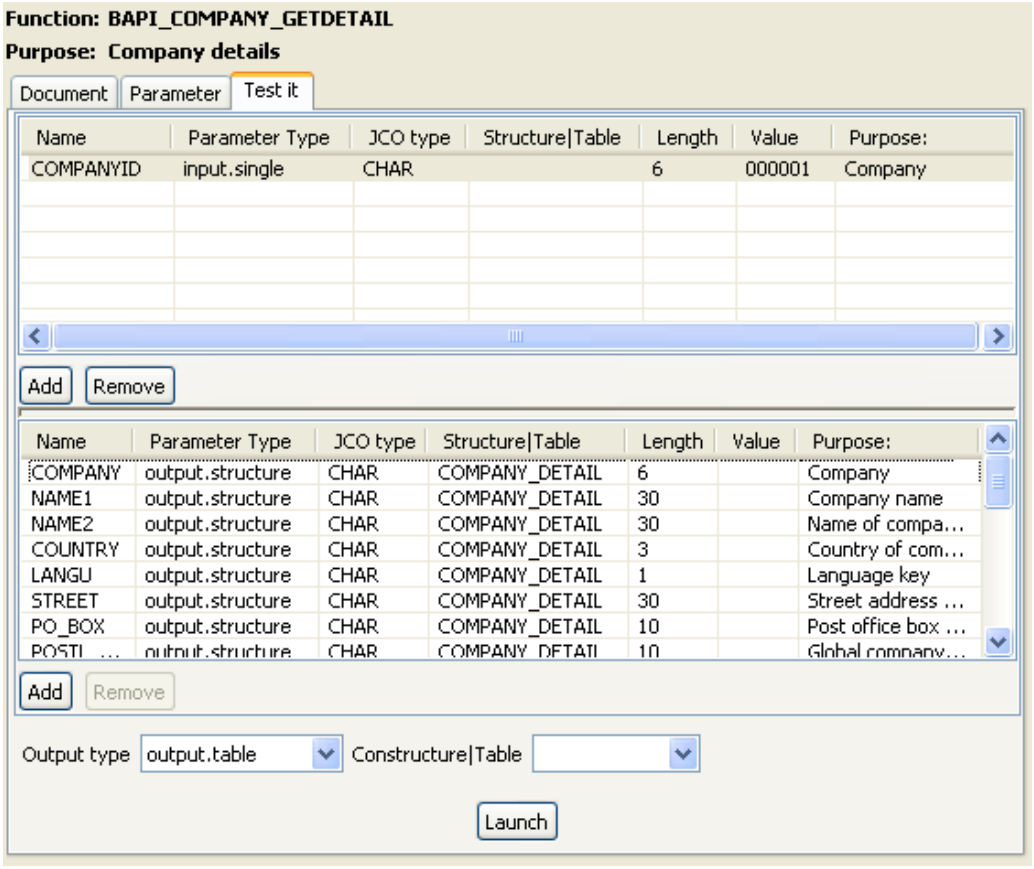


- In the **Parameter** view, click the **Output** tab to list the output parameters returned by *BAPI_COMPANY_GETDETAIL*. In this scenario, there are two output parameters: *COMPANY_DETAIL* and *RETURN*.



Each of these two “structure” parameters consists of numerous “single” parameters.

The **Test it** view allows you to add or delete input parameters according to the called function. In this scenario, we want to retrieve the metadata of the *COMPANY_DETAIL* “structure” parameter that consists of 14 “single” parameters.



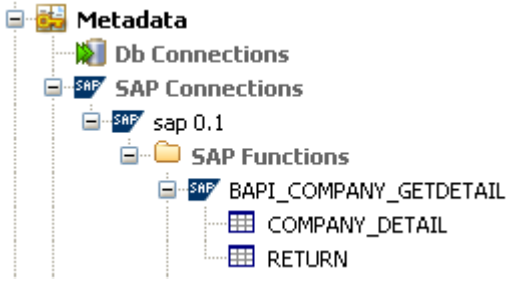
- In the **Value** column of the **COMPANYID** line in the first table, enter "*000001*" to send back company data corresponding to the value 000001.
- In the **Output type** list at the bottom of the wizard, select **output.table**.
- Click **Launch** at the bottom of the view to display the value of each "single" parameter returned by the *BAPI_COMPANY_GETDETAIL* function.
- Click **Finish** to close the wizard and create the connection.

The **sap** connection and the new schema **BAI_COMPANY_GETDETAIL** display under the **SAP Connections** node in the **Repository** tree view.

To retrieve the different schemas of the **BAPI_COMPANY_GETDETAIL** function, do the following:

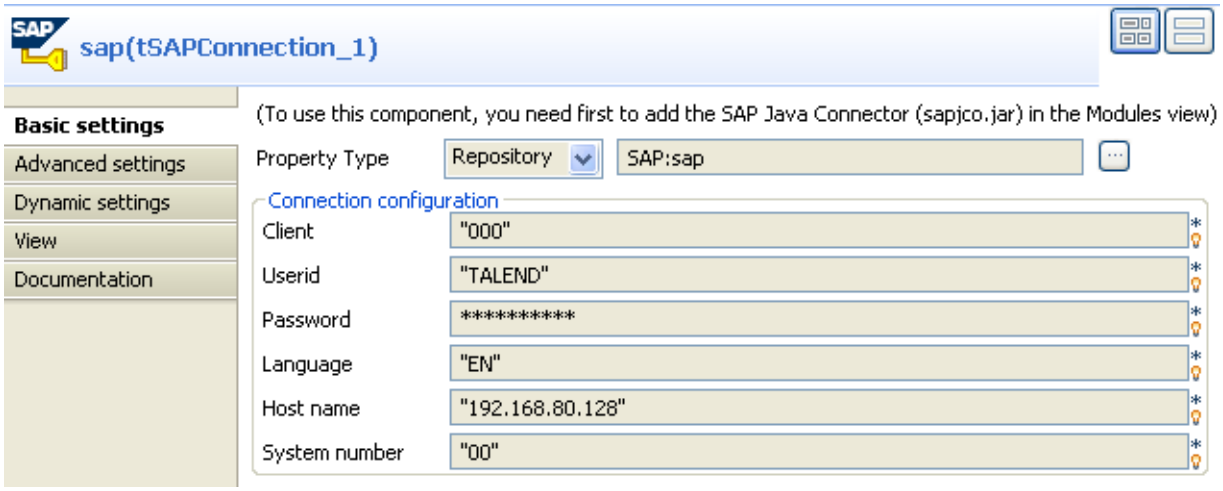
- Right-click **BAPI_COMPANY_GETDETAIL** in the **Repository** tree view and select **Retrieve schema** in the contextual menu.
- In the open dialog box, select the schemas you want to retrieve, **COMPANY_DETAIL** and **RETURN** in this scenario.
- Click **Next** to display the two selected schemas and then **Finish** to close the dialog box.

The two schemas display under the **BAPI_COMPANY_GETDETAIL** function in the **Repository** tree view.



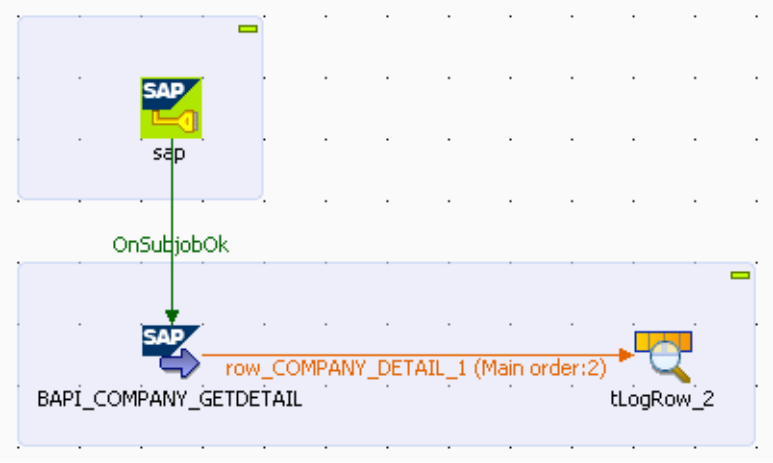
To retrieve the company metadata that corresponds to the 000001 value and display it in **Talend Open Studio**, do the following:

- In the **Repository** tree view, drop the SAP connection you already created to the design workspace to open a dialog box where you can select **tSAPConnection** from the component list and finally click **OK** to close the dialog box. The **tSAPConnection** component holding the SAP connection, *sap* in this example, displays on the design workspace.
- Double-click **tSAPConnection** to display the **Basic settings** view and define the component properties.



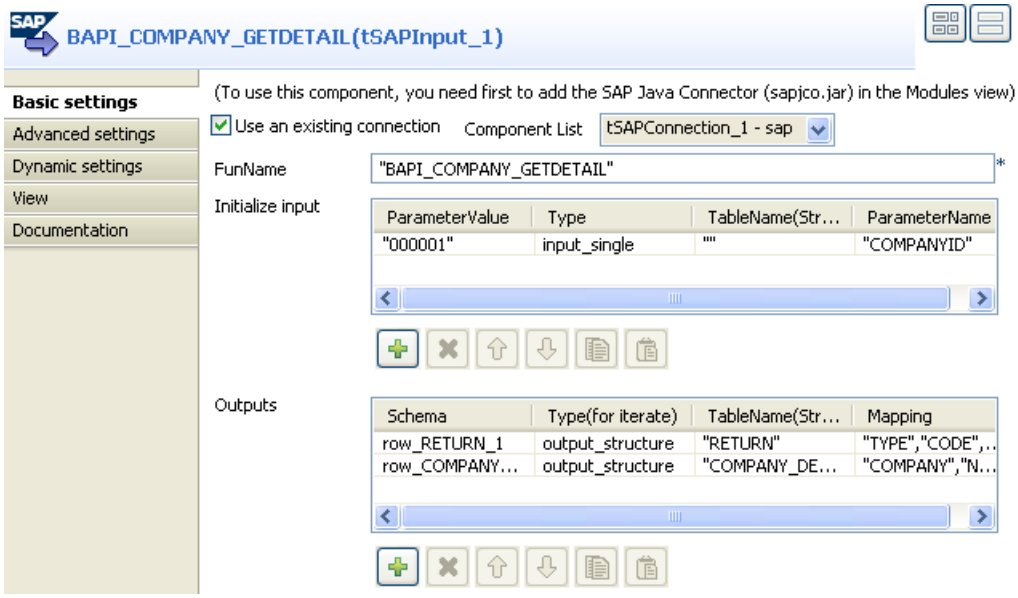
If you store connection details in the **Metadata** node in the **Repository** tree view, the **Repository** mode is selected in the **Property Type** list and the fields that follow are pre-filled. If not, you need to select **Built-in** as “property type” and fill in the connection details manually.

- In the **Repository** tree-view, expand **Metadata** and **sap** in succession and drop **RFC_READ_TABLE** to the design workspace to open a component list.
- Select **tSAPInput** from the component list and click **OK**.
- Drop **tFilterColumns** and **tLogRow** from the **Palette** to the design workspace.
- Connect **tSAPConnection** and **tSAPInput** using a **Trigger** > **OnSubJobOk** link
- To connect **tSAPInput** and **tLogRow**, right-click **tSAPInput** and select **Row** > **row_COMPANY_DETAIL_1** and then click **tLogRow**.



- In the design workspace, double click **tSAPInput** to display its **Basic settings** view and define the component properties**.**

The basic setting parameters for the **tSAPInput** component display automatically since the schema is stored in the **Metadata** node and the component is initialized by the SAP wizard.



- Select the **Use an existing connection** check box and then in the **Component List**, select the relevant **tSAPConnection** component, **sap** in this scenario.

In the **Initialize input** area, we can see the input parameter needed by the *BAPI_COMPANY_GETDETAIL* function.

In the **Outputs** area, we can see all different schemas of the **BAPI_COMPANY_GETDETAIL** function, in particular, **COMPANY_DETAIL** that we want to output.

- In the design workspace, double-click **tLogRow** to display the **Basic settings** view and define the component properties. For more information about this component, see *tLogRow on page 628*.
- Save your Job and press **F6** to execute it.

```
Starting job BAPI_SAP at 18:32 07/10/2009.

[statistics] connecting to socket on port 4100
[statistics] connected
.-----------------------------------.
|           #1. tLogRow_2           |
+--------------+--------------------+
| key          | value              |
+--------------+--------------------+
| COMPANY      | 000001             |
| NAME1        | Gesellschaft G00000|
| NAME2        |                    |
| COUNTRY      | DE                 |
| LANGU        | D                  |
| STREET       | Neurottstrasse 16  |
| PO_BOX       |                    |
| POSTL_COD1   | 69190              |
| CITY         | Walldorf           |
| CURRENCY     | EUR                |
| COUNTRY_ISO  | DE                 |
| CURRENCY_ISO | EUR                |
| LANGU_ISO    | DE                 |
+--------------+--------------------+
[statistics] disconnected
Job BAPI_SAP ended at 18:32 07/10/2009. [exit code=0]
```

tSAPInput retrieved from the SAP system the metadata of the *COMPANY_DETAIL* “structure” parameter and **tLogRow** displayed the information on the console.

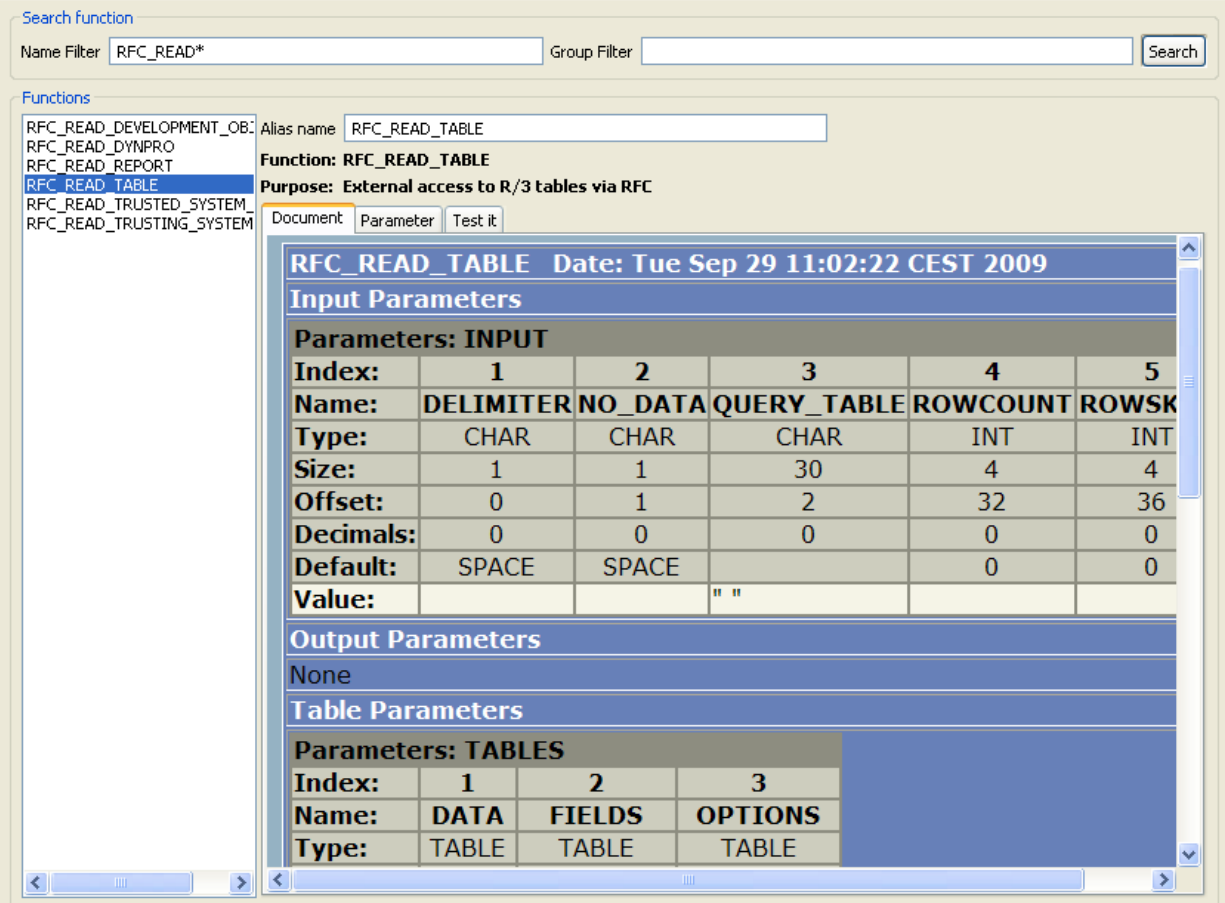
Scenario 2: Reading data in the different schemas of the RFC_READ_TABLE function

Talend SAP components (**tSAPInput** and **tSAPOutput**) as well as the SAP wizard are based on a library validated and provided by SAP (JCO) that allows the user to call functions and retrieve data from the SAP system at Table, RFC or BAPI, levels.

This scenario uses the SAP wizard that leads a user through dialog steps to create a SAP connection and call RFC and BAPI functions. This SAP wizard is available only for **Talend Integration Suite** users. If you are a user of **Talend Open Studio** or **Talend On Demand**, you need to set the basic settings for the **tSAPInput** component manually.

This java scenario uses the SAP wizard to first create a connection to the SAP system, and then call an RFC function to directly read from the SAP system a table called *SFLIGHT*. It finally displays in **Talend Open Studio** the structure of the *SFLIGHT* table stored in the SAP system.

- Create a connection to the SAP system using the SAP connection wizard, in this scenario the SAP connection is called *sap*.
- Call the *RFC_READ_TABLE* RFC function using the SAP wizard to access the table in the SAP system and see its structure.
- In the **Name filter** field, type in *RFC** and click the **Search** button to display all available RFC functions.



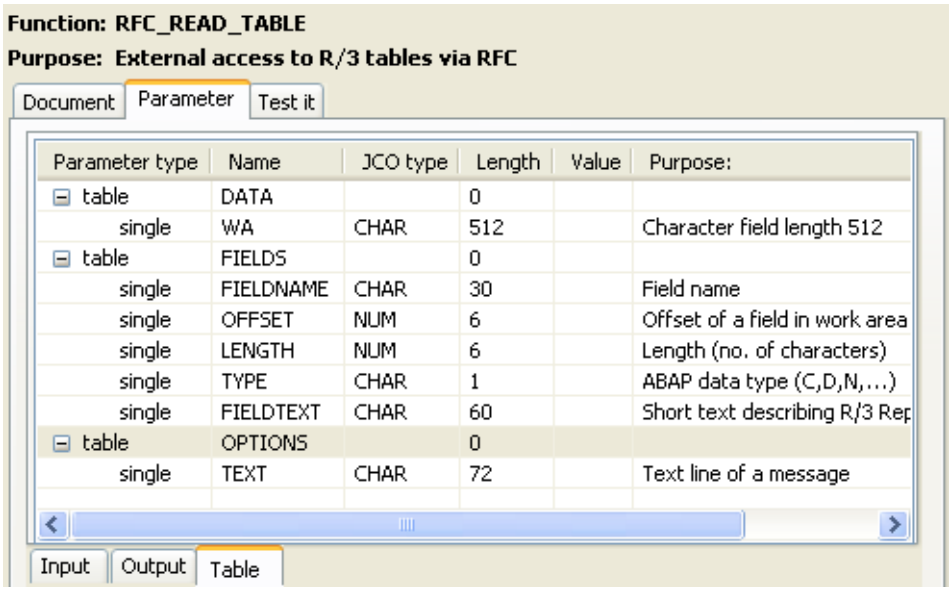
- Select *RFC_READ_TABLE* to display the schema that describe the table structure.

The three-tab view to the right of the wizard displays the metadata of the *RFC_READ_TABLE* function and allows you to set the necessary parameters.

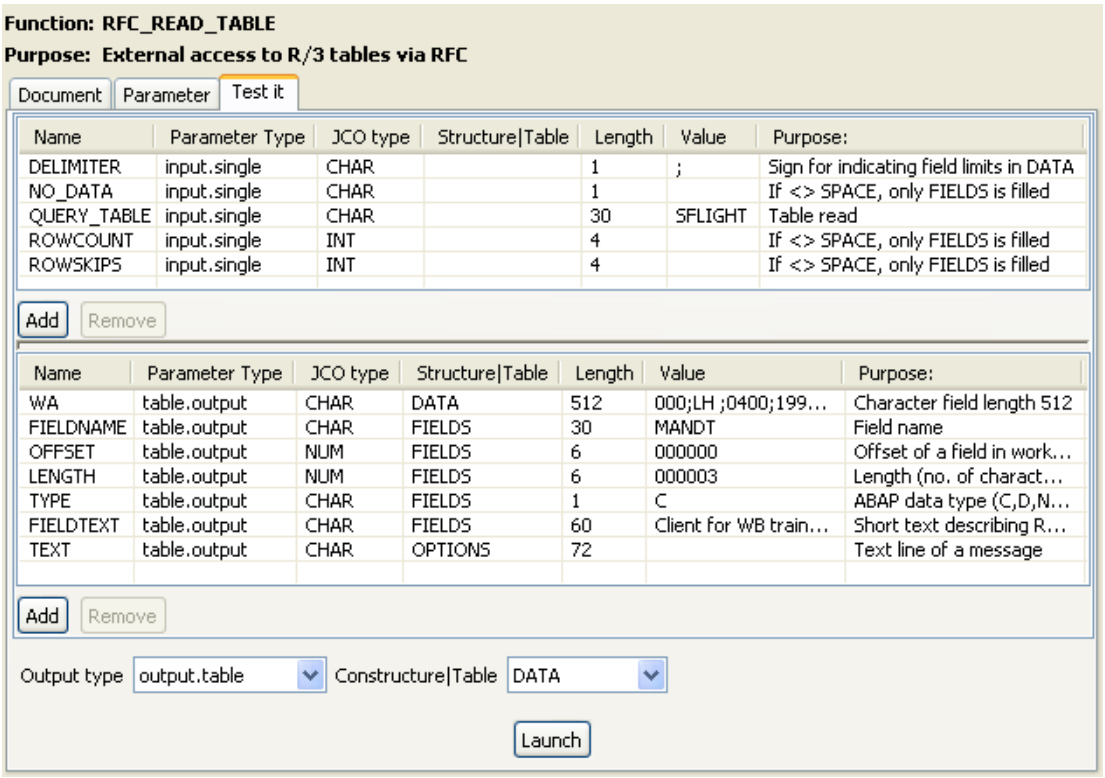
The **Document** view displays the SAP html document about the *RFC_READ_TABLE* function.

The **Parameter** view provides information about the parameters required by the *RFC_READ_TABLE* function to return parameter values.

- In the **Parameter** view, click the **Table** tab to show a description of the structure of the different tables of the *RFC_READ_TABLE* function.



The **Test it** view allows you to add or delete input parameters according to the called function. In this example, we want to retrieve the structure of the *SFLIGHT* table and not any data.



- In the **Value** column of the **DELIMITER** line, enter ";" as field separator.
- In the **Value** column of the **QUERY_TABLE** line, enter *SFLIGHT* as the table to query.
- In the **Output type** list at the bottom of the view, select **output.table**.
- In the **Constructure|Table** list, select **DATA**.
- Click **Launch** at the bottom of the view to display the parameter values returned by the *RFC_READ_TABLE* function. In this example, the delimiter is ";" and the table to read is *SFLIGHT*.

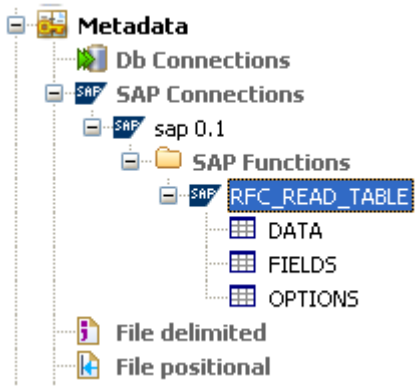
- Click **Finish** to close the wizard and create the connection.

The *sap* connection and the **RFC_READ_TABLE** function display under the **SAPConnections** node in the **Repository** tree view.

To retrieve the different schemas of the **RFC_READ_TABLE** function, do the following:

- In the **Repository** tree view, right-click **RFC_READ_TABLE** and select **Retrieve schema** in the contextual menu. A dialog box displays.
- Select in the list the schemas you want to retrieve, **DATA**, **FIELDS** and **OPTIONS** in this example.
- Click **Next** to open a new view on the dialog box and display these different schemas.
- Click **Finish** to validate your operation and close the dialog box.

The three schemas display under the **RFC_READ_TABLE** function in the **Repository** tree view.



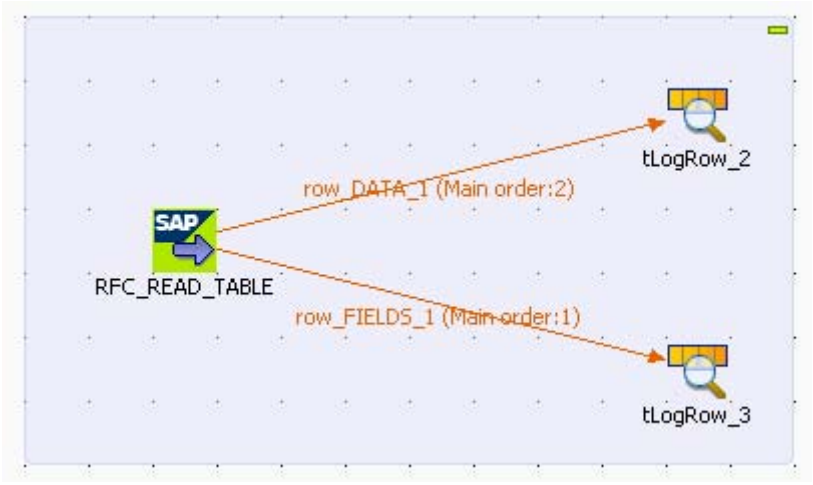
In this example, we want to retrieve the data and column names of the **SFLIGHT** table and display them in **Talend Open Studio**. To do that, proceed as the following:

- In the **Repository** tree view, drop the **RFC_READ_TABLE** function of the *sap* connection to the design workspace to open a dialog box where you can select **tSAPInput** from the component list and then click **OK** to close the dialog box. The **tSAPInput** component displays on the design workspace.
- Drop two **tLogRow** components from the **Palette** to the design workspace.

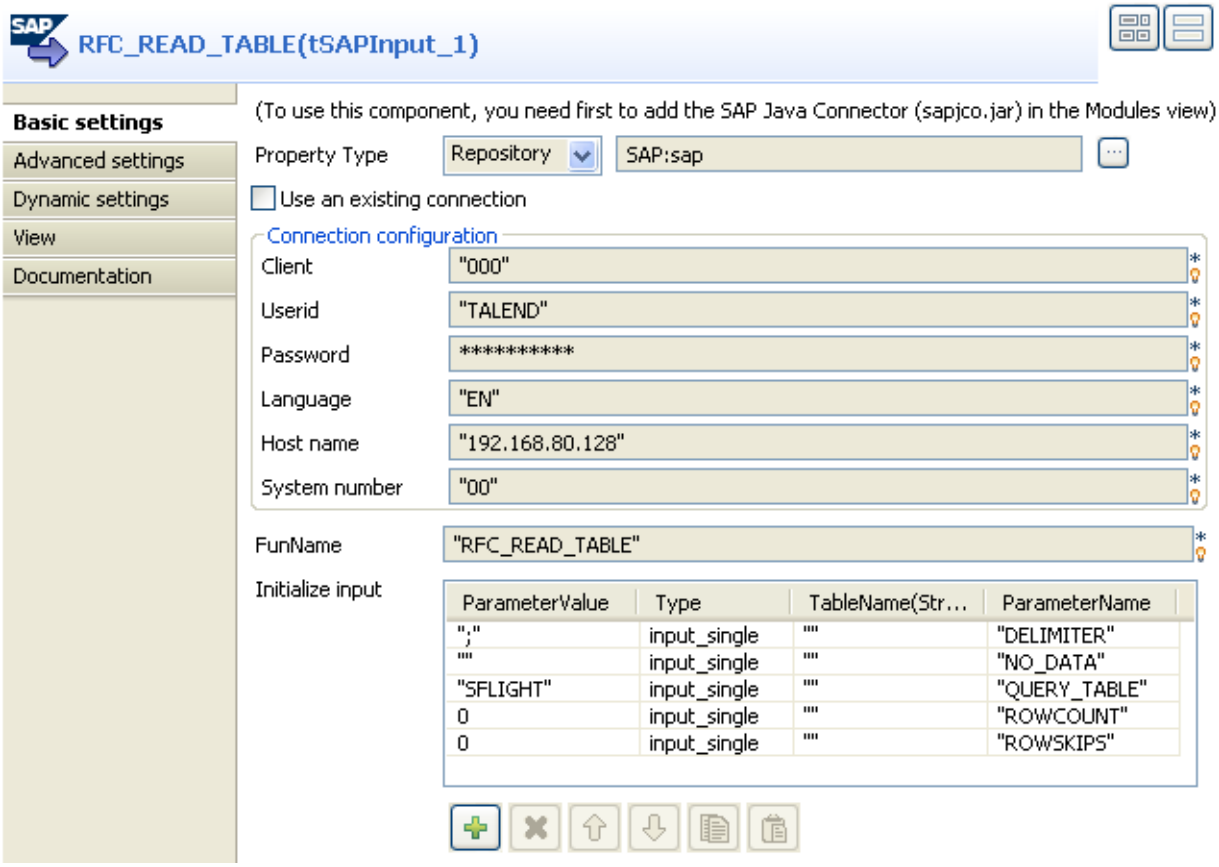
To connect components together:

- Right-click **tSAPInput** and select **Row** > **row_DATA_1** and click the first **tLogRow** component.
- Right-click **tSAPInput** and select **Row** > **row_FIELDS_1** and click the second **tLogRow** components.

In this example, we want to retrieve the **FIELDS** and **DATA** schemas and put them in two different output flows.



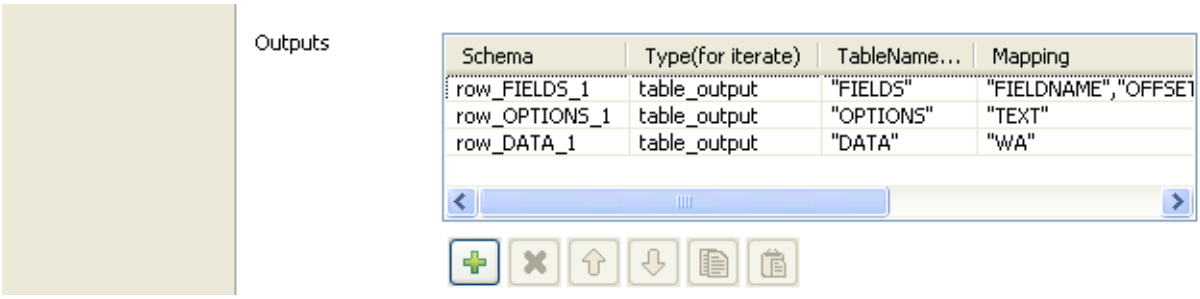
- In the design workspace, double-click **tSAPInput** to open the **Basic settings** view and display the component properties.



The basic setting parameters for the **tSAPInput** component display automatically since the schema is stored in the **Metadata** node and the component is initialized by the SAP wizard.

In the **Initialize input** area, we can see the input parameters necessary for the *RFC_READ_TABLE* function, the field delimiter ";" and the table name "**SFLIGHT"**.

In the **Outputs** area, we can see the different schemas of the **SFLIGHT** table.



- In the design workspace, double click each of the two **tLogRow** components to display the Basic settings view and define the component properties. For more information on the properties of tLogRow, see *tLogRow on page 628*.
- Save your Job and press **F6** to execute it.

```
Starting job jobsap at 16:13 01/10/2009.

[statistics] connecting to socket on port 4093
[statistics] connected
.----------+------+------+----+----------------------------------------.
|                                tLogRow_3                               |
|=---------+------+------+----+----------------------------------------=|
|FIELDNAME |OFFSET|LENGTH|TYPE|FIELDTEXT                               |
|=---------+------+------+----+----------------------------------------=|
|MANDT     |0     |3     |C   |Client for WB train. data model BC_Travel|
|CARRID    |4     |3     |C   |Airline carrier ID                      |
|CONNID    |8     |4     |N   |Flight connection Id                    |
|FLDATE    |13    |8     |D   |Flight date                             |
|PRICE     |22    |15    |P   |Airfare                                 |
|CURRENCY  |38    |5     |C   |Local currency of airline               |
|PLANETYPE |44    |10    |C   |Plane type                              |
|SEATSMAX  |55    |10    |X   |Maximum capacity                        |
|SEATSOCC  |66    |10    |X   |Occupied seats                          |
|PAYMENTSUM|77    |17    |P   |Total of current bookings               |
'----------+------+------+----+----------------------------------------'

.-------------------------------------------------------------------------------.
|                                    tLogRow_2                                  |
|=------------------------------------------------------------------------------=|
|WA                                                                             |
|=------------------------------------------------------------------------------=|
|000;LH ;0400;19950228;  899.00 ;DEM  ;A319      ;5E010000  ;03000000  ; 2639.00|
|000;LH ;0454;19951117;1499.00 ;DEM  ;A319      ;5E010000  ;02000000  ; 2949.00|
|000;LH ;0455;19950606;1090.00 ;USD  ;A319      ;DC000000  ;01000000  ; 1499.00|
|000;LH ;3577;19950428;6000.00 ;LIT  ;A319      ;DC000000  ;01000000  ;  600.00|
|000;SQ ;0026;19950228;  849.00 ;DEM  ;DC-10-10  ;7C010000  ;02000000  ; 1684.00|
'-------------------------------------------------------------------------------'

[statistics] disconnected
Job jobsap ended at 16:13 01/10/2009. [exit code=0]
```

The **tSAPInput** component retrieves from the SAP system the column names of the *SFLIGHT* table as well as the corresponding data. The **tLogRow** components display the information in a tabular form in the Console.

tSAPOutput

tSAPOutput Properties

Component family	Business	
Function	Writes to an SAP system.	
Purpose	Allows to write data into an SAP system.	
Basic settings	*Property type*	Either **Built-in** or **Repository**:
		Built-in: No property data stored centrally.
		Repository: Select the repository file where Properties are stored. The fields that come after are pre-filled in using the fetched data.
	Connection configuration	**Client type**: Enter your SAP usual connection code **Userid:** Enter the user connection Id. **Password**: Enter the password. **Language**: Specify a language. **Host name** Enter the SAP system IP address. **System number** Enter the system number.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the Repository. Click **Edit Schema** to make changes to the schema. Note that if you make changes, the schema automatically becomes built-in.
	Function name	Enter the name of the function you want to use to write data.
	Mapping	Set the parameters to select the data to write to the SAP system.
Usage	Usually used as an output component. An input component is required.	
Limitation	n/a	

Related scenario

For a related scenarios, see *Scenario 1: Retrieving metadata from the SAP system on page 70* and *Scenario 2: Reading data in the different schemas of the RFC_READ_TABLE function on page 76*.

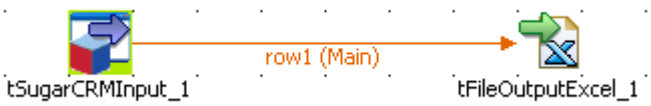
tSugarCRMInput

tSugarCRMInput Properties

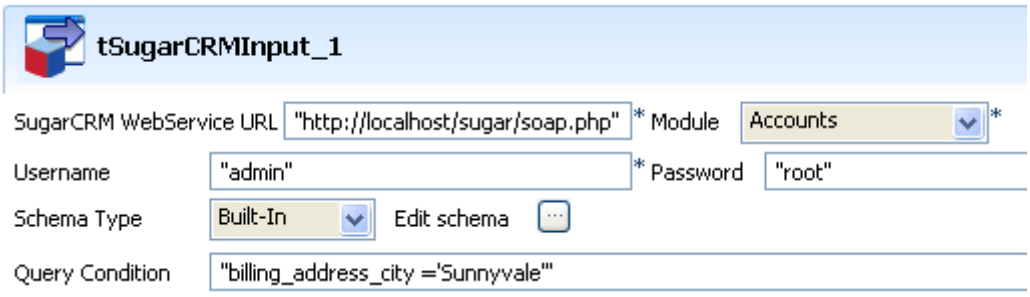
Component family	Business	
Function	Connects to a module of a Sugar CRM database via the relevant webservice.	
Purpose	Allows to extract data from a SugarCRM DB based on a query.	
Basic settings	*SugarCRM Webservice URL*	Type in the webservice URL to connect to the SugarCRM DB.
	Module	Select the relevant module in the list
	Username and *Password*	Type in the Webservice user authentication data.
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the Repository. Click **Edit Schema** to make changes to the schema. Note that if you make changes, the schema automatically becomes built-in. In this component the schema is related to the Module selected.
	Query condition	Type in the query to select the data to be extracted. Example: account_name= 'Talend'
Usage	Usually used as a Start component. An output component is required.	
Limitation	n/a	

Scenario: Extracting account data from SugarCRM

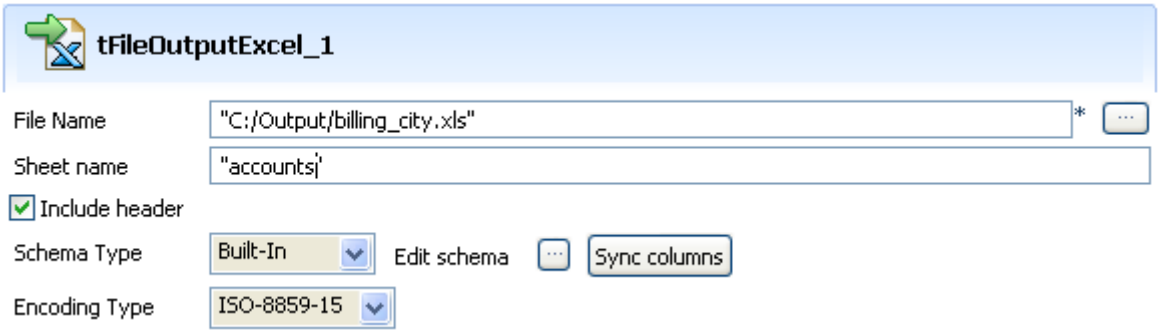
This scenario describes a two-component Job which aims at extracting account information from a SugarCRM database to an Excel output file.



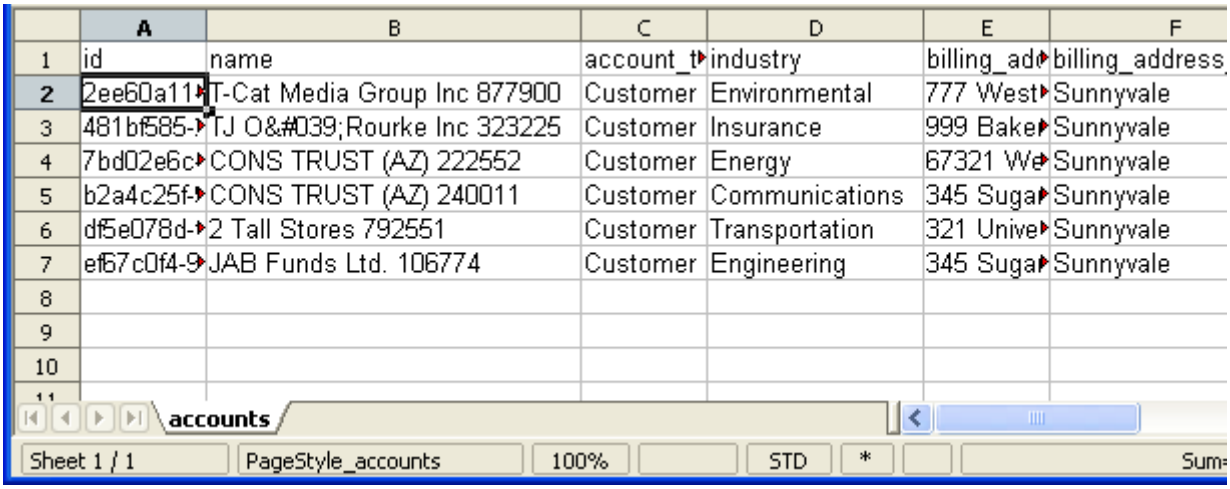
- Drop a **tSugarCRMInput** and a **tFileOutputExcel** component.
- Connect the input component to the output component using a main row link.
- On the **tSugarCRMInput** Component view, fill in the connection information in the SugarCRM Web Service URL as well as the **Username** and **Password** fields
- Then select the **Module** in the list of modules offered. In this example, *Accounts* is selected.



- The **Schema** is then automatically set according to the module selected. But you can change it and remove the columns that you don't require in the output.
- In the **Query Condition** field, type in the query you want to extract from the CRM. In this example: "billing_address_city='Sunnyvale'"
- Then select the **tFileOutputExcel** component.



- Set the destination file name as well as the **Sheet** name and select the **Include header** check box.
- Save the Job and press **F6** to run it.



	A	B	C	D	E	F
1	id	name	account_t	industry	billing_ad	billing_address_
2	2ee60a11	T-Cat Media Group Inc 877900	Customer	Environmental	777 West	Sunnyvale
3	481bf585-	TJ O'Rourke Inc 323225	Customer	Insurance	999 Bake	Sunnyvale
4	7bd02e6c	CONS TRUST (AZ) 222552	Customer	Energy	67321 We	Sunnyvale
5	b2a4c25f-	CONS TRUST (AZ) 240011	Customer	Communications	345 Suga	Sunnyvale
6	df5e078d-	2 Tall Stores 792551	Customer	Transportation	321 Unive	Sunnyvale
7	ef67c0f4-9	JAB Funds Ltd. 106774	Customer	Engineering	345 Suga	Sunnyvale
8						
9						
10						

The filtered data is output in the defined spreadsheet of the specified Excel type file.

tSugarCRMOutput

tSugarCRMOutput Properties

Component family	Business	
Function	Writes in a module of a Sugar CRM database via the relevant webservice.	
Purpose	Allows to write data into a SugarCRM DB.	
Basic settings	*SugarCRM Webservice URL*	Type in the webservice URL to connect to the SugarCRM DB.
	Module	Select the relevant module in the list
	Username and *Password*	Type in the Webservice user authentication data.
	Action	Insert or Update the data in the SugarCRM module.
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the Repository. Click **Edit Schema** to make changes to the schema. Note that if you make changes, the schema automatically becomes built-in. Click **Sync columns** to retrieve the schema from the previous component connected in the Job.
Usage	Used as an output component. An Input component is required.	
Limitation	n/a	

Related Scenario

No scenario is available for this component yet.

tVtigerCRMInput

tVtigerCRMInput Properties

Component family	Business/vTigerCRM	
Function	Connects to a module of a vTigerCRM database.	
Purpose	Allows to extract data from a vTigerCRM DB.	
Basic settings	*Server Address*	Type in the IP address of the vTigerCRM server
	Port	Type in the Port number to access the server
	Username and *Password*	Type in the user authentication data.
	Version	Type in the version of vTigerCRM you are using.
	Module	Select the relevant module in the list
	Method	Select the relevant method on the list. The method specifies the action you can carry out on the vTigerCRM module selected.
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the Repository. Click **Edit Schema** to make changes to the schema. Note that if you make changes, the schema automatically becomes built-in. In this component the schema is related to the Module selected.
Usage	Usually used as a Start component. An output component is required.	
Limitation	n/a	

Related Scenario

No scenario is available for this component yet.

tVtigerCRMOutput

tVtigerCRMOutput Properties

Component family	Business/vTigerCRM	
Function	Writes data into a module of a vTigerCRM database.	
Purpose	Allows to write data from a vTigerCRM DB.	
Basic settings	*Server Address*	Type in the IP address of the vTigerCRM server
	Port	Type in the Port number to access the server
	Username and *Password*	Type in the user authentication data.
	Version	Type in the version of vTigerCRM you are using.
	Module	Select the relevant module in the list
	Method	Select the relevant method on the list. The method specifies the action you can carry out on the vTigerCRM module selected.
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the Repository. Click **Edit Schema** to make changes to the schema. Note that if you make changes, the schema automatically becomes built-in. In this component the schema is related to the Module selected.
Usage	Used as an output component. An Input component is required.	
Limitation	n/a	

Related Scenario

No scenario is available for this component yet.

Custom Code components

This chapter details the major components that you can find in **Custom Code** group of the **Palette** of **Talend Open Studio**.

The Custom Code gathers components that covers on-the-fly specific code needs.

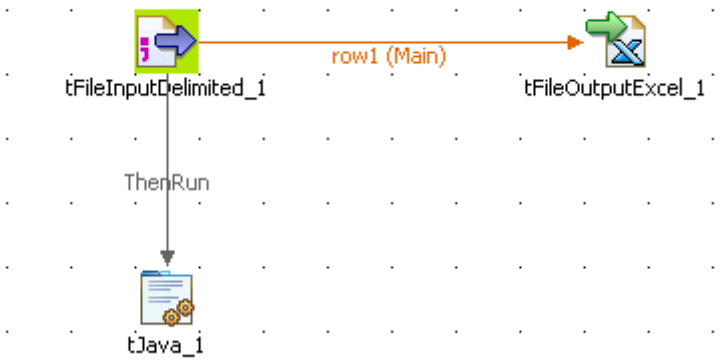
tJava

tJava Properties

Component family	Custom Code	
Function	**tJava** enables you to enter personalized code in order to integrate it in **Talend** program. You can execute this code only once.	
Purpose	**tJava** makes it possible to extend the functionalities of a **Talend** Job through using Java commands.	
Basic settings	*Code*	Type in the Java code you want to execute according to the task you need to perform. For further information about Java functions syntax specific to **Talend**, see **Talend Open Studio** online Help (Help Contents > Developer Guide > API Reference). For a complete Java reference, check http://java.sun.com/javaee/6/docs/api/
Usage	This component is generally used as a one-component subjob.	
Limitation	You should know Java language.	

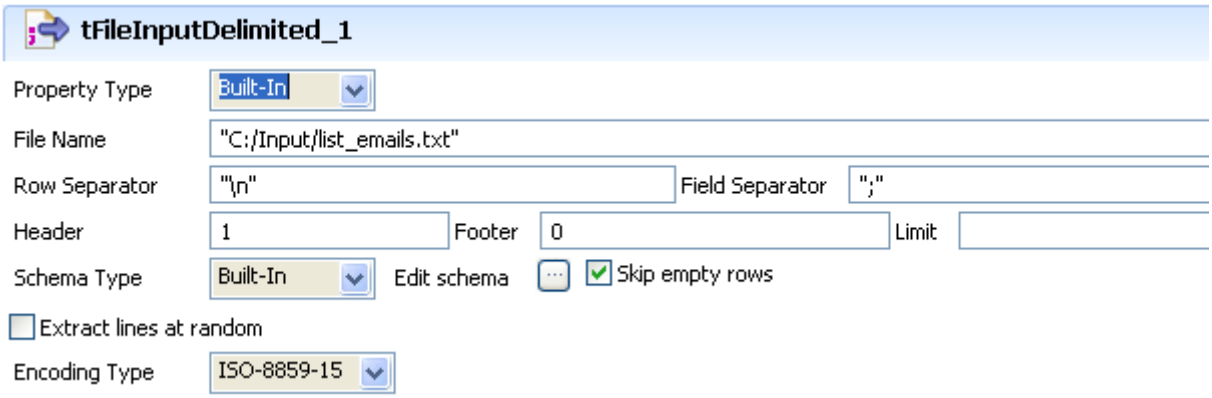
Scenario: Printing out a variable content

The following scenario is a simple demo of the extended application of the **tJava** component. The Job aims at printing out the number of lines being processed using a Java command and the global variable provided in **Talend Open Studio**.

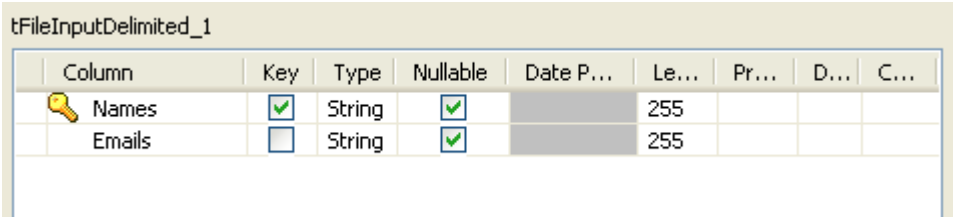


- Select and drop the following components from the **Palette** onto the design workspace: **tFileInputDelimited**, **tFileOutputExcel**, **tJava**.
- Connect the **tFileInputDelimited** to the **tFileOutputExcel** using a **Row Main** connection. The content from a delimited txt file will be passed on through the connection to an xls-type of file without further transformation.

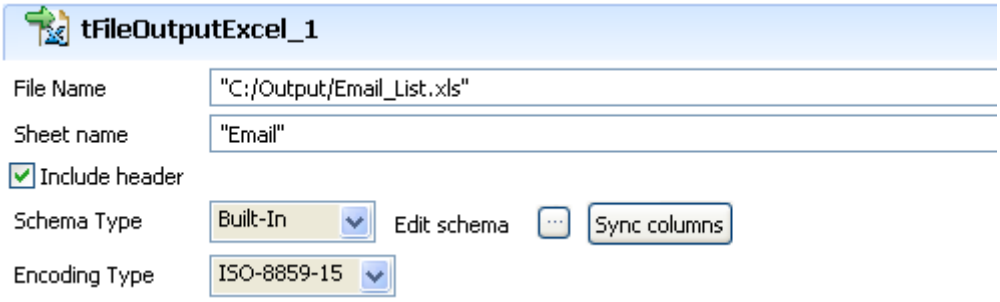
- Then connect the **tFileInputDelimited** component to the **tJava** component using a **Then Run** link. This link sets a sequence ordering **tJava** to be executed at the end of the main process.
- Set the **Basic settings** of the **tFileInputDelimited** component. The input file used in this example is a simple text file made of two columns: *Names* and their respective *Emails*.



- The schema has not been stored in the repository for this use case, therefore you need to set manually the two-column schema
- Click the **Edit Schema** button.

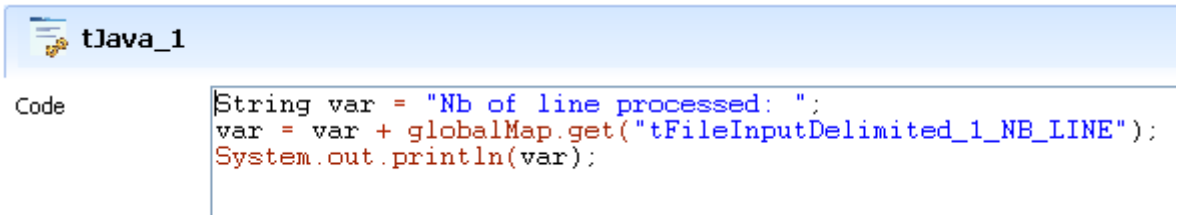


- When prompted, click **OK** to accept the propagation, so that the **tFileOutputExcel** component gets automatically set with the input schema. Therefore no need to set the schema again.
- Set the output file to receive the input content without changes. If the file does not exist already, it will get created.



In this example, the **Sheet name** is *Email* and the **Include Header** box is selected.

- Then select the **tJava** component to set the Java command to execute.



- In the **Code** area, type in the following command:

```
String var = "Nb of line processed: ";
var = var + globalMap.get("tFileInputDelimited_1_NB_LINE");
System.out.println(var);
```

In this use case, we use the *NB_Line* variable. To access the global variable list, press Ctrl + Space bar on your keyboard and select the relevant global parameter.

- Save your Job and press **F6** to execute it.

```
Starting job JavaDb at 13:53 20/08/2007.
Nb of line processed: 4
Job JavaDb ended at 13:53 20/08/2007. [exit code=0]
```

The content gets passed on to the Excel file defined and the Number of lines processed are displayed on the **Run** console.

tJavaFlex

tJavaFlex properties

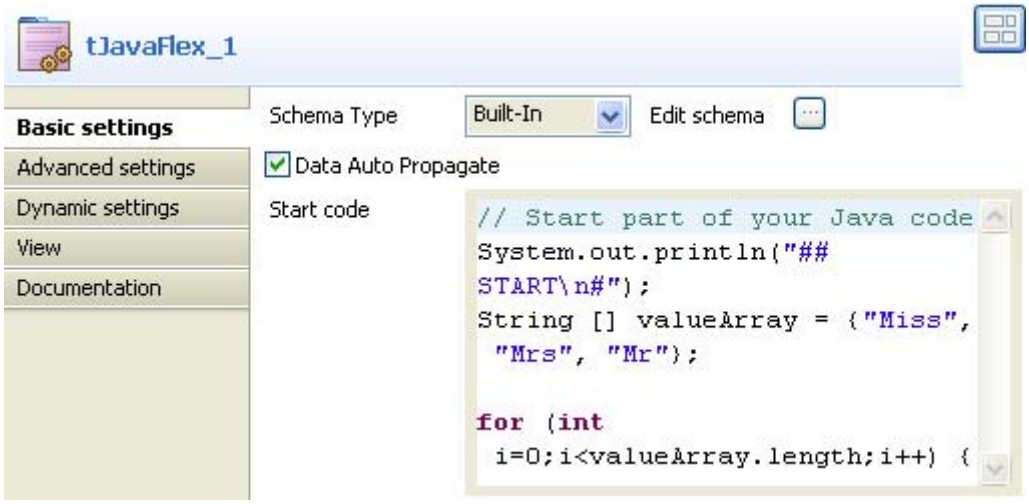
Component family	Custom Code	
Function	**tJavaFlex** enables you to enter personalized code in order to integrate it in **Talend** program. With **tJavaFlex**, you can enter the three java-code parts (start, main and end) that constitute a kind of component dedicated to do a desired operation.	
Objective	**tJava** makes it possible to extend the functionalities of a **Talend** Job through using Java commands.	
Basic settings	*Schema Type* and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either **built-in** or remotely stored in the **Repository**. Click **Edit Schema** to make changes to the schema. Note that if you make changes, the schema automatically becomes built-in.
		Built-in: The schema is created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	Data Auto Propagate	Select this check box to automatically propagate the data to the component that follows.  When you select this check box, you can not later do any transformation on the retrieved data by setting Java commands in the **Main code** field.
	Start code	Enter the Java code that will be called during the initialization phase.
	Main code	Enter the Java code to be applied for each line in the data flow.
	End code	Enter the Java code that will be called during the closing phase.
Advanced settings	*Import*	Enter the Java code that helps to import, if necessary, external libraries used in the **Main code** box of the **Basic settings** view.
	tStatCatcher Statistics	Select this check box to gather the job processing metadata at a job level as well as at each component level.
Usage	You can use this component as a start, intermediate or output component. You can as well use it as a one-component subjob.	

Limitation	You should know the Java language.

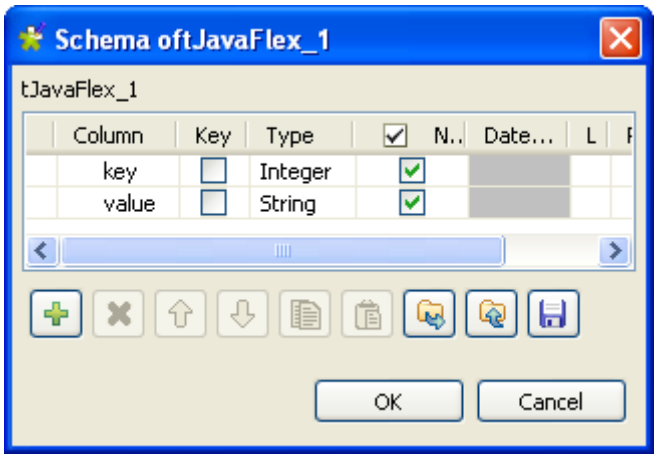
Scenario1: Generating data flow

This scenario describes a two-components Job that generates a three-line data flow describing different personal titles (Miss, Mrs, and Mr) and displaying them on the console.

- Drop **tJavaFlex** and **tLogRow** from the **Palette** onto the design workspace.
- Connect the components together using a **Row Main** link.
- Double-click **tJavaFlex** to display its **Basic settings** view and define its properties.



- From the **Schema Type** list, select **Built-in** and then click the three-dot button next to **Edit schema** to open the corresponding dialog box where you can define the data structure to pass to the component that follows.



- Click the plus button to add two columns: *key* and *value* and then set their types to **Integer** and **String** respectively.
- Click **Ok** to validate your changes and close the dialog box.
- In the **Basic settings** view of **tJavaFlex**, select the **Data Auto Propagate** check box to automatically propagate data to the component that follows.
  In this example, we do not want to do any transformation on the retrieved data.
- In the **Start code** field, enter the code to be executed in the initialization phase.
  In this example, the code indicates the initialization of **tJavaFlex** by displaying the START message and sets up the loop and the variables to be used afterwards in the Java code:

```
System.out.println("## START\n#");
String [] valueArray = {"Miss", "Mrs", "Mr"};

for (int i=0;i<valueArray.length;i++) {
```

Main code

```
// Main part of your Java code
(loop)

row1.key = i;
row1.value = valueArray[i];
```

- In the **Main code** field, enter the code you want to apply on each of the data rows.
  In this example, we want to display each key with its value:

```
row1.key = i;
row1.value = valueArray[i];
```

In the Main code," row1" corresponds to the name of the link that comes out of **tJavaFlex**. *If you rename this link, you have to modify the code of this field accordingly.*

- In the **End code** field, enter the code that will be executed in the closing phase.
  In this example, the brace (curly bracket) closes the loop and the code indicates the end of the execution of **tJavaFlex** by displaying the END message:

```
}
System.out.println("#\n## END");
```

End code

```
// End part of your Java code

}
System.out.println("#\n##
END");
```

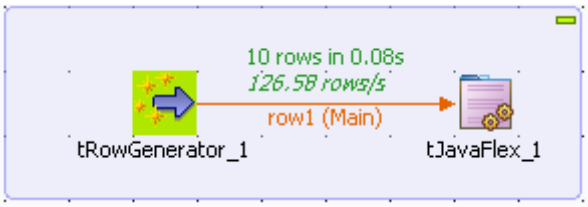
- If needed, double-click **tLogRow** and in its **Basic settings** view, click the button next to **Edit schema** to make sure that the schema has been correctly propagated.
- Save your Job and press **F6** to execute it.

```
Starting job tJavaFlex_scenario1 at 14:49 02/09/2009.

## START
#
0|Miss
1|Mrs
2|Mr
#
## END
Job tJavaFlex_scenario1 ended at 14:49 02/09/2009. [exit code=0]
```

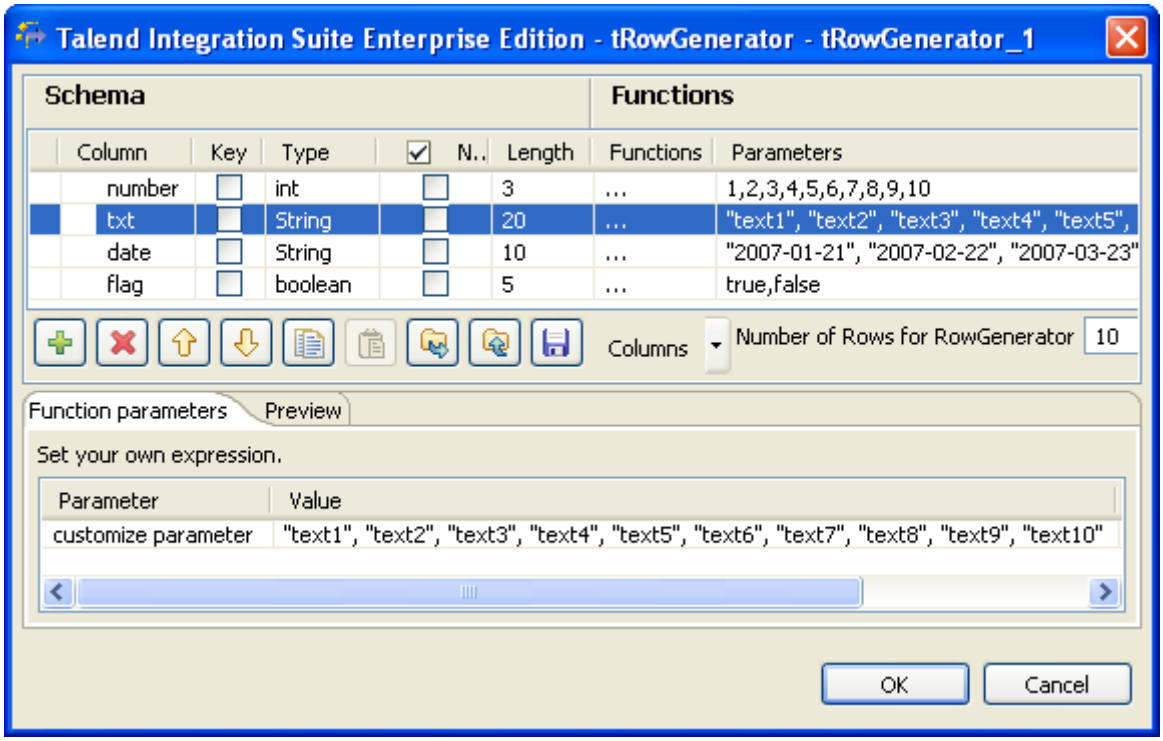
The three personal titles are displayed on the console along with their corresponding keys.

Scenario2: Processing rows of data with tJavaFlex

This scenario describes a two-component Job that generates random data and then collects that data and does some transformation on it line by line using Java code through the **tJavaFlex** component.

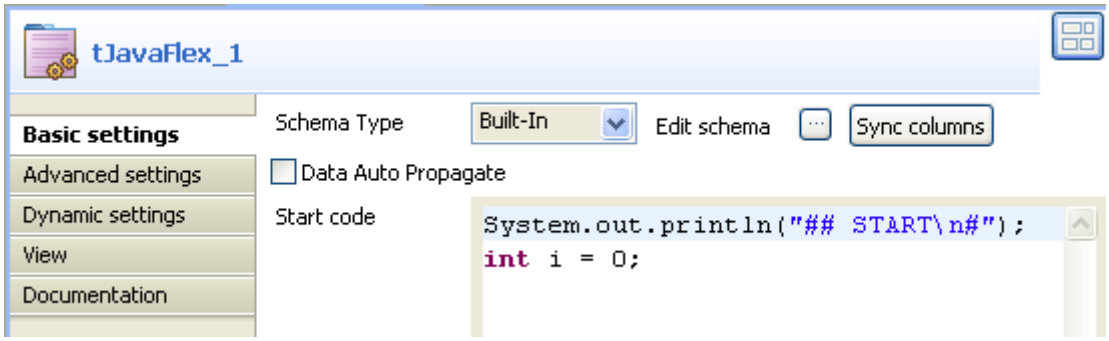


- Drop **tRowGenerator** and **tJavaFlex** from the **Palette** onto the design workspace.
- Connect the components together using a **Row Main** link.
- Double-click **tRowGenerator** to display its **Basic settings** view and the **[RowGenerator Editor]** dialog box where you can define the component properties.



- Click the plus button to add four columns: *number*, *txt*, *date* and *flag*.
- Define the schema and set the parameters to the four columns according to the above capture.

- In the **Functions** column, select the three-dot function (**...**) for each of the defined columns.
- In the **Parameters** column, enter 10 different parameters for each of the defined columns. These 10 parameters corresponds to the data that will be randomly generated when executing **tRowGenerator**.
- Click **OK** to validate your changes and close the editor.
- Double-click **tJavaFlex** to display its **Basic settings** view and define the components properties.



- Click **Sync columns** to retrieve the schema from the preceding component.
- In the **Start code** field, enter the code to be executed in the initialization phase.
  In this example, the code indicates the initialization of the **tJavaFlex** component by displaying the START message and defining the variable to be used afterwards in the Java code:

```
System.out.println("## START\n#");
int i = 0;
```

- In the **Main code** field, enter the code to be applied on each line of data.
  In this example, we want to show the number of each line starting from 0 and then the number and the random text transformed to upper case and finally the random date set in the editor of **tRowGenerator**. Then, we create a condition to show if the status is **true** or **false** and we increment the number of the line:

```
System.out.print(" row" + i + ":");
System.out.print("# number:" + row1.number);
System.out.print (" | txt:" + row1.txt.toUpperCase());
System.out.print(" | date:" + row1.date);
if(row1.flag) System.out.println(" | flag: true");
else  System.out.println(" | flag: false");

i++;
```

Main code

```
System.out.print(" row" + i + ":");
System.out.print("# number:" +
 row1.number);
System.out.print (" | txt:" +
 row1.txt.toUpperCase());
System.out.print(" | date:" +
 row1.date);
if(row1.flag) System.out.println(" |
flag: true");
else  System.out.println(" | flag:
false");

i++;
```

In the Main code filed, "row1" corresponds to the name of the link that comes out of **tJavaFlex**. *If you rename this link, you have to modify the code.*

- In the **End code** field, enter the code that will be executed in the closing phase.
  In this example, the code indicates the end of the execution of **tJavaFlex** by displaying the END message:
  `System.out.println("#\n## END");`

End code

```
System.out.println("#\n## END");
```

- Save your Job and press **F6** to execute it.

```
Starting job tJavaFlex_scenario2 at 18:35 02/09/2009.

## START
#
 row0:# number:10 | txt:TEXT5 | date:2006-05-25 | flag: false
 row1:# number:7 | txt:TEXT7 | date:2006-06-26 | flag: true
 row2:# number:5 | txt:TEXT2 | date:2006-05-25 | flag: false
 row3:# number:6 | txt:TEXT1 | date:2005-08-28 | flag: true
 row4:# number:8 | txt:TEXT9 | date:2006-05-25 | flag: false
 row5:# number:9 | txt:TEXT1 | date:2007-01-21 | flag: false
 row6:# number:7 | txt:TEXT5 | date:2004-11-21 | flag: true
 row7:# number:9 | txt:TEXT4 | date:2005-08-29 | flag: true
 row8:# number:4 | txt:TEXT6 | date:2006-06-26 | flag: false
 row9:# number:3 | txt:TEXT10 | date:2006-05-25 | flag: false
#
## END
Job tJavaFlex_scenario2 ended at 18:35 02/09/2009. [exit
code=0]
```

The console displays the randomly generated data that was modified by the java command set through **tJavaFlex**.

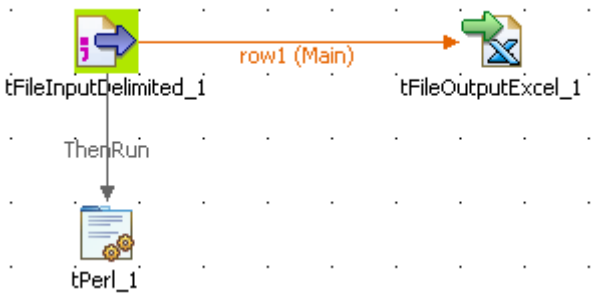
tPerl

tPerl properties

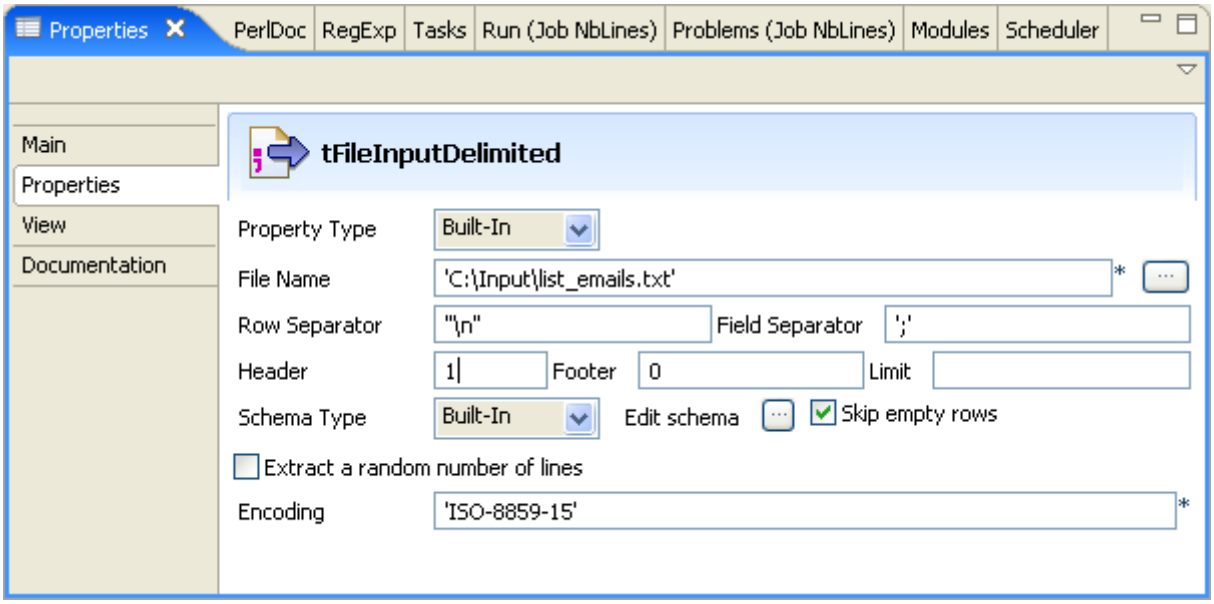
Component family	Processing	
Function	**tPerl** transforms any data entered as argument of Perl commands.	
Purpose	**tPerl** is an (Perl) editor that is a very flexible tool within a job.	
Basic settings	*Code*	Type in the Perl code based on the command and task you need to perform. For further information about Perl functions syntax, see **Talend Open Studio** online Help (under **Talend Open Studio User Guide** > Perl)
Usage	Typically used for debugging but can also be used to display a variable content.	
Limitation	This component requires an advanced Perl user level and is not meant to be used with a Row connection as is meant for single use.	

Scenario: Displaying a number of processed lines

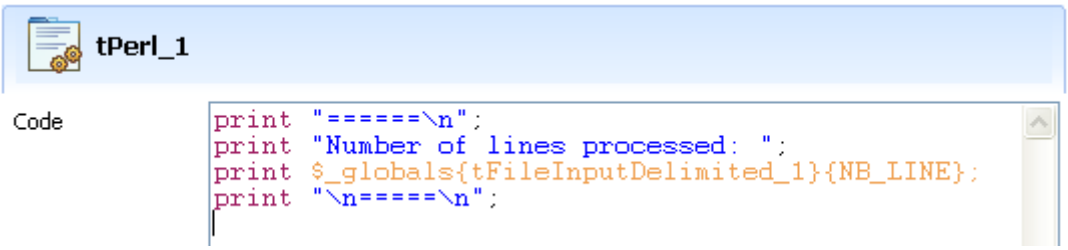
This scenario is a three-component job showing in the Log the number of rows being processed and output in an XML file.



- Drop three components from the **Palette** onto the design workspace: **tFileInputDelimited**, **tFileOutputExcel**, **tPerl**
- Right-click **tFileInputDelimited** and connect it to **tFileOutputExcel** using a main **Row**.
- Right-click again **tFileInputDelimited** and link it to **tPerl** using a **Trigger > ThenRun** link. This link means that, following the arrow direction, the first component (**tFileDelimited**) will run before the second component (**tPerl**).
- Click once on **tFileInputDelimited** and select the **Basic settings** tab to define the component properties.



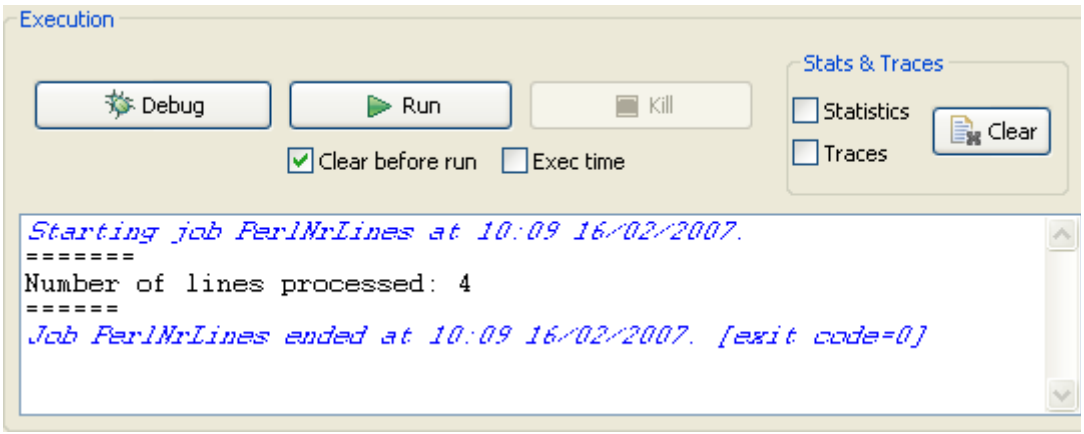
- The **Properties** are not reused from or for another job stored in the repository, but instead are used for this job only. Therefore select **Built-In** in the drop-down list.
- Enter a path or browse to the file containing the data to be processed. In this example, the text file gathers a list of names facing the relevant email addresses.
- Define the **Row** and **Field** separators. In this scenario, there is one name and the matching email per row. And the fields are separated by a semi-colon.
- The first row of the file contains the labels of the columns, therefore it should be ignored in the job. Therefore the Header field value is 1.
- There is no footer nor limit value to be defined for this scenario.
- The **Schema type** is also built-in in this case. Click on **Edit Schema** and describe the content of the input file. In this scenario, there are two columns labelled *Name* and *Emails*, of type String and with no length defined. Key field being *Email*.
- Select the **tFileOutputExcel** component and define it accordingly.
- Select the output file path, Sheet and synchronize the schema.
- Then define the **tPerl** sub-job in order to get the number of rows transferred to the XML Output file.



- Enter the Perl command `print` to get the variable containing the number of rows read in **tFileInputDelimited**. To access the list of available variables, press **Ctrl+Space** then select the relevant variable in the list.

- For a better readability in the **Run Job** log, add equal signs before and after the commands. Note also that commands, strings and variables are colored differently.
- Then switch to the **Run Job** tab and execute the job.

The job runs smoothly and create an output xml file following the two-field schema defined: Name and Email.



The Perl command result is shown in the job log.

Data Quality components

This chapter details the major components that you can find in the **Data Quality** group of the **Palette** of **Talend Open Studio**.

The Data Quality family groups dedicated components that help you improve the quality of your data. These components covers various needs such as narrow down filtering the unique row, calculating CRC, finding data based on fuzzy matching, and so on.

tAddCRCRow

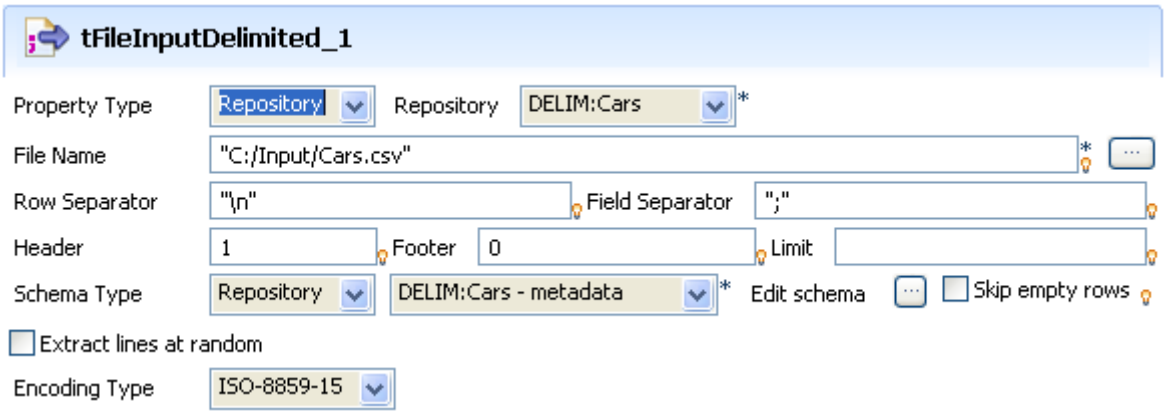
tAddCRCRow properties

Component family	Data quality	
Function	Calculates a surrogate key based on one or several columns and adds it to the defined schema	
Purpose	Providing a unique ID helps improving the quality of processed data.	
Basic settings	*Schema type* and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the Repository. In this component, a new CRC column is automatically added.
		Built-in: The schema will be created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused in various projects and job designs. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	Implication	select the check box facing the relevant columns to be used for the surrogate key checksum.
	CRC type	Select the CRC type length. The longer the CRC, the least overlap.
Usage	This component is an intermediary step. It requires an input flow as well as an output.	
Limitation	n/a	

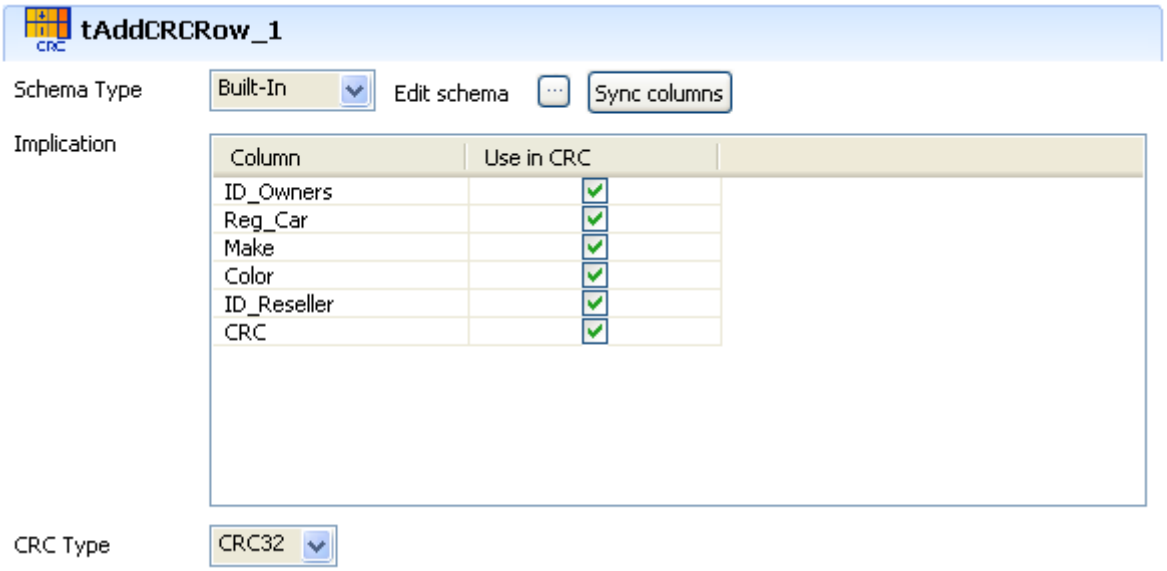
Scenario: Adding a surrogate key to a file

This scenario describes a Job adding a surrogate key to a delimited file schema.

- Drop the following components: **tFileInputDelimited**, **tAddCRCRow** and **tLogRow**.
- Connect them using a **Main row** connection.
- In the **tFileInputDelimited Component** view, set the **File Name** path and all related properties in case these are not stored in the **Repository**.

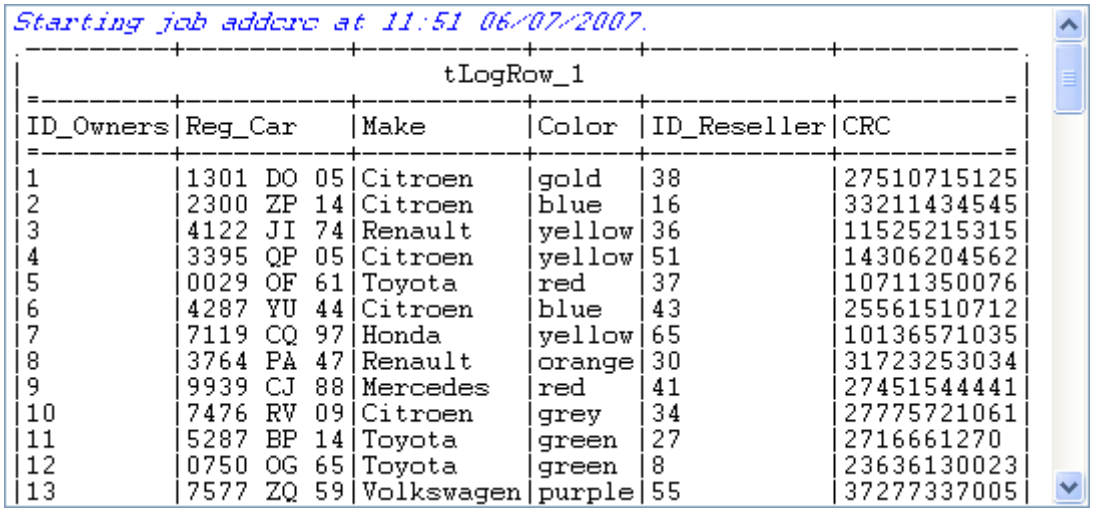


- Create the schema through the **Edit Schema** button, in case the schema is not stored already in the **Repository**. In Java, mind the data type column and in case of Date pattern to be filled in, check out http://java.sun.com/j2se/1.5.0/docs/api/index.html.
- In the **tAddCRCRow Component** view, select the check boxes of the input flow columns to be used to calculate the CRC.



- Notice that a CRC column (read-only) has been added at the end of the schema.
- Select **CRC32** as **CRC Type** to get a longer surrogate key.

- In the **Basic settings** view of **tLogRow**, select the **Print values in cells of a table** check box to display the output data in a table on the Console.
- Then save your Job and press **F6** to execute it.



```
Starting job addcrc at 11:51 06/07/2007.
.---------+----------+----------+------+-----------+-----------.
|                              tLogRow_1                         |
|=---------+----------+----------+------+-----------+-----------=|
|ID_Owners|Reg_Car    |Make      |Color |ID_Reseller|CRC        |
|=---------+----------+----------+------+-----------+-----------=|
|1        |1301 DO 05|Citroen   |gold  |38         |27510715125|
|2        |2300 ZP 14|Citroen   |blue  |16         |33211434545|
|3        |4122 JI 74|Renault   |yellow|36         |11525215315|
|4        |3395 QP 05|Citroen   |yellow|51         |14306204562|
|5        |0029 OF 61|Toyota    |red   |37         |10711350076|
|6        |4287 YU 44|Citroen   |blue  |43         |25561510712|
|7        |7119 CQ 97|Honda     |yellow|65         |10136571035|
|8        |3764 PA 47|Renault   |orange|30         |31723253034|
|9        |9939 CJ 88|Mercedes  |red   |41         |27451544441|
|10       |7476 RV 09|Citroen   |grey  |34         |27775721061|
|11       |5287 BP 14|Toyota    |green |27         |2716661270 |
|12       |0750 OG 65|Toyota    |green |8          |23636130023|
|13       |7577 ZQ 59|Volkswagen|purple|55         |37277337005|
```

An additional CRC Column has been added to the schema calculated on all previously selected columns (in this case all columns of the schema).

tFuzzyMatch

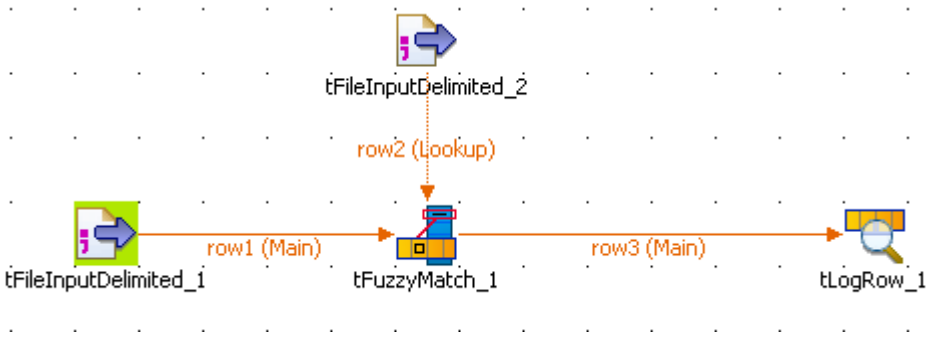
tFuzzyMatch properties

Component family	Data quality	
Function	Compares a column from the main flow with a reference column from the lookup flow and outputs the main flow data displaying the distance	
Purpose	Helps ensuring the data quality of any source data against a reference data source.	
Basic settings	*Schema type* and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the Repository. Two read-only columns, Value and Match are added to the output schema automatically.
		Built-in: The schema will be created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused in various projects and job designs. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	Matching type	Select the relevant matching algorithm among: **Levenshtein**: Based on the edit distance theory. It calculates the number of insertion, deletion or substitution required for an entry to match the reference entry. **Metaphone**: Based on a phonetic algorithm for indexing entries by their pronunciation. It first loads the phonetics of all entries of the lookup reference and checks all entries of the main flow against the entries of the reference flow. **Double Metaphone**: a new version of the Metaphone phonetic algorithm, that produces more accurate results than the original algorithm. It can return both a primary and a secondary code for a string. This accounts for some ambiguous cases as well as for multiple variants of surnames with common ancestry.
	Min Distance	(Levenshtein only) Set the minimum number of changes allowed to match the reference. If set to 0, only perfect matches are returned.
	Max Distance	(Levenshtein only) Set the maximum number of changes allowed to match the reference.

	Matching Column	Select the column of the main flow that needs to be checked against the reference (lookup) key column
	Unique Matching	Select this check box if you want to get the best match possible, in case several matches are available.
	Matching item separator	In case several matches are available, all of them are displayed unless the unique match box is selected. Define the delimiter between all matches.
Usage	This component is not startable (green background) and it requires two input components and an output component.	
Limitation/prerequisite	Perl users: Make sure the relevant packages are installed. Check the Module view for modules to be installed	

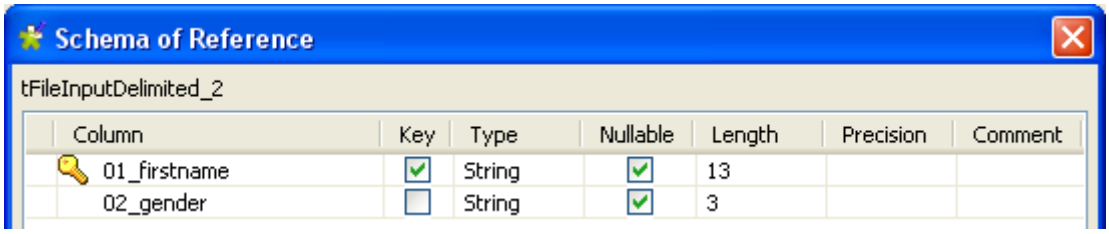
Scenario 1: Levenshtein distance of 0 in first names

This scenario describes a four-component Job aiming at checking the edit distance between the *First Name* column of an input file with the data of the reference input file. The output of this Levenshtein type check is displayed along with the content of the main flow on a table

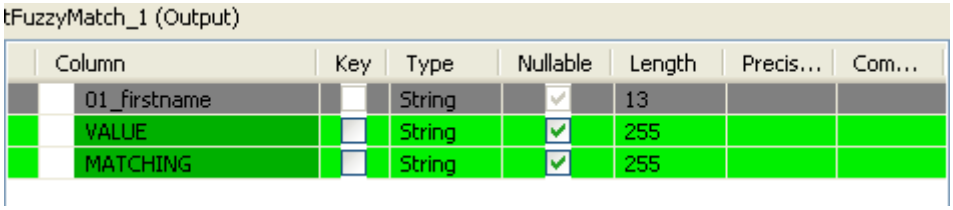


- Drag and drop the following components from the **Palette** to the design workspace: **tFileInputDelimited** (x2), **tFuzzyMatch**, **tFileOutputDelimited**.
- Define the first **tFileInputDelimited** Basic settings. Browse the system to the input file to be analyzed and most importantly set the schema to be used for the flow to be checked.
- In the schema, set the **Type** of data in the Java version, especially if you are in **Built-in** mode.
- Link the defined input to the **tFuzzyMatch** using a **Main** row link.
- Define the second **tFileInputDelimited** component the same way.

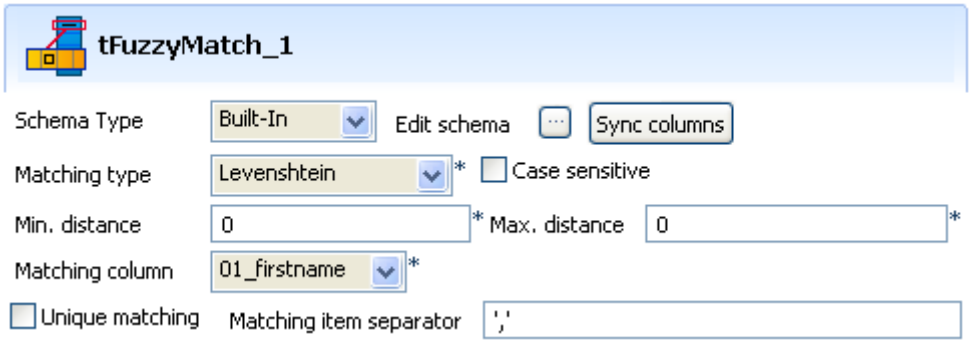
Make sure the reference column is set as key column in the schema of the lookup flow.



- Then connect the second input component to the **tFuzzyMatch** using a main row (which displays as a **Lookup** row on the design workspace).
- Select the **tFuzzyMatch** Basic settings.
- The **Schema** should match the **Main** input flow schema in order for the main flow to be checked against the reference.



- Note that two columns, **Value** and **Matching**, are added to the output schema. These are standard matching information and are read-only.
- Select the method to be used to check the incoming data. In this scenario, *Levenshtein* is the **Matching type** to be used.
- Then set the distance. In this method, the distance is the number of char changes (insertion, deletion or substitution) that needs to be carried out in order for the entry to fully match the reference.



- In this use case, we want the distance be of 0 for the min. or for the max. This means only the exact matches will be output.
- Also, clear the **Case sensitive** check box.
- And select the column of the main flow schema that will be selected. In this example, the first name.
- No need to select the **Unique matching** check box nor hence the separator.
- Link the **tFuzzyMatch** to the standard output **tLogRow**. No other parameters than the display delimiter is to be set for this scenario.

- Save the Job and press **F6** to execute the Job.

```
audra (1)||
audra (2)||
audrea||
audrey||
august (1)||
august (2)||
augusta||
auguste|0|auguste
augustijn||
augustin|0|augustin
augustine||
augusto||
augusts||
augustus||
aukusti||
```

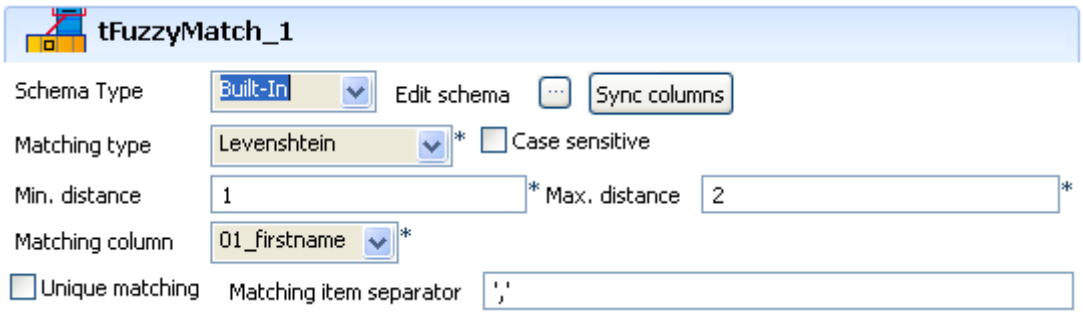
As the edit distance has been set to 0 (min and max), the output shows the result of a regular join between the main flow and the lookup (reference) flow, hence only full matches with Value of 0 are displayed.

A more obvious example is with a minimum distance of 1 and a max. distance of 2, see *Scenario 2: Levenshtein distance of 1 or 2 in first names on page 110*.

Scenario 2: Levenshtein distance of 1 or 2 in first names

This scenario is based on the scenario 1 described above. Only the min and max distance settings in **tFuzzyMatch** component get modified, which will change the output displayed.

- In the Component view of the **tFuzzyMatch**, change the min distance from 0 to 1. This excludes straight away the exact matches (which would show a distance of 0).
- Change also the max distance to 2 as the max distance cannot be lower than the min distance. The output will provide all matching entries showing a discrepancy of 2 characters at most.



- No other change of the setting is required.
- Make sure the Matching item separator is defined, as several references might be matching the main flow entry.
- Save the new Job and press **F6** to run it.

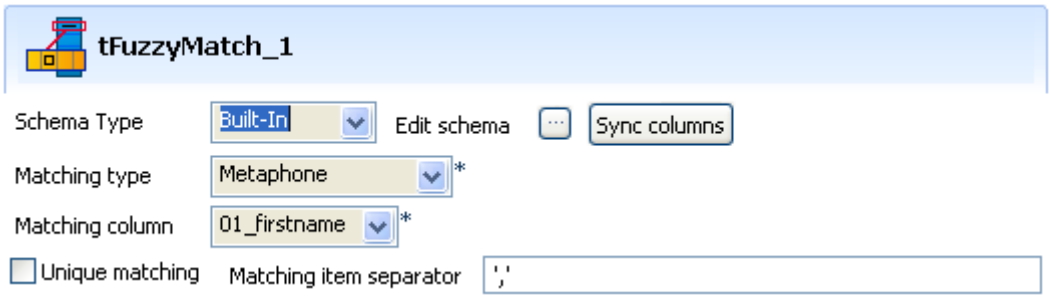
```
audrea|2|aude
audrey|2|aude
august (1)||
august (2)||
augusta|1|auguste
auguste|2|augustin
augustijn|1|augustin
augustin|2|auguste
augustine|1|augustin
augusto|1|auguste
augusts|1|auguste
augustus|2|auguste,augustin
aukusti|2|auguste,augustin
aulay||
aulus|2|jules
```

As the edit distance has been set to 2, some entries of the main flow match several reference entries.

You can also use another method, the metaphone, to assess the distance between the main flow and the reference,

Scenario 3: Metaphonic distance in first name

This scenario is based on the scenario 1 described above.



- Change the **Matching type** to **Metaphone**. There is no min nor max distance to set as the matching method is based on the discrepancies with the phonetics of the reference.
- Save the Job and press **F6**. The phonetics value is displayed along with the possible matches.

```
audrey||
august (1)|AKST|auguste
august (2)|AKST|auguste
augusta|AKST|auguste
auguste|AKST|auguste
augustijn||
augustin|AKSTN|augustin
augustine|AKSTN|augustin
augusto|AKST|auguste
augusts||
augustus||
aukusti|AKST|auguste
aulay||
aulus||
aune||
```

tIntervalMatch

tIntervalMatch properties

Component family	Data Quality	
Function	**tIntervalMatch** receives a main flow and aggregates it based on join to a lookup flow (Java) or a given lookup file (Perl). Then it matches a specified value to a range of values and returns related information.	
Purpose	Helps to return a value based on a Join relation.	
Basic settings	*Schema type* and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the Repository. Click **Edit Schema** to make changes to the schema. Note that if you make changes, the schema automatically becomes built-in.
		Built-in: The schema will be created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused in various projects and job flowcharts. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
Java only	*Search column*	Select the main flow column containing the values to be matched to a range of values
	Column (LOOKUP)	Select the lookup flow column containing the values to be returned when the Join is ok.
	Lookup Column min/ bounds strictly (min)	Select the column containing the min value of the tange. Check the box if the boundary is strict.
	Lookup Column max/ bounds strictly (max)	Select the column containing the max value of the tange. Select the check box if the boundary is strict.
Usage	This component handles flow of data therefore it requires input and output, hence is defined as an intermediary step.	
Limitation	n/a	

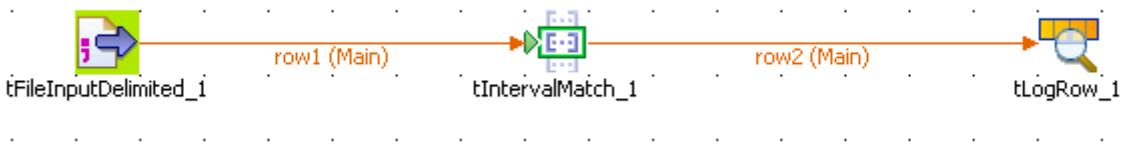
The Perl properties being quite different from the Java properties, they are described in a separate table below.

PERL basic settings	

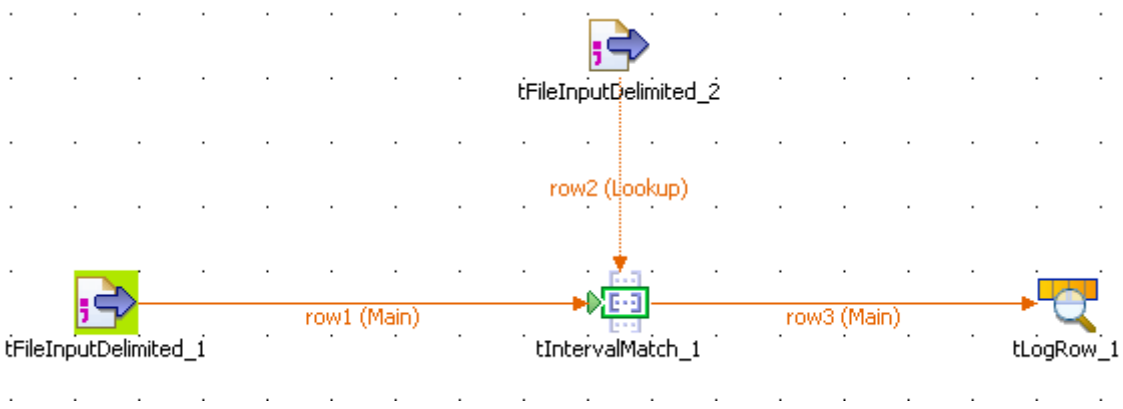
Basic settings	*Schema type* and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the Repository. Click **Edit Schema** to make changes to the schema. Note that if you make changes, the schema automatically becomes built-in.
		Built-in: The schema will be created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide
		Repository: The schema already exists and is stored in the Repository, hence can be reused in various projects and job flowcharts. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
Perl only	*File Name*	Enter the file that contains the range of values. It functions as a lookup flow.
	Field separator	Character, string or regular expression to separate fields in the lookup file.
	Row separator	String (ex: "\n"on Unix) to distinguish rows in the lookup file.
	Lookup index Column	Position of the **min** column in the lookup file: 0 for first col, 1 for second col, etc. Make sure the interval min and max columns are adjacent.
	Search column	Select the main flow column containing the values to be matched to a range of values
Usage	This component handles flow of data therefore it requires input and output, hence is defined as an intermediary step.	
Limitation	For the time being, the Perl version of the **tIntervalMatch** does not accept a real Lookup flow (but only a reference file in the actual component's settings)	

Scenario: identifying Ip country (Perl and Java)

The following scenario describes a Job designed in parallel in both languages, Perl and Java. In this Job, a incoming main flow provides 2 columns: *Documents* and *IP* dummy values. A second file used as lookup flow in Java and reference range file in Perl contains a list of sorted IP ranges and their corresponding country. This Job aims at retrieving each document's country from their IP value, in other words, creating a Join between the main flow and the lookup flow.

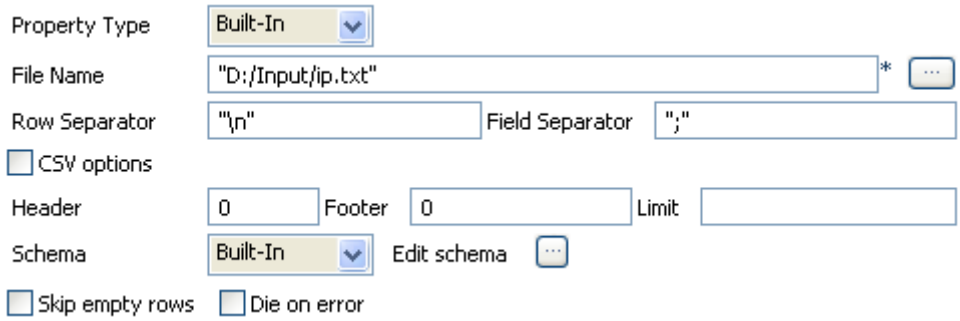


In **Perl**, the Job requires one **tFileInputDelimited**, a **tIntervalMatch** and a **tLogRow**.

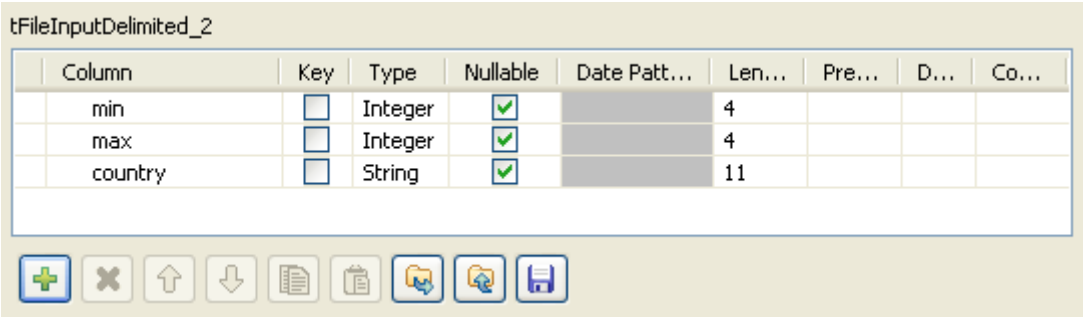


In **Java**, the Job requires one extra **tFileInputDelimited**, a **tIntervalMatch** and a **tLogRow**.

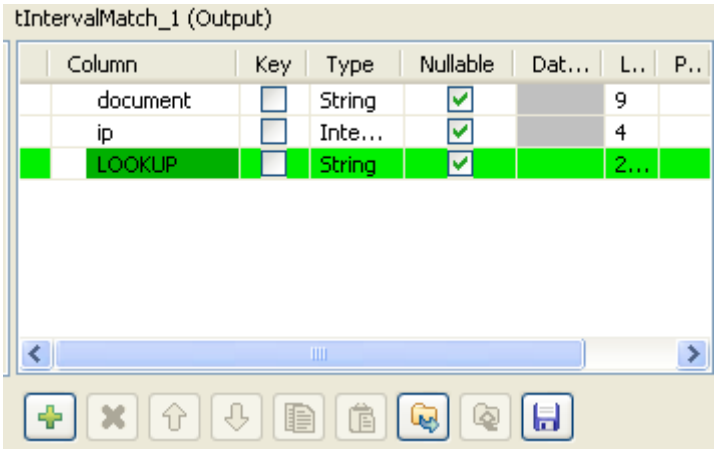
- Drop the components onto the design workspace.
- Set the basic settings of the **tFileInputDelimited** component.



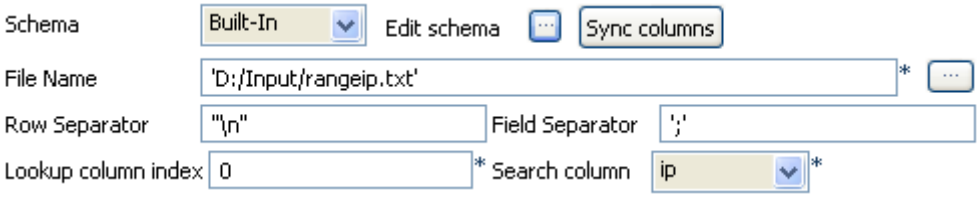
- The schema is made of two columns, respectively *Document* and *IP*
- (Java only) Set the **Type** column on **String** for the *Document* column and **Integer** for the *IP* column.
- (Java only) Set now the second **tFileInputDelimited** properties.



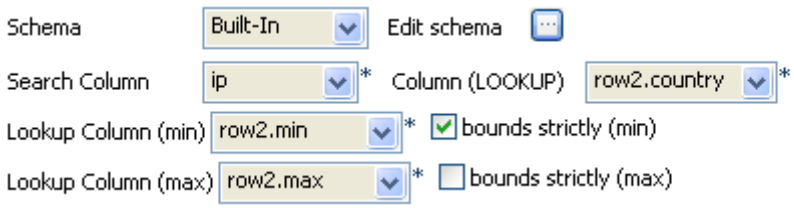
- (Java only) Don't forget to define the **Type** of data.
- (Both Java and Perl) Propagate the schema from the incoming main flow to the **tIntervalMatch** component.



- (Both Java and Perl) Note that the output schema from the **tIntervalMatch** component is read-only and is made of the input schema plus an extra Lookup column which will output the requested lookup data.
- Set the other properties of the **tIntervalMatch** component.
- (Perl only), the lookup file is defined directly in the setting of the **tIntervalMatch**.



- (Perl only) In **File Name** field, set the path to the lookup file. Set the Row and Field separator of the lookup file.
- (Perl only) In **Lookup column index** field, set the inferior bound of the data range. This corresponds to the position of the column containing the min value of the range (0 for the first column).
- (Perl only) No need to set the lookup values to be returned as all values from the lookup will be outputted.
- (Java only) Set the tIntervalMatch other properties such as the min and max column corresponding to the range bounds.



- (Java only) In the **Column Lookup** field, select the column where are the values to be returned.
- (Both Java and Perl) In the **Search column** field, select the main flow column containing the values to be matched to the range values.

- (Both Java and Perl) The **tLogRow** component does not require any specific setting for this Job.

Both Perl and Java Jobs output the same result with slight differences in the way they display.

```
Starting job IntervalN at 17:03 01/02/2008.
document1|1200|1001;2000;DEUTSCHLAND
document2|4500
document3|500|0;1000;FRANCE
document1|2201|2001;3000;ITALIA
Job IntervalN ended at 17:03 01/02/2008. [exit code=0
```

```
Starting job intervalMatch at 17:05 01/02/2008.
document1|1200|DEUTSCHLAND
document2|4500|
document3|500|FRANCE
document1|2201|ITALIA
Job intervalMatch ended at 17:05 01/02/2008. [exit c
```

The Perl results include the range values whereas the Java output only includes the requested return value (country).

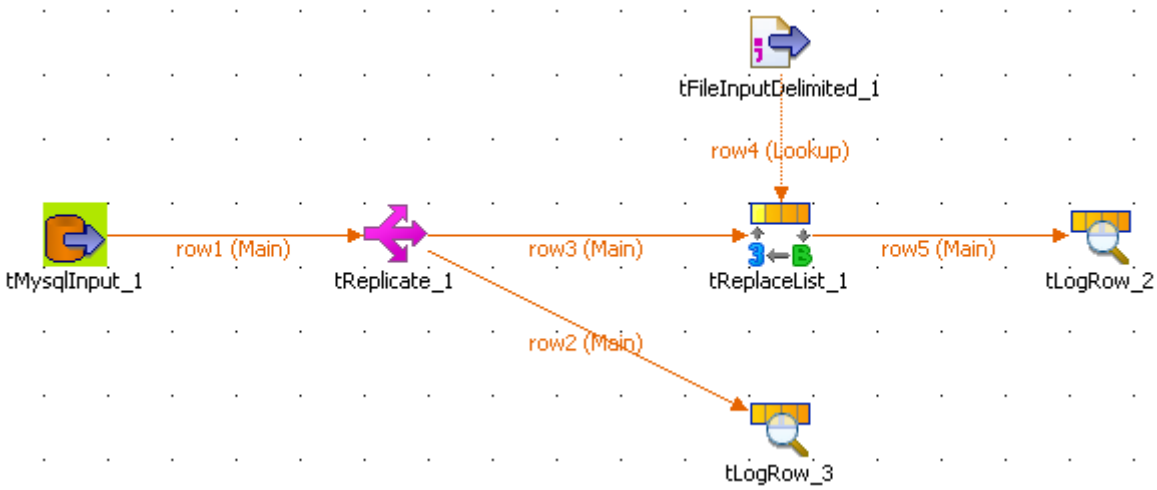
tReplaceList

tReplaceList Properties

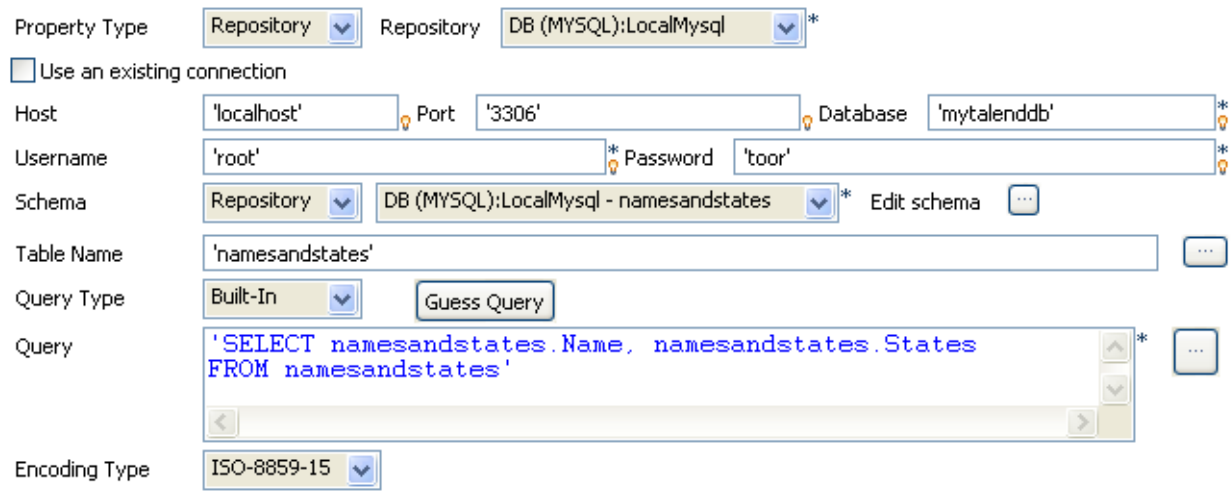
Component family	Processing	
Function	Carries out a Search & Replace operation in the input columns defined based on an external lookup.	
Purpose	Helps to cleanse all files before further processing.	
Basic settings	*Schema type* and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the Repository. Two read-only columns, Value and Match are added to the output schema automatically.
		Built-in: The schema will be created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused in various projects and job designs. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	Lookup search column	Type in the position number of the column to be searched in the lookup schema. **0**: first column read **1**: second column read **n**: position number of the column in the schema read.
	In order to ensure the uniqueness of values being searched, make sure this column is marked as Key in your lookup schema.	
	Lookup replacement column	Type in the position number of the column where the replacement values are stored. **0**: first column read **1**: second column read **n**: position number of the column in the schema read
	Column options	Select the columns of the main flow where the replacement is to be carried out.
Usage	This component is not startable as it requires an input flow. And it requires an output component.	

Scenario: Replacement from a reference file

The following Job (created in Perl) searches and replaces a list of countries with their corresponding codes. The relevant codes are taken from a reference file placed as lookup flow in the Job. The main flow is replicated and both outputs are displayed on the console, in order to show the main flow before and after replacement.

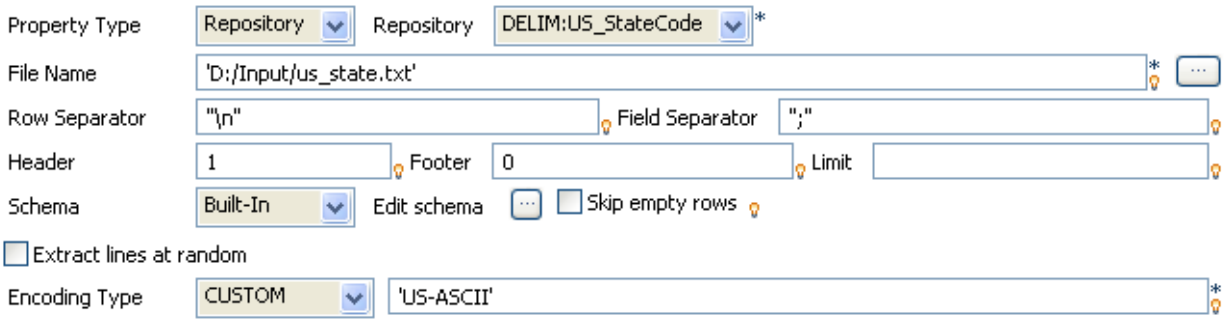


- Drop the following components from the **Palette** to the design workspace: **tMysqlInput**, **tFileInputDelimited**, **tReplicate**, **tReplaceList** and **tLogRow** (x2). Note that if your input schemas are stored in the Repository, you can simply drag and drop the relevant node from the Repository's Metadata Manager onto the design workspace to retrieve automatically the input components' setting. For more information, see *Drop components from the Metadata node* in **Talend Open Studio** User Guide.
- Connect the components using **Main Row** connections via a right-click on each component. Notice that the main row coming from the reference flow (**tFileInputDelimited**) is called a lookup row.
- Select the **tMysqlInput** component and set the input flow parameters.

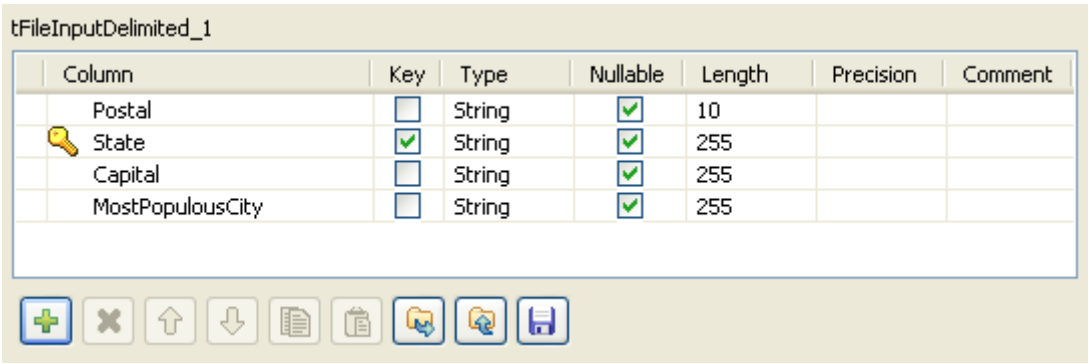


- The input schema is made of two columns: *Names*, *States*. The column States gathered the name of the United States of America which are to be replaced by their respective code.

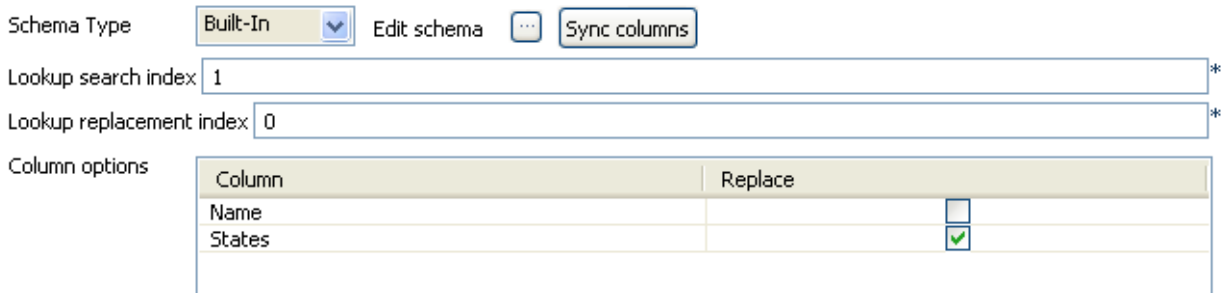
- In the **Query** field, make sure the *State* column is included in the `Select` statement. In this use case, all columns are selected.
- Check the **tReplicate** component setting. The schema is simply duplicated into two identical flows, but no change to the schema can be made.
- Then double-click on the **tFileInputDelimited** component, to set the reference file.



- The file includes two columns: *Postal*, *State* where *Postal* provides the zipcode corresponding to the name given in the respective row of the *State* column.
- The fields are delimited by semicolons and rows are separated by carriage returns.
- Edit the lookup flow schema.



- Make sure the lookup search column (in this use case: *State*) is a key, in order to ensure the uniqueness of the values being searched.
- Select the **tReplaceList** and set the operation to carry out.
- The schema is retrieved from the previous component of the main flow.



- In **Lookup search index** field, type in the position index of the column being searched. In this use case, *State* is the second column of the lookup input file, therefore type in **1** in this field.

- In **Lookup replacement index** field, fill in the position number of the column containing the replacement values, in this example: *Postal* for the State codes.
- In the **Column options** table, select the *States* column as in this use case, the State names are to be replaced with their corresponding code.
- In both **tLogRow** components, select the **Print values in table cells** check box for a better readability of the outputs.
- Save the Job and press **F6** to execute it.

```
Starting job ReplaceList at 15:06 22/10/2007.
.-----------------------------------.
|               tLogRow_3           |
+---------------------+-------------+
| Name                | States      |
+---------------------+-------------+
| 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|
'---------------------+-------------'

.-----------------------------.
|           tLogRow_2         |
+---------------------+-------+
| Name                | States|
+---------------------+-------+
| William Grant       | IA    |
| William Hoover      | NY    |
| Grover Lincoln      | ND    |
| Lyndon Jefferson    | OH    |
| Gerald Hayes        | WA    |
| Benjamin Grant      | ME    |
| George Pierce       | CT    |
| Jimmy Reagan        | AK    |
| Martin Hayes        | WA    |
| Franklin Jefferson  | IA    |
| Andrew Nixon        | NH    |
'---------------------+-------'
Job ReplaceList ended at 15:06 22/10/2007. [exit code=0]
```

The first flow output shows the State column with full state names as it comes from the main input flow.

The second flow output shows the States column after the State column names have been replaced with their respective codes.

tSchemaComplianceCheck

tSchemaComplianceCheck Properties

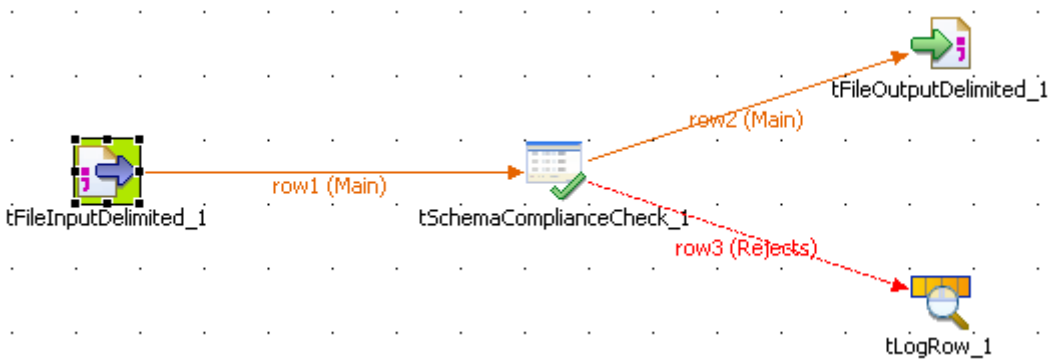
Component family	Data Quality	
Function	Validates all input rows against a reference schema or checks type, nullability, length of rows against reference values. The validation can be carried out in full or partly.	
Purpose	Helps ensuring the data quality of any source data against a reference data source.	
Basic settings	*Base Schema* and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the Repository. Describe the structure and nature of your data to be processed as it is.
		Built-in: The schema will be created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused in various projects and job designs. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
Java only	*Use another schema for compliance check*	Define a reference schema as you expect the data to be, in order to reject the non-compliant data. It can be restrictive on data type, null values, and/or length.
Perl only	*Date language/ Date format*	For the validation of date formats containing string such as 25 Dec 2007, use the Date Language field and to distinguish the way months and days are ordered, use the Date format field.
	Check all columns from schema	Select this check box if all checks are to be carried out on all columns of the base schema.
	Type	In **Perl**, select the check box of the column of which you want the data type to be verified against the base schema definition. In **Java**, select the type of which the data should be. This validation is mandatory for all columns.
	Null (empty or zero)	Select this check box to verify the nullability of the column against the base schema definition.
	Max length	Select this check box to verify the data length against the base schema length definition
Usage	This component is an intermediary step in the flow allowing to exclude from the main flow the non-compliant data. This component cannot be a start component as it requires an input flow. It also requires at least one output component to gather the validated flow, and possibly a second output component for rejected data using Rejects link. For more information, see *Rejects* in **Talend Open Studio** User Guide.	

Scenario: Validating dates against schema (java)

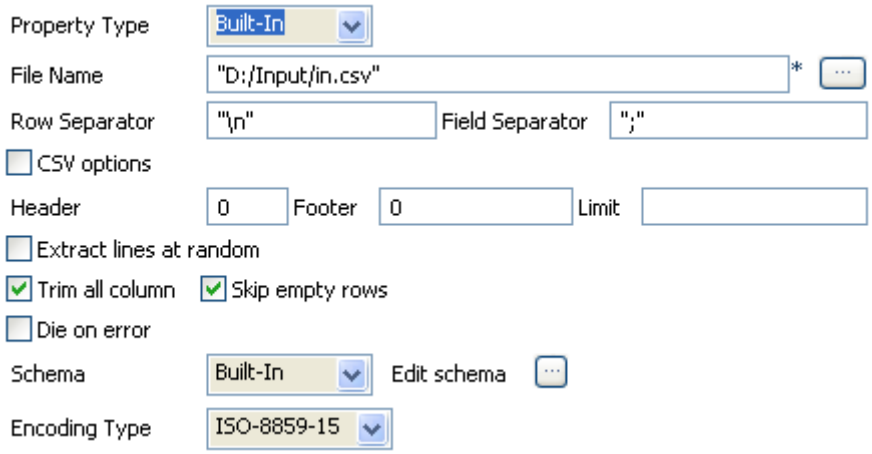
This very basic scenario shows how to check the type, null value and length of an incoming flow against a defined reference schema. The incoming flow comes from a simple csv file which contains heterogeneous data including wrong data type, data exceeding max length, wrong id and null values in non-nullable columns.

```
1;label 1 with max length 30;another label with max length 40;2007-10-01;not null;nullable;
2;label 2 with max length 30;another label with max length 40 and with length 51;2007-06-31;not null;
3;label 3 with max length exceed 30;another label with max length 40 and with length 51;2007-10-01;;;
4;label 4 with max length 30;another label with max length 40;2007-12-13;not null;;
5;label 5 with max length 30;another label with max length 40;2007-13-12;not null;;
6;label 6 with max length 30;another label with max length 40;13/12/2007;not null;;
7;label 7 with max length 30;another label with max length 40;12/13/2007;not null;;
8.6;label with max length 30;another label with max length 40;2007-10-01;not null;
9;label 9 with max length 30;another label with max length 40;2007-10-01;;;
10;label 10 with max length exceed 30;another label with max length 40 and with length
51;2007-10-01;not null;nullable but not null;
```

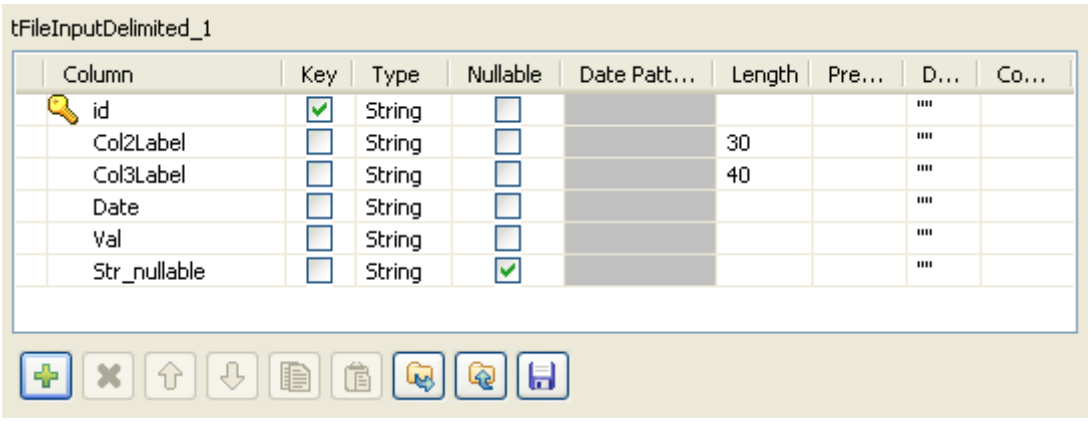
The output is double as the valid data are gathered into a dedicated delimited file, whereas the rejected data are displayed on the console.



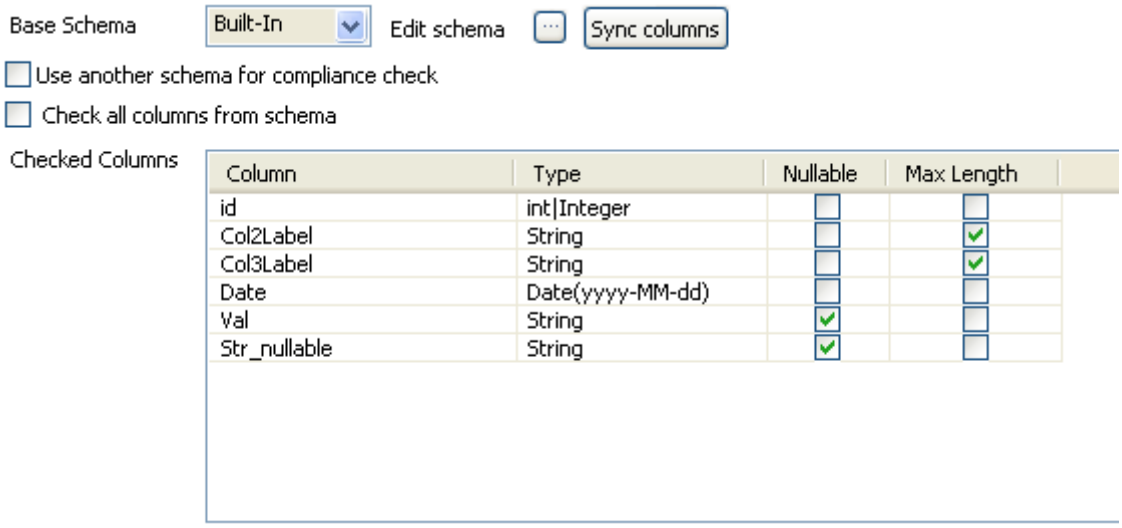
- Drop the following components: **tFileInputDelimited**, **tSchemaComplianceCheck**, **tFileOutputDelimited**, **tLogRow** from the **Palette** to the design workspace.
- Right-click on the **tFileInputDelimited** to connect it to the **tSchemaComplianceCheck** using a row main link.
- Then right-click on the **tSchemaComplianceCheck** component and select **Row > Main** to connect it to the **tFileOutputDelimited**. This output flow will gather the valid data.
- Right-click again on the **tSchemaComplianceCheck** component and, this time, select **Row > Rejects** link to connect it to the **tLogRow** component. This second output flow will gather the non-compliant data.
- Select the **Rejects** link that you just connected, and notice that the schema passed through to the **tLogRow** contains already two columns: *ErrorCode* and *ErrorMessage*. These two column are read-only and provides information about the rejected data, easing the error handling and troubleshooting if need be.
- Now define the properties of each component.



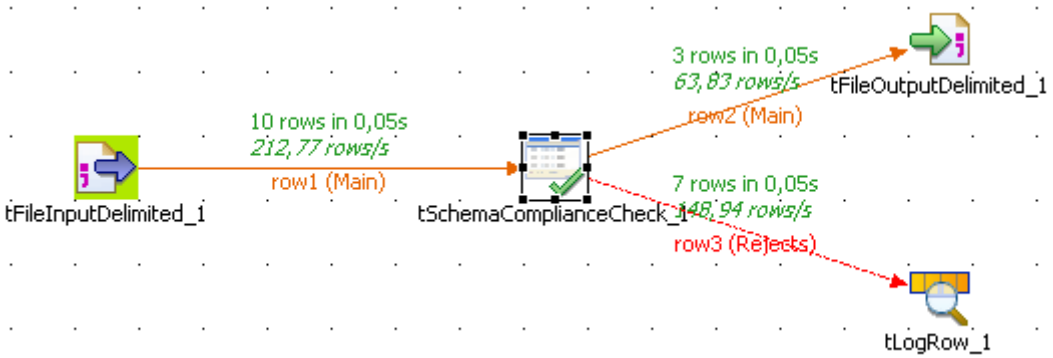
- On the **Component** view of the **tFileInputDelimited** component, leave the default parameters for a delimited file, and fill in the **File Name** browsing to the input file. In this example: *in.csv*
- Click **Edit Schema** to describe the data structure of this input file. The schema is made of six columns: *id*, *Col2Label*, *Col3Label*, *Date*, *Val* and *Str_Nullable*.



- Leave the Type field as permissive as possible (especially in Java). You will define the actual type of the data on the **tSchemaComplianceCheck** Component view.
- The *Str_nullable* column is the only nullable data. So select the relevant check box. *Col2Label* and *Col3Label* are length-limited to respectively *30* and *40* characters.
- Then double-click on **tSchemaComplianceCheck** component to set the validation parameters.



- The **Base schema** should be automatically propagated from the input component. If not, click on the **Sync columns** button.
- In this example, we use the **Checked columns** table to set the validation parameters. But you could also select the **Use another schema for compliance check** check box and define the schema description of the expected data.
- Select the **Check all columns from schema** check box if you want to check all the columns against all parameters (type, null values and length).
- On the **Checked Columns** table, set the checks to be performed. Below is the setting for a job designed in **Talend Open Studio** with Java generation language:
  - The *Id* type should be **Int**.
  - The *Col2Label* and *Col3Label* should be checked for their **Length**
  - The *Date* should of type **Date**
  - The *Val* column should be checked for **null** values (as it should not be null).
  - The *Str_nullable* should be checked for **null** values also (but it can be null).
- In Perl, select the relevant **Type** check boxes for the columns which the type needs to be selected (Date). Also, define the **Date format** and **Date language**.
- Then set the output components parameters.
- The schema is propagated automatically from the **tSchemaComplianceCheck** to the output components.
- Define the output file path on the **tFileOutputDelimited** properties.
- Then on the **Run** tab, select the **Statistics** check box to display the flow dispatching rates before launching the execution.



- You can notice that three rows from the input flow were validated and therefore were directed to the **tFileOutputDelimited**.
- Also seven rows were not compliant with the reference schema, and therefore were directed to the console.

```
Starting job compliancecheck2 at 11:53 18/12/2007.
[statistics] connecting to socket on port 3919
[statistics] connected
2|label 2 with max length 30|another label with max length 40 and with length
51|2007-06-31|not null||16|Col3Label:exceed max length;Date:Date format not valid
3|label 3 with max length exceed 30|another label with max length 40 and with length
51|2007-10-01|||16|Col2Label:exceed max length;Col3Label:exceed max length
5|label 5 with max length 30|another label with max length 40|2007-13-12|not
null||2|Date:Date format not valid
6|label 6 with max length 30|another label with max length 40|13/12/2007|not
null||2|Date:Date format not valid
7|label 7 with max length 30|another label with max length 40|12/13/2007|not
null||2|Date:Date format not valid
8.6|label with max length 30|another label with max length 40|2007-10-01|not null||2|id:wrong
type
10|label 10 with max length exceed 30|another label with max length 40 and with length
51|2007-10-01|not null|nullable but not null|16|Col2Label:exceed max length;Col3Label:exceed
max length
[statistics] disconnected
Job compliancecheck2 ended at 11:53 18/12/2007. [exit code=0]
```

You can notice that the extra output columns in the **Rejects** link provide the **error code** as well as an **error message** such as: *2|Date:Date format not valid* or *2|id:wrong type* in order to ease the identification of the error.

tUniqRow

tUniqRow Properties

Component family	Data Quality	
Function	Compares entries and removes the first encountered duplicate from the input flow.	
Purpose	Ensures data quality of input or output flow in a Job.	
Basic settings	*Schema type* and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the Repository. Click **Edit Schema** to make changes to the schema. Note that if you make changes, the schema automatically becomes built-in. Click **Sync columns** to retrieve the schema from the previous component connected in the Job.  *if you want the deduplication to be carried out on particular columns, define them on the schema.*
		Built-in: The schema will be created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused in various projects and job flowcharts. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	Case sensitive	Select this check box to consider the lower or upper case.
Usage	This component handles flow of data therefore it requires input and output, hence is defined as an intermediary step.	
Limitation	n/a	

Scenario: Unduplicating entries

Based on the **tSortRow** job, the **tUniqRow** component is added to the Job in order to have unique entries in the output flow. In fact, as the input data is randomly created, duplication cannot be avoided.

- On the **Basic settings** tab panel of the **tUniqRow** component, click **Edit Schema...** to set the **Key** on *Names* field to have unique entries in the output flow on this criteria.
- Select the **Case Sensitive** check box to differentiate lower case and upper case.
- Press **F6** to run the Job again. The console displays the sorted and unique results

```
Starting job UnduplicateJob at 11:02 03/01/2007.
5|3|Mickael|93
2|1|Pierrick|92
8|1|Steffie|89
4|3|Fabrice|75
3|4|Bertrand|67
7|1|Matthew|28
Job UnduplicateJob ended at 11:02 03/01/2007. [exit code=0]
```

Database components

This chapter details the major components that you can find in **Databases** group of the **Palette** of **Talend Open Studio**.

The Databases family groups most popular database connectors. These connectors cover various needs including: opening connection, reading and writing tables, committing transactions as a whole as well as performing rollback for error handlings. More than 40 RDBMS are natively supported.

tAccessInput

tAccessInput properties

Component family	Databases/Access	
Function	**tAccessInput** reads a database and extracts fields based on a query.	
Purpose	**tAccessInput** executes a DB query with a strictly defined statement which must correspond to the schema definition. Then it passes on the field list to the next component via a Main row link.	
Basic settings	*Property type*	Either Built-in or Repository
		Built-in: No property data stored centrally.
		Repository: Select the repository file where Properties are stored. The fields that come after are pre-filled in using the fetched data.
		Click this icon to open a database connection wizard and store the database connection parameters you set in the component **Basic settings** view. For more information about setting up and storing database connection parameters, see *Setting up a DB connection* of **Talend Open Studio** User Guide.
	Database	Name of the database
	Username and *Password*	DB user authentication data.
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either built-in or remotely stored in the Repository.
		Built-in: The schema is created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	Query type and Query	Enter your DB query paying particularly attention to properly sequence the fields in order to match the schema definition.
Advanced settings	*Encoding Type*	Select the encoding type from the list or select **Custom** and define it manually. This field is compulsory for DB data handling.
	Trim all the String/Char columns	Select this check box to remove leading and trailing whitespace from all the String/Char columns.
	Trim column	Remove leading and trailing whitespace from defined columns.

	*tState*Catcher Statistics	Select this check box to collect log data at the component level.
Usage	This component offers the flexibility benefit of the DB query and covers all possibilities of SQL queries.	

Related scenarios

For related topic, see **tDBInput** scenarios:

- *Scenario 1: Displaying selected data from DB table on page 163*
- *Scenario 2: Using StoreSQLQuery variable on page 164*

Related topic in description of *tContextLoad on page 652*.

tAccessOutput

tAccessOutput properties

Component family	Databases/Access	
Function	**tAccessOutput** writes, updates, makes changes or suppresses entries in a database.	
Purpose	**tAccessOutput** executes the action defined on the table and/or on the data contained in the table, based on the flow incoming from the preceding component in the Job.	
Basic settings	*Property type*	Either Built-in or Repository.
		Built-in: No property data stored centrally.
		Repository: Select the repository file where Properties are stored. The fields that come after are pre-filled in using the fetched data.
		Click this icon to open a database connection wizard and store the database connection parameters you set in the component **Basic settings** view. For more information about setting up and storing database connection parameters, see *Setting up a DB connection* of **Talend Open Studio** User Guide.
	Database	Name of the database
	Username and *Password*	DB user authentication data.
	Table	Name of the table to be written. Note that only one table can be written at a time
	Action on table	On the table defined, you can perform one of the following operations: **None**: No operation is carried out. **Drop and create a table**: The table is removed and created again. **Create a table**: The table does not exist and gets created. **Create a table if not exists**: The table is created if it does not exist. **Drop a table if exists and create**: The table is removed if it already exists and created again. **Clear a table**: The table content is deleted.

	Action on data	On the data of the table defined, you can perform: **Insert**: Add new entries to the table. If duplicates are found, Job stops. **Update**: Make changes to existing entries. **Insert or update**: Add entries or update existing ones. **Update or insert**: Update existing entries or create it if non existing. **Delete**: Remove entries corresponding to the input flow. *It is necessary to specify at least one column as a primary key on which the Update and Delete operations are based. You can do that by clicking Edit Schema and selecting the check box(es) next to the column(s) you want to set as primary key(s). For an advanced use, click the Advanced settings view where you can simultaneously define primary keys for the Update and Delete operations. To do that: Select the Use field options check box and then in the Key in update column, select the check boxes next to the column names you want to use as a base for the Update operation. Do the same in the Key in delete column for the Delete operation.*
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either built-in or remotely stored in the Repository.
		Built-in: The schema is created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
Advanced settings	*Encoding Type*	Select the encoding type from the list or select **Custom** and define it manually. This field is compulsory for DB data handling.
	Commit every	Number of rows to be completed before committing batches of rows together into the DB. This option ensures transaction quality (but not rollback) and, above all, better performance at executions.
	Additional Columns	This option is not offered if you create (with or without drop) the DB table. This option allows you to call SQL functions to perform actions on columns, which are not insert, nor update or delete actions, or action that require particular preprocessing.
		Name: Type in the name of the schema column to be altered or inserted as new column

		SQL expression: Type in the SQL statement to be executed in order to alter or insert the relevant column data.
		Position: Select **Before**, **Replace** or **After** following the action to be performed on the reference column.
		Reference column: Type in a column of reference that the **tDBOutput** can use to place or replace the new or altered column.
	Use field options	Select this check box to customize a request, especially when there is double action on data.
	Enable debug mode	Select this check box to display each step during processing entries in a database.
	*tState*Catcher Statistics	Select this check box to collect log data at the component level.
Usage	This component offers the flexibility benefit of the DB query and covers all possibilities of SQL queries.	

Related scenarios

For related topics, see:

- **tDBOutput** *Scenario: Displaying DB output on page 168*
- **tMySQLOutput** *Scenario 1: Adding a new column and altering data in a DB table on page 283*.

tAccessRow

tAccessRow properties

Component family	Databases/Access	
Function	**tAccessRow** is the specific component for this database query. It executes the SQL query stated onto the specified database. The row suffix means the component implements a flow in the job design although it doesn't provide output.	
Purpose	Depending on the nature of the query and the database, **tAccessRow** acts on the actual DB structure or on the data (although without handling data). The SQLBuilder tool helps you write easily your SQL statements.	
Basic settings	*Property type*	Either Built-in or Repository.
		Built-in: No property data stored centrally.
		Repository: Select the repository file where Properties are stored. The fields that come after are pre-filled in using the fetched data.
	Database	Name of the database
	Username and *Password*	DB user authentication data.
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either built-in or remotely stored in the Repository.
		Built-in: The schema is created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	Query type	Either Built-in or Repository.
		Built-in: Fill in manually the query statement or build it graphically using SQLBuilder
		Repository: Select the relevant query stored in the Repository. The Query field gets accordingly filled in.
	Query	Enter your DB query paying particularly attention to properly sequence the fields in order to match the schema definition.
	Commit every	Number of rows to be completed before committing batches of rows together into the DB. This option ensures transaction quality (but not rollback) and above all better performance on executions.

	Encoding Type	Select the encoding from the list or select Custom and define it manually. This field is compulsory for DB data handling.
	Die on error	Clear this check box to skip the row on error and complete the process for error-free rows.
Advanced settings	*Propagate QUERY's recordset*	Select this check box to insert the result of the query into a COLUMN of the current flow
	Encoding Type	Select the encoding from the list or select **Custom** and define it manually. This field is compulsory for DB data handling.
	Commit every	Number of rows to be completed before committing batches of rows together into the DB. This option ensures transaction quality (but not rollback) and above all better performance on executions.
	*tState*Catcher Statistics	Select this check box to collect log data at the component level.
Usage	This component offers the flexibility benefit of the DB query and covers all possibilities of SQL queries.	

Related scenarios

For related topics, see:

- **tDBSQLRow** *Scenario: Resetting a DB auto-increment on page 172*
- **tMySQLRow** *Scenario: Removing and regenerating a MySQL table index on page 301*.

tAS400Commit

tAS400Commit Properties

This component is closely related to **tAS400Connection** and **tAS400Rollback**. It usually doesn't make much sense to use these components independently in a transaction.

Component family	Databases/AS400	
Function	Validates the data processed through the Job into the connected DB.	
Purpose	Using a unique connection, this component commits in one go a global transaction instead of doing that on every row or every batch and thus provides gain in performance.	
Basic settings	*Component list*	Select the **tAS400Connection** component in the list if more than one connection are planned for the current Job.
	Close Connection	Clear this check box to continue to use the selected connection once the component has performed its task.
Usage	This component is to be used along with AS400 components, especially with **tAS400Connection** and **tAS400Rollback** components.	
Limitation	n/a	

Related scenario

This component is closely related to **tAS400Connection** and **tAS400Rollback**. It usually doesn't make much sense to use one of these without using a **tAS400Connection** component to open a connection for the current transaction.

For **tAS400Commit** related scenario, see *tMysqlConnection on page 269.*

tAS400Connection

tAS400Connection Properties

This component is closely related to **tAS400Commit** and **tAS400Rollback**. It usually doesn't make much sense to use one of the components without using a **tAS400Connection** component to open a connection for the current transaction.

Component family	Databases/AS400	
Function	Opens a connection to the database for a current transaction.	
Purpose	Allows to commit a whole job data in one go to the output database as one transaction when validated.	
Basic settings	*Property type*	Either Built-in or Repository.
		Built-in: No property data stored centrally.
		Repository: Select the repository file where Properties are stored. The fields that come after are pre-filled in using the fetched data.
	DB Version	Select the AS400 version in use
	Host	Database server IP address
	Database	Name of the database
	Username and *Password*	DB user authentication data.
	Encoding Type	Select the encoding from the list or select Custom and define it manually. This field is compulsory for DB data handling.
	Use or register a shared DB Connection	Select this check box to share your connection or fetch a connection shared by a parent or child Job. **Shared DB Connection Name:** set or type in the shared connection name.
Usage	This component is to be used along with AS400components, especially with **tAS400Commit** and **tAS400Rollback** components.	
Limitation	n/a	

Related scenario

This component is closely related to **tAS400Commit** and **tAS400Rollback**. It usually doesn't make much sense to use one of these without using a **tAS400Connection** component to open a connection for the current transaction.

For **tAS400Connection** related scenario, see *tMysqlConnection on page 269*.

tAS400Input

tAS400Input properties

Component family	Databases/AS400	
Function	**tAS400Input** reads a database and extracts fields based on a query.	
Purpose	**tAS400SInput** executes a DB query with a strictly defined statement which must correspond to the schema definition. Then it passes on the field list to the next component via a Main row link.	
Basic settings	*Property type*	Either Built-in or Repository
		Built-in: No property data stored centrally.
		Repository: Select the repository file where Properties are stored. The fields that come after are pre-filled in using the fetched data.
		Click this icon to open a database connection wizard and store the database connection parameters you set in the component **Basic settings** view. For more information about setting up and storing database connection parameters, see *Setting up a DB connection* of **Talend Open Studio** User Guide.
	DB Version	Select the AS 400 version in use
	Host	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database
	Username and *Password*	DB user authentication data.
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either built-in or remotely stored in the Repository.
		Built-in: The schema is created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	Query type and Query	Enter your DB query paying particularly attention to properly sequence the fields in order to match the schema definition.
Advanced settings	*Additional JDBC parameters*	Specify additional connection properties in the existing DB connection.

	Encoding Type	Select the encoding type from the list or select **Custom** and define it manually. This field is compulsory for DB data handling.
	Trim all the String/Char columns	Select this check box to remove leading and trailing whitespace from all the String/Char columns.
	Trim column	Remove leading and trailing whitespace from defined columns.
	*tState*Catcher Statistics	Select this check box to collect log data at the component level.
Usage	This component offers the flexibility benefit of the DB query and covers all possibilities of SQL queries.	

Related scenarios

For related topic, see **tDBInput** scenarios:

- *Scenario 1: Displaying selected data from DB table on page 163*
- *Scenario 2: Using StoreSQLQuery variable on page 164*

Related topic in **tContextLoad** *Scenario: Dynamic context use in MySQL DB insert on page 652*.

tAS400Output

tAS400Output properties

Component family	Databases/DB2	
Function	**tAS400Output** writes, updates, makes changes or suppresses entries in a database.	
Purpose	**tAS400Output** executes the action defined on the table and/or on the data contained in the table, based on the flow incoming from the preceding component in the Job.	
Basic settings	*Property type*	Either Built-in or Repository.
		Built-in: No property data stored centrally.
		Repository: Select the repository file where Properties are stored. The fields that come after are pre-filled in using the fetched data.
		Click this icon to open a database connection wizard and store the database connection parameters you set in the component **Basic settings** view. For more information about setting up and storing database connection parameters, see *Setting up a DB connection* of **Talend Open Studio** User Guide.
	DB Version	Select the AS400 version in use
	Host	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database
	Username and *Password*	DB user authentication data.
	Table	Name of the table to be written. Note that only one table can be written at a time
	Action on table	On the table defined, you can perform one of the following operations: **None**: No operation is carried out. **Drop and create a table**: The table is removed and created again. **Create a table**: The table does not exist and gets created. **Create a table if not exists**: The table is created if it does not exist. **Drop a table if exists and create**: The table is removed if it already exists and created again. **Clear a table**: The table content is deleted.

	Action on data	On the data of the table defined, you can perform: **Insert**: Add new entries to the table. If duplicates are found, Job stops. **Update**: Make changes to existing entries **Insert or update**: Add entries or update existing ones. **Update or insert**: Update existing entries or create it if non existing **Delete**: Remove entries corresponding to the input flow. *It is necessary to specify at least one column as a primary key on which the Update and Delete operations are based. You can do that by clicking Edit Schema and selecting the check box(es) next to the column(s) you want to set as primary key(s). For an advanced use, click the Advanced settings view where you can simultaneously define primary keys for the Update and Delete operations. To do that: Select the Use field options check box and then in the Key in update column, select the check boxes next to the column names you want to use as a base for the Update operation. Do the same in the Key in delete column for the Delete operation.*
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either built-in or remotely stored in the Repository.
		Built-in: The schema is created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	Die on error	Clear this check box to skip the row on error and complete the process for error-free rows.
Advanced settings	*Additional JDBC parameters*	Specify additional connection properties in the existing DB connection.
	Encoding Type	Select the encoding type from the list or select **Custom** and define it manually. This field is compulsory for DB data handling.
	Use commit control	Select this check box to have access to the **Commit every** field where you can define the commit operation. **Commit every**: Enter the number of rows to be completed before committing batches of rows together into the DB. This option ensures transaction quality (but not rollback) and, above all, better performance at execution.

	Additional Columns	This option is not offered if you create (with or without drop) the DB table. This option allows you to call SQL functions to perform actions on columns, which are not insert, nor update or delete actions, or action that require particular preprocessing.
		Name: Type in the name of the schema column to be altered or inserted as new column
		SQL expression: Type in the SQL statement to be executed in order to alter or insert the relevant column data.
		Position: Select **Before**, **Replace** or **After** following the action to be performed on the reference column.
		Reference column: Type in a column of reference that the **tDBOutput** can use to place or replace the new or altered column.
	*Use field option*s	Select this check box to customize a request, especially when there is double action on data.
	Enable debug mode	Select this check box to display each step during processing entries in a database.
	*tState*Catcher Statistics	Select this check box to collect log data at the component level.
Usage	This component offers the flexibility benefit of the DB query and covers all possibilities of SQL queries.	

Related scenarios

For related topics, see

- **tDBOutput** *Scenario: Displaying DB output on page 168*
- **tMySQLOutput** *Scenario 1: Adding a new column and altering data in a DB table on page 283*.

tAS400Row

tAS400Row properties

Component family	Databases/AS400	
Function	**tAS400Row** is the specific component for this database query. It executes the SQL query stated onto the specified database. The row suffix means the component implements a flow in the job design although it doesn't provide output.	
Purpose	Depending on the nature of the query and the database, **tAS400Row** acts on the actual DB structure or on the data (although without handling data). The SQLBuilder tool helps you write easily your SQL statements.	
Basic settings	*Property type*	Either Built-in or Repository.
		Built-in: No property data stored centrally.
		Repository: Select the repository file where Properties are stored. The fields that come after are pre-filled in using the fetched data.
	DB Version	Select the AS400 version in use
	Use an existing connection	Select this check box and click the relevant **tAS400Connection** component on the Component list to reuse the connection details you already defined.
	Host	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database
	Username and *Password*	DB user authentication data.
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either built-in or remotely stored in the Repository.
		Built-in: The schema is created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	Query type	Either Built-in or Repository.
		Built-in: Fill in manually the query statement or build it graphically using SQLBuilder
		Repository: Select the relevant query stored in the Repository. The Query field gets accordingly filled in.

	Query	Enter your DB query paying particularly attention to properly sequence the fields in order to match the schema definition.
	Die on error	Clear this check box to skip the row on error and complete the process for error-free rows.
Advanced settings	*Additional JDBC parameters*	Specify additional connection properties in the existing DB connection
	Propagate QUERY's recordset	Select this check box to insert the result of the query into a COLUMN of the current flow.
	Encoding Type	Select the encoding from the list or select **Custom** and define it manually. This field is compulsory for DB data handling.
	Commit every	Number of rows to be completed before committing batches of rows together into the DB. This option ensures transaction quality (but not rollback) and above all better performance on executions.
	*tState*Catcher Statistics	Select this check box to collect log data at the component level.
Usage	This component offers the flexibility benefit of the DB query and covers all possibilities of SQL queries.	

Related scenarios

For related topics, see:

- **tDBSQLRow** *Scenario: Resetting a DB auto-increment on page 172*
- **tMySQLRow** *Scenario: Removing and regenerating a MySQL table index on page 301*.

tCreateTable

You can find this component at the root of **Databases** group of the **Palette** of **Talend Open Studio**. **tCreateTable** covers needs related indirectly to the use of any database.

tCreateTable Properties

Component family	Databases	
Function	**tCreateTable** creates, drops and creates or clear the specified table.	
Purpose	This Java specific component helps create or drop any database table	
Basic settings	*Property type*	Either **Built-in** or **Repository**.
		Built-in: No property data stored centrally.
		Repository: Select the repository file where properties are stored. The fields that follow are pre-filled in using fetched data.
	Database Type	Select the DBMS type from the list.  The component properties may differ slightly according to the database type selected from the list.
	Table Action	Select the action to be carried out on the database among: **Create table**: when you know already that the table doesn't exist. **Create table when not exists**: when you don't know whether the table is already created or not **Drop and create table**: when you know that the table exists already and needs to be replaced.
MySQL	*Temporary table*	Select this check box if you want to save the created table temporarily.
MSSQLServer, MySQL, Oracle, PostgresPlus, Postgresql	*Use existing connection*	Select this check box in case you use a database connection component.
Oracle	*Connection* Type	Drop-down list of available drivers.
Access	*Access File*	Name and path of the file to be processed. Related topic:*Defining variables from the Component view* of **Talend Open Studio** User Guide.
Firebird	*Firebird File*	Name and path of the file to be processed. Related topic:*Defining variables from the Component view* of **Talend Open Studio** User Guide.
Interbase	*Interbase File*	Name and path of the file to be processed. Related topic:*Defining variables from the Component view* of **Talend Open Studio** User Guide.
SQLite	*SQLite File*	Name and path of the file to be processed. Related topic:*Defining variables from the Component view* of **Talend Open Studio** User Guide.

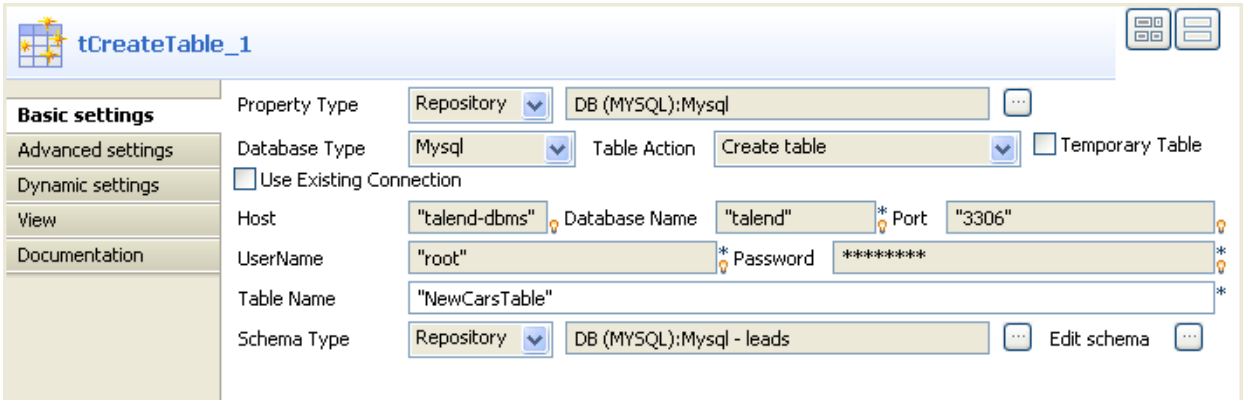
JavaDb	*Framework Type*	Select from the list a framework for your database.
HSQLDb	*Running Mode*	Select from the list the Server Mode that correspond to your DB setup.
HSQLDb	*Use TLS/SSL Sockets*	Select this check box to enable the secured mode, if required.
AS400/Oracle	*DB Version*	Select the database version in use.
All database types except Access, JavaDb, SQLite and ODBC	*Host*	Database server IP address
All database types except Access, Firebird, HSQLDb, SQLite and ODBC	*Database name*	Name of the database.
JavaDb	*DB Root Path*	Browse to your database root.
All database types except Access, AS400, Firebird, Interbase, JavaDb, SQLite and ODBC	*Port*	Listening port number of the DB server.
HSQLDb	*DB Alias*	Name of the database.
Informix	*DB Server*	Name of the database server.
ODBC	*ODBC Name*	Name of the database.
	Username and *Password*	DB user authentication data.
	Table name	Type in between quotes a name for the newly created table.
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either built-in or remotely stored in the Repository.  Reset the DB type by clicking the relevant button, to make sure data type is correct.
		Built-in: The schema is created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
Advanced settings	*Encoding*	Select the encoding from the list or select **Custom** and define it manually. This field is compulsory for DB data handling.
	tStatcatcher Statistics	Select this check box to gather the job processing metadata at a Job level as well as at each component level.
AS400/ MSSQL Server	*Additional JDBC Parameters*	Specify additional connection properties in the existing DB connection.

Usage	This component offers the flexibility benefit of the DB query and covers all possibilities of SQL queries. More scenarios are available for specific DB Input components

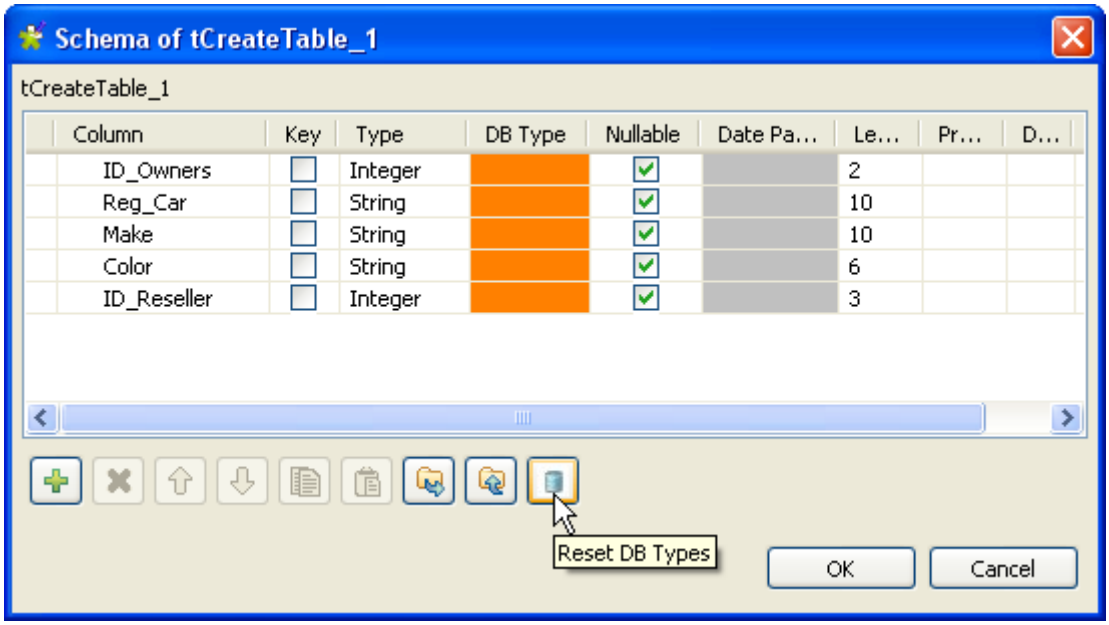
Scenario: Creating new table in a Mysql Database

The Job described below aims at creating a table in a database, made of a dummy schema taken from a delimited file schema stored in the Repository. This Job is composed of a single component.

- Drop a **tCreateTable** component from the **Databases** family in the **Palette** to the design workspace.
- In the **Basic settings** view, and from the **Database Type** list, select Mysql for this scenario.

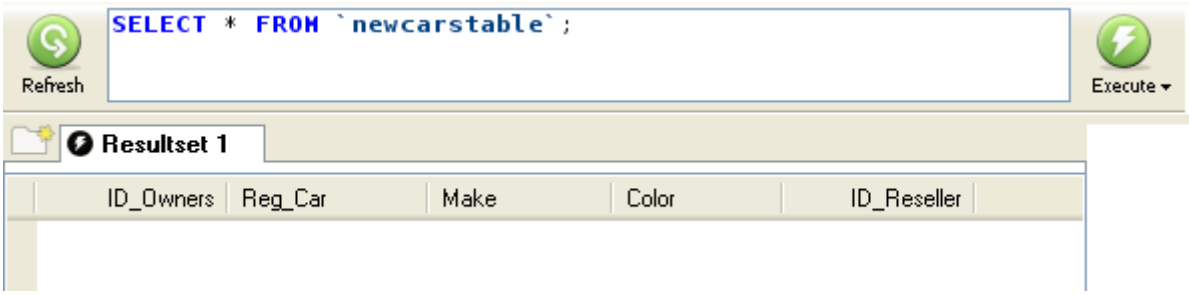


- From the **Table Action** list, select **Create table**.
- Select the **Use Existing Connection** check box only if you are using a dedicated DB connection component, see *tMysqlConnection on page 269*. In this example, we won't use this option.
- In the **Property type** field, select **Repository** so that the connection fields that follow are automatically filled in. If you have not defined a metadata DB connection entry for your DB connection, fill in manually the details as **Built-in**.
- In the **Table Name** field, fill in a name for the table to be created.
- If you want to retrieve the **Schema** from the Metadata (it doesn't need to be a DB connection Schema metadata), select Repository then the relevant entry.
- In any case (**Built-in** or **Repository**) click **Edit Schema** to check the data type mapping.



- Click the **Reset DB Types** button in case the DB type column is empty or shows discrepancies marks (orange color). This allows to map any data type to the relevant DB data type.
- Click **OK** to validate your changes and close the dialog box.
- Save your Job and press **F6** to execute it.

The table is created empty but with all columns defined in the Schema.



tDB2BulkExec

tDB2BulkExec properties

Component family	Databases/DB2	
Function	**tDB2BulkExec** executes the Insert action on the data provided.	
Purpose	As a dedicated component, **tDB2BulkExec** allows gains in performance during Insert operations to a DB2 database.	
Basic settings	*Property type*	Either **Built-in** or **Repository**.
		Built-in: No property data stored centrally.
		Repository: Select the repository file where Properties are stored. The fields that come after are pre-filled in using the fetched data.
	Use an existing connection	Select this check box and click the relevant **tDB2Connection** component on the **Component List** to reuse the connection details you already defined.
	Host	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database
	Table Schema	Name of the DB schema.
	Username and *Password*	DB user authentication data.
	Table	Name of the table to be written. Note that only one table can be written at a time
	Action on table	On the table defined, you can perform one of the following operations: **None**: No operation is carried out. **Drop and create table**: The table is removed and created again. **Create table**: The table does not exist and gets created. **Create table if not exists**: The table is created if it does not exist. **Drop table if exists and create**: The table is removed if it already exists and created again. **Clear table**: The table content is deleted.
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either built-in or remotely stored in the Repository.
		Built-in: You create the schema and store it locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.

		Repository: You have already created the schema and stored it in the Repository, hence can reuse it. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	Data file	Name of the file to be processed. Related topic:*Defining variables from the Component view* of **Talend Open Studio** User Guide.
	Action on data	On the data of the table defined, you can perform: **Insert**: Add new entries to the table. If duplicates are found, Job stops. **Update**: Make changes to existing entries **Insert or update**: Add entries or update existing ones. **Update or insert**: Update existing entries or create it if non existing **Delete**: Remove entries corresponding to the input flow.
Advanced settings	*Field terminated by*	Character, string or regular expression to separate fields.
	Date Format	Use this field to define the way months and days are ordered.
	Time Format	Use this field to define the way hours, minutes and seconds are ordered.
	Timestamp Format	Use this field to define the way date and time are ordered.
	*tState*Catcher Statistics	Select this check box to collect log data at the component level.
Usage	This dedicated component offers performance and flexibility of DB2 query handling.	

Related scenarios

For **tDB2BulkExec** related topics, see:

- **tMysqlOutputBulkExec** *Scenario: Inserting transformed data in MySQL database on page 293*.
- **tOracleBulkExec** *Scenario: Truncating and inserting file data into Oracle DB on page 329*.

tDB2Input

tDB2Input properties

Component family	Databases/DB2	
Function	**tDB2Input** reads a database and extracts fields based on a query.	
Purpose	**tDB2Input** executes a DB query with a strictly defined order which must correspond to the schema definition. Then it passes on the field list to the next component via a **Main** row link.	
Basic settings	*Property type*	Either **Built-in** or **Repository**.
		Built-in: No property data stored centrally.
		Repository: Select the repository file where Properties are stored. The fields that come after are pre-filled in using the fetched data.
	Use an existing connection	Select this check box and click the relevant **tDB2Connection** component on the Component list to reuse the connection details you already defined.
		Click this icon to open a database connection wizard and store the database connection parameters you set in the component **Basic settings** view. For more information about setting up and storing database connection parameters, see *Setting up a DB connection* of **Talend Open Studio** User Guide.
	Host	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database
	Username and *Password*	DB user authentication data.
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either built-in or remotely stored in the Repository.
		Built-in: The schema is created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	Query type and *Query*	Enter your DB query paying particularly attention to properly sequence the fields in order to match the schema definition.
Advanced settings	*Encoding Type*	Select the encoding type from the list or select **Custom** and define it manually. This field is compulsory for DB data handling.

	Trim all the String/Char columns	Select this check box to remove leading and trailing whitespace from all the String/Char columns.
	Trim column	Remove leading and trailing whitespace from defined columns.
	*tState*Catcher Statistics	Select this check box to collect log data at the component level.
Usage	This component covers all possibilities of SQL queries onto a DB2 database.	

Related scenarios

For related topics, see **tDBInput** scenarios:

- *Scenario 1: Displaying selected data from DB table on page 163*
- *Scenario 2: Using StoreSQLQuery variable on page 164*

See also the related topic in **tContextLoad** *Scenario: Dynamic context use in MySQL DB insert on page 652*.

tDB2Output

tDB2Output properties

Component family	Databases/DB2	
Function	**tDB2Output** writes, updates, makes changes or suppresses entries in a database.	
Purpose	**tDB2Output** executes the action defined on the table and/or on the data contained in the table, based on the flow incoming from the preceding component in the Job.	
Basic settings	*Property type*	Either Built-in or Repository.
		Built-in: No property data stored centrally.
		Repository: Select the repository file where Properties are stored. The fields that come after are pre-filled in using the fetched data.
	Use an existing connection	Select this check box and click the relevant **tDB2Connection** component on the Component list to reuse the connection details you already defined.
		Click this icon to open a database connection wizard and store the database connection parameters you set in the component **Basic settings** view. For more information about setting up and storing database connection parameters, see *Setting up a DB connection* of **Talend Open Studio** User Guide.
	Host	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database
	Username and *Password*	DB user authentication data.
	Table	Name of the table to be written. Note that only one table can be written at a time
	Action on table	On the table defined, you can perform one of the following operations: **None**: No operation is carried out. **Drop and create a table**: The table is removed and created again. **Create a table**: The table does not exist and gets created. **Create a table if not exists**: The table is created if it does not exist. **Drop a table if exists and create**: The table is removed if it already exists and created again. **Clear a table**: The table content is deleted.

	Action on data	On the data of the table defined, you can perform: **Insert**: Add new entries to the table. If duplicates are found, Job stops. **Update**: Make changes to existing entries **Insert or update**: Add entries or update existing ones. **Update or insert**: Update existing entries or create it if non existing **Delete**: Remove entries corresponding to the input flow. *It is necessary to specify at least one column as a primary key on which the Update and Delete operations are based. You can do that by clicking Edit Schema and selecting the check box(es) next to the column(s) you want to set as primary key(s). For an advanced use, click the Advanced settings view where you can simultaneously define primary keys for the Update and Delete operations. To do that: Select the Use field options check box and then in the Key in update column, select the check boxes next to the column names you want to use as a base for the Update operation. Do the same in the Key in delete column for the Delete operation.*
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either built-in or remotely stored in the Repository.
		Built-in: The schema is created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	Die on error	Clear this check box to skip the row on error and complete the process for error-free rows.
Advanced settings	*Encoding Type*	Select the encoding type from the list or select **Custom** and define it manually. This field is compulsory for DB data handling.
	Commit every	Enter the number of rows to be completed before committing batches of rows together into the DB. This option ensures transaction quality (but not rollback) and, above all, better performance at execution.
	Additional Columns	This option is not offered if you create (with or without drop) the DB table. This option allows you to call SQL functions to perform actions on columns, which are not insert, nor update or delete actions, or action that require particular preprocessing.

		Name: Type in the name of the schema column to be altered or inserted as new column
		SQL expression: Type in the SQL statement to be executed in order to alter or insert the relevant column data.
		Position: Select **Before**, **Replace** or **After** following the action to be performed on the reference column.
		Reference column: Type in a column of reference that the **tDBOutput** can use to place or replace the new or altered column.
	*Use field option*s	Select this check box to customize a request, especially when there is double action on data.
	Enable debug mode	Select this check box to display each step during processing entries in a database.
	*tState*Catcher Statistics	Select this check box to collect log data at the component level.
Usage	This component offers the flexibility benefit of the DB query and covers all possibilities of SQL queries.	

Related scenarios

For **tDB2Output** related topics, see

- **tDBOutput** *Scenario: Displaying DB output on page 168*
- **tMySQLOutput** *Scenario 1: Adding a new column and altering data in a DB table on page 283*.

tDB2Row

tDB2Row properties

Component family	Databases/DB2	
Function	**tDB2Row** is the specific component for this database query. It executes the SQL query stated onto the specified database. The row suffix means the component implements a flow in the job design although it doesn't provide output.	
Purpose	Depending on the nature of the query and the database, **tDB2Row** acts on the actual DB structure or on the data (although without handling data). The SQLBuilder tool helps you write easily your SQL statements.	
Basic settings	*Property type*	Either Built-in or Repository.
		Built-in: No property data stored centrally.
		Repository: Select the repository file where Properties are stored. The fields that come after are pre-filled in using the fetched data.
	Use an existing connection	Select this check box and click the relevant **tDB2Connection** component on the Component list to reuse the connection details you already defined.
	Host	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database
	Username and *Password*	DB user authentication data.
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either built-in or remotely stored in the Repository.
		Built-in: The schema is created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	Query type	Either Built-in or Repository.
		Built-in: Fill in manually the query statement or build it graphically using SQLBuilder
		Repository: Select the relevant query stored in the Repository. The Query field gets accordingly filled in.
	Query	Enter your DB query paying particularly attention to properly sequence the fields in order to match the schema definition.

	Die on error	Clear this check box to skip the row on error and complete the process for error-free rows.
Advanced settings	*Propagate QUERY's recordset*	Select this check box to insert the result of the query into a COLUMN of the current flow.
	Encoding Type	Select the encoding from the list or select **Custom** and define it manually. This field is compulsory for DB data handling.
	Commit every	Number of rows to be completed before committing batches of rows together into the DB. This option ensures transaction quality (but not rollback) and above all better performance on executions.
	*tState*Catcher Statistics	Select this check box to collect log data at the component level.
Usage	This component offers the flexibility benefit of the DB query and covers all possibilities of SQL queries.	

Related scenarios

For **tDB2Row** related topics, see:

- **tDBSQLRow** *Scenario: Resetting a DB auto-increment on page 172*
- **tMySQLRow** *Scenario: Removing and regenerating a MySQL table index on page 301*.

tDB2SCD

tDB2SCD belongs to two component families: Business Intelligence and Databases. For more information on it, see *tDB2SCD on page 2*.

tDB2SP

tDB2SP properties

Component family	Databases/DB2	
Function	**tDB2SP** calls the database stored procedure.	
Purpose	**tDB2SP** offers a convenient way to centralize multiple or complex queries in a database and call them easily.	
Basic settings	*Property type*	Either Built-in or Repository.
		Built-in: No property data stored centrally.
		Repository: Select the repository file where Properties are stored. The fields that come after are pre-filled in using the fetched data.
	Host	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database
	Username and *Password*	DB user authentication data.
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either built-in or remotely stored in the Repository.
		Built-in: The schema is created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	SP Name	Type in the exact name of the Stored Procedure
	Is Function / *Return result in*	Check this box, if a value only is to be returned. Select on the list the schema column, the value to be returned is based on.

	Parameters	Click the Plus button and select the various **Schema Columns** that will be required by the procedures. Note that the SP schema can hold more columns than there are parameters used in the procedure. Select the **Type** of parameter: **IN**: Input parameter **OUT**: Output parameter/return value **IN OUT**: Input parameters is to be returned as value, likely after modification through the procedure (function). **RECORDSET**: Input parameters is to be returned as a set of values, rather than single value. Check the *tParseRecordSet* component if you want to analyze a set of records from a database table or DB query and return single records.
	Encoding Type	Select the encoding from the list or select Custom and define it manually. This field is compulsory for DB data handling.
Usage	This component is used as intermediary component. It can be used as start component but only input parameters are thus allowed.	
Limitation	The Stored Procedures syntax should match the Database syntax.	

Related scenarios

For related topic, see **tMysqlSP** *Scenario: Finding a State Label using a stored procedure on page 307*.

Check as well the *tParseRecordSet* component if you want to analyze a set of records from a database table or DB query and return single records.

tDBInput

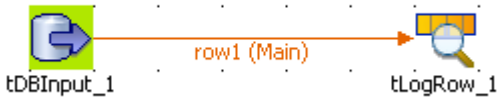
tDBInput properties

Component family	Databases/DB Generic	
Function	**tDBInput** reads a database and extracts fields based on a query.	
Purpose	**tDBInput** executes a DB query with a strictly defined order which must correspond to the schema definition. Then it passes on the field list to the next component via a Main row link.  For performance reasons, a specific Input component (e.g.: **tMySQLInput** for MySQL database) should always be preferred to the generic component.	
Basic settings	*Property type*	Either Built-in or Repository
		Built-in: No property data stored centrally.
		Repository: Select the repository file where Properties are stored. The fields that come after are pre-filled in using the fetched data.
		Click this icon to open a database connection wizard and store the database connection parameters you set in the component **Basic settings** view. For more information about setting up and storing database connection parameters, see *Setting up a DB connection* of **Talend Open Studio** User Guide.
	Connection type	Drop-down list of available DBMS drivers.
	Database	Name of the database
	Username and *Password*	DB user authentication data.
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either built-in or remotely stored in the Repository.
		Built-in: The schema is created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	Query	Enter your DB query paying particularly attention to properly sequence the fields in order to match the schema definition.
Advanced settings	*Encoding Type*	Select the encoding type from the list or select **Custom** and define it manually. This field is compulsory for DB data handling.

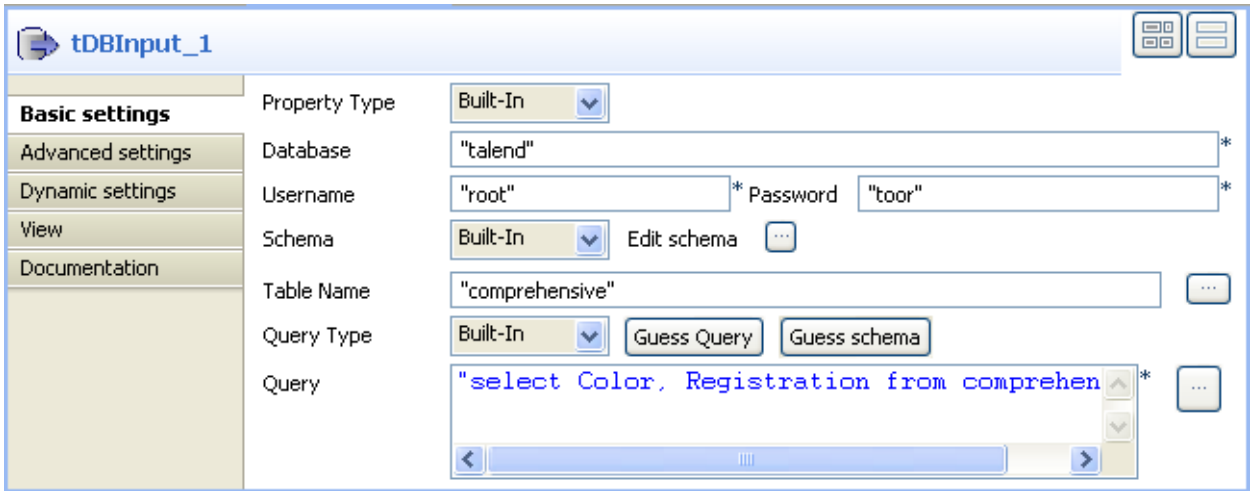
	Trim all the String/Char columns	Select this check box to remove leading and trailing whitespace from all the String/Char columns.
	Trim column	Remove leading and trailing whitespace from defined columns.
	*tState*Catcher Statistics	Select this check box to collect log data at the component level.
Usage	This component offers the flexibility benefit of the DB query and covers all possibilities of SQL queries using a generic ODBC connection.	

Scenario 1: Displaying selected data from DB table

The following scenario creates a two-component Job, reading data from a database using a DB query and outputting delimited data into the standard output (console).



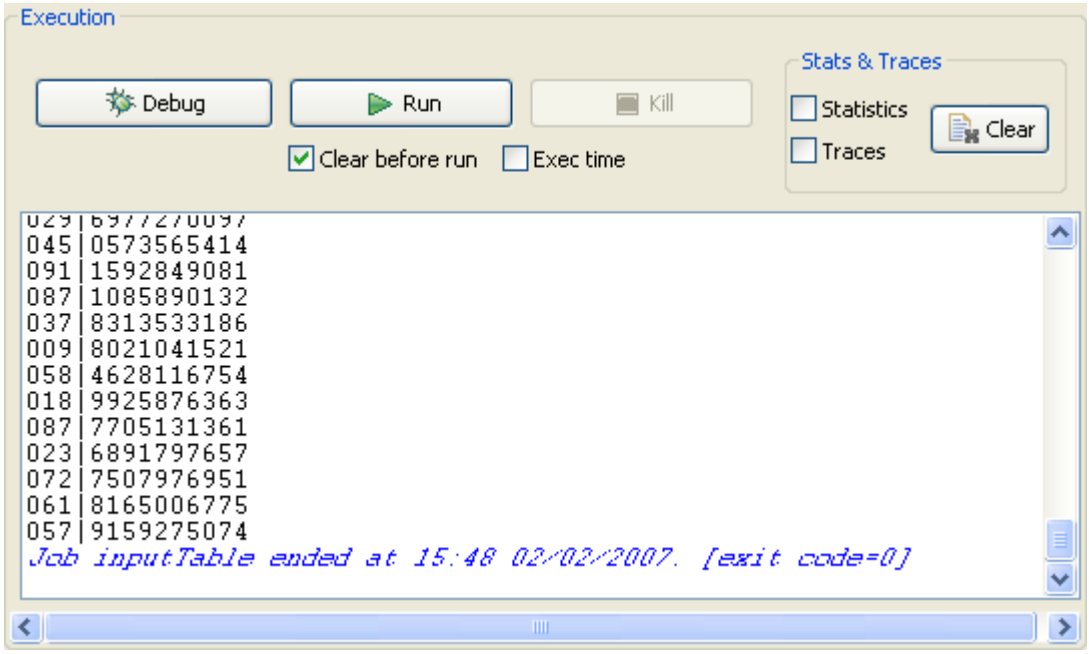
- Drop a **tDBInput** and **tLogRow** component from the **Palette** to the design workspace.
- Right-click on the **tDBInput** component and select *Row* > *Main*. Drag this main row link onto the **tLogRow** component and release when the plug symbol displays.
- Double-click the **tDBInput** so the **Component** view shows up, and define the properties:



- The component property data are **Built-In** for this scenario.
- Fill in the database name, the username and password in the corresponding fields.
- The schema is **Built-In**.This means that it is available for this Job and on this station only.
- Click on **Edit Schema** and create a 2-column description including shop code and sales.
- Enter the table name in the corresponding field.

- Type in the query making sure it includes all columns in the same order as defined in the Schema. In this case, as we'll select all columns of the schema, the asterisk symbol makes sense.
- Click on the second component to define it.
- Enter the fields separator. In this case, a pipe separator.
- Now go to the **Run** tab, and click on **Run** to execute the Job.

The DB is parsed and queried data is extracted from the specified table and passed on to the job log console. You can view the output file straight on the console.



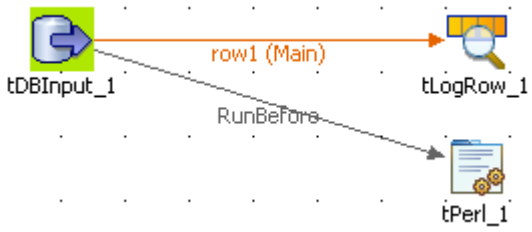
Scenario 2: Using StoreSQLQuery variable

StoreSQLQuery is a variable that can be used to debug a tDBInput scenario which does not operate correctly. It allows you to dynamically feed the SQL query set in your **tDBInput** component.

- Use the same scenario as scenario 1 above and add a third component, **tPerl**.

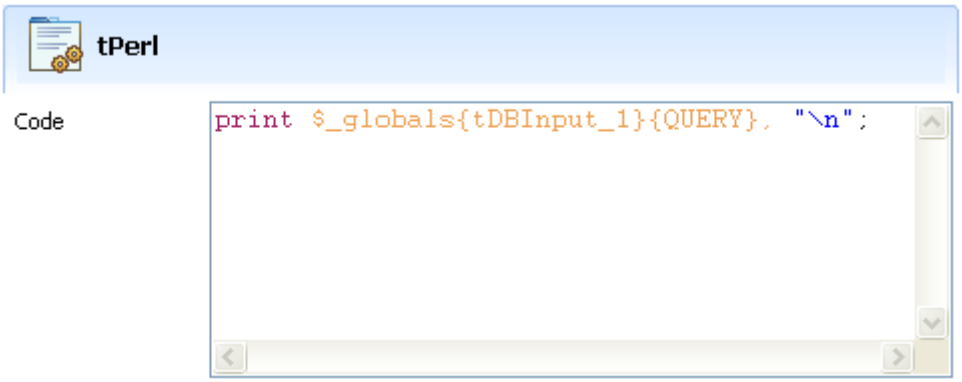
If you create scenario 1 using the Java language, you must use **tJava** in this step.

- Connect **tDBInput** component to **tPerl** component using a trigger connection of **ThenRun** type. In this case, we want the **tDBInput** to run before the **tPerl** component.

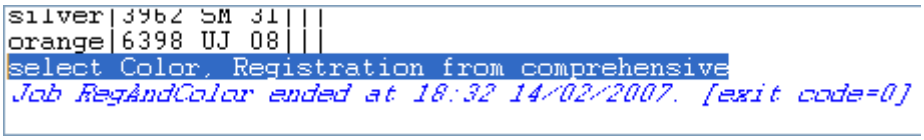


- Set both **tDBInput** and **tLogRow** component as in **tDBInput** scenario 1.

- Click anywhere on the design workspace to display the **Contexts** property panel.
- Create a new parameter called explicitly **StoreSQLQuery**. Enter a default value of 1. This value of 1 means the **StoreSQLQuery** is “true” for a use in the QUERY global variable.
- Click on the **tPerl** component and display the **Component** view. Enter the command Print to display the query content, press **Ctrl+Space bar** to access the variable list and select the global variable QUERY.



- Go to your **Run tab** and execute the Job.
- The query entered in the **tDBInput** component shows at the end of the job results, on the log:



tDBOutput

tDBOutput properties

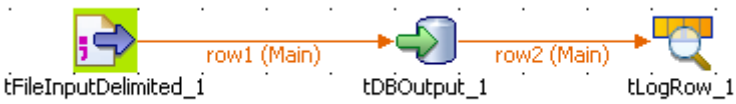
Component family	Databases	
Function	**tDBOutput** writes, updates, makes changes or suppresses entries in a database.	
Purpose	**tDBOutput** executes the action defined on the table and/or on the data contained in the table, based on the flow incoming from the preceding component in the Job. Specific Output component should always be preferred to generic component.	
Basic settings	*Property type*	Either Built-in or Repository.
		Built-in: No property data stored centrally.
		Repository: Select the repository file where Properties are stored. The fields that come after are pre-filled in using the fetched data.
		Click this icon to open a database connection wizard and store the database connection parameters you set in the component **Basic settings** view. For more information about setting up and storing database connection parameters, see *Setting up a DB connection* of **Talend Open Studio** User Guide.
	Connection type	List of available drivers.
	Database	Name of the database
	Username and *Password*	DB user authentication data.
	Table	Name of the table to be written. Note that only one table can be written at a time
	Action on table	On the table defined, you can perform one of the following operations: **None**: No operation is carried out. **Drop and create a table**: The table is removed and created again. **Create a table**: The table does not exist and gets created. **Create a table if not exists**: The table is created if it does not exist. **Drop a table if exists and create**: The table is removed if it already exists and created again. **Clear a table**: The table content is deleted.

	Action on data	On the data of the table defined, you can perform: **Insert**: Add new entries to the table. If duplicates are found, Job stops. **Update**: Make changes to existing entries **Insert or update**: Add entries or update existing ones. **Update or insert**: Update existing entries or create it if non existing **Delete**: Remove entries corresponding to the input flow. *It is necessary to specify at least one column as a primary key on which the Update and Delete operations are based. You can do that by clicking Edit Schema and selecting the check box(es) next to the column(s) you want to set as primary key(s). For an advanced use, click the Advanced settings view where you can simultaneously define primary keys for the Update and Delete operations. To do that: Select the Use field options check box and then in the Key in update column, select the check boxes next to the column names you want to use as a base for the Update operation. Do the same in the Key in delete column for the Delete operation.*
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either built-in or remotely stored in the Repository.
		Built-in: The schema is created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	Die on error	Clear this check box to skip the row on error and complete the process for error-free rows.
Advanced settings	*Encoding Type*	Select the encoding type from the list or select **Custom** and define it manually. This field is compulsory for DB data handling.
	Commit every	Enter the number of rows to be completed before committing batches of rows together into the DB. This option ensures transaction quality (but not rollback) and, above all, better performance at execution.
	Additional Columns	This option is not offered if you create (with or without drop) the DB table. This option allows you to call SQL functions to perform actions on columns, which are not insert, nor update or delete actions, or action that require particular preprocessing.

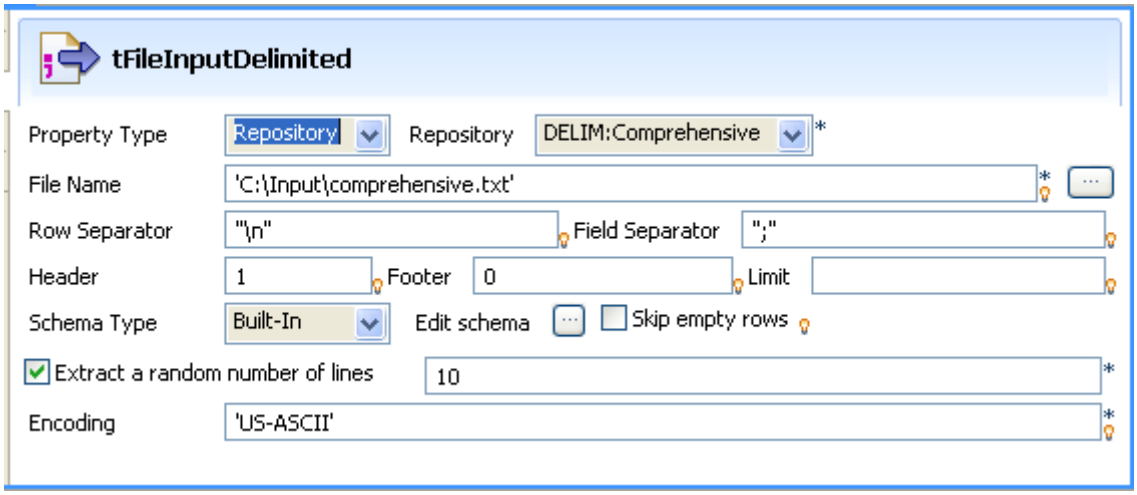
		Name: Type in the name of the schema column to be altered or inserted as new column
		SQL expression: Type in the SQL statement to be executed in order to alter or insert the relevant column data.
		Position: Select **Before**, **Replace** or **After** following the action to be performed on the reference column.
		Reference column: Type in a column of reference that the **tDBOutput** can use to place or replace the new or altered column.
	Use field options	Select this check box to customize a request, especially when there is double action on data.
	Enable debug mode	Select this check box to display each step during processing entries in a database.
	*tState*Catcher Statistics	Select this check box to collect log data at the component level.
Usage	This component offers the flexibility benefit of the DB query and covers all possibilities of SQL queries.	

Scenario: Displaying DB output

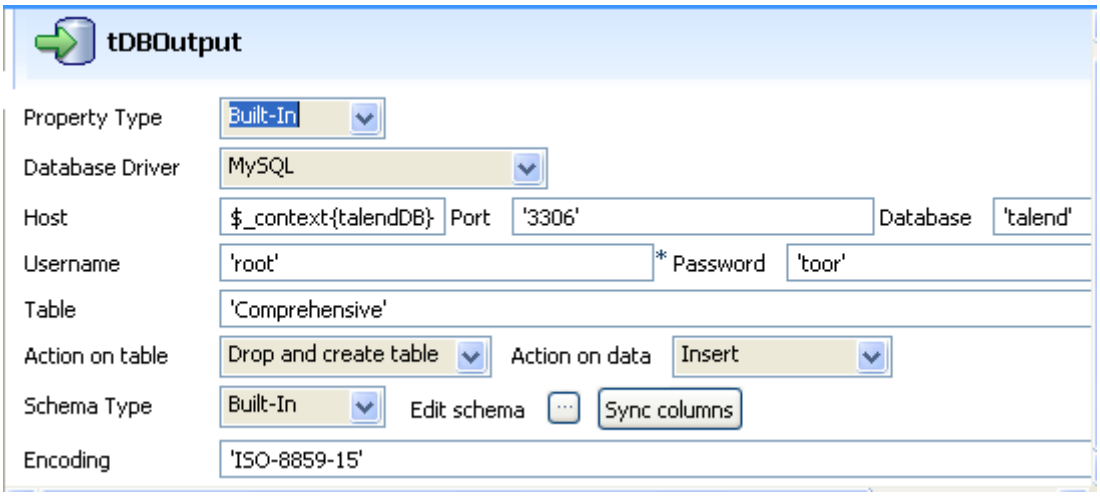
This following scenario is a three-component Job aiming at creating a new table in the database defined and filling it with data. The **tFileInputdelimited** passes on the Input flow to the **tDBoutput** component. As the content of a DB is not viewable as such, a **tLogRow** component is used to display the main flow on the **Run** console.



- Drop the three components required for this Job from the **Palette** to the design workspace.
- On the **Basic settings** tab of **tFileInputDelimited**, define the input flow parameters. In this use case, the file contains cars' owner id, makes, color and registration references organized as follows: semi-colon as field separator, carriage return as row separator. The input file contains a header row to be considered in the schema. If this file is already described in your metadata, you can retrieve the properties by selecting the relevant repository entry list.

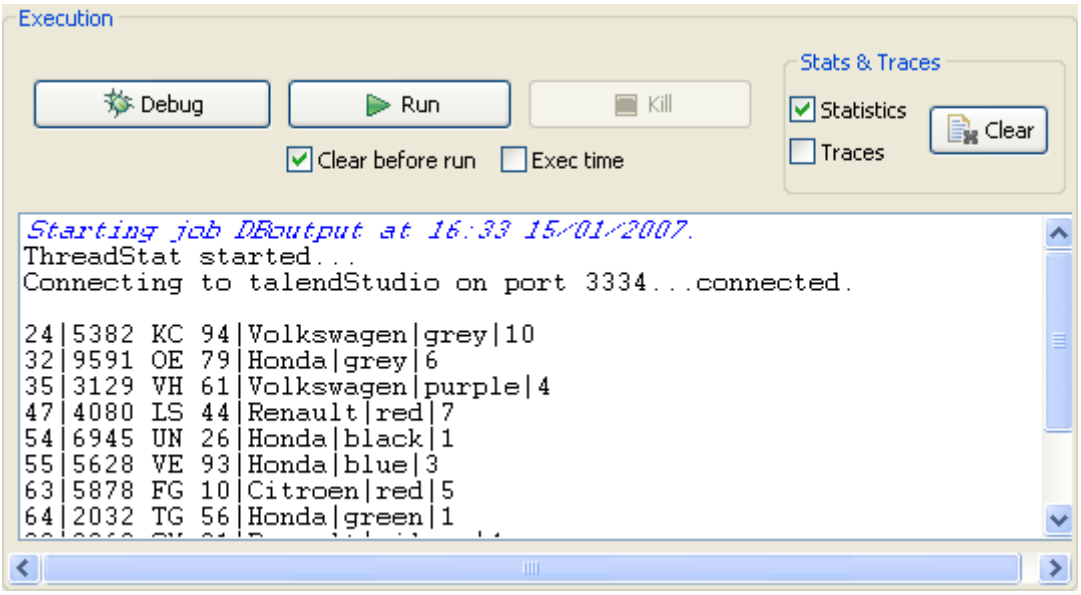


- And also, if your schema is already loaded in the Repository, select **Repository** as **Schema type** and choose the relevant metadata entry in the list. If you haven't defined the schema already, define the data structure in the built-in schema you edit.
- Restrict the extraction to 10 lines, for this example.
- Then define the **tDBOutput** component to configure the output flow. Select the database to connect to. Note that you can store all the database connection details in different context variables. For more information about how to create and use context variables *Defining Contexts and variables* in **Talend Open Studio** User Guide.



- Fill in the table name in the **Table** field. Then select the operations to be performed:
- As **Action on table**, select **Drop and create table** in the list. This allows you to overwrite the possible existing table with the new selected data. Alternatively you can insert only extra rows into an existing table, but note that duplicate management is not supported natively. See *tUniqRow Properties on page 126* for further information.
- As **Action on data**, select **Insert**. The data flow incoming as input will be thus added to the selected table.
- To view the output flow easily, connect the DBOuput component to an tLogRow component. Define the field separator as a pipe symbol. Press **F6** to execute the Job.

- As the processing can take some time to reach the **tLogRow** component, we recommend you to enable the **Statistics** functionality on the **Run** console.



Related topic: *tMysqlOutput properties on page 281*

tDBSQLRow

tDBSQLRow properties

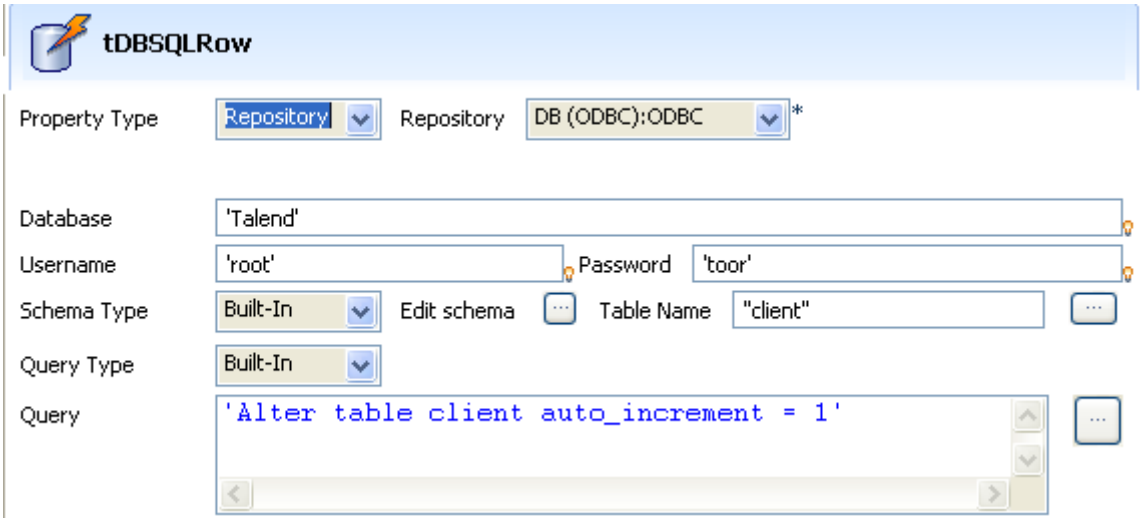
Component family	Databases/DB Generic	
Function	**tDBSQLRow** is the generic component for database query. It executes the SQL query stated onto the specified database. The row suffix means the component implements a flow in the job design although it doesn't provide output. For performance reasons, specific DB component should always be preferred to the generic component.	
Purpose	Depending on the nature of the query and the database, **tDBSQLRow** acts on the actual DB structure or on the data (although without handling data). The SQLBuilder tool helps you write easily your SQL statements.	
Basic settings	*Property type*	Either **Built-in** or **Repository**.
		Built-in: No property data stored centrally.
		Repository: Select the repository file where Properties are stored. The fields that come after are pre-filled in using the fetched data.
	Database	Name of the database
	Username and *Password*	DB user authentication data.
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either built-in or remotely stored in the Repository.
		Built-in: The schema is created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	Query type	Either Built-in or Repository.
		Built-in: Fill in manually the query statement or build it graphically using SQLBuilder
		Repository: Select the relevant query stored in the Repository. The Query field gets accordingly filled in.
	Query	Enter your DB query paying particularly attention to properly sequence the fields in order to match the schema definition.
Advanced settings	*Propagate QUERY's recordset*	Select this check box to insert the result of the query into a COLUMN of the current flow.

	Encoding Type	Select the encoding from the list or select **Custom** and define it manually. This field is compulsory for DB data handling.
	Commit every	Number of rows to be completed before committing batches of rows together into the DB. This option ensures transaction quality (but not rollback) and above all better performance on executions.
	*tState*Catcher Statistics	Select this check box to collect log data at the component level.
Usage	This component offers the flexibility benefit of the DB query and covers all possibilities of SQL queries. **Note**: Use the relevant DBRow component according to the DB type you use. Most of databases have their specific DBRow components.	

Scenario: Resetting a DB auto-increment

This scenario describes a single component Job which aims at re-initializing the DB auto-increment to 1. This job has no output and is generally to be used before running a script.

- Drag and drop a **tDBSQLRow** component from the **Palette** to the design workspace.
- On the **Basic settings** panel, fill in the DB connection properties.



- The general connection information to the database is stored in the Repository. The **Database Driver** is a generic ODBC driver.
- The **Schema type** is built-in for this Job and describes the Talend database structure. The schema doesn't really matter for this particular instance of Job as the action is made on the table auto-increment and not on data.

- The **Query type** is also built-in. Click on the three dot button to launch the SQLbuilder editor, or else type in directly in the Query area:
  Alter table <TableName> auto_increment = 1
- Then click **OK** to validate the **Basic settings**. Then press **F6** to run the Job.

The database autoincrement is reset to 1.

Related topics: *tMysqlRow properties on page 300*.

tFirebirdInput

tFirebirdInput properties

Component family	Databases/FireBird	
Function	**tFirebirdInput** reads a database and extracts fields based on a query.	
Purpose	**tFirebirdInput** executes a DB query with a strictly defined order which must correspond to the schema definition. Then it passes on the field list to the next component via a Main row link.	
Basic settings	*Property type*	Either Built-in or Repository
		Built-in: No property data stored centrally.
		Repository: Select the repository file where Properties are stored. The fields that come after are pre-filled in using the fetched data.
	Host	Database server IP address
	Database	Name of the database
	Username and *Password*	DB user authentication data.
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either built-in or remotely stored in the Repository.
		Built-in: The schema is created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	Query type and *Query*	Enter your DB query paying particularly attention to properly sequence the fields in order to match the schema definition.
Advanced settings	*Additional JDBC parameters*	Specify additional connection properties in the existing DB connection.
	Encoding Type	Select the encoding type from the list or select **Custom** and define it manually. This field is compulsory for DB data handling.
	Enable stream	When selected, helps to decide the row set to work with at a time and thus optimize performance.
	Trim all the String/Char columns	Select this check box to remove leading and trailing whitespace from all the String/Char columns.
	Trim column	Remove leading and trailing whitespace from defined columns.

	*tState*Catcher Statistics	Select this check box to collect log data at the component level.
Usage	This component covers all possibilities of SQL queries onto a FireBird database.	

Related scenarios

For related topics, see generic **tDBInput** scenarios:

- *Scenario 1: Displaying selected data from DB table on page 163*
- *Scenario 2: Using StoreSQLQuery variable on page 164*

See also related topic in **tContextLoad** *Scenario: Dynamic context use in MySQL DB insert on page 652*.

tFirebirdOutput

tFirebirdOutput properties

Component family	Databases/FireBird	
Function	**tFirebirdOutput** writes, updates, makes changes or suppresses entries in a database.	
Purpose	**tFirebirdOutput** executes the action defined on the table and/or on the data contained in the table, based on the flow incoming from the preceding component in the Job.	
Basic settings	*Property type*	Either Built-in or Repository.
		Built-in: No property data stored centrally.
		Repository: Select the repository file where Properties are stored. The fields that come after are pre-filled in using the fetched data.
	Host	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database
	Username and *Password*	DB user authentication data.
	Table	Name of the table to be written. Note that only one table can be written at a time
	Action on table	On the table defined, you can perform one of the following operations: **None**: No operation is carried out. **Drop and create a table**: The table is removed and created again. **Create a table**: The table does not exist and gets created. **Create a table if not exists**: The table is created if it does not exist. **Drop a table if exists and create**: The table is removed if it already exists and created again. **Clear a table**: The table content is deleted.

	Action on data	On the data of the table defined, you can perform: **Insert**: Add new entries to the table. If duplicates are found, Job stops. **Update**: Make changes to existing entries **Insert or update**: Add entries or update existing ones. **Update or insert**: Update existing entries or create it if non existing **Delete**: Remove entries corresponding to the input flow. *It is necessary to specify at least one column as a primary key on which the Update and Delete operations are based. You can do that by clicking Edit Schema and selecting the check box(es) next to the column(s) you want to set as primary key(s). For an advanced use, click the Advanced settings view where you can simultaneously define primary keys for the Update and Delete operations. To do that: Select the Use field options check box and then in the Key in update column, select the check boxes next to the column names you want to use as a base for the Update operation. Do the same in the Key in delete column for the Delete operation.*
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either built-in or remotely stored in the Repository.
		Built-in: The schema is created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	Die on error	Clear this check box to skip the row on error and complete the process for error-free rows.
Advanced settings	*Encoding Type*	Select the encoding type from the list or select **Custom** and define it manually. This field is compulsory for DB data handling.
	Commit every	Enter the number of rows to be completed before committing batches of rows together into the DB. This option ensures transaction quality (but not rollback) and, above all, better performance at execution.
	Additional Columns	This option is not offered if you create (with or without drop) the DB table. This option allows you to call SQL functions to perform actions on columns, which are not insert, nor update or delete actions, or action that require particular preprocessing.

		Name: Type in the name of the schema column to be altered or inserted as new column
		SQL expression: Type in the SQL statement to be executed in order to alter or insert the relevant column data.
		Position: Select **Before**, **Replace** or **After** following the action to be performed on the reference column.
		Reference column: Type in a column of reference that the **tDBOutput** can use to place or replace the new or altered column.
	*Use field option*s	Select this check box to customize a request, especially when there is double action on data.
	Enable debug mode	Select this check box to display each step during processing entries in a database.
	*tState*Catcher Statistics	Select this check box to collect log data at the component level.
Usage	This component offers the flexibility benefit of the DB query and covers all possibilities of SQL queries.	

Related scenarios

For related topics, see:

- **tDBOutput** *Scenario: Displaying DB output on page 168*
- **tMySQLOutput** *Scenario 1: Adding a new column and altering data in a DB table on page 283*.

tFirebirdRow

tFirebirdRow properties

Component family	Databases/FireBird	
Function	**tFirebirdRow** is the specific component for this database query. It executes the SQL query stated onto the specified database. The row suffix means the component implements a flow in the job design although it doesn't provide output.	
Purpose	Depending on the nature of the query and the database, **tFirebirdRow** acts on the actual DB structure or on the data (although without handling data). The SQLBuilder tool helps you write easily your SQL statements.	
Basic settings	*Property type*	Either Built-in or Repository.
		Built-in: No property data stored centrally.
		Repository: Select the repository file where Properties are stored. The fields that come after are pre-filled in using the fetched data.
	Use an existing connection	Select this check box and click the relevant **tFirebirdConnection** component on the Component list to reuse the connection details you already defined.
	Host	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database
	Username and *Password*	DB user authentication data.
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either built-in or remotely stored in the Repository.
		Built-in: The schema is created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	Query type	Either Built-in or Repository.
		Built-in: Fill in manually the query statement or build it graphically using SQLBuilder
		Repository: Select the relevant query stored in the Repository. The Query field gets accordingly filled in.

	Query	Enter your DB query paying particularly attention to properly sequence the fields in order to match the schema definition.
	Die on error	Clear this check box to skip the row on error and complete the process for error-free rows.
Advanced settings	*Propagate QUERY's recordset*	Select this check box to insert the result of the query into a COLUMN of the current flow.
	Encoding Type	Select the encoding from the list or select **Custom** and define it manually. This field is compulsory for DB data handling.
	Commit every	Number of rows to be completed before committing batches of rows together into the DB. This option ensures transaction quality (but not rollback) and above all better performance on executions.
	*tState*Catcher Statistics	Select this check box to collect log data at the component level.
Usage	This component offers the flexibility benefit of the DB query and covers all possibilities of SQL queries.	

Related scenarios

For related topics, see:

- **tDBSQLRow** *Scenario: Resetting a DB auto-increment on page 172*
- **tMySQLRow** *Scenario: Removing and regenerating a MySQL table index on page 301*.

tHSQLDbInput

tHSQLDbInput properties

Component family	Databases/HSQLDb	
Function	**tHSQLDbInput** reads a database and extracts fields based on a query.	
Purpose	**tHSQLDbInput** executes a DB query with a strictly defined order which must correspond to the schema definition. Then it passes on the field list to the next component via a Main row link.	
Basic settings	*Property type*	Either Built-in or Repository
		Built-in: No property data stored centrally.
		Repository: Select the repository file where Properties are stored. The fields that come after are pre-filled in using the fetched data.
		Click this icon to open a database connection wizard and store the database connection parameters you set in the component **Basic settings** view. For more information about setting up and storing database connection parameters, see *Setting up a DB connection* of **Talend Open Studio** User Guide.
	Running Mode	Select on the list the Server Mode corresponding to your DB setup.
	Use TLS/SSL sockets	select this check box to enable the secured mode if required.
	Host	Database server IP address
	Port	Listening port number of DB server.
	Database Alias	Alias name of the database
	Username and *Password*	DB user authentication data.
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either built-in or remotely stored in the Repository.
		Built-in: The schema is created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	Query type and *Query*	Enter your DB query paying particularly attention to properly sequence the fields in order to match the schema definition.

Advanced settings	*Trim all the String/Char columns*	Select this check box to remove leading and trailing whitespace from all the String/Char columns.
	Trim column	Remove leading and trailing whitespace from defined columns.
	*tState*Catcher Statistics	Select this check box to collect log data at the component level.
Usage	This component covers all possibilities of SQL queries onto an HSQLDb database.	

Related scenarios

For related topics, see **tDBInput** scenarios:

- *Scenario 1: Displaying selected data from DB table on page 163*
- *Scenario 2: Using StoreSQLQuery variable on page 164*

See also the related topic in **tContextLoad** *Scenario: Dynamic context use in MySQL DB insert on page 652*.

tHSQLDbOutput

tHSQLDbOutput properties

Component family	Databases/HSQLDb	
Function	**tHSQLDbOutput** writes, updates, makes changes or suppresses entries in a database.	
Purpose	**tHSQLDbOutput** executes the action defined on the table and/or on the data contained in the table, based on the flow incoming from the preceding component in the Job.	
Basic settings	*Property type*	Either Built-in or Repository.
		Built-in: No property data stored centrally.
		Repository: Select the repository file where Properties are stored. The fields that come after are pre-filled in using the fetched data.
		Click this icon to open a database connection wizard and store the database connection parameters you set in the component **Basic settings** view. For more information about setting up and storing database connection parameters, see *Setting up a DB connection* of **Talend Open Studio** User Guide.
	Running Mode	Select on the list the Server Mode corresponding to your DB setup.
	Use TLS/SSL sockets	Select this check box to enable the secured mode if required.
	Host	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database
	Username and *Password*	DB user authentication data.
	Table	Name of the table to be written. Note that only one table can be written at a time
	Action on table	On the table defined, you can perform one of the following operations: **None**: No operation is carried out. **Drop and create a table**: The table is removed and created again. **Create a table**: The table does not exist and gets created. **Create a table if not exists**: The table is created if it does not exist. **Drop a table if exists and create**: The table is removed if it already exists and created again. **Clear a table**: The table content is deleted.

	Action on data	On the data of the table defined, you can perform: **Insert**: Add new entries to the table. If duplicates are found, Job stops. **Update**: Make changes to existing entries **Insert or update**: Add entries or update existing ones. **Update or insert**: Update existing entries or create it if non existing **Delete**: Remove entries corresponding to the input flow.
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either built-in or remotely stored in the Repository.
		Built-in: The schema is created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	Die on error	Clear this check box to skip the row on error and complete the process for error-free rows.
Advanced settings	*Encoding Type*	Select the encoding type from the list or select **Custom** and define it manually. This field is compulsory for DB data handling.
	Commit every	Enter the number of rows to be completed before committing batches of rows together into the DB. This option ensures transaction quality (but not rollback) and, above all, better performance at execution.
	Additional Columns	This option is not offered if you create (with or without drop) the DB table. This option allows you to call SQL functions to perform actions on columns, which are not insert, nor update or delete actions, or action that require particular preprocessing.
		Name: Type in the name of the schema column to be altered or inserted as new column
		SQL expression: Type in the SQL statement to be executed in order to alter or insert the relevant column data.
		Position: Select **Before**, **Replace** or **After** following the action to be performed on the reference column.
		Reference column: Type in a column of reference that the **tDBOutput** can use to place or replace the new or altered column.
	Use field options	Select this check box to customize a request, especially when there is double action on data.
	Enable debug mode	Select this check box to display each step during processing entries in a database.

	*tState*Catcher Statistics	Select this check box to collect log data at the component level.
Usage	This component offers the flexibility benefit of the DB query and covers all possibilities of SQL queries.	

Related scenarios

For related topics, see

- **tDBOutput** *Scenario: Displaying DB output on page 168*
- **tMySQLOutput** *Scenario 1: Adding a new column and altering data in a DB table on page 283*.

tHSQLDbRow

tHSQLDbRow properties

Component family	Databases/HSQLDb	
Function	**tHSQLDbRow** is the specific component for this database query. It executes the SQL query stated onto the specified database. The row suffix means the component implements a flow in the job design although it doesn't provide output.	
Purpose	Depending on the nature of the query and the database, **tHSQLDbRow** acts on the actual DB structure or on the data (although without handling data). The SQLBuilder tool helps you write easily your SQL statements.	
Basic settings	*Property type*	Either Built-in or Repository.
		Built-in: No property data stored centrally.
		Repository: Select the repository file where Properties are stored. The fields that come after are pre-filled in using the fetched data.
	Running Mode	Select on the list the Server Mode corresponding to your DB setup.
	Use TLS/SSL sockets	Select this check box to enable the secured mode if required.
	Host	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database
	Username and *Password*	DB user authentication data.
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either built-in or remotely stored in the Repository.
		Built-in: The schema is created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	Query type	Either Built-in or Repository.
		Built-in: Fill in manually the query statement or build it graphically using SQLBuilder
		Repository: Select the relevant query stored in the Repository. The Query field gets accordingly filled in.

	Query	Enter your DB query paying particularly attention to properly sequence the fields in order to match the schema definition.
	Die on error	Clear this check box to skip the row on error and complete the process for error-free rows.
Advanced settings	*Propagate QUERY's recordset*	Select this check box to insert the result of the query into a COLUMN of the current flow.
	Encoding Type	Select the encoding from the list or select **Custom** and define it manually. This field is compulsory for DB data handling.
	Commit every	Number of rows to be completed before committing batches of rows together into the DB. This option ensures transaction quality (but not rollback) and above all better performance on executions.
	*tState*Catcher Statistics	Select this check box to collect log data at the component level.
Usage	This component offers the flexibility benefit of the DB query and covers all possibilities of SQL queries.	

Related scenarios

For related topics, see:

- **tDBSQLRow** *Scenario: Resetting a DB auto-increment on page 172*
- **tMySQLRow** *Scenario: Removing and regenerating a MySQL table index on page 301*.

tInformixInput

tInformixInput properties

Component family	Databases/Informix	
Function	**tInformixInput** reads a database and extracts fields based on a query.	
Purpose	**tInformixInput** executes a DB query with a strictly defined order which must correspond to the schema definition. Then it passes on the field list to the next component via a Main row link.	
Basic settings	*Property type*	Either Built-in or Repository
		Built-in: No property data stored centrally.
		Repository: Select the repository file where Properties are stored. The fields that come after are pre-filled in using the fetched data.
		Click this icon to open a database connection wizard and store the database connection parameters you set in the component **Basic settings** view. For more information about setting up and storing database connection parameters, see *Setting up a DB connection* of **Talend Open Studio** User Guide.
	Host	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database
	DB server	Name of the database server
	Username and *Password*	DB user authentication data.
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either built-in or remotely stored in the Repository.
		Built-in: The schema is created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	Query type and *Query*	Enter your DB query paying particularly attention to properly sequence the fields in order to match the schema definition.
	Encoding Type	Select the encoding from the list or select Custom and define it manually. This field is compulsory for DB data handling.
Usage	This component covers all possibilities of SQL queries onto a DB2 database.	

Related scenarios

For related topics, see **tDBInput** scenarios:

- *Scenario 1: Displaying selected data from DB table on page 163*
- *Scenario 2: Using StoreSQLQuery variable on page 164*

See also the **tContextLoad** *Scenario: Dynamic context use in MySQL DB insert on page 652*.

tInformixOutput

tInformixOutput properties

Component family	Databases/Informix	
Function	**tInformixOutput** writes, updates, makes changes or suppresses entries in a database.	
Purpose	**tInformixOutput** executes the action defined on the table and/or on the data contained in the table, based on the flow incoming from the preceding component in the Job.	
Basic settings	*Property type*	Either Built-in or Repository.
		Built-in: No property data stored centrally.
		Repository: Select the repository file where Properties are stored. The fields that come after are pre-filled in using the fetched data.
		Click this icon to open a database connection wizard and store the database connection parameters you set in the component **Basic settings** view. For more information about setting up and storing database connection parameters, see *Setting up a DB connection* of **Talend Open Studio** User Guide.
	Host	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database
	DB server	Name of the database server
	Username and *Password*	DB user authentication data.
	Table	Name of the table to be written. Note that only one table can be written at a time
	Action on table	On the table defined, you can perform one of the following operations: **None**: No operation is carried out. **Drop and create a table**: The table is removed and created again. **Create a table**: The table does not exist and gets created. **Create a table if not exists**: The table is created if it does not exist. **Drop a table if exists and create**: The table is removed if it already exists and created again.. **Clear a table**: The table content is deleted.

	Action on data	On the data of the table defined, you can perform: **Insert**: Add new entries to the table. If duplicates are found, Job stops. **Update**: Make changes to existing entries **Insert or update**: Add entries or update existing ones. **Update or insert**: Update existing entries or create it if non existing **Delete**: Remove entries corresponding to the input flow. *It is necessary to specify at least one column as a primary key on which the Update and Delete operations are based. You can do that by clicking Edit Schema and selecting the check box(es) next to the column(s) you want to set as primary key(s). For an advanced use, click the Advanced settings view where you can simultaneously define primary keys for the Update and Delete operations. To do that: Select the Use field options check box and then in the Key in update column, select the check boxes next to the column names you want to use as a base for the Update operation. Do the same in the Key in delete column for the Delete operation.*
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either built-in or remotely stored in the Repository.
		Built-in: The schema is created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	Die on error	Clear this check box to skip the row on error and complete the process for error-free rows.
Advanced settings	*Additional JDBC parameters*	Specify additional connection properties in the existing DB connection.
	Extend Insert	Select this check box to carry out a bulk insert of a definable set of lines instead of inserting lines one by one. The gain in system performance is huge. **Number of rows per insert**:: enter the number of rows to be inserted as one block. Note that too high value decreases performance due to memory issues.
	Encoding Type	Select the encoding type from the list or select **Custom** and define it manually. This field is compulsory for DB data handling.

	Commit every	Number of rows to be completed before committing batches of rows together into the DB. This option ensures transaction quality (but not rollback) and, above all, better performance at executions.
	Additional Columns	This option is not offered if you create (with or without drop) the DB table. This option allows you to call SQL functions to perform actions on columns, which are not insert, nor update or delete actions, or action that require particular preprocessing.
		Name: Type in the name of the schema column to be altered or inserted as new column
		SQL expression: Type in the SQL statement to be executed in order to alter or insert the relevant column data.
		Position: Select **Before**, **Replace** or **After** following the action to be performed on the reference column.
		Reference column: Type in a column of reference that the **tDBOutput** can use to place or replace the new or altered column.
	Use field options	Select this check box to customize a request, especially when there is double action on data.
	Enable debug mode	Select this check box to display each step during processing entries in a database.
	tStateCatcher Statistics	Select this check box to collect log data at the component level.
Usage	This component offers the flexibility benefit of the DB query and covers all possibilities of SQL queries.	

Related scenarios

For **tInformixOutput** related topics, see

- **tDBOutput** *Scenario: Displaying DB output on page 168*
- **tMySQLOutput** *Scenario 1: Adding a new column and altering data in a DB table on page 283*.

tInformixRow

tInformixRow properties

Component family	Databases/Informix	
Function	**tInformixRow** is the specific component for this database query. It executes the SQL query stated onto the specified database. The row suffix means the component implements a flow in the job design although it doesn't provide output.	
Purpose	Depending on the nature of the query and the database, **tInformixRow** acts on the actual DB structure or on the data (although without handling data). The SQLBuilder tool helps you write easily your SQL statements.	
Basic settings	*Property type*	Either Built-in or Repository.
		Built-in: No property data stored centrally.
		Repository: Select the repository file where Properties are stored. The fields that come after are pre-filled in using the fetched data.
	Host	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database
	Username and *Password*	DB user authentication data.
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either built-in or remotely stored in the Repository.
		Built-in: The schema is created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	Query type	Either Built-in or Repository.
		Built-in: Fill in manually the query statement or build it graphically using SQLBuilder
		Repository: Select the relevant query stored in the Repository. The Query field gets accordingly filled in.
	Query	Enter your DB query paying particularly attention to properly sequence the fields in order to match the schema definition.
	Die on error	Clear this check box to skip the row on error and complete the process for error-free rows.

Advanced settings	*Additional JDBC parameters*	Specify additional connection properties in the existing DB connection
	Propagate QUERY's recordset	Select this check box to insert the result of the query into a COLUMN of the current flow.
	Encoding Type	Select the encoding from the list or select **Custom** and define it manually. This field is compulsory for DB data handling.
	Commit every	Number of rows to be completed before committing batches of rows together into the DB. This option ensures transaction quality (but not rollback) and above all better performance on executions.
	*tState*Catcher Statistics	Select this check box to collect log data at the component level.
Usage	This component offers the flexibility benefit of the DB query and covers all possibilities of SQL queries.	

Related scenarios

For related topics, see:

- **tDBSQLRow** *Scenario: Resetting a DB auto-increment on page 172*
- **tMySQLRow** *Scenario: Removing and regenerating a MySQL table index on page 301*.

tIngresCommit

tIngresCommit Properties

This component is closely related to **tIngresConnection** and **tIngresRollback**. It usually does not make much sense to use these components independently in a transaction.

Component family	Databases/Ingres	
Function	Validates the data processed through the Job into the connected DB	
Purpose	Using a unique connection, this component commits in one go a global transaction instead of doing that on every row or every batch and thus provides gain in performance.	
Basic settings	*Component list*	Select the **tIngresConnection** component in the list if more than one connection are planned for the current Job.
	Close Connection	Clear this check box to continue to use the selected connection once the component has performed its task.
Usage	This component is to be used along with Ingres components, especially with **tIngresConnection** and **tIngresRollback**.	
Limitation	n/a	

Related scenario

For **tIngresCommit** related scenario, see *Scenario: Inserting data in mother/daughter tables on page 269*.

tIngresConnection

tIngresConnection Properties

This component is closely related to **tIngresCommit** and **tIngresRollback**. It usually does not make much sense to use one of these without using a **tIngresConnection** component to open a connection for the current transaction.

Component family	Databases/Ingres	
Function	Opens a connection to the database for a current transaction.	
Purpose	Allows to commit a whole job data in one go to the output database as one transaction when validated.	
Basic settings	*Property type*	Either **Built-in** or **Repository**.
		Built-in: No property data stored centrally.
		Repository: Select the repository file where Properties are stored. The fields that come after are pre-filled in using the fetched data.
	Server	Database server IP address.
	Port	Listening port number of DB server.
	Database	Name of the database
	Username and *Password*	DB user authentication data.
	Use or register a shared DB Connection	Select this check box to share your connection or fetch a connection shared by a parent or child Job. **Shared DB Connection Name:** set or type in the shared connection name.
Usage	This component is to be used along with Ingres components, especially with **tIngresCommit** and **tIngresRollback**.	
Limitation	n/a	

Related scenarios

For **tIngresConnection** related scenario, see *Scenario: Inserting data in mother/daughter tables on page 269*.

tIngresInput

tIngresInput properties

Component family	Databases/Ingres	
Function	**tIngresInput** reads a database and extracts fields based on a query.	
Purpose	**tIngresInput** executes a DB query with a strictly defined order which must correspond to the schema definition. Then it passes on the field list to the next component via a Main row link.	
Basic settings	*Property type*	Either Built-in or Repository
		Built-in: No property data stored centrally.
		Repository: Select the repository file where Properties are stored. The fields that come after are pre-filled in using the fetched data.
		Click this icon to open a database connection wizard and store the database connection parameters you set in the component **Basic settings** view. For more information about setting up and storing database connection parameters, see *Setting up a DB connection* of **Talend Open Studio** User Guide.
	Server	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database
	Username and *Password*	DB user authentication data.
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either built-in or remotely stored in the Repository.
		Built-in: The schema is created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	Query type and *Query*	Enter your DB query paying particularly attention to properly sequence the fields in order to match the schema definition.
Advanced settings	*Trim all the String/Char columns*	Select this check box to remove leading and trailing whitespace from all the String/Char columns.
	Trim column	Remove leading and trailing whitespace from defined columns.

	*tState*Catcher Statistics	Select this check box to collect log data at the component level.
Usage	This component covers all possibilities of SQL queries onto an Ingres database.	

Related scenarios

For related topics, see **tDBInput** scenarios:

- *Scenario 1: Displaying selected data from DB table on page 163*
- *Scenario 2: Using StoreSQLQuery variable on page 164*

See also, the **tContextLoad** *Scenario: Dynamic context use in MySQL DB insert on page 652*.

tIngresOutput

tIngresOutput properties

Component family	Databases/Ingres	
Function	**tIngresOutput** writes, updates, makes changes or suppresses entries in a database.	
Purpose	**tIngresOutput** executes the action defined on the table and/or on the data contained in the table, based on the flow incoming from the preceding component in the Job.	
Basic settings	*Property type*	Either Built-in or Repository.
		Built-in: No property data stored centrally.
		Repository: Select the repository file where Properties are stored. The fields that come after are pre-filled in using the fetched data.
		Click this icon to open a database connection wizard and store the database connection parameters you set in the component **Basic settings** view. For more information about setting up and storing database connection parameters, see *Setting up a DB connection* of **Talend Open Studio** User Guide.
	Host	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database
	Username and *Password*	DB user authentication data.
	Table	Name of the table to be written. Note that only one table can be written at a time
	Action on table	On the table defined, you can perform one of the following operations: **None**: No operation is carried out. **Drop and create a table**: The table is removed and created again. **Create a table**: The table does not exist and gets created. **Create a table if not exists**: The table is created if it does not exist. **Drop a table if exists and create**: The table is removed if it already exists and created again. **Clear a table**: The table content is deleted.

	Action on data	On the data of the table defined, you can perform: **Insert**: Add new entries to the table. If duplicates are found, Job stops. **Update**: Make changes to existing entries **Insert or update**: Add entries or update existing ones. **Update or insert**: Update existing entries or create it if non existing **Delete**: Remove entries corresponding to the input flow. *It is necessary to specify at least one column as a primary key on which the Update and Delete operations are based. You can do that by clicking Edit Schema and selecting the check box(es) next to the column(s) you want to set as primary key(s). For an advanced use, click the Advanced settings view where you can simultaneously define primary keys for the Update and Delete operations. To do that: Select the Use field options check box and then in the Key in update column, select the check boxes next to the column names you want to use as a base for the Update operation. Do the same in the Key in delete column for the Delete operation.*
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either built-in or remotely stored in the Repository.
		Built-in: The schema is created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	Die on error	Clear this check box to skip the row on error and complete the process for error-free rows.
Advanced settings	*Encoding Type*	Select the encoding type from the list or select **Custom** and define it manually. This field is compulsory for DB data handling.
	Commit every	Enter the number of rows to be completed before committing batches of rows together into the DB. This option ensures transaction quality (but not rollback) and, above all, better performance at execution.
	Additional Columns	This option is not offered if you create (with or without drop) the DB table. This option allows you to call SQL functions to perform actions on columns, which are not insert, nor update or delete actions, or action that require particular preprocessing.

		Name: Type in the name of the schema column to be altered or inserted as new column
		SQL expression: Type in the SQL statement to be executed in order to alter or insert the relevant column data.
		Position: Select **Before**, **Replace** or **After** following the action to be performed on the reference column.
		Reference column: Type in a column of reference that the **tDBOutput** can use to place or replace the new or altered column.
	Use field options	Select this check box to customize a request, especially when there is double action on data.
	Enable debug mode	Select this check box to display each step during processing entries in a database.
	*tState*Catcher Statistics	Select this check box to collect log data at the component level.
Usage	This component offers the flexibility benefit of the DB query and covers all possibilities of SQL queries.	

Related scenarios

For related topics, see:

- **tDBOutput** *Scenario: Displaying DB output on page 168*
- **tMySQLOutput** *Scenario 1: Adding a new column and altering data in a DB table on page 283*.

tIngresRollback

tIngresRollback properties

This component is closely related to **tIngresCommit** and **tIngresConnection**. It usually does not make much sense to use these components independently in a transaction.

Component family	Databases/Ingres	
Function	**tIngresRollback** cancels the transaction committed in the connected DB.	
Purpose	Avoids to commit part of a transaction involuntarily.	
Basic settings	*Component list*	Select the **tIngresConnection** component in the list if more than one connection are planned for the current Job.
	Close Connection	Clear this check box to continue to use the selected connection once the component has performed its task.
Usage	This component is to be used along with Ingres components, especially with **tIngresConnection** and **tIngresCommit**.	
Limitation	n/a	

Related scenarios

For **tIngresRollback** related scenario, see *Scenario: Rollback from inserting data in mother/daughter tables on page 299*.

tIngresRow

tIngresRow properties

Component family	Databases/Ingres	
Function	**tIngresRow** is the specific component for this database query. It executes the SQL query stated onto the specified database. The row suffix means the component implements a flow in the job design although it doesn't provide output.	
Purpose	Depending on the nature of the query and the database, **tIngresRow** acts on the actual DB structure or on the data (although without handling data). The SQLBuilder tool helps you write easily your SQL statements.	
Basic settings	*Property type*	Either Built-in or Repository.
		Built-in: No property data stored centrally.
		Repository: Select the repository file where Properties are stored. The fields that come after are pre-filled in using the fetched data.
	Host	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database
	Username and *Password*	DB user authentication data.
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either built-in or remotely stored in the Repository.
		Built-in: The schema is created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	Query type	Either Built-in or Repository.
		Built-in: Fill in manually the query statement or build it graphically using SQLBuilder
		Repository: Select the relevant query stored in the Repository. The Query field gets accordingly filled in.
	Query	Enter your DB query paying particularly attention to properly sequence the fields in order to match the schema definition.
	Die on error	Clear this check box to skip the row on error and complete the process for error-free rows.

Advanced settings	*Additional JDBC parameters*	Specify additional connection properties in the existing DB connection
	Propagate QUERY's recordset	Select this check box to insert the result of the query into a COLUMN of the current flow.
	Encoding Type	Select the encoding from the list or select **Custom** and define it manually. This field is compulsory for DB data handling.
	Commit every	Number of rows to be completed before committing batches of rows together into the DB. This option ensures transaction quality (but not rollback) and above all better performance on executions.
	*tState*Catcher Statistics	Select this check box to collect log data at the component level.
Usage	This component offers the flexibility benefit of the DB query and covers all possibilities of SQL queries.	

Related scenarios

For related topics, see:

- **tDBSQLRow** *Scenario: Resetting a DB auto-increment on page 172*
- **tMySQLRow** *Scenario: Removing and regenerating a MySQL table index on page 301*.

tIngresSCD

tIngresSCD belongs to two component families: Business Intelligence and Databases. For more information on it, see *tIngresSCD on page 4*.

tInterbaseInput

tInterbaseInput properties

Component family	Databases/Interbase	
Function	**tInterbaseInput** reads a database and extracts fields based on a query.	
Purpose	**tInterbaseInput** executes a DB query with a strictly defined order which must correspond to the schema definition. Then it passes on the field list to the next component via a Main row link.	
Basic settings	*Property type*	Either Built-in or Repository
		Built-in: No property data stored centrally.
		Repository: Select the repository file where Properties are stored. The fields that come after are pre-filled in using the fetched data.
		Click this icon to open a database connection wizard and store the database connection parameters you set in the component **Basic settings** view. For more information about setting up and storing database connection parameters, see *Setting up a DB connection* of **Talend Open Studio** User Guide.
	Host	Database server IP address
	Database	Name of the database
	Username and *Password*	DB user authentication data.
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either built-in or remotely stored in the Repository.
		Built-in: The schema is created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	Query type and *Query*	Enter your DB query paying particularly attention to properly sequence the fields in order to match the schema definition.
Advanced settings	*Encoding Type*	Select the encoding from the list or select Custom and define it manually. This field is compulsory for DB data handling.
	Trim all the String/Char columns	Select this check box to remove leading and trailing whitespace from all the String/Char columns.
	Trim column	Remove leading and trailing whitespace from defined columns.

	*tState*Catcher Statistics	Select this check box to collect log data at the component level.
Usage	This component covers all possibilities of SQL queries onto an Interbase database.	

Related scenarios

For related topics, see **tDBInput** scenarios:

- *Scenario 1: Displaying selected data from DB table on page 163*
- *Scenario 2: Using StoreSQLQuery variable on page 164*

See also the related topic in **tContextLoad** *Scenario: Dynamic context use in MySQL DB insert on page 652*.

tInterbaseOutput

tInterbaseOutput properties

Component family	Databases/Interbase	
Function	**tInterbaseOutput** writes, updates, makes changes or suppresses entries in a database.	
Purpose	**tInterbaseOutput** executes the action defined on the table and/or on the data contained in the table, based on the flow incoming from the preceding component in the Job.	
Basic settings	*Property type*	Either Built-in or Repository.
		Built-in: No property data stored centrally.
		Repository: Select the repository file where Properties are stored. The fields that come after are pre-filled in using the fetched data.
		Click this icon to open a database connection wizard and store the database connection parameters you set in the component **Basic settings** view. For more information about setting up and storing database connection parameters, see *Setting up a DB connection* of **Talend Open Studio** User Guide.
	Host	Database server IP address
	Database	Name of the database
	Username and *Password*	DB user authentication data.
	Table	Name of the table to be written. Note that only one table can be written at a time
	Action on table	On the table defined, you can perform one of the following operations: **None**: No operation is carried out. **Drop and create a table**: The table is removed and created again. **Create a table**: The table does not exist and gets created. **Create a table if not exists**: The table is created if it does not exist. **Drop table if exists and create**: The table is removed if it already exists and created again.. **Clear a table**: The table content is deleted.

	Action on data	On the data of the table defined, you can perform: **Insert**: Add new entries to the table. If duplicates are found, Job stops. **Update**: Make changes to existing entries **Insert or update**: Add entries or update existing ones. **Update or insert**: Update existing entries or create it if non existing **Delete**: Remove entries corresponding to the input flow. *It is necessary to specify at least one column as a primary key on which the Update and Delete operations are based. You can do that by clicking Edit Schema and selecting the check box(es) next to the column(s) you want to set as primary key(s). For an advanced use, click the Advanced settings view where you can simultaneously define primary keys for the Update and Delete operations. To do that: Select the Use field options check box and then in the Key in update column, select the check boxes next to the column names you want to use as a base for the Update operation. Do the same in the Key in delete column for the Delete operation.*
	Clear data in table	Wipes out data from the selected table before action.
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either built-in or remotely stored in the Repository.
		Built-in: The schema is created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	Die on error	Clear this check box to skip the row on error and complete the process for error-free rows.
Advanced settings	*Encoding Type*	Select the encoding type from the list or select **Custom** and define it manually. This field is compulsory for DB data handling.
	Commit every	Enter the number of rows to be completed before committing batches of rows together into the DB. This option ensures transaction quality (but not rollback) and, above all, better performance at execution.
	Additional Columns	This option is not offered if you create (with or without drop) the DB table. This option allows you to call SQL functions to perform actions on columns, which are not insert, nor update or delete actions, or action that require particular preprocessing.

		Name: Type in the name of the schema column to be altered or inserted as new column
		SQL expression: Type in the SQL statement to be executed in order to alter or insert the relevant column data.
		Position: Select **Before**, **Replace** or **After** following the action to be performed on the reference column.
		Reference column: Type in a column of reference that the **tDBOutput** can use to place or replace the new or altered column.
	*Use field option*s	Select this check box to customize a request, especially when there is double action on data.
	Enable debug mode	Select this check box to display each step during processing entries in a database.
	*tState*Catcher Statistics	Select this check box to collect log data at the component level.
Usage	This component offers the flexibility benefit of the DB query and covers all possibilities of SQL queries.	

Related scenarios

For related topics, see

- **tDBOutput** *Scenario: Displaying DB output on page 168*
- **tMySQLOutput** *Scenario 1: Adding a new column and altering data in a DB table on page 283*.

tInterbaseRow

tInterbaseRow properties

Component family	Databases/Interbase	
Function	**tInterbaseRow** is the specific component for this database query. It executes the SQL query stated onto the specified database. The row suffix means the component implements a flow in the job design although it doesn't provide output.	
Purpose	Depending on the nature of the query and the database, **tInterbaseRow** acts on the actual DB structure or on the data (although without handling data). The SQLBuilder tool helps you write easily your SQL statements.	
Basic settings	*Property type*	Either Built-in or Repository.
		Built-in: No property data stored centrally.
		Repository: Select the repository file where Properties are stored. The fields that come after are pre-filled in using the fetched data.
	Use an existing connection	Select this check box and click the relevant **tInterbaseConnection** component on the Component list to reuse the connection details you already defined.
	Host	Database server IP address
	Database	Name of the database
	Username and *Password*	DB user authentication data.
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either built-in or remotely stored in the Repository.
		Built-in: The schema is created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	Query type	Either Built-in or Repository.
		Built-in: Fill in manually the query statement or build it graphically using SQLBuilder
		Repository: Select the relevant query stored in the Repository. The Query field gets accordingly filled in.
	Query	Enter your DB query paying particularly attention to properly sequence the fields in order to match the schema definition.

	Die on error	Clear this check box to skip the row on error and complete the process for error-free rows.
Advanced settings	*Propagate QUERY's recordset*	Select this check box to insert the result of the query into a COLUMN of the current flow.
	Encoding Type	Select the encoding from the list or select **Custom** and define it manually. This field is compulsory for DB data handling.
	Commit every	Number of rows to be completed before committing batches of rows together into the DB. This option ensures transaction quality (but not rollback) and above all better performance on executions.
	*tState*Catcher Statistics	Select this check box to collect log data at the component level.
Usage	This component offers the flexibility benefit of the DB query and covers all possibilities of SQL queries.	

Related scenarios

For related topics, see:

- **tDBSQLRow** *Scenario: Resetting a DB auto-increment on page 172*
- **tMySQLRow** *Scenario: Removing and regenerating a MySQL table index on page 301*.

tJavaDBInput

tJavaDBInput properties

Component family	Databases/JavaDB	
Function	**tJavaDBInput** reads a database and extracts fields based on a query.	
Purpose	**tJavaDBInput** executes a DB query with a strictly defined order which must correspond to the schema definition. Then it passes on the field list to the next component via a Main row link.	
Basic settings	*Property type*	Either Built-in or Repository
		Built-in: No property data stored centrally.
		Repository: Select the repository file where Properties are stored. The fields that come after are pre-filled in using the fetched data.
		Click this icon to open a database connection wizard and store the database connection parameters you set in the component **Basic settings** view. For more information about setting up and storing database connection parameters, see *Setting up a DB connection* of **Talend Open Studio** User Guide.
	Framework	Select your Java database framework on the list
	Database	Name of the database
	DB root path	Browse to your database root.
	Username and *Password*	DB user authentication data.
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either built-in or remotely stored in the Repository.
		Built-in: The schema is created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	Query type and Query	Enter your DB query paying particularly attention to properly sequence the fields in order to match the schema definition.
Advanced settings	*Encoding Type*	Select the encoding from the list or select Custom and define it manually. This field is compulsory for DB data handling.
	Trim all the String/Char columns	Select this check box to remove leading and trailing whitespace from all the String/Char columns.

	Trim column	Remove leading and trailing whitespace from defined columns.
	*tState*Catcher Statistics	Select this check box to collect log data at the component level.
Usage	This component covers all possibilities of SQL queries onto a database.	

Related scenarios

For related topics, see **tDBInput** scenarios:

- *Scenario 1: Displaying selected data from DB table on page 163*
- *Scenario 2: Using StoreSQLQuery variable on page 164*

See also the related topic in **tContextLoad** *Scenario: Dynamic context use in MySQL DB insert on page 652*.

tJavaDBOutput

tJavaDBOutput properties

Component family	Databases/JavaDB	
Function	**tJavaDBOutput** writes, updates, makes changes or suppresses entries in a database.	
Purpose	**tJavaDBOutput** executes the action defined on the table and/or on the data contained in the table, based on the flow incoming from the preceding component in the Job.	
Basic settings	*Property type*	Either Built-in or Repository.
		Built-in: No property data stored centrally.
		Repository: Select the repository file where Properties are stored. The fields that come after are pre-filled in using the fetched data.
		Click this icon to open a database connection wizard and store the database connection parameters you set in the component **Basic settings** view. For more information about setting up and storing database connection parameters, see *Setting up a DB connection* of **Talend Open Studio** User Guide.
	Framework	Select your Java database framework on the list
	Database	Name of the database
	DB root path	Browse to your database root.
	Username and *Password*	DB user authentication data.
	Table	Name of the table to be written. Note that only one table can be written at a time
	Action on table	On the table defined, you can perform one of the following operations: **None**: No operation is carried out. **Drop and create a table**: The table is removed and created again. **Create a table**: The table does not exist and gets created. **Create a table if not exists**: The table is created if it does not exist. **Drop table if exists and create**: The table is removed if it already exists and created again. **Clear a table**: The table content is deleted.

	Action on data	On the data of the table defined, you can perform: **Insert**: Add new entries to the table. If duplicates are found, Job stops. **Update**: Make changes to existing entries **Insert or update**: Add entries or update existing ones. **Update or insert**: Update existing entries or create it if non existing **Delete**: Remove entries corresponding to the input flow. *It is necessary to specify at least one column as a primary key on which the Update and Delete operations are based. You can do that by clicking Edit Schema and selecting the check box(es) next to the column(s) you want to set as primary key(s). For an advanced use, click the Advanced settings view where you can simultaneously define primary keys for the Update and Delete operations. To do that: Select the Use field options check box and then in the Key in update column, select the check boxes next to the column names you want to use as a base for the Update operation. Do the same in the Key in delete column for the Delete operation.*
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either built-in or remotely stored in the Repository.
		Built-in: The schema is created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	Die on error	Clear this check box to skip the row on error and complete the process for error-free rows.
Advanced settings	*Encoding Type*	Select the encoding type from the list or select **Custom** and define it manually. This field is compulsory for DB data handling.
	Commit every	Enter the number of rows to be completed before committing batches of rows together into the DB. This option ensures transaction quality (but not rollback) and, above all, better performance at execution.
	Additional Columns	This option is not offered if you create (with or without drop) the DB table. This option allows you to call SQL functions to perform actions on columns, which are not insert, nor update or delete actions, or action that require particular preprocessing.

		Name: Type in the name of the schema column to be altered or inserted as new column
		SQL expression: Type in the SQL statement to be executed in order to alter or insert the relevant column data.
		Position: Select **Before**, **Replace** or **After** following the action to be performed on the reference column.
		Reference column: Type in a column of reference that the **tDBOutput** can use to place or replace the new or altered column.
	*Use field option*s	Select this check box to customize a request, especially when there is double action on data.
	Enable debug mode	Select this check box to display each step during processing entries in a database.
	*tState*Catcher Statistics	Select this check box to collect log data at the component level.
Usage	This component offers the flexibility benefit of the DB query and covers all possibilities of SQL queries.	

Related scenarios

For related topics, see:

- **tDBOutput** *Scenario: Displaying DB output on page 168*
- **tMySQLOutput** *Scenario 1: Adding a new column and altering data in a DB table on page 283*.

tJavaDBRow

tJavaDBRow properties

Component family	Databases/JavaDB	
Function	**tJavaDBRow** is the specific component for this database query. It executes the SQL query stated onto the specified database. The row suffix means the component implements a flow in the job design although it doesn't provide output.	
Purpose	Depending on the nature of the query and the database, **tJavaDBRow** acts on the actual DB structure or on the data (although without handling data). The SQLBuilder tool helps you write easily your SQL statements.	
Basic settings	*Property type*	Either Built-in or Repository.
		Built-in: No property data stored centrally.
		Repository: Select the repository file where Properties are stored. The fields that come after are pre-filled in using the fetched data.
	Framework	Select your Java database framework on the list
	Database	Name of the database
	DB root path	Browse to your database root.
	Username and *Password*	DB user authentication data.
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either built-in or remotely stored in the Repository.
		Built-in: The schema is created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	Query type	Either Built-in or Repository.
		Built-in: Fill in manually the query statement or build it graphically using SQLBuilder
		Repository: Select the relevant query stored in the Repository. The Query field gets accordingly filled in.
	Query	Enter your DB query paying particularly attention to properly sequence the fields in order to match the schema definition.
	Die on error	Clear this check box to skip the row on error and complete the process for error-free rows.

Advanced settings	*Propagate QUERY's recordset*	Select this check box to insert the result of the query into a COLUMN of the current flow.
	Encoding Type	Select the encoding from the list or select **Custom** and define it manually. This field is compulsory for DB data handling.
	Commit every	Number of rows to be completed before committing batches of rows together into the DB. This option ensures transaction quality (but not rollback) and above all better performance on executions.
	*tState*Catcher Statistics	Select this check box to collect log data at the component level.
Usage	This component offers the flexibility benefit of the DB query and covers all possibilities of SQL queries.	

Related scenarios

For related topics, see:

- **tDBSQLRow** *Scenario: Resetting a DB auto-increment on page 172*
- **tMySQLRow** *Scenario: Removing and regenerating a MySQL table index on page 301*.

tJDBCColumnList

tJDBCColumnList Properties

Component family	Databases/JDBC	
Function	Iterates on all columns of a given table through a defined JDBC connection.	
Purpose	Lists all column names of a given JDBC table.	
Basic settings	*Component list*	Select the **tJDBCConnection** component in the list if more than one connection are planned for the current Job.
	Table name	Enter the name of the tabe.
Usage	This component is to be used along with JDBC components, especially with **tJDBCConnection**.	
Limitation	n/a	

Related scenario

For **tJDBCColumnList** related scenario, see *Scenario: Iterating on a DB table and listing its column names on page 264.*

tJDBCCommit

tJDBCCommit Properties

This component is closely related to **tJDBCConnection** and **tJDBCRollback**. It usually doesn't make much sense to use JDBC components independently in a transaction.

Component family	Databases/JDBC	
Function	Validates the data processed through the Job into the connected DB.	
Purpose	Using a unique connection, this component commits in one go a global transaction instead of doing that on every row or every batch and thus provides gain in performance.	
Basic settings	*Component list*	Select the **tJDBCConnection** component in the list if more than one connection are planned for the current Job.
	Close Connection	Clear this check box to continue to use the selected connection once the component has performed its task.
Usage	This component is to be used along with JDBC components, especially with the **tJDBCConnection** and **tJDBCRollback** components.	
Limitation	n/a	

Related scenario

This component is closely related to **tJDBCConnection** and **tJDBCRollback**. It usually doesn't make much sense to use JDBC components without using the **tJDBCConnection** component to open a connection for the current transaction.

For **tJDBCCommit** related scenario, see *tMysqlConnection on page 269*.

tJDBCConnection

tJDBCConnection Properties

This component is closely related to **tJDBCCommit** and **tJDBCRollback**. It usually doesn't make much sense to use one of JDBC components without using the **tJDBCConnection** component to open a connection for the current transaction.

Component family	Databases/JDBC	
Function	Opens a connection to the database for a current transaction.	
Purpose	Allows to commit a whole job data in one go to the output database as one transaction when validated.	
Basic settings		
	JDBC URL	Enter the JDBC URL to connect to the desired DB. For example, enter: *jdbc:mysql://IP address/database name* to connect to a mysql database.
	Driver JAR	Select from the drop-down list a desired available driver, or download one from a local directory through clicking the three dots [...] button.
	Driver Class	Enter the driver class related o your connection. For example, enter *com.mysql.jdbc.Driver* as a driver class to connect to a mysql database.
	Username and *Password*	Enter your DB authentication data.
	Encoding type	Select the encoding type from the list or select Custom and define it manually. This field is compulsory for DB data handling.
	Use or register a shared DB Connection	Select this check box to share your connection or fetch a connection shared by a parent or child Job. **Shared DB Connection Name:** set or type in the shared connection name.
Usage	This component is to be used along with JDBC components, especially with the **tJDBCCommit** and **tJDBCRollback** components.	
Limitation	n/a	

Related scenario

This component is closely related to **tOracleCommit** and **tOracleRollback**. It usually doesn't make much sense to use one of JDBC components without using the **tJDBCConnection** component to open a connection for the current transaction.

For **tJDBCConnection** related scenario, see *tMysqlConnection on page 269.*

tJDBCInput

tJDBCInput properties

Component family	Databases/JDBC	
Function	**tJDBC** reads any database using a JDBC API connection and extracts fields based on a query.	
Purpose	**tJDBC** executes a DB query with a strictly defined order which must correspond to the schema definition. Then it passes on the field list to the next component via a Main row link.	
Basic settings	*Property type*	Either Built-in or Repository
		Built-in: No property data stored centrally.
		Repository: Select the repository file where Properties are stored. The fields that come after are pre-filled in using the fetched data.
	Use an existing connection	Select this check box and click the relevant **tJDBCConnection** component on the Component list to reuse the connection details you already defined.
		Click this icon to open a database connection wizard and store the database connection parameters you set in the component **Basic settings** view. For more information about setting up and storing database connection parameters, see *Setting up a DB connection* of **Talend Open Studio** User Guide.
	JDBC URL	Type in the database location path
	Driver JAR	Select the driver JAR on the list or click the three button to add a new JAR to the list.
	Class Name	Type in the Class name to be pointed to in the driver.
	Username and *Password*	DB user authentication data.
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either built-in or remotely stored in the Repository.
		Built-in: The schema is created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	Table Name	Type in the name of the table

	Query type and *Query*	Enter your DB query paying particularly attention to properly sequence the fields in order to match the schema definition.
Advanced settings	*Encoding Type*	Select the encoding from the list or select Custom and define it manually. This field is compulsory for DB data handling.
	Use cursor	When selected, helps to decide the row set to work with at a time and thus optimize performance.
	Trim all the String/Char columns	Select this check box to remove leading and trailing whitespace from all the String/Char columns.
	Trim column	Remove leading and trailing whitespace from defined columns.
	tStatCatcher Statistics	Select this check box to collect log data at the component level.
Usage	This component covers all possibilities of SQL queries onto any JDBC connected database.	

Related scenarios

Related topic in **tDBInput** scenarios:

- *Scenario 1: Displaying selected data from DB table on page 163*
- *Scenario 2: Using StoreSQLQuery variable on page 164*

Related topic in **tContextLoad** *Scenario: Dynamic context use in MySQL DB insert on page 652*.

tJDBCOutput

tJDBCOutput properties

Component family	Databases/JDBC	
Function	**tJDBCOutput** writes, updates, makes changes or suppresses entries in any type of database connected to a JDBC API.	
Purpose	**tJDBCOutput** executes the action defined on the data contained in the table, based on the flow incoming from the preceding component in the Job.	
Basic settings	*Property type*	Either Built-in or Repository
		Built-in: No property data stored centrally.
		Repository: Select the repository file where Properties are stored. The fields that come after are pre-filled in using the fetched data.
	Use an existing connection	Select this check box and click the relevant **tJDBCConnection** component on the Component list to reuse the connection details you already defined.
		Click this icon to open a database connection wizard and store the database connection parameters you set in the component **Basic settings** view. For more information about setting up and storing database connection parameters, see *Setting up a DB connection* of **Talend Open Studio** User Guide.
	JDBC URL	Type in the database location path
	Driver JAR	Select the driver JAR on the list or click the three button to add a new JAR to the list.
	Class Name	Type in the Class name to be pointed to in the driver.
	Username and *Password*	DB user authentication data.
	Table	Name of the table to be written. Note that only one table can be written at a time

	Action on data	On the data of the table defined, you can perform: **Insert**: Add new entries to the table. If duplicates are found, Job stops. **Update**: Make changes to existing entries **Insert or update**: Add entries or update existing ones. **Update or insert**: Update existing entries or create it if non existing **Delete**: Remove entries corresponding to the input flow. *It is necessary to specify at least one column as a primary key on which the Update and Delete operations are based. You can do that by clicking Edit Schema and selecting the check box(es) next to the column(s) you want to set as primary key(s). For an advanced use, click the Advanced settings view where you can simultaneously define primary keys for the Update and Delete operations. To do that: Select the Use field options check box and then in the Key in update column, select the check boxes next to the column names you want to use as a base for the Update operation. Do the same in the Key in delete column for the Delete operation.*
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either built-in or remotely stored in the Repository.
		Built-in: The schema is created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	Die on error	Clear this check box to skip the row on error and complete the process for error-free rows.
Advanced settings	*Encoding Type*	Select the encoding type from the list or select **Custom** and define it manually. This field is compulsory for DB data handling.
	Commit every	Enter the number of rows to be completed before committing batches of rows together into the DB. This option ensures transaction quality (but not rollback) and, above all, better performance at execution.
	Additional Columns	This option is not offered if you create (with or without drop) the DB table. This option allows you to call SQL functions to perform actions on columns, which are not insert, nor update or delete actions, or action that require particular preprocessing.

		Name: Type in the name of the schema column to be altered or inserted as new column
		SQL expression: Type in the SQL statement to be executed in order to alter or insert the relevant column data.
		Position: Select **Before**, **Replace** or **After** following the action to be performed on the reference column.
		Reference column: Type in a column of reference that the **tDBOutput** can use to place or replace the new or altered column.
	*Use field option*s	Select this check box to customize a request, especially when there is double action on data.
	Enable debug mode	Select this check box to display each step during processing entries in a database.
	*tState*Catcher Statistics	Select this check box to collect log data at the component level.
Usage	This component offers the flexibility benefit of a connection to any type of DB and covers all possibilities of SQL queries.	

Related scenarios

For **tJDBCOutput** related topics, see:

- **tDBOutput** *Scenario: Displaying DB output on page 168*
- **tMySQLOutput** *Scenario 1: Adding a new column and altering data in a DB table on page 283*.

tJDBCRollback

tJDBCRollback properties

This component is closely related to **tJDBCCommit** and **tJDBCConnection**. It usually doesn't make much sense to use JDBC components independently in a transaction..

Component family	Databases/JDBC	
Function	Cancels the transaction committed in the connected DB.	
Purpose	Allows to avoid to commit part of a transaction involuntarily.	
Basic settings	*Component list*	Select the **tJDBCConnection** component in the list if more than one connection are planned for the current Job.
	Close Connection	Clear this check box to continue to use the selected connection once the component has performed its task.
Usage	This component is to be used along with Oracle components, especially with **tOracleConnection** and **tOracleCommit** components.	
Limitation	n/a	

Related scenario

This component is closely related to **tJDBCConnection** and **tJDBCCommit**. It usually doesn't make much sense to use JDBC components without using the **tJDBCConnection** component to open a connection for the current transaction.

For **tJDBCRollback** related scenario, see *tMysqlRollback on page 299*.

tJDBCRow

tJDBCRow properties

Component family	Databases/JDBC	
Function	**tJDBCRow** is the component for any type database using a JDBC API. It executes the SQL query stated onto the specified database. The row suffix means the component implements a flow in the job design although it doesn't provide output.	
Purpose	Depending on the nature of the query and the database, **tJDBCRow** acts on the actual DB structure or on the data (although without handling data). The SQLBuilder tool helps you write easily your SQL statements.	
Basic settings	*Use an existing connection*	Select this check box and click the relevant **tJDBCConnection** component on the Component list to reuse the connection details you already defined.
	JDBC URL	Type in the database location path
	Driver JAR	Select the driver JAR on the list or click the three button to add a new JAR to the list.
	Class Name	Type in the Class name to be pointed to in the driver.
	Username and *Password*	DB user authentication data.
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either built-in or remotely stored in the Repository.
		Built-in: The schema is created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	Query type	Either Built-in or Repository.
		Built-in: Fill in manually the query statement or build it graphically using SQLBuilder
		Repository: Select the relevant query stored in the Repository. The Query field gets accordingly filled in.
	Query	Enter your DB query paying particularly attention to properly sequence the fields in order to match the schema definition.
	Die on error	Clear this check box to skip the row on error and complete the process for error-free rows.
Advanced settings	*Propagate QUERY's recordset*	Select this check box to insert the result of the query into a COLUMN of the current flow.

	Encoding Type	Select the encoding from the list or select **Custom** and define it manually. This field is compulsory for DB data handling.
	Commit every	Number of rows to be completed before committing batches of rows together into the DB. This option ensures transaction quality (but not rollback) and above all better performance on executions.
	*tState*Catcher Statistics	Select this check box to collect log data at the component level.
Usage	This component offers the flexibility benefit of any type DB JDBC connection and covers all possibilities of SQL queries.	

Related scenarios

For related topics, see:

- **tDBSQLRow** *Scenario: Resetting a DB auto-increment on page 172*
- **tMySQLRow** *Scenario: Removing and regenerating a MySQL table index on page 301*.

tJDBCSP

tJDBCSP Properties

Component family	Databases/JDBC	
Function	**tJDBCSP** calls the specified database stored procedure.	
Purpose	**tJDBCSP** offers a convenient way to centralize multiple or complex queries in a database and call them easily.	
Basic settings	*JDBC URL*	Type in the database location path
	Driver JAR	Select the driver JAR on the list or click the three button to add a new JAR to the list.
	Class Name	Type in the Class name to be pointed to in the driver.
	Username and *Password*	DB user authentication data.
	Schema and *Edit Schema*	In SP principle, the schema is an input parameter. A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either built-in or remotely stored in the Repository.
		Built-in: The schema is created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	SP Name	Type in the exact name of the Stored Procedure
	Is Function / Return result in	Select this check box , if a value only is to be returned. Select on the list the schema column, the value to be returned is based on.
	Parameters	Click the Plus button and select the various **Schema Columns** that will be required by the procedures. Note that the SP schema can hold more columns than there are paramaters used in the procedure. Select the **Type** of parameter: **IN**: Input parameter **OUT**: Output parameter/return value **IN OUT**: Input parameters is to be returned as value, likely after modification through the procedure (function). **RECORDSET**: Input parameters is to be returned as a set of values, rather than single value. Check the *tParseRecordSet* component if you want to analyze a set of records from a database table or DB query and return single records.

Usage	This component is used as intermediary component. It can be used as start component but only input parameters are thus allowed.
Limitation	The Stored Procedures syntax should match the Database syntax.

Related scenario

For related scenarios, see:

- **tMysqlSP** *Scenario: Finding a State Label using a stored procedure on page 307*.
- **tOracleSP** *Scenario: Checking number format using a stored procedure on page 351*

Check as well the *tParseRecordSet* component if you want to analyze a set of records from a database table or DB query and return single records.

tJDBCTableList

tJDBCTableList Properties

Component family	Databases/JDBC	
Function	Iterates on a set of table names through a defined JDBC connection.	
Purpose	Lists the names of a given set of JDBC tables using a select statement based on a Where clause.	
Basic settings	*Component list*	Select the **tJDBCConnection** component in the list if more than one connection are planned for the current Job.
	Where clause for table name selection	Enter the Where clause to identify the tables to iterate on.
Usage	This component is to be used along with JDBC components, especially with **tJDBCConnection**.	
Limitation	n/a	

Related scenario

For **tJDBCTableList** related scenario, see *Scenario: Iterating on a DB table and listing its column names on page 264*.

tLDAPInput

tLDAPInput Properties

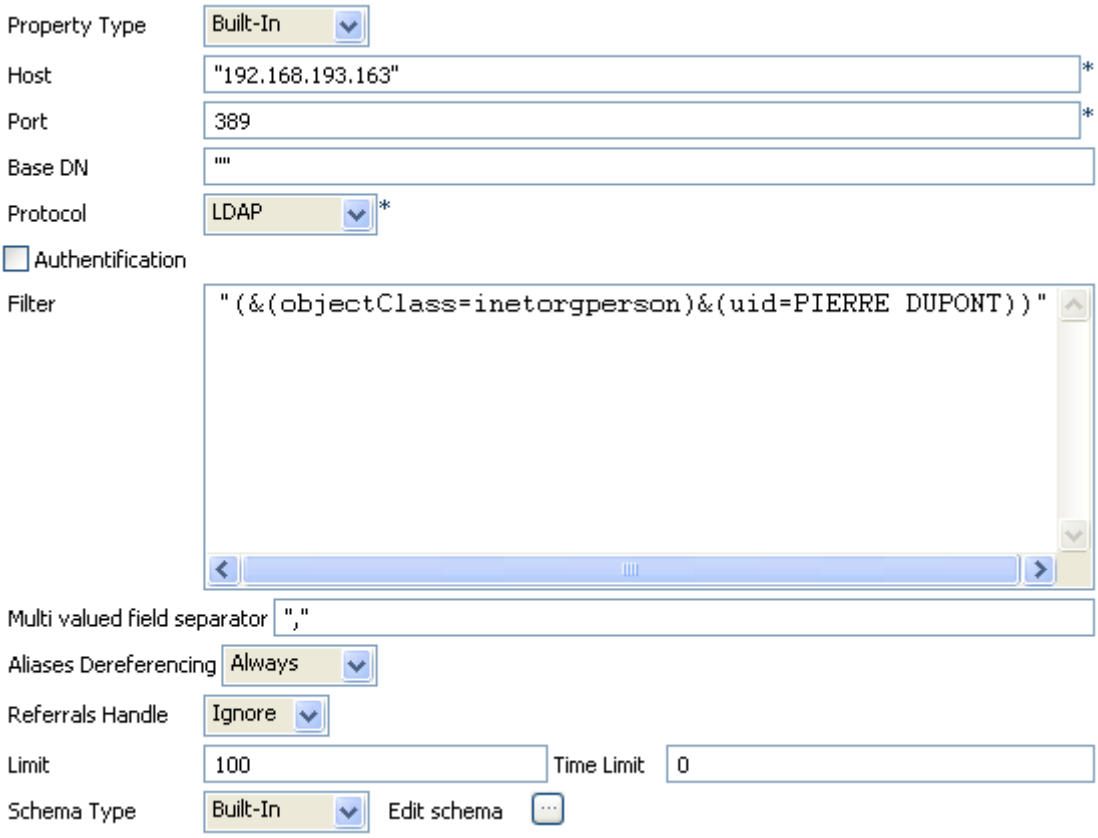
Component family	Databases/LDAP	
Function	**tLDAPInput** reads a directory and extracts data based on the defined filter.	
Purpose	**tLDAPInput** executes an LDAP query based on the given filter and corresponding to the schema definition. Then it passes on the field list to the next component via a Main row link.	
Basic settings	*Property type*	Either Built-in or Repository
		Built-in: No property data stored centrally.
		Repository: Select the Repository file where Properties are stored. When selected, the fields to follow are pre-filled in using fetched data.
		Click this icon to open a database connection wizard and store the database connection parameters you set in the component **Basic settings** view. For more information about setting up and storing database connection parameters, see *Setting up a DB connection* of **Talend Open Studio** User Guide.
	Host	LDAP Directory server IP address
	Port	Listening port number of server.
	Base DN	Path to user's authorised tree leaf.
	Protocol	Select the protocol type on the list. **LDAP** : no encryption is used **LDAPS**: secured LDAP **TLS**: certificate is used
	Authentication *User* and *Password*	Select the **Authentication** check box if LDAP login is required. Note that the login must match the LDAP syntax requirement to be valid. e.g.: "cn=Directory Manager".
	Filter	Type in the filter as expected by the LDAP directory db.
	Multi valued field separator	Type in the value separator in multi-value fields.
	Alias dereferencing	Select the option on the list. **Never** allows to improve search performance if you are sure that no aliases is to be dereferenced. By default, **Always** is to be used: **Always**: Always dereference aliases **Never**: Never dereferences aliases. **Searching**:Dereferences aliases only after name resolution. **Finding**: Dereferences aliases only during name resolution

	Referrals handle	Select the option on the list: **Ignore**: does not handle request redirections **Follow**:does handle request redirections
	Limit	Fill in a limit number of records to be read If needed.
	Time Limit	Fill in a timeout period for the directory. access
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either built-in or remotely stored in the Repository.
		Built-in: The schema is created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
Usage	This component covers all possibilities of LDAP queries. Note: Press **Ctrl + Space bar** to access the global variable list, including the **GetResultName** variable to retrieve automatically the relevant Base	

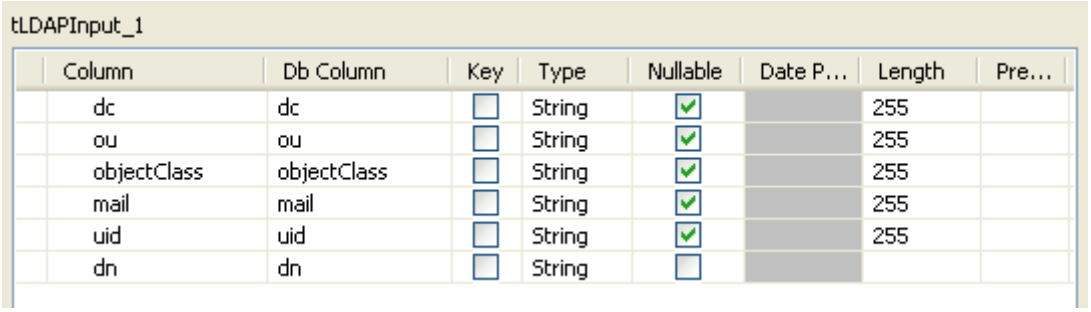
Scenario: Displaying LDAP directory's filtered content

The Job described below simply filters the LDAP directory and displays the result on the console.

- Drop the **tLDAPInput** component along with a **tLogRow** from the **Palette** to the design workspace.
- Set the **tLDAPInput** properties.
- Set the **Property type** on **Repository** if you stored the LDAP connection details in the **Metadata Manager** in the **Repository**. Then select the relevant entry on the list.
- In **Built-In** mode, fill in the **Host** and **Port** information manually. Host can be the IP address of the LDAP directory server or its DNS name.
- No particular **Base DN** is to be set.



- Then select the relevant **Protocol** on the list. In this example: a simple **LDAP** protocol is used.
- Select the **Authentication** check box and fill in the login information if required to read the directory. In this use case, no authentication is needed.
- In the **Filter** area, type in the command, the data selection is based on. In this example, the filter is: `(&(objectClass=inetorgperson)&(uid=PIERRE DUPONT))`.
- Fill in **Multi-valued field separator** with a comma as some fields may hold more than one value, separated by a comma.
- As we don't know if some aliases are used in the LDAP directory, select **Always** on the list.
- Set **Ignore** as **Referral handling**.
- Set the limit to **100** for this use case.



- Set the **Schema** as required by your LDAP directory. In this example, the schema is made of 6 columns including the `objectClass` and `uid` columns which get filtered on.
- In the **tLogRow** component, no particular setting is required.

```
Starting job testLDAPInput at 18:05 18/09/2007.
DATA|top,person,organizationalPerson,inetorgperson,a4400user|mhirt78@talend.com|PIERRE DUPONT|
Job testLDAPInput ended at 18:05 18/09/2007. [exit code=0]
```

Only one entry of the directory corresponds to the filter criteria given in the **tLDAPInput** component.

tLDAPOutput

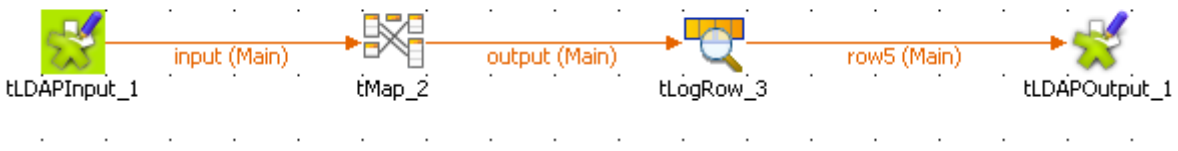
tLDAPOutput Properties

Component family	Databases/LDAP	
Function	**tLDAPOutput** writes into an LDAP directory.	
Purpose	**tLDAPOutput** executes an LDAP query based on the given filter and corresponding to the schema definition. Then it passes on the field list to the next component via a Main row link.	
Basic settings	*Property type*	Either Built-in or Repository
		Built-in: No property data stored centrally.
		Repository: Select the repository file where Properties are stored. The fields that come after are pre-filled in using the fetched data.
		Click this icon to open a database connection wizard and store the database connection parameters you set in the component **Basic settings** view. For more information about setting up and storing database connection parameters, see *Setting up a DB connection* of **Talend Open Studio** User Guide.
	Host	LDAP Directory server IP address
	Port	Listening port number of server.
	Base DN	Path to user's authorized tree leaf.
	Protocol	Select the protocol type on the list. **LDAP** : no encryption is used **LDAPS**: secured LDAP **TLS**: certificate is used
	User and *Password*	Fill in the **User** and **Password** as required by the directory Note that the login must match the LDAP syntax requirement to be valid. e.g.: "cn=Directory Manager".
	Multi valued field separator	Character, string or regular expression to separate data in a multi-value field.
	Alias dereferencing	Select the option on the list. **Never** allows to improve search performance if you are sure that no aliases is to be dereferenced. By default, **Always** is to be used: **Always**: Always dereference aliases **Never**: Never dereferences aliases. **Searching**:Dereferences aliases only after name resolution. **Finding**: Dereferences aliases only during name resolution
	Referrals handle	Select the option on the list: **Ignore**: does not handle request redirections **Follow**:does handle request redirections

	Insert mode	Select the editing mode on the list: **Add**: add a value in a multi-value attribute, **Insert**: insert new data, **Updata**: updates the existing data, **Delete**: remove the selected data from the directory, **Insert or Update**: insert new data or update existing ones.
	DN Column Name	Select in the list the type of the LDAP input entity used.
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either built-in or remotely stored in the Repository.
		Built-in: The schema is created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	Die on error	Clear this check box to skip errors and complete the process.
Advanced settings	*tStatCatcher Statistics*	Select this check box to gather the job processing metadata at a job level as well as at each component level.
Usage	This component covers all possibilities of LDAP queries. Note: Press **Ctrl + Space bar** to access the global variable list, including the **GetResultName** variable to retrieve automatically the relevant DN Base	

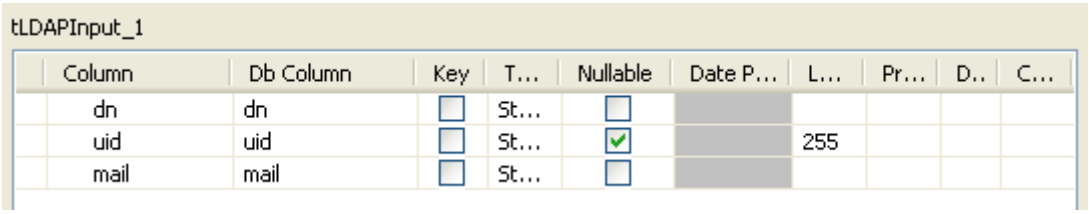
Scenario: Editing data in an LDAP directory

The following scenario describes a Job that reads an LDAP directory, updates the email of a selected entry and displays the output before writing the LDAP directory. To keep it simple, no alias dereferencing nor referral handling is performed. This scenario is based on **LDAPInput**'s *Scenario: Displaying LDAP directory's filtered content on page 235*. The result returned was a single entry, related to an organisational person, whom email is to be updated.

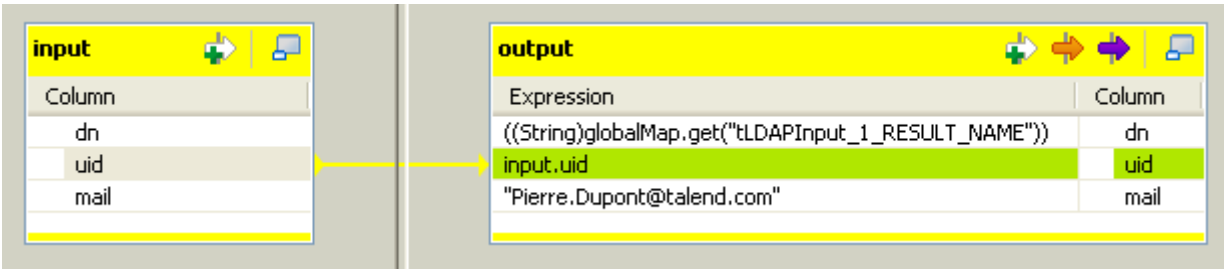


- Drop the **tLDAPInput**, **tLDAPOutput**, **tMap** and **tLogRow** components from the **Palette** to the design workspace.
- Connect the input component to the **tMap** then to the **tLogRow** and to the output component.
- In the **tLDAPInput** Component view, set the connection details to the LDAP directory server as well as the filter as described in *Scenario: Displaying LDAP directory's filtered content on page 235*.

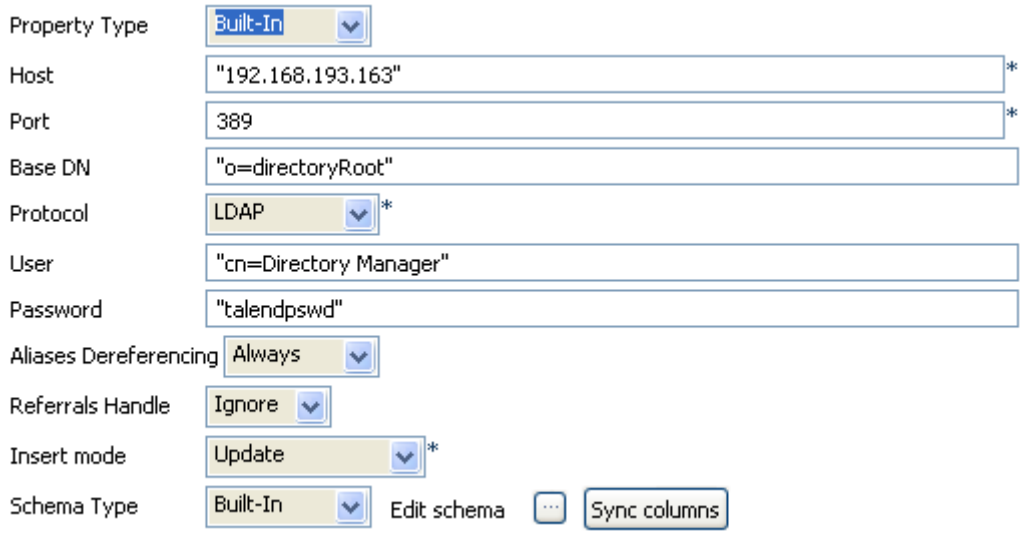
- Change the schema to make it simpler, by removing the unused fields: *dc*, *ou*, *objectclass*.



- Then open the mapper to set the edit to be carried out.
- Drag & drop the uid column from the input table to the output as no change is required on this column.



- In the **Expression** field of the *dn* column (output), fill in with the exact expression expected by the LDAP server to reach the target tree leaf and allow directory writing on the condition that you haven't set it already in the **Base DN** field of the **tLDAPOutput** component.
- In this use case, the **GetResultName** global variable is used to retrieve this path automatically. Press **Ctrl+Space bar** to access the variable list and select `tLDAPInput_1_RESULT_NAME`.
- In the *mail* column's expression field, type in the new email that will overwrite the current data in the LDAP directory. In this example, we change to *Pierre.Dupont@talend.com*.
- Click **OK** to validate the changes.
- The **tLogRow** component doesn't need any particular setting.
- Then select the **tLDAPOutput** component to set the directory writing properties.



- Set the **Port** and **Host** details manually if they aren't stored in the **Repository**.
- In **Base DN** field, set the highest tree leaf you have the rights to access. If you haven't set previously the exact and full path of the target DN you want to access, then fill in it here. In this use case, the full DN is provided by the *dn* output from the tMap component, therefore only the highest accessible leaf is given: *o=directoryRoot*.
- Select the relevant protocol to be used: **LDAP** for this example.
- Then fill in the **User** and **Password** as expected by the LDAP directory.
- Use the default setting of **Alias Dereferencing** and **Referral Handling** fields, respectively **Always** and **Ignore**.
- The **Insert mode** for this use case is **Update** (the email address).
- The schema was provided by the previous component through the propagation operation.
- Save the Job and execute.

```
Starting job LDAPInputnew at 14:17 20/09/2007.
uid=PIERRE DUPONT,ou=DATA,o=TALENDM,o=TALEND|PIERRE DUPONT|Pierre.Dupont@talend.com
Job LDAPInputnew ended at 14:17 20/09/2007. [exit code=0]
```

The output shows the following fields: *dn*, *uid* and *mail* as defined in the Job.

tMSSqlBulkExec

tMSSqlBulkExec properties

tMSSqlOutputBulk and **tMSSqlBulkExec** components are used together to first output the file that will be then used as parameter to execute the SQL query stated. These two steps compose the **tMSSqlOutputBulkExec** component, detailed in a separate section. The interest in having two separate elements lies in the fact that it allows transformations to be carried out before the data loading in the database.

Component family	Databases/MSSql	
Function	Executes the Insert action on the data provided.	
Purpose	As a dedicated component, tMSSqlBulkExec offers gains in performance while carrying out the Insert operations to a MSSql database	
Basic settings	*Property type*	Either Built-in or Repository.
		Built-in: No property data is stored centrally.
		Repository: Select the **Repository** file where Properties are stored. When selected, the fields to follow are pre-filled in using fetched data.
	Use an existing connection	Select this check box and click the relevant **tMSSqlConnection** component on the Component list to reuse the connection details you already defined.
	Host	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database.
	Schema	Name of the schema.
	Username and *Password*	DB user authentication data.
	Table	Name of the table to be written. Note that only one table can be written at a time and that the table must exist for the insert operation to succeed.
	Action on table	On the table defined, you can perform one of the following operations: **None**: No operation is carried out. **Drop and create table**: The table is removed and created again. **Create table**: The table does not exist and gets created. **Create table if not exists**: The table is created if it does not exist. **Clear table**: The table content is deleted. **Truncate table**: The table content is deleted. You do not have the possibility to rollback the operation.

	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either built-in or remotely stored in the Repository.
		Built-in: The schema is created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	Remote File Name	Name of the file to be processed. Related topic:*Defining variables from the Component view* of **Talend Open Studio** User Guide.
Advanced settings	*Action*	Select the action to be carried out **Bulk insert** **Bulk update** **Bcp query out** Depending on the action selected, the requied information varies.
Bulk insert & Bulk update	*Additional JDBC parameters*	Specify additional connection properties in the existing DB connection.
	Fields terminated	Character, string or regular expression to separate fields.
	Rows terminated	Character, string or regular expression to separate rows.
	First row	Type in the number of the row where the action should start
	Code page	This value can be any of the followings: **OEM** (by default value) **ACP** **RAW** User-defined
	Data file type	Select the type of data being handled.
	Output	Select the type of output for the standard output of the MSSql database: **to console**, **to global variable**.
	*tState*Catcher Statistics	Select this check box to collect log data at the component level.
Bcp query out	*Fields terminated*	Character, string or regular expression to separate fields.
	Rows terminated	Character, string or regular expression to separate rows.
	Data file type	Select the type of data being handled.

	Output	Select the type of output to pass the processed data onto: **to console**: data is viewed in the Log view. **to global variable**: data is put in output variable linked to a tsystem component
	*tState*Catcher Statistics	Select this check box to collect log data at the component level.
Usage	This component is to be used along with **tMSSqlOutputBulk** component. Used together, they can offer gains in performance while feeding a MSSql database.	

Related scenarios

For uses cases in relation with **tMSSqlBulkExec**, see the following scenarios:

- **tMysqlOutputBulk** *Scenario: Inserting transformed data in MySQL database on page 293*
- **tMysqlOutputBulkExec** *Scenario: Inserting data in MySQL database on page 297*

tMSSqlColumnList

tMSSqlColumnList Properties

Component family	Databases/MS SQL	
Function	Iterates on all columns of a given table through a defined MS SQL connection.	
Purpose	Lists all column names of a given MSSql table.	
Basic settings	*Component list*	Select the **tMSSqlConnection** component in the list if more than one connection are planned for the current job.
	Table name	Enter the name of the tabe.
Usage	This component is to be used along with MSSql components, especially with **tMSSqlConnection**.	
Limitation	n/a	

Related scenario

For **tMSSqlColumnList** related scenario, see *Scenario: Iterating on a DB table and listing its column names on page 264*.

tMSSqlInput

tMSSqlInput properties

Component family	Databases/MS SQL Server	
Function	**tMSSqlInput** reads a database and extracts fields based on a query.	
Purpose	**tMSSqlInput** executes a DB query with a strictly defined order which must correspond to the schema definition. Then it passes on the field list to the next component via a Main row link.	
Basic settings	*Property type*	Either Built-in or Repository
		Built-in: No property data stored centrally.
		Repository: Select the repository file where Properties are stored. The fields that come after are pre-filled in using the fetched data.
		Click this icon to open a database connection wizard and store the database connection parameters you set in the component **Basic settings** view. For more information about setting up and storing database connection parameters, see *Setting up a DB connection* of **Talend Open Studio** User Guide.
	Use an existing connection	Select this check box and click the relevant **tMSSqlConnection** component on the Component list to reuse the connection details you already defined.
	Host	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database
	Schema	Name of the schema
	Username and *Password*	DB user authentication data.
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either built-in or remotely stored in the Repository.
		Built-in: The schema is created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	Query type and *Query*	Enter your DB query paying particularly attention to properly sequence the fields in order to match the schema definition.

Advanced settings	*Additional JDBC parameters*	Specify additional connection properties in the existing DB connection.
	Encoding Type	Select the encoding from the list or select Custom and define it manually. This field is compulsory for DB data handling.
	Trim all the String/Char columns	Select this check box to remove leading and trailing whitespace from all the String/Char columns.
	Trim column	Remove leading and trailing whitespace from defined columns.
	*tState*Catcher Statistics	Select this check box to collect log data at the component level.
Usage	This component covers all possibilities of SQL queries onto a MS SQL server database..	

Related scenarios

Related topics in **tDBInput** scenarios:

- *Scenario 1: Displaying selected data from DB table on page 163*
- *Scenario 2: Using StoreSQLQuery variable on page 164*

Related topic in **tContextLoad** *Scenario: Dynamic context use in MySQL DB insert on page 652*.

tMSSqlOutput

tMSSqlOutput properties

Component family	Databases/MS SQL server	
Function	**tMSSqlOutput** writes, updates, makes changes or suppresses entries in a database.	
Purpose	**tMSSqlOutput** executes the action defined on the table and/or on the data contained in the table, based on the flow incoming from the preceding component in the job.	
Basic settings	*Property type*	Either Built-in or Repository.
		Built-in: No property data stored centrally.
		Repository: Select the repository file where Properties are stored. The fields that come after are pre-filled in using the fetched data.
		Click this icon to open a database connection wizard and store the database connection parameters you set in the component **Basic settings** view. For more information about setting up and storing database connection parameters, see *Setting up a DB connection* of **Talend Open Studio** User Guide.
	Use an existing connection	Select this check box and click the relevant **tMSSqlConnection** component on the Component list to reuse the connection details you already defined.
	Host	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database
	Schema	Name of the schema
	Username and *Password*	DB user authentication data.
	Table	Name of the table to be written. Note that only one table can be written at a time

	Action on table	On the table defined, you can perform one of the following operations: **None**: No operation is carried out. **Drop and create a table**: The table is removed and created again. **Create a table**: The table does not exist and gets created. **Create a table if not exists**: The table is created if it does not exist. **Drop a table if exists and create**: The table is removed if it already exists and created again. **Clear a table**: The table content is deleted. **Truncate table**: The table content is deleted. You do not have the possibility to rollback the operation.
	Turn on identity insert	Select this check box to use your own sequence for the identity value of the inserted records (instead of having the SQL Server pick the next sequential value).
	Action on data	On the data of the table defined, you can perform: **Insert**: Add new entries to the table. If duplicates are found, job stops. **Update**: Make changes to existing entries **Insert or update**: Add entries or update existing ones. **Update or insert**: Update existing entries or create it if non existing **Delete**: Remove entries corresponding to the input flow. **Insert if not exist** : Add new entries to the table if they do not exist. *It is necessary to specify at least one column as a primary key on which the Update and Delete operations are based. You can do that by clicking Edit Schema and selecting the check box(es) next to the column(s) you want to set as primary key(s). For an advanced use, click the Advanced settings view where you can simultaneously define primary keys for the Update and Delete operations. To do that: Select the Use field options check box and then in the Key in update column, select the check boxes next to the column names you want to use as a base for the Update operation. Do the same in the Key in delete column for the Delete operation.*
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either built-in or remotely stored in the Repository.
		Built-in: The schema is created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.

		Repository: The schema already exists and is stored in the Repository, hence can be reused. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	Die on error	Clear this check box to skip the row on error and complete the process for error-free rows.
Advanced settings	*Additional JDBC parameters*	Specify additional connection properties in the existing DB connection.
	Encoding Type	Select the encoding type from the list or select **Custom** and define it manually. This field is compulsory for DB data handling.
	Commit every	Enter the number of rows to be completed before committing batches of rows together into the DB. This option ensures transaction quality (but not rollback) and, above all, better performance at execution.
	Additional Columns	This option is not offered if you create (with or without drop) the DB table. This option allows you to call SQL functions to perform actions on columns, which are not insert, nor update or delete actions, or action that require particular preprocessing.
		Name: Type in the name of the schema column to be altered or inserted as new column
		SQL expression: Type in the SQL statement to be executed in order to alter or insert the relevant column data.
		Position: Select **Before**, **Replace** or **After** following the action to be performed on the reference column.
		Reference column: Type in a column of reference that the **tDBOutput** can use to place or replace the new or altered column.
	Use field options	Select this check box to customize a request, especially when there is double action on data.
	Enable debug mode	Select this check box to display each step during processing entries in a database.
	Support null in "SQL WHERE" statement	Select this check box if you want to deal with the Null values contained in a DB table. Make sure the Nullable check box is selected for the corresponding columns in the schema.
	tStateCatcher Statistics	Select this check box to collect log data at the component level.
	Use Batch Size	When selected, enables you to define the number of lines in each processed batch.
Usage	This component offers the flexibility benefit of the DB query and covers all possibilities of SQL queries.	

Related scenarios

For **tMSSqlOutput** related topics, see:

- **tDBOutput** *Scenario: Displaying DB output on page 168*
- **tMySQLOutput** *Scenario 1: Adding a new column and altering data in a DB table on page 283*.

tMSSqlOutputBulk

tMSSqlOutputBulk properties

tMSSqlOutputBulk and **tMSSqlBulkExec** components are used together to first output the file that will be then used as parameter to execute the SQL query stated. These two steps compose the **tMSSqlOutputBulkExec** component, detailed in a separate section. The interest in having two separate elements lies in the fact that it allows transformations to be carried out before the data loading.

Component family	Databases/MSSql	
Function	Writes a file with columns based on the defined delimiter and the MSSql standards	
Purpose	Prepares the file to be used as parameter in the INSERT query to feed the MSSql database.	
Basic settings	*Property type*	Either Built-in or Repository.
		Built-in: No property data stored centrally.
		Repository: Select the Repository file where Properties are stored. When selected, the fields to follow are pre-filled in using fetched data.
	File Name	Name of the file to be processed. Related topic: *Defining variables from the Component view* of **Talend Open Studio** User Guide.
	Append	Select this check box to add the new rows at the end of the records
	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the Repository.
		Built-in: The schema will be created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused in various projects and job designs. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
Advanced settings	*Row separator*	String (ex: “\n”on Unix) to distinguish rows.
	Field separator	Character, string or regular expression to separate fields.
	Include header	Select this check to include the column header.

	Encoding Type	Select the encoding from the list or select Custom and define it manually. This field is compulsory for DB data handling.
	tStaCatcher statistics	Select this check box to collect log data at the component level.
Usage	This component is to be used along with **tMSSqlBulkExec** component. Used together they offer gains in performance while feeding a MSSql database.	

Related scenarios

For uses cases in relation with **tMSSqlOutputBulk**, see the following scenarios:

- **tMysqlOutputBulk** *Scenario: Inserting transformed data in MySQL database on page 293*
- **tMysqlOutputBulkExec** *Scenario: Inserting data in MySQL database on page 297*

tMSSqlOutputBulkExec

tMSSqlOutputBulkExec properties

Component family	Databases/MSSql	
Function	Executes the action on the data provided.	
Purpose	As a dedicated component, it allows gains in performance during Insert operations to a MSSql database.	
Basic settings	*Action*	Select the action to be carried out **Bulk insert** **Bulk update**
	Property type	Either Built-in or Repository.
		Built-in: No property data stored centrally.
		Repository: Select the repository file where Properties are stored. The fields that come after are pre-filled in using the fetched data.
	Use an existing connection	Select this check box and click the relevant **tMSSqlConnection** component on the Component list to reuse the connection details you already defined.
	Host	Database server IP address
	Port	Listening port number of DB server.
	DB name	Name of the database
	Schema	Name of the database schema
	Username and *Password*	DB user authentication data.
	Table	Name of the table to be written. Note that only one table can be written at a time and that the table must exist for the insert operation to succeed.
	Action on table	On the table defined, you can perform one of the following operations: **None**: No operation is carried out. **Drop and create a table**: The table is removed and created again. **Create a table**: The table does not exist and gets created. **Create a table if not exists**: The table is created if it does not exist. **Truncate table**: The table content is deleted. You do not have the possibility to rollback the operation. **Clear a table**: The table content is deleted. You have the possibility to rollback the operation.

	Schema and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either built-in or remotely stored in the Repository.
		Built-in: You create and store the schema locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: You have already created the schema and stored it in the Repository. You can reuse it in various projects and job flowcharts. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	File Name	Name of the file to be processed. Related topic:*Defining variables from the Component view* of **Talend Open Studio** User Guide.
	Append	Select this check box to add the new rows at the end of the records
Advanced settings	*Additional JDBC parameters*	Specify additional connection properties in the existing DB connection.
	Field separator	Character, string or regular expression to separate fields.
	Row separator	String (ex: “\n”on Unix) to distinguish rows.
	First row	Type in the number of the row where the action should start.
	Include header	Select this heck box to include the column header.
	Code page	**OEM** code pages used to map a specific set of characters to numerical code point values.
	Data file type	Select the type of data being handled.
	Encoding Type	Select the encoding from the list or select Custom and define it manually. This field is compulsory for DB data handling.
	tStaCatcher statistics	Select this check box to collect log data at the component level.
Usage	This component is mainly used when no particular transformation is required on the data to be loaded onto the database.	
Limitation	n/a	

Related scenarios

For uses cases in relation with **tMSSqlOutputBulkExec**, see the following scenarios:

- **tMysqlOutputBulk** *Scenario: Inserting transformed data in MySQL database on page 293*
- **tMysqlOutputBulkExec** *Scenario: Inserting data in MySQL database on page 297*

tMSSqlRow

tMSSqlRow properties

Component family	Databases/DB2	
Function	**tMSSqlRow** is the specific component for this database query. It executes the SQL query stated onto the specified database. The row suffix means the component implements a flow in the job design although it doesn't provide output.	
Purpose	Depending on the nature of the query and the database, **tMSSqlRow** acts on the actual DB structure or on the data (although without handling data). The SQLBuilder tool helps you write easily your SQL statements.	
Basic settings	*Property type*	Either Built-in or Repository.
		Built-in: No property data stored centrally.
		Repository: Select the repository file where Properties are stored. The fields that come after are pre-filled in using the fetched data.
	Use an existing connection	Select this check box and click the relevant **tMSSqlConnection** component on the Component list to reuse the connection details you already defined.
	Host	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database
	Schema	Name of the schema
	Username and *Password*	DB user authentication data.
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either built-in or remotely stored in the Repository.
		Built-in: The schema is created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	Table name	Name of the table to be used.
	Turn on identity insert	Select this check box to use your own sequence for the identity value of the inserted records (instead of having the SQL Server pick the next sequential value).
	Query type	Either Built-in or Repository.

		Built-in: Fill in manually the query statement or build it graphically using SQLBuilder
		Repository: Select the relevant query stored in the Repository. The Query field gets accordingly filled in.
	Query	Enter your DB query paying particularly attention to properly sequence the fields in order to match the schema definition.
	Die on error	Clear this check box to skip the row on error and complete the process for error-free rows.
Advanced settings	*Additional JDBC parameters*	Specify additional connection properties in the existing DB connection
	Propagate QUERY's recordset	Select this check box to insert the result of the query into a COLUMN of the current flow.
	Encoding Type	Select the encoding from the list or select **Custom** and define it manually. This field is compulsory for DB data handling.
	Commit every	Number of rows to be completed before committing batches of rows together into the DB. This option ensures transaction quality (but not rollback) and above all better performance on executions.
	*tState*Catcher Statistics	Select this check box to collect log data at the component level.
Usage	This component offers the flexibility benefit of the DB query and covers all possibilities of SQL queries.	

Related scenarios

For related topics, see:

- **tDBSQLRow** *Scenario: Resetting a DB auto-increment on page 172*
- **tMySQLRow** *Scenario: Removing and regenerating a MySQL table index on page 301*.

tMSSqlSCD

tMSSqlSCD belongs to two component families: Business Intelligence and Databases. For more information on it, see *tMSSqlSCD on page 10*.

tMSSqlSP

tMSSqlSP Properties

Component family	Databases/MSSql	
Function	**tMSSqlSP** calls the database stored procedure.	
Purpose	**tMSSqlSP** offers a convenient way to centralize multiple or complex queries in a database and call them easily.	
Basic settings	*Property type*	Either Built-in or Repository.
		Built-in: No property data stored centrally.
		Repository: Select the repository file where Properties are stored. The fields that come after are pre-filled in using the fetched data.
	Use an existing connection	Select this check box and click the relevant **tMSSqlConnection** component on the Component list to reuse the connection details you already defined.
	Host	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database.
	Schema	Name of the schema.
	Username and *Password*	DB user authentication data.
	Schema and *Edit Schema*	In SP principle, the schema is an input parameter. A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either built-in or remotely stored in the Repository.
		Built-in: The schema is created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	SP Name	Type in the exact name of the Stored Procedure
	Is Function / *Return result in*	Select this check box, if only a value is to be returned. Select on the list the schema column, the value to be returned is based on.

	Parameters	Click the Plus button and select the various **Schema Columns** that will be required by the procedures. Note that the SP schema can hold more columns than there are paramaters used in the procedure. Select the **Type** of parameter: **IN**: Input parameter **OUT**: Output parameter/return value **IN OUT**: Input parameters is to be returned as value, likely after modification through the procedure (function). **RECORDSET**: Input parameters is to be returned as a set of values, rather than single value. Check the *tParseRecordSet* component if you want to analyze a set of records from a database table or DB query and return single records.
Advanced settings	*Additional JDBC parameters*	Specify additional connection properties in the existing DB connection.
	Encoding Type	Select the encoding type from the list or select **Custom** and define it manually. This field is compulsory for DB data handling
	tStateCatcher Statistics	Select this check box to collect log data at the component level.
Usage	This component is used as intermediary component. It can be used as start component but only input parameters are thus allowed.	
Limitation	The Stored Procedures syntax should match the Database syntax.	

Related scenario

For related scenarios, see:

- **tMysqlSP** *Scenario: Finding a State Label using a stored procedure on page 307*.
- **tOracleSP** *Scenario: Checking number format using a stored procedure on page 351*

Check as well the *tParseRecordSet* component if you want to analyze a set of records from a database table or DB query and return single records.

tMSSqlTableList

tMSSqlTableList Properties

Component family	Databases/MS SQL	
Function	Iterates on a set of table names through a defined MS SQL connection.	
Purpose	Lists the names of a given set of MSSql tables using a select statement based on a Where clause.	
Basic settings	*Component list*	Select the **tMSSqlConnection** component in the list if more than one connection are planned for the current job.
	Where clause for table name selection	Enter the Where clause to identify the tables to iterate on.
Advanced settings	*tStateCatcher Statistics*	Select this check box to collect log data at the component level.
Usage	This component is to be used along with MSSql components, especially with **tMSSqlConnection**.	
Limitation	n/a	

Related scenario

For **tMSSqlTableList** related scenario, see *Scenario: Iterating on a DB table and listing its column names on page 264*.

tMysqlBulkExec

tMysqlBulkExec properties

tMysqlOutputBulk and **tMysqlBulkExec** components are used together to first output the file that will be then used as parameter to execute the SQL query stated. These two steps compose the **tMysqlOutputBulkExec** component, detailed in a separate section. The interest in having two separate elements lies in the fact that it allows transformations to be carried out before the data loading in the database.

Component family	Databases/Mysql	
Function	Executes the Insert action on the data provided.	
Purpose	As a dedicated component, **tMysqlBulkExec** offers gains in performance while carrying out the Insert operations to a Mysql database	
Basic settings	*Property type*	Either Built-in or Repository.
		Built-in: No property data stored centrally.
		Repository: Select the Repository file where Properties are stored. When selected, the fields to follow are pre-filled in using fetched data.
	Use an existing connection	Select this check box when using a **tMysqlConnection** component.
	Host	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database
	Username and *Password*	DB user authentication data.
	Action on table	On the table defined, you can perform one of the following operations: **None**: No operation is carried out. **Drop and create table**: The table is removed and created again. **Create table**: The table does not exist and gets created. **Create table if not exists**: The table is created if it does not exist. **Truncate table**: The table content is deleted. You do not have the possibility to rollback the operation. **Clear table**: The table content is deleted. You have the possibility to rollback the operation.
	Table	Name of the table to be written. Note that only one table can be written at a time and that the table must exist for the insert operation to succeed.
	Local file Name	Name of the file to be processed. Related topic:*Defining variables from the Component view* of **Talend Open Studio** User Guide.

	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either built-in or remotely stored in the Repository.
		Built-in: The schema is created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
Advanced settings	*Additional JDBC parameters*	Specify additional connection properties in the existing DB connection.
	Lines terminated by	Character or sequence of characters used to separate lines.
	Fields terminated by	Character, string or regular expression to separate fields.
	Escaped by	Character of the row to be escaped.
	Enclosed by	Character used to enclose text.
	Action on data	On the data of the table defined, you can perform: **Insert records in table**: Add new records to the table. **Update records in table**: Make changes to existing records. **Replace records in table**: replace existing records with new one.
	Encoding Type	Select the encoding from the list or select **Custom** and define it manually. This field is compulsory for DB data handling.
	tStatCatcher Statistics	Select this check box to collect log data at the component level.
Usage	This component is to be used along with **tMysqlOutputBulk** component. Used together, they can offer gains in performance while feeding a Mysql database.	
Limitation	n/a	

Related scenarios

For uses cases in relation with **tMysqlBulkExec**, see the following scenarios:

- **tMysqlOutputBulk** *Scenario: Inserting transformed data in MySQL database on page 293*
- **tMysqlOutputBulkExec** *Scenario: Inserting data in MySQL database on page 297*
- **tOracleBulkExec** *Scenario: Truncating and inserting file data into Oracle DB on page 329*

tMysqlColumnList

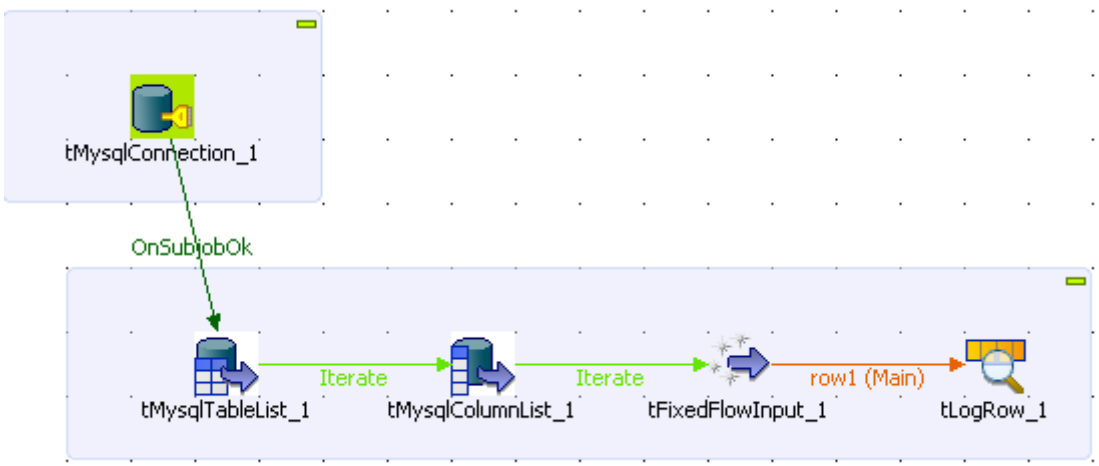
tMysqlColumnList Properties

Component family	Databases/MySQL	
Function	Iterates on all columns of a given table through a defined Mysql connection.	
Purpose	Lists all column names of a given Mysql table.	
Basic settings	*Component list*	Select the **tMysqlConnection** component in the list if more than one connection are planned for the current job.
	Table name	Enter the name of the table.
Usage	This component is to be used along with Mysql components, especially with **tMysqlConnection**.	
Limitation	n/a	

Scenario: Iterating on a DB table and listing its column names

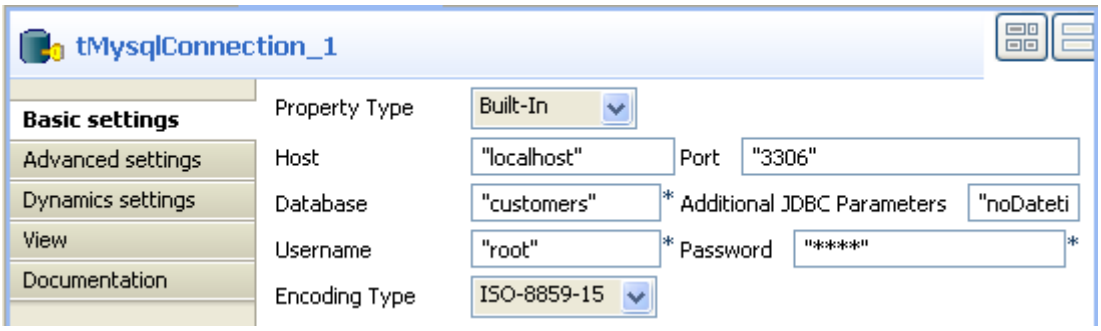
The following Java scenario creates a five-component job that iterates on a given table name from a Mysql database using a Where clause and lists all column names present in the table.

- Drop the following components from the **Palette** onto the design workspace: **tMysqlConnection**, **tMysqlTableList**, **tMysqlColumnList**, **tFixedFlowInput**, and **tLogRow**.
- Connect **tMysqlConnection** to **tMysqlTableList** using an **OnSubjobOk** link.
- Connect **tMysqlTableList**, **tMysqlColumnList**, and **tFixedFlowInput** using **Iterate** links.
- Connect **tFixedFlowInput** to **tLogRow** using a **Row Main** link.



- In the design workspace, select **tMysqlConnection** and click the **Component** tab to define its basic settings.
- In the **Basic settings** view, set the database connection details manually or select them from the context variable list, through a **Ctrl+Space** click in the corresponding field if you have stored them locally as Metadata DB connection entries.

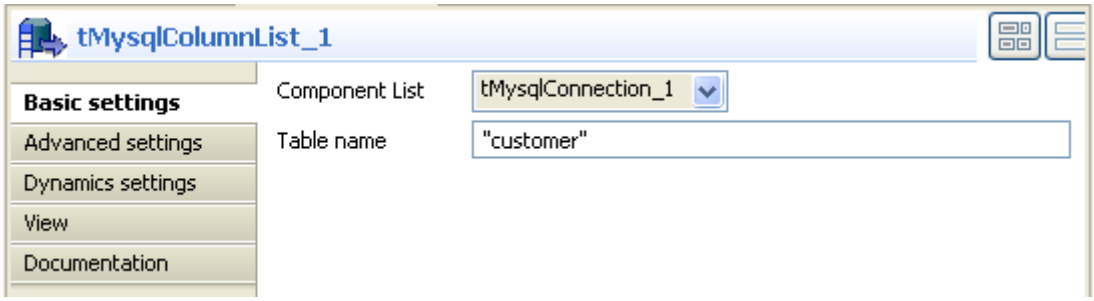
For more information about Metadata, see *Defining Metadata items* of **Talend Open Studio** User Guide.



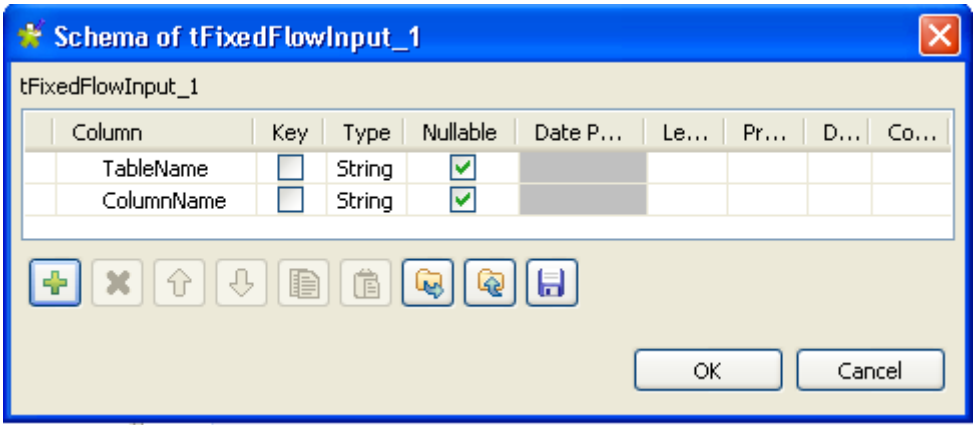
In this example, we want to connect to a Mysql database called *customers*.

- In the design workspace, select **tMysqlTableList** and click the **Component** tab to define its basic settings.

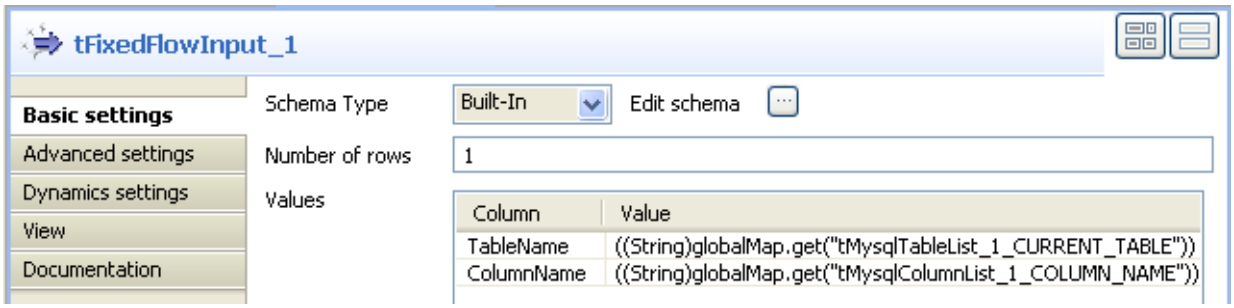
- On the **Component list**, select the relevant Mysql connection component if more than one connection is used.
- Enter a Where clause using the right syntax in the corresponding field to iterate on the table name(s) you want to list on the console.
  In this scenario, the table we want to iterate on is called *customer*.
- In the design workspace, select **tMysqlColumnList** and click the **Component** tab to define its basic settings.



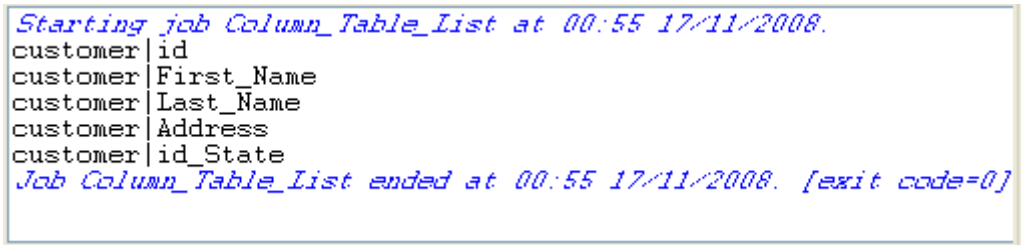
- On the **Component list**, select the relevant Mysql connection component if more than one connection is used.
- In the **Table name** field, enter the name of the DB table you want to list its column names. In this scenario, we want to list the columns present in the DB table called *customer*.
- In the design workspace, select **tFixedFlowInput** and click the **Component** tab to define its basic settings.
- Set the **Schema Type** to **Built-In** and click the three-dot [...] button next to **Edit Schema** to define the data you want to use as input. In this scenario, the schema is made of two columns, the first for the table name and the second for the column name.



- Click **OK** to close the dialog box, and accept propagating the changes when prompted by the system. The defined columns display in the **Values** panel of the **Basic settings** view.
- Click in the **Value** cell for each of the two defined columns and press **Ctrl+Space** to access the global variable list.
- From the global variable list, select *((String)globalMap.get("tMysqlTableList_1_CURRENT_TABLE"))* and *((String)globalMap.get("tMysqlColumnList_1_COLUMN_NAME"))* for the *TableName* and *ColumnName* respectively.



- In the design workspace, select **tLogRow**.
- Click the **Component** tab and define the basic settings for **tLogRow** as needed.
- Save your job and press **F6** to execute it.



The name of the DB table is displayed on the console along with all its column names.

tMysqlCommit

tMysqlCommit Properties

This component is closely related to **tMysqlConnection** and **tMysqlRollback**. It usually doesn't make much sense to use these components independently in a transaction.

Component family	Databases/MySQL	
Function	Validates the data processed through the job into the connected DB	
Purpose	Using a unique connection, this component commits in one go a global transaction instead of doing that on every row or every batch and thus provides gain in performance.	
Basic settings	*Component list*	Select the **tMysqlConnection** component in the list if more than one connection are planned for the current job.
	Close Connection	Clear this check box to continue to use the selected connection once the component has performed its task.
Usage	This component is to be used along with Mysql components, especially with **tMysqlConnection** and **tMysqlRollback** components.	
Limitation	n/a	

Related scenario

This component is closely related to **tMysqlConnection** and **tMysqlRollback**. It usually doesn't make much sense to use one of these without using a **tMysqlConnection** component to open a connection for the current transaction.

For **tMysqlCommit** related scenario, see *tMysqlConnection on page 269*.

tMysqlConnection

tMysqlConnection Properties

This component is closely related to **tMysqlCommit** and **tMysqlRollback**. It usually doesn't make much sense to use one of these without using a **tMysqlConnection** component to open a connection for the current transaction.

Component family	Databases/MySQL	
Function	Opens a connection to the database for a current transaction.	
Purpose	Allows to commit a whole job data in one go to the output database as one transaction when validated.	
Basic settings	*Property type*	Either Built-in or Repository.
		Built-in: No property data stored centrally.
		Repository: Select the repository file where Properties are stored. The fields that come after are pre-filled in using the fetched data.
	Host	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database
	Username and *Password*	DB user authentication data.
	Encoding Type	Select the encoding from the list or select Custom and define it manually. This field is compulsory for DB data handling.
	Use or register a shared DB Connection	Select this check box to share your connection or fetch a connection shared by a parent or child Job. **Shared DB Connection Name:** set or type in the shared connection name.
Usage	This component is to be used along with Mysql components, especially with **tMysqlCommit** and **tMysqlRollback** components.	
Limitation	n/a	

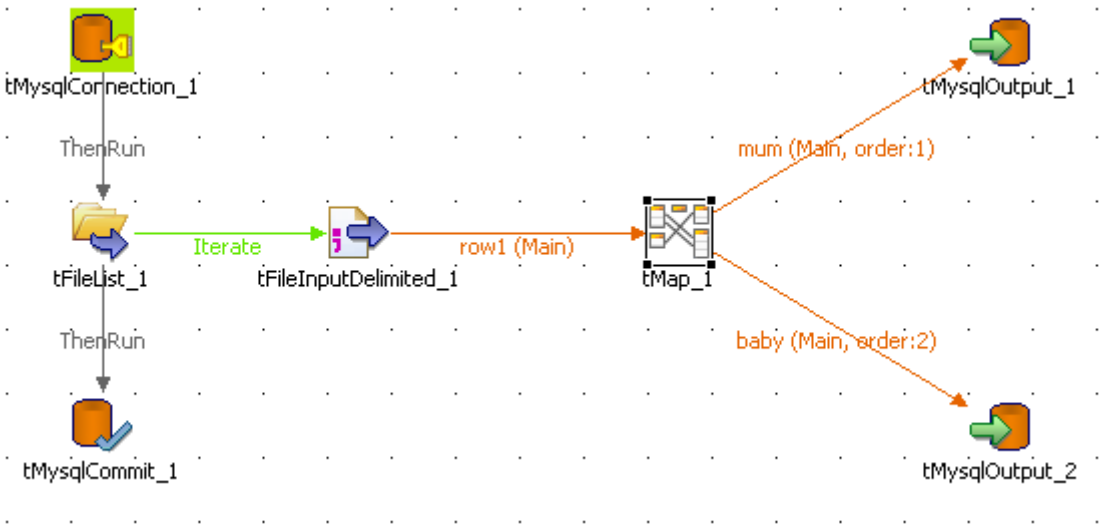
Scenario: Inserting data in mother/daughter tables

The following job is dedicated to advanced database users, who want to carry out multiple table insertions using a parent table id to feed a child table. As a prerequisite to this job, follow the steps described below to create the relevant tables using an engine such as *innodb*.

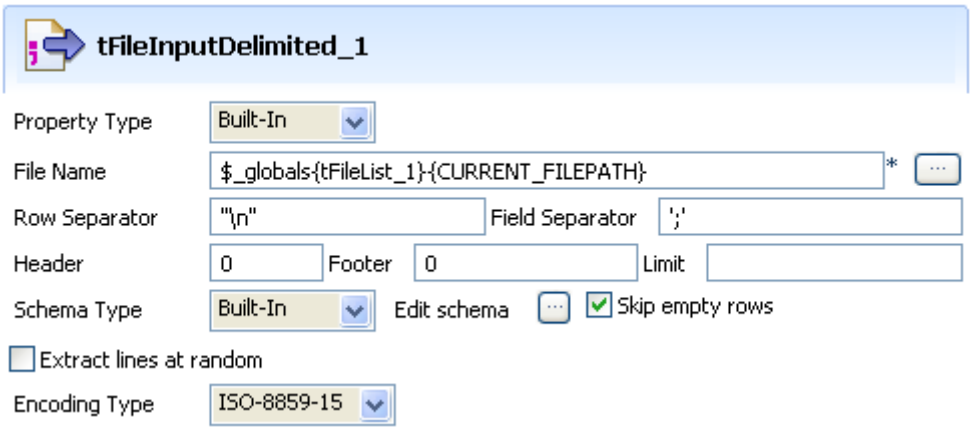
- In a command line editor, connect to your Mysql server.
- Once connected to the relevant database, type in the following command to create the parent table: create table f1090_mum(id int not null auto_increment, name varchar(10), primary key(id)) engine=innodb;

- Then create the second table: create table baby (id_baby int not null, years int) engine=innodb;

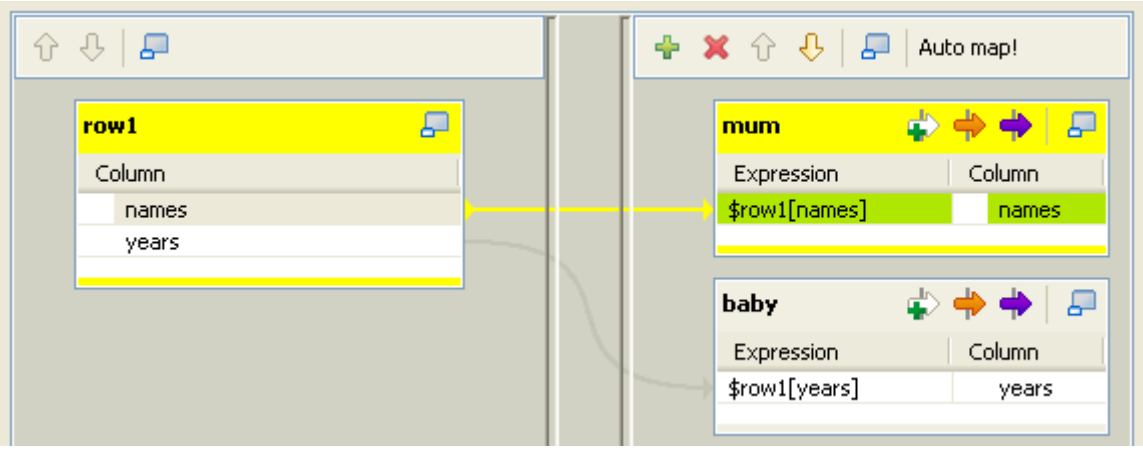
Back into **Talend Open Studio**, the job requires seven components including **tMysqlConnection** and **tMysqlCommit**.



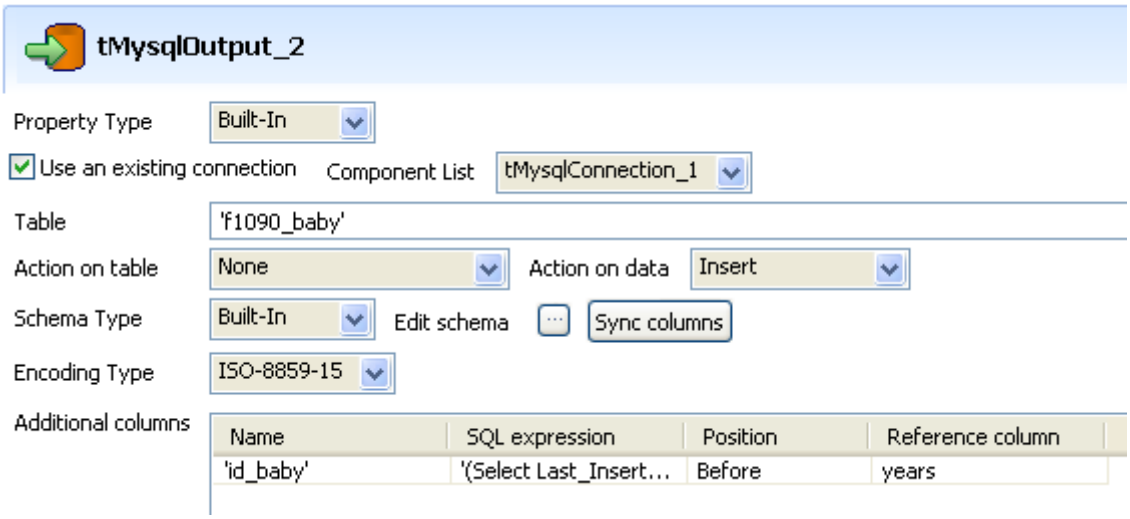
- Drag and drop the following components from the **Palette**: **tFileList**, **tFileInputDelimited**, **tMap**, **tMysqlOutput** (x2).
- Connect the **tFileList** component to the input file component using an **Iterate** link as the name of the file to be processed will be dynamically filled in from the **tFileList** directory using a global variable.
- Connect the **tFileInputDelimited** component to the **tMap** and dispatch the flow between the two output Mysql DB components. Use a **Row** link for each for these connections representing the main data flow.
- Set the **tFileList** component properties, such as the directory. name where files will be fetched from.
- Add a **tMysqlConnection** component and connect it to the starter component of this job, in this example, the **tFileList** component using a ThenRun link to define the execution order.
- In the **tMysqlConnection** Component view, set the connection details manually or fetch them from the Repository if you centrally stored them as a Metadata DB connection entry. For more information about Metadata, see *Defining Metadata items* in **Talend Open Studio** User Guide.
- On the **tFileInputDelimited** component's **Basic settings** panel, press Ctrl+Space bar to access the variable list. Set the **File Name** field to the global variable: *$_globals{tFileList_1}{CURRENT_FILEPATH}*



- Set the rest of the fields as usual, defining the row and field separators according to your file structure.
- Then set the schema manually through the **Edit schema** feature or select the schema from the Repository. In Java version, make sure the data type is correctly set, in accordance with the nature of the data processed.
- Change the encoding if different from the default one.
- In the **tMap** Output area, add two output tables, one called mum for the parent table, the second called baby, for the child table.
- Drag the *Name* column from the **Input** area, and drop it to the mum table.
- Drag the *Years* column from the **Input** area and drop it to the baby table.



- Make sure the mum table is on the top of the baby table as the order is determining for the flow sequence hence the DB insert to perform correctly.
- Then connect the output row link to distribute correctly the flow to the relevant DB output component.
- In each of the **tMysqlOutput** components' **Basic settings** panel, select the **Use an existing connection** check box to retrieve the **tMysqlConnection** details.
- Notice (in Perl version) that the **Commit every** field doesn't show anymore as you are supposed to use the **tMysqlCommit** instead to manage the global transaction commit. In Java version, ignore the field as this command will get overridden by the **tMysqlCommit**.



- Set the **Table** name making sure it corresponds to the correct table, in this example either *f1090_mum* or *f1090_baby*.
- There is no action on the table as they are already created.
- Select **Insert** as **Action on data** for both output components.
- Click on Sync columns to retrieve the schema set in the tMap.
- Change the encoding type if need be.
- In the **Additional columns** area of the DB output component corresponding to the child table (*f1090_baby*), set the id_baby column so that it reuses the id from the parent table.
- In the **SQL expression** field type in: '(Select Last_Insert_id())'
- The position is *Before* and the **Reference column** is *years*.
- Add the **tMysqlCommit** component to the design workspace and connect it from the **tFileList** component using a **ThenRun** connection in order for the job to terminate with the transaction commit.
- On the **tMysqlCommit Component** view, select in the list the connection to be used.

Save your job and press **F6** to run it.

```
mysql> select * from f1090_mum
    -> ;
+----+--------+
| id | names  |
+----+--------+
|  6 | john   |
|  7 | bruce  |
|  8 | beth   |
|  9 | andrew |
| 10 | donald |
| 11 | betty  |
| 12 | john   |
| 13 | bruce  |
| 14 | beth   |
| 15 | andrew |
| 16 | donald |
| 17 | betty  |
+----+--------+
12 rows in set (0.00 sec)
```

```
mysql> select * from f1090_baby
    -> ;
+---------+-------+
| id_baby | years |
+---------+-------+
|       6 |    10 |
|       7 |    23 |
|       8 |    34 |
|       9 |    10 |
|      10 |    23 |
|      11 |    34 |
|      12 |    10 |
|      13 |    23 |
|      14 |    34 |
|      15 |    10 |
|      16 |    23 |
|      17 |    34 |
+---------+-------+
12 rows in set (0.00 sec)
```

The parent table *id* has been reused to feed the *id_baby* column.

tMysqlInput

tMysqlInput properties

Component family	Databases/MySQL	
Function	**tMysqlInput** reads a database and extracts fields based on a query.	
Purpose	**tMysqlInput** executes a DB query with a strictly defined order which must correspond to the schema definition. Then it passes on the field list to the next component via a **Main** row link.	
Basic settings	*Property type*	Either Built-in or Repository
		Built-in: No property data stored centrally.
		Repository: Select the repository file where Properties are stored. The fields that come after are pre-filled in using the fetched data.
		Click this icon to open a database connection wizard and store the database connection parameters you set in the component **Basic settings** view. For more information about setting up and storing database connection parameters, see *Setting up a DB connection* of **Talend Open Studio** User Guide.
	Use existing connection	Select this check box when using a **tMySQLConnection** component.
	Host	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database
	Username and *Password*	DB user authentication data.
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either built-in or remotely stored in the Repository.
		Built-in: The schema is created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	Table Name	Name of the table to be read.
	Query type and *Query*	Enter your DB query paying particularly attention to properly sequence the fields in order to match the schema definition.
Advanced settings	*Additional JDBC parameters*	Specify additional connection properties in the existing DB connection.

	Encoding Type	Select the encoding type from the list or select **Custom** and define it manually. This field is compulsory for DB data handling.
	Enable stream	When selected, helps to decide the row set to work with at a time and thus optimize performance.
	Trim all the String/Char columns	Select this check box to remove leading and trailing whitespace from all the String/Char columns.
	Trim column	Remove leading and trailing whitespace from defined columns.
	tStateCatcher Statistics	Select this check box to collect log data at the component level.
Usage	This component covers all possibilities of SQL queries onto a Mysql database.	

Related scenarios

Related topic in **tDBInput** scenarios:

- *Scenario 1: Displaying selected data from DB table on page 163*
- *Scenario 2: Using StoreSQLQuery variable on page 164*

Related topic in **tContextLoad** *Scenario: Dynamic context use in MySQL DB insert on page 652*.

tMysqlLastInsertId

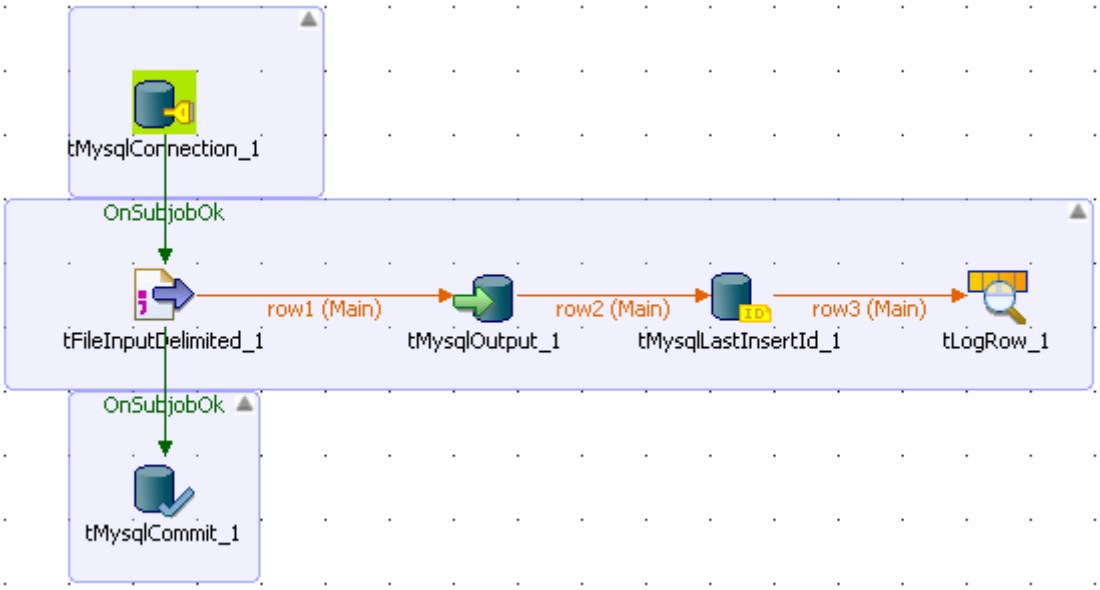
tMysqlLastInsertId properties

Component family	Databases	
Function	**tMysqlLastInsertId** fetches the last inserted ID from a selected MySQL Connection.	
Purpose	**tMysqlLastInsertId** allows to get the primary key value of the record that was last inserted in a Mysql table by a user.	
Basic settings	*Schema type* and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either built-in or remotely stored in the Repository.
		Built-in: You create and store the schema locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: You have already created the schema and stored it in the Repository. You can reuse it in various projects and job flow charts. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	Component list	Select the relevant **tMysqlConnection** component in the list if more than one connection is planned for the current job.
Usage	This component is to be used as an intermediary component.	
Limitation	n/a	

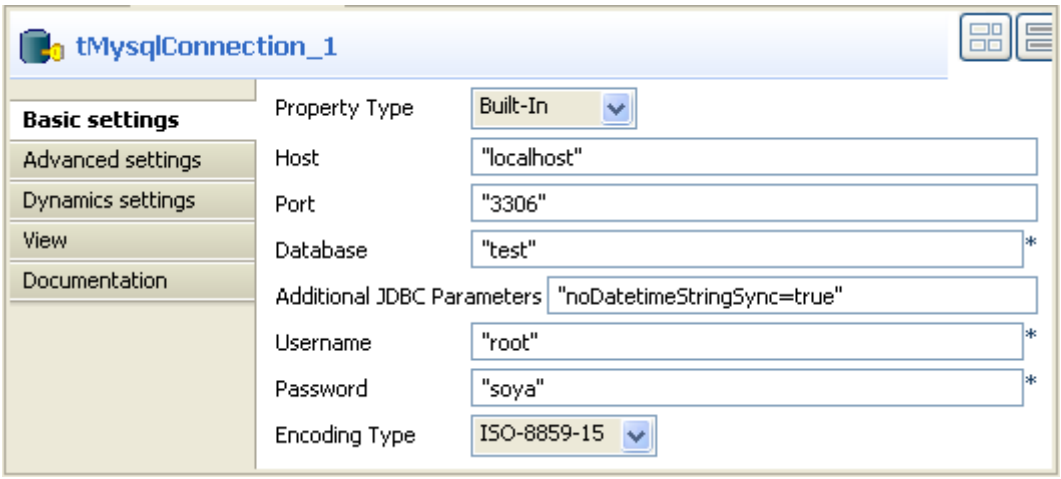
Scenario: Get the ID for the last inserted record

The following Java scenario creates a job that opens a connection to Mysql database, writes the defined data into the database, and finally fetches the last inserted ID on the existing connection.

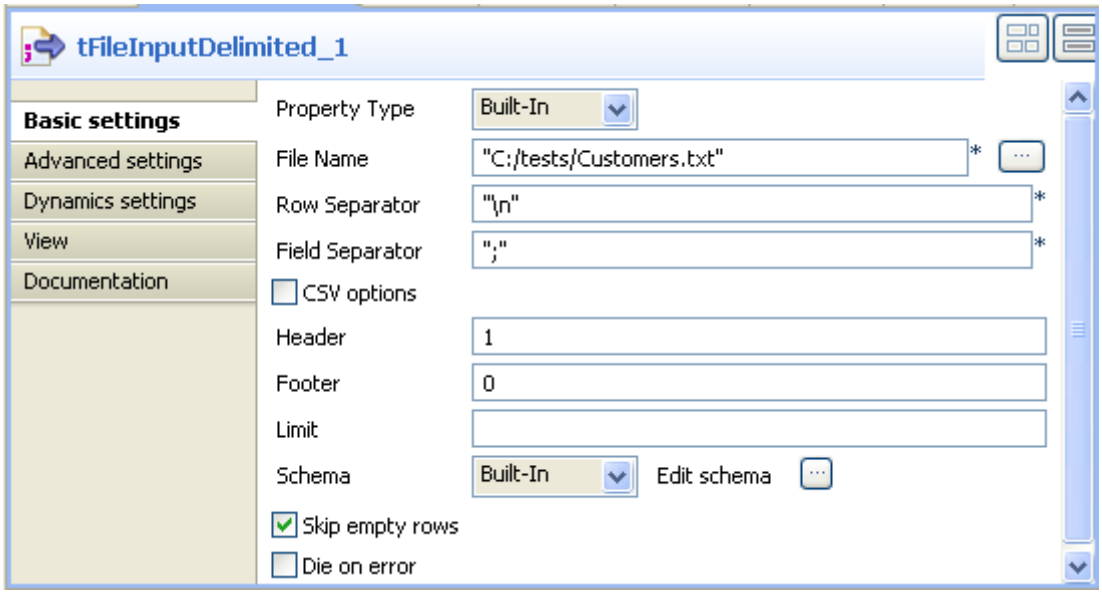
- Drop the following components from the **Palette** onto the design workspace: **tMySqlConnection**, **tMySqlCommit**, **tFileInputDelimited**, **tMySqlOutput**, **tMysqlLastInsertId**, and **tLogRow**.
- Connect **tMySqlConnection** to **tFileInputDelimited** using an **OnSubjobOk** link.
- Connect **tFileInputDelimited** to **tMySqlCommit** using an **OnSubjobOk** link.
- Connect **tFileInputdelimited** to the three other components using **Row Main** links.



- In the design workspace, select **tMysqlConnection**.
- Click the **Component** tab to define the basic settings for **tMysqlConnection**.
- In the **Basic settings** view, set the connection details manually or select them from the context variable list, through a **Ctrl+Space** click in the corresponding field if you stored them locally as Metadata DB connection entries. For more information about Metadata, see *Defining Metadata items* of **Talend Open Studio** User Guide.

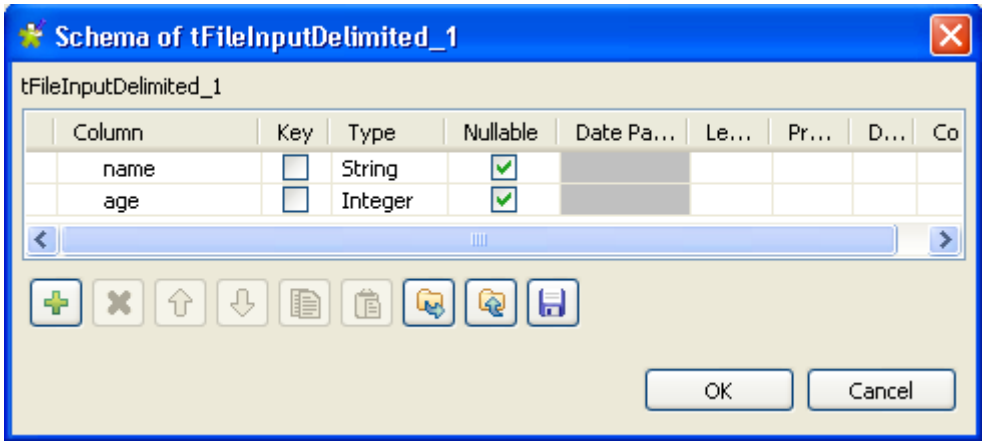


- In the design workspace, select **tMysqlCommit** and click the **Component** tab to define its basic settings.
- On the **Component List**, select the relevant **tMysqlConnection** if more than one connection is used.
- In the design workspace, select **tFileInputDelimited**.
- Click the **Component** tab to define the basic settings of **tFileInputDelimited**.



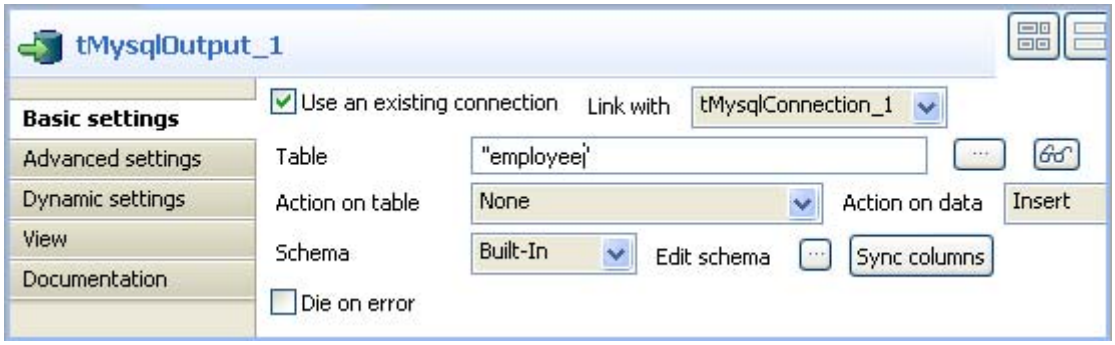
- Set **Property Type** to **Built-In**.
- Fill in a path to the processed file in the **File Name** field. The file used in this example is *Customers*.
- Define the **Row separator** that allow to identify the end of a row. Then define the **Field separator** used to delimit fields in a row.
- Set the header, the footer and the number of processed rows as necessary. In this scenario, we have one header.
- Set **Schema** to **Built in** and click the three-dot button next to **Edit Schema** to define the data to pass on to the next component.

Related topics: *Setting a built-in schema* and *Setting a Repository schema* of **Talend Open Studio** User Guide.

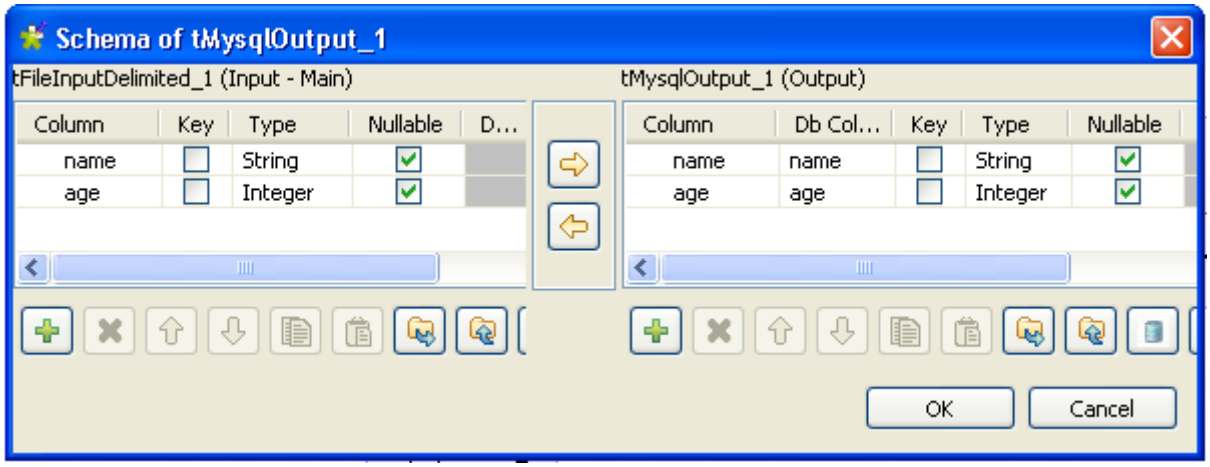


In this scenario, the schema consists of two columns, *name* and *age*. The first holds three employees' names and the second holds the corresponding age for each.

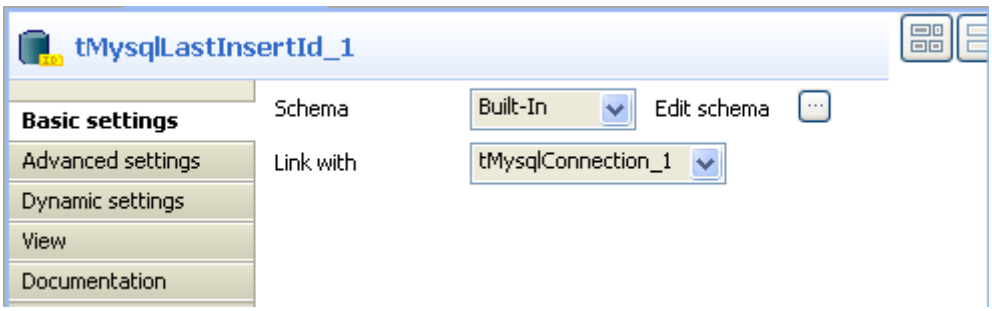
- In the design workspace, select **tMySqlOutput**.
- Click the **Component** tab to define the basic settings of **tMySqlOuptput**.



- Select the **Use an existing connection** check box.
- In the **Table** field, enter the name of the table where to write the employees' list, in this example: *employee*.
- Select relevant actions on the **Action on table** and **Action on data** lists. In this example, no action is carried out on table, and the action carried out on data is *Insert*.
- Set **Schema** to **Built-In** and click **Sync columns** to synchronize columns with the previous component. In this example, the schema to be inserted into the MySql database table consists of the two columns *name* and *age*.

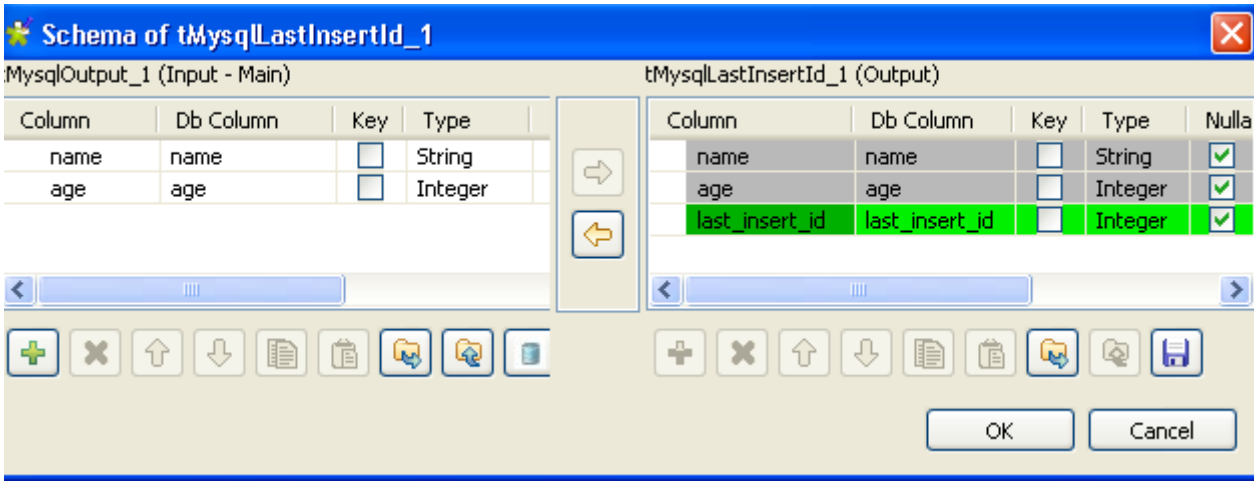


- In the design workspace, select **tMySqlLastInsertId**.
- Click the **Component** tab to define the basic settings of **tMySqlLastInserId**.



- On the **Component List**, select the relevant **tMysqlConnection**, if more than one connection is used.

- Set **Schema** to **Built-In** and click **Sync columns** to synchronize columns with the previous component. In the output schema of **tMySqlLastInsertId**, you can see the read-only column *last_insert_id* that will fetch the last inserted ID on the existing connection.



- In the design workspace, select **tLogRow** and click the **Component** tab to define its basic settings. For more information, see *tLogRow on page 628*.
- Save your job and press **F6** to execute it.

```
Starting job lastinserted at 11:12 01/09/2008.
.--------+---+--------------.
|          tLogRow_1          |
|=--------+---+--------------=|
|name     |age|last_insert_id|
|=--------+---+--------------=|
|Marie    |24 |40            |
|Patrick  |22 |41            |
|Pierrick|27 |42            |
'--------+---+--------------'

Job lastinserted ended at 11:12 01/09/2008. [exit code=0]
```

tMysqlLastInsertId fetched the last inserted ID for each line on the existing connection.

tMysqlOutput

tMysqlOutput properties

Component family	Databases/MySQL	
Function	**tMysqlOutput** writes, updates, makes changes or suppresses entries in a database.	
Purpose	**tMysqlOutput** executes the action defined on the table and/or on the data contained in the table, based on the flow incoming from the preceding component in the job.	
Basic settings	*Property type*	Either Built-in or Repository.
		Built-in: No property data stored centrally.
		Repository: Select the repository file where Properties are stored. The fields that come after are pre-filled in using the fetched data.
		Click this icon to open a database connection wizard and store the database connection parameters you set in the component **Basic settings** view. For more information about setting up and storing database connection parameters, see *Setting up a DB connection* of **Talend Open Studio** User Guide.
	Use existing connection	Select this check box when using a **tMySQLConnection** component.
	Host	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database
	Username and *Password*	DB user authentication data.
	Table	Name of the table to be written. Note that only one table can be written at a time
	Action on table	On the table defined, you can perform one of the following operations: **None**: No operation is carried out. **Drop and create a table**: The table is removed and created again. **Create a table**: The table does not exist and gets created. **Create a table if not exists**: The table is created if it does not exist. **Drop a table if exists and create**: The table is removed if it already exists and created again. **Clear a table**: The table content is deleted.

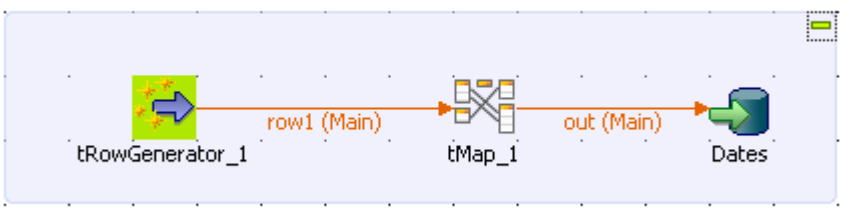
	Action on data	On the data of the table defined, you can perform: **Insert**: Add new entries to the table. If duplicates are found, job stops. **Update**: Make changes to existing entries **Insert or update**: Add entries or update existing ones. **Update or insert**: Update existing entries or create it if non existing **Delete**: Remove entries corresponding to the input flow. *It is necessary to specify at least one column as a primary key on which the Update and Delete operations are based. You can do that by clicking Edit Schema and selecting the check box(es) next to the column(s) you want to set as primary key(s). For an advanced use, click the Advanced settings view where you can simultaneously define primary keys for the Update and Delete operations. To do that: Select the Use field options check box and then in the Key in update column, select the check boxes next to the column names you want to use as a base for the Update operation. Do the same in the Key in delete column for the Delete operation.*
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either built-in or remotely stored in the Repository.
		Built-in: The schema is created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	Die on error	Clear this check box to skip the row on error and complete the process for error-free rows.
Advanced settings	*Additional JDBC parameters*	Specify additional connection properties in the existing DB connection.
	Extend Insert	Select this check box to carry out a bulk insert of a definable set of lines instead of inserting lines one by one. The gain in system performance is huge. **Number of rows per insert**:: enter the number of rows to be inserted as one block. Note that too high value decreases performance due to memory issues.
	Encoding Type	Select the encoding type from the list or select **Custom** and define it manually. This field is compulsory for DB data handling.

	Commit every	Number of rows to be completed before committing batches of rows together into the DB. This option ensures transaction quality (but not rollback) and, above all, better performance at executions.
	Additional Columns	This option is not offered if you create (with or without drop) the DB table. This option allows you to call SQL functions to perform actions on columns, which are not insert, nor update or delete actions, or action that require particular preprocessing.
		Name: Type in the name of the schema column to be altered or inserted as new column
		SQL expression: Type in the SQL statement to be executed in order to alter or insert the relevant column data.
		Position: Select **Before**, **Replace** or **After** following the action to be performed on the reference column.
		Reference column: Type in a column of reference that the **tDBOutput** can use to place or replace the new or altered column.
	Use field options	Select this check box to customize a request, especially when there is double action on data.
	Enable debug mode	Select this check box to display each step during processing entries in a database.
	tStateCatcher Statistics	Select this check box to collect log data at the component level.
Usage	This component offers the flexibility benefit of the DB query and covers all possibilities of SQL queries.	

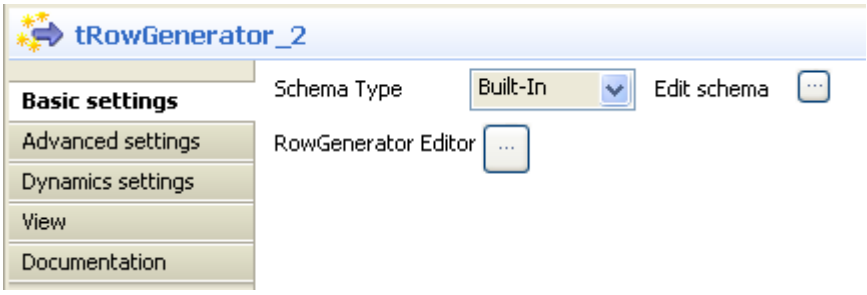
Scenario 1: Adding a new column and altering data in a DB table

This Java scenario is a three-component job that aims at creating random data using a **tRowGenerator**, duplicating a column to be altered using the **tMap** component, and eventually altering the data to be inserted based on an SQL expression using the **tMysqlOutput** component.

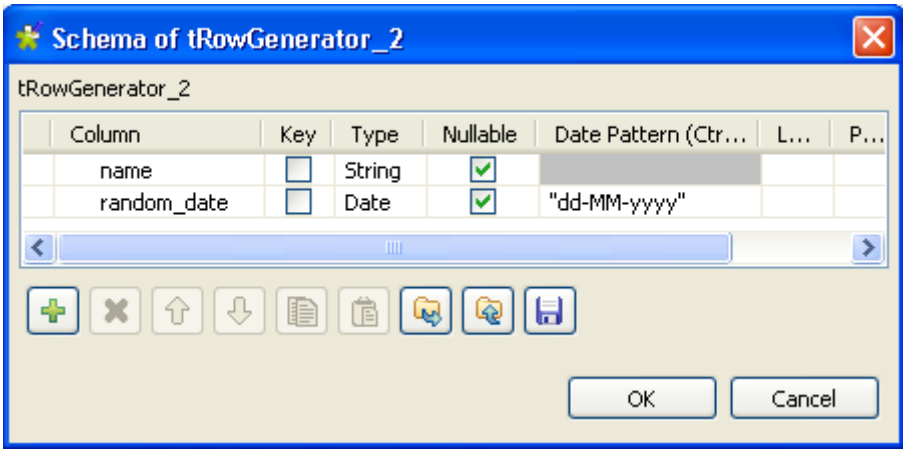
- Drop the following components from the **Palette** onto the design workspace: **tRowGenerator**, **tMap** and **tMySQLOutput**.
- Connect **tRowGenerator**, **tMap**, and **tMysqlOutput** using the **Row Main** link.



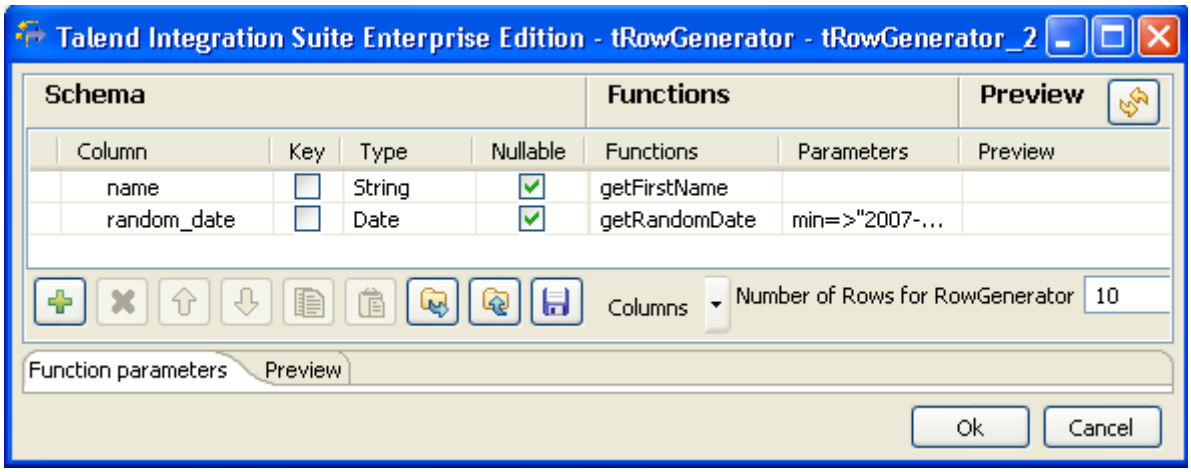
- In the design workspace, select **tRowGenerator** to display its **Basic settings** view.



- Set the **Schema Type** to **Built-In**.
- Click the **Edit schema** three-dot button to define the data to pass on to the **tMap** component, two columns in this scenario, *name* and *random_date*.

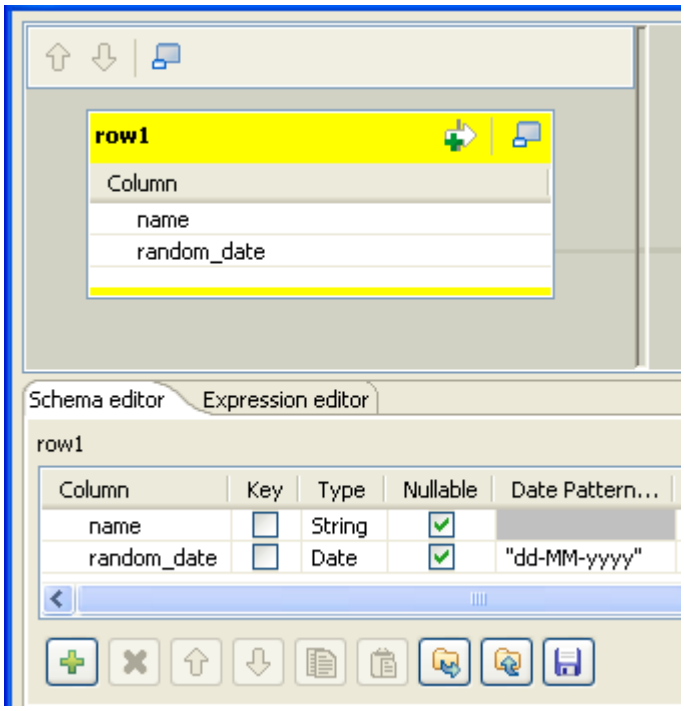


- Click **OK** to close the dialog box.
- Click the **RowGenerator Editor** three-dot button to open the editor and define the data to be generated.

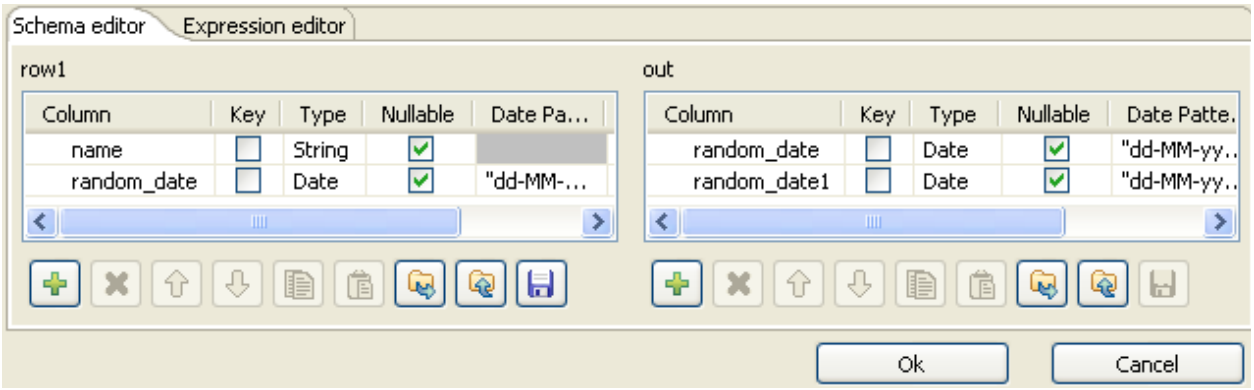


- Click in the corresponding **Functions** fields and select a function for each of the two columns, `getFirstName` for the first column and `getrandomDate` for the second column.
- In the **Number of Rows for Rowgenerator** field, enter 10 to generate ten first name rows and click **Ok** to close the editor.

- Double-click the **tMap** component to open the Map editor. The Map editor opens displaying the input metadata of the **tRowGenerator** component.

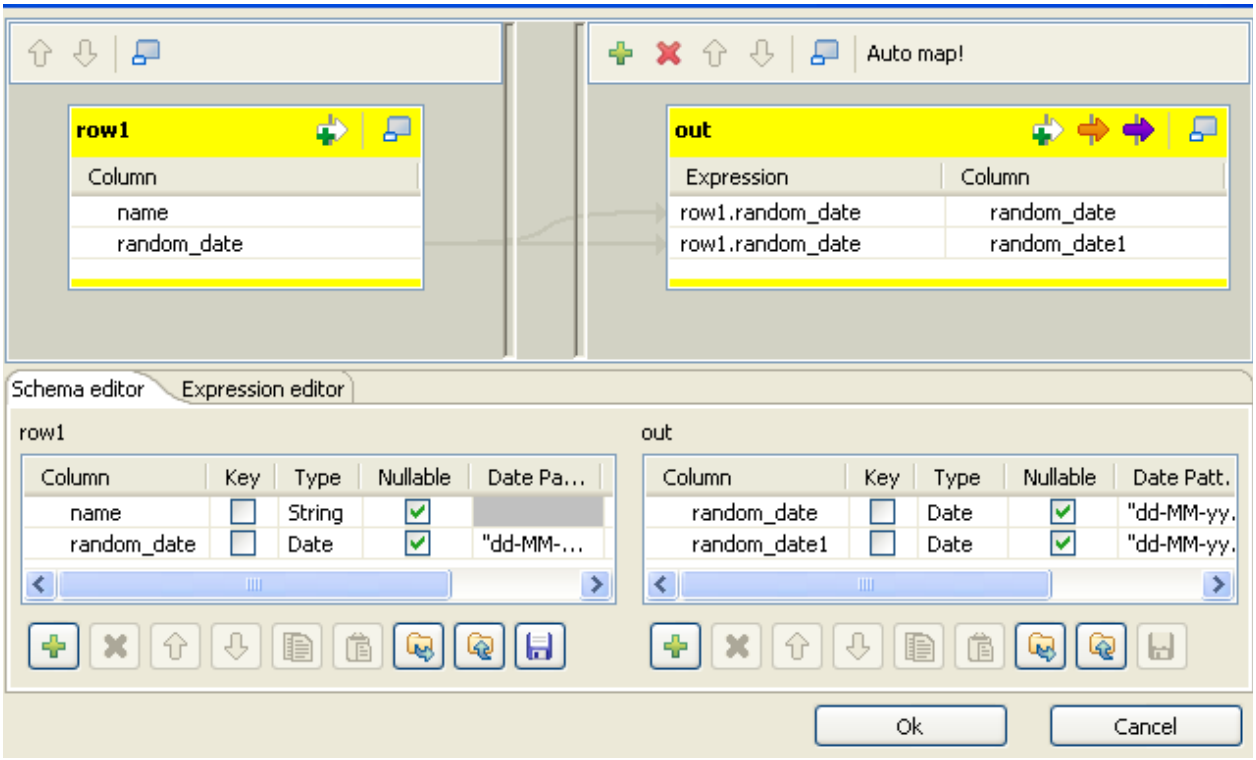


- In the **Schema editor** panel of the Map editor, click the plus button of the output table to add two rows and define the first as *random_date* and the second as *random_date1*.

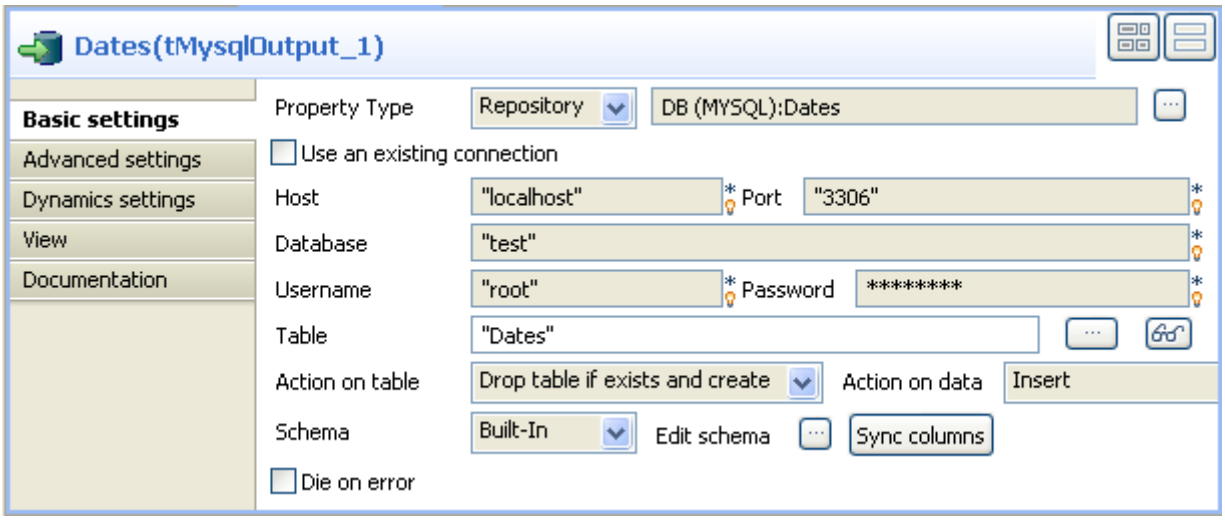


In this scenario, we want to duplicate the *random_date* column and adapt the schema in order to alter the data in the output component.

- In the Map editor, drag the *random_date* row from the input table to the *random_date* and *random_date1* rows in the output table.



- Click **OK** to close the editor.
- In the design workspace, double-click the **tMysqlOutput** component to display its **Basic settings** view and set its parameters.

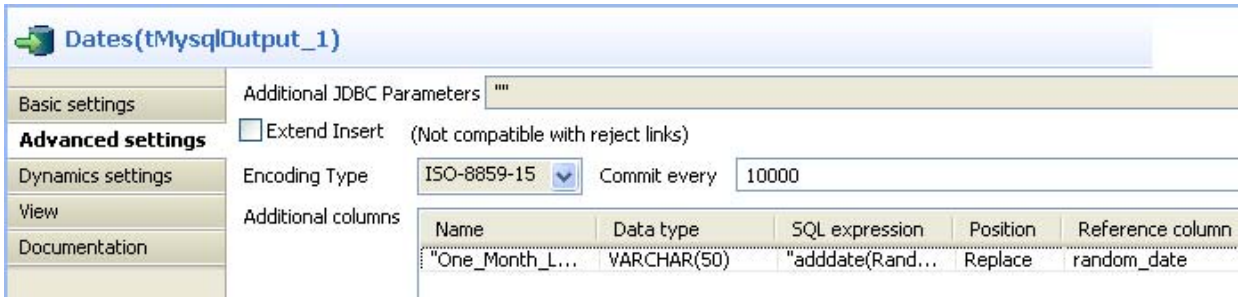


- Set **Property Type** to **Repository** and then click the three-dot button to open the **[Repository content]** dialog box and select the correct DB connection. The connection details display automatically in the corresponding fields.

  If you have not stored the DB connection details in the **Metadata** entry in the **Repository**, select **Built-in** on the property type list and set the connection detail manually.

- Click the three-dot button next to the **Table** field and select the table to be altered, *Dates* in this scenario.

- On the **Action on table** list, select **Drop table if exists and create**, select *Insert* on the **Action on data** list.
- If needed, click **Sync columns** to synchronize with the columns coming from the **tMap** component.
- Click the **Advanced settings** tab to display the corresponding view and set the advanced parameters.



- In the **Additional Columns** area, set the alteration to be performed on columns.
  In this scenario, the *One_month_later* column replaces *random_date_1*. Also, the data itself gets altered using an SQL expression that adds one month to the randomly picked-up date of the *random_date_1* column. ex: 2007-08-12 becomes 2007-09-12.

  -Enter *One_Month_Later* in the **Name** cell.
  -In the **SQL expression** cell, enter the relevant addition script to be performed, `"adddate(Random_date, interval 1 month)"` in this scenario.
  -Select **Replace** on the **Position** list.
  -Enter *Random_date1* on the **Reference column** list.

  For this job we duplicated the *random_date_1* column in the DB table before replacing one instance of it with the *One_Month_Later* column. The aim of this workaround was to be able to view upfront the modification performed.

- Save your job and press **F6** to execute it.

The new *One_month_later* column replaces the *random_date1* column in the DB table and adds one month to each of the randomly generated dates.

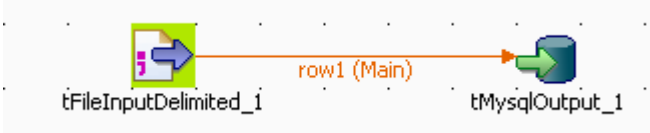
Random_date	One_Month_Later
2007-03-13 21:52:09	2007-04-13 21:52:09
2008-11-02 15:33:29	2008-12-02 15:33:29
2008-05-17 18:47:15	2008-06-17 18:47:15
2007-09-06 17:44:36	2007-10-06 17:44:36
2008-11-17 11:46:36	2008-12-17 11:46:36
2007-05-19 00:46:58	2007-06-19 00:46:58
2008-04-20 15:36:00	2008-05-20 15:36:00
2007-07-24 21:06:08	2007-08-24 21:06:08
2008-04-03 13:24:17	2008-05-03 13:24:17
2007-06-29 13:22:23	2007-07-29 13:22:23

Related topic: *tDBOutput properties on page 166*.

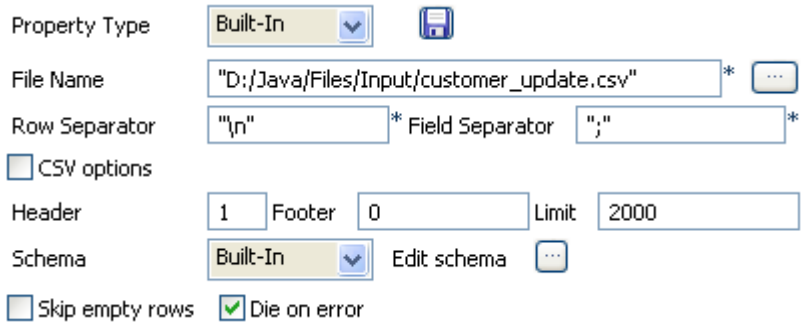
Scenario 2: Updating data in a database table

This Java scenario describes a two-component Job that updates data in a MySQL table according to that in a delimited file.

- Drop **tFileInputDelimited** and **tMySQLOutput** from the **Palette** onto the design workspace.
- Connect the two components together using a **Row Main** link.



- Double-click **tFileInputDelimited** to display its **Basic settings** view and define the component properties.
- From the **Property Type** list, select **Repository** if you have already stored the metadata of the delimited file in the **Metadata** node in the **Repository** tree view. Otherwise, select **Built-In** to define manually the metadata of the delimited file.
  For more information about storing metadata, see *Setting up a File Delimited schema on page 184* of **Talend Open Studio** User Guide.



- In the **File Name** field, click the three-dot button and browse to the source delimited file that contains the modifications to propagate in the MySQL table.
  In this example, we use the *customer_update* file that holds four columns: *id*, *CustomerName*, *CustomerAddress* and *idState*. Some of the data in these four columns is different from that in the MySQL table.

id	CustomerName	CustomerAddress	idState
858	Froggy's Gourmet Catering	1831 Beverly Place #9D	4
859	Dependable Plumbing and Sewer	1550 Ridge Rd.	25
860	Lickmen Restoration	1235 Easton Rd.	40
861	Acturial Enterprises Ltd.	3148 cottonwood Ct.	18
862	Rythmics Ltd.	857 Woodbine Rd	30
863	Acturial Enterprises Ltd.	1482 Concorde Circle	48
864	Crosstracks Car Wash	218 Oakridge Ave.	39
865	Meonits & Mogogni Inc.	616 Cobblestone Cir.	17
866	Foy Aviation	2220 Grant Blvd.	50
867	Ebert Music Center	12 Broadview Lane	29
868	janice Mann Accounting Service	1660 Park Ave.	9
869	Johnson, Erico & Co CPA's	2922 Twin Oaks Drive	40
870	Corbins, Rodriguez, & Savocchi	115 Pleasent Ave.	18
871	Nina's Snow Plowing	3385 University Ave.	20
872	Darcy Frame and Matting Servic	1101 Deerfield Place	47
873	Marks, Marks, and Kaplan Ltd.	1949 Cloverdale Rd.	9

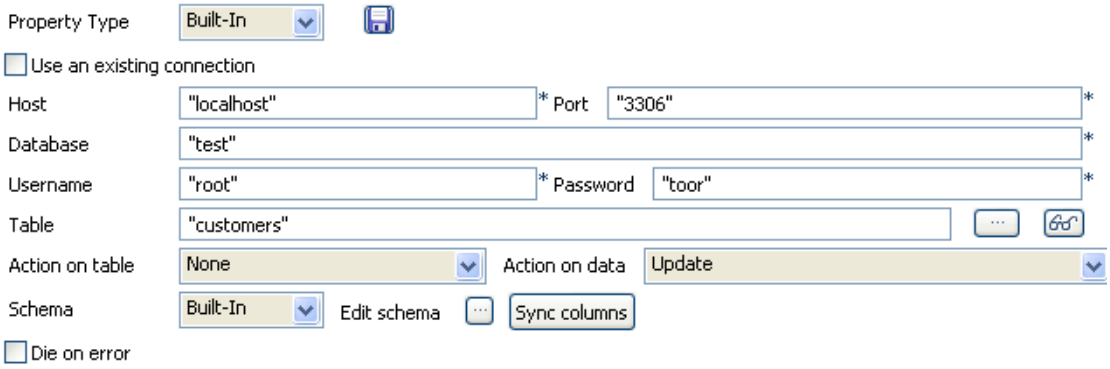
- Define the row and field separators used in the source file in the corresponding fields.
- If needed, set **Header**, **Footer** and **Limit**.
  In this example, **Header** is set to 1 since the first row holds the names of columns, therefore it should be ignored. Also, the number of processed lines is limited to 2000.
- Select **Built in** from the **Schema** list then click the three-dot button next to **Edit Schema** to open a dialog box where you can describe the data structure of the source delimited file that you want to pass to the component that follows.

Column	Key	Type	Nullable
id	☑	Integer	☑
CustomerName	☐	String	☑
CustomerAddress	☐	String	☑
idState	☐	Integer	☑

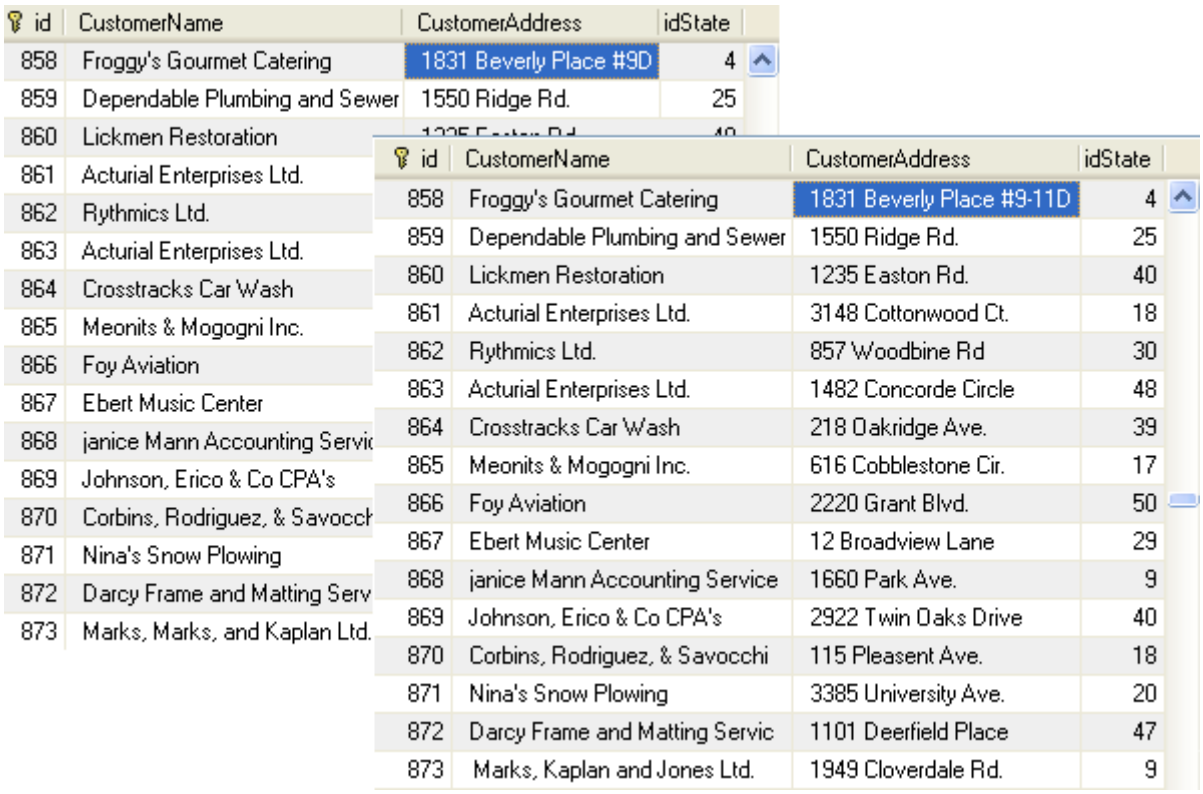
- Select the **Key** check box(es) next to the column name(s) you want to define as key column(s).

  It is necessary to define at least one column as a key column for the Job to be executed correctly. Otherwise, the Job is automatically interrupted and an error message displays on the console.

- In the design workspace, double-click **tMysqlOutput** to open its **Basic settings** view where you can define its properties.



- Click **Sync columns** to retrieve the schema of the preceding component. If needed, click the three-dot button next to **Edit schema** to open a dialog box where you can check the retrieved schema.
- From the **Property Type** list, select **Repository** if you have already stored the connection metadata in the **Metadata** node in the **Repository** tree view. Otherwise, select **Built-In** to define manually the connection information.
  For more information about storing metadata, see *Setting up a DB connection on page 168* of **Talend Open Studio** User Guide.
- Fill in the database connection information in the corresponding fields.
- In the **Table** field, enter the name of the table to update.
- From the **Action on table** list, select the operation you want to perform, **None** in this example since the table already exists.
- From the **Action on data** list, select the operation you want to perform on the data, **Update** in this example.
- Save your Job and press **F6** to execute it.



id	CustomerName	CustomerAddress	idState
858	Froggy's Gourmet Catering	1831 Beverly Place #9D	4
859	Dependable Plumbing and Sewer	1550 Ridge Rd.	25
860	Lickmen Restoration		
861	Acturial Enterprises Ltd.		
862	Rythmics Ltd.		
863	Acturial Enterprises Ltd.		
864	Crosstracks Car Wash		
865	Meonits & Mogogni Inc.		
866	Foy Aviation		
867	Ebert Music Center		
868	janice Mann Accounting Servic		
869	Johnson, Erico & Co CPA's		
870	Corbins, Rodriguez, & Savocch		
871	Nina's Snow Plowing		
872	Darcy Frame and Matting Serv		
873	Marks, Marks, and Kaplan Ltd.		

id	CustomerName	CustomerAddress	idState
858	Froggy's Gourmet Catering	1831 Beverly Place #9-11D	4
859	Dependable Plumbing and Sewer	1550 Ridge Rd.	25
860	Lickmen Restoration	1235 Easton Rd.	40
861	Acturial Enterprises Ltd.	3148 Cottonwood Ct.	18
862	Rythmics Ltd.	857 Woodbine Rd	30
863	Acturial Enterprises Ltd.	1482 Concorde Circle	48
864	Crosstracks Car Wash	218 Oakridge Ave.	39
865	Meonits & Mogogni Inc.	616 Cobblestone Cir.	17
866	Foy Aviation	2220 Grant Blvd.	50
867	Ebert Music Center	12 Broadview Lane	29
868	janice Mann Accounting Service	1660 Park Ave.	9
869	Johnson, Erico & Co CPA's	2922 Twin Oaks Drive	40
870	Corbins, Rodriguez, & Savocchi	115 Pleasent Ave.	18
871	Nina's Snow Plowing	3385 University Ave.	20
872	Darcy Frame and Matting Servic	1101 Deerfield Place	47
873	Marks, Kaplan and Jones Ltd.	1949 Cloverdale Rd.	9

Using you DB browser, you can verify if the MySQL table, *customers*, has been modified according to the delimited file.

In the above example, the database table has always the four columns *id*, *CustomerName*, *CustomerAddress* and *idState*, but certain fields have been modified according to the data in the delimited file used.

tMysqlOutputBulk

tMysqlOutputBulk properties

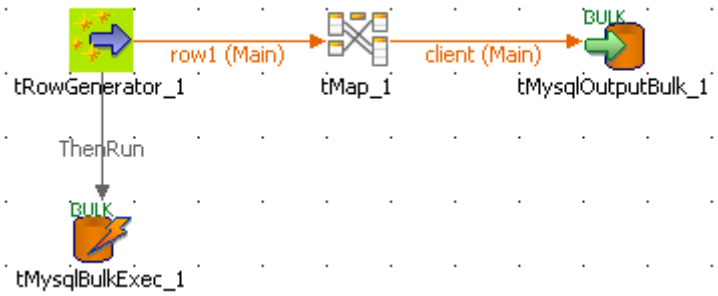
tMysqlOutputBulk and **tMysqlBulkExec** components are used together to first output the file that will be then used as parameter to execute the SQL query stated. These two steps compose the **tMysqlOutputBulkExec** component, detailed in a separate section. The interest in having two separate elements lies in the fact that it allows transformations to be carried out before the data loading.

Component family	Databases/MySQL	
Function	Writes a file with columns based on the defined delimiter and the MySql standards	
Purpose	Prepares the file to be used as parameter in the INSERT query to feed the MySQL database.	
Basic settings	*Property type*	Either Built-in or Repository.
		Built-in: No property data stored centrally.
		Repository: Select the Repository file where Properties are stored. The following fields are pre-filled in using fetched data.
	File Name	Name of the file to be processed. Related topic:*Defining variables from the Component view* of **Talend Open Studio** User Guide
	Field separator	Character, string or regular expression to separate fields.
	Row separator	String (ex: “\n”on Unix) to distinguish rows.
	Append	Select this check box to add the new rows at the end of the file
	Include header	Select this check box to include the column header to the file.
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the Repository.
		Built-in: The schema will be created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused in various projects and job designs. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.

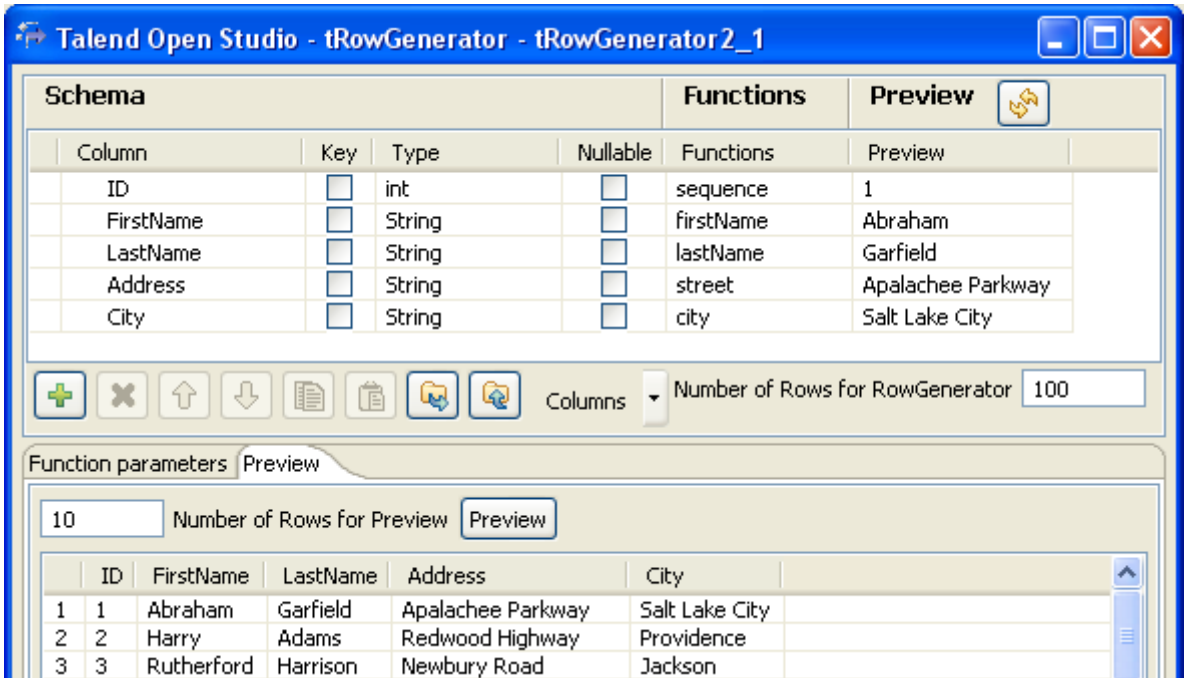
	Encoding Type	Select the encoding from the list or select Custom and define it manually. This field is compulsory for DB data handling.
Usage	This component is to be used along with **tMySQlBulkExec** component. Used together they offer gains in performance while feeding a MySQL database.	

Scenario: Inserting transformed data in MySQL database

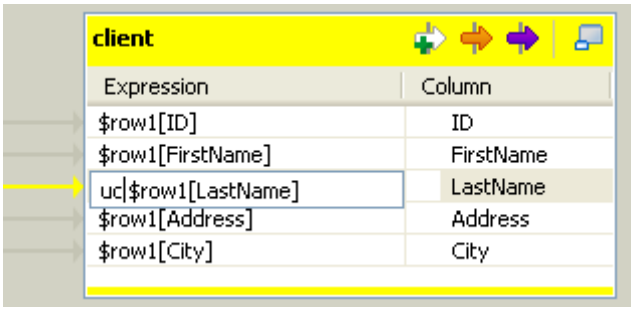
This scenario describes a four-component job which aims at fueling a database with data contained in a file, including transformed data. Two steps are required in this job, first step is to create the file, that will then be used in the second step. The first step includes a tranformation phase of the data included in the file.



- Drag and drop a **tRowGenerator**, a **tMap**, a **tMysqlOutputBulk** as well as a **tMysqlBulkExec** component.
- Connect the main flow using **row main** links.
- And connect the start component (**tRowgenerator** in this example) to the **tMysqlBulkExec** using a **trigger** connection, of type **ThenRun**.
- A **tRowGenerator** is used to generate random data. Double-click on the **tRowGenerator** component to launch the editor.
- Define the schema of the rows to be generated and the nature of data to generate. In this example, the *clients* file to be produced will contain the following columns: *ID*, *First Name*, *Last Name*, *Address*, *City* which all are defined as string data but the ID that is of integer type.

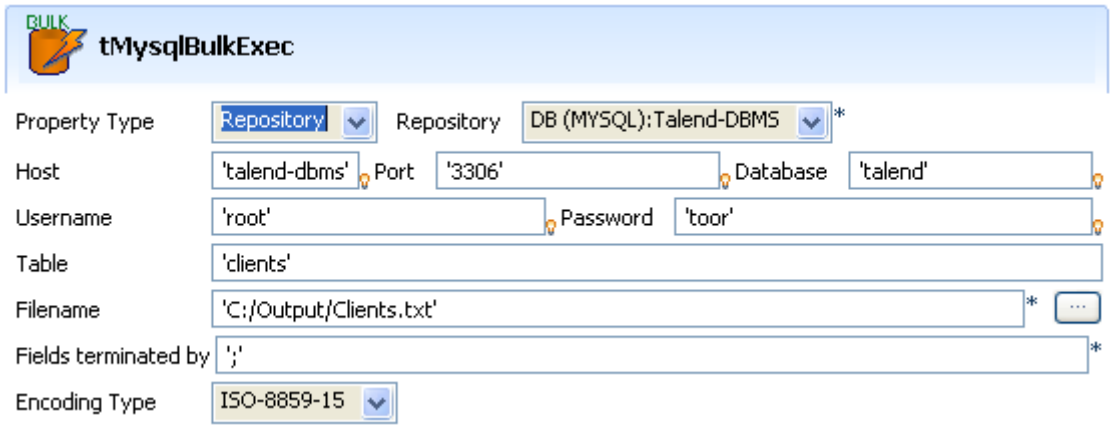


- Some schema information don't necessarily need to be displayed. To hide them away, click on **Columns** list button next to the toolbar, and uncheck the relevant entries, such as **Precision** or **Parameters**.
- Use the plus button to add as many columns to your schema definition.
- Click the Refresh button to preview the first generated row of your output.
- Then select the **tMap** component to set the transformation.
- Drag and drop all columns from the input table to the output table.

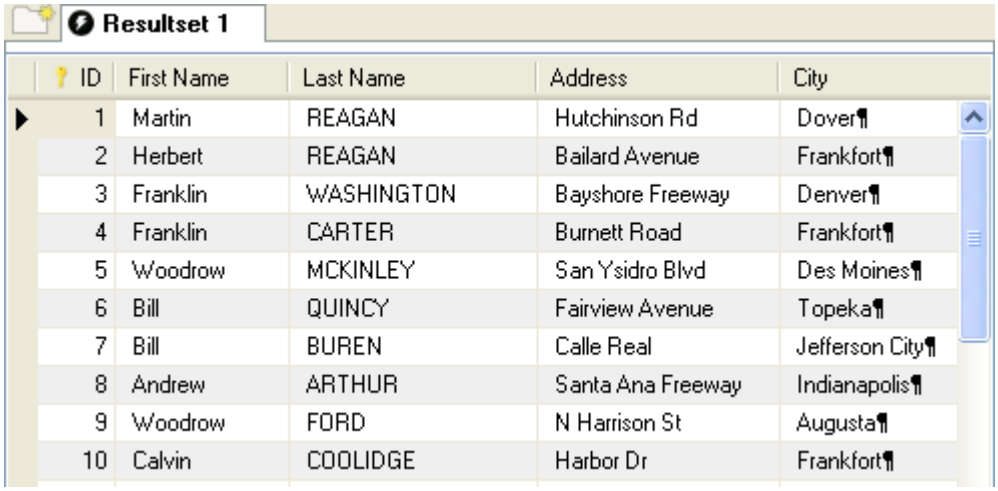


- Apply the transformation on the *LastName* column by adding uc in front of it.
- Click **OK** to validate the transformation.
- Then double-click on the **tMysqlOutputBulk** component.
- Define the name of the file to be produced in **File Name** field. If the delimited file information is stored in the **Repository**, select it in **Property type** field, to retrieve relevant data. In this use case the file name is *clients.txt*.
- The schema is propagated from the **tMap** component, if you accepted it when prompted.
- In this example, don't include the header information as the table should already contain it.

- The encoding is the default one for this use case.
- Click OK to validate the output.
- Then double-click on the **tMysqlBulkExec** to set the INSERT query to be executed.
- Define the database connection details. We recommend you to store this type of information in the **Repository**, so that you can retrieve them at any time for any job.



- Set the table to be filled in with the collected data, in the **Table** field.
- Fill in the column delimiters in the **Field terminated by** area.
- Make sure the encoding corresponds to the data encoding.
- Then press **F6** to run the job.



ID	First Name	Last Name	Address	City
1	Martin	REAGAN	Hutchinson Rd	Dover¶
2	Herbert	REAGAN	Bailard Avenue	Frankfort¶
3	Franklin	WASHINGTON	Bayshore Freeway	Denver¶
4	Franklin	CARTER	Burnett Road	Frankfort¶
5	Woodrow	MCKINLEY	San Ysidro Blvd	Des Moines¶
6	Bill	QUINCY	Fairview Avenue	Topeka¶
7	Bill	BUREN	Calle Real	Jefferson City¶
8	Andrew	ARTHUR	Santa Ana Freeway	Indianapolis¶
9	Woodrow	FORD	N Harrison St	Augusta¶
10	Calvin	COOLIDGE	Harbor Dr	Frankfort¶

The *clients* database table is filled with data from the file including upper-case *last name* as transformed in the job.

For simple Insert operations that don't include any transformation, the use of **tMysqlOutputBulkExec** allows to spare a step in the process hence to gain some performance.

Related topic: *tMysqlOutputBulkExec properties on page 296*

tMysqlOutputBulkExec

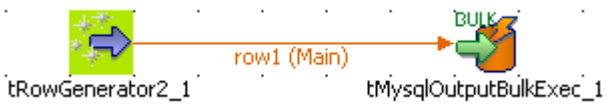
tMysqlOutputBulkExec properties

Component family	Databases/MySQL	
Function	Executes the Insert action on the data provided.	
Purpose	As a dedicated component, it allows gains in performance during Insert operations to a MySQL database.	
Basic settings	*Property type*	Either Built-in or Repository.
		Built-in: No property data stored centrally.
		Repository: Select the repository file where Properties are stored. The fields that come after are pre-filled in using the fetched data.
	Host	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database
	Username and *Password*	DB user authentication data.
In Java, use tCreateTable as substitute for this function.	*Action on table*	On the table defined, you can perform one of the following operations: **None**: No operation is carried out. **Drop and create a table**: The table is removed and created again. **Create a table**: The table does not exist and gets created. **Create a table if not exists**: The table is created if it does not exist. **Clear a table**: The table content is deleted.
	Table	Name of the table to be written. Note that only one table can be written at a time and that the table must exist for the insert operation to succeed.
	Local FileName	Name of the file to be processed. Related topic:*Defining variables from the Component view* of **Talend Open Studio** User Guide
Advanced settings	*Additional JDBC parameters*	Specify additional connection properties in the existing DB connection.
	Row separator	String (ex: “\n”on Unix) to distinguish rows.
	Field separator	Character, string or regular expression to separate fields.
	Escape char	Character of the row to be escaped
	Text enclosure	Character used to enclose text.
	Create directory if not exists	This check box is selected by default. It creates a directory to hold the output table if required.

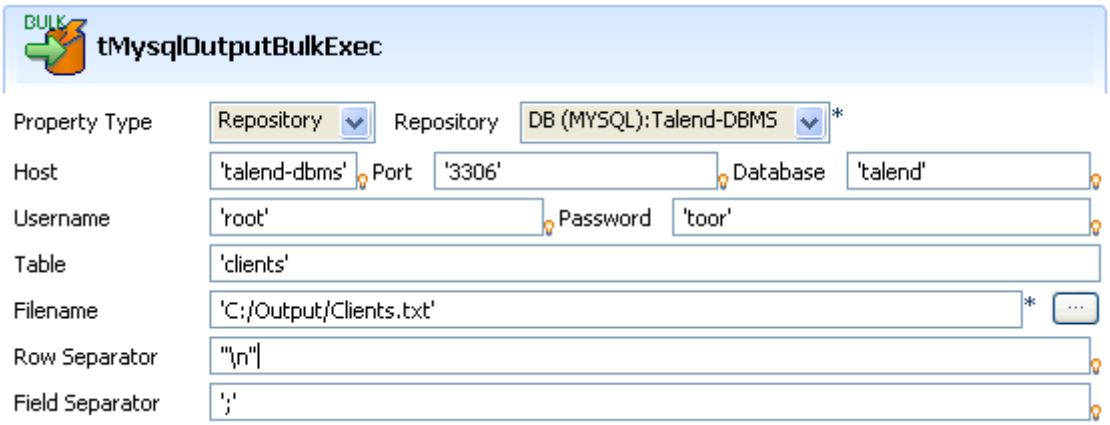
	Custom the flush	Custom the memory used to temporarily store output data.
	Action on data	On the data of the table defined, you can perform: **Insert records in table**: Add new records to the table. **Update records in table**: Make changes to existing records. **Replace records in table**: replace existing records with new one.
	Encoding Type	Select the encoding from the list or select Custom and define it manually. This field is compulsory for DB data handling.
Usage	This component is mainly used when no particular transformation is required on the data to be loaded onto the database.	
Limitation	n/a	

Scenario: Inserting data in MySQL database

This scenario describes a two-component job which carries out the same operation as the one described for *tMysqlOutputBulk properties on page 292* and *tMysqlBulkExec properties on page 262*, although no transformation of data is performed.



- Drop a **tRowGenerator** and a **tMysqlOutputBulkExec** component from the **Palette** to the design workspace.
- The tRowGenerator is to be set the same way as in the *Scenario: Inserting transformed data in MySQL database on page 293*. The schema is made of four columns including: *ID*, *First Name*, *Last Name*, *Address* and *City*.
- Then set the DB connection if needed, the best practices being to store the connection details in the Metadata repository.
- Then fill in the table to be filled in with the generated data in the **Table** field.
- And the name of the file to be loaded in **File Name** field.



Then press F6 to execute the job.

The result should be pretty much the same as in *Scenario: Inserting transformed data in MySQL database on page 293*, but the data might differ as these are regenerated randomly everytime the job is run.

tMysqlRollback

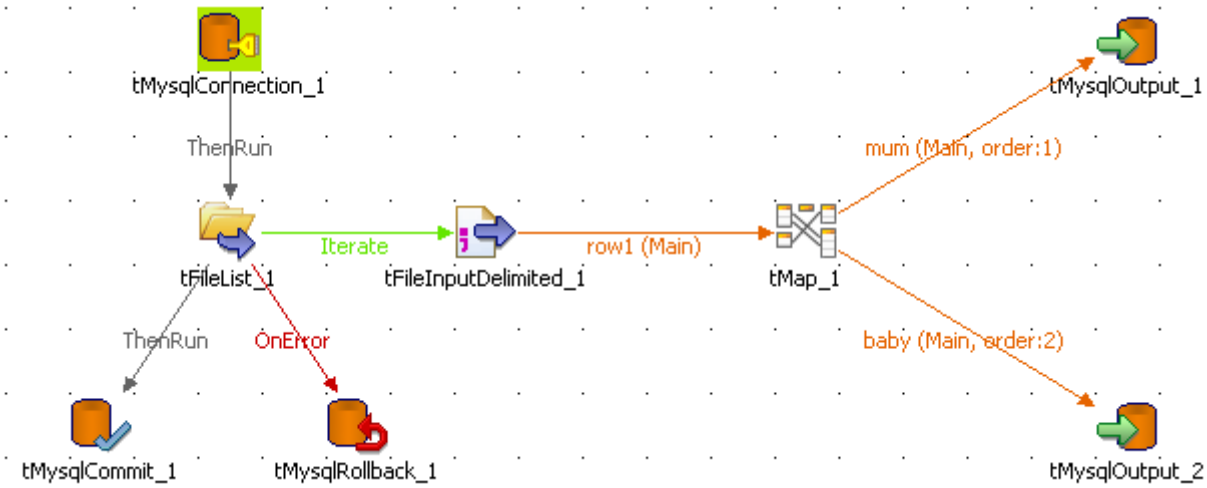
tMysqlRollback properties

This component is closely related to **tMysqlCommit** and **tMysqlConnection**. It usually doesn't make much sense to use these components independently in a transaction.

Component family	Databases	
Function	Cancel the transaction commit in the connected DB.	
Purpose	Avoids to commit part of a transaction involuntarily.	
Basic settings	*Component list*	Select the **tMysqlConnection** component in the list if more than one connection are planned for the current job.
	Close Connection	Clear this check box to continue to use the selected connection once the component has performed its task.
Usage	This component is to be used along with Mysql components, especially with **tMysqlConnection** and **tMysqlCommit** components.	
Limitation	n/a	

Scenario: Rollback from inserting data in mother/daughter tables

Based on the tMysqlConnection *Scenario: Inserting data in mother/daughter tables on page 269*, insert a rollback funtion in order to prevent unwanted commit.



- Drag and drop a **tMysqlRollback** to the design workspace and connect it to the Start component.
- Set the Rollback unique field on the relevant DB connection.

This complementary element to the job ensures that the transaction won't be partly committed.

tMysqlRow

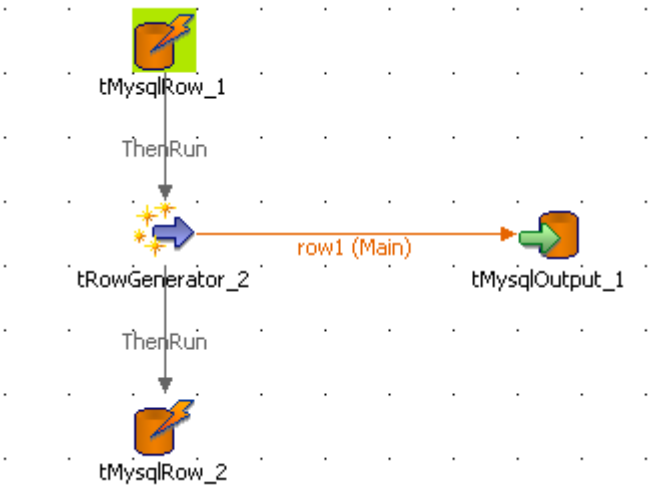
tMysqlRow properties

Component family	Databases/MySQL	
Function	**tMysqlRow** is the specific component for this database query. It executes the SQL query stated onto the specified database. The row suffix means the component implements a flow in the job design although it doesn't provide output.	
Purpose	Depending on the nature of the query and the database, **tMysqlRow** acts on the actual DB structure or on the data (although without handling data). The SQLBuilder tool helps you write easily your SQL statements.	
Basic settings	*Property type*	Either Built-in or Repository.
		Built-in: No property data stored centrally.
		Repository: Select the repository file where Properties are stored. The fields that come after are pre-filled in using the fetched data.
	Use an existing connection	Select this check box and click the relevant **tMySqlConnection** component on the Component list to reuse the connection details you already defined.
	Host	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database
	Username and *Password*	DB user authentication data.
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either built-in or remotely stored in the Repository.
		Built-in: The schema is created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	Query type	Either Built-in or Repository.
		Built-in: Fill in manually the query statement or build it graphically using SQLBuilder
		Repository: Select the relevant query stored in the Repository. The Query field gets accordingly filled in.

	Query	Enter your DB query paying particularly attention to properly sequence the fields in order to match the schema definition.
	Die on error	Clear this check box to skip the row on error and complete the process for error-free rows.
Advanced settings	*Additional JDBC parameters*	Specify additional connection properties in the existing DB connection
	Propagate QUERY's recordset	Select this check box to insert the result of the query into a COLUMN of the current flow
	Encoding Type	Select the encoding from the list or select **Custom** and define it manually. This field is compulsory for DB data handling.
	Commit every	Number of rows to be completed before committing batches of rows together into the DB. This option ensures transaction quality (but not rollback) and above all better performance on executions.
	tStateCatcher Statistics	Select this check box to collect log data at the component level.
Usage	This component offers the flexibility benefit of the DB query and covers all possibilities of SQL queries.	

Scenario: Removing and regenerating a MySQL table index

This scenario describes a four-component job that removes a table index, applies a select insert action onto a table then regenerates the index.



- Select and drop the following components onto the design workspace: **tMysqlRow** (x2), **tRowGenerator**, and **tMysqlOutput**.
- Connect **tRowGenerator** to **tMysqlIntput**.
- Using a **ThenRun** connections, link the first **tMysqlRow** to **tRowGenerator** and **tRowGenerator** to the second **tMysqlRow**.

- Select the **tMysqlRow** to fill in the DB **Basic settings**.
- In **Property type** as well in **Schema type**, select the relevant DB entry in the list.
- The DB connection details and the table schema are accordingly filled in.
- Propagate the properties and schema details onto the other components of the job.
- The query being stored in the **Metadata** area of the **Repository**, you can also select **Repository** in the **Query type** field and the relevant query entry.
- If you didn't store your query in the **Repository**, type in the following SQL statement to alter the database entries: drop index <index_name> on <table_name>
- Select the second **tMysqlRow** component, check the DB properties and schema.
- Type in the SQL statement to recreate an index on the table using the following statement: create index <index_name> on <table_name> (<column_name>)
  The **tRowGenerator** component is used to generate automatically the columns to be added to the DB output table defined.
- Select the **tMysqlOutput** component and fill in the DB connection properties either from the Repository or manually the DB connection details are specific for this use only. The table to be fed is named: *comprehensive*.
- The schema should be automatically inherited from the data flow coming from the **tLogRow**. Edit the schema to check its structure and check that it corresponds to the schema expected on the DB table specified.
- The **Action on table** is *None* and the **Action on data** is *Insert*.
- No additional Columns is required for this job.
- Press **F6** to run the job.

If you manage to watch the action on DB data, you can notice that the index is dropped at the start of the job and recreated at the end of the insert action.

Related topics: *tDBSQLRow properties on page 171*.

tMysqlSCD

tMysqlSCD belongs to two component families: Business Intelligence and Databases. For more information on it, see *tMysqlSCD on page 12*.

tMysqlSCDELT

tMysqlSCDELT Properties

Component family	Databases/MySQL	
Function	**tMysqlSCDELT** reflects and tracks changes in a dedicated MySQL SCD table.	
Purpose	**tMysqlSCD**ELT addresses Slowly Changing Dimension needs through SQL queries (server-side processing mode), and logs the changes into a dedicated MySQL SCD table.	
Basic settings	*Property type*	Either Built-in or Repository.
		Built-in: No property data stored centrally. Enter properties manually.
		Repository: Select the repository file where Properties are stored. The fields that come after are pre-filled in using the fetched data.
	Use an existing connection	Select this check box and click the relevant **tMySqlConnection** component on the Component list to reuse the connection details you already defined.
	Host	The IP address of the database server.
	Port	Listening port number of database server.
	Database	Name of the database
	Username and *Password*	User authentication data for a dedicated database.
	Source table	Name of the input MySQL SCD table.
	Table	Name of the table to be written. Note that only one table can be written at a time
	Action on table	Select to perform one of the following operations on the table defined: **None**: No action carried out on the table. **Drop and create the table**: The table is removed and created again **Create a table**: A new table gets created. **Create a table if not exists**: A table gets created if it does not exist. **Clear a table**: The table content is deleted. You have the possibility to rollback the operation. **Truncate a table**: The table content is deleted. You don not have the possibility to rollback the operation.
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either built-in or remotely stored in the Repository.

		Built-in: The schema is created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	Surrogate Key	Select the surrogate key column from the list.
	Creation	Select the method to be used for the surrogate key generation.
	Source Keys	Select one or more columns to be used as keys, to ensure the unicity of incoming data.
	Use SCD Type 1 fields	Use type 1 if tracking changes is not necessary. SCD Type 1 should be used for typos corrections for example. Select the columns of the schema that will be checked for changes.
	Use SCD Type 2 fields	Use type 2 if changes need to be tracked down. SCD Type 2 should be used to trace updates for example. Select the columns of the schema that will be checked for changes. **Start date**: Adds a column to your SCD schema to hold the strat date value. You can select one of the input schema columns as Start Date in the SCD table. **End Date**: Adds a column to your SCD schema to hold the end date value for the record. When the record is currently active, the End Date column shows a null value, or you can select **Fixed Year value** and fill it in with a fictive year to avoid having a null value in the End Date field. **Log Active Status**: Adds a column to your SCD schema to hold the **true** or **false** status value. This column helps to easily spot the active record. **Log versions**: Adds a column to your SCD schema to hold the version number of the record.
Usage	This component is used as an output component. It requires an input component and Row main link as input.	

Related Scenario

For related topics, see *tMysqlSCD on page 12 Scenario: Tracking changes using Slowly Changing Dimensions (type 0 through type 3) on page 15*.

tMysqlSP

tMysqlSP Properties

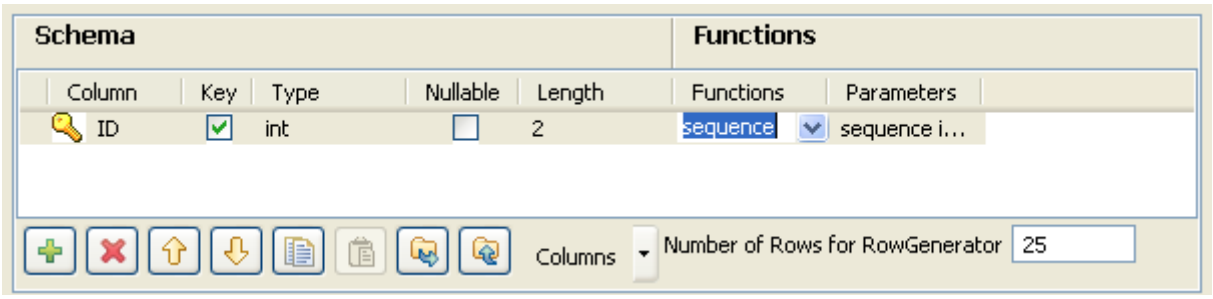
Component family	Databases/Mysql	
Function	**tMysqlSP** calls the database stored procedure.	
Purpose	**tMysqlSP** offers a convenient way to centralize multiple or complex queries in a database and call them easily.	
Basic settings	*Property type*	Either Built-in or Repository.
		Built-in: No property data stored centrally.
		Repository: Select the repository file where Properties are stored. The fields that come after are pre-filled in using the fetched data.
	Host	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database
	Username and *Password*	DB user authentication data.
	Encoding Type	Select the encoding from the list or select Custom and define it manually. This field is compulsory for DB data handling.
	Schema type and *Edit Schema*	In SP principle, the schema is an input parameter. A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either built-in or remotely stored in the Repository.
		Built-in: The schema is created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	SP Name	Type in the exact name of the Stored Procedure
	Is Function / Return result in	Select this check box, if a value only is to be returned. Select on the list the schema column, the value to be returned is based on.

	Parameters	Click the Plus button and select the various **Schema Columns** that will be required by the procedures. Note that the SP schema can hold more columns than there are paramaters used in the procedure. Select the **Type** of parameter: **IN**: Input parameter **OUT**: Output parameter/return value **IN OUT**: Input parameters is to be returned as value, likely after modification through the procedure (function). **RECORDSET**: Input parameters is to be returned as a set of values, rather than single value. Check the *tParseRecordSet* component if you want to analyze a set of records from a database table or DB query and return single records.
Usage	This component is used as intermediary component. It can be used as start component but only input parameters are thus allowed.	
Limitation	The Stored Procedures syntax should match the Database syntax.	

Scenario: Finding a State Label using a stored procedure

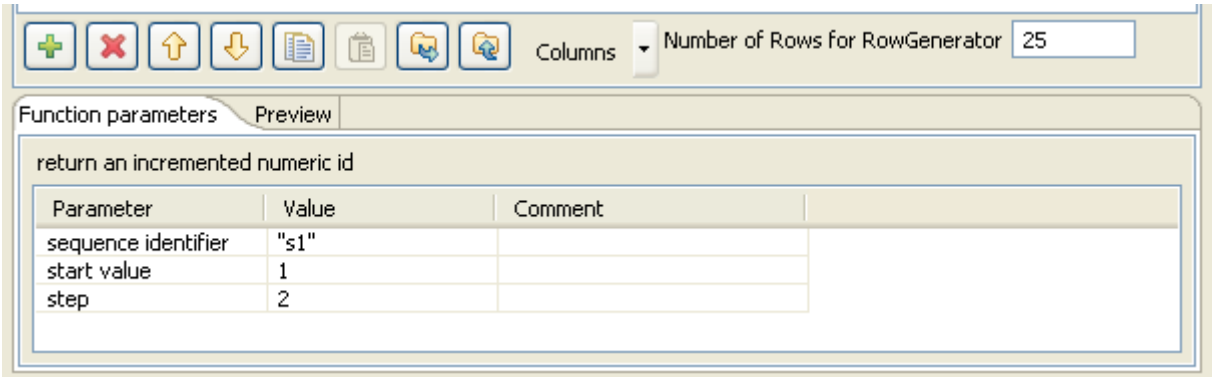
The following job aims at finding the State labels matching the odd State IDs in a Mysql two-column table. A stored procedure is used to carry out this operation.

- Drag and drop the following components used in this example: **tRowGenerator**, **tMysqlSP**, **tLogRow**.
- Connect the components using the **Row Main** link.
- The **tRowGenerator** is used to generate the odd id number. Double-click on the component to launch the editor.

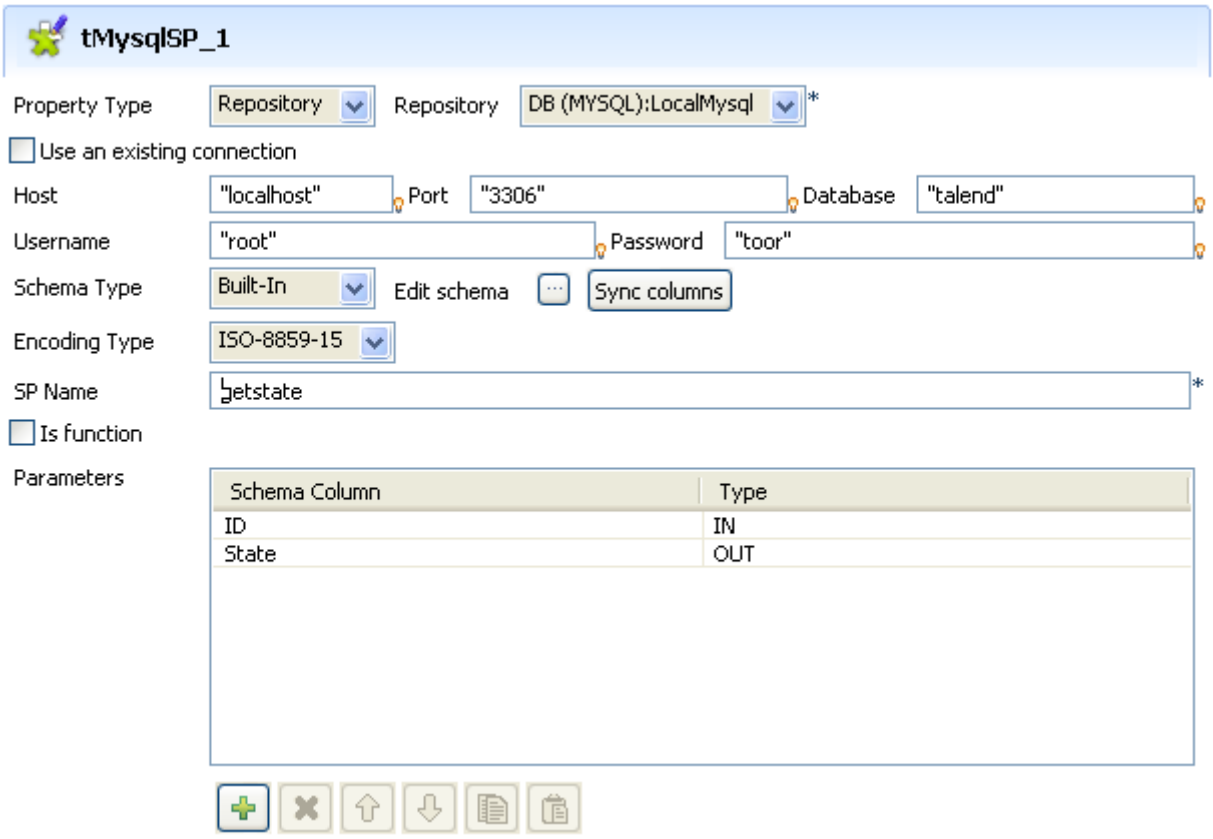


- Click on the **Plus** button to add a column to the schema to generate.
- Select the **Key** check box and define the **Type** to **Int**.
- The **Length** equals to 2 digits max.

- Use the preset function called **sequence** but customize the Parameters in the lower part of the window.



- Change the **Value** of **step** from *1* to *2* for this example, still starting from 1.
- Set the **Number of generated rows** to 25 in order for all the odd State id (of 50 states) to be generated.
- Click **OK** to validate the configuration.
- Then select the **tMysqlSP** component and define its properties.

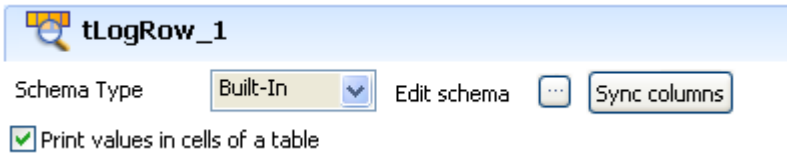


- Set the **Property type** field to **Repository** and select the relevant entry on the list. The connection details get filled in automatically.
- Else, set manually the connection information.
- Click **Sync Column** to retrieve the generated schema from the preceding component.

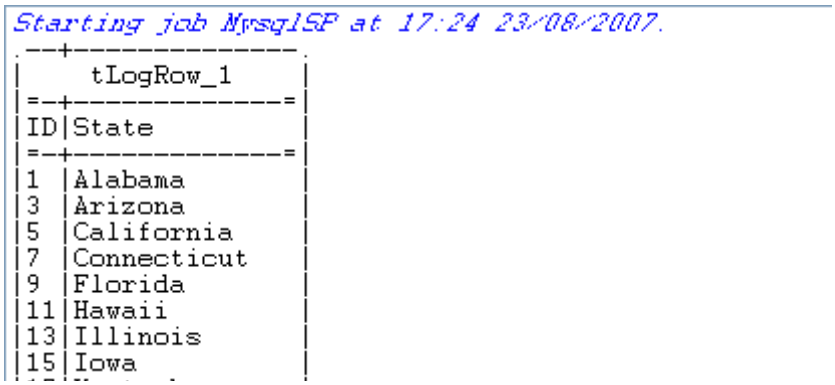
- Then click **Edit Schema** and add an extra column to hold the State Label to be output, in addition to the ID.
- Select the encoding type on the list.
- Type in the name of the procedure in the **SP Name** field as it is called in the Database. In this example, *getstate*. The procedure to be executed states as follows:

```
DROP PROCEDURE IF EXISTS `talend`.`getstate` $$
CREATE DEFINER=`root`@`localhost` PROCEDURE `getstate`(IN pid
INT, OUT pstate VARCHAR(50))
BEGIN
 SELECT LabelState INTO pstate FROM us_states WHERE idState =
pid;
END $$
```

- In the **Parameters** area, click the plus button to add a line to the table.
- Set the **Column** field to *ID*, and the **Type** field to *IN* as it will be given as input parameter to the procedure.
- Add a second line and set the **Column** field to *State* and the **Type** to *Out* as this is the output parameter to be returned.
- Eventually, set the **tLogRow** component properties.



- Synchronize the schema with the preceding component.
- And select the **Print values in cells of a table** check box for reading convenience.
- Then save your Job and execute it.



The output shows the state labels corresponding to the odd state ids as defined in the procedure.

Check the *tParseRecordSet* component if you want to analyze a set of records from a database table or DB query and return single records.

tMysqlTableList

tMysqlTableList Properties

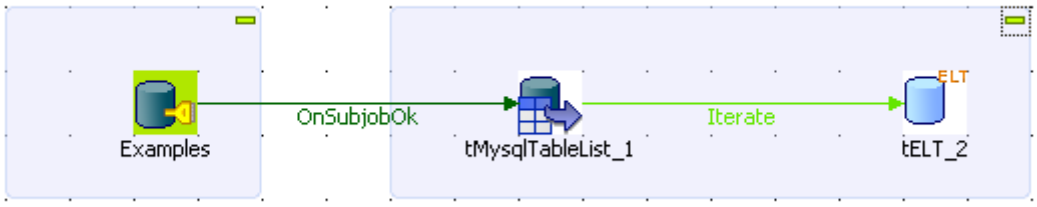
Component family	Databases/MySQL	
Function	Iterates on a set of table names through a defined Mysql connection.	
Purpose	Lists the names of a given set of Mysql tables using a select statement based on a Where clause.	
Basic settings	*Component list*	Select the **tMysqlConnection** component in the list if more than one connection are planned for the current job.
	Where clause for table name selection	Enter the Where clause to identify the tables to iterate on.
Usage	This component is to be used along with Mysql components, especially with **tMysqlConnection**.	
Limitation	n/a	

Scenario: Iterating on DB tables and deleting their content using a user-defined SQL template

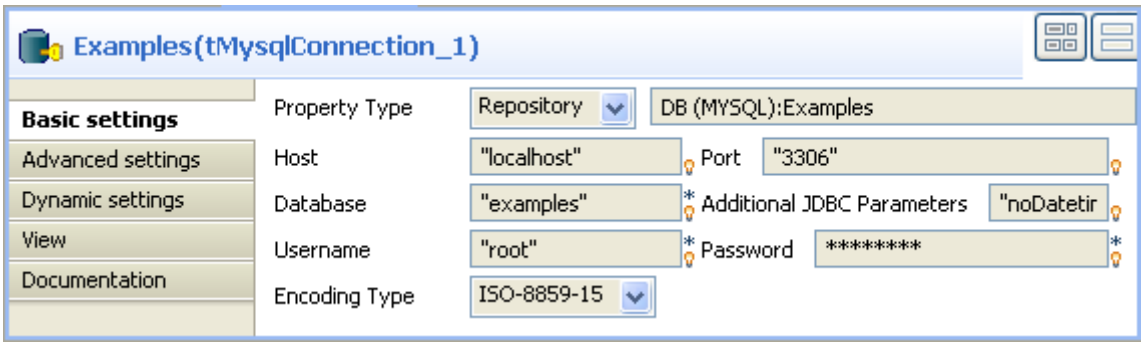
The following Java scenario creates a three-component job that iterates on given table names from a MySQL database using a WHERE clause. It then deletes the content of the tables directly on the DBMS using a user-defined SQL template.

For advanced use, start with creating a connection to the database that contains the tables you want to empty of their content.

- In the **Repository** tree view, expand **Metadata** and right click **DB Connections** to create a connection to the relevant database and to store the connection information locally.
  For more information about Metadata, see *Defining Metadata items* of **Talend Open Studio** User Guide.
  Otherwise, drop a **tMySQLConnection** component in the design workspace and fill the connection details manually.
- Drop the database connection you created from the **Repository** onto the design workspace. The **[Components]** dialog box displays.
- Select **tMySQLConnection** and click **OK**.
  The **tMySQLConnection** components displays on the design workspace with all connection details automatically filled in its **Basic settings** view.
- Drop the following two components from the **Palette** onto the design workspace: **tMysqlTableList** and **tELT**.
- Connect **tMysqlConnection** to **tMysqlTableList** using an **OnSubjobOk** link.

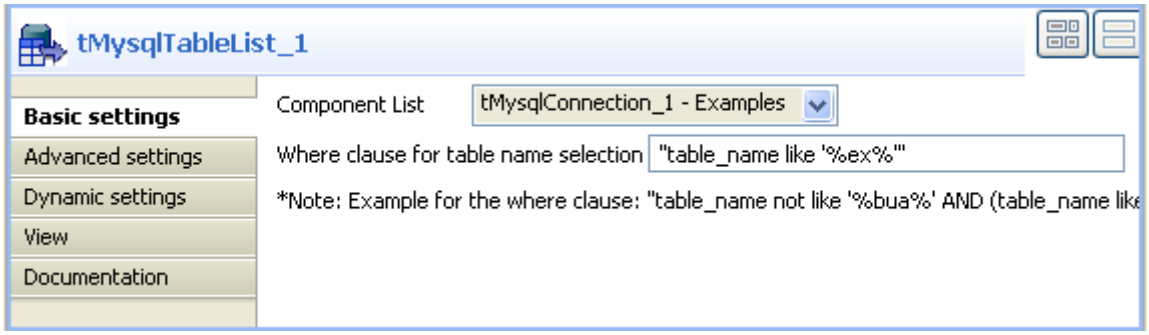


- Connect **tMysqlTableList** to **tELT** using an **Iterate** link.
- If needed, double-click **tMysqlConnection** to display its **Basic settings** view and verify the connection details.

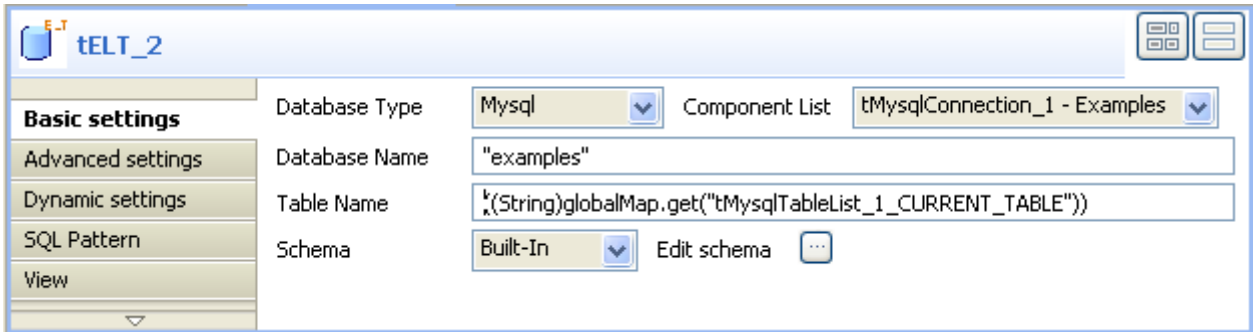


In this example, we want to connect to a MySQL database called *examples*.

- In the design workspace, double-click **tMysqlTableList** to display its **Basic settings** view and define its settings.



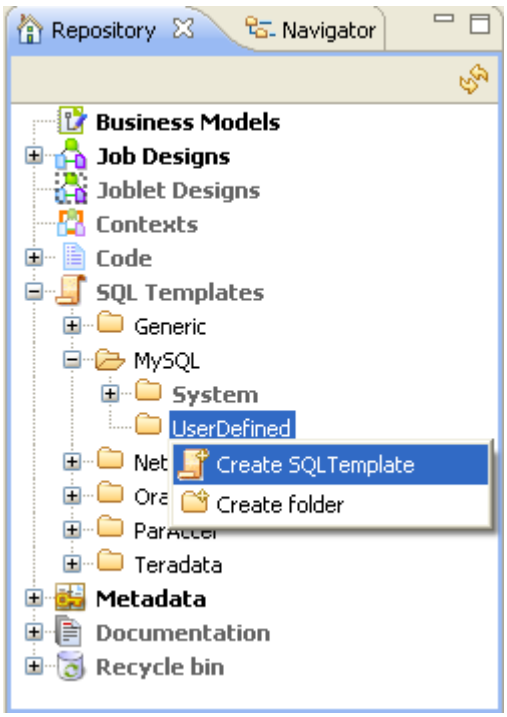
- On the **Component list**, select the relevant MySQL connection component if more than one connection is used.
- Enter a WHERE clause using the right syntax in the corresponding field to iterate on the table name(s) you want to delete the content of.
  In this scenario, we want the job to iterate on all the tables which names start with “ex”.
- In the design workspace, double-click **tELT** to display its **Basic settings** view and define its settings.



- In **Database Name**, enter the name of the database containing the tables you want to process.
- On the **Component list**, select the relevant MySQL connection component if more than one connection is used.
- Click in the **Table name** field and press **Ctrl+Space** to access the global variable list.
- From the global variable list, select *((String)globalMap.get("tMysqlTableList_1_CURRENT_TABLE"))*.

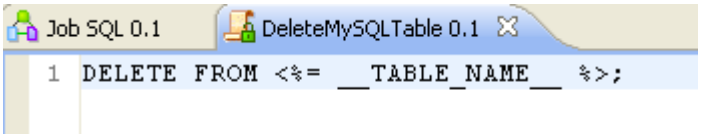
To create the user-defined SQL template:

- In the **Repository** tree view, expand **SQL Templates** and **MySQL** in succession.

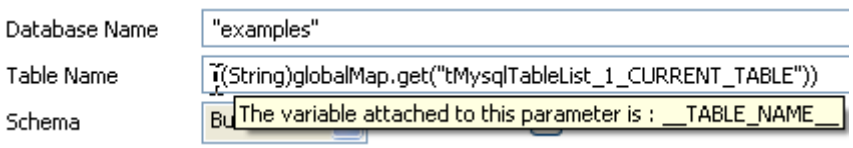


- Right-click **UserDefined** and select **Create SQLTemplate** from the drop-down list. The **New SQLTemplate** wizard opens.

- Enter a name for the new SQL template and fill in the other fields If needed and then click **Finish** to close the wizard.
  An SQL pattern editor opens on the design workspace.
- Delete the existing code and enter the code necessary to carry out the desired action, deleting the content of all tables which names start with “ex” in this example.



In the SQL template code, you must use the correct variable name attached to the table name parameter (“__TABLE-NAME__” in this example).
To display the variable name used, put your pointer in the **Table Name** field in the basic settings of the **tELT** component.

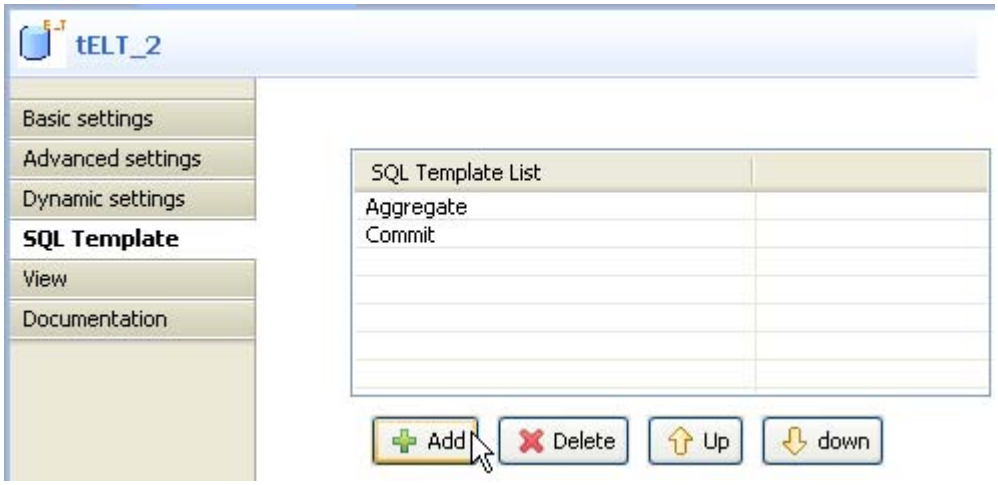


- Press **Ctrl+S** to save the new user-defined SQL template.

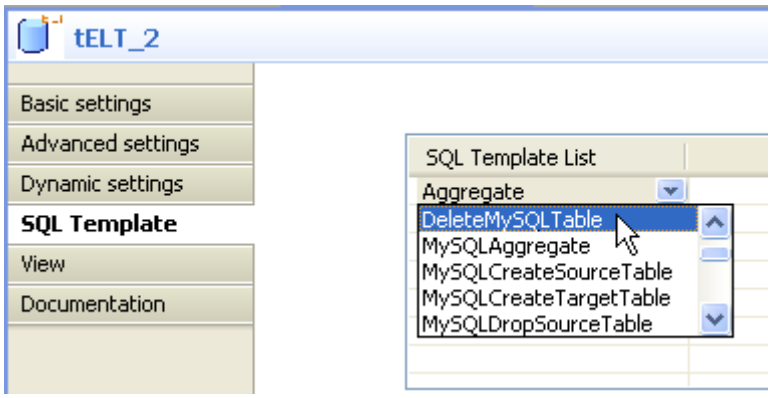
The next step is to add the new user-defined SQL template to the SQL template list in the **tELT** component.

To add the user-defined SQL template to the SQL template list:

- In the **Component** view of **tELT**, click the **SQL Templates** tab to display the **SQLTemplate List**.



- Click the **Add** button and add two SQL template lines.
- Click in the first line to display a drop-down arrow and then click the arrow to display the SQL template list.



- Select in the list the user-defined SQL template you already created.
- Make sure that the SQL template in the second line is **Commit**.
- Save your job and press **F6** to execute it.

All tables in the MySQL `examples` database which names begin with “ex” are emptied from their content.

Related scenario

For **tMysqlTableList** related scenario, see *Scenario: Iterating on a DB table and listing its column names on page 264*.

tNetezzaBulkExec

tNetezzaBulkExec properties

Component family	Databases/Netezza	
Function	Executes the Insert action on the data provided.	
Purpose	As a dedicated component, **tNetezzaBulkExec** offers gains in performance while carrying out the Insert operations to a Netezza database	
Basic settings	*Property type*	Either Built-in or Repository.
		Built-in: No property data stored centrally.
		Repository: Select the Repository file where Properties are stored. When selected, the fields to follow are pre-filled in using fetched data.
	Use an existing connection	Select this check box when you are using the component **tNetezzaConnection**.
	Host	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database.
	Username and *Password*	DB user authentication data.
	Table	Name of the table to be written. Note that only one table can be written at a time and that the table must exist for the insert operation to succeed.
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either built-in or remotely stored in the Repository.
		Built-in: The schema is created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	File Name	Name of the file to be processed. Related topic:*Defining variables from the Component view* of **Talend Open Studio** User Guide.
Advanced settings	*Field Separator*	Character, string or regular expression to separate fields.
	Require quotes (“) around data files	Select this check box to use data enclosure characters.
	Row Separator	String (ex: “\n”on Unix) to distinguish rows.
	Escape character	Character of the row to be escaped.

	Date format / Date delimiter	Use Date format to distinguish the way years, months and days are represented in a string. Use Date delimiter to specify the separator between date values.
	Time format/ Time delimiter	Use Time format to distinguish the time is represented in a string. Use Time delimiter to specify the separator between time values.
	Encoding	Select the encoding from the list or select Custom and define it manually. This field is compulsory for DB data handling.
	Max Errors	Enter the maximum error limit that will not stop the process.
	Skip Rows	Enter the number of rows to be skipped.
	tStatCatcher Statistics	Select this check box to collect log data at the component level.
Usage	This component offers performance and flexibility of Netezza DB query handling.	
Limitation	n/a	

Related scenarios

For uses cases in relation with **tNetezzaBulkExec**, see the following scenarios:

- **tMysqlOutputBulk** *Scenario: Inserting transformed data in MySQL database on page 293*.
- **tMysqlOutputBulkExec** *Scenario: Inserting data in MySQL database on page 297*.
- **tOracleBulkExec** *Scenario: Truncating and inserting file data into Oracle DB on page 329*.

tNetezzaCommit

tNetezzaCommit Properties

This component is closely related to **tNetezzaConnection** and **tNetezzaRollback**. It usually does not make much sense to use these components independently in a transaction.

Component family	Databases/Netezza	
Function	**tNetezzaCommit** validates the data processed through the Job into the connected DB	
Purpose	Using a unique connection, this component commits in one go a global transaction instead of doing that on every row or every batch and thus provides gain in performance.	
Basic settings	*Component list*	Select the **tNetezzaConnection** component in the list if more than one connection are planned for the current Job.
	Close Connection	Clear this check box to continue to use the selected connection once the component has performed its task.
Usage	This component is to be used along with Netezza components, especially with **tNetezzaConnection** and **tNetezzaRollback**.	
Limitation	n/a	

Related scenario

This component is closely related to **tNetezzaConnection** and **tNetezzaRollback**. It usually does not make much sense to use one of these without using a **tNetezzaConnection** component to open a connection for the current transaction.

For **tNetezzaCommit** related scenario, see *Scenario: Inserting data in mother/daughter tables on page 269*.

tNetezzaConnection

tNetezzaConnection Properties

This component is closely related to **tNetezzaCommit** and **tNetezzaRollback**. It usually does not make much sense to use one of these without using a **tNetezzaConnection** component to open a connection for the current transaction.

Component family	Databases/Netezza	
Function	**tNetezzaConnection** opens a connection to the database for a current transaction.	
Purpose	This component allows to commit job data in one go to the output database as one transaction when validated.	
Basic settings	*Property type*	Either Built-in or Repository.
		Built-in: No property data stored centrally.
		Repository: Select the repository file where Properties are stored. The fields that come after are pre-filled in using the fetched data.
	Host	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database
	Username and *Password*	DB user authentication data.
	Additional JDBC Parameters	Specify additional connection properties in the existing DB connection.
	Encoding Type	Select the encoding from the list or select Custom and define it manually. This field is compulsory for DB data handling.
	Use or register a shared DB Connection	Select this check box to share your connection or fetch a connection shared by a parent or child Job. **Shared DB Connection Name:** set or type in the shared connection name.
Usage	This component is to be used along with Netezza components, especially with **tNetezzaCommit** and **tNetezzaRollback**.	
Limitation	n/a	

Related scenarios

For **tNetezzaConnection** related scenario, see *Scenario: Inserting data in mother/daughter tables on page 269*.

tNetezzaInput

tNetezzaInput properties

Component family	Databases/Netezza	 
Function	**tNetezzaInput** reads a database and extracts fields based on a query.	
Purpose	**tNetezzaInput** executes a DB query with a strictly defined order which must correspond to the schema definition. Then it passes on the field list to the next component via a Main row link.	
Basic settings	*Property type*	Either Built-in or Repository.
		Built-in: No property data stored centrally.
		Repository: Select the repository file where Properties are stored. The fields that come after are pre-filled in using the fetched data.
		Click this icon to open a database connection wizard and store the database connection parameters you set in the component **Basic settings** view. For more information about setting up and storing database connection parameters, see *Setting up a DB connection* of **Talend Open Studio** User Guide.
	Use existing connection	Select this check box when using a **tNetezzaConnection** component.
	Host	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database
	Username and *Password*	DB user authentication data.
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either built-in or remotely stored in the Repository.
		Built-in: The schema is created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	Table Name	Name of the table to be read.
	Query type and *Query*	Enter your DB query paying particularly attention to properly sequence the fields in order to match the schema definition.

Advanced settings	*Encoding Type*	Select the encoding type from the list or select **Custom** and define it manually. This field is compulsory for DB data handling.
	Use cursor	When selected, helps to decide the row set to work with at a time and thus optimize performance.
	Trim all the String/Char columns	Select this check box to remove leading and trailing whitespace from all the String/Char columns.
	Trim column	Remove leading and trailing whitespace from defined columns.
	tStatCatcher Statistics	Select this check box to collect log data at the component level.
Usage	This component covers all possibilities of SQL queries onto a Netezza database.	

Related scenarios

Related scenarios for **tNetezzaInput** are:

- *Scenario 1: Displaying selected data from DB table on page 163.*
- *Scenario 2: Using StoreSQLQuery variable on page 164.*

Related topic in **tContextLoad** *Scenario: Dynamic context use in MySQL DB insert on page 652.*

tNetezzaOutput

tNetezzaOutput properties

Component family	Databases/Netezza	
Function	**tNetezzaOutput** writes, updates, makes changes or suppresses entries in a database.	
Purpose	**tNetezzaOutput** executes the action defined on the table and/or on the data contained in the table, based on the flow incoming from the preceding component in the designed Job.	
Basic settings	*Property type*	Either Built-in or Repository.
		Built-in: No property data stored centrally.
		Repository: Select the repository file where Properties are stored. The fields that come after are pre-filled in using the fetched data.
		Click this icon to open a database connection wizard and store the database connection parameters you set in the component **Basic settings** view. For more information about setting up and storing database connection parameters, see *Setting up a DB connection* of **Talend Open Studio** User Guide.
	Use existing connection	Select this check box when using a **tNetezzaConnection** component.
	Host	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database
	Username and *Password*	DB user authentication data.
	Table	Name of the table to be written. Note that only one table can be written at a time
	Action on table	On the table defined, you can perform one of the following operations: **None**: No operation is carried out. **Drop and create a table**: The table is removed and created again. **Create a table**: The table does not exist and gets created. **Create a table if not exists**: The table is created if it does not exist. **Drop a table if exists and create**: The table is removed if it already exists and created again. **Clear a table**: The table content is deleted.

	Action on data	On the data of the table defined, you can perform: **Insert**: Add new entries to the table. If duplicates are found, job stops. **Update**: Make changes to existing entries **Insert or update**: Add entries or update existing ones. **Update or insert**: Update existing entries or create it if non existing **Delete**: Remove entries corresponding to the input flow. *It is necessary to specify at least one column as a primary key on which the Update and Delete operations are based. You can do that by clicking Edit Schema and selecting the check box(es) next to the column(s) you want to set as primary key(s). For an advanced use, click the Advanced settings view where you can simultaneously define primary keys for the Update and Delete operations. To do that: Select the Use field options check box and then in the Key in update column, select the check boxes next to the column names you want to use as a base for the Update operation. Do the same in the Key in delete column for the Delete operation.*
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either built-in or remotely stored in the Repository.
		Built-in: The schema is created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	Die on error	Clear this check box to skip the row on error and complete the process for error-free rows.
Advanced settings	*Additional JDBC parameters*	Specify additional connection properties in the existing DB connection.
	Extend Insert	Select this check box to carry out a bulk insert of a definable set of lines instead of inserting lines one by one. The gain in system performance is huge. **Number of rows per insert**: enter the number of rows to be inserted as one block. Note that too high value decreases performance due to memory issues.
	Encoding Type	Select the encoding type from the list or select **Custom** and define it manually. This field is compulsory for DB data handling.

	Commit every	Number of rows to be completed before committing batches of rows together into the DB. This option ensures transaction quality (but not rollback) and, above all, better performance at executions.
	Additional Columns	This option is not offered if you create (with or without drop) the DB table. This option allows you to call SQL functions to perform actions on columns, which are not insert, nor update or delete actions, or action that require particular preprocessing.
		Name: Type in the name of the schema column to be altered or inserted as new column
		SQL expression: Type in the SQL statement to be executed in order to alter or insert the relevant column data.
		Position: Select **Before**, **Replace** or **After** following the action to be performed on the reference column.
		Reference column: Type in a column of reference that the **tDBOutput** can use to place or replace the new or altered column.
	Use field options	Select this check box to customize a request, especially when there is double action on data.
	tStateCatcher Statistics	Select this check box to collect log data at the component level.
Usage	This component offers the flexibility benefit of the DB query and covers all possibilities of SQL queries.	

Related scenarios

For **tNetezzaOutput** related topics, see:

- **tDBOutput** *Scenario: Displaying DB output on page 168*
- **tMySQLOutput** *Scenario 1: Adding a new column and altering data in a DB table on page 283*.

tNetezzaRollback

tNetezzaRollback properties

This component is closely related to **tNetezzaCommit** and **tNetezzaConnection**. It usually does not make much sense to use these components independently in a transaction.

Component family	Databases/Netezza	
Function	**tNetezzaRollback** cancels the transaction committed in the connected DB.	
Purpose	This component avoids to commit part of a transaction involuntarily.	
Basic settings	*Component list*	Select the **tNetezzaConnection** component in the list if more than one connection are planned for the current job.
	Close Connection	Clear this check box to continue to use the selected connection once the component has performed its task.
Usage	This component is to be used along with Netezza components, especially with **tNetezzaConnection** and **tNetezzaCommit**.	
Limitation	n/a	

Related scenarios

For **tNetezzaRollback** related scenario, see *Scenario: Rollback from inserting data in mother/daughter tables on page 299*.

tNetezzaRow

tNetezzaRow properties

Component family	Databases/Netezza	
Function	**tNetezzaRow** is the specific component for this database query. It executes the SQL query stated onto the specified database. The row suffix means that the component implements a flow in the job design although it does not provide output.	
Purpose	Depending on the nature of the query and the database, **tNetezzaRow** acts on the actual DB structure or on the data (although without handling data). The SQLBuilder tool helps you write easily your SQL statements.	
Basic settings	*Property type*	Either Built-in or Repository.
		Built-in: No property data stored centrally.
		Repository: Select the repository file where Properties are stored. The fields that come after are pre-filled in using the fetched data.
	Use an existing connection	Select this check box and click the relevant **tNetezzaConnection** component on the **Component list** to reuse the connection details you already defined.
	Host	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database
	Username and *Password*	DB user authentication data.
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either built-in or remotely stored in the Repository.
		Built-in: The schema is created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	Table Name	Enter the name of the table to be processed.
	Query type	Either Built-in or Repository.
		Built-in: Fill in manually the query statement or build it graphically using SQLBuilder
		Repository: Select the relevant query stored in the Repository. The Query field gets accordingly filled in.

	Query	Enter your DB query paying particularly attention to properly sequence the fields in order to match the schema definition.
	Die on error	Clear this check box to skip the row on error and complete the process for error-free rows.
Advanced settings	*Additional JDBC parameters*	Specify additional connection properties in the existing DB connection
	Propagate QUERY's recordset	Select this check box to insert the result of the query into a COLUMN of the current flow
	Encoding Type	Select the encoding from the list or select **Custom** and define it manually. This field is compulsory for DB data handling.
	Commit every	Number of rows to be completed before committing batches of rows together into the DB. This option ensures transaction quality (but not rollback) and above all better performance on executions.
	*tState*Catcher Statistics	Select this check box to collect log data at the component level.
Usage	This component offers the flexibility benefit of the DB query and covers all possibilities of SQL queries.	

Related scenarios

For a **tNetezzaRow** related scenario, see *Scenario: Removing and regenerating a MySQL table index on page 301*.

tOracleBulkExec

tOracleBulkExec properties

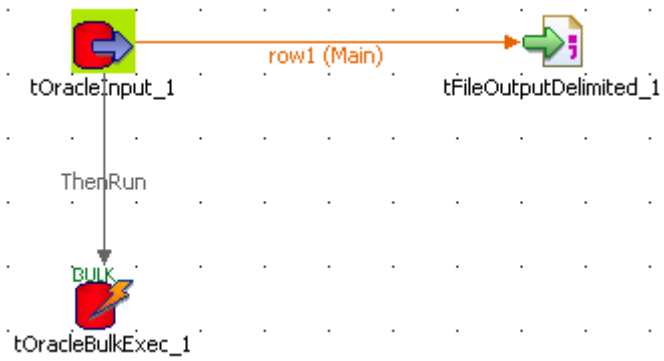
Component family	Databases/Oracle	
Function	**tOracleBulkExec** inserts, appends, replaces or truncate data in an Oracle database.	
Purpose	As a dedicated component, it allows gains in performance during operations performed on data of an Oracle database.	
Basic settings	*Property type*	Either Built-in or Repository.
		Built-in: No property data stored centrally.
		Repository: Select the repository file where Properties are stored. The fields that come after are pre-filled in using the fetched data.
	Use an existing connection	Select this check box when you are using the component **tOracleConnection**.
	Connection type	Drop-down list of available drivers
	DB Version	Select the Oracle version in use
	Host	IP address of the database server
	Port	Port number listening the database server
	Database	Database name.
	Schema	Schema name.
Perl only	*Service Name*	Oracle Service Name or SID in Oracle database. In Java projects, the the full database connection details are required.
	Username and *Password*	DB user authentication data.
	Table	Name of the table to be written. Note that only one table can be written at a time
	Action on table	On the table defined, you can perform one of the following operations: **None**: No operation is carried out. **Drop and create a table**: The table is removed and created again. **Create a table**: The table does not exist and gets created. **Create a table if doesn't exist**: The table is created if it does not exist. **Clear a table**: The table content is deleted. **Truncate table**: The table content is deleted. You do not have the possibility to rollback the operation.
	Data file name	Name of the file to be processed. Related topic:*Defining variables from the Component view* of **Talend Open Studio** User Guide.

	Action on data	On the data of the table defined, you can perform: **Insert**: Inserts rows to an empty table. If duplicates are found, job stops. **Update**: Update the existing data of the table. **Append**: Adds rows to the existing data of the table **Replace**: Overwrites some rows of the table **Truncate**: Drops table entries and inserts new input flow data.
	Schema type and *Edit schema*	In Stored Procedure principle, the schema is an input parameter. A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either built-in or remotely stored in the Repository.
Advanced settings	*Advanced separator (for number)*	Select this check box to change the separator used for the numbers.
	Use existing control file	Select this check box if you use a control file (.ctl) and specify its path in the **.ctl file name** field.
	Record format	Define the record format: **Default**: format parameters are set by default. **Stream**: set **Record terminator**. **Fixed**: set the **Record length**. **Variable**: set the **Field size of the record length**.
	Specify .ctl file's INTO TABLE clause manually	Select this check box to manually fill in the **INTO TABLE clause** of the control file.
	Fields terminated by	Character, string or regular expression to separate fields: **None**: no separator is used. **Whitespace**: the separator used is a space. **EOF (used for loading LOBs from lobfile)**: the separator used is an EOF character (End Of File). **Other terminator**: Set another terminator in the **Field terminator** field.
	Use fields enclosure	Select this check box if you want to use enclosing characters for the text: **Fields enclosure (left part)**: character delimiting the left of the field. **Field enclosure (right part)**: character delimiting the right of the field.
	Use schema's Date Pattern to load Date field	Select this check box to use the date pattern of the schema in the date field.
	Specify field condition	Select this check box to define data loading condition.
	Preserve blanks	Select this check box to preserve the blanks.
	Trailing null columns	Select this check box to load null columns.
	Load options	Define load options: **Parameter** : select one of the loading parameters. **Value** : type the value of the selected parameter.

	NLS Language	In the list, select the language used for the data that are not used in Unicode.
	Set Parameter NLS_TERRITORY	Select this check box to modify the territory conventions used for day and weeks numbering. Your OS value is the default value used.
	Encoding Type	Select the encoding type from the list or select **Custom** and define it manually. This field is compulsory for database data handling.
	Output	Select the type of output for the standard output of the Oracle database: **to console**, **to global variable**.
	Convert columns and table names to uppercase	Select this check box to uppercase the names of the columns and the name of the table.
	tStatCatcher Statistics	Select this check box to collect log data at the component level.
Perl only	*Fields terminated by*	Character, string or regular expression to separate fields.
Perl only	*Fields optionnally enclosed by*	Data enclosure characters.
Perl only	*Encoding*	Select the encoding from the list or select Custom and define it manually. This field is compulsory for DB data handling.
Usage	This dedicated component offers performance and flexibility of Oracle DB query handling.	

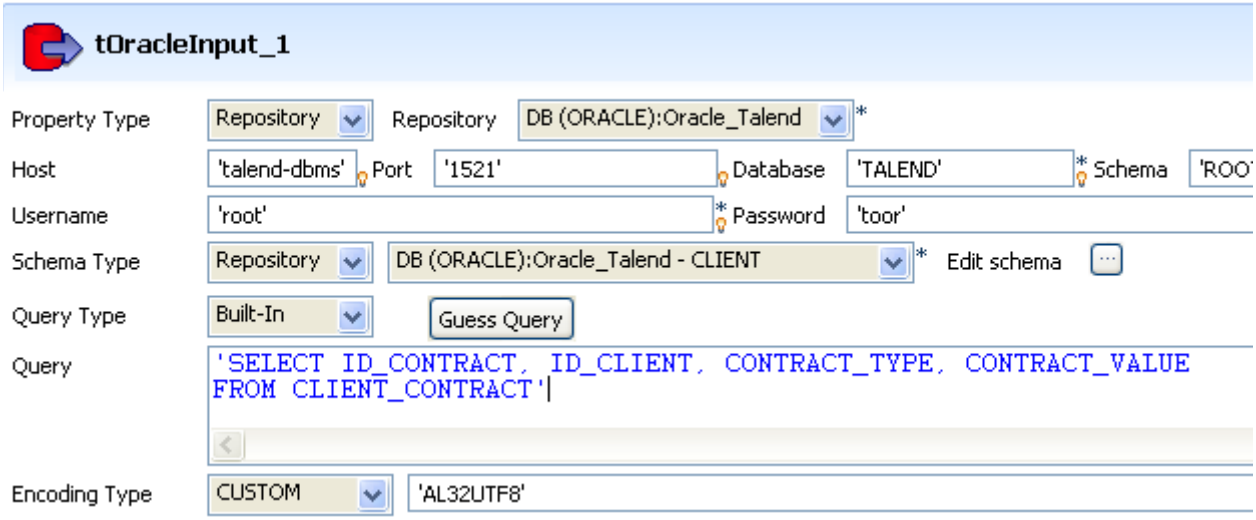
Scenario: Truncating and inserting file data into Oracle DB

This scenario describes how to truncate the content of an Oracle DB and load an input file content. The related job is composed of three components that respectively creates the content, output this content into a file to be loaded onto the Oracle database after the DB table has been truncated.



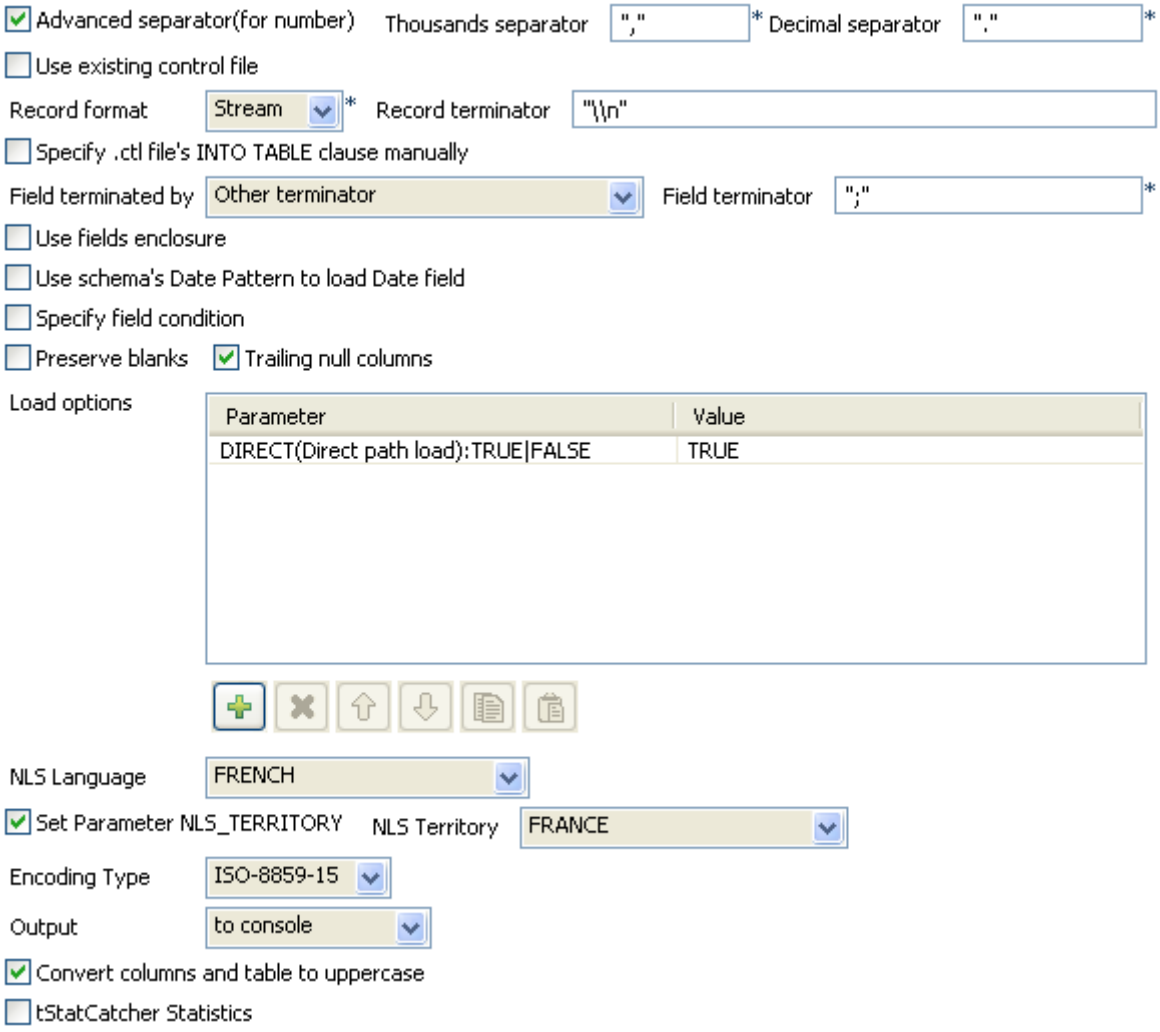
- Drop the following components: **tOracleInput**, **tFileOutputDelimited**, **tOracleBulkExec** from the **Palette** to the design workspace

- Connect the **tOracleInput** with the **tFileOutputDelimited** using a **row main** link.
- And connect the **tOracleInput** to the **tOracleBulkExec** using a **ThenRun** trigger link.
- Define the Oracle connection details. We recommend you to store the DB connection details in the Metadata repository in order to retrieve them easily at any time in any job.



- Define the schema, if it isn't stored either in the **Repository**. In this example, the schema is as follows: *ID_Contract*, *ID_Client*, *Contract_type*, *Contract_Value*.
- Change the default encoding to *AL32UTF8* encoding type.
- Define the **tFileOutputDelimited** component parameters, including output **File Name**, **Row separator** and **Fields delimiter**.
- Set also the **encoding** to the Oracle encoding type as above.
- Then double-click on the **tOracleBulkExec** to define the DB feeding properties.

- In the **Property Type**, select **Repository** mode if you stored the database connection details under the **Metadata** node of the **Repository** or select **Built-in** mode to define them manually. In this scenario, we use the **Built-in** mode.
- Thus, set the connection parameters in the following fields: **Host**, **Port**, **Database**, **Schema**, **Username**, and **Password**.
- Fill in the name of the **Table** to be fed and the **Action on data** to be carried out, in this use case: **insert**.
- In the **Schema Type** field, select **Built-in** mode, and click **[...]** button next to the **Edit schema** field to describe the structure of the data to be passed on to the next component.
- Click the **Advanced settings** view to configure the advanced settings of the component.



- Select the Use an existing control file check box if you want to use a control file (.ctl) storing the status of the physical structure of the database. Or, fill in the following fields manually: **Record format**, **Specify .ctl file's INTO TABLE clause manually**, **Field terminated by**, **Use field enclosure**, **Use schema's Date Pattern to load Date field**, **Specify field condition**, **Preserve blanks**, **Trailing null columns**, **Load options**, **NLS Language** et **Set Parameter NLS_TERRITORY** according to your database.
- Define the **encoding** as in preceding steps.

- For this scenario, in the **Output** field, select **to console** to output the standard output f the database to the console.

Press **F6** to run the job. The log output displays in the **Run** tab and the table is fed with the parameter file data.

Related topic: *Scenario: Inserting data in MySQL database on page 297*.

tOracleCommit

tOracleCommit Properties

This component is closely related to **tOracleConnection** and **tOracleRollback**. It usually doesn't make much sense to use these components independently in a transaction.

Component family	Databases/Oracle	
Function	Validates the data processed through the job into the connected DB	
Purpose	Using a unique connection, this component commits in one go a global transaction instead of doing that on every row or every batch and thus provides gain in performance.	
Basic settings	*Component list*	Select the **tOracleConnection** component in the list if more than one connection are planned for the current job.
	Close Connection	Clear this check box to continue to use the selected connection once the component has performed its task.
Usage	This component is to be used along with Oracle components, especially with **tOracleConnection** and **tOracleRollback** components.	
Limitation	n/a	

Related scenario

This component is closely related to **tOracleConnection** and **tOracleRollback**. It usually doesn't make much sense to use one of these without using a **tOracleConnection** component to open a connection for the current transaction.

For **tOracleCommit** related scenario, see *tMysqlConnection on page 269*.

tOracleConnection

tOracleConnection Properties

This component is closely related to **tOracleCommit** and **tOracleRollback**. It usually doesn't make much sense to use one of these without using a **tOracleConnection** component to open a connection for the current transaction.

Component family	Databases/Oracle	
Function	Opens a connection to the database for a current transaction.	
Purpose	Allows to commit a whole job data in one go to the output database as one transaction when validated.	
Basic settings	*Property type*	Either Built-in or Repository.
		Built-in: No property data stored centrally.
		Repository: Select the repository file where Properties are stored. The fields that come after are pre-filled in using the fetched data.
	Connection type	Drop-down list of available drivers.
	DB Version	Select the Oracle version in use
	Host	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database
	Schema	Name of the schema
	Username and *Password*	DB user authentication data.
	Encoding Type	Select the encoding from the list or select Custom and define it manually. This field is compulsory for DB data handling.
	Use or register a shared DB Connection	Select this check box to share your connection or fetch a connection shared by a parent or child Job. **Shared DB Connection Name:** set or type in the shared connection name.
Usage	This component is to be used along with Oracle components, especially with **tOracleCommit** and **tOracleRollback** components.	
Limitation	n/a	

Related scenario

This component is closely related to **tOracleCommit** and **tOracleRollback**. It usually doesn't make much sense to use one of these without using a **tOracleConnection** component to open a connection for the current transaction.

For **tOracleConnection** related scenario, see *tMysqlConnection on page 269.*

tOracleInput

tOracleInput properties

Component family	Databases/Oracle	
Function	**tOracleInput** reads a database and extracts fields based on a query.	
Purpose	**tOracleInput** executes a DB query with a strictly defined order which must correspond to the schema definition. Then it passes on the field list to the next component via a Main row link.	
Basic settings	*Property type*	Either Built-in or Repository
		Built-in: No property data stored centrally.
		Repository: Select the repository file where Properties are stored. The fields that come after are pre-filled in using the fetched data.
		Click this icon to open a database connection wizard and store the database connection parameters you set in the component **Basic settings** view. For more information about setting up and storing database connection parameters, see *Setting up a DB connection* of **Talend Open Studio** User Guide.
	Connection type	Drop-down list of available drivers.
	DB Version	Select the Oracle version in use
	Use existing connection	Select this check box when using a **tOracleConnection** component.
	Host	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database
	Oracle schema	Oracle schema name.
	Username and *Password*	DB user authentication data.
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either built-in or remotely stored in the Repository.
		Built-in: The schema is created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	Table name	Database table name.

	Query type and *Query*	Enter your DB query paying particularly attention to properly sequence the fields in order to match the schema definition.
Advanced settings	*tState*Catcher Statistics	Select this check box to collect log data at the component level.
	Encoding Type	Select the encoding type from the list or select **Custom** and define it manually. This field is compulsory for DB data handling.
	Use cursor	When selected, helps to decide the row set to work with at a time and thus optimize performance.
	Trim all the String/Char columns	Select this check box to remove leading and trailing whitespace from all the String/Char columns.
	Trim column	Remove leading and trailing whitespace from defined columns.
Usage	This component covers all possibilities of SQL queries onto a Oracle database.	

Related scenarios

Related topics in **tDBInput** scenarios:

- *Scenario 1: Displaying selected data from DB table on page 163*
- *Scenario 2: Using StoreSQLQuery variable on page 164*

Related topic in **tContextLoad** *Scenario: Dynamic context use in MySQL DB insert on page 652*.

tOracleOutput

tOracleOutput properties

Component family	Databases/Oracle	
Function	**tOracleOutput** writes, updates, makes changes or suppresses entries in a database.	
Purpose	**tOracleOutput** executes the action defined on the table and/or on the data contained in the table, based on the flow incoming from the preceding component in the job.	
Basic settings	*Property type*	Either Built-in or Repository.
		Built-in: No property data stored centrally.
		Repository: Select the repository file where Properties are stored. The fields that come after are pre-filled in using the fetched data.
		Click this icon to open a database connection wizard and store the database connection parameters you set in the component **Basic settings** view. For more information about setting up and storing database connection parameters, see *Setting up a DB connection* of **Talend Open Studio** User Guide.
	Use an existing connection	Select this check box when using a **tOracleConnection** component.
	Connection type	Drop-down list of available drivers.
	DB Version	Select the Oracle version in use
	Host	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database
	Username and *Password*	DB user authentication data.
	Table	Name of the table to be written. Note that only one table can be written at a time
	Action on table	On the table defined, you can perform one of the following operations: **None**: No operation is carried out. **Drop and create a table**: The table is removed and created again. **Create a table**: The table does not exist and gets created. **Create a table if not exists**: The table is created if it does not exist. **Drop a table if exists and create**: The table is removed if it already exists and created again. **Clear a table**: The table content is deleted.

	Action on data	On the data of the table defined, you can perform: **Insert**: Add new entries to the table. If duplicates are found, job stops. **Update**: Make changes to existing entries **Insert or update**: Add entries or update existing ones. **Update or insert**: Update existing entries or create it if non existing **Delete**: Remove entries corresponding to the input flow. *It is necessary to specify at least one column as a primary key on which the Update and Delete operations are based. You can do that by clicking Edit Schema and selecting the check box(es) next to the column(s) you want to set as primary key(s). For an advanced use, click the Advanced settings view where you can simultaneously define primary keys for the Update and Delete operations. To do that: Select the Use field options check box and then in the Key in update column, select the check boxes next to the column names you want to use as a base for the Update operation. Do the same in the Key in delete column for the Delete operation.*
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either built-in or remotely stored in the Repository.
		Built-in: The schema is created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	Die on error	Clear this check box to skip the row on error and complete the process for error-free rows.
Advanced settings	*Encoding Type*	Select the encoding type from the list or select **Custom** and define it manually. This field is compulsory for DB data handling.
	Commit every	Enter the number of rows to be completed before committing batches of rows together into the DB. This option ensures transaction quality (but not rollback) and, above all, better performance at execution.
	Additional Columns	This option is not offered if you create (with or without drop) the DB table. This option allows you to call SQL functions to perform actions on columns, which are not insert, nor update or delete actions, or action that require particular preprocessing.

		Name: Type in the name of the schema column to be altered or inserted as new column
		SQL expression: Type in the SQL statement to be executed in order to alter or insert the relevant column data.
		Position: Select **Before**, **Replace** or **After** following the action to be performed on the reference column.
		Reference column: Type in a column of reference that the **tDBOutput** can use to place or replace the new or altered column.
	Use field options	Select this check box to customize a request, especially when there is double action on data.
	Enable debug mode	Select this check box to display each step during processing entries in a database.
	tStateCatcher Statistics	Select this check box to collect log data at the component level.
	Use Batch Size	When selected, enables you to define the number of lines in each processed batch.
Usage	This component offers the flexibility benefit of the DB query and covers all possibilities of SQL queries.	

Related scenarios

For **tOracleOutput** related topics, see:

- **tDBOutput** *Scenario: Displaying DB output on page 168*
- **tMySQLOutput** *Scenario 1: Adding a new column and altering data in a DB table on page 283*.

tOracleOutputBulk

tOracleOutputBulk properties

tOracleOutputBulk and **tOracleBulkExec** components are used together to first output the file that will be then used as parameter to execute the SQL query stated. These two steps compose the **tOracleOutputBulkExec** component, detailed in a separate section. The interest in having two separate elements lies in the fact that it allows transformations to be carried out before the data loading.

Component family	Databases/Oracle	
Function	Writes a file with columns based on the defined delimiter and the Oracle standards	
Purpose	Prepares the file to be used as parameter in the INSERT query to feed the Oracle database.	
Basic settings	*Property type*	Either Built-in or Repository.
		Built-in: No property data stored centrally.
		Repository: Select the Repository file where Properties are stored. The following fields are pre-filled in using fetched data.
	File Name	Name of the file to be processed. Related topic:*Defining variables from the Component view* of **Talend Open Studio** User Guide.
	Append	Select this check box to add the new rows at the end of the file
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the Repository.
		Built-in: The schema will be created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused in various projects and job designs. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
Advanced settings	*Advanced separator (for number)*	Select this check box to change data separators for numbers: **Thousands separator**: define separators you want to use for thousands. **Decimal separator**: define separators you want to use for decimals.

	Field separator	Character, string or regular expression to separate fields.
	Row separator	String (ex: “\n”on Unix) to separate rows.
	Encodind Type	Select the encoding from the list or select Custom and define it manually. This field is compulsory for DB data handling.
	tStatCatcher Statistics	Select this check box to gather the job processing metadata at a job level as well as at each component level.
Usage	This component is to be used along with **tOracleBulkExec** component. Used together they offer gains in performance while feeding a Oracle database.	

Related scenarios

For uses cases in relation with **tOracleOutputBulk**, see the following scenarios:

- **tMysqlOutputBulk** *Scenario: Inserting transformed data in MySQL database on page 293*
- **tMysqlOutputBulkExec** *Scenario: Inserting data in MySQL database on page 297*
- **tOracleBulkExec** *Scenario: Truncating and inserting file data into Oracle DB on page 329*

tOracleOutputBulkExec

tOracleOutputBulkExec properties

Component family	Databases/Oracle	
Function	Executes the Insert action on the data provided.	
Purpose	As a dedicated component, it allows gains in performance during Insert operations to an Oracle database.	
Basic settings	*Property type*	Either Built-in or Repository.
		Built-in: No property data stored centrally.
		Repository: Select the repository file where Properties are stored. The fields that come after are pre-filled in using the fetched data.
	Use an existing connection	Select this check box and click the relevant **tOracleConnection** component on the Component list to reuse the connection details you already defined.
	Connection type	List of available drivers
	DB Version	Select the Oracle version in use
	Host	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database
	Schema	Name of the schema
	Username and *Password*	DB user authentication data.
	Table	Name of the table to be written. Note that only one table can be written at a time and that the table must exist for the insert operation to succeed.
	Action on table	On the table defined, you can perform one of the following operations: **None**: No operations is carried out. **Drop and create the table**: The table is removed and created again. **Create a table**: The table does not exist and gets created. **Create table if doesn't exist**: The table is created if does not exist. **Clear a table**: The table content is deleted. **Truncate table**: The table content is deleted. You do not have the possibility to rollback the operation.
	File Name	Name of the file to be processed. Related topic:*Defining variables from the Component view* of **Talend Open Studio** User Guide.

	Create directory if not exists	This check box is selected by default. It creates a directory to hold the output table if required.
	Append	Select this check box to add the new rows at the end of the file.
	Action on data	On the data of the table defined, you can perform: **Insert**: Add new entries to the table. If duplicates are found, job stops. **Update**: Make changes to existing entries **Insert or update**: Add entries or update existing ones. **Update or insert**: Update existing entries or create it if non existing **Truncate**: Remove all entries from table.
	Field separator	Character, string or regular expression to separate fields.
	Encoding Type	Select the encoding from the list or select Custom and define it manually. This field is compulsory for DB data handling.
Advanced settings	*Advanced separator (for number)*	Select this check box to change data separators for numbers: **Thousands separator**: define separators you want to use for thousands. **Decimal separator**: define separators you want to use for decimals.
	Use existing control file	Select this check box and browse to the .ctl control file you want to use.
	Field separator	Character, string or regular expression to separate fields.
	Row separator	String (ex: “\n”on Unix) to separate rows.
	Specify .ctl file’s INTO TABLE clause manually	Select this check box to enter manually the INTO TABLE clause of the control file directly into the code.
	Use schema’s Date Pattern to load Date field	Select this check box to use the date model indicated in the schema for dates.
	Specify field condition	Select this check box to define a condition for loading data.
	Preserve blanks	Select this check box to preserve blank spaces.
	Trailing null columns	Select this check box to load data with all empty columns.
	Load options	Click the plus button to add lines and define the options for loading your data: **Parameter**: select from the list a data loading parameter. **Value**: define a value for the selected parameter.
	NLS Language	From the drop-down list, select the language for your data if the data is not in Unicode.

	Set Parameter NLS_TERRITORY	Select this check box to modify the conventions used for date and time formats. The default value is that of the operating system.
	Encoding Type	Select the encoding from the list or select **Custom** and define it manually. This field is compulsory for DB data handling.
	Output	Select the type of output for the standard output of the Oracle database: **to console**, **to global variable**.
	Convert columns and table names to uppercase	Select this check box to put columns and table names in upper case.
	tStatCatcher Statistics	Select this check box to gather the job processing metadata at a job level as well as at each component level.
Usage	This component is mainly used when no particular transformation is required on the data to be loaded onto the database.	
Limitation	n/a	

Related scenarios

For uses cases in relation with **tOracleOutputBulkExec**, see the following scenarios:

- **tMysqlOutputBulk** *Scenario: Inserting transformed data in MySQL database on page 293*.
- **tMysqlOutputBulkExec** *Scenario: Inserting data in MySQL database on page 297*.
- **tOracleBulkExec** *Scenario: Truncating and inserting file data into Oracle DB on page 329*.

tOracleRollback

tOracleRollback properties

This component is closely related to **tOracleCommit** and **tOracleConnection**. It usually doesn't make much sense to use these components independently in a transaction.

Component family	Databases	
Function	Cancel the transaction commit in the connected DB.	
Purpose	Avoids to commit part of a transaction involuntarily.	
Basic settings	*Component list*	Select the **tOracleConnection** component in the list if more than one connection are planned for the current job.
	Close Connection	Clear this check box to continue to use the selected connection once the component has performed its task.
Usage	This component is to be used along with Oracle components, especially with **tOracleConnection** and **tOracleCommit** components.	
Limitation	n/a	

Related scenario

This component is closely related to **tOracleConnection** and **tOracleCommit**. It usually doesn't make much sense to use one of these without using a **tOracleConnection** component to open a connection for the current transaction.

For **tOracleRollback** related scenario, see *tMysqlRollback on page 299*.

tOracleRow

tOracleRow properties

Component family	Databases/Oracle	
Function	**tOracleRow** is the specific component for this database query. It executes the SQL query stated onto the specified database. The row suffix means the component implements a flow in the job design although it doesn't provide output.	
Purpose	Depending on the nature of the query and the database, **tOracleRow** acts on the actual DB structure or on the data (although without handling data). The SQLBuilder tool helps you write easily your SQL statements.	
Basic settings	*Property type*	Either Built-in or Repository.
		Built-in: No property data stored centrally.
		Repository: Select the repository file where Properties are stored. The fields that come after are pre-filled in using the fetched data.
	Use an existing connection	Select this check box and click the relevant **tOracleConnection** component on the Component list to reuse the connection details you already defined.
	Connection type	Drop-down list of available drivers.
	Host	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database
	Username and *Password*	DB user authentication data.
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either built-in or remotely stored in the Repository.
		Built-in: The schema is created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	Query type	Either Built-in or Repository.
		Built-in: Fill in manually the query statement or build it graphically using SQLBuilder
		Repository: Select the relevant query stored in the Repository. The Query field gets accordingly filled in.

	Query	Enter your DB query paying particularly attention to properly sequence the fields in order to match the schema definition.
	Die on error	Clear this check box to skip the row on error and complete the process for error-free rows.
Advanced settings	*Propagate QUERY's recordset*	Select this check box to insert the result of the query into a COLUMN of the current flow.
	Encoding Type	Select the encoding from the list or select **Custom** and define it manually. This field is compulsory for DB data handling.
	Commit every	Number of rows to be completed before committing batches of rows together into the DB. This option ensures transaction quality (but not rollback) and above all better performance on executions.
	*tState*Catcher Statistics	Select this check box to collect log data at the component level.
Usage	This component offers the flexibility benefit of the DB query and covers all possibilities of SQL queries.	

Related scenarios

For related topics, see:

- **tDBSQLRow** *Scenario: Resetting a DB auto-increment on page 172*
- **tMySQLRow** *Scenario: Removing and regenerating a MySQL table index on page 301*.

tOracleSCD

tOracleSCD belongs to two component families: Business Intelligence and Databases. For more information on it, see *tOracleSCD on page 23*.

tOracleSP

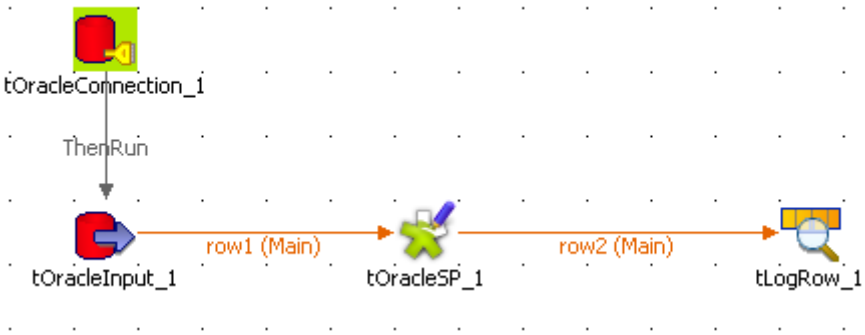
tOracleSP Properties

Component family	Databases/Oracle	
Function	**tOracleSP** calls the database stored procedure.	
Purpose	**tOracleSP** offers a convenient way to centralize multiple or complex queries in a database and call them easily.	
Basic settings	*Property type*	Either Built-in or Repository.
		Built-in: No property data stored centrally.
		Repository: Select the repository file where Properties are stored. The fields that come after are pre-filled in using the fetched data.
	DB Version	Select the Oracle version in use
	Host	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database
	Schema	Name of the Schema
	Username and *Password*	DB user authentication data.
	Encoding Type	Select the encoding from the list or select Custom and define it manually. This field is compulsory for DB data handling.
	Schema type and *Edit Schema*	In SP principle, the schema is an input parameter. A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either built-in or remotely stored in the Repository.
		Built-in: The schema is created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	SP Name	Type in the exact name of the Stored Procedure (or Function)
	Is Function / Return result in	Check this box, if the stored procedure is a function and one value only is to be returned. Select on the list the schema column, the value to be returned is based on.

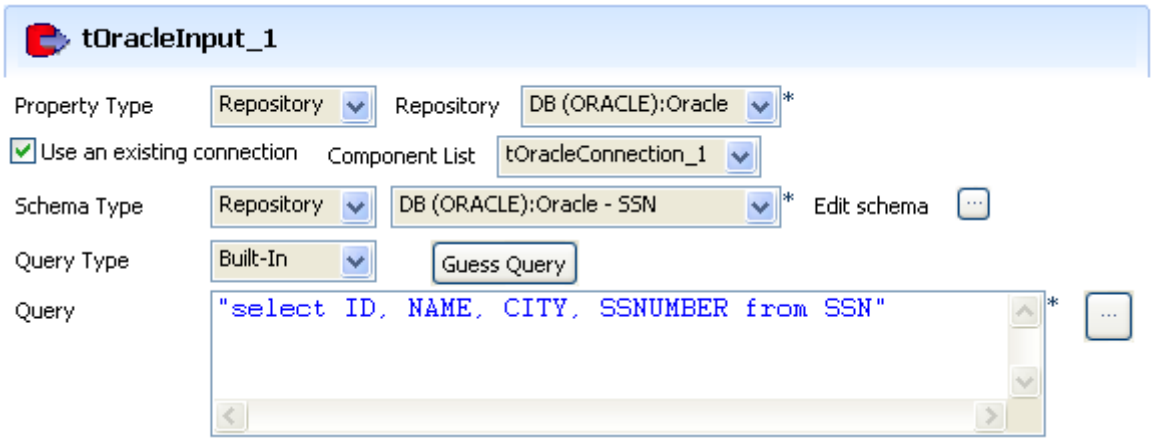
	Parameters	Click the Plus button and select the various **Schema Columns** that will be required by the procedures. Note that the SP schema can hold more columns than there are paramaters used in the procedure. Select the **Type** of parameter: **IN**: Input parameter **OUT**: Output parameter/return value **IN OUT**: Input parameter is to be returned as value, likely after modification through the procedure (function). **RECORDSET**: Input parameters is to be returned as a set of values, rather than single value. Check the *tParseRecordSet* component if you want to analyze a set of records from a database table or DB query and return single records.
Usage	This component is used as intermediary component. It can be used as start component but only input parameters are thus allowed.	
Limitation	The Stored Procedures syntax should match the Database syntax.	

Scenario: Checking number format using a stored procedure

The following job aims at connecting to an Oracle Database containing Social Security Numbers and their holders' name, calling a stored procedure that checks the SSN format of against a standard ###-##-#### format. Then the verification output results, 1 for valid format and 0 for wrong format get displayed onto the execution console.



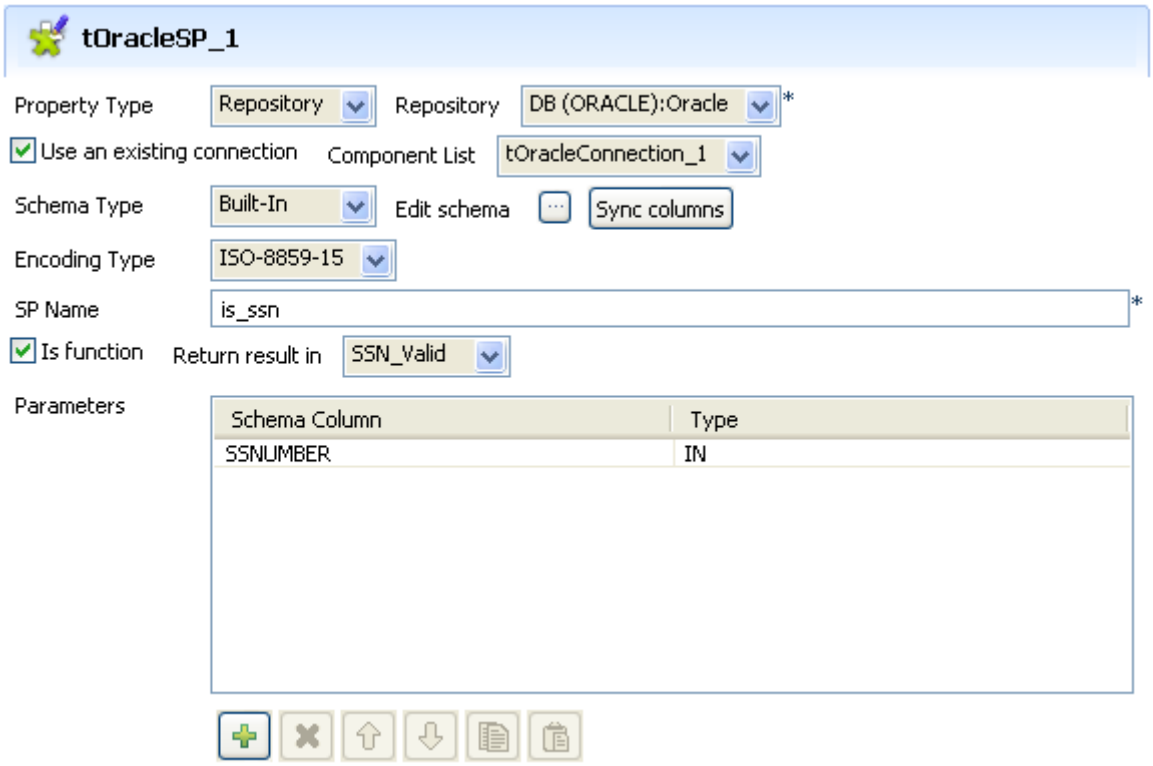
- Drag and drop the following components from the **Palette**: **tOracleConnection**, **tOracleInput**, **tOracleSP** and **tLogRow**.
- Link the **tOracleConnection** to the tOracleInput using a **Then Run** connection as no data is handled here.
- And connect the other components using a **Row Main** link as rows are to be passed on as parameter to the SP component and to the console.
- In the **tOracleConnection**, define the details of connection to the relevant Database. You will then be able to reuse this information in all other DB-related components.
- Then select the **tOracleInput** and define its properties.



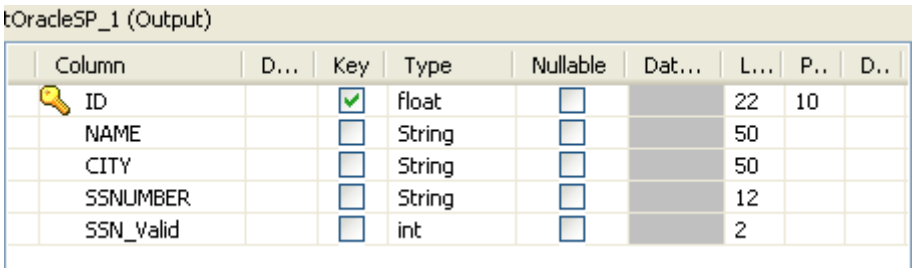
- Select the **Use an existing connection** check box and select the **tOracleConnection** component in the list in order to reuse the connection details that you already set.
- Select **Repository** as **Property type** as the Oracle schema is defined in the DB Oracle connection entry of the Repository. If you haven't recorded the Oracle DB details in the **Repository**, then fill in the Schema name manually.
- Then select **Repository** as **Schema type**, and retrieve the relevant schema corresponding to your Oracle DB table.

	ID	NAME	CITY	SSNUMBER
1	1	Jack	LA	123-45-6789
2	2	Tom	NYC	123-A5-6789
3	3	Bill	SF	123=45-6789
4	4	Jana	NYC	236-52-2956
5	5	Brandon	SLC	561-52-B267

- In this example, the SSN table has a four-column schema that includes *ID*, *NAME*, *CITY* and *SSNUMBER*.
- In the **Query** field, type in the following Select query or select it in the list, if you stored it in the Repository.
  ```
  select ID, NAME, CITY, SSNUMBER from SSN
  ```
- Then select the **tOracleSP** and define its **Basic settings**.



- Like for the **tOracleInput** component, select **Repository** in the **Property type** field and select the **Use an existing connection** check box, then select the relevant entries in the respective list.
- The schema used for the **tOracleSP** slightly differs from the input schema. Indeed, an extra column (*SSN_Valid*) is added to the Input schema. This column will hold the format validity status (*1* or *0*) produced by the procedure.

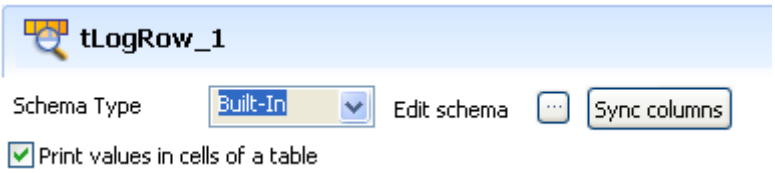


- Then select the **Encoding type** in the list.
- In the **SP Name** field, type in the exact name of the stored procedure (or function) as called in the Database. In this use case, the stored procedure name is *is_ssn*.

- The basic function used in this particular example is as follows:

```
CREATE OR REPLACE FUNCTION is_ssn(string_in VARCHAR2) RETURN
PLS_INTEGER
IS
-- validating ###-##-#### format
BEGIN
  IF TRANSLATE(string_in, '0123456789A', 'AAAAAAAAAAB') =
     'AAA-AA-AAAA' THEN
    RETURN 1;
  END IF;
  RETURN 0;
END is_ssn;
/
```

- As a return value is expected in this use case, the procedure acts as a function, so select the **Is function** check box.
- The only return value expected is based on the *ssn_valid* column, hence select the relevant list entry.
- In the **Parameters** area, define the input and output parameters used in the procedure. In this use case, only the *SSNumber* column from the schema is used in the procedure.
- Click the plus sign to add a line to the table and select the relevant column (*SSNumber*) and type (*IN*).
- Then select the tLogRow component and click Sync Column to make sure the schema is passed on from the preceding **tOracleSP** component.



- Select the **Print values in cells of a table** check box to facilitate the output reading.
- Then save your job and press **F6** to run it.

```
Starting job OracleSP at 15:14 23/08/2007.
.---+-------+----+-----------+---------.
|               tLogRow_1                |
|=--+-------+----+-----------+---------=|
|ID |NAME   |CITY|SSNUMBER   |SSN_Valid|
|=--+-------+----+-----------+---------=|
|1.0|Jack   |LA  |123-45-6789|1        |
|2.0|Tom    |NYC |123-A5-6789|0        |
|3.0|Bill   |SF  |123=45-6789|0        |
|4.0|Jana   |NYC |236-52-2956|1        |
|5.0|Brandon|SLC |561-52-B267|0        |
'---+-------+----+-----------+---------'

Job OracleSP ended at 15:14 23/08/2007. [exit code=0]
```

On the console, you can read the output results. All input schema columns are displayed eventhough they are not used as parameters in the stored procedure.

The final column shows the expected return value, i.e. whether the SS Number checked is valid or not.

Check the *tParseRecordSet* component if you want to analyze a set of records from a database table or DB query and return single records.

tParseRecordSet

You can find this component at the root of **Databases** group of the **Palette** of **Talend Open Studio**. **tParseRecordSet** covers needs related indirectly to the use of any database.

tParseRecordSet properties

Component family	Databases	
Function	**tParseRecordSet** parses a set of records from a database table or DB query and possibly returns single records.	
Purpose	.Allows to parse a recordset rather than individual records from a table.	
Basic settings	*Prev. Comp. Column list*	Set the column from the database that holds the recordset.
	Schema type and *Edit Schema*	In SP principle, the schema is an input parameter. A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either built-in or remotely stored in the Repository.
		Built-in: The schema is created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	Attribute table	Set the position value of each column for single records from the recordset.
Usage	This component is used as intermediary component. It can be used as start component but only input parameters are thus allowed.	
Limitation	This component is mainly designed for a use with the SP component Recordset feature.	

Scenario

No scenario is available for this component yet.

tPostgresqlBulkExec

tPostgresqlBulkExec properties

tPostgresqlOutputBulk and **tPostgresqlBulkExec** components are used together to first output the file that will be then used as parameter to execute the SQL query stated. These two steps compose the **tPostgresqlOutputBulkExec** component, detailed in a separate section. The interest in having two separate elements lies in the fact that it allows transformations to be carried out before the data loading in the database.

Component family	Databases/Postgresql	
Function	Executes the Insert action on the data provided.	
Purpose	As a dedicated component, **tPostgresqlBulkExec** offers gains in performance while carrying out the Insert operations to a Postgresql database	
Basic settings	*Property type*	Either Built-in or Repository.
		Built-in: No property data stored centrally.
		Repository: Select the repository file where Properties are stored. The fields that come after are pre-filled in using the fetched data.
	Use an existing connection	Select this check box and click the relevant **tPostgrresqlConnection** component on the Component list to reuse the connection details you already defined.
	Host	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database.
	Schema	Name of the schema.
	Username and *Password*	DB user authentication data.
	Table	Name of the table to be written. Note that only one table can be written at a time and that the table must exist for the insert operation to succeed.
	Action on table	On the table defined, you can perform one of the following operations: **None**: No operation is carried out. **Drop and create table**: The table is removed and created again. **Create table**: The table does not exist and gets created. **Create table if not exists**: The table is created if it does not exist. **Clear table**: The table content is deleted. **Truncate table**: The table content is deleted. You don not have the possibility to rollback the operation.

	File Name	Name of the file to be processed. Related topic:*Defining variables from the Component view* of **Talend Open Studio** User Guide.
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either built-in or remotely stored in the Repository.
		Built-in: The schema is created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	Fields terminated by	Character, string or regular expression to separate fields.
Usage	This component is to be used along with **tPostgresqlOutputBulk** component. Used together, they can offer gains in performance while feeding a Postgresql database.	
Limitation	n/a	

Related scenarios

For uses cases in relation with **tPostgresqlBulkExec**, see the following scenarios:

- **tMysqlOutputBulk** *Scenario: Inserting transformed data in MySQL database on page 293*
- **tMysqlOutputBulkExec** *Scenario: Inserting data in MySQL database on page 297*
- **tOracleBulkExec** *Scenario: Truncating and inserting file data into Oracle DB on page 329*

tPostgresqlCommit

tPostgresqlCommit Properties

This component is closely related to **tPostgresqlCommit** and **tPostgresqlRollback**. It usually doesn't make much sense to use these components independently in a transaction.

Component family	Databases/Postgresql	
Function	Validates the data processed through the job into the connected DB	
Purpose	Using a unique connection, this component commits in one go a global transaction instead of doing that on every row or every batch and thus provides gain in performance.	
Basic settings	*Component list*	Select the **tPostgresqlConnection** component in the list if more than one connection are planned for the current job.
	Close Connection	Clear this check box to continue to use the selected connection once the component has performed its task.
Usage	This component is to be used along with Postgresql components, especially with **tPostgresqlConnection** and **tPostgresqlRollback** components.	
Limitation	n/a	

Related scenario

This component is closely related to **tPostgresqlConnection** and **tPostgresqlRollback**. It usually doesn't make much sense to use one of these without using a **tPostgresqlConnection** component to open a connection for the current transaction.

For **tPostgresqlCommit** related scenario, see *tMysqlConnection on page 269*.

tPostgresqlConnection

tPostgresqlConnection Properties

This component is closely related to **tPostgresqlCommit** and **tPostgresqlRollback**. It usually doesn't make much sense to use one of these without using a **tPostgresqlConnection** component to open a connection for the current transaction.

Component family	Databases/Postgresql	
Function	Opens a connection to the database for a current transaction.	
Purpose	Allows to commit a whole job data in one go to the output database as one transaction when validated.	
Basic settings	*Property type*	Either Built-in or Repository.
		Built-in: No property data stored centrally.
		Repository: Select the repository file where Properties are stored. The fields that come after are pre-filled in using the fetched data.
	Host	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database
	Schema	Exact name of the schema
	Username and *Password*	DB user authentication data.
	Encoding Type	Select the encoding from the list or select Custom and define it manually. This field is compulsory for DB data handling.
	Use or register a shared DB Connection	Select this check box to share your connection or fetch a connection shared by a parent or child Job. **Shared DB Connection Name:** set or type in the shared connection name.
Usage	This component is to be used along with Postgresql components, especially with **tPostgresqlCommit** and **tPostgresqlRollback** components.	
Limitation	n/a	

Related scenario

This component is closely related to **tPostgresqlCommit** and **tPostgresqlRollback**. It usually doesn't make much sense to use one of these without using a **tPostgresqlConnection** component to open a connection for the current transaction.

For **tPostgresqlConnection** related scenario, see *tMysqlConnection on page 269*.

tPostgresqlInput

tPostgresqlInput properties

Component family	Databases/ PostgreSQL	
Function	**tPostgresqlInput** reads a database and extracts fields based on a query.	
Purpose	**tPostgresqlInput** executes a DB query with a strictly defined order which must correspond to the schema definition. Then it passes on the field list to the next component via a Main row link.	
Basic settings	*Property type*	Either Built-in or Repository
		Built-in: No property data stored centrally.
		Repository: Select the repository file where Properties are stored. The fields that come after are pre-filled in using the fetched data.
		Click this icon to open a database connection wizard and store the database connection parameters you set in the component **Basic settings** view. For more information about setting up and storing database connection parameters, see *Setting up a DB connection* of **Talend Open Studio** User Guide.
	Use existing connection	Select this check box when using a **tPostgresqlConnection** component.
	Host	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database
	Schema	Exact name of the schema
	Username and *Password*	DB user authentication data.
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either built-in or remotely stored in the Repository.
		Built-in: The schema is created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	Table name	Name of the table to be read.
	Query type and *Query*	Enter your DB query paying particularly attention to properly sequence the fields in order to match the schema definition.

Advanced settings	*Encoding Type*	Select the encoding type from the list or select **Custom** and define it manually. This field is compulsory for DB data handling.
	Use cursor	When selected, helps to decide the row set to work with at a time and thus optimize performance.
	Trim all the String/Char columns	Select this check box to remove leading and trailing whitespace from all the String/Char columns.
	Trim column	Remove leading and trailing whitespace from defined columns.
	*tState*Catcher Statistics	Select this check box to collect log data at the component level.
Usage	This component covers all possibilities of SQL queries onto a Postgresql database.	

Related scenarios

Related topics in **tDBInput** scenarios:

- *Scenario 1: Displaying selected data from DB table on page 163*
- *Scenario 2: Using StoreSQLQuery variable on page 164*

Related topic in **tContextLoad** *Scenario: Dynamic context use in MySQL DB insert on page 652*.

tPostgresqlOutput

tPostgresqlOutput properties

Component family	Databases/Postgresql	
Function	**tPostgresqlOutput** writes, updates, makes changes or suppresses entries in a database.	
Purpose	**tPostgresqlOutput** executes the action defined on the table and/or on the data contained in the table, based on the flow incoming from the preceding component in the job.	
Basic settings	*Property type*	Either Built-in or Repository.
		Built-in: No property data stored centrally.
		Repository: Select the repository file where Properties are stored. The fields that come after are pre-filled in using the fetched data.
		Click this icon to open a database connection wizard and store the database connection parameters you set in the component **Basic settings** view. For more information about setting up and storing database connection parameters, see *Setting up a DB connection* of **Talend Open Studio** User Guide.
	Use existing connection	Select this check box when using a **tPostgresqlConnection** component.
	Host	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database
	Schema	Exact name of the schema
	Username and *Password*	DB user authentication data.
	Table	Name of the table to be written. Note that only one table can be written at a time
	Action on table	On the table defined, you can perform one of the following operations: **None**: No operation is carried out. **Drop and create a table**: The table is removed and created again. **Create a table**: The table does not exist and gets created. **Create a table if not exists**: The table is created if it does not exist. **Drop a table if exists and create**: The table is removed if already exists and created again. **Clear a table**: The table content is deleted.

	Action on data	On the data of the table defined, you can perform: **Insert**: Add new entries to the table. If duplicates are found, job stops. **Update**: Make changes to existing entries **Insert or update**: Add entries or update existing ones. **Update or insert**: Update existing entries or create it if non existing **Delete**: Remove entries corresponding to the input flow. *It is necessary to specify at least one column as a primary key on which the Update and Delete operations are based. You can do that by clicking Edit Schema and selecting the check box(es) next to the column(s) you want to set as primary key(s). For an advanced use, click the Advanced settings view where you can simultaneously define primary keys for the Update and Delete operations. To do that: Select the Use field options check box and then in the Key in update column, select the check boxes next to the column names you want to use as a base for the Update operation. Do the same in the Key in delete column for the Delete operation.*
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either built-in or remotely stored in the Repository.
		Built-in: The schema is created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	Die on error	Clear this check box to skip the row on error and complete the process for error-free rows.
Advanced settings	*Encoding Type*	Select the encoding type from the list or select **Custom** and define it manually. This field is compulsory for DB data handling.
	Commit every	Enter the number of rows to be completed before committing batches of rows together into the DB. This option ensures transaction quality (but not rollback) and, above all, better performance at execution.
	Additional Columns	This option is not offered if you create (with or without drop) the DB table. This option allows you to call SQL functions to perform actions on columns, which are not insert, nor update or delete actions, or action that require particular preprocessing.

		Name: Type in the name of the schema column to be altered or inserted as new column
		SQL expression: Type in the SQL statement to be executed in order to alter or insert the relevant column data.
		Position: Select **Before**, **Replace** or **After** following the action to be performed on the reference column.
		Reference column: Type in a column of reference that the **tDBOutput** can use to place or replace the new or altered column.
	*Use field option*s	Select this check box to customize a request, especially when there is double action on data.
	Enable debug mode	Select this check box to display each step during processing entries in a database.
	*tState*Catcher Statistics	Select this check box to collect log data at the component level.
Usage	This component offers the flexibility benefit of the DB query and covers all possibilities of SQL queries.	

Related scenarios

For **tPostgresqlOutput** related topics, see:

- **tDBOutput** *Scenario: Displaying DB output on page 168*
- **tMySQLOutput** *Scenario 1: Adding a new column and altering data in a DB table on page 283*.

tPostgresqlOutputBulk

tPostgresqlOutputBulk properties

tPostgresqlOutputBulk and **tPostgresqlBulkExec** components are used together to first output the file that will be then used as parameter to execute the SQL query stated. These two steps compose the **tPostgresqlOutputBulkExec** component, detailed in a separate section. The interest in having two separate elements lies in the fact that it allows transformations to be carried out before the data loading.

Component family	Databases/Postgresql	
Function	Writes a file with columns based on the defined delimiter and the Postgresql standards	
Purpose	Prepares the file to be used as parameter in the INSERT query to feed the Postgresql database.	
Basic settings	*Property type*	Either Built-in or Repository.
		Built-in: No property data stored centrally.
		Repository: Select the Repository file where Properties are stored. The following fields are pre-filled in using fetched data.
	File Name	Name of the file to be processed. Related topic:*Defining variables from the Component view* of **Talend Open Studio** User Guide.
	Field separator	Character, string or regular expression to separate fields.
	Row separator	String (ex: “\n”on Unix) to distinguish rows.
	Append	Select this check box to add the new rows at the end of the file
	Include header	Select this check box to include the column header to the file.
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the Repository.
		Built-in: The schema will be created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused in various projects and job designs. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.

	Encoding Type	Select the encoding from the list or select Custom and define it manually. This field is compulsory for DB data handling.
Usage	This component is to be used along with **tPostgresqlBulkExec** component. Used together they offer gains in performance while feeding a Postgresql database.	

Related scenarios

For uses cases in relation with **tPostgresqlOutputBulk**, see the following scenarios:

- **tMysqlOutputBulk** *Scenario: Inserting transformed data in MySQL database on page 293*
- **tMysqlOutputBulkExec** *Scenario: Inserting data in MySQL database on page 297*
- **tOracleBulkExec** *Scenario: Truncating and inserting file data into Oracle DB on page 329*

tPostgresqlOutputBulkExec

tPostgresqlOutputBulkExec properties

Component family	Databases/Postgresql	
Function	Executes the Insert action on the data provided.	
Purpose	As a dedicated component, it allows gains in performance during Insert operations to a Postgresql database.	
Basic settings	*Property type*	Either Built-in or Repository.
		Built-in: No property data stored centrally.
		Repository: Select the repository file where Properties are stored. The fields that come after are pre-filled in using the fetched data.
	Host	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database
	Username and *Password*	DB user authentication data.
	Table	Name of the table to be written. Note that only one table can be written at a time and that the table must exist for the insert operation to succeed.
	File Name	Name of the file to be processed. Related topic:*Defining variables from the Component view* of **Talend Open Studio** User Guide
	Field separator	Character, string or regular expression to separate fields.
	Row separator	String (ex: “\n”on Unix) to distinguish rows.
Usage	This component is mainly used when no particular tranformation is required on the data to be loaded onto the database.	

Related scenarios

For uses cases in relation with **tPostgresqlOutputBulkExec**, see the following scenarios:

- **tMysqlOutputBulk** *Scenario: Inserting transformed data in MySQL database on page 293*
- **tMysqlOutputBulkExec** *Scenario: Inserting data in MySQL database on page 297*
- **tOracleBulkExec** *Scenario: Truncating and inserting file data into Oracle DB on page 329*

tPostgresqlRollback

tPostgresqlRollback properties

This component is closely related to **tPostgresqlCommit** and **tPostgresqlConnection**. It usually doesn't make much sense to use these components independently in a transaction.

Component family	Databases	
Function	Cancel the transaction commit in the connected DB.	
Purpose	Avoids to commit part of a transaction involuntarily.	
Basic settings	*Component list*	Select the **tPostgresqlConnection** component in the list if more than one connection are planned for the current job.
	Close Connection	Clear this check box to continue to use the selected connection once the component has performed its task.
Usage	This component is to be used along with Postgresql components, especially with **tPostgresqlConnection** and **tPostgresqlCommit** components.	
Limitation	n/a	

Related scenario

This component is closely related to **tPostgresqlConnection** and **tPostgresqlCommit**. It usually doesn't make much sense to use one of them without using a **tPostgresqlConnection** component to open a connection for the current transaction.

For **tPostgresqlRollback** related scenario, see *tMysqlRollback on page 299*.

tPostgresqlRow

tPostgresqlRow properties

Component family	Databases/Postgresql	
Function	**tPostgresqlRow** is the specific component for the database query. It executes the SQL query stated onto the specified database. The row suffix means the component implements a flow in the job design although it doesn't provide output.	
Purpose	Depending on the nature of the query and the database, **tPostgresqlRow** acts on the actual DB structure or on the data (although without handling data). The SQLBuilder tool helps you write easily your SQL statements.	
Basic settings	*Property type*	Either Built-in or Repository.
		Built-in: No property data stored centrally.
		Repository: Select the repository file where Properties are stored. The fields that come after are pre-filled in using the fetched data.
	Use an existing connection	Select this check box and click the relevant **tPostgresqlConnection** component on the Component list to reuse the connection details you already defined.
	Host	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database
	Schema	Exact name of the schema
	Username and *Password*	DB user authentication data.
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either built-in or remotely stored in the Repository.
		Built-in: The schema is created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	Query type	Either Built-in or Repository.
		Built-in: Fill in manually the query statement or build it graphically using SQLBuilder
		Repository: Select the relevant query stored in the Repository. The Query field gets accordingly filled in.

	Query	Enter your DB query paying particularly attention to properly sequence the fields in order to match the schema definition.
	Die on error	Clear this check box to skip the row on error and complete the process for error-free rows.
Advanced settings	*Propagate QUERY's recordset*	Select this check box to insert the result of the query into a COLUMN of the current flow.
	Encoding Type	Select the encoding from the list or select **Custom** and define it manually. This field is compulsory for DB data handling.
	Commit every	Number of rows to be completed before committing batches of rows together into the DB. This option ensures transaction quality (but not rollback) and above all better performance on executions.
	*tState*Catcher Statistics	Select this check box to collect log data at the component level.
Usage	This component offers the flexibility benefit of the DB query and covers all possibilities of SQL queries.	

Related scenarios

For related topics, see:

- **tDBSQLRow** *Scenario: Resetting a DB auto-increment on page 172*
- **tMySQLRow** *Scenario: Removing and regenerating a MySQL table index on page 301*.

tSASInput

Before being able to benefit from all functional objectives of the SAS components, make sure to install the following three modules: sas.core.jar, sas.intrnet.javatools.jar and sas.svc.connection.jar in the path lib > java in your Talend Open Studio directory. You can later verify, if needed whether the modules are successfully installed through the Modules view of the Studio.

tSASInput properties

Component family	Databases/SAS	
Function	**tSASInput** reads a database and extracts fields based on a query.	
Purpose	**tSASInput** executes a DB query with a strictly defined statement which must correspond to the schema definition. Then it passes on the field list to the component that follows via a **Main** row link.	
Basic settings	*Property type*	Either **Built-in** or **Repository**.
		Built-in: No property data stored centrally.
		Repository: Select the repository file where Properties are stored. The fields that come after are pre-filled in using the fetched data.
		Click this icon to open a database connection wizard and store the database connection parameters you set in the component **Basic settings** view. For more information about setting up and storing database connection parameters, see *Setting up a DB connection* of Talend Open Studio User Guide.
	Host name	SAS server IP address.
	Port	Listening port number of server.
	Librefs	Enter the directory name that holds the table to read followed by its access path. For example: `"TpSas 'C:/SAS/TpSas'"`
	Username and *Password*	DB user authentication data.
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either built-in or remotely stored in the Repository.
		Built-in: The schema is created and stored locally for this component only. Related topic: *Setting a built-in schema* of Talend Open Studio User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused. Related topic: *Setting a Repository schema* of Talend Open Studio User Guide.
	Table Name	Enter the name of the table to read preceded by the directory name that holds it. For example: `"TpSas.Customers"`.

	Query type	The query can be built-in for a particular job or for commonly used query, it can be stored in the repository to ease the query reuse.
	Query	If your query is not stored in the Repository, type in your DB query paying particularly attention to properly sequence the fields in order to match the schema definition.
Advanced settings	*tStateCatcher Statistics*	Select this check box to gather the job processing metadata at a job level as well as at each component level.
Usage	This component covers all possibilities of SQL queries onto a database using an SAS connection.	

Related scenarios

For related topics, see **tDBInput** scenarios:

- *Scenario 1: Displaying selected data from DB table on page 163.*
- *Scenario 2: Using StoreSQLQuery variable on page 164.*

See also the related topic in **tContextLoad** *Scenario: Dynamic context use in MySQL DB insert on page 652*.

tSASOutput

*Before being able to benefit from all functional objectives of the SAS components, make sure to install the following three modules: sas.core.jar, sas.intrnet.javatools.jar and sas.svc.connection.jar in the path lib > java in your **Talend Open Studio** directory. You can later verify, if needed whether the modules are successfully installed through the Modules view of the Studio.*

tSASOutput properties

Component family	Databases/SAS	
Function	**tSASOutput** writes, updates, makes changes or suppresses entries in a database.	
Purpose	**tSASOutput** executes the action defined on the table and/or on the data contained in the table, based on the incoming flow from the preceding component in the Job.	
Basic settings	*Use an existing connection*	Select this check box and click the relevant **tSASConnection** component on the **Component** list to reuse the connection details you already defined.
	SAS URL	Enter the URL to connect to the desired DB.
	Driver JAR	In the drop down list, select a desired available driver, or download one from a local directory through clicking the three-dot button.
	Class Name	Type in the Class name to be pointed to in the driver.
	Username and *Password*	DB user authentication data.
	Table	Name of the table to read.

	Action on data	On the data of the table defined, you can perform: **Insert**: Add new entries to the table. If duplicates are found, job stops. **Update**: Make changes to existing entries **Insert or update**: Add entries or update existing ones. **Update or insert**: Update existing entries or create it if non existing **Delete**: Remove entries corresponding to the input flow. *It is necessary to specify at least one column as a primary key on which the Update and Delete operations are based. You can do that by clicking Edit Schema and selecting the check box(es) next to the column(s) you want to set as primary key(s). For an advanced use, click the Advanced settings view where you can define primary keys simultaneously for the Update and Delete operations. To do that: Select the Use field options check box and then in the "Key in update" column, select the check boxes next to the column names you want to use as a base for the Update operation. Do the same in the "Key in delete" column for the Delete operation.*
	Clear data in table	Select this check box to delete data in the selected table before any operation.
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either built-in or remotely stored in the Repository.
		Built-in: The schema is created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	Die on error	Clear this check box to skip the row on error and complete the process for error-free rows.
Advanced settings	*Encoding Type*	Select the encoding type from the list or select **Custom** and define it manually. This field is compulsory for DB data handling.
	Commit every	Enter the number of rows to be completed before committing batches of rows together into the DB. This option ensures transaction quality (but not rollback) and, above all, better performance at execution.

	Additional Columns	This option is not offered if you create (with or without drop) the DB table. This option allows you to call SQL functions to perform actions on columns, which are not insert, nor update or delete actions, or action that require particular preprocessing.
		Name: Type in the name of the schema column to be altered or inserted as a new column.
		SQL expression: Type in the SQL statement to be executed in order to alter or insert the relevant column data.
		Position: Select **Before**, **Replace** or **After** following the action to be performed on the reference column.
		Reference column: Type in a column of reference that the **tSASOutput** can use to place or replace the new or altered column.
	Use field options	Select this check box to customize a request, especially when there is double action on data.
	Enable debug mode	Select this check box to display each step during processing entries in a database.
	tStatCatcher Statistics	Select this check box to collect log data at the component level.
Usage	This component offers the flexibility benefit of a connection to any type of DB and covers all possibilities of SQL queries.	

Related scenarios

For **tJDBCOutput** related topics, see:

- **tDBOutput** *Scenario: Displaying DB output on page 168*
- **tMySQLOutput** *Scenario 1: Adding a new column and altering data in a DB table on page 283*.

tSQLiteConnection

SQLiteConnection properties

This component is closely related to **tSQLiteCommit** and **tSQLiteRollback**. It usually does not make much sense to use one of these without using a **tSQLiteConnection** to open a connection for the current transaction.

Component family	Databases/SQLite	
Function	**tSQLiteConnection** opens a connection to the database for a current transaction.	
Purpose	Allows to commit a whole job data in one go to the output database as one transaction when validated.	
Basic settings	*Property type*	Either **Built-in** or **Repository**.
		Built-in: No property data stored centrally.
		Repository: Select the repository file where Properties are stored. The fields that come after are pre-filled in using the fetched data.
	Database	Name of the database.
	Use or register a shared DB Connection	Select this check box to share your connection or fetch a connection shared by a parent or child Job. shared connection name. **Shared DB Connection Name:** set or type in the shared connection name.
Advanced settings	*Auto commit*	Select this check box to automatically commit a transaction when it is completed.
	tStatCatcher Statistics	Select this check box to gather the job processing metadata at a Job level as well as at each component level.
Usage	This component is to be used along with SQLite components, especially with **tSQLiteCommit** and **tSQLiteRollback**.	
Limitation	n/a	

Related scenarios

This component is closely related to **tSQLiteCommit** and **tSQLiteRollback**. It usually does not make much sense to use one of these without using a **tSQLiteConnection** component to open a connection for the current transaction.

For **tSQLiteConnection** related scenario, see *tMysqlConnection on page 269*.

tSQLiteInput

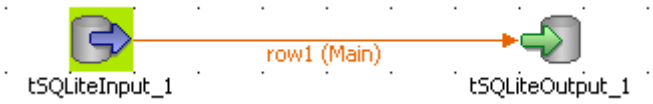
tSQLiteInput Properties

Component family	Databases	
Function	**tSQLiteInput** reads a database file and extracts fields based on an SQL query. As it embeds the SQLite engine, no need of connecting to any database server.	
Purpose	**tSQLiteInput** executes a DB query with a defined command which must correspond to the schema definition. Then it passes on rows to the next component via a Main row link.	
Basic settings	*Property type*	Either Built-in or Repository
		Built-in: No property data stored centrally.
		Repository: Select the repository file where Properties are stored. The fields that come after are pre-filled in using the fetched data.
	Use an existing connection	Select this check box and click the relevant **tSQLiteConnection** component on the Component list to reuse the connection details you already defined.
		Click this icon to open a database connection wizard and store the database connection parameters you set in the component **Basic settings** view. For more information about setting up and storing database connection parameters, see *Setting up a DB connection* of **Talend Open Studio** User Guide.
	Database	Filepath to the SQLite database file.
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either built-in or remotely stored in the Repository.
		Built-in: The schema is created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	Query type	The query can be built-in for a particular job or for commonly used query, it can be stored in the repository to ease the query reuse.
	Query	If your query is not stored in the Repository, type in your DB query paying particularly attention to properly sequence the fields in order to match the schema definition.

Advanced settings	*Encoding Type*	Select the encoding type from the list or select **Custom** and define it manually. This field is compulsory for DB data handling.
	Trim all the String/Char columns	Select this check box to remove leading and trailing whitespace from all the String/Char columns.
	Trim column	Remove leading and trailing whitespace from defined columns.
	*tState*Catcher Statistics	Select this check box to collect log data at the component level.
Usage	This component is standalone as it includes the SQLite engine. This is a startable component that can iniate a data flow processing.	

Scenario: Filtering SQlite data

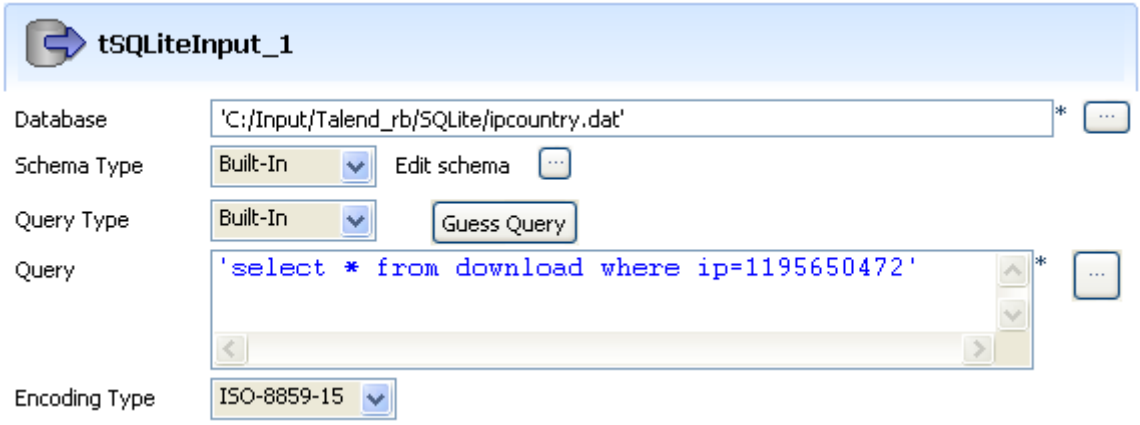
This scenario describes a rather simple job which uses a select statement based on a filter to extract rows from a source SQLite Database and feed an output SQLite table.



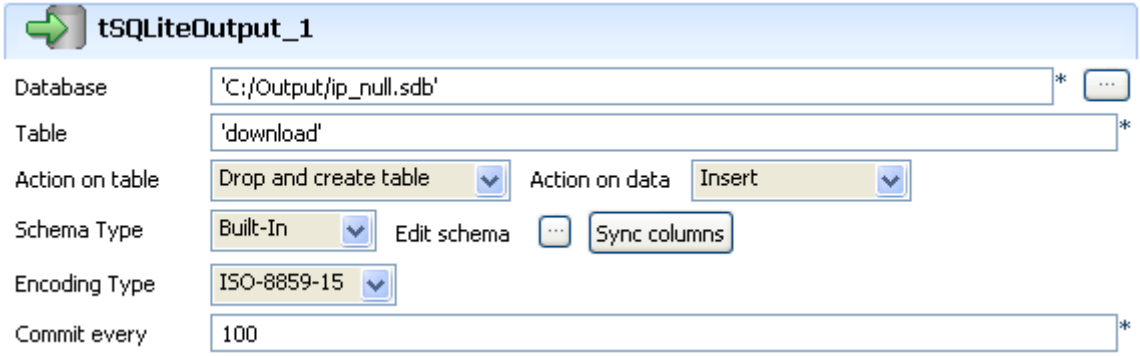
- Drop from the **Palette**, a **tSQLiteInput** and a **tSQLiteOutput** component from the **Palette** to the design workspace.
- Connect the input to the output using a row main link.
- On the **tSQLiteInput** Basic settings, type in or browse to the SQLite Database input file.

	id	version	download_date	ip	type	type_os
1	20027	TOS-Win32-200610	13/11/2006	1191947907	1	101
2	20028	TOS-Win32-200610	13/11/2006	1195650472	1	102
3	20030	TOS-Win32-200610	13/11/2006	3375565745	1	103
4	20031	TOS-Win32-200610	13/11/2006	1195650472	1	104
5	20032	TOS-Win32-200610	13/11/2006	1195650472	1	105
6	20033	TOS-Win32-200610	13/11/2006	1104872453	1	106
7	20034	TOS-Win32-200610	13/11/2006	1104872453	1	107
8	20036	TOS-Win32-200610	13/11/2006	1190898057	1	108
9	20037	TOS-Win32-200610	13/11/2006	1190898057	1	109
10	20038	TOS-Win32-200610	13/11/2006	1348977142	1	110
11	20040	TOS-Win32-200610	13/11/2006	3581349521	1	11
12	20041	TOS-Win32-200610	13/11/2006	1190898057	1	12
13	20043	TOS-Win32-200610	13/11/2006	1196485544	1	13
14	20044	TOS-Win32-200610	13/11/2006	1066743463	1	14
15	20045	TOS-Win32-200610	13/11/2006	1196485544	1	15
16	20046	TOS-Win32-200610	13/11/2006	3624217794	1	16

- The file contains hundreds of lines and includes an **ip** column which the select statement will based on
- On the **tSQLite** Basic settings, edit the schema for it to match the table structure.



- In the **Query** field, type in your select statement based on the *ip* column.
- Select the right encoding parameter.
- On the **tSQLiteOutput** component **Basic settings** panel, select the **Database** filepath.



- Type in the **Table** to be fed with the selected data.
- Select the **Action on table** and **Action on Data**. In this use case, the action on table is *Drop and create* and the action on data is *Insert*.
- The schema should be synchronized with the input schema.
- Select the encoding and define the threshold to commit.
- Save the job and run it.

	id	version	download_date	ip	type	type_os
1	20028	TOS-Win32-200610	13/11/2006	1195650472	1	102
2	20031	TOS-Win32-200610	13/11/2006	1195650472	1	104
3	20032	TOS-Win32-200610	13/11/2006	1195650472	1	105

The queried data are returned in the defined SQLite file.

tSQLiteOutput

tSQLiteOutput Properties

Component family	Databases	
Function	**tSQLiteOutput** writes, updates, makes changes or suppresses entries in an SQLite database. As it embeds the SQLite engine, no need of connecting to any database server.	
Purpose	**tSQLiteOutput** executes the action defined on the table and/or on the data contained in the table, based on the flow incoming from the preceding component in the job.	
Basic settings	*Property type*	Either Built-in or Repository.
		Built-in: No property data stored centrally.
		Repository: Select the repository file where Properties are stored. The fields that come after are pre-filled in using the fetched data.
	Use an existing connection	Select this check box and click the relevant **tSQLiteConnection** component on the Component list to reuse the connection details you already defined.
		Click this icon to open a database connection wizard and store the database connection parameters you set in the component **Basic settings** view. For more information about setting up and storing database connection parameters, see *Setting up a DB connection* of **Talend Open Studio** User Guide.
	Database	Filepath to the Database file
	Table	Name of the table to be written. Note that only one table can be written at a time
	Action on table	On the table defined, you can perform one of the following operations: **None**: No operation is carried out. **Drop and create a table**: The table is removed and created again. **Create a table**: The table does not exist and gets created. **Create a table if not exists**: The table is created if it does not exist. **Drop a table if exists and create**: The table is removed if it already exists and created again. **Clear a table**: The table content is deleted.

	Action on data	On the data of the table defined, you can perform: **Insert**: Add new entries to the table. If duplicates are found, job stops. **Update**: Make changes to existing entries **Insert or update**: Add entries or update existing ones. **Update or insert**: Update existing entries or create it if non existing **Delete**: Remove entries corresponding to the input flow. *It is necessary to specify at least one column as a primary key on which the Update and Delete operations are based. You can do that by clicking Edit Schema and selecting the check box(es) next to the column(s) you want to set as primary key(s). For an advanced use, click the Advanced settings view where you can simultaneously define primary keys for the Update and Delete operations. To do that: Select the Use field options check box and then in the Key in update column, select the check boxes next to the column names you want to use as a base for the Update operation. Do the same in the Key in delete column for the Delete operation.*
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either built-in or remotely stored in the Repository.
		Built-in: The schema is created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	Die on error	Clear this check box to skip the row on error and complete the process for error-free rows.
Advanced settings	*Encoding Type*	Select the encoding type from the list or select **Custom** and define it manually. This field is compulsory for DB data handling.
	Commit every	Enter the number of rows to be completed before committing batches of rows together into the DB. This option ensures transaction quality (but not rollback) and, above all, better performance at execution.
	Additional Columns	This option is not offered if you create (with or without drop) the DB table. This option allows you to call SQL functions to perform actions on columns, which are not insert, nor update or delete actions, or action that require particular preprocessing.

		Name: Type in the name of the schema column to be altered or inserted as new column
		SQL expression: Type in the SQL statement to be executed in order to alter or insert the relevant column data.
		Position: Select **Before**, **Replace** or **After** following the action to be performed on the reference column.
		Reference column: Type in a column of reference that the **tDBOutput** can use to place or replace the new or altered column.
	*Use field option*s	Select this check box to customize a request, especially when there is double action on data.
	Enable debug mode	Select this check box to display each step during processing entries in a database.
	*tState*Catcher Statistics	Select this check box to collect log data at the component level.
Usage	This component is requried to be connected to an Input component.	

Related Scenario

For scenarios related to **tSQLiteOutput**, see *tSQLiteInput on page 378.*

tSQLiteRow

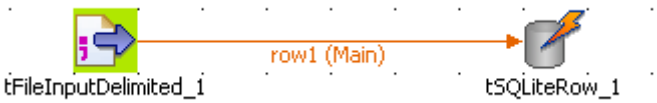
tSQLiteRow Properties

Component family	Databases	
Function	**tSQLiteRow** executes the defined query onto the specified database and uses the parameters bound with the column.	
Purpose	A prepared statement uses the input flow to replace the placeholders with the values for each parameters defined. This component can be very useful for updates.	
Basic settings	*Property type*	Either Built-in or Repository.
		Built-in: No property data stored centrally.
		Repository: Select the repository file where Properties are stored. The fields that come after are pre-filled in using the fetched data.
	Use an existing connection	Select this check box and click the relevant **tSQLiteConnection** component on the Component list to reuse the connection details you already defined.
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either built-in or remotely stored in the Repository.
		Built-in: The schema is created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	Query type	Either Built-in or Repository.
		Built-in: Fill in manually the query statement or build it graphically using SQLBuilder
		Repository: Select the relevant query stored in the Repository. The Query field gets accordingly filled in.
	Query	Enter your DB query paying particularly attention to properly sequence the fields in order to match the schema definition.
	Die on error	Clear this check box to skip the row on error and complete the process for error-free rows.
Advanced settings	*Propagate QUERY's recordset*	Select this check box to insert the result of the query into a COLUMN of the current flow.
	Encoding Type	Select the encoding from the list or select **Custom** and define it manually. This field is compulsory for DB data handling.

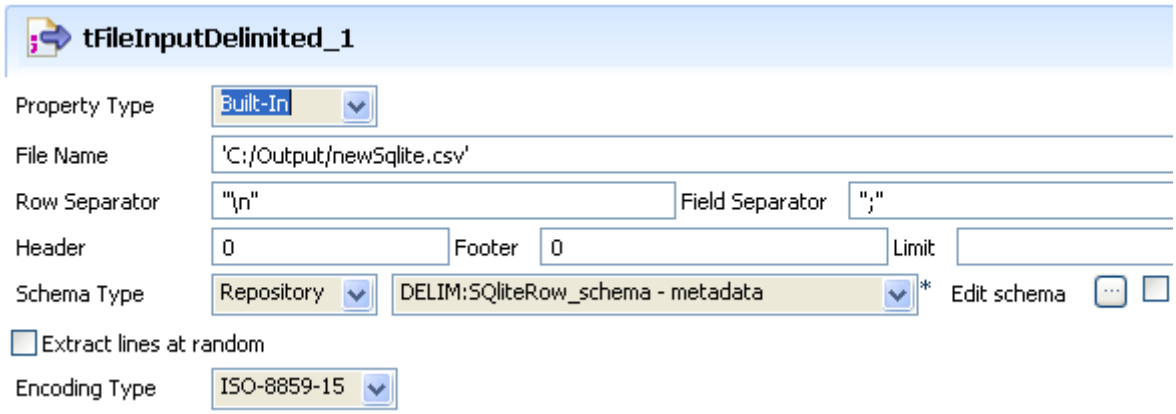
	Commit every	Number of rows to be completed before committing batches of rows together into the DB. This option ensures transaction quality (but not rollback) and above all better performance on executions.
	*tState*Catcher Statistics	Select this check box to collect log data at the component level.
Usage	This component offers the flexibility benefit of the DB query and covers all possibilities of SQL queries.	

Scenario: Updating SQLite rows

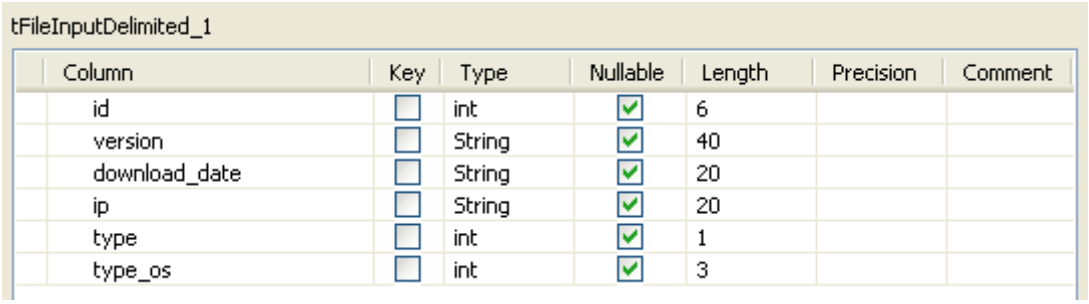
This scenario describes a job which updates an SQLite database file based on a prepared statement and using a delimited file.



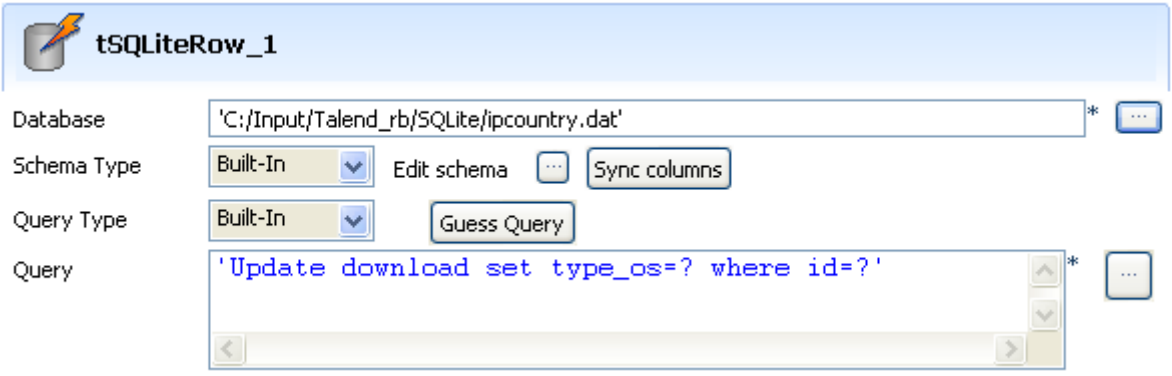
- Drop a **tFileInputDelimited** and a **tSQLiteRow** component from the **Palette** to the design workspace.
- On the **tFileInputDelimited Basic settings** panel, browse to the input file that will be used to update rows in the database.



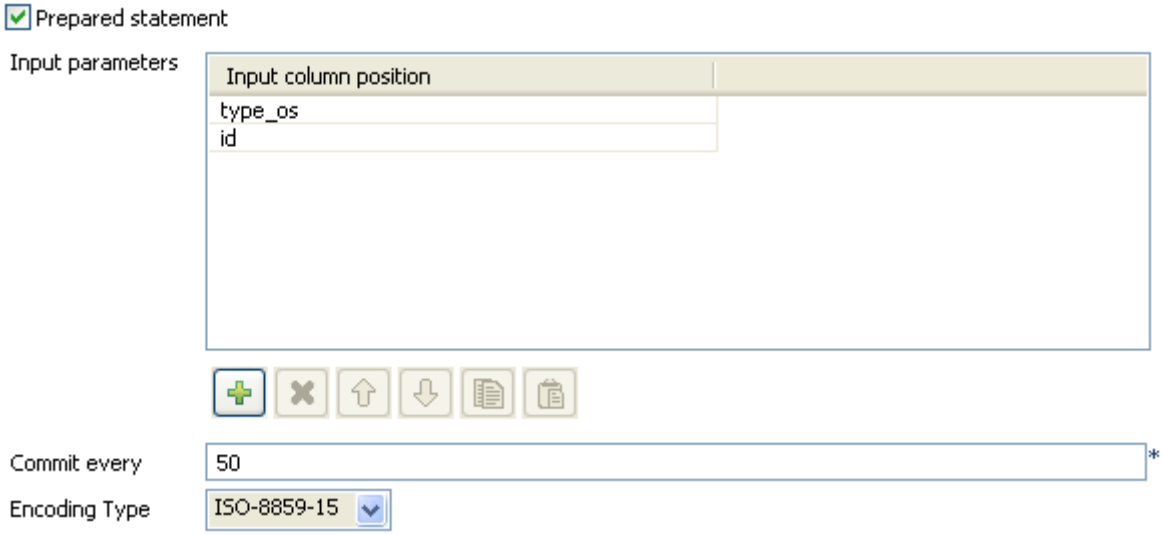
- There is no header nor footer. The Row separator is a carriage return and the field separator is a semi-colon.
- Edit the schema in case it is not stored in the Repository.



- Make sure the length and type are respectively correct and large enough to define the columns.
- Then in the **tSQLiteRow Basic settings** panel, set the **Database** filepath to the file to be updated.



- The schema is read-only as it is required to match the input schema.
- Type in the query or retrieve it from the Repository. In this use case, we updated the *type_os* for the *id* defined in the Input flow. The statement is as follows: 'Update download set type_os=? where id=?'
- Then select the **Prepared statement** check box to display the placeholders' parameter table.



- In the Input parameters table, add as many lines as necessary to cover all placeholders. In this scenario, *type_os* and *id* are to be defined.

- Set the **Commit every** field and select the **Encoding type** in the list.
- Save the job and press **F6** to run it.

The *dowload* table from the SQLite database is thus updated with new *type_os* code according to the delimited input file.

tSybaseBulkExec

tSybaseBulkExec Properties

Component family	Databases	
Function	Executes the Insert action on the data provided.	
Purpose	As a dedicated component, it allows gains in performance during Insert operations to a Sybase database.	
Basic settings	*Property type*	Either **Built-in** or **Repository**.
		Built-in: No property data stored centrally.
		Repository: Select the repository file where Properties are stored. The fields that come after are pre-filled in using the fetched data.
	Use an existing connection	Select this check box and click the relevant **tSybaseConnection** component on the Component list to reuse the connection details you already defined.
	Server	Database server IP address
	Port	Listening port number of DB server.
	Database	Database name
	Username and *Password*	DB user authentication data.
	Bcp Utility	Name of the utility to be used to copy data over to the Sybase server.
	Server	IP address of the database server for the Bcp utility connection.
	Batch size	Number of lines in each processed batch.
	Table	Name of the table to be written. Note that only one table can be written at a time and that the table must exist for the insert operation to succeed.
	Action on table	On the table defined, you can perform one of the following operations: **None**: No operation is carried out. **Drop and create a table**: The table is removed and created again. **Create a table**: The table does not exist and gets created. **Create a table if not exists**: The table is created if it does not exist. **Clear a table**: The table content is deleted. **Truncate table**: The table content is deleted. You do not have the possibility to rollback the operation.
	File Name	Name of the file to be processed. Related topic:*Defining variables from the Component view* of **Talend Open Studio** User Guide.

	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either built-in or remotely stored in the Repository.
		Built-in: You create and store the schema locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: You have already created and stored the schema in the Repository, hence can be reused. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
Advanced settings	*Action on data*	On the data of the table defined, you can perform: **Bulk Insert**: Add multiple entries to the table. If duplicates are found, job stops. **Bulk Update**: Make simultaneous changes to multiple entries.
	Field Terminator	Character, string or regular expression to separate fields.
	Row Terminator	String (ex: “\n” in Unix) to separate lines.
	Head row	Number of head lines to be ignored in the beginning of a file.
	Encoding Type	Select the encoding from the list or select **Custom** and define it manually. This field is compulsory for DB data handling.
	Output	Select the type of output for the standard output of the Sybase database: **to console**, **to global variable**.
	tStataCatcher statistics	Select this check box to gather the job processing metadata at a job level as well as at each component level.
Usage	This component is mainly used when no particular transformation is required on the data to be loaded onto the database.	
Limitation	As opposed to the Oracle dedicated bulk component, no action on data is possible using this Sybase dedicated component.	

Related scenarios

For **tSybaseBulkExec** related topics, see:

- **tMysqlOutputBulkExec** *Scenario: Inserting transformed data in MySQL database on page 293*
- **tOracleBulkExec** *Scenario: Truncating and inserting file data into Oracle DB on page 329*.

tSybaseCommit

tSybaseCommit Properties

This component is closely related to **tSybaseConnection** and **tSybaseRollback**. It usually does not make much sense to use these components independently in a transaction.

Component family	Databases/Sybase	 
Function	**tSybaseCommit** validates the data processed through the Job into the connected DB	
Purpose	Using a unique connection, this component commits in one go a global transaction instead of doing that on every row or every batch and thus provides gain in performance.	
Basic settings	*Component list*	Select the **tSybaseConnection** component in the list if more than one connection are planned for the current Job.
	Close Connection	Clear this check box to continue to use the selected connection once the component has performed its task.
Usage	This component is to be used along with Sybase components, especially with **tSybaseConnection** and **tSybaseRollback**.	
Limitation	n/a	

Related scenario

This component is closely related to **tSybaseConnection** and **tSybaseRollback**. It usually does not make much sense to use one of these without using a **tSybaseConnection** component to open a connection for the current transaction.

For **tSybaseCommit** related scenario, see *Scenario: Inserting data in mother/daughter tables on page 269*.

tSybaseConnection

tSybaseConnection Properties

This component is closely related to **tSybaseCommit** and **tSybaseRollback**. It usually does not make much sense to use one of these without using a **tSybaseConnection** component to open a connection for the current transaction.

Component family	Databases/Sybase	
Function	**tSybaseConnection** opens a connection to the database for a current transaction.	
Purpose	This component allows to commit job data in one go to the output database as one transaction when validated.	
Basic settings	*Property type*	Either **Built-in** or **Repository**.
		Built-in: No property data stored centrally.
		Repository: Select the repository file where Properties are stored. The fields that come after are pre-filled in using the fetched data.
	Host	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database.
	Username and *Password*	DB user authentication data.
	Encoding Type	Select the encoding type from the list or select **Custom** and define it manually. This field is compulsory for DB data handling.
	Use or register a shared DB Connection	Select this check box to share your connection or fetch a connection shared by a parent or child Job. **Shared DB Connection Name:** set or type in the shared connection name.
Usage	This component is to be used along with Sybase components, especially with **tSybaseCommit** and **tSybaseRollback**.	
Limitation	n/a	

Related scenarios

For **tSybaseConnection** related scenario, see *Scenario: Inserting data in mother/daughter tables on page 269*.

tSybaseInput

tSybaseInput Properties

Component family	Databases/Sybase	
Function	**tSybaseInput** reads a database and extracts fields based on a query.	
Purpose	**tSybaseInput** executes a DB query with a strictly defined order which must correspond to the schema definition. Then it passes on the field list to the next component via a Main row link.	
Basic settings	*Property type*	Either Built-in or Repository
		Built-in: No property data stored centrally.
		Repository: Select the repository file where Properties are stored. The fields that come after are pre-filled in using the fetched data.
		Click this icon to open a database connection wizard and store the database connection parameters you set in the component **Basic settings** view. For more information about setting up and storing database connection parameters, see *Setting up a DB connection* of **Talend Open Studio** User Guide.
	Use an existing connection	Select this check box and click the relevant **tSybaseConnection** component on the Component list to reuse the connection details you already defined.
	Server	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database
	Sybase Schema	Exact name of the Sybase schema.
	Username and *Password*	DB user authentication data.
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either built-in or remotely stored in the Repository.
		Built-in: The schema is created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	Table Name	Name of the table to read.

	Query type and *Query*	Enter your DB query paying particularly attention to properly sequence the fields in order to match the schema definition.
Advanced settings	*Trim all the String/Char columns*	Select this check box to remove leading and trailing whitespace from all the String/Char columns.
	Trim column	Remove leading and trailing whitespace from defined columns.
	*tState*Catcher Statistics	Select this check box to collect log data at the component level.
Usage	This component covers all possibilities of SQL queries onto a Sybase database.	

Related scenarios

Related topic in **tDBInput** scenarios:

- *Scenario 1: Displaying selected data from DB table on page 163*
- *Scenario 2: Using StoreSQLQuery variable on page 164*

Related topic in **tContextLoad** *Scenario: Dynamic context use in MySQL DB insert on page 652*.

tSybaseIQBulkExec

tSybaseIQBulkExec Properties

Component family	Databases/Sybase IQ	
Function	**tSybaseIQBulkExec** uploads a bulk file in a Sybase IQ database.	
Purpose	As a dedicated component, it allows gains in performance during Insert operations to a Sybase IQ database.	
Basic settings	*Property type*	Either **Built-in** or **Repository**.
		Built-in: No property data stored centrally.
		Repository: Select the repository file where Properties are stored. The fields that come after are pre-filled in using the fetched data.
	Use an existing connection	Select this check box and click the relevant **tSybaseConnection** component on the **Component List** to reuse the connection details you already defined.
	Host	Database server IP address
	Port	Listening port number of DB server.
	Database	Database name
	Username and *Password*	DB user authentication data.
	Table	Name of the table to be written. Note that only one table can be written at a time and that the table must exist for the insert operation to succeed.
	Action on table	On the table defined, you can perform one of the following operations: **None**: No operation is carried out. **Drop and create table**: The table is removed and created again. **Create table**: The table does not exist and gets created. **Create table if not exists**: The table is created if it does not exist. **Clear table**: The table content is deleted. **Truncate table**: The table content is deleted. You do not have the possibility to rollback the operation.
	Local filename	Name of the file to be processed. Related topic:*Defining variables from the Component view* of **Talend Open Studio** User Guide.
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either built-in or remotely stored in the Repository.

		Built-in: You create and store the schema locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: You have already created and stored the schema in the Repository, hence can be reused. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
Advanced settings	*Lines terminated by*	Character or sequence of characters used to separate lines.
	Field Terminated by	Character, string or regular expression to separate fields.
	Use enclosed quotes	Select this check box to use data enclosure characters.
	Use fixed length	Select this check box to set a fixed width for data lines.
	tStatCatcher Statistics	Select this check box to gather the job processing metadata at a job level as well as at each component level.
Usage	This dedicated component offers performance and flexibility of Sybase IQ DB query handling.	
Limitation	As opposed to the Oracle dedicated bulk component, no action on data is possible using this Sybase dedicated component.	

Related scenarios

For **tSybaseIQBulkExec** related topics, see:

- **tMysqlOutputBulkExec** *Scenario: Inserting transformed data in MySQL database on page 293*
- **tOracleBulkExec** *Scenario: Truncating and inserting file data into Oracle DB on page 329*.

tSybaseIQOutputBulkExec

tSybaseIQOutputBulkExec properties

Component family	Databases/Sybase IQ	
Function	Executes the Insert action on the data provided.	
Purpose	As a dedicated component, it allows gains in performance during Insert operations to a Sybase IQ database.	
Basic settings	*Property type*	Either **Built-in** or **Repository**.
		Built-in: No property data stored centrally.
		Repository: Select the repository file where Properties are stored. The fields that come after are pre-filled in using the fetched data.
	Use an existing connection	Select this check box and click the relevant **tSybaseConnection** component on the Component list to reuse the connection details you already defined.
	Host	Database server IP address.
	Port	Listening port number of DB server.
	Database	Name of the database
	Username and *Password*	DB user authentication data.
	Table	Name of the table to be written. Note that only one table can be written at a time and that the table must exist for the insert operation to succeed.
	Action on table	On the table defined, you can perform one of the following operations: **None**: No operation is carried out. **Drop and create a table**: The table is removed and created again. **Create a table**: The table does not exist and gets created. **Create a table if not exists**: The table is created if it does not exist. **Clear a table**: The table content is deleted.
	File Name	Name of the file to be processed. Related topic:*Defining variables from the Component view* of **Talend Open Studio** User Guide.
	Append the file	select this check box to add the new rows at the end of the records.
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either built-in or remotely stored in the Repository.

		Built-in: You create and store the schema locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: You have already created and stored the schema in the Repository, hence can be reused. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
Advanced settings	*Fields terminated by*	Character, string or regular expression to separate fields.
	Lines terminated by	Character or sequence of characters used to separate lines.
	Use enclose quotes	Select this check box to use data enclosure characters.
	Include Head	Select this heck box to include the column header.
	Encoding Type	Select the encoding type from the list or select **Custom** and define it manually. This field is compulsory for DB data handling.
	tStatCatcher Statistics	Select this check box to collect log data at the component level.
Usage	This component is mainly used when no particular transformation is required on the data to be loaded onto the database.	
Limitation	n/a	

Related scenarios

For use cases in relation with **tSybaseIQOutputBulkExec**, see the following scenarios:

- **tMysqlOutputBulk** *Scenario: Inserting transformed data in MySQL database on page 293*.
- **tMysqlOutputBulkExec** *Scenario: Inserting data in MySQL database on page 297*.
- **tOracleBulkExec** *Scenario: Truncating and inserting file data into Oracle DB on page 329*.

tSybaseOutput

tSybaseOutput Properties

Component family	Databases/Sybase	
Function	**tSybaseOutput** writes, updates, makes changes or suppresses entries in a database.	
Purpose	**tSybaseOutput** executes the action defined on the table and/or on the data contained in the table, based on the flow incoming from the preceding component in the job.	
Basic settings	*Property type*	Either Built-in or Repository.
		Built-in: No property data stored centrally.
		Repository: Select the repository file where Properties are stored. The fields that come after are pre-filled in using the fetched data.
	Use an existing connection	Select this check box and click the relevant **tJSybaseConnection** component on the Component list to reuse the connection details you already defined.
		Click this icon to open a database connection wizard and store the database connection parameters you set in the component **Basic settings** view. For more information about setting up and storing database connection parameters, see *Setting up a DB connection* of **Talend Open Studio** User Guide.
	Server	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database
	Sybase Schema	Exact name of the Sybase schema.
	Username and *Password*	DB user authentication data.
	Table	Name of the table to be written. Note that only one table can be written at a time
	Action on table	On the table defined, you can perform one of the following operations: **None**: No operation is carried out. **Drop and create a table**: The table is removed and created again. **Create a table**: The table does not exist and gets created. **Create a table if not exists**: The table is created if it does not exist. **Drop a table if exists and create**: The table is removed if it already exists and created again. **Clear a table**: The table content is deleted.

	Turn on identity insert	Select this check box to use your own sequence for the identity value of the inserted records (instead of having the SQL Server pick the next sequential value).
	Action on data	On the data of the table defined, you can perform: **Insert**: Add new entries to the table. If duplicates are found, job stops. **Update**: Make changes to existing entries **Insert or update**: Add entries or update existing ones. **Update or insert**: Update existing entries or create it if non existing **Delete**: Remove entries corresponding to the input flow. *It is necessary to specify at least one column as a primary key on which the Update and Delete operations are based. You can do that by clicking Edit Schema and selecting the check box(es) next to the column(s) you want to set as primary key(s). For an advanced use, click the Advanced settings view where you can simultaneously define primary keys for the Update and Delete operations. To do that: Select the Use field options check box and then in the Key in update column, select the check boxes next to the column names you want to use as a base for the Update operation. Do the same in the Key in delete column for the Delete operation.*
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either built-in or remotely stored in the Repository.
		Built-in: The schema is created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	Die on error	Clear this check box to skip the row on error and complete the process for error-free rows.
Advanced settings	*Commit every*	Enter the number of rows to be completed before committing batches of rows together into the DB. This option ensures transaction quality (but not rollback) and, above all, better performance at execution.
	Additional Columns	This option is not offered if you create (with or without drop) the DB table. This option allows you to call SQL functions to perform actions on columns, which are not insert, nor update or delete actions, or action that require particular preprocessing.

		Name: Type in the name of the schema column to be altered or inserted as new column
		SQL expression: Type in the SQL statement to be executed in order to alter or insert the relevant column data.
		Position: Select **Before**, **Replace** or **After** following the action to be performed on the reference column.
		Reference column: Type in a column of reference that the **tDBOutput** can use to place or replace the new or altered column.
	Use field options	Select this check box to customize a request, especially when there is double action on data.
	Enable debug mode	Select this check box to display each step during processing entries in a database.
	tStateCatcher Statistics	Select this check box to collect log data at the component level.
Usage	This component offers the flexibility benefit of the DB query and covers all possibilities of SQL queries.	

Related scenarios

For use cases in relation with **tSybaseOutput**, see:

- **tDBOutput** *Scenario: Displaying DB output on page 168*
- **tMySQLOutput** *Scenario 1: Adding a new column and altering data in a DB table on page 283*.

tSybaseOutputBulk

tSybaseOutputBulk properties

tSybaseOutputBulk and **tSybaseBulkExec** components are used together to first output the file that will be then used as parameter to execute the SQL query stated. These two steps compose the **tSybaseOutputBulkExec** component, detailed in a separate section. The interest in having two separate elements lies in the fact that it allows transformations to be carried out before the data loading.

Component family	Databases/Sybase	
Function	Writes a file with columns based on the defined delimiter and the Sybase standards	
Purpose	Prepares the file to be used as parameter in the INSERT query to feed the Sybase database.	
Basic settings	*Property type*	Either **Built-in** or **Repository**.
		Built-in: No property data stored centrally.
		Repository: Select the Repository file where you have stored Properties. The fields that come after are pre-filled in using fetched data.
	File Name	Name of the file to be processed. Related topic:*Defining variables from the Component view* of **Talend Open Studio** User Guide.
	Append	Select this check box to add the new rows at the end of the file.
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the Repository.
		Built-in: You create and store the schema locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: You have already created and stored the schema in the Repository, hence can be reused in various projects and job designs. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
Advanced settings	*Row separator*	String (ex: “\n”on Unix) to distinguish rows.
	Field separator	Character, string or regular expression to separate fields.
	Include header	Select this check box to include the column header in the file.

	Encoding Type	Select the encoding from the list or select Custom and define it manually. This field is compulsory for DB data handling.
	tStateCatcher Statistics	Select this check box to collect log data at the component level
Usage	This component is to be used along with **tSybaseBulkExec** component. Used together they offer gains in performance while feeding a Sybase database.	

Related scenarios

For uses cases in relation with **tSybaseOutputBulk**, see the following scenarios:

- **tMysqlOutputBulk** *Scenario: Inserting transformed data in MySQL database on page 293*
- **tMysqlOutputBulkExec** *Scenario: Inserting data in MySQL database on page 297*
- **tOracleBulkExec** *Scenario: Truncating and inserting file data into Oracle DB on page 329*

tSybaseOutputBulkExec

tSybaseOutputBulkExec properties

Component family	Databases/Sybase	
Function	Executes the Insert action on the data provided.	
Purpose	As a dedicated component, it allows gains in performance during Insert operations to a Sybase database.	
Basic settings	*Property type*	Either Built-in or Repository.
		Built-in: No property data stored centrally.
		Repository: Select the repository file where Properties are stored. The fields that come after are pre-filled in using the fetched data.
	Use an existing connection	Select this check box and click the relevant **tSybaseConnection** component on the Component list to reuse the connection details you already defined.
	Server	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database
	Username and *Password*	DB user authentication data.
	Bcp utility	Name of the utility to be used to copy data over to the Sybase server.
	Batch row number	Number of lines in each processed batch.
	Table	Name of the table to be written. Note that only one table can be written at a time and that the table must exist for the insert operation to succeed.
	Action on table	On the table defined, you can perform one of the following operations: **None**: No operation is carried out. **Drop and create a table**: The table is removed and created again. **Create a table**: The table does not exist and gets created. **Create a table if not exists**: The table is created if it does not exist. **Clear a table**: The table content is deleted.
	File Name	Name of the file to be processed. Related topic:*Defining variables from the Component view* of **Talend Open Studio** User Guide.
	Append	Select this check box to add the new rows at the end of the records.

	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either built-in or remotely stored in the Repository.
		Built-in: You create and store the schema locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: You have already created and stored the schema in the Repository, hence can be reused. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
Advanced settings	*Action on data*	On the data of the table defined, you can perform: **Bulk Insert**: Add multiple entries to the table. If duplicates are found, job stops. **Bulk Update**: Make simultaneous changes to multiple entries.
	Field terminator	Character, string or regular expression to separate fields.
	DB Row terminator	String (ex: “\n”on Unix) to distinguish rows in the DB.
	First row	Type in the number of the file row where the action should start at.
	FILE Row terminator	Character, string or regular expression to separate fields in a file.
	Include Head	Select this heck box to include the column header.
	Encoding Type	Select the encoding from the list or select Custom and define it manually. This field is compulsory for DB data handling.
	Output	Select the type of output for the standard output of the Sybase database: **to console,** **to global variable.**
	tStatCatcher Statistics	Select this check box to collect log data at the component level.
Usage	This component is mainly used when no particular transformation is required on the data to be loaded onto the database.	
Limitation	n/a	

Related scenarios

For use cases in relation with **tSybaseOutputBulkExec**, see the following scenarios:

- **tMysqlOutputBulk** *Scenario: Inserting transformed data in MySQL database on page 293*
- **tMysqlOutputBulkExec** *Scenario: Inserting data in MySQL database on page 297*
- **tOracleBulkExec** *Scenario: Truncating and inserting file data into Oracle DB on page 329*

tSybaseRollback

tSybaseRollback properties

This component is closely related to **tSybaseCommit** and **tSybaseConnection**. It usually does not make much sense to use these components independently in a transaction.

Component family	Databases/Sybase	
Function	**tSybaseRollback** cancels the transaction committed in the connected DB.	
Purpose	This component avoids to commit part of a transaction involuntarily.	
Basic settings	*Component list*	Select the **tSybaseConnection** component in the list if more than one connection are planned for the current job.
	Close Connection	Clear this check box to continue to use the selected connection once the component has performed its task.
Usage	This component is to be used along with Sybase components, especially with **tSybaseConnection** and **tSybaseCommit**.	
Limitation	n/a	

Related scenarios

For **tSybaseRollback** related scenario, see *Scenario: Rollback from inserting data in mother/daughter tables on page 299*.

tSybaseRow

tSybaseRow Properties

Component family	Databases/Sybase	
Function	**tSybaseRow** is the specific component for this database query. It executes the SQL query stated onto the specified database. The row suffix means the component implements a flow in the job design although it doesn't provide output.	
Purpose	Depending on the nature of the query and the database, **tSybaseRow** acts on the actual DB structure or on the data (although without handling data). The SQLBuilder tool helps you write easily your SQL statements.	
Basic settings	*Property type*	Either Built-in or Repository.
		Built-in: No property data stored centrally.
		Repository: Select the repository file where Properties are stored. The fields that come after are pre-filled in using the fetched data.
	Use an existing connection	Select this check box and click the relevant **tSybaseConnection** component on the Component list to reuse the connection details you already defined.
	Server	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database
	Sybase Schema	Exact name of the sybase schema.
	Username and *Password*	DB user authentication data.
	Table Name	Name of the table to be processed.
	Turn on identity insert	Select this check box to use your own sequence for the identity value of the inserted records (instead of having the SQL Server pick the next sequential value).
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either built-in or remotely stored in the Repository.
		Built-in: The schema is created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	Query type	Either Built-in or Repository.

		Built-in: Fill in manually the query statement or build it graphically using SQLBuilder
		Repository: Select the relevant query stored in the Repository. The Query field gets accordingly filled in.
	Query	Enter your DB query paying particularly attention to properly sequence the fields in order to match the schema definition.
	Die on error	Clear this check box to skip the row on error and complete the process for error-free rows.
Advanced settings	*Propagate QUERY's recordset*	Select this check box to insert the result of the query into a COLUMN of the current flow.
	Commit every	Number of rows to be completed before committing batches of rows together into the DB. This option ensures transaction quality (but not rollback) and above all better performance on executions.
	*tState*Catcher Statistics	Select this check box to collect log data at the component level.
Usage	This component offers the flexibility benefit of the DB query and covers all possibilities of SQL queries.	

Related scenarios

For **tSybaseRow** related topics, see:

- **tDBSQLRow** *Scenario: Resetting a DB auto-increment on page 172*
- **tMySQLRow** *Scenario: Removing and regenerating a MySQL table index on page 301*.

tSybaseSCD

tSybaseSCD belongs to two component families: Business Intelligence and Databases. For more information on it, see *tSybaseSCD on page 25*.

tSybaseSCDELT

tSybaseSCDELT Properties

Component family	Databases/Sybase	
Function	**tSybaseSCDELT** reflects and tracks changes in a dedicated Sybase SCD table.	
Purpose	**tSybaselSCDELT** addresses Slowly Changing Dimension needs through SQL queries (server-side processing mode), and logs the changes into a dedicated Sybase SCD table.	
Basic settings	*Property type*	Either **Built-in** or **Repository**.
		Built-in: No property data stored centrally. Enter properties manually.
		Repository: Select the repository file where Properties are stored. The fields that come after are pre-filled in using the fetched data.
	Use an existing connection	Select this check box and click the relevant **tSybaseConnection** component on the **Component List** to reuse the connection details you already defined.
	Host	The IP address of the database server.
	Port	Listening port number of database server.
	Database	Name of the database
	Username and *Password*	User authentication data for a dedicated database.
	Source table	Name of the input MySQL SCD table.
	Table	Name of the table to be written. Note that only one table can be written at a time
	Action on table	Select to perform one of the following operations on the table defined: **None**: No action carried out on the table. **Drop and create table**: The table is removed and created again **Create table**: A new table gets created. **Create table if not exists**: A table gets created if it does not exist. **Clear table**: The table content is deleted. You have the possibility to rollback the operation. **Truncate table**: The table content is deleted. You don not have the possibility to rollback the operation.
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either built-in or remotely stored in the Repository.

		Built-in: The schema is created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	Surrogate Key	Select the surrogate key column from the list.
	Creation	Select the method to be used for the surrogate key generation.
	Source Key	Select one or more columns to be used as keys, to ensure the unicity of incoming data.
	Use SCD Type 1 fields	Use type 1 if tracking changes is not necessary. SCD Type 1 should be used for typos corrections for example. Select the columns of the schema that will be checked for changes.
	Use SCD Type 2 fields	Use type 2 if changes need to be tracked down. SCD Type 2 should be used to trace updates for example. Select the columns of the schema that will be checked for changes. **Start date**: Adds a column to your SCD schema to hold the start date value. You can select one of the input schema columns as Start Date in the SCD table. **End Date**: Adds a column to your SCD schema to hold the end date value for the record. When the record is currently active, the End Date column shows a null value, or you can select **Fixed Year value** and fill it in with a fictive year to avoid having a null value in the End Date field. **Log Active Status**: Adds a column to your SCD schema to hold the **true** or **false** status value. This column helps to easily spot the active record. **Log versions**: Adds a column to your SCD schema to hold the version number of the record.
Usage	This component is used as an output component. It requires an input component and Row main link as input.	

Related Scenario

For related topics, see *tMysqlSCD on page 12 Scenario: Tracking changes using Slowly Changing Dimensions (type 0 through type 3) on page 15*.

tSybaseSP

tSybaseSP properties

Component family	Databases/Sybase	
Function	**tSybaseSP** calls the database stored procedure.	
Purpose	**tSybaseSP** offers a convenient way to centralize multiple or complex queries in a database and call them easily.	
Basic settings	*Property type*	Either Built-in or Repository.
		Built-in: No property data stored centrally.
		Repository: Select the repository file where Properties are stored. The fields that come after are pre-filled in using the fetched data.
	Use an existing connection	Select this check box and click the relevant **tSybaseConnection** component on the Component list to reuse the connection details you already defined.
	Host	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database
	Username and *Password*	DB user authentication data.
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either built-in or remotely stored in the Repository.
		Built-in: The schema is created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	SP Name	Type in the exact name of the Stored Procedure
	Is Function / *Return result in*	Select this check box, if a value is to be returned. Select on the list the schema column, the value to be returned is based on.
	Timeout Interval	Maximum waiting time for the results of the stored procedure.

	Parameters	Click the Plus button and select the various **Schema Columns** that will be required by the procedures. Note that the SP schema can hold more columns than there are parameters used in the procedure. Select the **Type** of parameter: **IN**: Input parameter **OUT**: Output parameter/return value **IN OUT**: Input parameters is to be returned as value, likely after modification through the procedure (function). **RECORDSET**: Input parameters is to be returned as a set of values, rather than single value. Check the *tParseRecordSet* component if you want to analyze a set of records from a database table or DB query and return single records.
Advanced settings	*Encoding Type*	Select the encoding from the list or select Custom and define it manually. This field is compulsory for DB data handling.
	tStatCatcher Statistics	Select this check box to gather the job processing metadata at a job level as well as at each component level.
Usage	This component is used as intermediary component. It can be used as start component but only input parameters are thus allowed.	
Limitation	The Stored Procedures syntax should match the Database syntax.	

Related scenarios

For related topic, see **tMysqlSP** *Scenario: Finding a State Label using a stored procedure on page 307*.

Check as well the *tParseRecordSet* component if you want to analyze a set of records from a database table or DB query and return single records.

tTeradataInput

tTeradataInput Properties

Component family	Databases/Teradata	
Function	**tTeradataInput** reads a database and extracts fields based on a query.	
Purpose	**tTeradataInput** executes a DB query with a strictly defined order which must correspond to the schema definition. Then it passes on the field list to the next component via a Main row link.	
Basic settings	*Property type*	Either Built-in or Repository
		Built-in: No property data stored centrally.
		Repository: Select the repository file where Properties are stored. The fields that come after are pre-filled in using the fetched data.
	Use an existing connection	Select this check box and click the relevant **tTeradataConnection** component on the Component list to reuse the connection details you already defined.
		Click this icon to open a database connection wizard and store the database connection parameters you set in the component **Basic settings** view. For more information about setting up and storing database connection parameters, see *Setting up a DB connection* of **Talend Open Studio** User Guide.
	Host	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database
	Username and *Password*	DB user authentication data.
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either built-in or remotely stored in the Repository.
		Built-in: The schema is created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide
		Repository: The schema already exists and is stored in the Repository, hence can be reused. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	Query type and *Query*	Enter your DB query paying particularly attention to properly sequence the fields in order to match the schema definition.

Advanced settings	*Encoding Type*	Select the encoding type from the list or select **Custom** and define it manually. This field is compulsory for DB data handling.
	Trim all the String/Char columns	Select this check box to remove leading and trailing whitespace from all the String/Char columns.
	Trim column	Remove leading and trailing whitespace from defined columns.
	tStateCatcher Statistics	Select this check box to collect log data at the component level.
Usage	This component covers all possibilities of SQL queries onto a Teradata database.	

Related scenarios

Related topics in generic **tDBInput** scenarios:

- *Scenario 1: Displaying selected data from DB table on page 163*
- *Scenario 2: Using StoreSQLQuery variable on page 164*

Related topic in **tContextLoad** *Scenario: Dynamic context use in MySQL DB insert on page 652*.

tTeradataOutput

tTeradataOutput Properties

Component family	Databases/Teradata	
Function	**tTeradataOutput** writes, updates, makes changes or suppresses entries in a database.	
Purpose	**tTeradataOutput** executes the action defined on the table and/or on the data contained in the table, based on the flow incoming from the preceding component in the job.	
Basic settings	*Property type*	Either Built-in or Repository.
		Built-in: No property data stored centrally.
		Repository: Select the repository file where Properties are stored. The fields that come after are pre-filled in using the fetched data.
	Use an existing connection	Select this check box and click the relevant **tTeradataConnection** component on the Component list to reuse the connection details you already defined.
		Click this icon to open a database connection wizard and store the database connection parameters you set in the component **Basic settings** view. For more information about setting up and storing database connection parameters, see *Setting up a DB connection* of **Talend Open Studio** User Guide.
	Host	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database
	Username and *Password*	DB user authentication data.
	Table	Name of the table to be written. Note that only one table can be written at a time
	Action on table	On the table defined, you can perform one of the following operations: **None**: No operation is carried out. **Drop and create a table**: The table is removed and created again. **Create a table**: The table does not exist and gets created. **Create a table if not exists**: The table is created if it does not exist. **Drop a table if exists and ceate**: The table is removed if it already exists and created again. **Clear a table**: The table content is deleted.

	Action on data	On the data of the table defined, you can perform: **Insert**: Add new entries to the table. If duplicates are found, job stops. **Update**: Make changes to existing entries **Insert or update**: Add entries or update existing ones. **Update or insert**: Update existing entries or create it if non existing **Delete**: Remove entries corresponding to the input flow. *It is necessary to specify at least one column as a primary key on which the Update and Delete operations are based. You can do that by clicking Edit Schema and selecting the check box(es) next to the column(s) you want to set as primary key(s). For an advanced use, click the Advanced settings view where you can simultaneously define primary keys for the Update and Delete operations. To do that: Select the Use field options check box and then in the Key in update column, select the check boxes next to the column names you want to use as a base for the Update operation. Do the same in the Key in delete column for the Delete operation.*
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either built-in or remotely stored in the Repository.
		Built-in: The schema is created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	Die on error	Clear this check box to skip the row on error and complete the process for error-free rows.
Advanced settings	*Commit every*	Enter the number of rows to be completed before committing batches of rows together into the DB. This option ensures transaction quality (but not rollback) and, above all, better performance at execution.
	Additional Columns	This option is not offered if you create (with or without drop) the DB table. This option allows you to call SQL functions to perform actions on columns, which are not insert, nor update or delete actions, or action that require particular preprocessing.
		Name: Type in the name of the schema column to be altered or inserted as new column

		SQL expression: Type in the SQL statement to be executed in order to alter or insert the relevant column data.
		Position: Select **Before**, **Replace** or **After** following the action to be performed on the reference column.
		Reference column: Type in a column of reference that the **tDBOutput** can use to place or replace the new or altered column.
	Use field options	Select this check box to customize a request, especially when there is double action on data.
	Enable debug mode	Select this check box to display each step during processing entries in a database.
	tStateCatcher Statistics	Select this check box to collect log data at the component level.
	Use Batch Size	When selected, enables you to define the number of lines in each processed batch.
Usage	This component offers the flexibility benefit of the DB query and covers all possibilities of SQL queries.	

Related scenarios

For related topics, see:

- **tDBOutput** *Scenario: Displaying DB output on page 168*
- **tMySQLOutput** *Scenario 1: Adding a new column and altering data in a DB table on page 283*.

tTeradataRow

tTeradataRow Properties

Component family	Databases/Teradata	
Function	**tTeradataRow** is the specific component for this database query. It executes the SQL query stated onto the specified database. The row suffix means the component implements a flow in the job design although it doesn't provide output.	
Purpose	Depending on the nature of the query and the database, **tTeradataRow** acts on the actual DB structure or on the data (although without handling data). The SQLBuilder tool helps you write easily your SQL statements.	
Basic settings	*Property type*	Either Built-in or Repository.
		Built-in: No property data stored centrally.
		Repository: Select the repository file where Properties are stored. The fields that come after are pre-filled in using the fetched data.
	Use an existing connection	Select this check box and click the relevant **tTeradataConnection** component on the Component list to reuse the connection details you already defined.
	Host	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database
	Username and *Password*	DB user authentication data.
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either built-in or remotely stored in the Repository.
		Built-in: The schema is created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	Query type	Either Built-in or Repository.
		Built-in: Fill in manually the query statement or build it graphically using SQLBuilder
		Repository: Select the relevant query stored in the Repository. The Query field gets accordingly filled in.

	Query	Enter your DB query paying particularly attention to properly sequence the fields in order to match the schema definition.
	Commit every	Number of rows to be completed before committing batches of rows together into the DB. This option ensures transaction quality (but not rollback) and above all better performance on executions.
	Die on error	Clear this check box to skip the row on error and complete the process for error-free rows.
Advanced settings	*Propagate QUERY's recordset*	Select this check box to insert the result of the query into a COLUMN of the current flow.
	Commit every	Number of rows to be completed before committing batches of rows together into the DB. This option ensures transaction quality (but not rollback) and above all better performance on executions.
	*tState*Catcher Statistics	Select this check box to collect log data at the component level.
Usage	This component offers the flexibility benefit of the DB query and covers all possibilities of SQL queries.	

Related scenarios

For related topics, see:

- **tDBSQLRow** *Scenario: Resetting a DB auto-increment on page 172*
- **tMySQLRow** *Scenario: Removing and regenerating a MySQL table index on page 301*.

tVerticaBulkExec

tVerticaBulkExec Properties

tVerticaOutputBulk and **tVerticaBulkExec** components are used together to first output the file that will be then used as parameter to execute the SQL query stated. These two steps compose the **tVerticaOutputBulkExec** component, detailed in a separate section. The interest in having two separate elements lies in the fact that it allows transformations to be carried out before the data loading in the database.

Component family	Databases/Vertica	
Function	Executes the Insert action on the data provided.	
Purpose	As a dedicated component, **tVerticaBulkExec** offers gains in performance while carrying out the Insert operations to a Mysql database	
Basic settings	*Property type*	Either Built-in or Repository.
		Built-in: No property data stored centrally.
		Repository: Select the Repository file where Properties are stored. When selected, the fields to follow are pre-filled in using fetched data.
	Use an existing connection	Select this check box when using a **tVerticaConnection** component.
	Host	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database
	Username and *Password*	DB user authentication data.
	Action on table	On the table defined, you can perform one of the following operations: **None**: No operation is carried out. **Drop and create table**: The table is removed and created again. **Create table**: The table does not exist and gets created. **Create table if not exists**: The table is created if it does not exist. **Clear table**: The table content is deleted. You have the possibility to rollback the operation.
	Table	Name of the table to be written. Note that only one table can be written at a time and that the table must exist for the insert operation to succeed.
	Schema type and *Edit schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either built-in or remotely stored in the Repository.

		Built-in: The schema is created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	Remote Filename	Name of the file to be processed. Related topic:*Defining variables from the Component view* of **Talend Open Studio** User Guide.
	Exit job if no row was loaded	The Job automatically stops if no row has been loaded.
	Fields terminated by	Character, string or regular expression to separate fields.
	Null string	String displayed to indicate that the value is null.
	tStatCatcher Statistics	Select this check box to collect log data at the component level.
Usage	This component is to be used along with **tVerticaOutputBulk** component. Used together, they can offer gains in performance while feeding a Vertica database.	

Related scenarios

For related topics, see:

- **tMysqlOutputBulk** *Scenario: Inserting transformed data in MySQL database on page 293*.
- **tMysqlOutputBulkExec** *Scenario: Inserting data in MySQL database on page 297*.
- **tOracleBulkExec** *Scenario: Truncating and inserting file data into Oracle DB on page 329* du composant.

tVerticaCommit

tVerticaCommit Properties

This component is closely related to **tVerticaConnection** and **tVerticaRollback**. It usually doesn't make much sense to use these components independently in a transaction.

Component family	Databases/Vertica	
Function	**tVerticaConnection** validates the data processed through the job into the connected DB.	
Purpose	Using a unique connection, this component commits in one go a global transaction instead of doing that on every row or every batch and thus provides gain in performance.	
Basic settings	*Component list*	Select the **tVerticaConnection** component in the list if more than one connection are planned for the current job.
	Close connection	Clear this check box to continue to use the selected connection once the component has performed its task.
Advanced settings	*tStatCatcher Statistics*	Select this check box to collect log data at the component level.
Usage	This component is to be used along with Mysql components, especially with **tVerticaConnection** and **tVerticaRollback** components.	
Limitation	n/a	

Related scenario

This component is closely related to **tVerticaConnection** and **tVerticaRollback**. It usually doesn't make much sense to use one of these without using a **tVerticaConnection** component to open a connection for the current transaction.

For **tVerticaCommit** related scenario, see *tVerticaConnection on page 423*.

tVerticaConnection

tVerticaConnection Properties

This component is closely related to **tVerticaCommit** and **tVerticaRollback**. It usually doesn't make much sense to use one of these without using a **tVerticaConnection** component to open a connection for the current transaction.

Component family	Databases/Vertica	
Function	**tVerticaConnection** opens a connection to the database for a current transaction.	
Purpose	**tVerticaConnection** allows to commit a whole job data in one go to the output database as one transaction when validated.	
Basic settings	*Property type*	Either Built-in or Repository.
		Built-in: No property data stored centrally.
		Repository: Select the repository file where Properties are stored. The fields that come after are pre-filled in using the fetched data.
	Host	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database
	Username and *Password*	DB user authentication data.
	Encoding Type	Select the encoding from the list or select Custom and define it manually. This field is compulsory for DB data handling.
	Use or register a shared DB Connection	Select this check box to share your connection or fetch a connection shared by a parent or child Job. **Shared DB Connection Name:** set or type in the shared connection name.
Advanced settings	*tStatCatcher Statistics*	Select this check box to collect log data at the component level.
Utilisation	This component is to be used along with Mysql components, especially with **tVerticaCommit** and **tVerticaRollback** components.	
Limitation	n/a	

Related scenario

This component is closely related to **tVerticaCommit** and **tVerticaRollback**. It usually doesn't make much sense to use one of these without using a **tVerticaConnection** component to open a connection for the current transaction.

For **tVerticaConnection** related scenario, see *tMysqlConnection on page 269*.

tVerticaInput

tVerticaInput Properties

Component family	Databases/Vertica	 
Function	**tVerticaInput** reads a database and extracts fields based on a query.	
Purpose	**tVerticaInput** executes a DB query with a strictly defined order which must correspond to the schema definition. Then it passes on the field list to the next component via a **Main** row link.	
Basic settings	*Property type*	Either Built-in or Repository
		Built-in: No property data stored centrally.
		Repository: Select the repository file where Properties are stored. The fields that come after are pre-filled in using the fetched data.
		Click this icon to open a database connection wizard and store the database connection parameters you set in the component **Basic settings** view. For more information about setting up and storing database connection parameters, see *Setting up a DB connection* of **Talend Open Studio** User Guide.
	Use existing connection	Select this check box when using a **tVerticaConnection** component.
	Host	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database
	Username and *Password*	DB user authentication data.
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either built-in or remotely stored in the Repository.
		Built-in: The schema is created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	Table Name	Name of the table to be read.
	Query type and *Query*	Enter your DB query paying particularly attention to properly sequence the fields in order to match the schema definition.
Advanced settings	*Trim all the String/Char columns*	Select this check box to remove leading and trailing whitespace from all the String/Char columns.

	Trim column	Remove leading and trailing whitespace from defined columns.
	tStatCatcher Statistics	Select this check box to collect log data at the component level.
Utilisation	This component covers all possibilities of SQL queries onto a Vertica database.	

Related scenarios

Related topics in **tDBInput** scenarios:

- *Scenario 1: Displaying selected data from DB table on page 163*
- *Scenario 2: Using StoreSQLQuery variable on page 164*

Related topic in **tContextLoad** *Scenario: Dynamic context use in MySQL DB insert on page 652*.

tVerticaOutput

tVerticaOutput Properties

Component family	Databases/Vertica	
Function	**tVerticaOutput** writes, updates, makes changes or suppresses entries in a database.	
Purpose	**tVerticaOutput** executes the action defined on the table and/or on the data contained in the table, based on the flow incoming from the preceding component in the job.	
Basic settings	*Property type*	Either Built-in or Repository.
		Built-in: No property data stored centrally.
		Repository: Select the repository file where Properties are stored. The fields that come after are pre-filled in using the fetched data.
		Click this icon to open a database connection wizard and store the database connection parameters you set in the component **Basic settings** view. For more information about setting up and storing database connection parameters, see *Setting up a DB connection* of **Talend Open Studio** User Guide.
	Use existing connection	Select this check box when using a **tVerticaConnection** component.
	Host	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database
	Username and *Password*	DB user authentication data.
	Table	Name of the table to be written. Note that only one table can be written at a time
	Action on table	On the table defined, you can perform one of the following operations: **None**: No operation is carried out. **Drop and create a table**: The table is removed and created again. **Create a table**: The table does not exist and gets created. **Create a table if not exists**: The table is created if it does not exist. **Drop a table if exists and create**: The table is removed if it already exists and created again. **Clear a table**: The table content is deleted.

	Action on data	On the data of the table defined, you can perform: **Insert**: Add new entries to the table. If duplicates are found, job stops. **Update**: Make changes to existing entries **Insert or update**: Add entries or update existing ones. **Update or insert**: Update existing entries or create it if non existing **Delete**: Remove entries corresponding to the input flow. *It is necessary to specify at least one column as a primary key on which the Update and Delete operations are based. You can do that by clicking Edit Schema and selecting the check box(es) next to the column(s) you want to set as primary key(s). For an advanced use, click the Advanced settings view where you can simultaneously define primary keys for the Update and Delete operations. To do that: Select the Use field options check box and then in the Key in update column, select the check boxes next to the column names you want to use as a base for the Update operation. Do the same in the Key in delete column for the Delete operation.*
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either built-in or remotely stored in the Repository.
		Built-in: The schema is created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	Die on error	Clear this check box to skip the row on error and complete the process for error-free rows.
Advanced settings	*Encoding*	Select the encoding type from the list or select **Custom** and define it manually. This field is compulsory for DB data handling.
	Commit every	Number of rows to be completed before committing batches of rows together into the DB. This option ensures transaction quality (but not rollback) and, above all, better performance at executions.
	Use Batch Size	Number of lines in each processed batch.
	Additional Columns	This option is not offered if you create (with or without drop) the DB table. This option allows you to call SQL functions to perform actions on columns, which are not insert, nor update or delete actions, or action that require particular preprocessing.

		Name: Type in the name of the schema column to be altered or inserted as new column
		SQL expression: Type in the SQL statement to be executed in order to alter or insert the relevant column data.
		Position: Select **Before**, **Replace** or **After** following the action to be performed on the reference column.
		Reference column: Type in a column of reference that the **tDBOutput** can use to place or replace the new or altered column.
	Use field options	Select this check box to customize a request, especially when there is double action on data.
	Enable debug mode	Select this check box to display each step during processing entries in a database.
	tStatCatcher Statistics	Select this check box to collect log data at the component level.
Utilisation	This component offers the flexibility benefit of the DB query and covers all possibilities of SQL queries.	

Scénarios associés

For **tVerticaOutput** related topics, see:

- **tDBOutput** *Scenario: Displaying DB output on page 168*
- **tMySQLOutput** *Scenario 1: Adding a new column and altering data in a DB table on page 283*.

tVerticaOutputBulk

tVerticaOutputBulk Properties

tVerticaOutputBulk and **tVerticaBulkExec** components are used together to first output the file that will be then used as parameter to execute the SQL query stated. These two steps compose the **tVerticaOutputBulkExec** component, detailed in a separate section. The interest in having two separate elements lies in the fact that it allows transformations to be carried out before the data loading.

Component family	Databases/Vertica	
Function	**tVerticaBulkOutputExec** writes a file with columns based on the defined delimiter and the Vertica standards.	
Purpose	**tVerticaBulkOutputExec** prepares the file to be used as parameter in the INSERT query to feed the Vertica database.	
Basic settings	*Property type*	Either Built-in or Repository.
		Built-in: No property data stored centrally.
		Repository: Select the Repository file where Properties are stored. The following fields are pre-filled in using fetched data.
	File Name	Name of the file to be processed. Related topic:*Defining variables from the Component view* of **Talend Open Studio** User Guide
	Append	Select this check box to add the new rows at the end of the file.
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the Repository.
		Built-in: The schema will be created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused in various projects and job designs. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
Advanced settings	*Row separator*	String (ex: “\n”on Unix) to distinguish rows.
	Field separator	Character, string or regular expression to separate fields.
	Include header	Select this check box to include the column header to the file.

	Encoding Type	Select the encoding from the list or select **Custom** and define it manually. This field is compulsory for DB data handling.
	tStatCatcher Statistics	Select this check box to collect log data at the component level.
Utilisation	This component is to be used along with **tVerticaBulkExec** component. Used together they offer gains in performance while feeding a Vertica database.	

Related scenarios

For uses cases in relation with **tVerticaOutputBulk**, see the following scenarios:

- **tMysqlOutputBulk** *Scenario: Inserting transformed data in MySQL database on page 293*
- **tMysqlOutputBulkExec** *Scenario: Inserting data in MySQL database on page 297*

tVerticaOutputBulkExec

tVerticaOutputBulkExec Properties

Component family	Databases/Vertica	
Function	**tVerticaOutputBulkExec** executes the Insert action on the data provided.	
Purpose	As a dedicated component, it allows gains in performance during Insert operations to a Vertica database.	
Basic settings	*Property Type*	Either Built-in or Repository.
		Built-in: No property data stored centrally.
		Repository: Select the repository file where Properties are stored. The fields that come after are pre-filled in using the fetched data.
	Host	Database server IP address
	Port	Listening port number of DB server.
	DB Name	Name of the database
	Username and *Password*	DB user authentication data.
	Table	Name of the table to be written. Note that only one table can be written at a time and that the table must exist for the insert operation to succeed.
	Action on table	On the table defined, you can perform one of the following operations: **None**: No operation is carried out. **Drop and create a table**: The table is removed and created again. **Create a table**: The table does not exist and gets created. **Create a table if not exists**: The table is created if it does not exist. **Clear a table**: The table content is deleted.
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the Repository.
		Built-in: The schema will be created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused in various projects and job designs. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.

	File Name	Name of the file to be processed. Related topic:*Defining variables from the Component view* of **Talend Open Studio** User Guide.
	Append	Select this check box to add the new rows at the end of the file
Advanced settings	*Exit job if no row was loaded*	The Job automatically stops if no row has been loaded.
	Field Separator	Character, string or regular expression to separate fields.
	Null string	String displayed to indicate that the value is null.
	Include header	select this check box to include the column header to the file.
	Encoding Type	Select the encoding from the list or select **Custom** and define it manually. This field is compulsory for DB data handling.
	tStatCatcher Statistics	Select this check box to collect log data at the component level.
Usage	This component is mainly used when no particular transformation is required on the data to be loaded onto the database.	
Limitation	n/a	

Related scenarios

For uses cases in relation with **tVerticaOutputBulkExec**, see the following scenarios:

- **tMysqlOutputBulk** *Scenario: Inserting transformed data in MySQL database on page 293*
- **tMysqlOutputBulkExec** *Scenario: Inserting data in MySQL database on page 297*

tVerticaRollback

tVerticaRollback Properties

This component is closely related to **tVerticaCommit** and **tVerticaConnection**. It usually doesn't make much sense to use these components independently in a transaction.

Component family	Databases/Vertica	
Function	**tVerticaRollback** cancels the transaction commit in the connected DB.	
Purpose	**tVerticaRollback** avoids to commit part of a transaction involuntarily.	
Basic settings	*Component list*	Select the **VerticaConnection** component in the list if more than one connection are planned for the current job.
	Close Connection	Clear this check box to continue to use the selected connection once the component has performed its task.
Advanced settings	*tStatCatcher Statistics*	Select this check box to collect log data at the component level.
Usage	This component is to be used along with Mysql components, especially with **tVerticaConnection** and **tVerticaCommit** components.	
Limitation	n/a	

Related scenario

For **tVerticaRollback** related scenario, see *Scenario: Rollback from inserting data in mother/daughter tables on page 299*.

tVerticaRow

tVerticaRow Properties

Component family	Databases/Vertica	
Function	**tVerticaRow** is the specific component for this database query. It executes the SQL query stated onto the specified database. The row suffix means the component implements a flow in the job design although it doesn't provide output.	
Purpose	Depending on the nature of the query and the database, **tVerticaRow** acts on the actual DB structure or on the data (although without handling data). The SQLBuilder tool helps you write easily your SQL statements.	
Basic settings	*Property type*	Either Built-in or Repository.
		Built-in: No property data stored centrally.
		Repository: Select the repository file where Properties are stored. The fields that come after are pre-filled in using the fetched data.
	Use an existing connection	Select this check box and click the relevant **tMySqlConnection** component on the Component list to reuse the connection details you already defined.
	Port	Listening port number of DB server.
	Database	Name of the database
	Username and *Password*	DB user authentication data.
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either built-in or remotely stored in the Repository.
		Built-in: The schema is created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	Table name	Name of the table to process.
	Query type	Either Built-in or Repository.
		Built-in: Fill in manually the query statement or build it graphically using SQLBuilder
		Repository: Select the relevant query stored in the Repository. The Query field gets accordingly filled in.

	Query	Enter your DB query paying particularly attention to properly sequence the fields in order to match the schema definition.
	Die on error	Clear this check box to skip the row on error and complete the process for error-free rows.
Advanced settings	*Propagate QUERY's recordset*	Select this check box to insert the result of the query into a COLUMN of the current flow
	Encoding Type	Select the encoding from the list or select **Custom** and define it manually. This field is compulsory for DB data handling.
	Commit every	Number of rows to be completed before committing batches of rows together into the DB. This option ensures transaction quality (but not rollback) and above all better performance on executions.
	tStateCatcher Statistics	Select this check box to collect log data at the component level.
Usage	This component offers the flexibility benefit of the DB query and covers all possibilities of SQL queries.	

Related scenario

For related topics, see:

- **tDBSQLRow** *Scenario: Resetting a DB auto-increment on page 172*
- **tMySQLRow** *Scenario: Removing and regenerating a MySQL table index on page 301*.

ELT components

This chapter details the major components that you can find in **ELT** group of the **Palette** of **Talend Open Studio**.

The ELT family groups most popular database connectors, as well as processing components, all dedicated to the ELT mode where the target DBMS becomes the transformation engine.

This mode supports Teradata, Oracle, Netezza, Dataupia, QuickFire, Datallegro & Vertica.

tELTAggregate

tELTAggregate properties

Component family	ELT	
Function	**tELTAggregate** collects data values from one or more columns with the intent to manage the collection as a single unit. This component has real-time capabilities since it runs the data transform on the DBMS itself.	
Purpose	Helps to provide a set of matrix based on values or calculations.	
Basic settings	*Component list*	Select the relevant DB connection component in the list if you use more than one connection in the current Job.
	Database name	Name of the database.
	Source table name	Name of the table holding the data you want to collect values from.
	Target table name	Name of the table you want to write the collected and transformed data in.
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the Repository. Click **Edit Schema** to make changes to the schema. Note that if you make changes, the schema automatically becomes built-in.
		Built-in: You create and store the schema locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: You have already created the schema and stored it in the Repository. You can reuse it in various projects and job flowcharts. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide
	Operations	Select the type of operation along with the value to use for the calculation and the output field.
		Output Column: Select the destination field in the list.
		Function: Select any of the following operations to perform on data: count, min, max, avg, sum, first, last, list, and count (distinct).
		Input column position: Select the input column from which you want to collect the values to be aggregated.
	Group by	Define the aggregation sets, the values of which will be used for calculations.

		Output Column: Select the column label in the list offered according to the schema structure you defined. You can add as many output columns as you wish to make more precise aggregations.
		Input Column position: Match the input column label with your output columns, in case the output label of the aggregation set needs to be different.
SQL pattern	*SQLPattern List*	**To add a default system SQL pattern:** Click the **Add** button to add the default system SQL pattern(s) in the **SQLPattern List**. Click in the SQL pattern field and then click the arrow to display the system SQL pattern list. Select the desired system SQL pattern provided by **Talend**. **Note:** You can create your own SQL patterns and add them to the **SQLPattern List**. **To create a user-defined SQL pattern:** -Select a system pattern from the **SQLPattern list** and click on its code in the code box. You will be prompted by the system to create a new pattern. -Click **Yes** to open the SQL pattern wizard. -Define your new SQL pattern in the corresponding fields and click **Finish** to close the wizard. An SQL pattern editor opens where you can enter the pattern code. -Click the **Add** button to add the new created pattern to the **SQLPattern list**. For more information, see *Using SQL Templates* of **Talend Open Studio** User Guide.
Usage	This component is used as an intermediate component with other relevant DB components, especially the DB connection and commit components.	
Limitation	n/a	

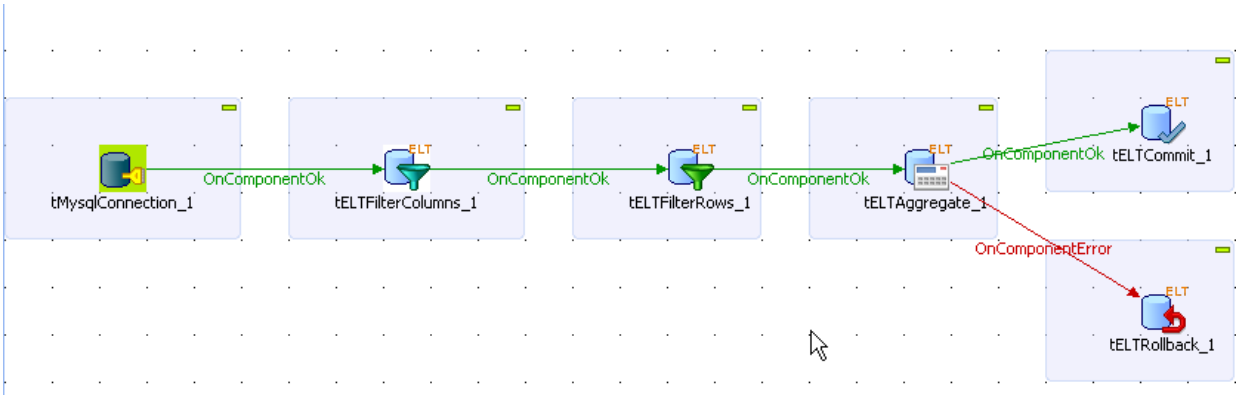
Scenario: Filtering and aggregating table columns directly on the DBMS

The following Java scenario creates a Job that opens a connection to a Mysql database and:

- instantiates the schemas from a database table whose rows match the column names specified in the filter,
- filters a column in the same database table to have only the data that matches a WHERE clause,
- collects data grouped by specific value(s) from the filtered column and writes aggregated data in a target database table.

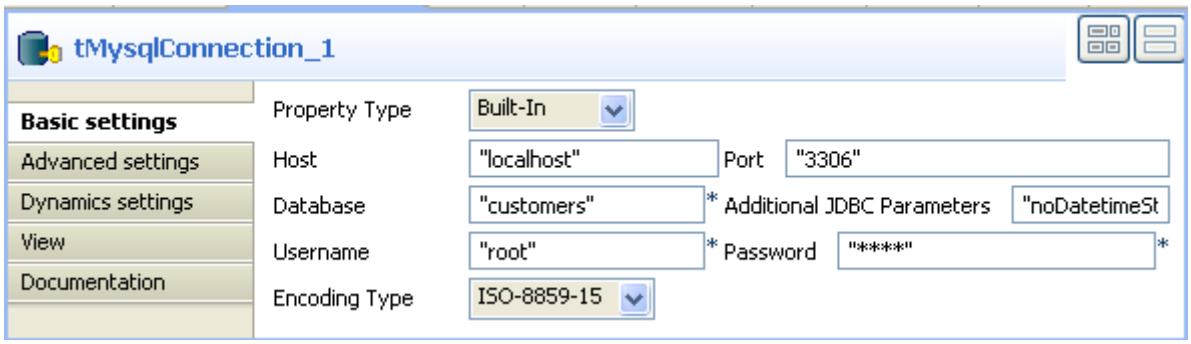
To filter and aggregate database table columns:

- Drop the following components from the **Palette** onto the design workspace: **tELTMysqlconnection**, **tELTFilterColumns**, **tELTFilterRows**, **tELTAggregae**, **tELTCommit**, and **tELTRollback**.
- Connect the five first components using **OnComponentOk** links.
- Connect **tELTAggregate** to **tELTRollback** using an **OnComponentError** link.

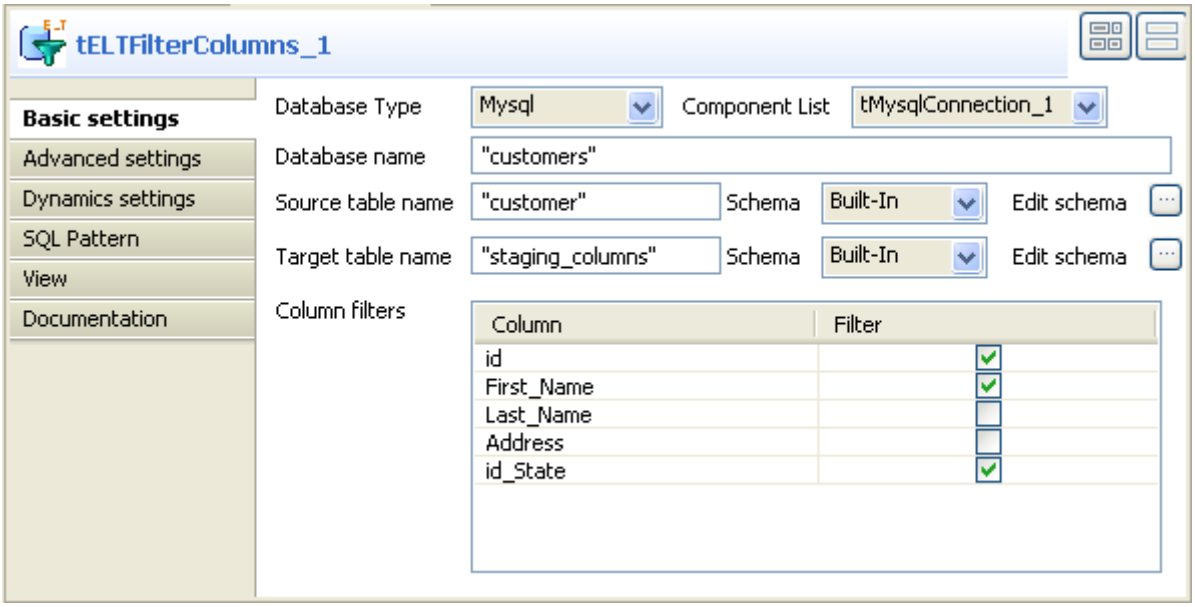


- In the design workspace, select **tMysqlConnection** and click the **Component** tab to define the basic settings for **tMysqlConnection**.
- In the **Basic settings** view, set the database connection details manually or select them from the context variable list, through a **Ctrl+Space** click in the corresponding field if you have stored them locally as Metadata DB connection entries.

For more information about Metadata, see *Defining Metadata items* of **Talend Open Studio** User Guide.



- In the design workspace, select **tELTFilterColumns** and click the **Component** tab to define its basic settings.



- On the **Database type** list, select the relevant database.
- On the **Component list**, select the relevant database connection component if more than one connection is used.
- Enter the names for the database, source table, and target table in the corresponding fields and click the three-dot buttons next to **Edit schema** to define the data structure in the source and target tables.

When you define the data structure for the source table, column names automatically appear in the **Column** list in the **Column filters** panel.

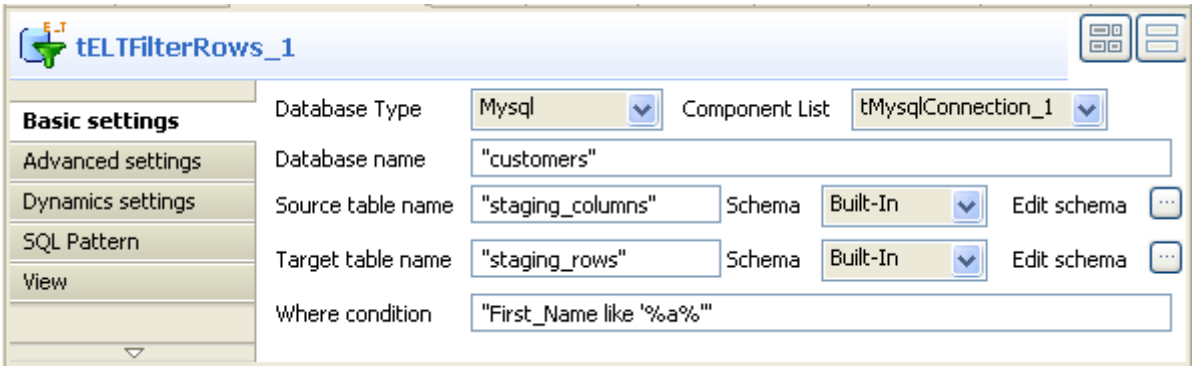
In this scenario, the source table has five columns: *id*, *First_Name*, *Last_Name*, *Address*, and *id_State*.

- In the **Column filters** panel, set the column filter by selecting the check boxes of the columns you want to write in the source table.

In this scenario, the **tELTFilterColumns** component instantiates only three columns: *id*, *First_Name*, and *id_State* from the source table.

In the **Component** view, you can click the **SQL Pattern** tab and add system SQL patterns or create your own and use them within your Job to carry out the coded operation. For more information, see *tELTFilterColumns Properties on page 445*.

- In the design workspace, select **tELTFilterRows** and click the **Component** tab to define its basic settings.



- On the **Database type** list, select the relevant database.
- On the **Component list**, select the relevant database connection component if more than one connection is used.
- Enter the names for the database, source table, and target table in the corresponding fields and click the three-dot buttons next to **Edit schema** to define the data structure in the source and target tables.

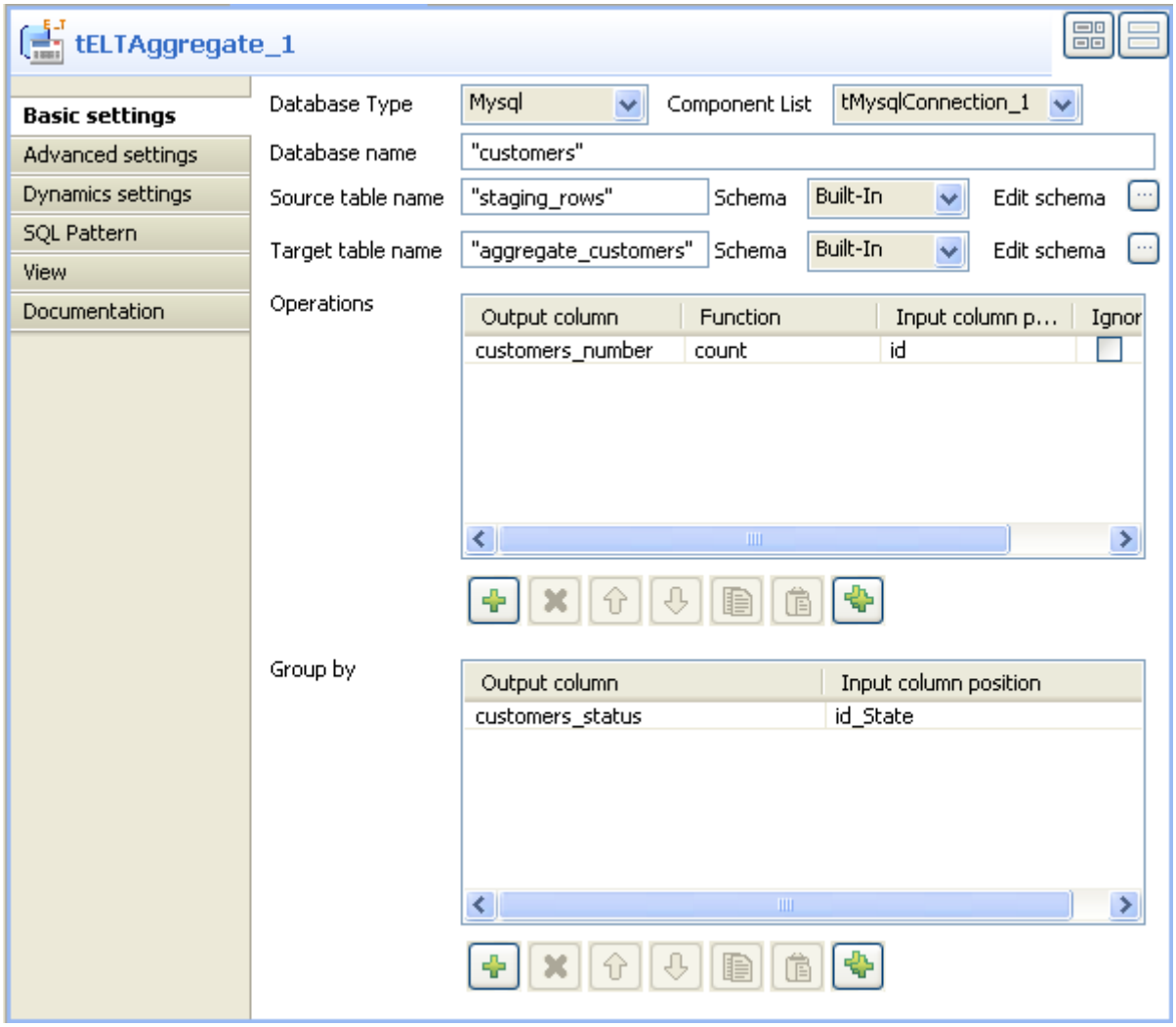
In this scenario, the source table has the three initially instantiated columns: *id, First_Name,* and *id_State* and the source table has the same three-column schema.

- In the **Where condition** field, enter a WHERE clause to extract only those records that fulfill the specified criterion.

In this scenario, the **tELTFilterRows** component filters the *First_Name* column in the source table to extract only the first names that contain the “a” letter.

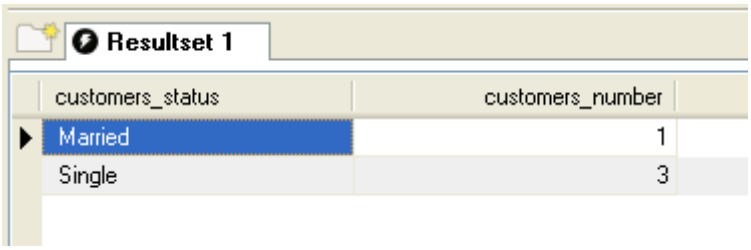
- In the design workspace, select **tELTAggregate** and click the **Component** tab to define its basic settings.
- On the **Database type** list, select the relevant database.
- On the **Component list**, select the relevant database connection component if more than one connection is used.
- Enter the names for the database, source table, and target table in the corresponding fields and click the three-dot buttons next to **Edit schema** to define the data structure in the source and target tables.

The schema for the source table consists of the three columns: *id*, *First_Name*, and *id_State*. The schema for the target table consists of two columns: *customers_status* and *customers_number.* In this scenario, we want to group customers by their marital status and count customer number in each marital group. To do that, we define the **Operations** and **Grouped by** panels accordingly.



- In the **Operations** panel, click the plus button to add one or more lines and then click in the **Output column** line to select the output column that will hold the counted data.
- Click in the **Function** line and select the operation to be carried on.
- In the **Group by** panel, click the plus button to add one or more lines and then click in the **Output column** line to select the output column that will hold the aggregated data.
- In the design workspace, select **tELTCommit** and click the **Component** tab to define its basic settings.
- On the **Database type** list, select the relevant database.
- On the **Component list**, select the relevant database connection component if more than one connection is used.
- Do the same for **tELTRollback**.
- Save your Job and press **F6** to execute it.

A two-column table *aggregate_customers* is created in the database. It groups customers according to their marital status and count customer number in each marital group.


Resultset 1
customers_status
customers_number
Married
1
Single
3

tELTFilterColumns

tELTFilterColumns Properties

Component family	ELT	
Function	**tELTFilterColumns** makes specified changes to the defined schema of the database table based on column name mapping. This component has real-time capabilities since it runs the data filtering on the DBMS itself	
Purpose	Helps homogenizing schemas either on the columns order or by removing unwanted columns or adding new columns.	
Basic settings	*Component list*	Select the relevant DB connection component in the list if you use more than one connection in the current Job.
	Database name	Name of the database.
	Source table name	Name of the table holding the data you want to filter.
	Target table name	Name of the table you want to write the filtered data in.
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the Repository. Click **Edit Schema** to make changes to the schema. Note that if you make changes, the schema automatically becomes built-in.
		Built-in: You create and store the schema locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: You have already created the schema and stored it in the Repository. You can reuse it in various projects and job flowcharts. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.

SQL pattern	*SQLPattern List*	**To add a default system SQL pattern:** Click the **Add** button to add the default system SQL pattern(s) in the **SQLPattern List**. Click in the SQL pattern field and then click the arrow to display the system SQL pattern list. Select the desired system SQL pattern provided by Talend. **Note:** You can create your own SQL patterns and add them to the **SQLPattern List**. **To create a user-defined SQL pattern:** -Select a system pattern from the **SQLPattern list** and click on its code in the code box. You will be prompted by the system to create a new pattern. -Click **Yes** to open the SQL pattern wizard. -Define your new SQL pattern in the corresponding fields and click **Finish** to close the wizard. An SQL pattern editor opens where you can enter the pattern code. -Click the **Add** button to add the new created pattern to the **SQLPattern list**. For more information, see *Using SQL Templates* of **Talend Open Studio** User Guide.
Usage	This component is used as an intermediate component with other relevant DB components, especially DB connection components.	
Limitation	n/a	

Related Scenario

For a related scenario, see *Scenario: Filtering and aggregating table columns directly on the DBMS on page 439*.

tELTFilterRows

tELTFilterRows Properties

Component family	ELT	
Function	**tELTFilterRows** allows you to define a row filter on one table. This component has real-time capabilities since it runs the data filtering on the DBMS itself.	
Purpose	Helps parametrizing row filters for any source data against a Where clause.	
Basic settings	*Component list*	Select the relevant DB connection component in the list if you use more than one connection in the current Job.
	Database name	Name of the database.
	Source table name	Name of the table holding the data you want to filter.
	Target table name	Name of the table you want to write the filtered data in.
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the Repository. Click **Edit Schema** to make changes to the schema. Note that if you make changes, the schema automatically becomes built-in.
		Built-in: You create and store the schema locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: You have already created the schema and stored it in the Repository. You can reuse it in various projects and job flowcharts. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	Where condition	Use a WHERE clause to set the criteria that the rows must meet. You can use the WHERE clause to select specific rows from the table that match certain specified criteria or conditions.

SQL pattern	*SQLPattern List*	**To add a default system SQL pattern:** Click the **Add** button to add the default system SQL pattern(s) in the **SQLPattern List**. Click in the SQL pattern field and then click the arrow to display the system SQL pattern list. Select the desired system SQL pattern provided by Talend. **Note:** You can create your own SQL patterns and add them to the **SQLPattern List**. **To create a user-defined SQL pattern:** -Select a system pattern from the **SQLPattern list** and click on its code in the code box. You will be prompted by the system to create a new pattern. -Click **Yes** to open the SQL pattern wizard. -Define your new SQL pattern in the corresponding fields and click **Finish** to close the wizard. An SQL pattern editor opens where you can enter the pattern code. -Click the **Add** button to add the new created pattern to the **SQLPattern list**. For more information, see *Using SQL Templates* of **Talend Open Studio** User Guide.
Usage	This component is used as an intermediate component with other DB components, especially DB connection components.	
Limitation	n/a	

Realted Scenario

For a related scenario, see *Scenario: Filtering and aggregating table columns directly on the DBMS on page 439*.

tELTMysqlInput

tELTMysqlInput properties

The **tELTMysqlInput**, **tELTMysqlOutput**, and **tELTMysqlMap** components are closely related together in regard to their operating condition. These components should be used to handle MySQL DB schemas to generate Insert statements including clauses, which are to be executed to the DB output table defined.

Component family	ELT	
Function	Provides the table schema to be used for the SQL statement to execute.	
Purpose	Allows to add as many Input tables as required for the most complicated Insert statement.	
Basic settings	*Schema type* and *Edit Schema*	A schema is a row description, i.e., it defines the nature and number of fields to be processed. The schema is either built-in or remotely stored in the Repository. The Schema defined is then passed on to the ELT Mapper to be included to the Insert SQL statement.
		Built-in: The schema is created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
Usage	**tELTMysqlInput** is to be used along with the **tELTMysqlMap**. Note that the Output link to be used with these components has to reflect faithfully the name of the table  Note that the ELT components do not handle actual data flow but only schema information.	

Related scenarios

For uses cases in relation with **tELTMysqlInput**, see **tELTMysqlMap** scenarios:

- *Scenario1: Aggregating table columns and filtering on page 452*
- *Scenario 2: ELT using Alias table on page 455*

tELTMysqlMap

tELTMysqlMap properties

The **tELTMysqlInput**, **tELTMysqlOutput**, and **tELTMysqlMap** are closely related together in regard to their operating condition. These components should be used to handle MySQL DB schemas to generate Insert statements including clauses, which are to be executed to the DB output table defined.

Component family	ELT	
Function	Helps to graphically build the SQL statement using the table provided as input.	
Purpose	Uses the tables provided as input, to feed the parameter in the built statement. The statement can include inner or outer joins to be implemented between tables or between one table and its aliases.	
Basic settings	*Property type*	Either Built-in or Repository.
		Built-in: No property data stored centrally.
		Repository: Select the Repository file where Properties are stored. The following fields are pre-filled in using fetched data.
	Host	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database
	Username and *Password*	DB user authentication data.
	Encoding	Select the encoding from the list or select Custom and define it manually. This field is compulsory for DB data handling.
	Preview	The preview is an instant shot of the Mapper data. It becomes available when Mapper properties have been filled in with data. The preview synchronization takes effect only after saving changes.
	Map editor	The ELT Map editor allows you to define the output schema as well as build graphically the SQL statement to be executed.
Usage	**tELTMysqlMap** is used along with a **tELTMysqlInput** and **tELTMysqlOutput**. Note that the Output link to be used with these components has to reflect faithfully the name of the tables. Note that the ELT components do not handle actual data flow but only schema information.	

Connecting ELT components

The ELT components do not handle any data as such but table schema information that will be used to build the SQL query to execute.

Therefore the only connection required to connect these components together is a simple link.

The output name you give to this link when creating it should always be the exact name of the table to be accessed as this parameter will be used in the SQL statement generated.

Related topic: *Link connection* of **Talend Open Studio** User Guide.

Mapping and joining tables

In the ELT Mapper, you can select specific columns from input schemas and include them in the output schema.

- As you would do it in the regular Mapper editor, simply drag & drop the content from the input schema towards the output table defined.
- Use the Ctrl and Shift keys for multiple selection of contiguous or non contiguous table columns.

You can implement explicit joins to retrieve various data from different tables.

- Click on the **Join** drop-down list and select the relevant explicit join.
- Possible joins include: **Inner Join**, **Left Outer Join**, **Right Outer Join** or **Full Outer Join** and **Cross Join**.
- By default the **Inner Join** is selected.

You can also create **Alias** tables to retrieve various data from the same table.

- In the Input area, click on the plus (+) button to create an Alias.
- Define the table to base the alias on.
- Type in a new name for the alias table, preferably not the same as the main table.

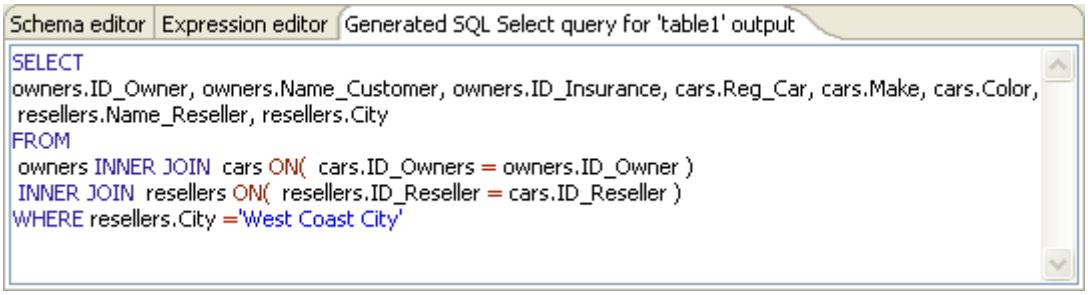
Adding where clauses

You can also restrict the Select statement based on a Where clause. Click on the **Add filter row** button at the top of the output table and type in the relevant restriction to be applied.

Make sure that all input components are linked correctly to the ELT Map component to be able to implement all inclusions, joins and clauses.

Generating the SQL statement

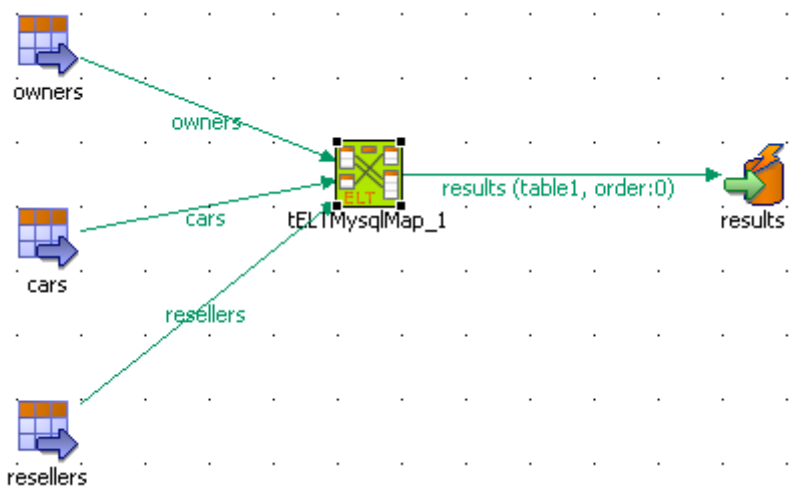
The mapping of elements from the input schemas to the output schemas create instantly the corresponding Select statement.



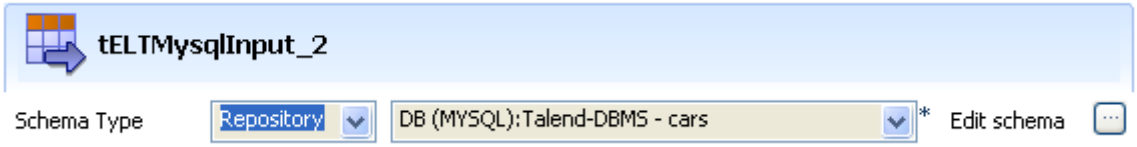
The clause are also included automatically.

Scenario1: Aggregating table columns and filtering

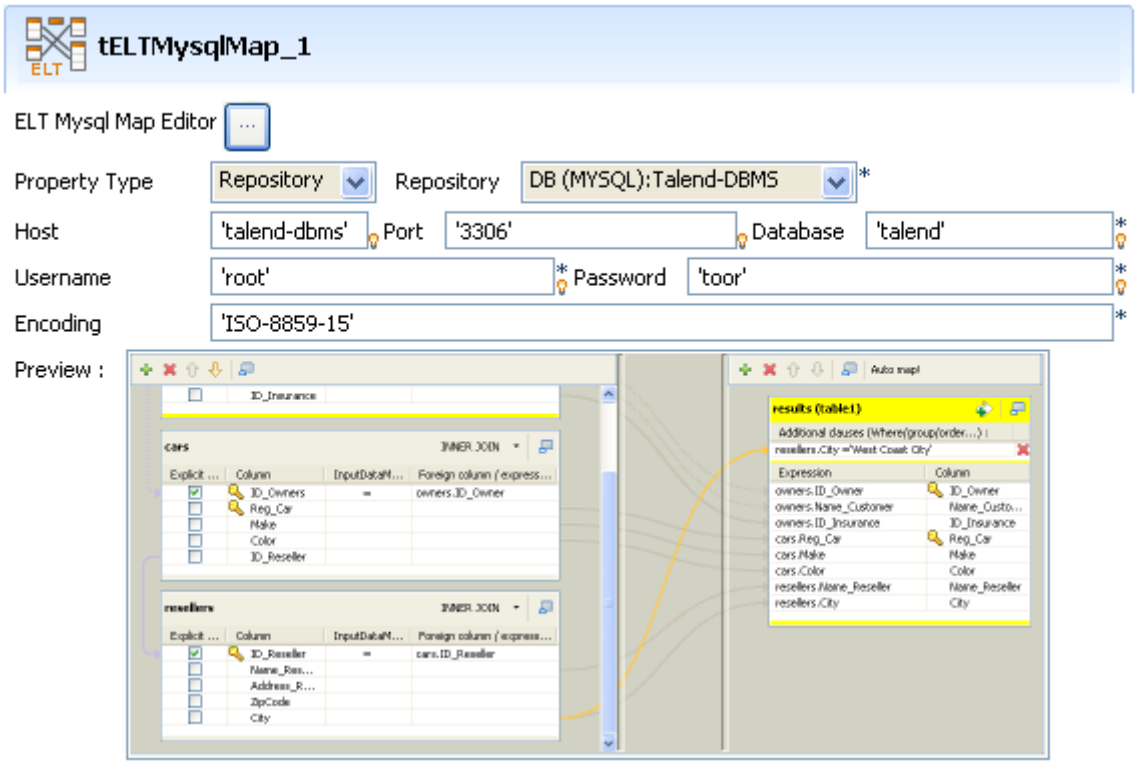
This scenario describes a Job gathering together several Input DB table schemas and implementing a clause to filter the resulting output using an SQL statement.



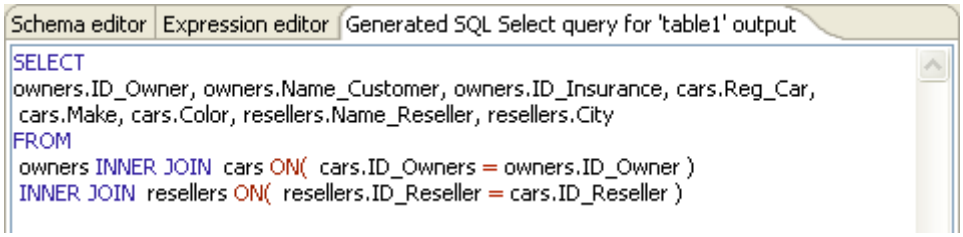
- Drop the following components: **tELTMysqlIntput**, **tELTMysqlMap**, **tELTMysqlOutput** from the **Palette** to the design workspace.
- Three input components are required for this job.
- Connect the three ELT input components to the ELT mapper using links named following strictly the actual DB table names: *owners*, *cars* and *resellers*.
- Then connect the ELT mapper to the ELT Output component using another link that you call *results*.
- All three Input schemas are stored in the **Metadata** area of the **Repository**. They can therefore be easily retrieved.



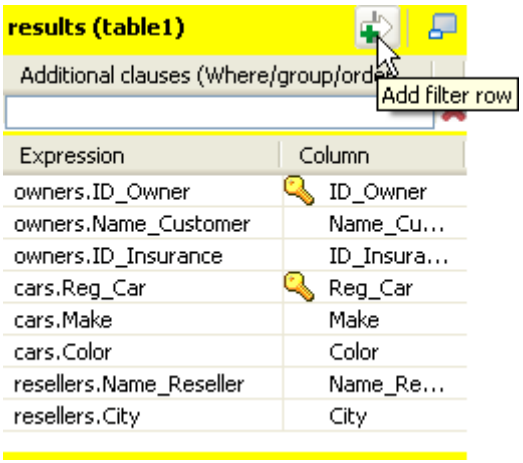
- Click on the ELT mapper component to define the Database connection details.
- The Database connection details are stored in the Repository again



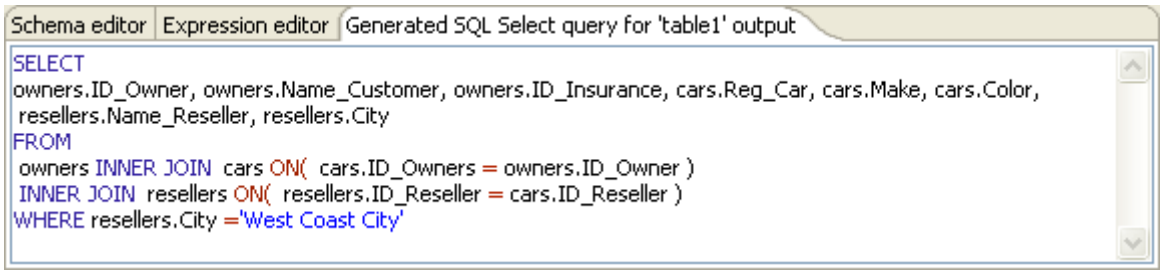
- The default encoding for Mysql database is retained.
- Launch the ELT Map editor to set up the join between Input tables
- Drag & drop the *ID_Owner* column from the *Owners* table to the corresponding column of the *cars* table.
- Select **INNER JOIN** in the *Cars* table join list, and select the **Explicit Join** check box, in front of the *ID_Owners*.
- Drag the *ID_Resellers* column from the *Cars* table to the *Resellers* table to set up the second join. Select here again **INNER JOIN** in the list of the Resellers table and check the **Explicit Join** box of the relevant column.
- Then select the columns to be aggregated into the output.
- Select all columns from the *Cars* and *Owners* table and only the *Reseller_Name* and *City* columns from the *Resellers* table.
- Drag & drop them to the *Results* output table.
- The mapping displays in yellow and the joins display in dark violet.
- Click on the **Generated SQL Select query** tab to display the corresponding SQL Statement.



- Then implement a filter on the output table.
- Click on the **Add filter row** button of the output table.



- Restrict the Select using a Where clause such as: resellers.City ='West Coast City'
- See the reflected where clause on the Generated SQL Select query tab.



- Click **OK** to save the ELT Map setting.
- Define the ELT Output in the Component view of the **tELTMysqlOutput** component.
- The **Action on table** is **Drop and create table** for this use case and the only action available on data in MySQL is **Insert**.
- The schema is to be synchronized with the **tELTMysqlMap** component as it aggregates several source schemas.

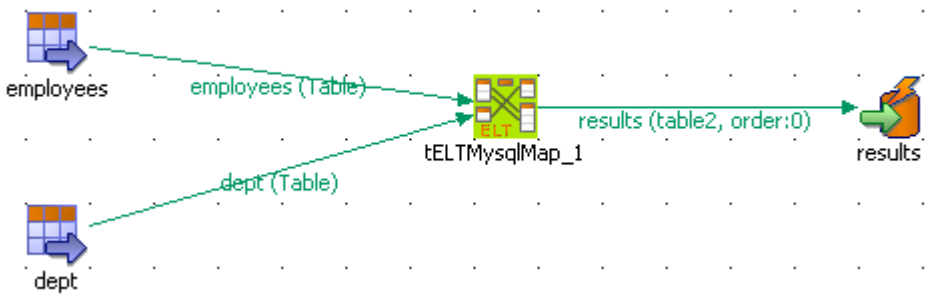
Resultset 1

ID...	Name_Customer	ID_Insur...	Reg_Car	Make	Color	Name_Reseller	City
4	hirtken	ENX9366	6225 GT 57	Renault	green	Best Cars Shop	West Coast City
16	kennan	RTA8580	0601 SQ 67	Renault	blue	Cars & Pickup Re...	West Coast City
31	antken	QDG0199	5729 DJ 52	Renault	blue	Cars & Pickup Sp...	West Coast City
33	hirtken	MYA1613	0427 LR 72	Toyota	gold	Best Cars Specialist	West Coast City
34	nanant	XBM2459	7355 IB 28	BMW	purple	Best Cars Resale	West Coast City
39	nanneng	CSP8847	8402 JE 03	Volkswa...	blue	Cars & Pickup Re...	West Coast City
62	gallken	LZP1021	3940 ZW 19	BMW	green	All you need Outlet	West Coast City
65	oinele	ANX9956	6523 DY 26	Mercedes	gold	Best Cars Shop	West Coast City
76	nengle	SLG5853	2087 IW 01	Toyota	gold	Best Cars Outlet	West Coast City
77	boot	QFS6844	3795 GN 95	Toyota	blue	All you need Outlet	West Coast City
88	carbo	EHN3338	7752 OB 89	Mercedes	blue	Cars & Pickup Re...	West Coast City
89	bobouh	UMT0348	0462 FV 53	Toyota	purple	Best Cars Outlet	West Coast City

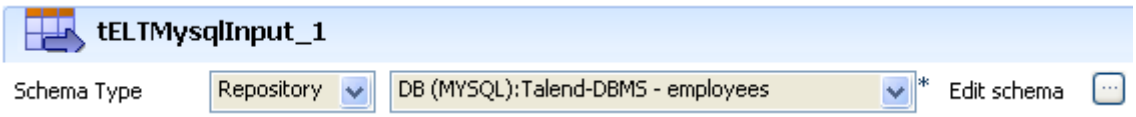
All selected data are inserted in the *results* table as specified in the SQL statement defined respecting the clause.

Scenario 2: ELT using Alias table

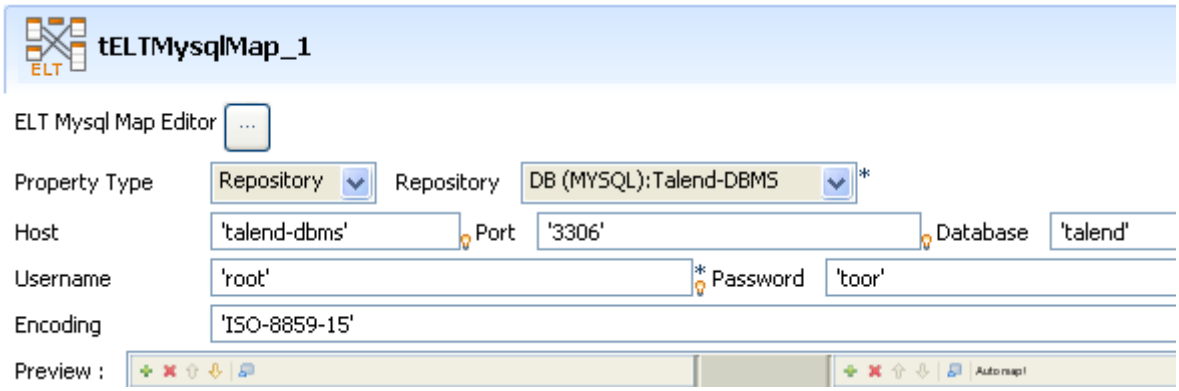
This scenario describes a Job that uses an Alias table. The *employees* table contains all employees details as well as the ID of their respective manager, which are also considered as employees and hence included in the *employees* table. The dept table contains location and department information about the employees.



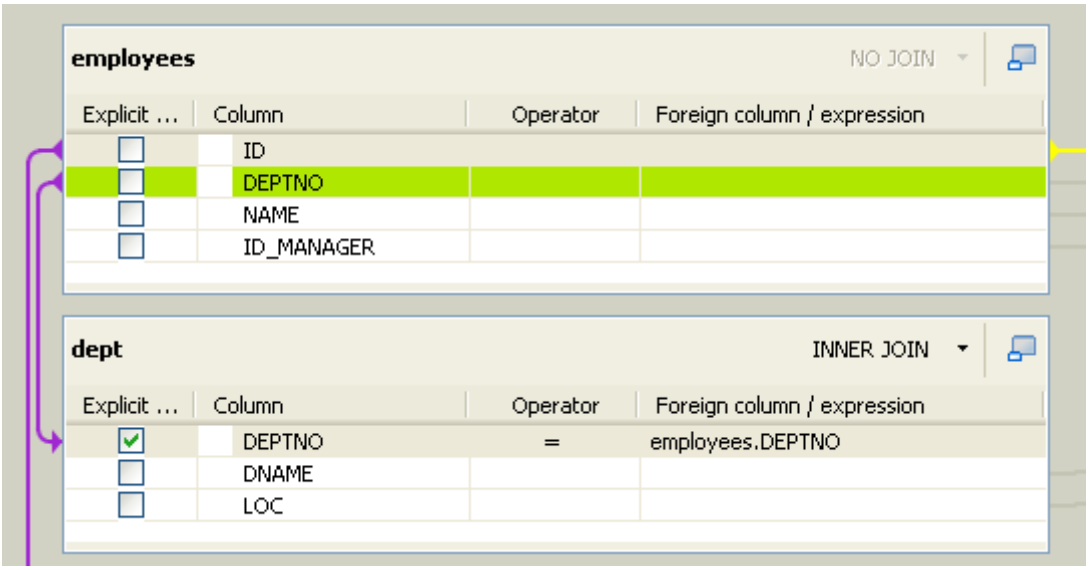
- Drag and drop **tELTMysqlInput** components to retrieve respectively the *employees* and *dept* table schemas.
- In this use case, both schemas are stored in the Repository and can therefore be easily retrieved.



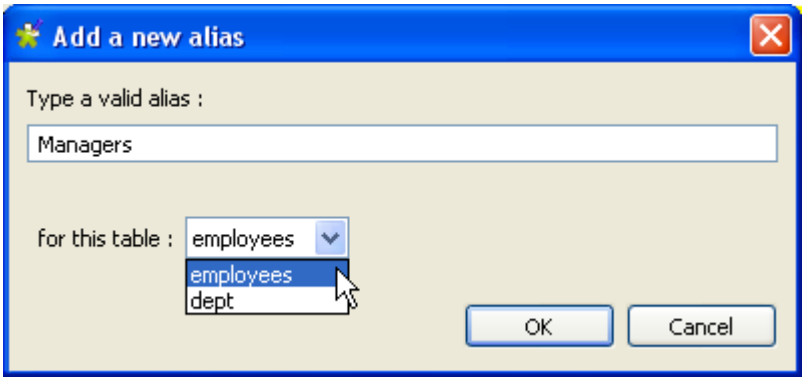
- Then select the **tELTMysqlMap** and define the Mysql database connection details.
- Here again the connection information is stored in the Repository's Metadata.



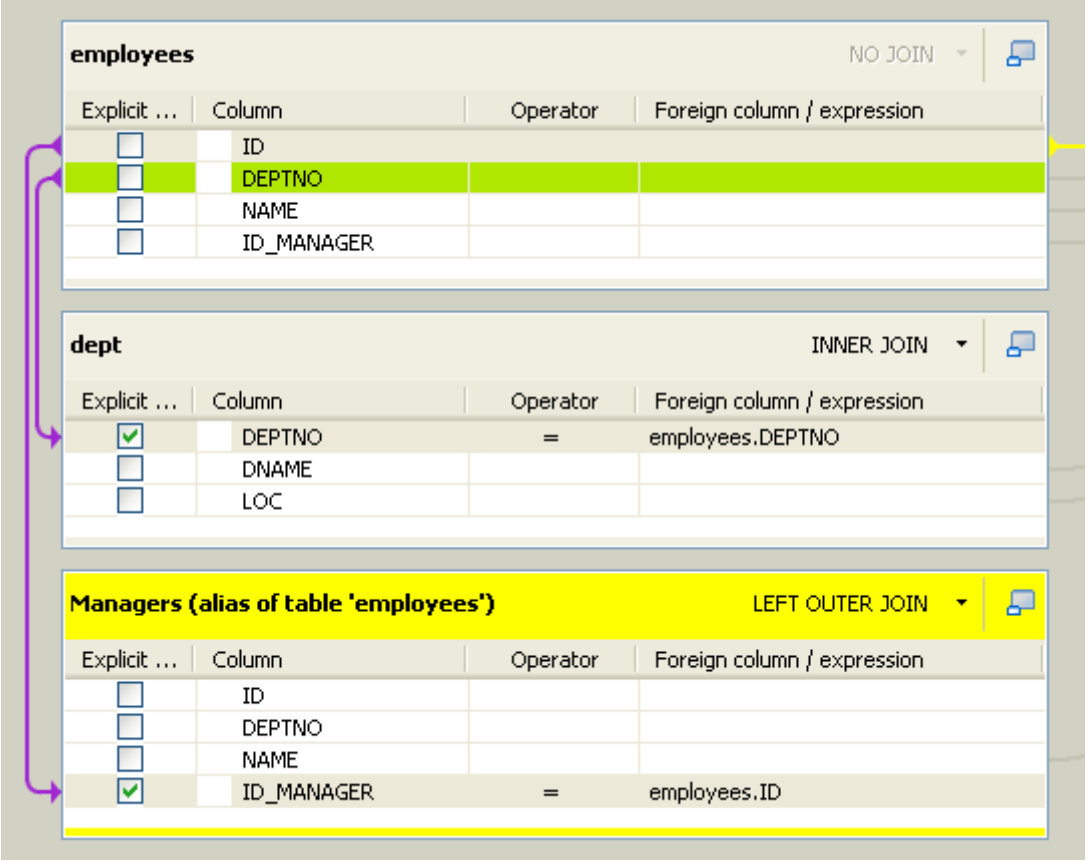
- Click on the button to launch the ELT Map editor.
- First make sure the correct input table is positioned at the top of the Input area, as the Joins are highly dependent on this position.
- In this example, the *employees* table should be on top.



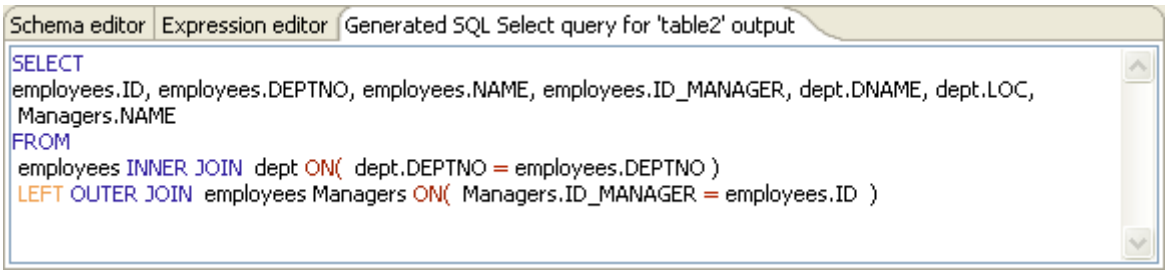
- Drag and drop the *DeptNo* column from the *employees* table to the *dept* table to set up the **join** between both input tables.
- Select the **Explicit Join** check box and define the join as an **Inner Join**.
- Then create the **Alias** table based on the *employees* table



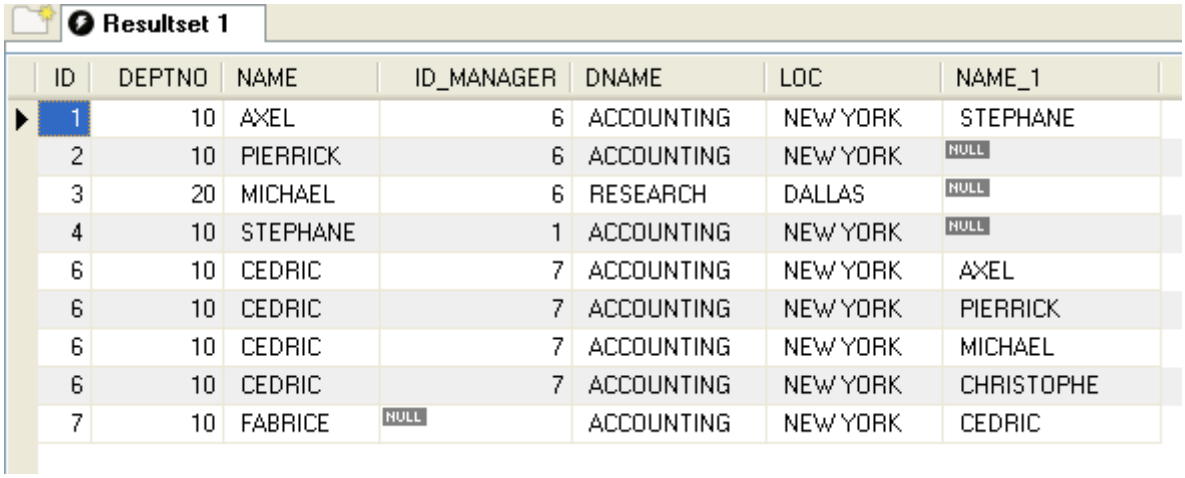
- Name it *Managers* and click OK to display it as a new Input table in the ELT mapper.
- Drag & drop the *ID* column from the *employees* table to the *ID_Manager* column of the newly added *Managers* table.
- Select the **Explicit Join** check box and define it as **Left Outer Join**, in order for results to be output even though they contain a **Null** value.



- Drag and drop the content of both Input tables, *employees* and *dept*, as well as the *Name* column from the *Manager* table to the Output table.
- Click on the **Generated SQL Select query** tab to display the query to be executed.



- Then click on the Output component and define the **Action on data** on **Insert**.
- Make sure the schema is synchronized with the Output table from the ELT mapper before running the Job through **F6** or via the toolbar.



Resultset 1

ID	DEPTNO	NAME	ID_MANAGER	DNAME	LOC	NAME_1
1	10	AXEL	6	ACCOUNTING	NEW YORK	STEPHANE
2	10	PIERRICK	6	ACCOUNTING	NEW YORK	NULL
3	20	MICHAEL	6	RESEARCH	DALLAS	NULL
4	10	STEPHANE	1	ACCOUNTING	NEW YORK	NULL
6	10	CEDRIC	7	ACCOUNTING	NEW YORK	AXEL
6	10	CEDRIC	7	ACCOUNTING	NEW YORK	PIERRICK
6	10	CEDRIC	7	ACCOUNTING	NEW YORK	MICHAEL
6	10	CEDRIC	7	ACCOUNTING	NEW YORK	CHRISTOPHE
7	10	FABRICE	NULL	ACCOUNTING	NEW YORK	CEDRIC

The *Department* information as well as the *Employees* entries are coupled in the output, and the *Manager Name* could be retrieved via the explicit join.

tELTMysqlOutput

tELTMysqlOutput properties

The **tELTMysqlInput**, **tELTMysqlOutput**, and **tELTMysqlMap** are closely related together in regard to their operating condition. These components should be used to handle MySQL DB schemas to generate Insert statements including clauses, which are to be executed to the DB output table defined.

Component family	ELT	
Function	Carries out the action on the table specified and inserts the data according to the output schema defined the ELT Mapper.	
Purpose	Executes the SQL Insert statement to the Mysql database	
Basic settings  *In Java, use tCreateTable as substitute for this function.*	*Action on table*	On the table defined, you can perform one of the following operations: **None**: No operation carried out **Drop and create the table**: The table is removed and created again **Create a table**: The table doesn't exist and gets created. If the table exists, an error is generated and the Job is stopped. **Create table if doesn't exist**: Create the table if needed and carries out the action on data anyway. **Clear a table**: The table content is deleted
	Action on data	On the data of the table defined, you can perform the following operation: **Insert**: Add new entries to the table. If duplicates are found, Job stops. Note that in Mysql ELT, only Insert operation is available.
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either built-in or remotely stored in the Repository.
		Built-in: The schema is created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	Encoding	Select the encoding from the list or select Custom and define it manually. This field is compulsory for DB data handling.

Usage	**tELTMysqlOutput** is to be used along with the **tELTMysqlMap**. Note that the Output link to be used with these components has to reflect faithfully the name of the table. Note that the ELT components do not handle actual data flow but only schema information.

Related scenarios

For uses cases in relation with **tELTMysqlOutput**, see **tELTMysqlMap** scenarios:

- *Scenario1: Aggregating table columns and filtering on page 452*
- *Scenario 2: ELT using Alias table on page 455*

tELTOracleInput

tELTOracleInput properties

All three ELT Oracle components are closely related together in regard to their operating condition. These components should be used to handle Oracle DB schemas to generate Insert, Update or Delete statements including clauses, which are to be executed to the DB output table defined.

Component family	ELT	
Function	Provides the table schema to be used for the SQL statement to execute.	
Purpose	Allows to add as many Input tables as required for the most complicated Insert statement.	
Basic settings	*Schema type* and *Edit Schema*	A schema is a row description, i.e., it defines the nature and number of fields to be processed. The schema is either built-in or remotely stored in the Repository. The Schema defined is then passed on to the ELT Mapper to be included to the Insert SQL statement.
		Built-in: The schema is created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
Usage	**tELTOracleInput** is to be used along with the **tELTOracleMap**. Note that the Output link to be used with these components has to reflect faithfully the name of the table  Note that the ELT components do not handle actual data flow but only schema information.	

Related scenarios

For uses cases in relation with **tELTOracleInput**, see **tELTOracleMap** *Scenario: Updating Oracle DB entries on page 463*.

tELTOracleMap

tELTOracleMap properties

All three ELT Oracle components are closely related together in regard to their operating condition. These components should be used to handle Oracle DB schemas to generate Insert, Update or Delete statements including clauses, which are to be executed to the DB output table defined.

Component family	ELT	
Function	Helps to graphically build the SQL statement using the table provided as input.	
Purpose	Uses the tables provided as input, to feed the parameter in the built statement. The statement can include inner or outer joins to be implemented between tables or between one table and its aliases.	
Basic settings	*Property type*	Either Built-in or Repository.
		Built-in: No property data stored centrally.
		Repository: Select the Repository file where Properties are stored. The following fields are pre-filled in using fetched data.
	Host	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database
	Username and *Password*	DB user authentication data.
	Encoding	Select the encoding from the list or select Custom and define it manually. This field is compulsory for DB data handling.
	Preview	The preview is an instant shot of the Mapper data. It becomes available when Mapper properties have been filled in with data. The preview synchronization takes effect only after saving changes.
	Map editor	The ELT Map editor allows you to define the output schema as well as build graphically the SQL statement to be executed.
Usage	**tELTOracleMap** is used along with a **tELTOracleInput** and **tELTOracleOutput**. Note that the Output link to be used with these components has to reflect faithfully the name of the tables. Note that the ELT components do not handle actual data flow but only schema information.	

Connecting ELT components

For detailed information regarding ELT component connections, see *Connecting ELT components on page 450*.

Related topic: *Link connection* of **Talend Open Studio** User Guide.

Mapping and joining tables

In the ELT Mapper, you can select specific columns from input schemas and include them in the output schema.

For detailed information regarding the table schema mapping and joining, see *Mapping and joining tables on page 463*.

Adding where clauses

For details regarding the clause handling, see *Adding where clauses on page 463*.

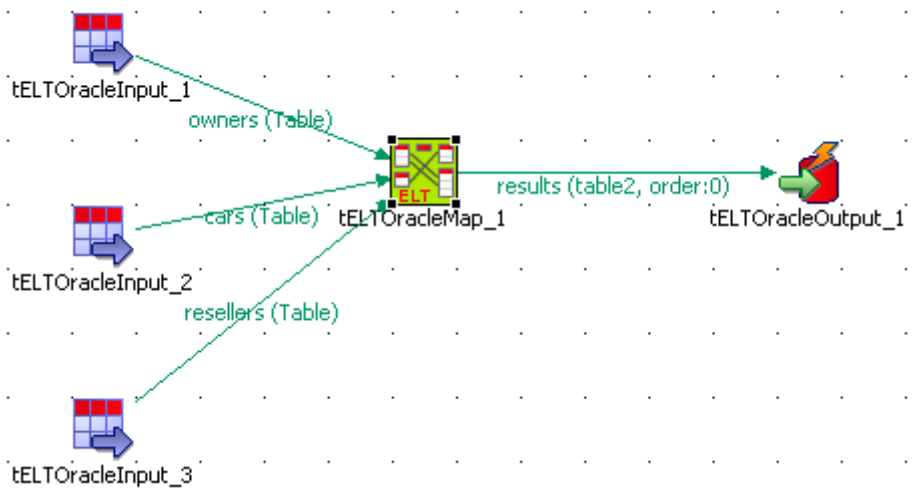
Generating the SQL statement

The mapping of elements from the input schemas to the output schemas create instantly the corresponding Select statement.

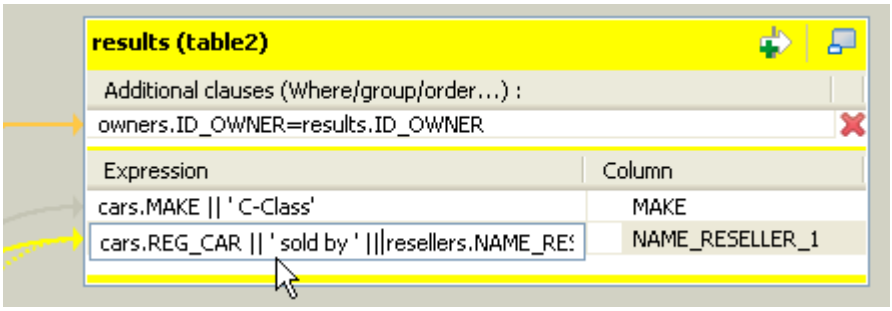
The clause defined internally in the ELT Mapper are also included automatically.

Scenario: Updating Oracle DB entries

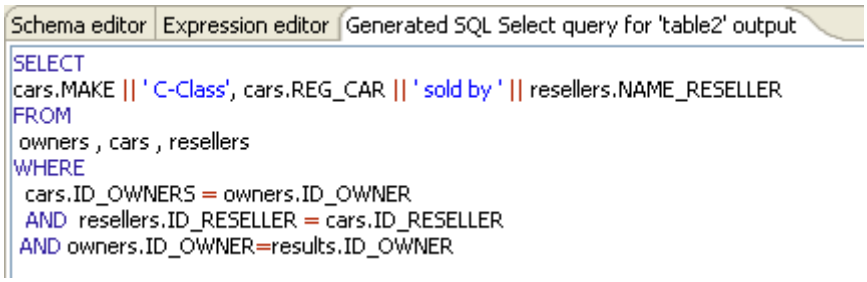
This scenario relies on the Job described in ELT MySQL components, *Scenario1: Aggregating table columns and filtering on page 452*. As the update action on the data is available in Oracle DB, this scenario describes a Job updating particular entries of the *results* table, adding a *model* to the *make* column of the *cars* table.



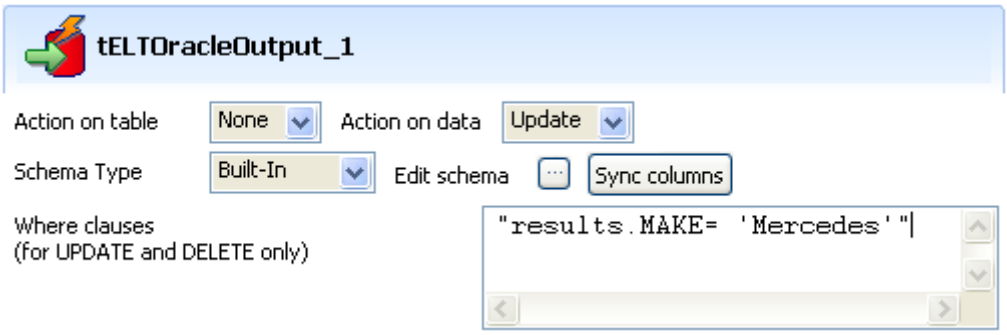
- Define all three Input components as described in *Scenario1: Aggregating table columns and filtering on page 452*.
- When connecting the ELT Input components to the ELT mapper, make sure you use the relevant table names as these table names will be used as parameters in the SQL statement generated in the ELT mapper.
- Remove the additional clause used to filter the output columns.
- Add a new filter row to the output table in the ELT mapper to setup a relationship between input and output tables: owners.ID_OWNER=results.ID_OWNER



- Remove also all the columns unused for the **Update** action on the output table.
- Then apply the update to the *Make* column adding the mention *C-Class* preceding by a double-pipe.
- And also add the mention *Sold by* in front of the *reseller name* column from the *resellers* table.
- Check the **Generated SQL select query** to be executed.



- Click **OK** to validate the changes in the ELT mapper. And make sure the Oracle DB connection details are filled in the **tELTOracleMap** component Basic settings.
- Select the **tELTOracleOutput** component to define the Action on data to be carried out.



- There is no action on the table, and the **Action on data** is set to **Update**.
- Check that the Schema type corresponds to the output table from the ELT Mapper.
- In the **Where clause** area, add an additional clause: results.MAKE= ‘Mercedes’.
- Then press **F6** to run the Job and check the results table in a DB viewer.

```
Starting job ELTOracleUpdate at 12:52 11/04/2007.
Updating with :
UPDATE results SET (MAKE,NAME_RESELLER) = (SELECT cars.MAKE || '
C-Class', cars.REG_CAR || ' sold by ' || resellers.NAME_RESELLER
FROM  owners , cars , resellers WHERE    cars.ID_OWNERS =
owners.ID_OWNER   AND  resellers.ID_RESELLER = cars.ID_RESELLER
AND owners.ID_OWNER=results.ID_OWNER)   WHERE results.MAKE=
'Mercedes'

--> 2 rows updated.

Job ELTOracleUpdate ended at 12:52 11/04/2007. [exit code=0]
```

The Job executes the query generated and updates the relevant rows.

BMW	green	98	All you need Outlet	West Coast City
Mercedes C-Class	gold	3	6523 DY 26 sold by Best Cars Shop	West Coast City
Toyota	gold	84	Best Cars Outlet	West Coast City
Toyota	blue	98	All you need Outlet	West Coast City
Mercedes C-Class	blue	58	7752 OB 89 sold by Cars & Pickup Resale	West Coast City

tELTOracleOutput

tELTOracleOutput properties

All three ELT Oracle components are closely related together in regard to their operating condition. These components should be used to handle Oracle DB schemas to generate Insert, Update or Delete statements including clauses, which are to be executed to the DB output table defined.

Component family	ELT	
Function	Carries out the action on the table specified and inserts the data according to the output schema defined the ELT Mapper.	
Purpose	Executes the SQL Insert statement to the Mysql database	
Basic settings  *In Java, use tCreate Table as substitute for this function.*	*Action on table*	On the table defined, you can perform one of the following operations: **None**: No operation carried out **Drop and create the table**: The table is removed and created again **Create a table**: The table doesn't exist and gets created. If the table exists, an error is generated and the Job is stopped. **Create table if doesn't exist**: Create the table if needed and carries out the action on data anyway. **Clear a table**: The table content is deleted
	Action on data	On the data of the table defined, you can perform the following operation: **Insert**: Add new entries to the table. If duplicates are found, Job stops. **Update**: updates entries in the table.
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either built-in or remotely stored in the Repository.
		Built-in: The schema is created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	Encoding	Select the encoding from the list or select Custom and define it manually. This field is compulsory for DB data handling.
Usage	**tELTOracleOutput** is to be used along with the **tELTOracleMap**. Note that the Output link to be used with these components has to reflect faithfully the name of the table.  Note that the ELT components do not handle actual data flow but only schema information.	

Related scenarios

For uses cases in relation with **tELTOracleOutput**, see **tELTOracleMap** *Scenario: Updating Oracle DB entries on page 463*.

tELTTeradataInput

tELTTeradataInput properties

All three ELT Teradata components are closely related together in regard to their operating condition. These components should be used to handle Teradata DB schemas to generate Insert statements including clauses, which are to be executed to the DB output table defined.

Component family	ELT	
Function	Provides the table schema to be used for the SQL statement to execute.	
Purpose	Allows to add as many Input tables as required for the most complicated Insert statement.	
Basic settings	*Schema type* and *Edit Schema*	A schema is a row description, i.e., it defines the nature and number of fields to be processed. The schema is either built-in or remotely stored in the Repository. The Schema defined is then passed on to the ELT Mapper to be included to the Insert SQL statement.
		Built-in: The schema is created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
Usage	**tELTTeradataInput** is to be used along with the **tELTTeradataMap**. Note that the Output link to be used with these components has to reflect faithfully the name of the table  Note that the ELT components do not handle actual data flow but only schema information.	

Related scenarios

For uses cases in relation with **tELTTeradataInput**, see **tELTMysqlMap** scenarios:

- *Scenario1: Aggregating table columns and filtering on page 452*
- *Scenario 2: ELT using Alias table on page 455*

tELTTeradataMap

tELTTeradataMap properties

All three ELT Teradata components are closely related together in regard to their operating condition. These components should be used to handle Teradata DB schemas to generate Insert statements including clauses, which are to be executed to the DB output table defined.

Component family	ELT	
Function	Helps to graphically build the SQL statement using the table provided as input.	
Purpose	Uses the tables provided as input, to feed the parameter in the built statement. The statement can include inner or outer joins to be implemented between tables or between one table and its aliases.	
Basic settings	*Property type*	Either Built-in or Repository.
		Built-in: No property data stored centrally.
		Repository: Select the Repository file where Properties are stored. The following fields are pre-filled in using fetched data.
	Host	Database server IP address
	Port	Listening port number of DB server.
	Database	Name of the database
	Username and *Password*	DB user authentication data.
	Encoding	Select the encoding from the list or select Custom and define it manually. This field is compulsory for DB data handling.
	Preview	The preview is an instant shot of the Mapper data. It becomes available when Mapper properties have been filled in with data. The preview synchronization takes effect only after saving changes.
	Map editor	The ELT Map editor allows you to define the output schema as well as build graphically the SQL statement to be executed.
Usage	**tELTTeradataMap** is used along with a **tELTTeradataInput** and **tELTTeradataOutput**. Note that the Output link to be used with these components has to reflect faithfully the name of the tables. The ELT components do not handle actual data flow but only schema information.	

Connecting ELT components

For detailed information regarding ELT component connections, see *Connecting ELT components on page 450*.

Related topic: *Link connection* of **Talend Open Studio** User Guide.

Mapping and joining tables

In the ELT Mapper, you can select specific columns from input schemas and include them in the output schema.

For detailed information regarding the table schema mapping and joining, see *Mapping and joining tables on page 463*.

Adding WHERE clauses

For details regarding the clause handling, see *Adding where clauses on page 463*.

Generating the SQL statement

The mapping of elements from the input schemas to the output schemas create instantly the corresponding Select statement.

The clause defined internally in the ELT Mapper are also included automatically.

Related scenarios

For uses cases in relation with **tELTTeradataMap**, see **tELTMysqlMap** scenarios:

- *Scenario1: Aggregating table columns and filtering on page 452*
- *Scenario 2: ELT using Alias table on page 455*

tELTTeradataOutput

tELTTeradataOutput properties

All three ELT Teradata components are closely related together in regard to their operating condition. These components should be used to handle Teradata DB schemas to generate Insert statements including clauses, which are to be executed to the DB output table defined.

Component family	ELT	
Function	Carries out the action on the table specified and inserts the data according to the output schema defined the ELT Mapper.	
Purpose	Executes the SQL Insert statement to the Teradata database	
Basic settings *In Java, use tCreate Table as substitute for this function.*	*Action on table*	On the table defined, you can perform one of the following operations: **None**: No operation carried out **Drop and create the table**: The table is removed and created again **Create a table**: The table doesn't exist and gets created. If the table exists, an error is generated and the Job is stopped. **Create table if doesn't exist**: Create the table if needed and carries out the action on data anyway. **Clear a table**: The table content is deleted
	Action on data	On the data of the table defined, you can perform the following operation: **Insert**: Add new entries to the table. If duplicates are found, Job stops. Note that in Teradata ELT, only Insert operation is available.
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either built-in or remotely stored in the Repository.
		Built-in: The schema is created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	Encoding	Select the encoding from the list or select Custom and define it manually. This field is compulsory for DB data handling.
Usage	**tELTTeradataOutput** is to be used along with the **tELTTeradataMap**. Note that the Output link to be used with these components has to reflect faithfully the name of the table. Note that the ELT components do not handle actual data flow but only schema information.	

Related scenarios

For uses cases in relation with **tELTTeradataOutput**, see **tELTMysqlMap** scenarios:

- *Scenario1: Aggregating table columns and filtering on page 452*
- *Scenario 2: ELT using Alias table on page 455*

File components

This chapter details the major components that you can find in **File** group of the **Palette** of **Talend Open Studio**.

The File family groups components that read and write data in all types of files, from the most popular to the most specific format (in the Input and Output subfamilies). In addition, the Management subfamily groups File-dedicated components that perform various tasks on files, including unarchiving, deleting, copying, comparing files and so on.

tAdvancedFileOutputXML

tAdvancedFileOutputXML belongs to two component families: File and XML. For more information on **tAdvancedFileOutputXML,** see *tAdvancedFileOutputXML on page 808*.

tApacheLogInput

tApacheLogInput properties

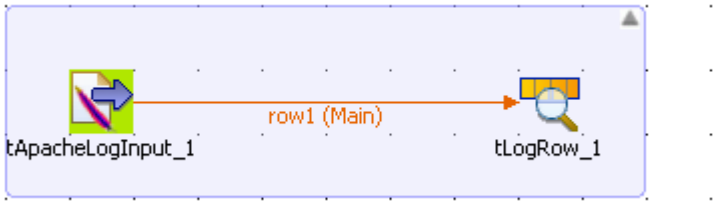
Component family	File/Input	
Function	**tApacheLogInput** reads the access-log file for an Apache HTTP server.	
Purpose	**tApachLogInput** helps to effectively manage the Apache HTTP Server,. It is necessary to get feedback about the activity and performance of the server as well as any problems that may be occurring.	
Basic settings	*Property type*	Either Built-in or Repository.
		Built-in: No property data stored centrally.
		Repository: Select the Repository file where Properties are stored. The fields to follow are pre defined using fetched data.
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the Repository. In the context of **tApacheLogInput** usage, the schema is read-only**.**
		Built-in: You can create the schema and store it locally for this component. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: You have already created and stored the schema in the Repository. You can reuse it in various projects and job flowcharts. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	File Name	Name of the file to be processed. Related topic:*Defining variables from the Component view* of **Talend Open Studio** User Guide.
	Die on error	Clear this check box to skip every line that contains an error and complete the process for error-free lines.
Usage	This component can be used as standalone component.	
Limitation	n/a	

Scenario: Reading an Apache access-log file

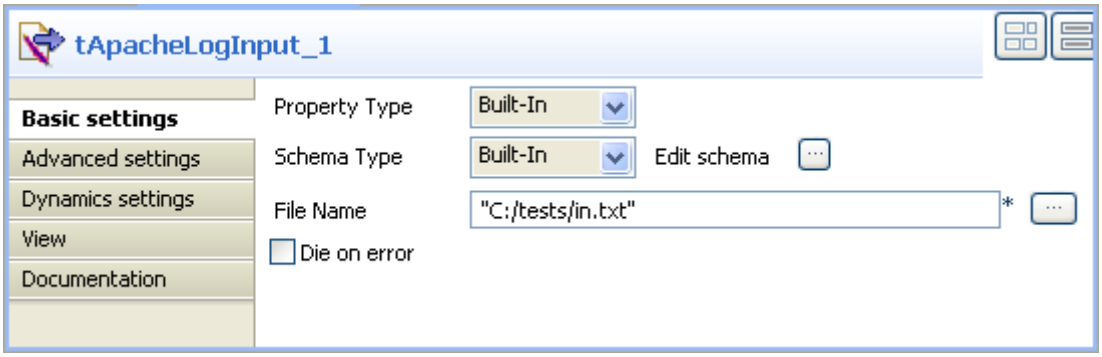
The following scenario creates a two-component Job, which aims at reading the access-log file for an Apache HTTP server and displaying the output in the **Run** log console.

- Drop a **tApacheLogInput** component and a **tLogRow** component from the **Palette** onto the design workspace.

- Right-click on the **tApacheLogInput** component and connect it to the **tLogRow** component using a **Main Row** link.



- In the design workspace, select **tApacheLogInput**.
- Click the **Component** tab to define the basic settings for **tApacheLogInput**.



- Set **Property Type** and **Schema Type** to **Built-In**.
- If desired, click the **Edit schema** button to see the read-only columns.
- In the **File Name** field, enter the file path or browse to the access-log file you want to read.
- In the design workspace, select **tLogRow** and click the **Component** tab to define its basic settings. For more information, see *tLogRow on page 628*.
- Press **F6** to execute the Job.

```
Starting job apach at 12:11 01/09/2008.
194.84.172.4|-|-|27/May/2007|03:06:12|+0200|GET|/forum/index.php|HTTP/
1.0|200|19024|http://www.talendforge.org/forum/index.php|Mozilla/4.0
(compatible; MSIE 6.0; Windows NT 5.1; .NET CLR 1.1.4322)
88.154.28.65|-|-|27/May/2007|05:44:21|+0200|GET|/bugs/view.php?id=205|
HTTP/1.0|200|19456|http://le-gall.net/pierrick/blog/|Mozilla/4.0
(compatible; MSIE 5.0; Windows 98)
72.9.105.42|-|-|27/May/2007|08:53:01|+0200|GET|/forum/register.php|HT
TP/1.0|200|21669|http://www.talendforge.org/register.php|Mozilla/4.0
(compatible; MSIE 7.0b; Windows NT 6.0)
76.171.68.99|-|-|27/May/2007|19:12:33|+0200|GET|/forum/login.php|HTTP/
1.0|200|17445|http://talendforge.org/login.php|Mozilla/4.0
(compatible; MSIE 6.0; Windows NT 5.1) Opera 7.54 [en]
Job apach ended at 12:11 01/09/2008. [exit code=0]
```

The log lines of the defined file are displayed on the console.

tCreateTemporaryFile

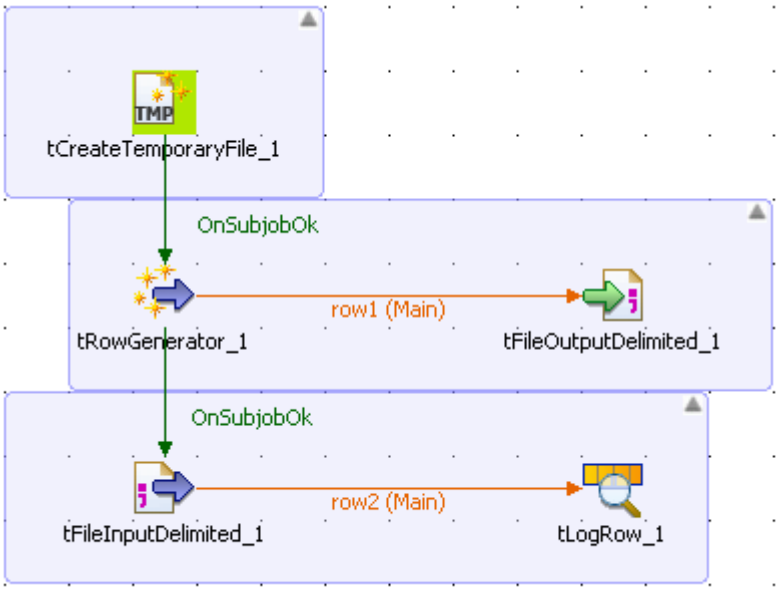
tCreateTempraryFile properties

Component family	File/Management	
Function	**tCreateTemporaryFile** creates and manages temporary files.	
Purpose	**tCreateTemporaryFile** helps to create a temporary file and puts it in a defined directory. This component allows either to keep the temporary file or to delete it after job execution.	
Basic settings	*Remove file when execution is over*	Select this check box to delete the temporary file after job execution.
	Directory	Enter the path to the directory where temporary files are stored.
	Template	Enter a name to the temporary file respecting the template.
	Suffix	Enter the filename extension to indicate the file format you want to give to the temporary file.
Usage	**tCreateTemporaryFile** provides the possibility to manage temporary files so that the memory can be freed for other ends and thus optimizes system performance.	

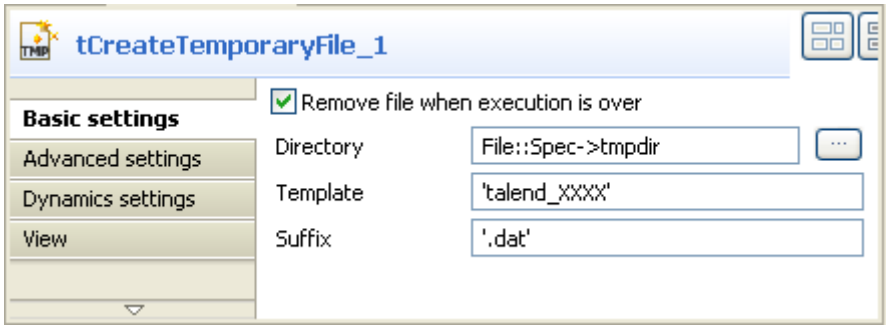
Scenario: Creating a temporary file and writing data in it

The following Perl scenario describes a simple Job that creates an empty temporary file in a defined directory, writes data in it and deletes it after job execution.

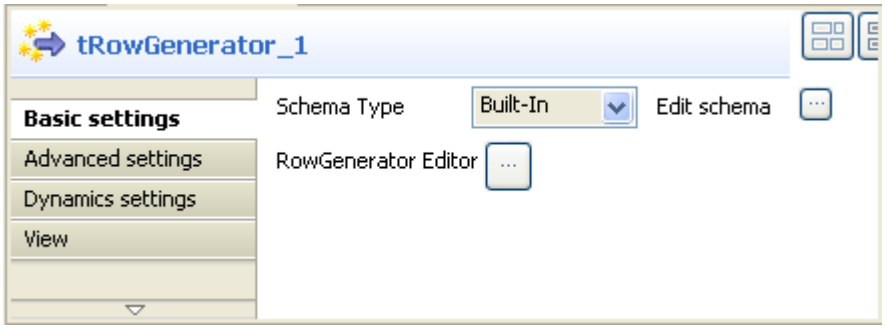
- Drop the following components from the **Palette** onto the design workspace: **tCreate temporaryFile**, **tRowgenerator**, **tFileOutputDelimited**, **tFileInputdelimited** and **tLogFile**.
- Connect **tCreateTemporaryFile** to **tRowGenerator** using a **SubjobOk** link.
- Connect **tRowGenerator** to **tFileOutputDelimited** using a **Row Main** link.
- Connect **tRowGenerator** to **tFileInputDelimited** using a **SubjobOk** link.
- Connect **tFileInputDelimited** to **tLogRow** using a **Row Main** link.



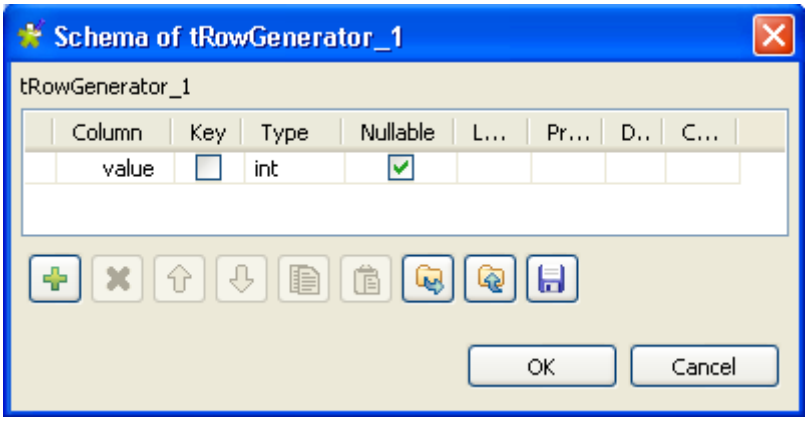
- In the design workspace, select **tCreateTemporaryFile**.
- Click the **Component** tab to define the basic settings for **tCreateTemporaryFile**.



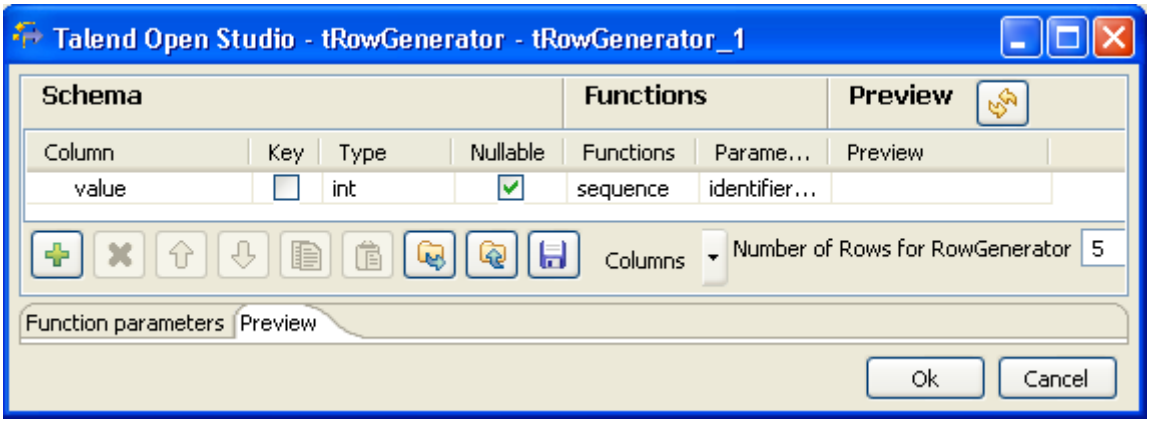
- Select the **Remove file when execution is over** check box to delete the created temporary file when job execution is over.
- Click the three-dot button next to the **Directory** field to browse to the directory where temporary files will be stored, or enter the path manually.
- In the **Template** field, enter a name for the temporary file respecting the template format.
- In the **Suffix** field, enter a filename extension to indicate the file format you want to give to the temporary file.
- In the design workspace, select **tRowgenerator** and click the **Component** tab to define its basic settings.



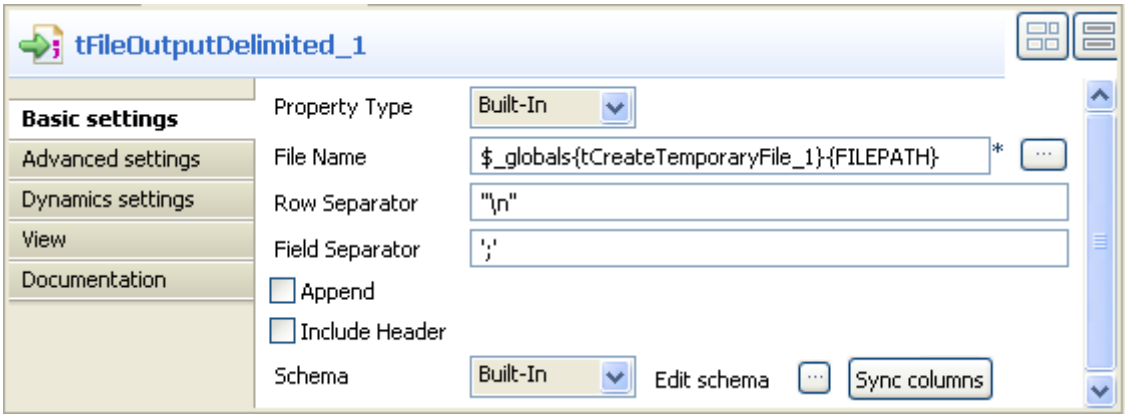
- Set the **Schema Type** to **Built-In**.
- Click the **Edit schema** three-dot button to define the data to pass on to the **tFileOutputDelimited** component, one column in this scenario, *value*.



- Click **OK** to close the dialog box.
- Click the **RowGenerator Editor** three-dot button to open the editor dialog box.



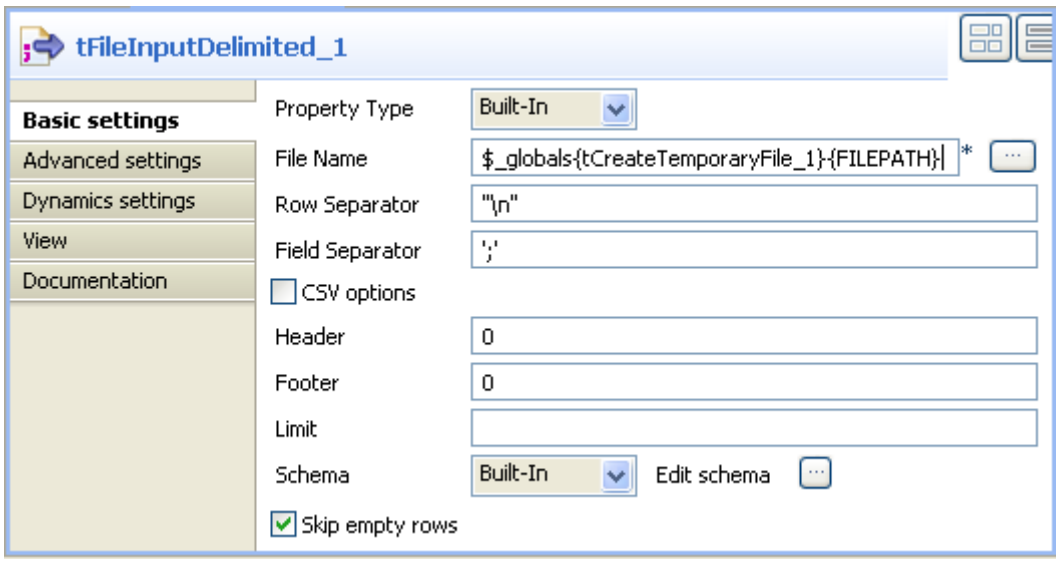
- In the **Number of Rows for Rowgenerator** field, enter 5 to generate five rows and click **Ok** to close the dialog box.
- In the design workspace, select **tFileOutputDelimited** and click the **Component** tab to define its basic settings.



- Set **Property Type** to **Built-In**.
- Click in the **File Name** field and use the **Ctrl+Space bar** combination to access the variable completion list. To output data in the created temporary file, select `$_globals{tCreateTemporaryFile_1}{FILEPATH}` on the Perl global variable list.
- Set the row and field separators in their corresponding fields as needed.
- Set **Schema** to **Built-In** and click **Sync columns** to synchronize input and output columns. Note that the row connection feeds automatically the output schema.

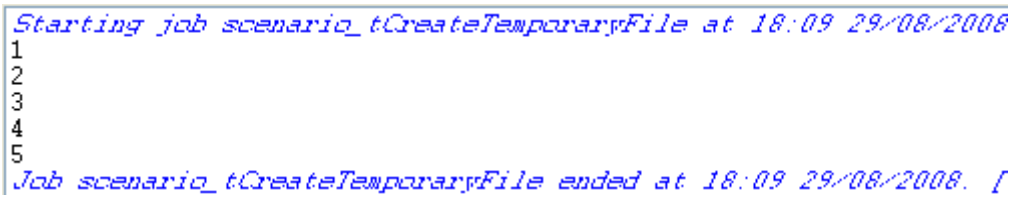
For more information about schema types, see *Setting a built-in schema* and *Setting a Repository schema* of **Talend Open Studio** User Guide.

- In the design workspace, select the **tFileInputDelimited** component.
- Click the **Component** tab to define the basic settings of **tFileInputDelimited**.



- Set property type to **Build in**.
- Click in the **File Name** field and use the **Ctrl+Space bar** combination to access the variable completion list. To read data in the created temporary file, select `$_globals{tCreateTemporaryFile_1}{FILEPATH}` on the Perl global variable list.

- Set the row and field separators in their corresponding fields as needed.
- Set **Schema** to **Built in** and click **Edit schema** to define the data to pass on to the **tLogRow** component. The schema consists of one column here, *value*.
- Save the Job and press **F6** to execute the Job.



The temporary file is created in the defined directory during job execution and the five generated rows are written in it. The temporary file is deleted when job execution is over.

tFileCompare

tFileCompare properties

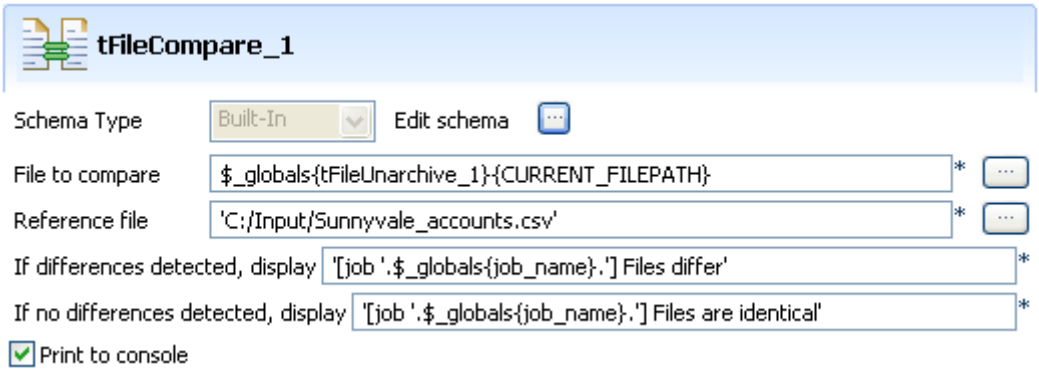
Component family	File/Management	
Function	Compares two files and provides comparison data (based on a read-only schema)	
Purpose	Helps at controlling the data quality of files being processed.	
Basic settings	*Schema type* and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either built-in or remotely stored in the Repository but in this case, the schema is read-only.
	File to compare	Filepath to the file to be checked.
	Reference file	Filepath to the file, the comparison is based on.
	If differences are detected, display *If no difference detected, display*	Type in a message to be displayed in the **Run** console based on the result of the comparison.
	Print to console	Select this check box to display the message.
Usage	This component can be used as standalone component but it is usually linked to an output component to gather the log data.	
Limitation	n/a	

Scenario: Comparing unzipped files

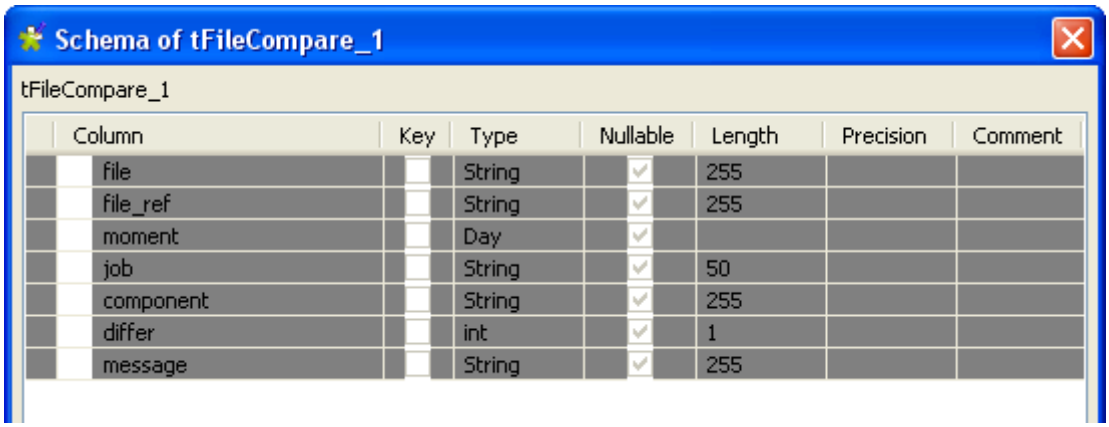
This scenario describes a Job unarchiving a file and comparing it to a reference file to make sure it didn't change. The output of the comparison is stored into a delimited file and a message displays in the console.

- Drag and drop the following components: **tFileUnarchive**, **tFileCompare**, and **tFileOutputDelimited**.
- Link the **tFileUnarchive** to the **tFileCompare** with **Iterate** connection.
- Connect the **tFileCompare** to the output component, using a **Main** row link.
- In the **tFileUnarchive** component **Basic settings**, fill in the path to the archive to unzip.

- In the **Extraction Directory** field, fill in the destination folder for the unarchived file.
- In the **tFileCompare Basic settings**, set the **File to compare**. Press *Ctrl+Space bar* to display the list of global variables. Select *$_globals{tFileUnarchive_1}{CURRENT_FILEPATH}* or *"((String)globalMap.get("tFileUnarchive_1_CURRENT_FILEPATH"))"* according to the language you work with, to fetch the file path from the **tFileUnarchive** component.



- And set the **Reference file** to base the comparison on it.
- In the messages fields, set the messages you want to see in case the files differ or in case the files are identical, for example: '[job '.$_globals{job_name}.'] Files differ' if you work with Perl or *"[job " + jobName + "] Files differ"* if you work in Java.
- Select the **Print to Console** check box, for the message defined to display at the end of the execution.
- The schema is read-only and contains standard information data. Click **Edit schema** to have a look to it.



- Then set the output component as usual with semi-colon as data separators.
- Save your Job and press **F6** to run it.

```
Starting job CompareFiles at 14:11 19/06/2007.
[job CompareFiles] Files differ
Job CompareFiles ended at 14:11 19/06/2007. [exit code=0]
```

The message set is displayed to the console and the output shows the schema information data.

```
File;file_ref;moment;job;component;differ;message
C:\Input\Accounts\Sunnyvale_accounts_new.xls;C:/Input/Sunnyvale_accounts.csv;
2007-06-19 14:11:59;CompareFiles;tFileCompare_1;1;[job CompareFiles] Files
differ
```

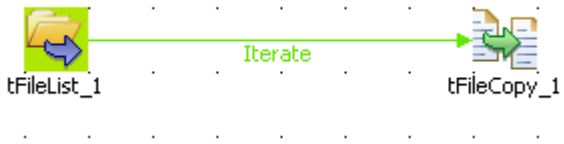
tFileCopy

tFileCopy Properties

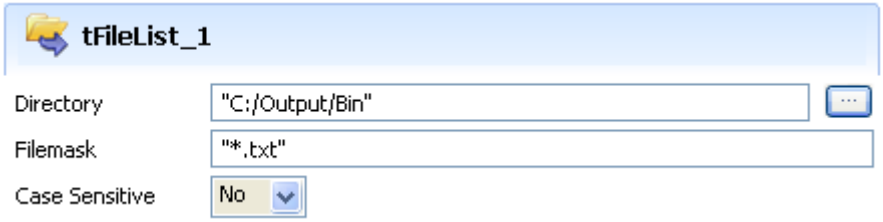
Component family	File/Management	
Function	Copies a source file into a target directory and can remove the source file if so defined.	
Purpose	Helps to streamline processes by automating recurrent and tedious tasks such as copy.	
Basic settings	*File Name*	Path to the file to be copied or moved
	Destination	Path to the directory where the file is copied/moved to.
	Remove source file	Select this check box to move the file to the destination.
	Replace existing file	Select this check box to overwrite any existing file with the newly copied file.
Usage	This component can be used as standalone component.	
Limitation	n/a	

Scenario: Restoring files from bin

This scenario describes a Job that iterates on a list of files, copies each file from the defined source directory to a target directory. It then removes the copied files from the source directory.

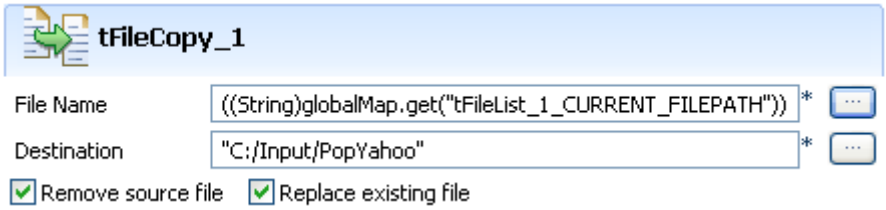


- Drop a **tFileList** and a **tFileCopy** from the **Palette** to the design workspace.
- Link both components using an **Iterate** link.
- In the **tFileList Basic settings**, set the directory for the iteration loop.



- Set the **Filemask** to *"*.txt"* to catch all files with this extension. For this use case, the case is not sensitive.

- Then select the **tFileCopy** to set its **Basic settings**.



- In the File Name field, press Ctrl+Space bar to access the list of variables.
- Select the global variable *((String)globalMap.get("tFileList_1_CURRENT_FILEPATH"))* if you work in Java, or *$_globals{tFileList_1}{CURRENT_FILEPATH}* if you work in Perl. This way, all files from the source directory can be processed.
- Select the **Remove Source file** check box to get rid of the file that have been copied.
- Select the **Replace existing file** check box to overwrite any file possibly present in the destination directory.
- Save your Job and press **F6**.

The files are copied onto the destination folder and are removed from the source folder.

tFileDelete

tFileDelete Properties

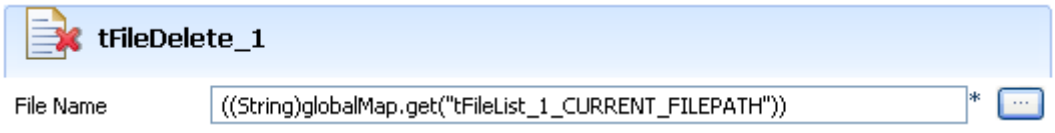
Component family	File/Management	
Function	Suppresses a file from a defined directory.	
Purpose	Helps to streamline processes by automating recurrent and tedious tasks such as delete.	
Basic settings	*File Name*	Path to the file to be copied or moved
Usage	This component can be used as standalone component.	
Limitation	n/a	

Scenario: Deleting files

This very simple scenario describes a Job deleting files from a given directory.

- Drop the following components: **tFileList**, **tFileDelete**, **tJava** from the **Palette** to the design workspace.
- In the **tFileList Basic settings**, set the directory to loop on in the **Directory** field.

- The filemask is *"*.txt"* and no case check is to carry out.
- In the **tFileDelete Basic settings** panel, set the **File Name** field in order for the current file in selection in the **tFileList** component be deleted. This allows to delete all files contained in the directory defined earlier on.



- press **Ctrl+Space bar** to access the list of global variables. In Java, the relevant variable to collect the current file is: ((String)globalMap.get("tFileList_1_CURRENT_FILEPATH")).
- Then in the **tJava** component, define the message to be displayed in the standard output (Run console). In this Java use case, type in the Code field, the following script:
  System.out.println(((String)globalMap.get("tFileList_1_CURRENT_FILE"))
  + " has been deleted!");
- Then save your Job and press **F6** to run it.

```
Starting job FileDel at 18:29 20/06/2007.
16.txt has been deleted!
15.txt has been deleted!
14.txt has been deleted!
13.txt has been deleted!
12.txt has been deleted!
11.txt has been deleted!
10.txt has been deleted!
09.txt has been deleted!
08.txt has been deleted!
07.txt has been deleted!
06.txt has been deleted!
05.txt has been deleted!
04.txt has been deleted!
03.txt has been deleted!
02.txt has been deleted!
01.txt has been deleted!
Job FileDel ended at 18:29 20/06/2007. [exit code=0]
```

The message set in the **tJava** component displays in the log, for each file that has been deleted through the **tFileDelete** component.

tFileExist

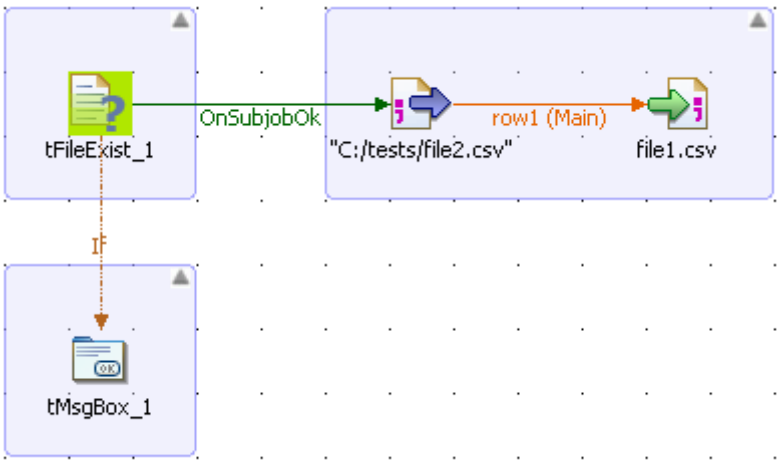
tFileExist Properties

Component family	File/Management	
Function	**tFileExist** checks if a file exists or not.	
Purpose	**tFileExists** helps to streamline processes by automating recurrent and tedious tasks such as checking if a file exists.	
Basic settings	*File Name*	Path to the file you want to check if it exists or not.
Usage	This component can be used as standalone component.	
Limitation	n/a	

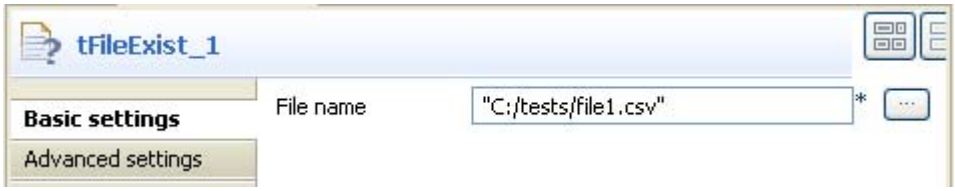
Scenario: Checking for the presence of a file and creating it if it does not exist

This scenario describes a simple Job that: checks if a given file exists, displays a graphical message to confirm that the file does not exist, reads the input data in another given file and writes it in an output delimited file.

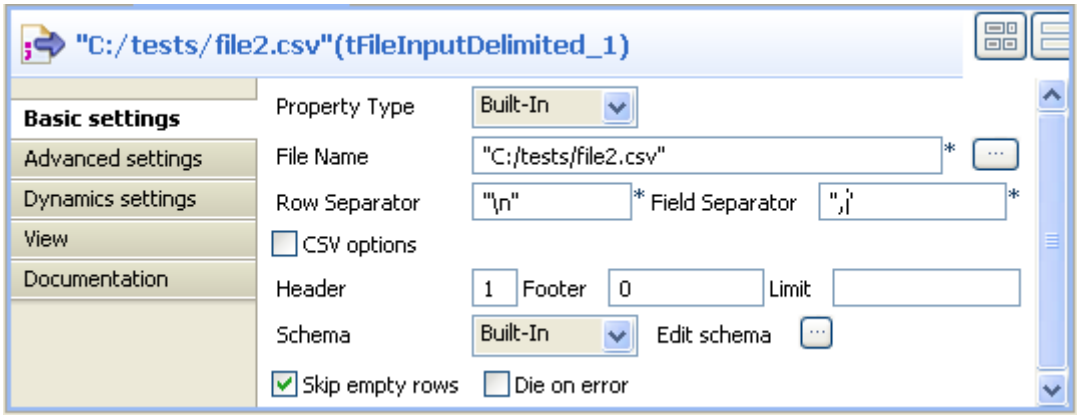
- Drop the following components from the **Palette** onto the design workspace: **tFileExist**, **tFileInputDelimited**, **tFileOutputDelimited**, and **tMsgBox**.
- Connect **tFileExist** to **tFile InputDelimited** using an **OnSubjobOk** and to **tMsBox** using a **Run If** link.



- Connect **tFileInputDelimited** to **tFileOutputDelimite** using a **Row Main** link.
- In the design workspace, select **tFileExist** and click the **Component** tab to define its basic settings.

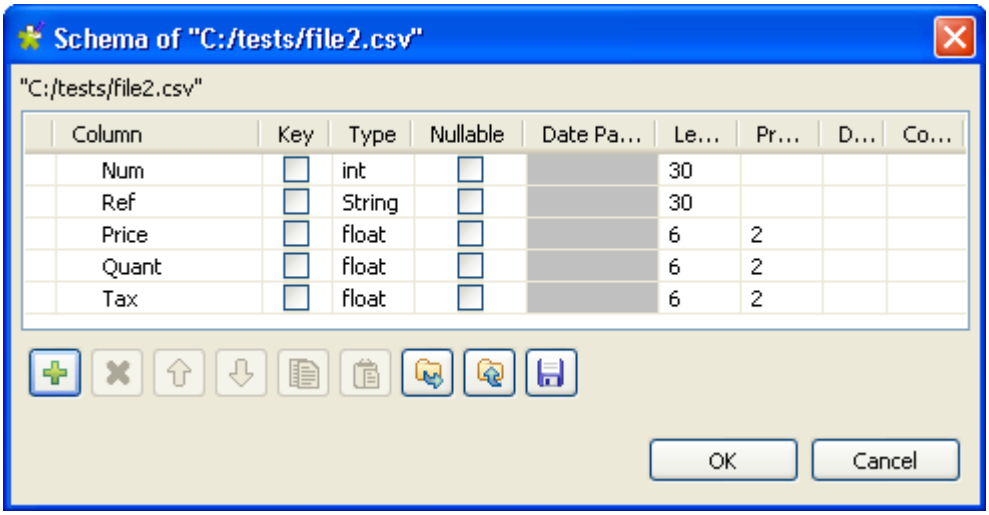


- In the **File name** field, enter the file path or browse to the file you want to check if it exists or not.
- In the design workspace, select **tFileInputDelimited** and click the **Component** tab to define its basic settings.



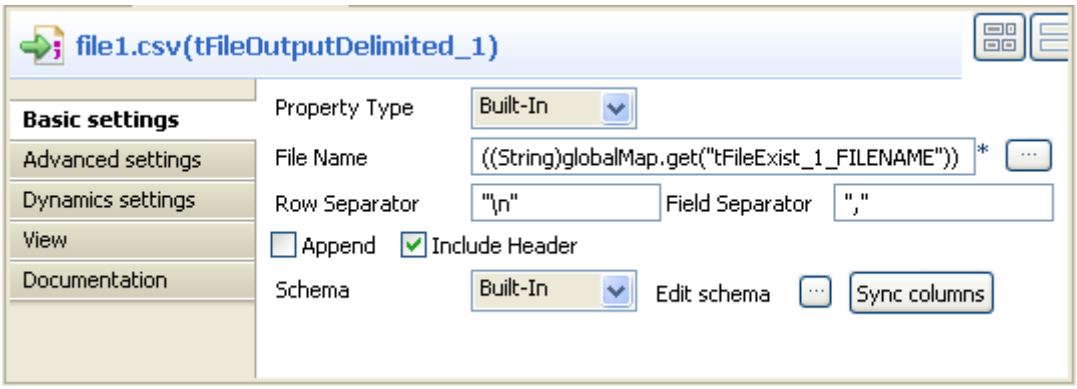
- Browse to the input file you want to read to fill out the **File Name** field.
- Set the row and field separators in their corresponding fields.
- Set the header, footer and number of processed rows as needed. In this scenario, there is one header in our table.
- Set **Schema** to **Built in** and click the **Edit schema** button to define the data to pass on to the **tFileOutputDelimited** component. Define the data present in the file to read, file2 in this scenario.

For more information about schema types, see *Setting a built-in schema* and *Setting a Repository schema* of **Talend Open Studio** User Guide.

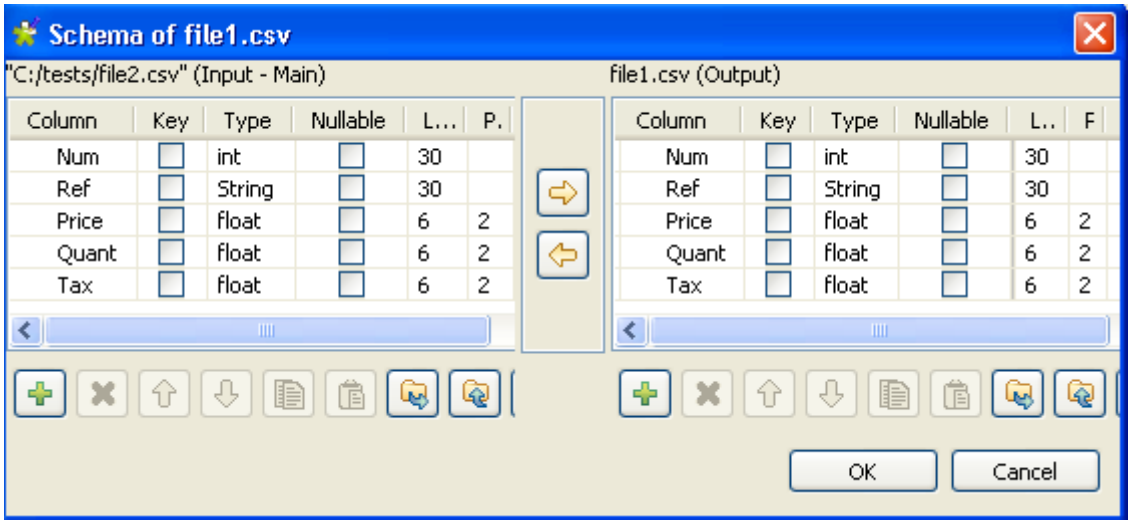


The schema in file2 consists of five columns: *Num*, *Ref*, *Price*, *Quant*, and *tax*.

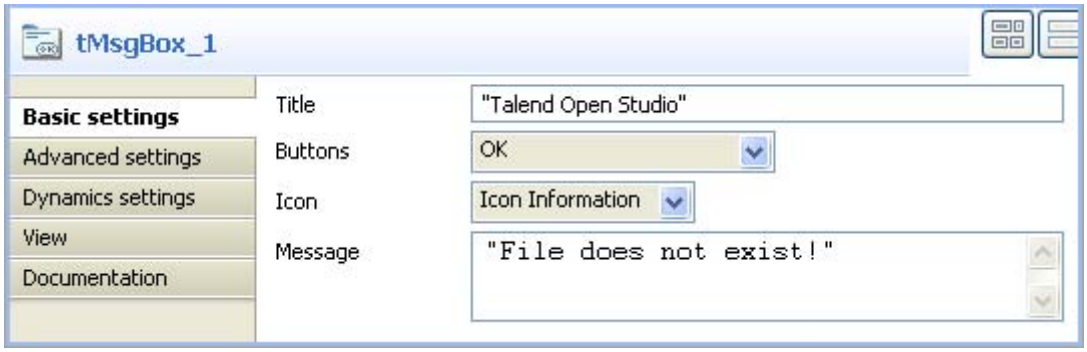
- In the design workspace, select the **tFileOutputDelimited** component.
- Click the **Component** tab to define the basic settings of **tFileOutputDelimited**.



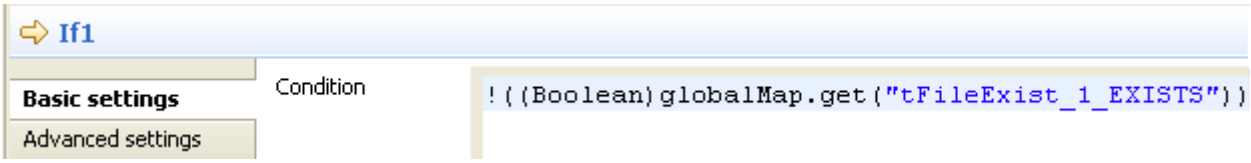
- Set property type to **Build in**.
- In the **File name** field, press **Ctrl+Space** to access the variable list and select the global variable FILENAME.
- Set the row and field separators in their corresponding fields.
- Select the **Include Header** check box as file2 in this scenario includes a header.
- Set **Schema** to **Built in** and click **Sync columns** to synchronize the output file schema (file1) with the input file schema (file2).



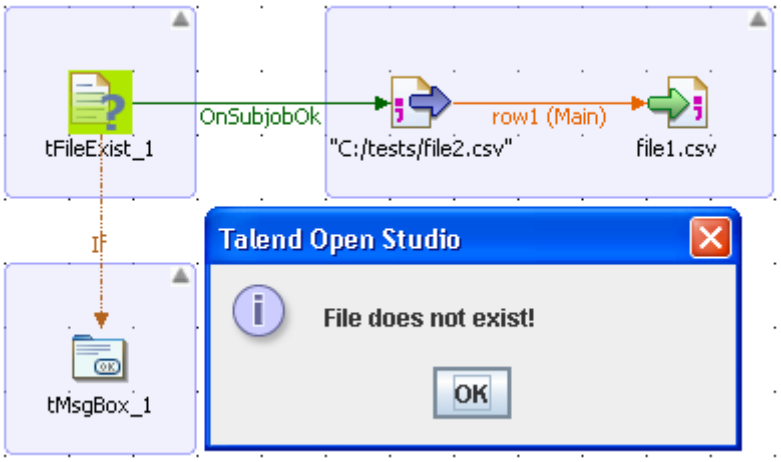
- In the design workspace, select the **tMsBox** component.
- Click the **Component** tab to define the basic settings of **tMsBox**.



- Click the **If** link to display its properties in the **Basic settings** view.
- In the **Condition** panel, press **Ctrl+Space** to access the variable list and select the global variable EXISTS. Type an exclamation mark before the variable to negate the meaning of the variable.



Save your Job and press **F6** to execute it.



A dialog box appears to confirm that the file does not exists.

Click **OK** to close the dialog box and continue the job execution process. The missing file, file1 in this scenario, got written in a delimited file in the defined place.

tFileFetch

tFileFetch belongs to two component families: File and Internet. For more information on it, see *tFileFetch on page 548*.

tFileInputDelimited

tFileInputDelimited properties

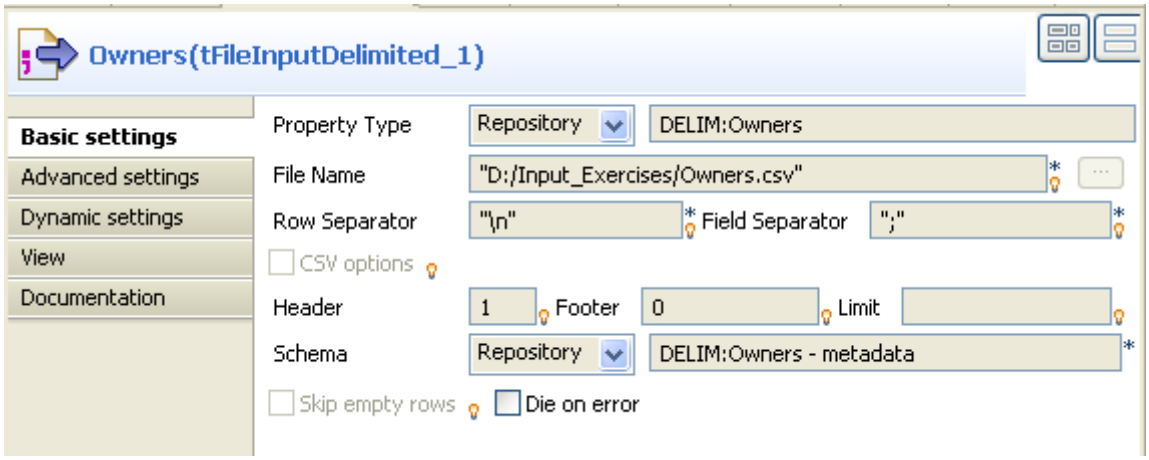
Component family	File/Input	
Function	**tFileInputDelimited** reads a given file row by row with simple separated fields.	
Purpose	Opens a file and reads it row by row to split them up into fields then sends fields as defined in the Schema to the next job component, via a Row link.	
Basic settings	*Property type*	Either Built-in or Repository.
		Built-in: No property data stored centrally.
		Repository: Select the Repository file where Properties are stored. The following fields are pre-filled in using fetched data.
	File Name	Name of the file to be processed. Related topic:*Defining variables from the Component view* of **Talend Open Studio** User Guide
	Row separator	String (ex: “\n”on Unix) to distinguish rows.
	Field separator	Character, string or regular expression to separate fields.
	CSV options	Select this check box to include CSV specific parameters such as **Escape char** and **Text enclosure**.
	Header	Number of rows to be skipped in the beginning of file
	Footer	Number of rows to be skipped at the end of the file.
	Limit	Maximum number of rows to be processed. If Limit = 0, no row is read or processed.
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the Repository. Click **Edit Schema** to make changes to the schema. Note that if you make changes, the schema automatically becomes built-in. Click **Sync columns** to retrieve the schema from the previous component connected in the Job.
		Built-in: The schema will be created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused in various projects and job flowcharts. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	Skip empty rows	Select this check box to skip empty rows.
	Die on error	Clear this check box to skip the row on error and complete the process for error-free rows.

Advanced settings	*Advanced separator (for number)*	Select this check box to modify the separators used for numbers.
	Extract lines at random	Select this check box to set the number of lines to be extracted randomly.
	Encoding Type	Select the encoding type from the list or select **Custom** and define it manually. This field is compulsory for DB data handling.
	Trim all columns	Select this check box to remove leading and trailing whitespace from all columns.
	Check each row structure against schema	Select this check box to synchronize every row against the input schema.
	Check columns to trim	Select the check box next to the column name you want to remove leading and trailing whitespace from.
	Split row before field	Select this check box to split rows before splitting fields.
	tStatCatcher Statistics	Select this check box to gather the processing metadata at the Job level as well as at each component level.
Usage	Use this component to read a file and separate fields contained in this file using a defined separator.	

Scenario: Delimited file content display

The following scenario creates a two-component Job, which aims at reading each row of a file, selecting delimited data and displaying the output in the **Run** log console.

- Drop a **tFileInputDelimited** component from the **Palette** to the design workspace.
- Drop a **tLogRow** component the same way.
- Right-click on the **tFileInputDelimited** component and select *Row* > *Main*. Then drag it onto the **tLogRow** component and release when the plug symbol shows up.
- Select the **tFileInputDelimited** component again, and define its Basic settings:



- Fill in a path to the file in the **File Name** field. This field is mandatory.
- Define the **Row separator** allowing to identify the end of a row. Then define the **Field separator** used to delimit fields in a row.
- In this scenario, the header and footer limits are not set. And the **Limit** number of processed rows is set on 50.
- Select either a local (**Built-in**) or a remotely managed (**Repository**) **Schema type** to define the data to pass on to the **tLogRow** component.
- You can load and/or edit the schema via the **Edit Schema** function.

Related topics: *Setting a built-in schema* and *Setting a Repository schema* of **Talend Open Studio** User Guide.

- As selected, the empty rows will be ignored.
- Enter the encoding standard the input file is encoded in. This setting is meant to ensure encoding consistency throughout all input and output files.
- Select the **tLogRow** and define the **Field separator** to use for the output display. Related topic: *tLogRow on page 628*.
- Select the **Print schema column name in front of each value** check box to retrieve the column labels in the output displayed.
- Go to **Run** tab, and click on **Run** to execute the Job.

The file is read row by row and the extracted fields are displayed on the Run log as defined in both components **Basic settings**.

```
Starting job FileDel at 10:45 12/04/2007.
ID_Owner: 2|Name: lebouh|ID_Insurance: PKI2906
ID_Owner: 3|Name: bouhnan|ID_Insurance: BNU9147
ID_Owner: 4|Name: hirtle|ID_Insurance: TEV8360
ID_Owner: 5|Name: bobouh|ID_Insurance: WPM3151
ID_Owner: 6|Name: carbone|ID_Insurance: IFY9885
```

The Log sums up all parameters in a header followed by the result of the Job.

tFileInputMSDelimited

tFileInputMSDelimited belongs to two component families: File and MultiSchema. For more information on **tFileInputMSDelimited**, see *tFileInputMSDelimited on page 662*.

tFileInputEBCDIC

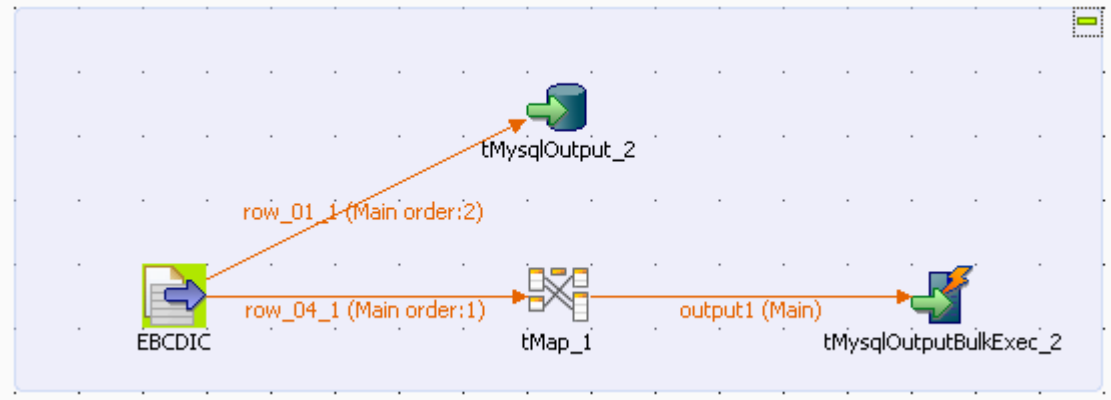
tFileInputEBCDIC properties

Component family	File/Input	
Function	**tFileInputEBCDIC** reads an EBCDIC file and extracts data depending on the selected schema.	
Purpose	**tFileInputEBCDIC** opens a file and reads it in order to separate the data, based on the file structure description (schemas), and to send the file data and metadata to the next job component(s), via a Row link.	
Basic settings	*Property type*	Either **Built-in** or **Repository**.
		Built-in: No property data stored centrally.
		Repository: Select the repository file where Properties are stored. The fields that come after are pre-filled in using the fetched data.
	Schema(s)	Add the various schemas to output to the next job component(s).
	Data file	Select the EBCDIC file containing the data to be processed.
	Xc2j file	Select the xc2j file, transforming the EBCDIC schema(s) into an intermediary XML file.
Usage	Use this component to read an EBCDIC file and to output the data separately depending on the schemas identified in the file.	

Scenario: Extracting data from an EBCDIC file and populating a database

This scenario uses the **[Copybook Connection]** wizard that guides users through the different steps necessary to create a Copybook connection and to retrieve the EBCDIC schemas. This wizard is available only for **Talend Integration Suite** users. If you are using **Talend Open Studio** or **Talend On Demand**, you need to set the basic settings for the **tFileInputEBCDIC** component manually.

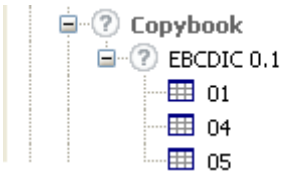
The following scenario is a four-component Job that aims at: reading an EBCDIC file which contains information concerning clients and their financial transactions, extracting and transforming this data, and finally creating two tables in a database, based on the two schemas, clients and transactions, extracted from the original EBCDIC file.



This Java scenario uses the EBCDIC Connection wizard to set up a connection to the Copybook file and to generate an xc2j file, which allows the retrieval and transformation of the different file schemas.

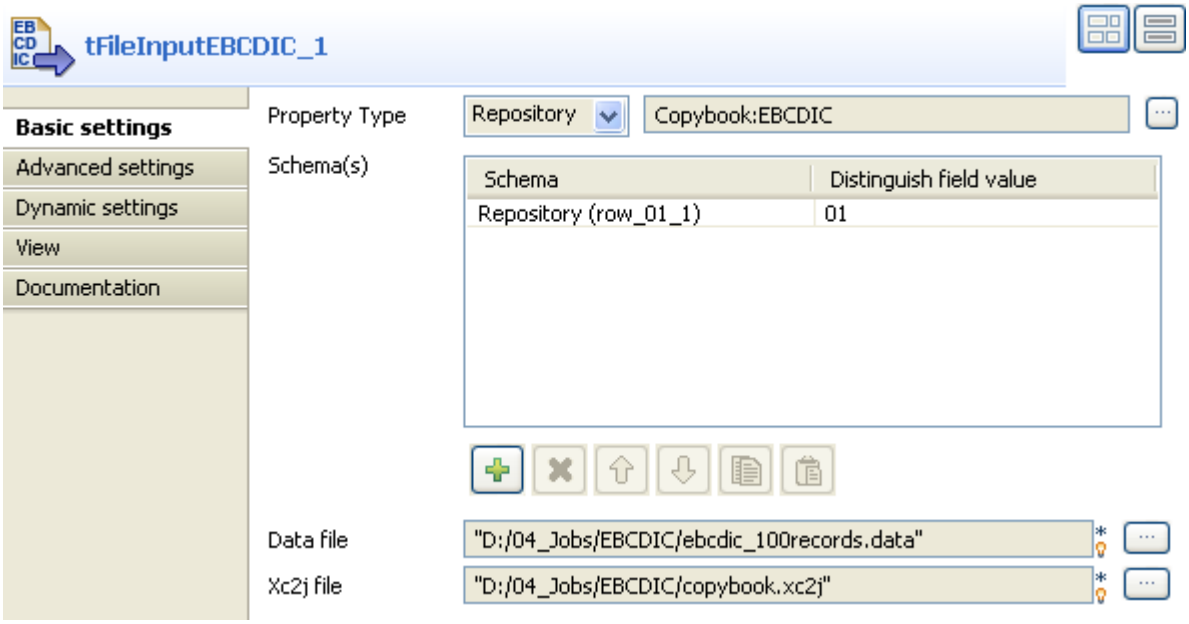
- Create a connection to the Copybook file, which describes the structure of your EBCDIC file. In this scenario, the Copybook connection is called EBCDIC.
- Retrieve the file schemas.

Once the Copybook connection has been created and the schemas retrieved, using the EBCDIC and Schema wizards, the new schemas appear under the node **Metadata** > **Copybook**. They are called *01*, *04* and *05*.



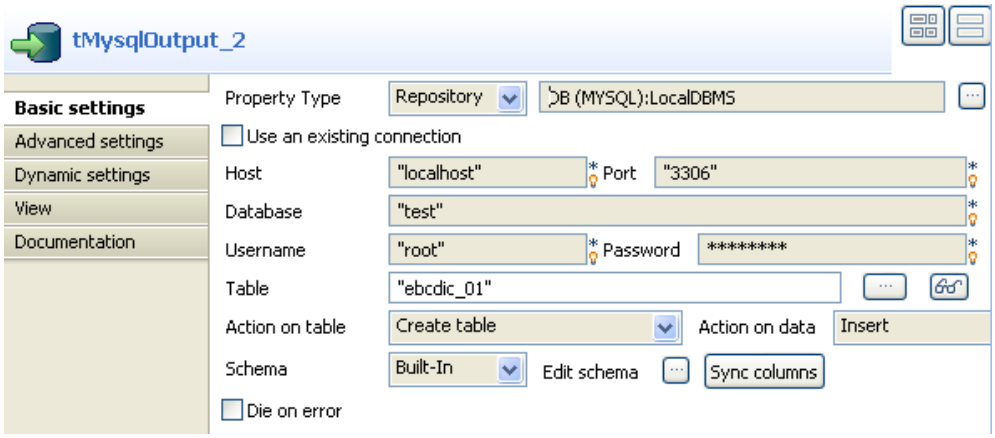
In order to retrieve the different file structures and to use them in **Talend Open Studio**:

- Drop schema 01 from the **Repository** tree view to the design workspace. This automatically creates the **tFileInputEBCDIC** input component.
- Drop the **tMysqlOutput** component from the **Palette** to the design workspace.
- Double-click **tFileInputEBCDIC** to display the **Basic settings** view, then define the component properties:



The metadata is automatically defined in the **Property Type**, **Schema(s)**, **Data file** and **Xc2j file** fields. The **Property Type** field shows which metadata has been used for the component. The **Schema field** shows which schema will be transmitted to the following component. The **Data file** field shows the path to the file that holds the EBCDIC data. The **Xc2j file** field shows the path to the file which enables to extract the schema describing the EBCDIC file structure. If you are in **Built-In** mode, you have to fill these fields manually.

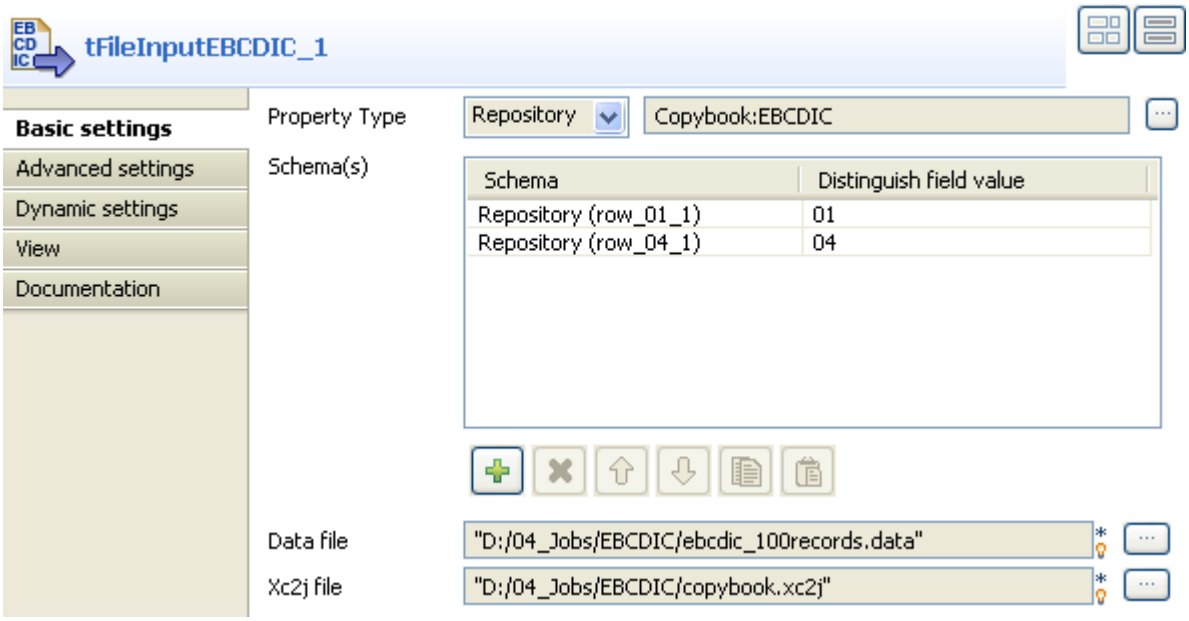
- In the design workspace, right-click **tFileInputEBCDIC**, select **Row** > **row_01_1** from the menu, then click **tMysqlOutput** to connect the components together.
- Double-click **tMysqlOutput** to display the **Basic settings** view, then define the component properties.



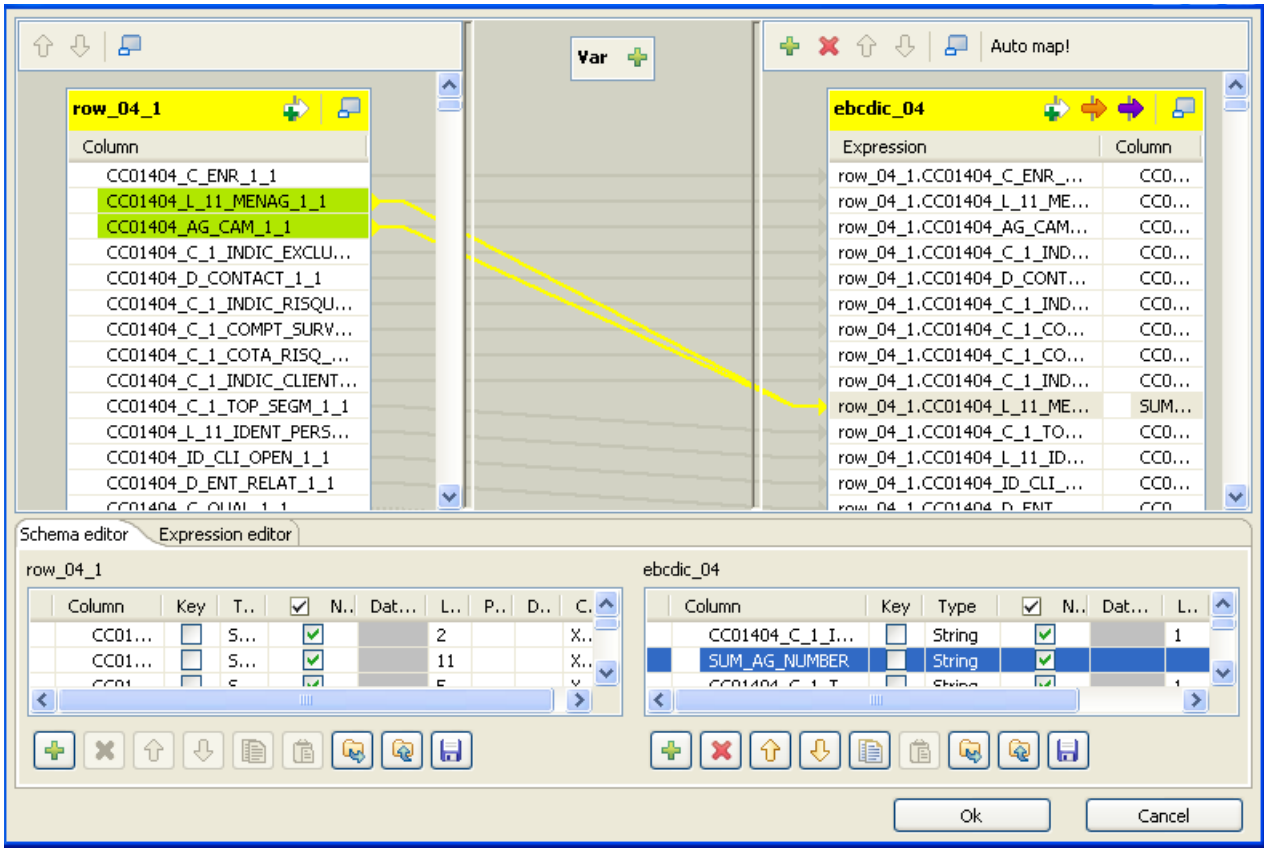
- In the **Property Type** list, select Repository and click the button **[...]**. Select the database connection you want to use, which is centralized in the metadata of the Repository. The **Host**, **Port**, **Database**, **Username** and **Password** fields are automatically filled. If you are in **Built-In** mode, you have to fill these fields manually.
- In the **Table** field, enter the name of the table to be created, which will contain the data extracted from the EBCDIC file.
- In the **Action on table** field, select the option **Create table**.

At this stage, the Job retrieves the schema 01 from the EBCDIC file and transfers it, as well as the corresponding data, to the database. We now need to retrieve, from the EBCDIC file, the schema 04 and its data, then transform and transmit the data to the same database. To do this:

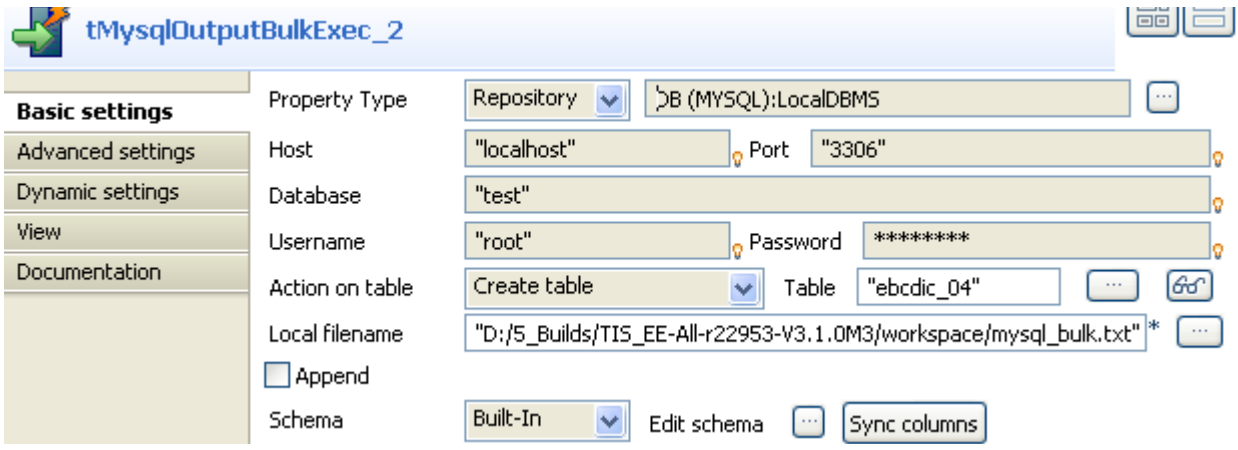
- Drop the **tMap** and **tMysqlOutputBulkExec** components to the design workspace.
- Double-click the **tFileInputEBCDIC** to display the **Basic settings** view, then define the component properties.



- In the **Schema(s)** field, click the plus button to add a line.
- Click in this line and then click the three-dot button that displays to open a dialog box. Select the **Create schema from repository** button to retrieve the schema defined in the EBCDIC metadata, then select 04 from the drop-down list.
- Click **OK** to close the dialog box.
- If you did not retrieve the schema from the **Repository** tree view, select **Create schema for built-in** and manually enter the name and description of your schema.
  The two schemas 01 and 04 appear in the Schema(s) field of the **tFileInputEBCDIC** component.
- In order to connect these two components, right-click **tFileInputEBCDIC**, select **Row** > **row_04_1** in the menu and click the **tMap** component. Then right-click **tMap**, drag a link over to **tMysqlOutputBulkExec** and release the right-click button. In the dialog box that opens up, fill in the name of the *ebcdic_04* output file.
- Double-click **tMap** to open up the **tMap Editor**.



- Select all the columns from the **row_04_1** table and drag them towards the **ebcdic_04** table.
- In the table **ebcdic_04**, located in the **Schema editor** area at the bottom of the editor, click the **plus** button to add a column to the schema. Name this column *SUM_AG_NUMBER*.
- In the table **row_04_1**, to the left of the editor, press **Ctrl** and select the **CC01404_L_11_MENAG_1_1** and **CC01404_AG_CAM_1_1** columns. Drag them to the new column *SUM_AG_NUMBER* in table **ebcdic_04**. Add the sign + between the two concatenated columns so that you have: `row_04_1.CC01404_L_11_MENAG_1_1 + row_04_1.CC01404_AG_CAM_1_1`.
- Click **OK**. to validate your changes and close the editor.
- In the design workspace, double-click **tMysqlOutputBulkExec** to display the **Basic settings** view, then define the component properties:



- In the **Property Type** list, select **Repository** and click the three-dot button to display a dialog bow where you can select the database connection you want to use, which is centralized in the **Metadata** folder of the **Repository** tree view. The **Host**, **Port**, **Database**, **Username** and **Password** fields are automatically filled. If you are in **Built-In** mode, you have to fill these fields manually.
- In the **Table** field, enter the name of the table to be created, which will contain the data extracted from the EBCDIC file.
- In the **Action on table** field, select the option **Create table**.
- Press **Ctrl+S** to save your Job and click the **Run** view. Select the **Statistics** and **Exec time** check boxes, then click **Run** to execute the Job.

The two tables are created in the database. They contain the structure, as well as the clients and transaction data, from the original EBCDIC file.

CC01394_C_ENR	CC01394_C_...	CC01394_IND...	CC01394_D_T...	CC01394_C_4_HH...	CC01394_D_D2_ARRE
01	891	E	20080519	2356	20080430

CC...	CC01404_L...	CC01404_D_...	CC01404_L_11_ID...	CC01404_...	CC014...	CC01404_L_C
04	00000003	20070512	00000035644	19990303	MME	WILSON
04	00000004	20080208	00000035647	19500101	MME	HARRISON
04	00000005		00000035654	19660608	MR	ADAMS
04	00000006	20080326	00000035670	19980804	MME	JACKSON
04	00000007	20080429	00000035671	19660722	MR	JACKSON
04	00000008	20071001	00000035692	19500101	MME	PIERCE
04	00000009	20080123	00000035693	19720826	MR	MCKINLEY

tFileInputExcel

tFileInputExcel properties

Component family	File/Input	 
Function	**tFileInputExcel** reads an Excel file (.xls or .xlsx) and extracts data line by line.	
Purpose	**tFileInputExcel** opens a file and reads it row by row to split data up into fields using regular expressions. Then sends fields as defined in the schema to the next component in the job via a **Row** link.	
Basic settings	*Property type*	Either **Built-in** or **Repository**.
		Built-in: No property data stored centrally.
		Repository: Select the repository file where Properties are stored. The fields that come after are pre-filled in using the fetched data.
		Click this icon to open a connection wizard and store the Excel file connection parameters you set in the component **Basic settings** view. For more information about setting up and storing file connection parameters, see *Setting up a File XML schema* of **Talend Open Studio** User Guide.
	File Name	Name and path of the .xls file to be processed. Related topic: *Defining variables from the Component view* of **Talend Open Studio** User Guide.
	All sheets	Select this check box to process all sheets of the Excel file.
	Sheet list	Click the plus button to add as many lines as needed to the list of the excel sheets to be processed: **Sheet (name or position)**: enter the name or position of the excel sheet to be processed. **Use Regex**: select this check box if you want to use a regular expression to filter the sheets to process.
	Header	Number of records to be skipped in the beginning of the file.
	Footer	Number of records to be skipped at the end of the file.
	Limit	Maximum number of lines to be processed.
	Affect each sheet(header&footer)	Select this check box if you want to apply the parameters set in the **Header** and **Footer** fields to all excel sheets to be processed.
	Die on error	Clear this check box to skip the row on error and complete the process for error-free rows.
	First column and *Last column*	Define the range of the columns to be processed through setting the first and last columns in the **First column** and **Last column** fields respectively.

	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the Repository. Click **Edit Schema** to make changes to the schema. Note that if you make changes, the schema automatically becomes built-in. Click **Sync columns** to retrieve the schema from the previous component connected in the Job.
		Built-in: The schema will be created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused in various projects and job flowcharts. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
Advanced settings	*Advanced separator*	Select this check box to change the used data separators.
	Encoding Type	Select the encoding from the list or select **Custom** and define it manually. This field is compulsory for DB data handling.
	Read real values for numbers	Select this check box to read numbers in real values.
	Stop to read on empty rows	Select this check box to ignore empty lines.
	Don't validate the cells	Select this check box to in order not to validate data.
	Ignore the warning	Select this check box to ignore all warnings generated to indicate errors in the Excel file.
	tStateCatcher Statistics	Select this check box to gather the job processing metadata at a job level as well as at each component level.
Usage	Use this component to read an Excel file and to output the data separately depending on the schemas identified in the file.	

Related scenarios

No scenario is available for this component yet.

tFileInputFullRow

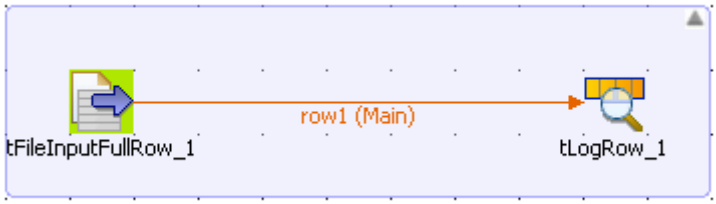
tFileInputFull Row properties

Component family	File/Input	
Function	**tFileInputFullRow** reads a given file row by row.	
Purpose	**tFileInputFullRow** opens a file and reads it row by row and sends complete rows as defined in the Schema to the next job component, via a Row link.	
Basic settings	*Schema type* and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the Repository. Click **Edit schema** to make changes to the schema. Note that if you make changes, the schema automatically becomes built-in. Click **Sync columns** to retrieve the schema from the previous component connected to **tFileInputFullRow**.
	File Name	Name of the file to be processed. Related topic:*Defining variables from the Component view* of **Talend Open Studio** User Guide.
	Row separator	String (ex: “\n”on Unix) to separate rows.
	Header	Number of rows to be skipped at the beginning of a file
	Footer	Number of rows to be skipped at the end of a file.
	Limit	Maximum number of rows to be processed. If Limit = 0, no row is read or processed.
	Skip empty rows	Select this check box to skip empty rows.
Usage	Use this component to read full rows in delimited files that can get very large.	

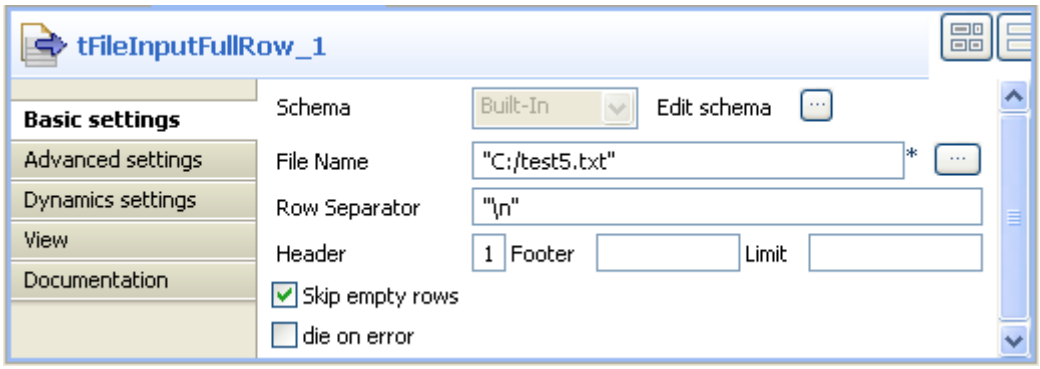
Scenario: Reading full rows in a delimited file

The following scenario creates a two-component Job that aims at reading complete rows in a file and displaying the output in the **Run** log console.

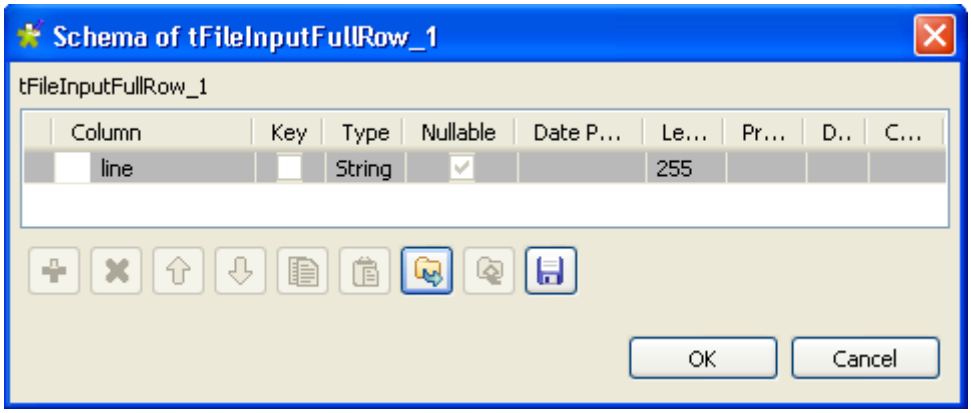
- Drop a **tFileInputFullRow** and a **tLogRow** from the **Palette** onto the design workspace.
- Right-click on the **tFileInputFullRow** component and connect it to **tLogRow** using a **Row Main** link.



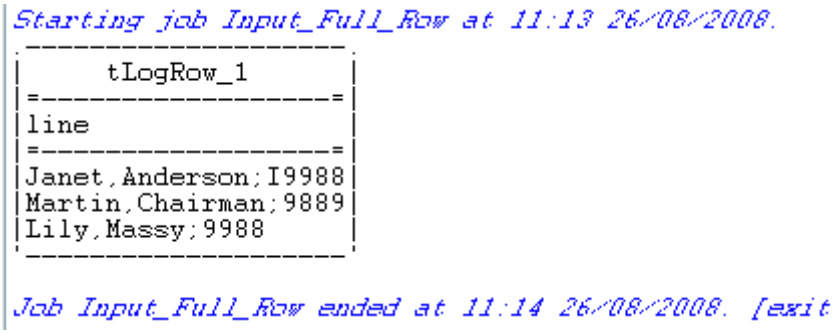
- In the design workspace, select **tFileInputFullRow**.
- Click the **Component** tab to define the basic settings for **tFileInputFullRow**.



- In the **Basic settings** view, set **Schema** to **Built-In**.
- Click the three-dot [...] button next to the **Edit schema** field to see the data to pass on to the **tLogRow** component. Note that the schema is read-only and it consists of one column, *line*.



- Fill in a path to the file to process in the **File Name** field, or click the three-dot [...] button. This field is mandatory. In this scenario, the file to read is *test5*. It holds three rows where each row consists of tow fields separated by a semi colon.
- Define the **Row separator** that allows to identify the end of a row.
- Set the **Header** to 1, in this scenario the footer and the number of processed rows are not set.
- In the design workspace, select **tLogRow** and click the **Component** tab to define its basic settings. For more information, see *tLogRow on page 628*.
- Save your Job and press **F6** to execute it.



tFileInputFullRow reads the three rows one by one ignoring field separators, and the complete rows are displayed on the **Run** console.

To extract only fields from rows, you must use **tExtractDelimitedFields**, **tExtractPositionalFields**, and **tExtractRegexFields**. For more information, see *tExtractDelimitedFields on page 736*, *tExtractPositionalFields on page 740* and *tExtractRegexFields on page 742*.

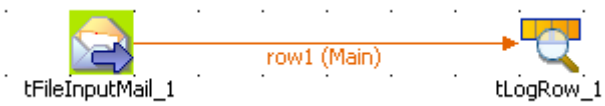
tFileInputMail

tFileInputMail properties

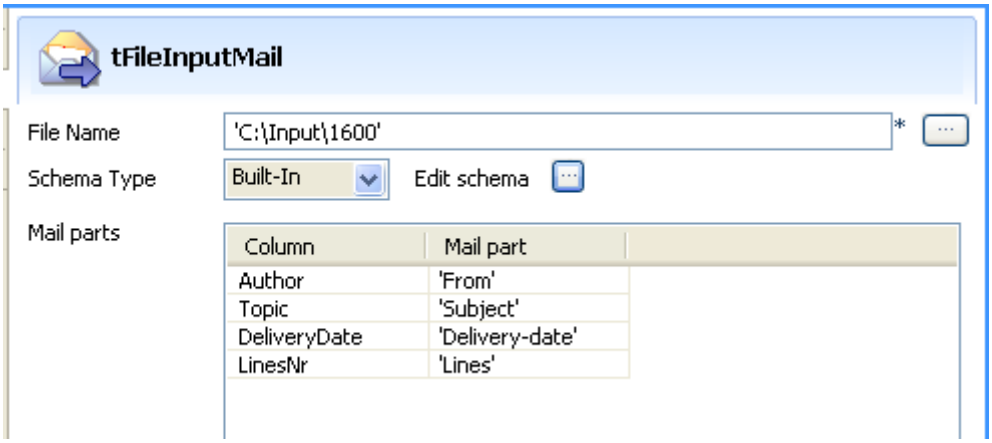
Component family	File/Input	
Function	reads the header and content parts of an email file defined	
Purpose	helps to extract standard key data from emails	
Basic settings	*File name*	Browse to the source email file
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the **Repository**. Click **Edit Schema** to make changes to the schema. Note that if you make changes, the schema automatically becomes built-in. Click **Sync columns** to retrieve the schema from the previous component connected in the Job.
		Built-in: The schema will be created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused in various projects and job flowcharts. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	Mail parts	**Column**: This field is automatically populated with the columns defined in the schema that you propagated.
		Mail part: Type in the label of the header part or body to be displayed on the output.
Usage	This component handles flow of data therefore it requires output, hence is defined as an intermediary step.	
Limitation	n/a	

Scenario: Extracting key fields from email

This two-component scenario is aimed at extracting some key standard fields and displaying the values on the **Run** console.



- Drop a **tFileInputMail** and a **tLogRow** component from the **Palette** to the design workspace.
- On the **Basic settings** tab, define the email parameters:



- Browse to the mail File to be processed. Define the schema including all columns you want to retrieve on your output.
- Once the schema is defined, click **OK** to propagate it into the **Mail parts** table
- On the **Mail part** column of the table, type in the actual header or body standard keys that will be used to retrieve the values to be displayed.
- Define the tLogRow in order for the values to be separated by a carriage return. On Windows OS, type in *\n* between double quotes.
- Press F6 to run the Job and display the output flow on the execution console.

```
Starting job FileInputMail at 11:42 18/01/2007.
"Brice L" <Brice.L@technologies.fr>
Re: Gestion de transactions
Tue, 16 Jan 2007 16:30:02 +0100
111
Job FileInputMail ended at 11:42 18/01/2007. [exit code=0]
```

The header key values are extracted as defined in the **Mail parts** table. Indeed, the author, topic, delivery date and number of lines are part of the output displayed.

tFileInputMSPositional

tFileInputMSPositional belongs to two component families: File and MultiSchema. For more information on **tFileInputMSPositional**, see *tFileInputMSDelimited on page 662*.

tFileInputPositional

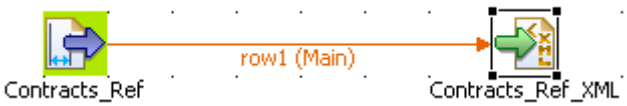
tFileInputPositional properties

Component family	File/Input	
Function	**tFileInputPositional** reads a given file row by row and extracts fields based on a pattern.	
Purpose	This component opens a file and reads it row by row to split them up into fields then sends fields as defined in the schema to the next job component, via a Row link.	
Basic settings	*Property type*	Either **Built-in** or **Repository**.
		Built-in: No property data stored centrally.
		Repository: Select the Repository file where Properties are stored. The fields that follow are pre-filled in using fetched data.
	File Name	Name of the file to be processed. Related topic:*Defining variables from the Component view* of **Talend Open Studio** User Guide
	Row separator	String (ex: “\n”on Unix) to distinguish rows.
	Customize	Select this check box to customize the data format of the positional file and define the table columns: **Column**: Select the column you want to customize. **Size**: Enter the column size. **Padding char**: Type in between quotes the padding characters used. A space by default. **Alignment**: Select the appropriate alignment parameter.
	Pattern	Length values separated by commas, interpreted as a string between quotes. Make sure the values entered in this field are consistent with the schema defined.
	Skip empty rows	select this check box to skip empty rows.
	Die on error	Clear this check box to skip the row on error and complete the process for error-free rows.
	Header	Number of rows to be skipped in the beginning of file
	Footer	Number of rows to be skipped at the end of the file.
	Limit	Maximum number of rows to be processed. If Limit = 0, no row is read or processed.
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the Repository.
		Built-in: The schema will be created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.

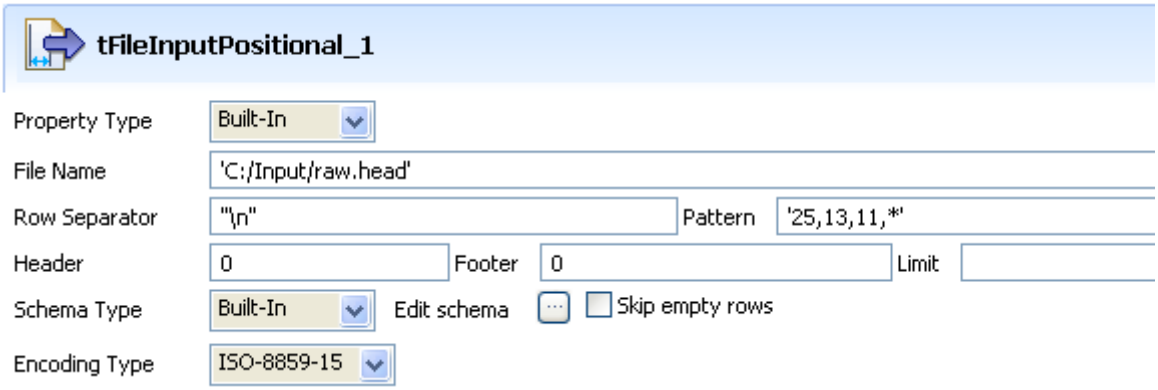
		Repository: The schema already exists and is stored in the Repository, hence can be reused in various projects and job flowcharts. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
Advanced settings	*Needed to process rows longer than 100 000 characters*	Select this check box if the rows to be processed in the input file are longer than 100 000 characters.
	Advanced separator	Select this check box to change the expected data separator.
	Trim all columns	Select this check box to remove leading and trailing whitespaces from defined columns.
	Encoding	Select the encoding type from the list or select **Custom** and define it manually. This field is compulsory for DB data handling.
	tStatCatcher Statistics	Select this check box to gather the job processing metadata at a job level as well as at each component level.
Usage	Use this component to read a file and separate fields using a position separator value.	

Scenario: From Positional to XML file

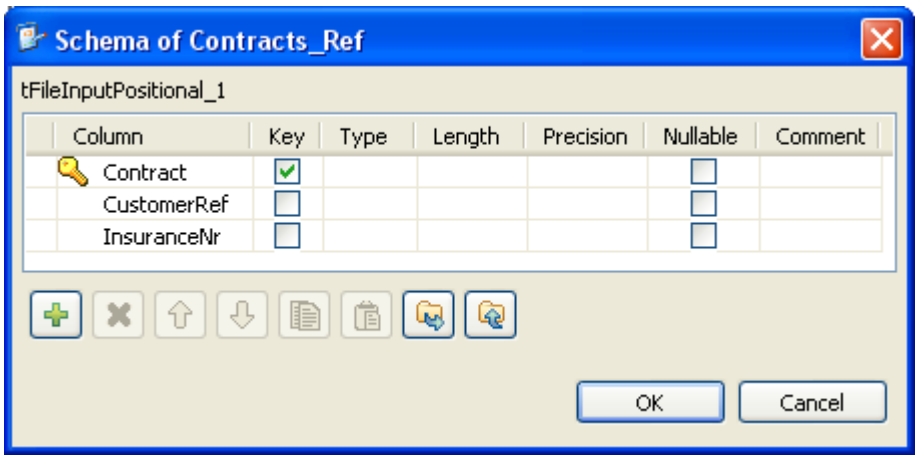
The following scenario creates a two-component Job, which aims at reading data of an Input file and outputting selected data (according to the data position) into an XML file.



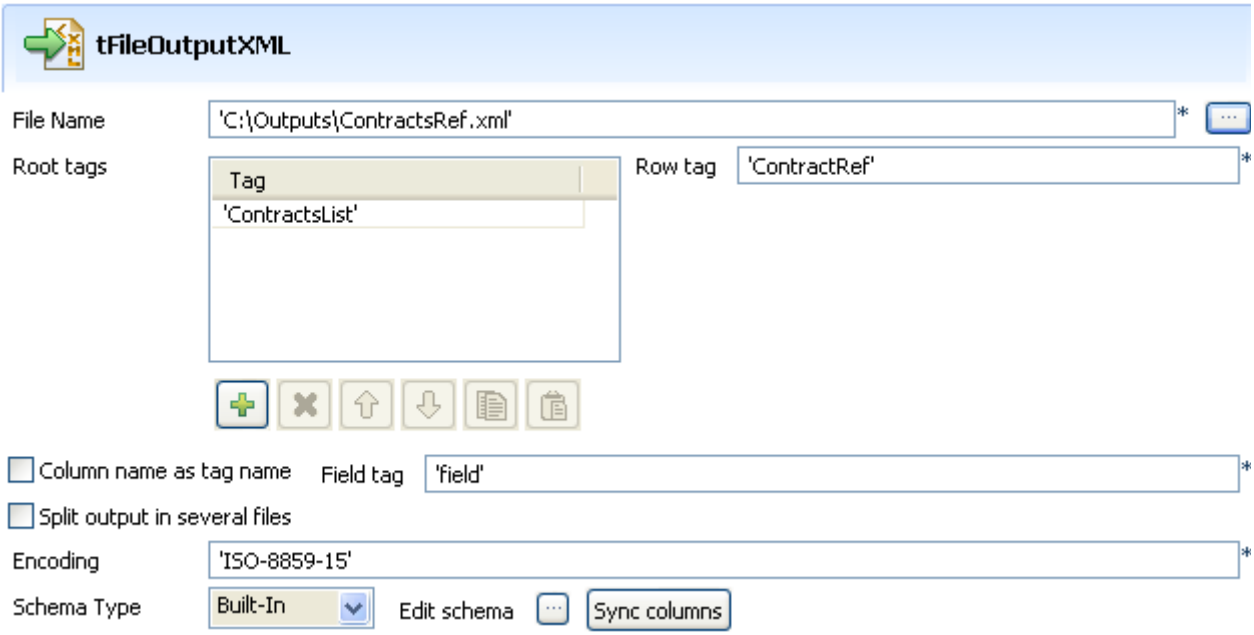
- Drop a **tFileInputPositional** component from the **Palette** to the design workspace. The file contains raw data, in this case, contract nr, customer references and insurance numbers.
- Drop a **tFileOutputXML** component as well. This file is meant to receive the references in a structured way.
- Right-click the **tFileInputPositional** component and select **Row** > **Main**. Then drag it onto the **tFileOutputXML** component and release when the plug symbol shows up.
- Select the **tFileInputPositional** component again, and define its properties.
- The job properties are built-in for this scenario. As opposed to the Repository, this means that the **Property type** is set for this station only.



- Fill in a path to the file in the **File Name** field. This field is mandatory.
- Define the **Row separator** identifying the end of a row, by default, a carriage return.
- Then define the **Pattern** to delimit fields in a row. The pattern is a series of length values corresponding to the values of your input files. The values should be entered between quotes, and separated by a comma. Make sure the values you enter match the schema defined.
- In this scenario, the header, footer and limit fields are not set. But depending on the input file structure, you may need to define them.
- Select a **Schema type** to define the data to pass on to the **tFileOutputXML** component.
- You can load and/or edit the schema via the **Edit Schema** function. For this schema, define three columns, respectively *Contracts*, *CustomerRef* and *InsuranceNr* matching the three value lengths defined.

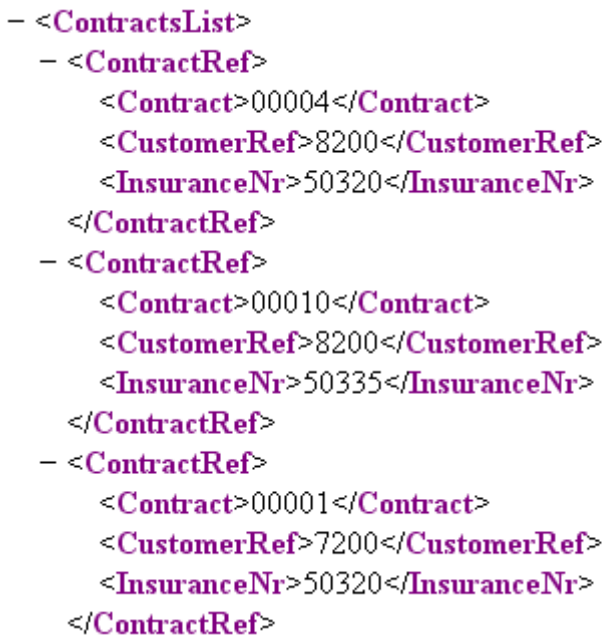


- Then define the second component Basic settings:
- Enter the XML output file path.



- Enter a root tag (or more), to wrap the XML structure output, in this case ‘ContractsList’.
- Define the row tag that will wrap each line data, in this case ‘ContractRef’.
- Select the **Column name as tag name** check box to reuse the column label from the input schema as tag label. By default, ‘field’ is used for each column value data.
- Enter the **Encoding** standard, the input file is encoded in. Note that, for the time being, the encoding consistency verification is not supported.
- Select the **Schema type**. If the row connection is already implemented, the schema is automatically synchronized with the Input file schema. Else, click on **Sync columns**.
- Go to the **Run** tab, and click on **Run** to execute the Job.

The file is read row by row and split up into fields based on the length values defined in the **Pattern** field. You can open it using any standard XML editor.


− <ContractsList>
− <ContractRef>
<Contract>00004</Contract>
<CustomerRef>8200</CustomerRef>
<InsuranceNr>50320</InsuranceNr>
</ContractRef>
− <ContractRef>
<Contract>00010</Contract>
<CustomerRef>8200</CustomerRef>
<InsuranceNr>50335</InsuranceNr>
</ContractRef>
− <ContractRef>
<Contract>00001</Contract>
<CustomerRef>7200</CustomerRef>
<InsuranceNr>50320</InsuranceNr>
</ContractRef>

tFileInputProperties

tFileInputProperties properties

Component family	File/Input	
Function	**tFileInputProperties** reads a text file row by row and extracts the fields.	
Purpose	**tFileInputProperties** opens a text file and reads it row by row then separates the fields according to the model key = value.	
Basic settings	*Schema type* and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remotely stored in the Repository but for this component, the schema is read-only. It is made of two column, *Key* and *Value*, corresponding to the parameter name and the parameter value to be copied.
	File format	Select from the list your file format, either: **.properties** or **.ini**.
		.properties: data in the configuration file is written in two lines and structured according to the following way: key = value.
		.ini: data in the configuration file is written in two lines and structured according to the following way: key = value and re-grouped in sections. **Section Name**: enter the section name on which the iteration is based.
	File Name	Name or path to the file to be processed. Related topic: *Defining variables from the Component view* of **Talend Open Studio** User Guide.
Usage	Use this component to read a text file and separate data according to the structure key = value.	

Scenario: Reading and matching the keys and the values of different .properties files and outputting the results in a glossary

This four-component Java Job reads two .properties files, one in French and the other in English. The data in the two input files is mapped to output a glossary matching the English and French terms.

The two input files used in this scenario hold localization strings for the **tMysqlInput** component in **Talend Open Studio**.

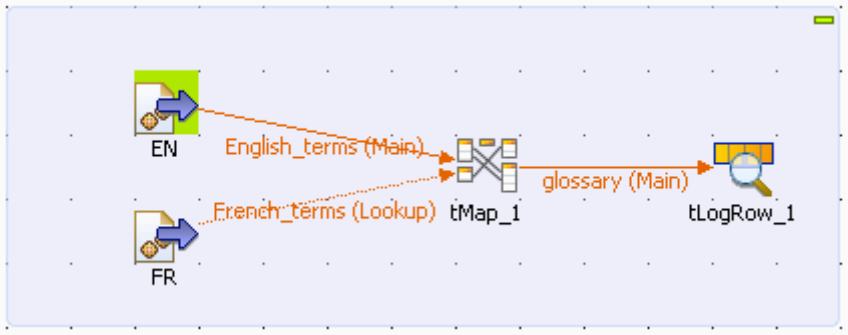
tMysqlInput_messages_fr.properties

```
1  DBD-ODBC.INFO=Requis pour les connexions
2  DBD-Oracle.INFO=Requis par Oracle
3  DBD-Pg.INFO=Requis par PostgreSQL
4  DBD-mysql.INFO=Requis par MySQL
5  DBNAME.NAME=Base de données
6  DBTABLE.NAME=Nom de table
7  ENABLE_STREAM.NAME=Activer la diffusion
8  ENCODING.NAME=Encodage
9  HELP=org.talend.help.tMysqlInput
10 HOST.NAME=Hôte
11 LONG_NAME=Lit une table MySQL et extrait
12 NB_LINE.NAME=Nombre de ligne
13 NULL_CHAR.NAME=Caractère Null
14 PASS.NAME=Mot de passe
15 PORT.NAME=Port
16 PROPERTIES.NAME=Paramètres JDBC additionn
17 QUERY.NAME=Requête
18 QUERYSTORE.NAME=Type de requête
19 SCHEMA.NAME=Schéma
20 SCHEMA_DB.NAME=Schéma
```

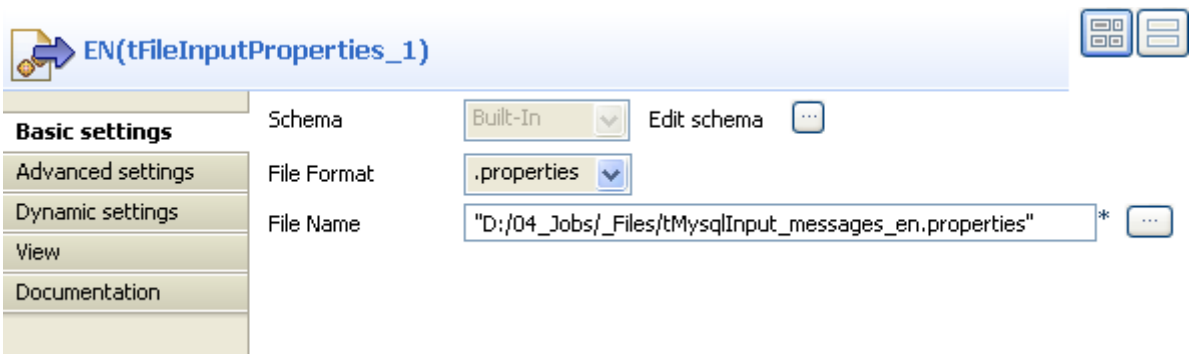
tMysqlInput_messages_en.properties

```
1  DBD-ODBC.INFO=Required for ODBC-like
2  DBD-Oracle.INFO=Required for Oracle
3  DBD-Pg.INFO=Required for PostgreSQL
4  DBD-mysql.INFO=Required for MySQL
5  DBNAME.NAME=Database
6  DBTABLE.NAME=Table Name
7  ENABLE_STREAM.NAME=Enable stream
8  ENCODING.NAME=Encoding
9  HELP=org.talend.help.tMysqlInput
10 HOST.NAME=Host
11 LONG_NAME=Reads a MySQL table and ext
12 NB_LINE.NAME=Number of line
13 NULL_CHAR.NAME=Null Char
14 PASS.NAME=Password
15 PORT.NAME=Port
16 PROPERTIES.NAME=Additional JDBC Param
17 QUERY.NAME=Query
18 QUERYSTORE.NAME=Query Type
19 SCHEMA.NAME=Schema
20 SCHEMA_DB.NAME=Schema
```

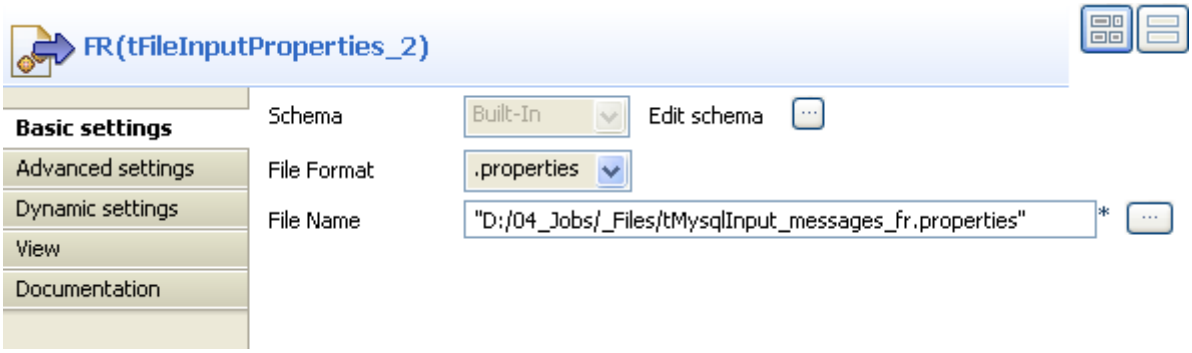
- Drop the following components from the **Palette** onto the design workspace: **tFileInputProperties** (x2), **tMap**, and **tLogRow**.
- Connect the component together using **Row** > **Main** links. The second properties file, *FR*, is used as a lookup flow.



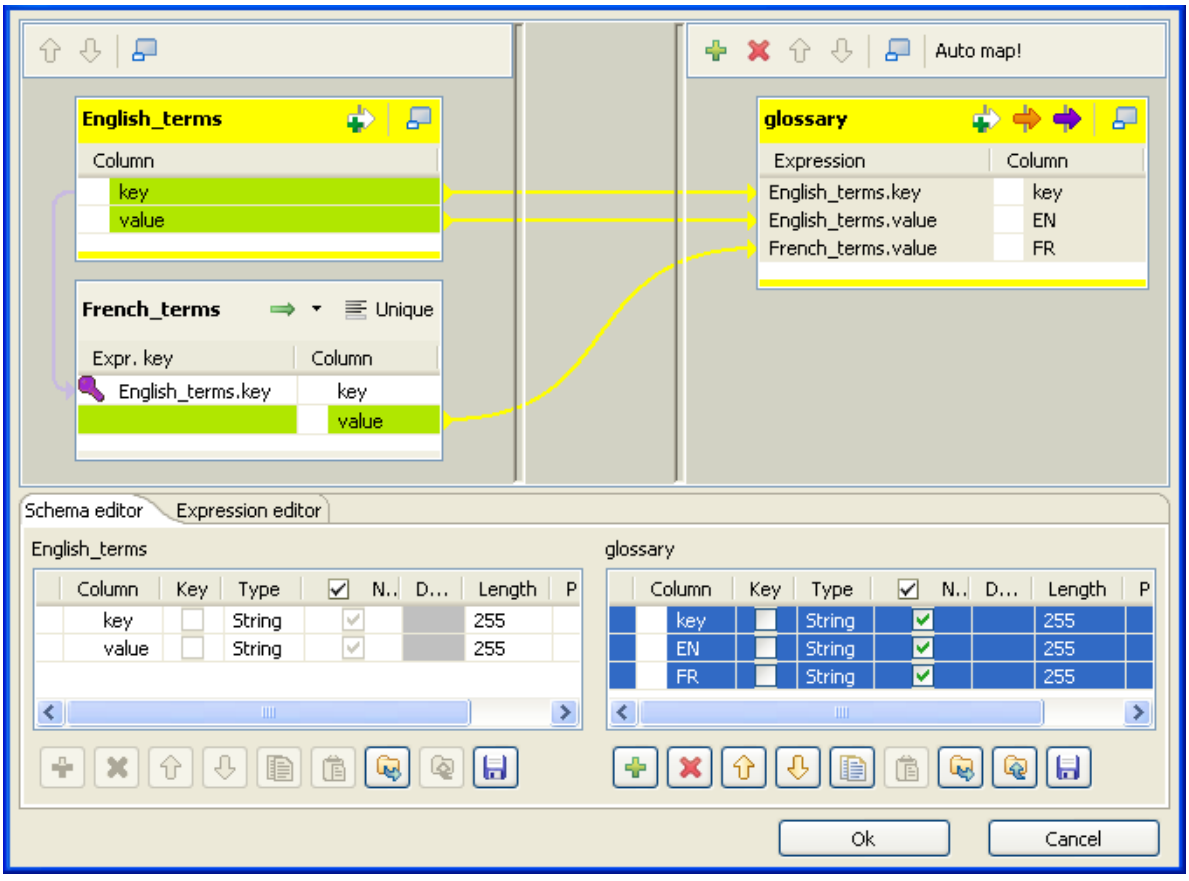
- Double-click the first **tFileInputProperties** component to open its **Basic settings** view and define its properties.



- In the **File Format** field, select your file format.
- In the **File Name** field, click the three-dot button and browse to the input .properties file you want to use.
- Do the same with the second **tFileInputProperties** and browse to the French properties file this time.



- Double-click the **tMap** component to open the tMap editor.



- Select all columns from the *English_terms* table and drop them to the output table.
- Select the *key* column from the *English_terms* table and drop it to the *key* column in the *French_terms* table.
- In the **glossary** table in the lower right corner of the tMap editor, rename the *value* field as *EN* because it will hold the values of the English file.

- Click the plus button to add a line to the **glossary** table and rename it as *FR*.
- In the **Length** field, set the maximum length to *255*.
- In the upper left corner of the tMap editor, select the *value* column in the *English_terms* table and drop it to the *FR* column in the *French_terms* table.
- Click **OK** to validate your changes and close the editor.
- In the design workspace, double-click **tLogRow** to display its **Basic settings** and define the component properties.
- Click **Sync Columns** to retrieve the schema from the preceding component.
- Save your Job and press **F6** to execute it.

```
Starting job tFileInputProperties at 15:25 29/05/2009.

PORT.NAME | Port | Port
HELP | org.talend.help.tMysqlInput | org.talend.help.tMysqlInput
STRING_QUOTE.NAME | String Quote | Séparateur de chaine de
caractère
DBTABLE.NAME | Table Name | Nom de table
QUERYSTORE.NAME | Query Type | Type de requête
SQL_SYNTAX.NAME | Sql Syntax | Syntaxe SQL
TYPE.ITEM.PGSQL | PostgreSQL | PostgreSQL
TYPE.NAME | Database Driver | Pilote de base de données
TABLE.NAME | Table Name | Nom de table
ENCODING.NAME | Encoding | Encodage
QUERY.NAME | Query | Requête
DBD-ODBC.INFO | Required for ODBC-like connection | Requis pour
les connexions de type ODBC
```

The glossary displays on the console listing three columns holding: the key name in the first column, the English term in the second, and the corresponding French term in the third.

tFileInputRegex

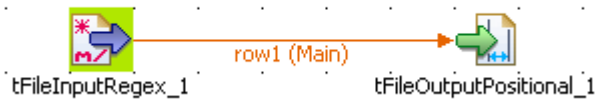
tFileInputRegex properties

Component family	File/Input	
Function	Powerful feature which can replace number of other components of the File family. Requires some advanced knowledge on regular expression syntax	
Purpose	Opens a file and reads it row by row to split them up into fields using regular expressions. Then sends fields as defined in the Schema to the next job component, via a Row link.	
Basic settings	*Property type*	Either Built-in or Repository.
		Built-in: No property data stored centrally.
		Repository: Select the Repository file where Properties are stored. The following fields are pre-filled in using fetched data.
	File Name	Name of the file to be processed. Related topic:*Defining variables from the Component view* of **Talend Open Studio** User Guide.
	Row separator	String (ex: “\n”on Unix) to distinguish rows.
	Regex	This field is Perl or Java compatible and can contain multiple lines. Type in your regular expressions including the subpattern matching the fields to be extracted. **Note**: In Java, antislashes need to be doubled in regexp *Regex syntax is different in Java/Perl and requires doubel/single quotes respectively.*
	Header	Number of rows to be skipped in the beginning of file
	Footer	Number of rows to be skipped at the end of the file.
	Limit	Maximum number of rows to be processed. If Limit = 0, no row is read or processed.
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the Repository.
		Built-in: The schema will be created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused in various projects and job flowcharts. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	Skip empty rows	Select this check box to skip empty rows.

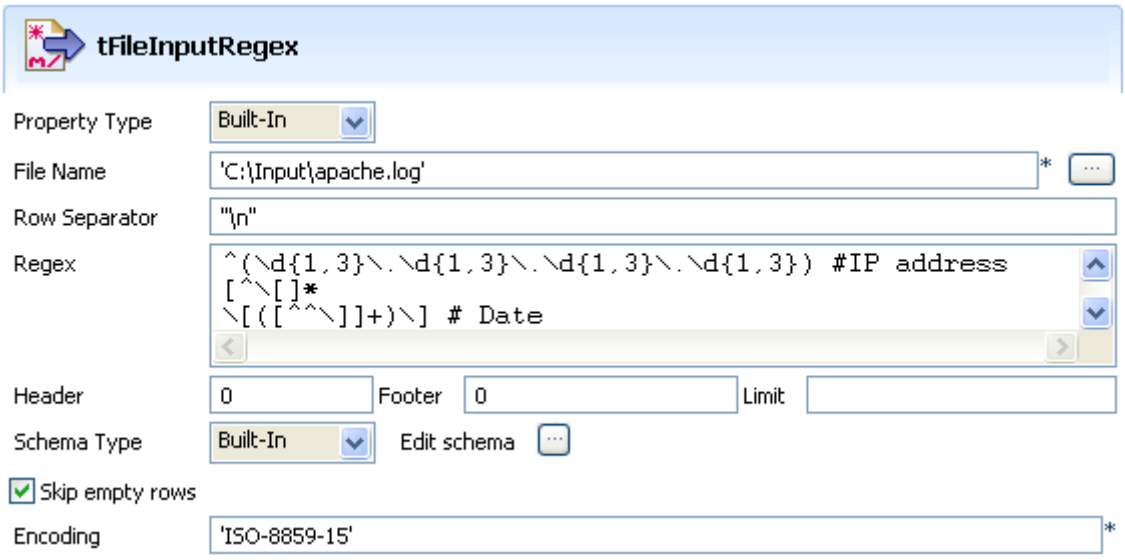
	Encoding	Select the encoding from the list or select Custom and define it manually. This field is compulsory for DB data handling.
Usage	Use this component to read a file and separate fields contained in this file according to the defined Regex.	
Limitation	n/a	

Scenario: Regex to Positional file

The following scenario creates a two-component Job, reading data from an Input file using regular expression and outputting delimited data into an XML file.



- Drop a **tFileInputRegex** component from the **Palette** to the design workspace.
- Drop a **tFileOutputPositional** component the same way.
- Right-click on the **tFileInputRegex** component and select **Row** > **Main**. Drag this main row link onto the **tFileOutputPositional** component and release when the plug symbol displays.
- Select the **tFileInputRegex** again so the **Component** view shows up, and define the properties:

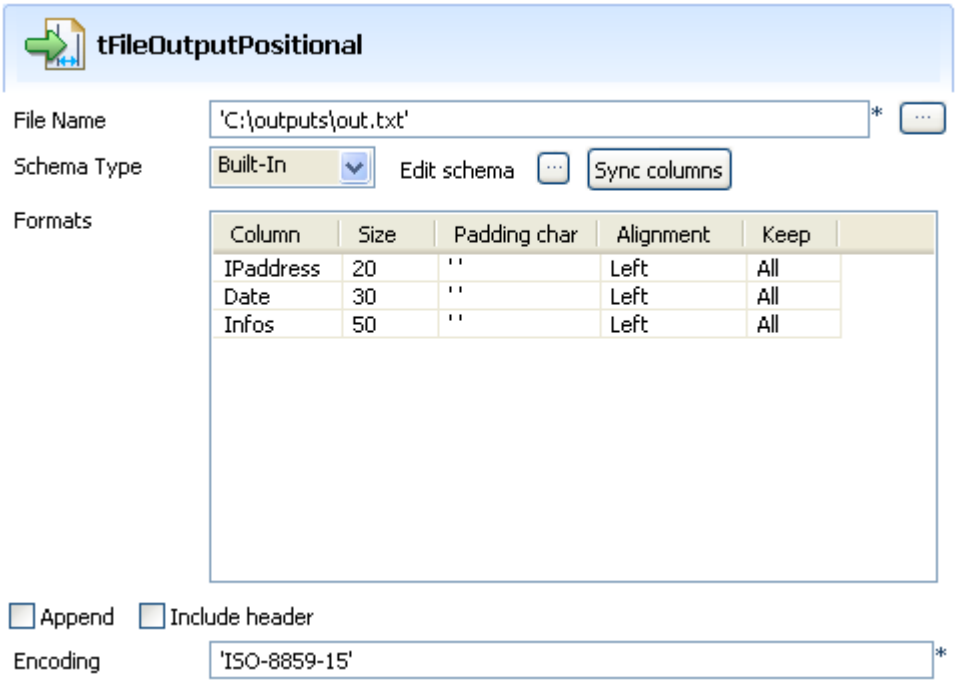


- The Job is built-in for this scenario. Hence, the Properties are set for this station only.
- Fill in a path to the file in **File Name** field. This field is mandatory.
- Define the **Row separator** identifying the end of a row.

- Then define the **Regular expression** in order to delimit fields of a row, which are to be passed on to the next component. You can type in a regular expression using Perl code, and on mutiple lines if needed.

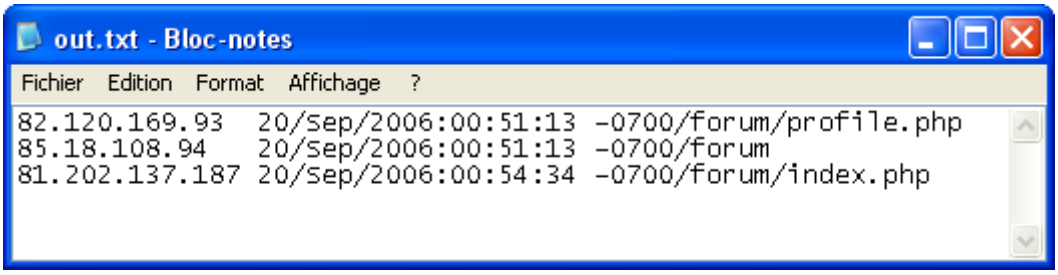
Make sure to use the correct Regex syntax according to the generation language in use as the syntax is different in Java/Perl, and include the regex in double/single quotes respectively.

- In this expression, make sure you include all subpatterns matching the fields to be extracted.
- In this scenario, ignore the header, footer and limit fields.
- Select a local (**Built-in**) **Schema type** to define the data to pass on to the **tFileOutputPositional** component.
- You can load or create the schema through the **Edit Schema** function.
- Then define the second component properties:



- Enter the Positional file output path.
- Enter the **Encoding** standard, the output file is encoded in. Note that, for the time being, the encoding consistency verification is not supported.
- Select the **Schema type**. Click on **Sync columns** to automatically synchronize the schema with the Input file schema.
- Now go to the **Run** tab, and click on **Run** to execute the Job.

The file is read row by row and split up into fields based on the **Regular Expression** definition. You can open it using any standard file editor.


out.txt - Bloc-notes
Fichier Edition Format Affichage ?
82.120.169.93 20/Sep/2006:00:51:13 -0700/forum/profile.php
85.18.108.94 20/Sep/2006:00:51:13 -0700/forum
81.202.137.187 20/Sep/2006:00:54:34 -0700/forum/index.php

tFileInputXML

tFileInputXML belongs to two component families: File and XML. For more information on **tFileInputXML**, see *tFileInputXML on page 825*.

tFileInputMSXML

tFileInputMSXML belongs to two component families: File and MultiSchema. For more information on **tFileInputMSXML**, see *tFileInputMSXML on page 673*.

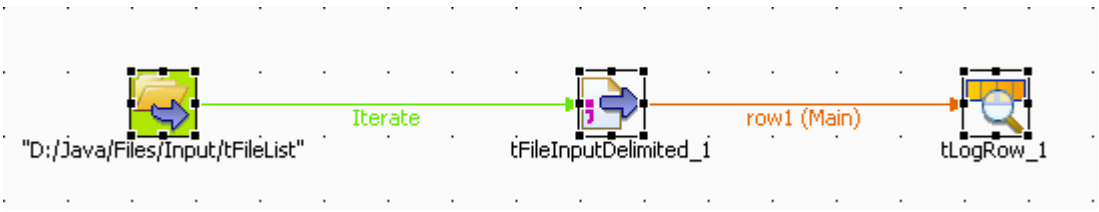
tFileList

tFileList properties

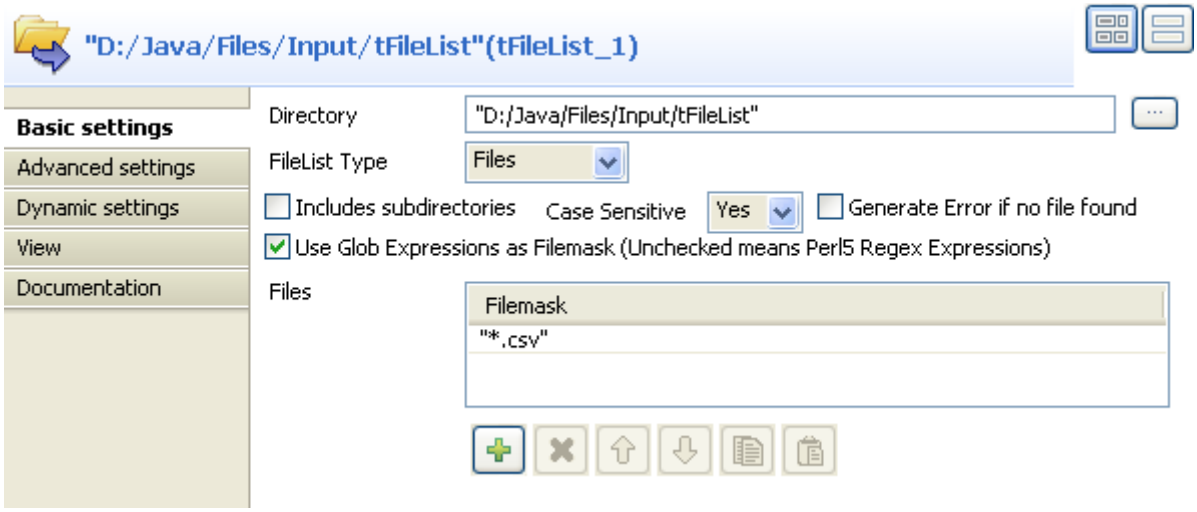
Component family	File/Management	
Function	**tFileList** iterates on files or folders of a set directory.	
Purpose	**tFileList** retrieves a set of files or folders based on a filemask pattern and iterates on each unity.	
Basic settings	*Directory*	Path to the directory where files are stored
	FileList Type	Select in the list the type of the input you want to iterate on: **Files** if the input is a set of files, **Directories** if the input is a set of directories, **Both** if the input is a set of the above two types.
	Include subdirectories	Select this check box if the selected input source type include sub-directories.
	Case Sensitive	Set the case mode from the list to either create or not create case sensitive filter on filenames.
	Generate Error if no file found	Select this check box to generate an error message if no files or directories are found.
	Use Glob Expressions as Filemask (Unchecked means Perl5 Regex Expressions)	This check box is selected by default. It allows to filter the results using a Global Expression (**Glob Expressions**). Clear this check box to filter results using a Regex Expression of the type **Perl5**.
	Files	Click the plus button to add as many filter lines as needed: **Filemask**: in the added filter lines, type in a filename or a filemask using special characters or regular expressions.
Usage	**tFilelist** provides a list of files or folders from a defined directory on which it iterates	

Scenario: Iterating on a file directory

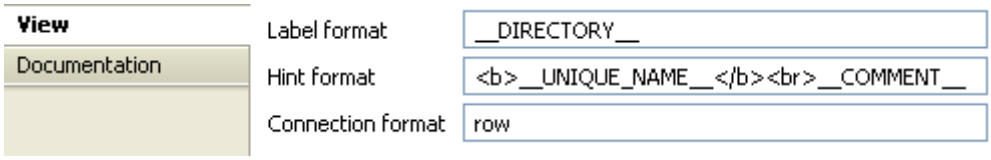
The following scenario creates a three-component Job, which aims at listing files from a defined directory, reading each file by iteration, selecting delimited data and displaying the output in the **Run** log console.



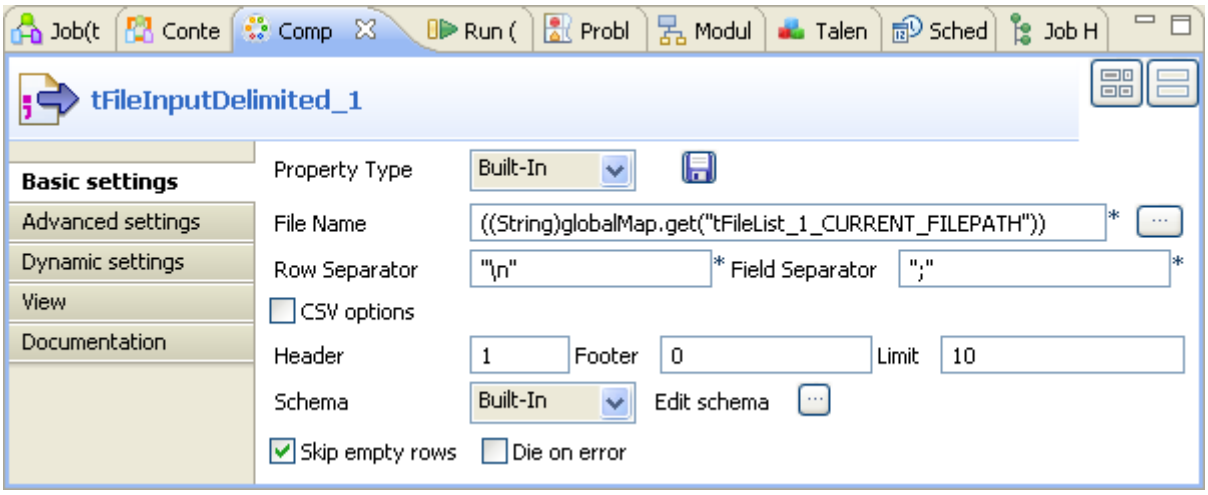
- Drop the following components from the **Palette** to the design workspace: **tFileList**, **tFileInputDelimited**, and **tLogRow**.
- Right-click on the **tFileList** component, and pull an **Iterate** connection to the **tFileInputDelimited** component. Then pull a **Main** row from the **tFileInputDelimited** to the **tLogRow** component.
- Double-click **tFileList** to display its **Basic settings** view and define its properties.



- Browse to the **Directory** that holds the files you want to process. To display the path on the Job itself, use the label (__DIRECTORY__) that shows up when you put the pointer anywhere in the **Directory** field. Type in this lable in the **Label Format** field you can find if you click the **View** tab in the **Basic settings** view.



- In the **Basic settings** view and from the **FileList Type** list, select the source type you want to process, **Files** in this example.
- In the **Case sensitive** list, select a case mode, **Yes** in this example to create case sensitive filter on file names.
- Keep the **Use Glob Expressions as Filemask** check box selected if you want to use global expressions to filter files.
- In the **Filemask** field, define a file mask, use special characters if need be.
- Double-click **tFileInputDelimited** to display its **Basic settings** view and set its properties.



- Enter the **File Name** field using a variable containing the current filename path, as you filled in the **Basic settings** of **tFileList**. Press **Ctrl+Space bar** to access the autocomplete list of variables.
- Select the global variable `((String)globalMap.get("tFileList_1_CURRENT_FILEPATH"))` if you work in Java and, or `$_globals{tFileList_1}{CURRENT_FILEPATH}` if you work in Perl. This way, all files in the input directory can be processed.
- Fill in all other fields as detailed in the **tFileInputDelimited** section. Related topic: *tFileInputDelimited properties on page 494*.
- Select the last component, **tLogRow**, to display its **Basic settings** view and fill in the separator to be used to distinguish field content displayed on the console. Related topic: *tLogRow on page 628*.

```
Starting job test at 17:16 21/09/2009.

+--+-----------------------------+-----------------------+-------+
|                              tLogRow_1                           |
|=-+-----------------------------+-----------------------+------=|
|id|CustomerName                 |CustomerAddress        |idState|
|=-+-----------------------------+-----------------------+------=|
|1 |Griffith Paving and Sealcoatin|talend@apres91         |7      |
|2 |Bill's Dive Shop             |511 Maple Ave.  Apt. 1B|35     |
|3 |Childress Child Day Care     |662 Lyons Circle       |1      |
|4 |Facelift Kitchen and Bath    |220 Vine Ave.          |41     |
|5 |Terrinni & Son Auto and Truck|770 Exmoor Rd.         |5      |
|6 |Kermit the Pet Shop          |1860 Parkside Ln.      |28     |
|7 |Tub's Furniture Store        |807 Old Trail Rd.      |15     |
|8 |Toggle & Myerson Ltd         |618 Sheriden rd.       |9      |
|9 |Childress Child Day Care     |788 Tennyson Ave.      |12     |
|10|Elle Hypnosis and Therapy Cent|2032 Northbrook Ct.    |1      |
+--+-----------------------------+-----------------------+-------+
+--+-----------------------------+-----------------------+-------+
|                              tLogRow_1                           |
|=-+-----------------------------+-----------------------+------=|
|id|CustomerName                 |CustomerAddress        |idState|
|=-+-----------------------------+-----------------------+------=|
|1 |Glenwood Credit Union        |511 Maple Ave.  Apt. 1B|46     |
|2 |Gourmet the Frog             |788 Tennyson Ave.      |1      |
|3 |Acturial Enterprises Ltd.    |3385 University Ave.   |34     |
|4 |Salt & Pepper Catering Service|965 Marion Place Apt. 65C|44     |
|5 |Rythmics Ltd.                |1875 Roger Williams Ave.|22     |
|6 |Parkway Auto Body            |1859 Green Bay Rd.     |27     |
|7 |Futoons Cartoons Emporium    |1486 Old Deerfield Rd. |24     |
|8 |Glenn Oaks Office Supplies   |1882 St. Johns         |10     |
|9 |Garfield Appliance Service   |3150 Skokie Valley Rd. |33     |
|10|New Dehli Auto Exchange      |1957 Huntington Ave.   |49     |
+--+-----------------------------+-----------------------+-------+
Job test ended at 17:16 21/09/2009. [exit code=0]
```

The Job iterates on the defined directory, and reads all included files. Then delimited data is passed on to the last component which displays it on the console.

For other scenarios using **tFileList**, see *tFileCopy on page 485*.

tFileOutputEBCDIC

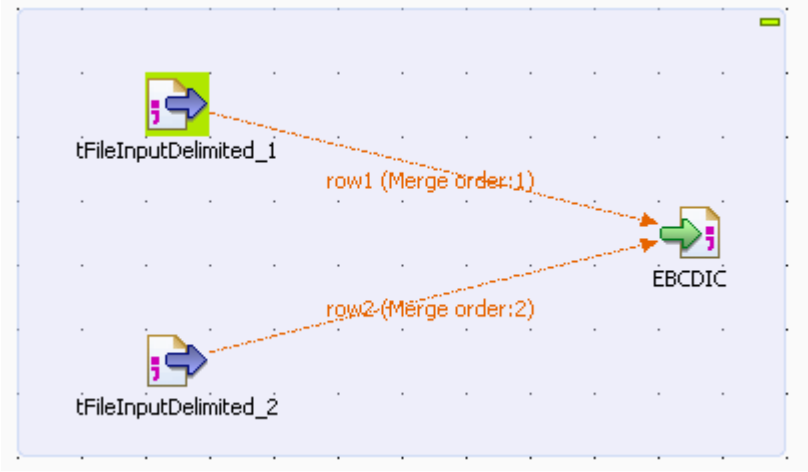
tFileOutputEBCDIC properties

Component family	File/Output	
Function	The **tFileOutputEBCDIC** writes ab EBCDIC file based on various source data files, each of them with a different schema.	
Purpose	This component writes an EBCDIC file with data extracted from files based on their schemas.	
Basic settings	*Property type*	Either **Built-in** or **Repository**.
		Built-in: No property data stored centrally.
		Repository: Select the repository file where Properties are stored. The fields that come after are pre-filled in using the fetched data.
	Data file	Select the EBCDIC file containing the data to process.
	Xc2j file	Select the xc2j transformation file.
Usage	Use this component to write an EBCDIC file and to output the data separately depending on the schemas identified in the incoming file.	

Scenario: Creating an EBCDIC file using two delimited files

This scenario uses the **[Copybook Connection]** wizard that guides users through the different steps to create a Copybook connection and to retrieve the EBCDIC schemas. This wizard is available only for **Talend Integration Suite** users. If you are using **Talend Open Studio** or **Talend On Demand**, you need to set the basic settings for the **tFileInputEBCDIC** component manually.

The following scenario is a three-component Job that aims at writing an EBCDIC-format file using two delimited files with different schemas.



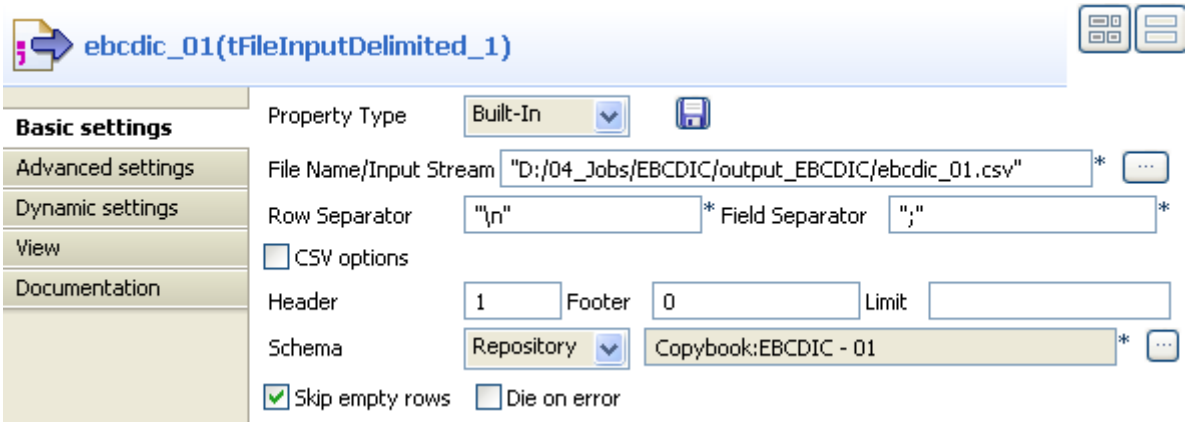
This Java scenario uses the EBCDIC Connection wizard to set up a connection to the Copybook file and to generate an xc2j file, which allows the retrieval and transformation of the different file schemas.

- Create a connection to the Copybook file, which describes the structure of your EBCDIC file. In this scenario, the Copybook connection is called EBCDIC.
- Retrieve the file schemas.

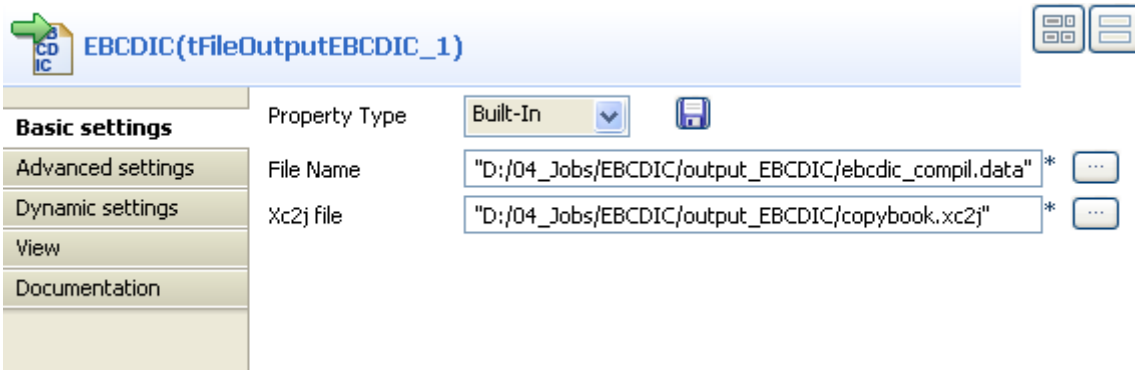
Once the Copybook connection has been created and the schemas retrieved, using the EBCDIC and Schema wizards, the new schemas appear under the node **Metadata** > **Copybook**. They are called *01*, *04* and *05*.

To create an EBCDIC file based on two delimited files in **Talend Open Studio** :

- Drop the following components from the Palette to the design workspace: **tFileInputDelimited** (x2) and **tFileOutputEBCDIC**.
- To connect them together, right-click on each **tFileInputDelimited** component, select **Row** > **Main** in the contextual menu and click on the **tFileOutputEBCDIC** component.
- Double-click on the first **tFileInputDelimited** component to display the **Basic settings** view and set the component properties.



- In the **File Name** field, browse to the delimited file via the three-dot button **[...]**.
- In the **Schema** field, select **Repository**, then click the three-dot button and, when prompted, select the schema corresponding to your file, under the **Copybook** node.
- In the **Header** field, set the number of fields that are used as “headers”, *1* in this example.
- Set the properties for the second **tFileInputDelimited** component the same way as for the first component.
- Double-click the **tFileOutputEBCDIC** component to display the **Basic settings** view and set the component properties:



- In the **Data file** field, enter or browse to the directory path and the EBCDIC file name that is to be created based on both delimited files.
- In the **Xc2j file** field, enter or browse to the path to the file allowing to extract the schema that describes the EBCDIC structure file.
- Save your Job via **Ctrl+S** and click on the **Run** view, select the **Statistics** and **Exec time** check boxes then click **Run** to execute the Job.

tFileOutputExcel

tFileOutputExcel Properties

Component family	File/Output	
Function	**tFileOutputExcel** outputs data to an MS Excel type of file.	
Purpose	**tFileOutputExcel** writes an MS Excel file with separated data value according to a defined schema.	
Basic settings	*File name*	Name or path to the output file. Related topic: *Defining variables from the Component view* of **Talend Open Studio** User Guide
	Sheet name	Name of the xsl sheet.
	Include header	Select this check box to include a header row to the output file.
	Append existing file	Select this check box to add the new lines at the end of the file. **Append existing sheet**: Select this check box to add the new lines at the end of the Excel sheet.
	Is absolute Y pos.	Select this check box to add information in specified cells: **First cell X**: cell position on the X-axis (X-coordinate or Abcissa). **First cell Y**: cell position on the Y-axis (Y-coordinate). **Keep existing cell format**: select this check box to retain the original layout and format of the cell you want to write into.
	Font	Select in the list the font you want to use.
	Define all columns auto size	Select this check box if you want the size of all your columns to be defined automatically. Otherwise, select the **Auto size** check boxes next to the column names you want their size to be defined automatically.
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the Repository.
		Built-in: The schema will be created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused in various projects and job designs. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.

Advanced settings	*Create directory if not exists*	This check box is selected by default. This option creates the directory that will hold the output files if it does not already exist.
	Advanced separator (for number)	Select this check box to modify the separators you want to use for numbers: **Thousands separator**: define separators for thousands. **Decimal separator**: define separators for decimals.
	Encoding Type	Select the encoding type from the list or select **Custom** and define it manually. This field is compulsory for DB data handling.
	tStatCatcher Statistics	Select this check box to gather the job processing metadata at a job level as well as at each component level.
Usage	Use this component to write an XML file with data passed on from other components using a **Row** link.	
Limitation	n/a	

Related scenario

For **tFileOutputExcel** related scenario, see *tSugarCRMInput on page 83*.

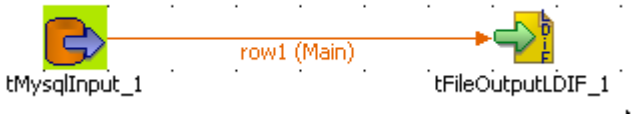
tFileOutputLDIF

tFileOutputLDIF Properties

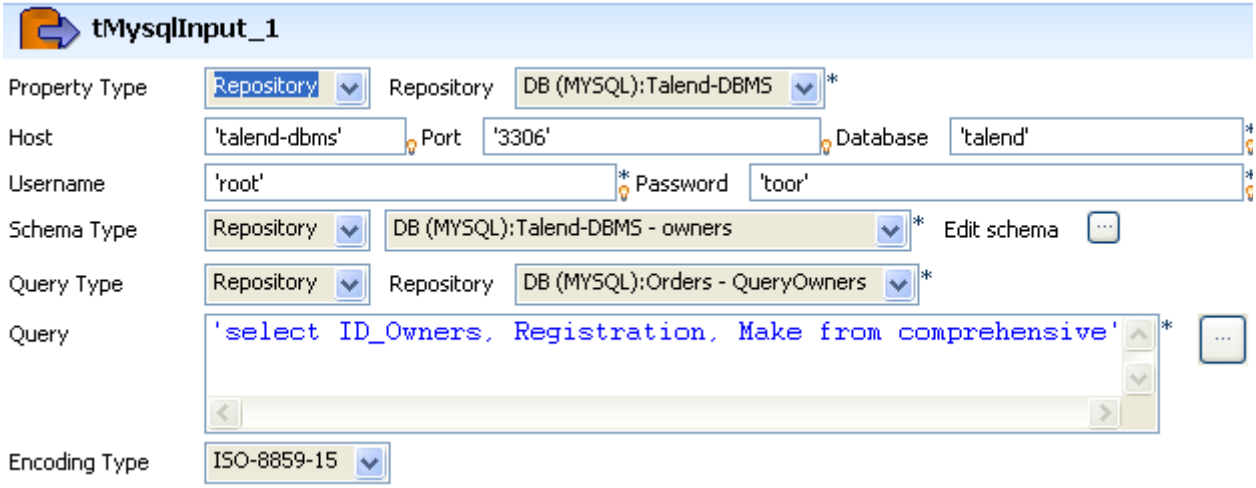
Component family	File/Output	
Function	**tFileOutputLDIF** outputs data to an LDIF type of file which can then be loaded into a LDAP directory.	
Purpose	**tFileOutputLDIF** writes or modifies a LDIF file with data separated in respective entries based on the schema defined,.or else deletes content from an LDIF file.	
Basic settings	*File name*	Name or path to the output file. Related topic: *Defining variables from the Component view* of **Talend Open Studio** User Guide.
	Wrap	Wraps the file content, every defined number of characters.
	Change type	Select **Add**, **Modify** or **Delete** to respectively create an LDIF file, modify or remove an existing LDIF file. In case of modification, set the type of attribute changes to be made.
	Change on attributes	Select **Add**, **Modify** or **Delete** to respectively add a new attribute to the file, replace the attributes with new ones or suppress attributes from the file defined.
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the Repository.
		Built-in: The schema will be created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused in various projects and job designs. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	Sync columns	Click to synchronize the output file schema with the input file schema. The Sync function only displays once the Row connection is linked with the Output component.
	Encoding	Select the encoding from the list or select Custom and define it manually. This field is compulsory for DB data handling.
Usage	Use this component to write an XML file with data passed on from other components using a Row link.	
Limitation	n/a	

Scenario: Writing DB data into an LDIF-type file

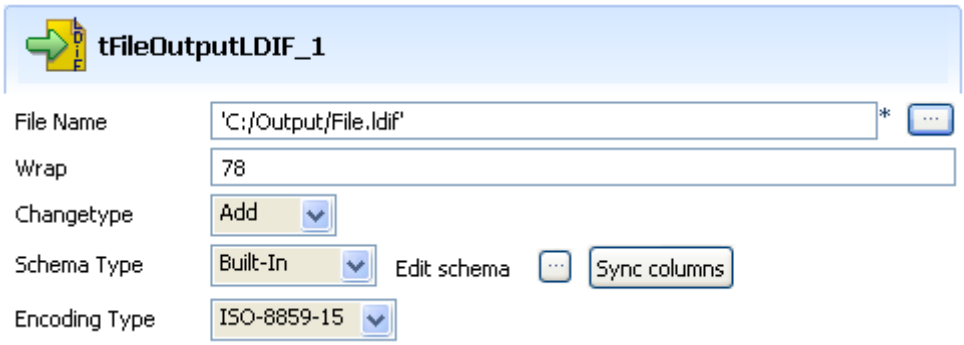
This scenario describes a two component Job which aims at extracting data from a database table and writing this data into a new output LDIF file.



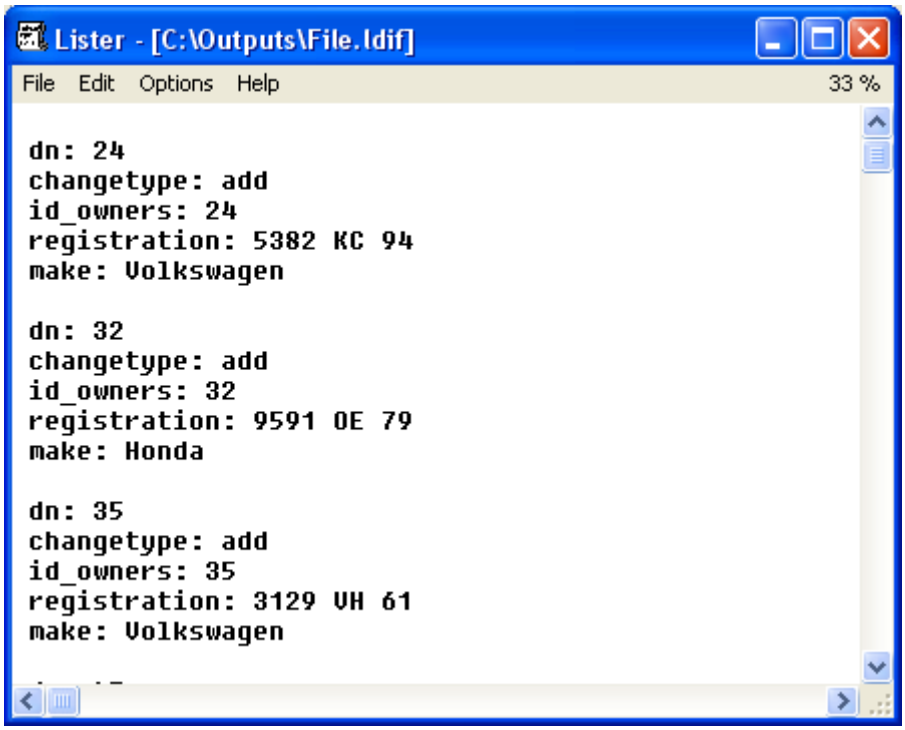
- Drop a **tDBInput** and a **tFileOutputLDIF** component from the **Palette** to the design area. Bind them together using a **Row > Main** link.
- Select the **tDBInput** component, and go to the **Component** panel then select the **Basic settings** tab.
- If you stored the DB connection details in a **Metadata** entry in the Repository, set the **Property type** as well as the **Schema type** on **Repository** and select the relevant metadata entry. All other fields are filled in automatically, and retrieve the metadata-stored parameters.



- Alternatively select **Built-in** as **Property type** and **Schema type** and fill in the DB connection and schema fields manually.
- Then double-click on **tFileOutpuLDIF** and define the **Basic settings**.
- Browse to the folder where you store the Output file. In this use case, a new LDIF file is to be created. Thus type in the name of this new file.
- In the **Wrap** field, enter the number of characters held on one line. The text coming afterwards will get wrapped onto the next line.



- Select **Add** as **Change Type** as the newly created file is by definition empty. In case of modification type of Change, you'll need to define the nature of the modification you want to make to the file.
- As **Schema Type**, select **Built-in** and use the **Sync Columns** button to retrieve the input schema definition.
- Press F6 to short run the Job.



The LDIF file created contains the data from the DB table and the type of change made to the file, in this use case, addition.

tFileOutputProperties

tFileOutputProperties properties

Component family	File/Output	
Function	**tFileInputProperties** writes a configuration file of the type **.ini** or **.properties**.	
Purpose	**tFileInputProperties** writes a configuration file containing text data organized according to the model key = value.	
Basic settings	*Schema type* and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remotely stored in the Repository but for this component, the schema is read-only. It is made of two column, *Key* and *Value*, corresponding to the parameter name and the parameter value to be copied.
	File format	Select from the list file format: either **.properties** or **.ini**.
		.properties: data in the configuration file is written in two lines and structured according to the following way: key = value.
		.ini: data in the configuration file is written in two lines and structured according to the following way: key = value and re-grouped in sections. **Section Name**: enter the section name on which the iteration is based.
	File Name	Name or path to the file to be processed. Related topic: *Defining variables from the Component view* of **Talend Open Studio User Guide**.
Usage	Use this component to write files where data is organized according to the structure key = value.	

Related scenarios

For a related scenario, see *Scenario: Reading and matching the keys and the values of different .properties files and outputting the results in a glossary on page 517* of the ***tFileInputProperties*** component.

tFileOutputXML

tFileOtputXML belongs to two component families: File and XML. For more information on **tFileOutputXML,** see *tFileOutputXML on page 828*.

tFileProperties

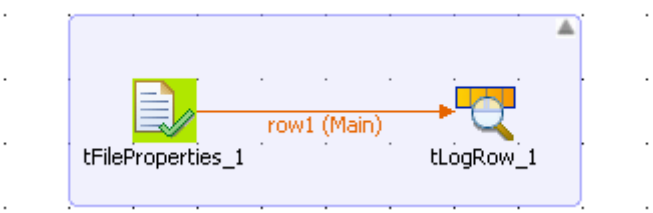
tFileProperties Properties

Component family	File/Management	
Function	**tFileProperties** creates a single row flow that displays the properties of the processed file.	
Purpose	**tFileProperties** allows to have information about the main properties of a defined file.	
Basic settings	*Schema type*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the Repository.
		Built-in: You create and store the schema locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: You have already created the schema and stored it in the Repository. You can reuse it in various projects and job designs. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	Edit schema	The number of the read-only lines is different between Java and Perl.
	File	Name or path to the file to be processed. Related topic: *Defining variables from the Component view* of **Talend Open Studio** User Guide.
Usage	This component can be used as standalone component.	
Limitation	n/a	

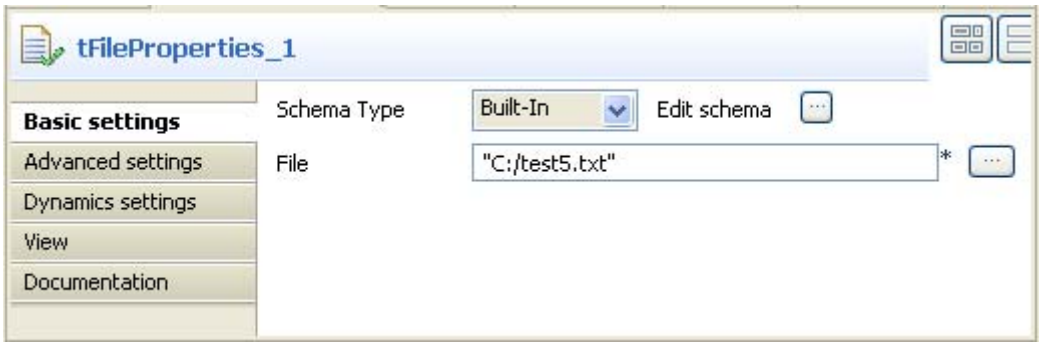
Scenario: Displaying the properties of a processed file

This Java scenario describes a very simple Job that displays the properties of the specified file.

- Drop a **tFileProperties** component and a **tLogRow** component from the **Palette** onto the design workspace.
- Right-click on **tFileProperties** and connect it to **tLogRow** using a **Main Row** link.



- In the design workspace, select **tFileProperties**.
- Click the **Component** tab to define the basic settings of **tFileProperties**.



- Set **Schema Type** to **Built in.**.
- If desired, click the **Edit schema** button to see the read-only columns.
- In the **File** field, enter the file path or browse to the file you want to display the properties for.
- In the design workspace, select **tLogRow** and click the **Component** tab to define its basic settings. For more information, see *tLogRow on page 628*.
- Press **F6** to execute the Job.

```
Starting job File_Properties at 14:57 26/08/2008.
.-------------------------------------------------.
|                 #1. tLogRow_1                   |
+--------------+----------------------------------+
| key          | value                            |
+--------------+----------------------------------+
| abs_path     | C:\test5.txt                     |
| dirname      | C:\                              |
| basename     | test5.txt                        |
| mode_string  | rw                               |
| size         | 86                               |
| mtime        | 1219674736421                    |
| mtime_string | Mon Aug 25 16:32:16 CEST 2008    |
+--------------+----------------------------------+

Job File_Properties ended at 14:57 26/08/2008. [exit code=0]
```

The properties of the defined file are displayed on the console.

tFileUnarchive

tFileUnarchive Properties

Component family	File/Management	
Function	Decompresses the archive file provided as parameter and put it in the extraction directory.	
Purpose	Unarchives a file of any format (zip, rar...) that is mostlikely to be processed.	
Basic settings	*Archive file*	File path to the archive
	Extract Directory	Folder where the unarchived file is put
 *Java only*	*Use archive name as root directory / Extract file paths*	Select this check box to reproduce the whole path to the file or if none exists create a new folder
 *Perl only*	*Use Command line tools*	Select this check box to use another unarchiving tool than the one provided by default in the Perl package.
Usage	This component can be used as a standalone component but it can also be used within a Job as a Start component using an **Iterate** link.	
Limitation	n/a	

Related scenario

For **tFileUnarchive** related scenario, see *tFileCompare on page 482*.

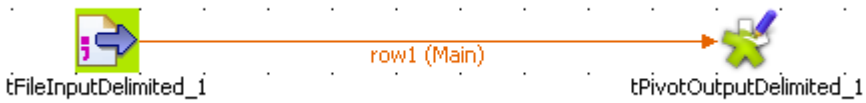
tPivotOutputDelimited

tPivotOutputDelimited Properties

Component family	File/Output	
Function	**tPivotOutputDelimited** outputs data based on an aggregation operation carried out on a pivot column.	
Purpose	**tPivotOutputDelimited** allows to fine-tune the selection of data to output	
Basic settings	*Pivot column*	Select the column from the incoming flow that will be used as pivot for the aggregation operation.
	Aggregation column	Select the column from the incoming flow that contains the data to be aggregated.
	Aggregation function	Select the function to be used in case several values are available for the pivot column.
	Group by	Define the aggregation sets, the values of which will be used for calculations.
		Input Column: Match the input column label with your output columns, in case the output label of the aggregation set needs to be different.
	File Name	Name or path to the output file. Related topic: *Defining variables from the Component view* of **Talend Open Studio** User Guide.
	Field separator	Character, string or regular expression to separate fields of the output file.
	Row separator	String (ex: “\n”on Unix) to distinguish rows in the output file.
Usage	This component requires an input flow.	
Limitation	n/a	

Scenario: Using a pivot column to aggregate data

The following scenario describes a Job that aggregates data from a delimited input file, using a defined pivot column.

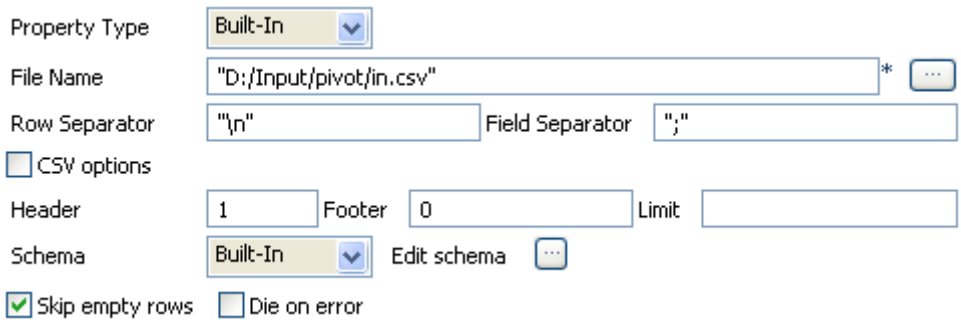


- Drop the following component from the **Palette** to the design workspace: **tFileInputDelimited**, **tPivotOutputDelimited**.

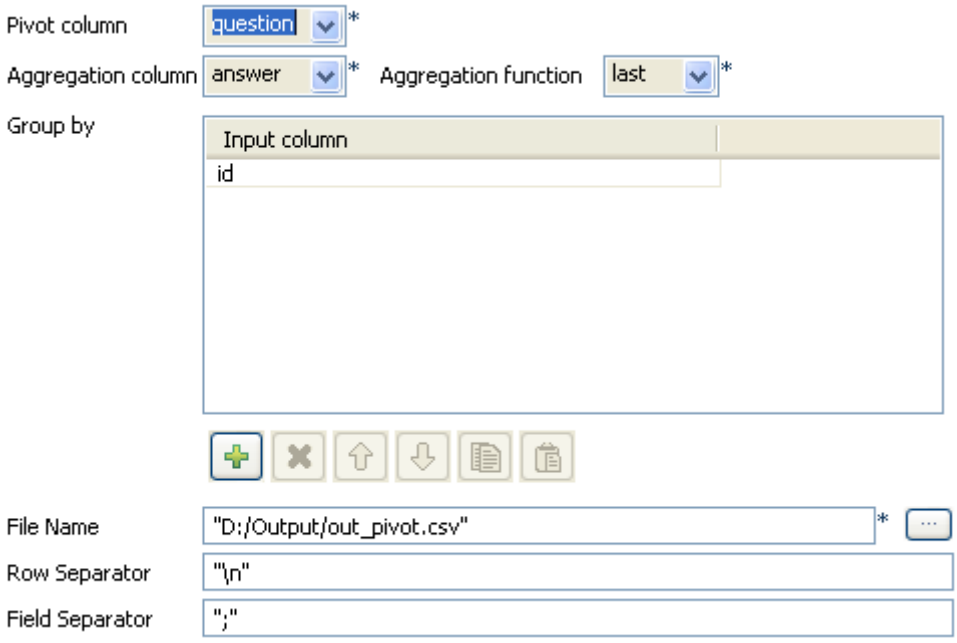
- The file to use as input file is made of 3 columns, including: *ID*, *Question* and the corresponding *Answer*

Id	Question	Answer
1	Name	Juan
2	Name	Jean
3	Name	John
1	Gender	M
2	Gender	F
3	Gender	M
1	Surgery	Yes
2	Surgery	No
3	Surgery	Yes
1	Age	45
2	Age	23
3	Age	42
1	Name	Mary

- On your design workspace, select the **tFileInputDelimited** component
- Define the basic settings, on the **Component** view.



- Browse to the input file to fill out the **File Name** field.
- Define the **Row** and **Field** separators, in this example, respectively: carriage return and semi-colon
- As the file contains a header line, define it also.
- Set the schema describing the three columns: *ID*, *Questions*, *Answers*.
- Then select the **tPivotOutputDelimited** and set its properties on the **Basic Settings** tab of the **Component** view.



- In the **Pivot column** field, select the pivot column from the input schema. this is often the column presenting most duplicates (pivot aggregation values).
- In the **Aggregation column** field, select the column from the input schema that should gets aggregated.
- In the **Aggregation function** field, select the function to be used in case duplicates are found out.
- In the **Group by** table, add an Input column, that will be used to group by the aggregation column.
- In the **File Name** field, browse to the output file path. And on the **Row** and **Field separator** fields, set the separators for the aggregated output rows and data.

Then, press **F6** to execute the Job. The output file shows the newly aggregated data.

id	Name	Gender	Surgery	Age
1	Mary	M	Yes	45
2	Jean	M	No	23
3	John	F	Yes	42

Internet components

This chapter details the major components that you can find in **Internet** group of the **Palette** of **Talend Open Studio**.

The Internet family groups all components that help you access content stored on the Internet, through various means including Web services, RSS flows, SCP, MOM, Emails, FTP and so on.

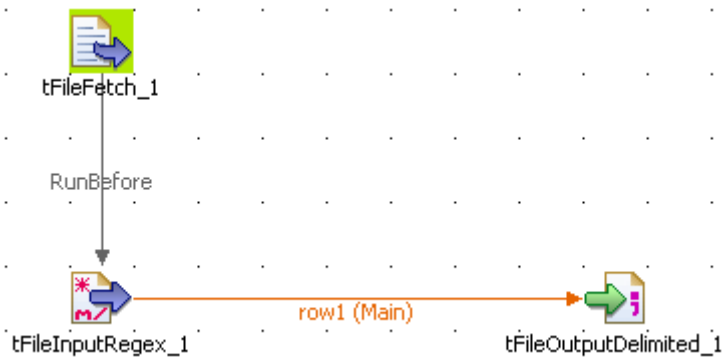
tFileFetch

tFileFetch properties

Component family	Internet	
Function	**tFileFetch** retrieves a file from HTTP	
Purpose	**tFileFetch** allows to fetch data contained in a file through HTTP protocol.	
Basic settings	*URI*	Type in the URI of the HTTP site where the file is to be fetched from.
	Destination Directory	Browse to the destination folder where the file fetched will be placed.
	Destination Filename	Type in a new name for the file fetched, if need be.
Usage	This component is generally used as a start component to feed the input flow of a Job and is often connected to the Job through a **ThenRun** link.	
Limitation	n/a	

Scenario: Fetching data through HTTP

This scenario describes a three-component Job which retrieves data from an HTTP website and select data that will be stored into a delimited file.



- Drop a **tFileFetch**, a **tFileInputRegex** and a **tFileOutputDelimited** onto your design workspace.
- In the **tFileFetch Basic settings** panel, type in the URI where the file to be fetched can retrieved from.
- In the **Destination directory** field, browse to the folder where the fetched file is to be stored.
- In the **Filename** field, type in a new name for the file if you want it to be changed. In this example, *filefetch.txt*.

- Select the **tFileInputRegex**, set the **File name** so that it corresponds to the file fetched earlier.
- Using a regular expression, in the **Regex** field, select the relevant data from the fetched file. In this example: <td(?: class="leftalign")?> \s* (t\w+) \s* </td>

Make sure to use the correct Regex syntax according to the generation language in use as the syntax is different in Java/Perl, and include the Regex in double/single quotes accordingly.

- Define the **header**, **footer** and **limit** if need be. In this case, we'll ignore these fields.
- Define also the schema describing the flow to be passed on to the final output.
- The schema should be automatically propagated to the final output, but to be sure, check the schema in the **Basic settings** panel of the tFileOutputDelimited component.
- Then press **F6** to run the Job.

tFTPConnection

tFTPConnection properties

Component family	Internet/FTP	
Function	**tFTPConnection** opens an FTP connection for the current transaction.	
Purpose	**tFTPConnection** allows to open an FTP connection to transfer files in one transaction.	
Basic settings	*Host*	IP address of the FTP server.
	Port	Number of listening port of the FTP server.
	Username and Password	FTP user authentication data.
	Connect mode	Select the mode to use: **Active** or **Passive**
Usage	This component is typically used as a single-component sub-job. It is used along with other FTP components.	
Limitation	n/a	

Related scenarios

For a related scenario, see *Scenario: Putting files on a remote FTP server on page 557*.

For a related scenario, see *Scenario: Iterating on a remote directory on page 553*.

For a related scenario using a different protocol, see *Scenario: Getting files from a remote SCP server on page 582*.

tFTPDelete

tFTPDelete properties

Component family	Internet/FTP	
Function	This component deletes defined files via an FTP connection.	
Purpose	**tFTPDelete** deletes files on a remote FTP server.	
Basic settings	*Host*	FTP IP address
	Port	Listening port number of the FTP site.
	Username and *Password*	FTP user authentication data.
	Remote directory	Source directory where the files to be deleted are located.
	Files	File name or path to the files to be deleted.
Usage	This component is typically used as a single-component sub-job but can also be used as an output or end object.	
Limitation	n/a	

Related scenario

For **tFTPDelete** related scenario, see *Scenario: Putting files on a remote FTP server on page 557*.

For **tFTPDelete** related scenario using a different protocol, see *Scenario: Getting files from a remote SCP server on page 582*.

tFTPFileList

tFTPFileList properties

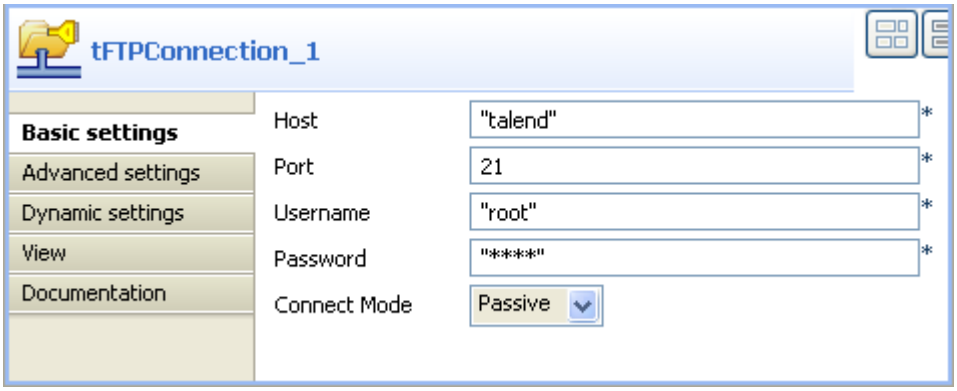
Component family	Internet/FTP	
Function	**tFTPFileList** iterates on files and/or folders of a given directory on a remote host.	
Objective	**tFTPFileList** retrieves files and /or folders based on a defined filemask pattern and iterates on each of them by connecting to a remote directory via an FTP protocol.	
Basic settings	*Use an existing connection/Component List*	Select this check box and in the **Component List** click the relevant connection component to reuse the connection details you already defined.
	Host	FTP IP address.
	Port	Listening port number of the FTP server.
	Username and *Password* (or *Private key*)	User authentication information.
	Remote directory	Path to the remote directory.
	SFTPSupport/ Authentication method	Select this check box and then in the **Authentication method** list, select the SFTP authentication method: **Password**: Type in the password required in the relevant field. **Public key**: Type in the private key or click the three dot button next to the **Private key** field to browse to it.  If you select **Public Key** as the SFTP authentication method, make sure that the key is added to the agent or that no passphrase (secret phrase) is required.
	Files	Click the plus button to add the lines you want to use as filters: **Filemask**: enter the filename or filemask using wildcharacters (*) or regular expressions.
	Connect Mode	Select the SFTP connection mode you want to use: **Active**: You determine the connection port to use to allow data transfer. **Passive**: the FTP server determines the connection port to use to allow data transfer.
Usage	This component is typically used as a single-component sub-job but can also be used with other components.	

Scenario: Iterating on a remote directory

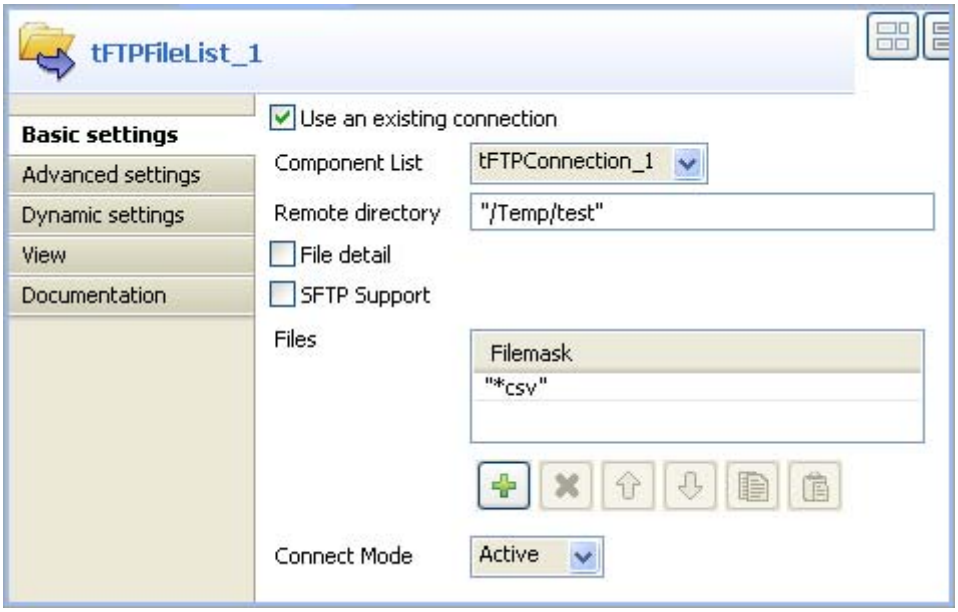
The following Java scenario describes a three-component Job that connects to an FTP server, lists files held in a remote directory based on a filemask and finally recuperates and saves the files in a defined local directory.

- Drop the following components from the Palette to the design workspace: **tFTPConnection**, **tFTPFileList** and **tFTPGet**.

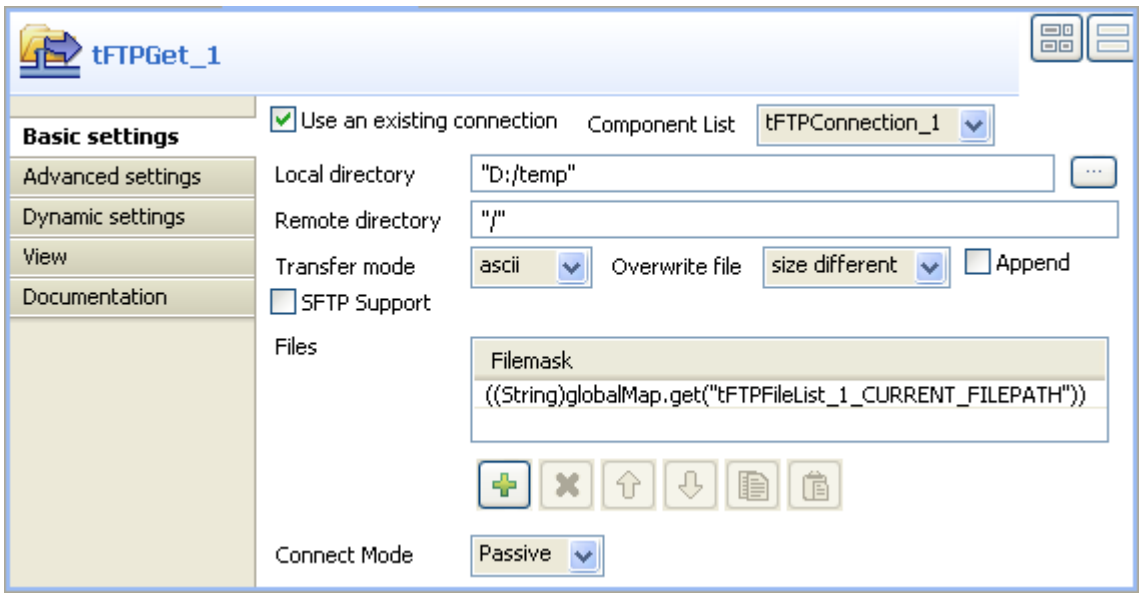
- Link **tFTPConnection** to **tFTPFileList** using an **OnSubjobOk** connection and then **tFTPFileList** to **tFTPGet** using an **Iterate** connection.
- Double-click **tFTPConnection** to display its **Basic settings** view and define the component properties.



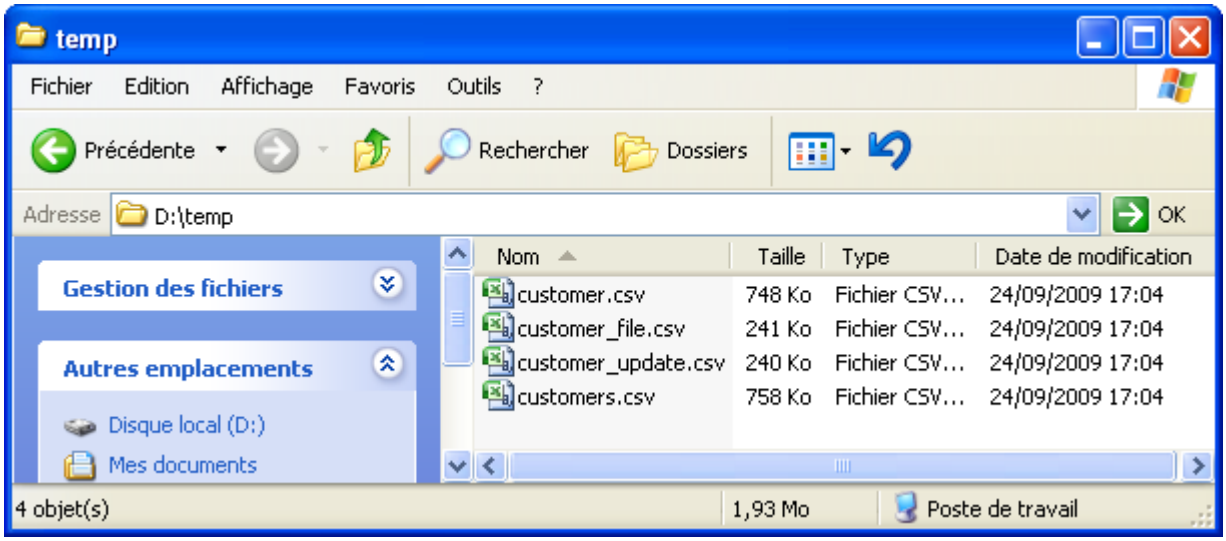
- In the **Host** field, enter the IP address of the FTP server.
- In the **Port** field, enter the listening port number.
- In the **Username** and **Password** fields, enter your authentication information for the FTP server.
- In the **Connect Mode** list, select the FTP connection mode you want to use, **Passive** in this example.
- Double-click **tFTPFileList** to open its **Basic settings** view and define the component properties.



- Select the **Use an existing connection** check box and in the **Component list**, click the relevant FTP connection component, **tFTPConnection_1** in this scenario. Connection information are automatically filled in.
- In the **Remote directory** field, enter the relative path of the directory that holds the files to be listed.
- In the **Filemask** field, click the plus button to add one line and then define a file mask to filter the data to recuperate. You can use special characters if need be. In this example, we want only to recuperate delimited files (**csv*).
- In the **Connect Mode** list, select the FTP server connection mode you want to use, **Active** in this example.
- Double-click **tFTPGet** to display its **Basic settings** view and define the components properties.



- Select the **Use an existing connection** check box and in the **Component list**, click the relevant FTP connection component, **tFTPConnection_1** in this scenario. Connection information are automatically filled in.
- In the **Local directory** field, enter the relative path for the output local directory where you want to write the recuperated files.
- In the **Remote directory** field, enter the relative path of the remote directory that holds the file to be recuperated.
- In the **Transfer Mode** list, select the FTP transfer mode you want to use, **ascii** in this example.
- In the **Overwrite file** field, select an option for you want to use for the transferred files.
- In the **Files** area, click the plus button to add a line in the **Filemask** list, then click in the added line and press**Ctrl+Space** to access the variable list. In the list, select the global variable `((String)globalMap.get("tFTPFileList_1_CURRENT_FILEPATH"))` to process all files in the remote directory.
- In the **Connect Mode** list, select the connection mode to the FTP server you want to use.
- Save your Job and press **F6** to execute it.



All .csv files held in the remote directory on the FTP server are listed in the defined directory, as defined in the filemask. Then the files are recuperated and saved in the defined local output directory.

tFTPGet

tFTPGet properties

Component family	Internet/FTP	
Function	This component retrieves defined files via an FTP connection.	
Purpose	**tFTPGet** retrieves selected files from a defined remote FTP directory and copy them into a local directory.	
Basic settings	*Host*	FTP IP address
	Port	Listening port number of the FTP server.
	Username	FTP user name.
	Password	FTP password.
	Local directory	Path to destination location of the file.
	Remote directory	Path to source directory where the files can be fetched.
	Transfer mode	Different FTP transfer modes.
	Overwrite file	List of available options for the transferred file.
	Files	File names or path to the files to be transferred.
Usage	This component is typically used as a single-component sub-job but can also be used as output or end object.	
Limitation	n/a	

Related scenario

For **tFTPGet** related scenario, see *Scenario: Putting files on a remote FTP server on page 557*.

For **tFTPGet** related scenario, see *Scenario: Iterating on a remote directory on page 553*.

For **tFTPGet** related scenario using a different protocol, see *Scenario: Getting files from a remote SCP server on page 582*.

tFTPPut

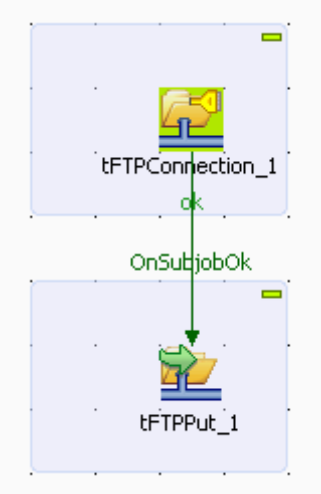
tFTPPut properties

Component family	Internet/FTP	
Function	This component copies defined files via an FTP connection.	
Purpose	**tFTPPut** copies selected files from a defined local directory to a destination remote FTP directory.	
Basic settings	*Use an existing connection/Component List*	A connection needs to be open to allow the loop to check for FTP data on the defined DB.
	Host	FTP IP address.
	Port	Listening port number of the FTP server.
	Username	FTP user name.
	Password	FTP password.
	Local directory	Path to the source location of the file(s).
	Remote directory	Path to the destination directory of the file(s).
	Transfer mode	Different FTP transfer modes.
	Overwrite file or *Append*	List of available options for the transferred file
	SFTP Support and *Authentication method*	**SFTP Support**: select this check box to set an authentication method. **Authentication method**: select the authentication method from the list.
	Files	Click the **[+]** button to add a new line, then fill in the columns. **Filemask**: file names or path to the files to be transferred. **New name**: name to give the FTP file after the transfer.
Usage	This component is typically used as a single-component sub-job but can also be used as output or end object.	
Limitation	n/a	

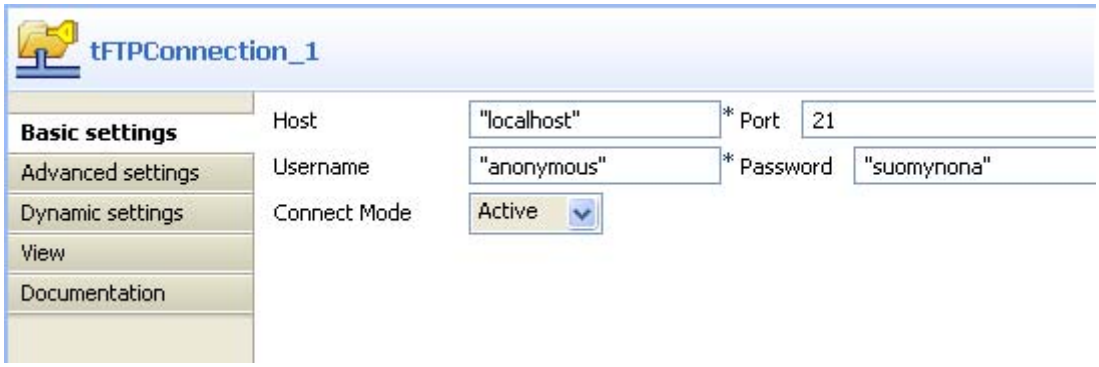
Scenario: Putting files on a remote FTP server

This two-component Job allows to open a connection to a remote FTP server in order to put specific files on the remote server in one transaction.

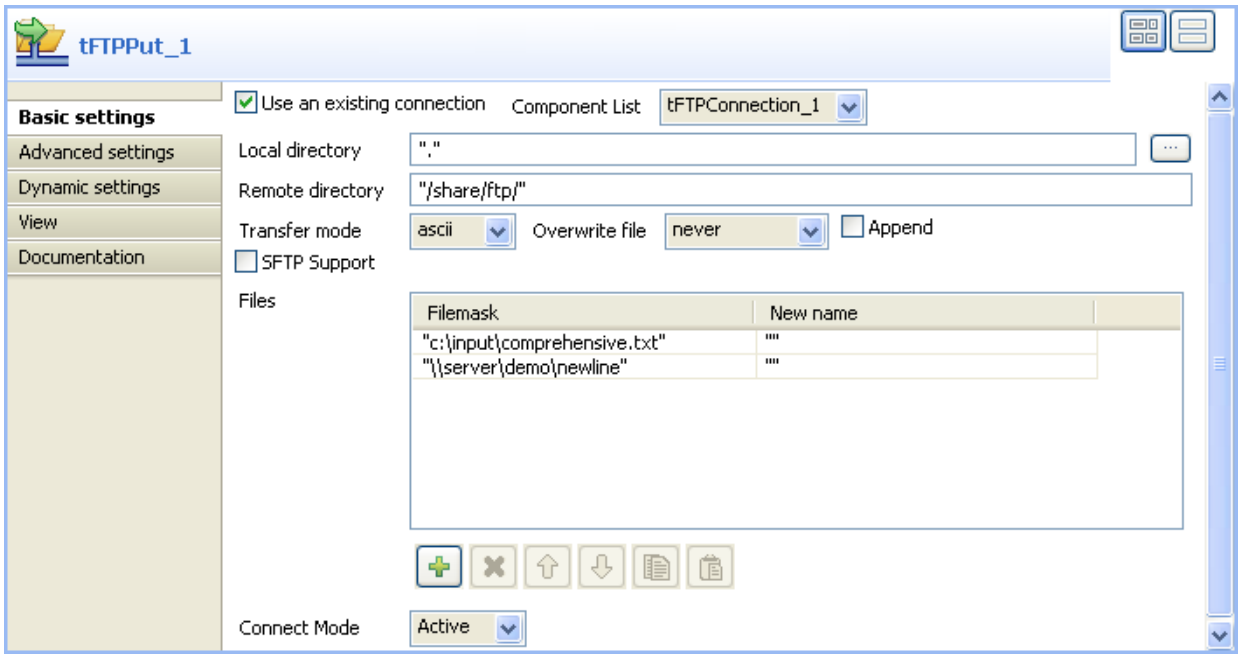
- Drop **tFTPConnection** and **tFTPPut** from the **Palette** onto the design workspace. **tFTPConnection** allows to perform all operations in one transaction.
- Connect the two components together using an **OnSubJobOK** link.



- Double-click **tFTPConnection** to display its **Basic settings** view and define its properties.



- In the **Host** field, enter the IP address of the server.
- In the **Port** field, enter the number of the listening port.
- In the **Username** and **Password** fields, enter your login and password for the remote server.
- From the **Connect Mode** list, select the FTP connection mode you want to use, **Active** in this example.
- In the design workspace, double-click **tFTPPut** to display its **Basic settings** view and define its properties.



- Select the **Use an existing connection** check box and then select **tFTPConnection_1** from the **Component List**. Connection information are automatically filled in.
- In the **Local directory** field, enter the path to the local directory containing the files, if all your files are in the same directory. If files are in different directories, enter the path for each file in the **Filemask** column of the **Files** table.
- In the **Remote directory** field, enter the path to the destination directory on the remote server.
- From the **Transfer mode** list, select the transfer mode to use.
- From the **Overwrite file** list, select an option for the transferred file.
- In the **Files** table, click twice the plus button to add two lines to the **Filemask** column and then fill in the filemasks of all files to be copied onto the remote directory.
- Save you Job and click **F6** to execute it.

Files defined in the **Filemask** column are copied on the remote server.

tMomInput

tMomInput Properties

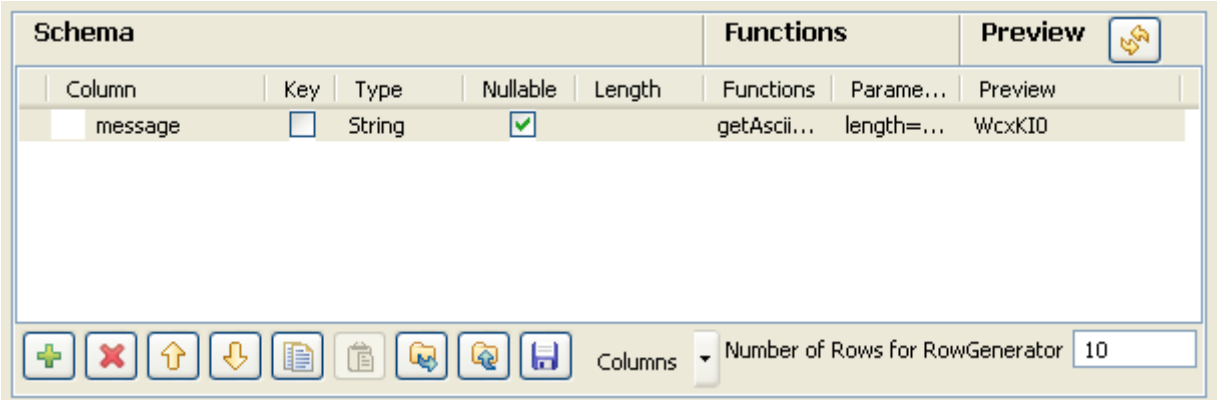
Component family	Internet	
Function	Fetches a message from a queue on a Message-Oriented middle ware system and passes it to the next component.	
Purpose	**tMomInput** makes it possible to set up asynchronous communications via a MOM server.	
Basic settings	*MQ Server*	Select in the list the MOM server to be used. According to the server selected, the parameters required slightly differ.
	Host/Port	Fill in the Host name or IP address of the MOM server as well as the Port.
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. In the context of **tMomInput** usage, the schema is read-only. It is made of two columns: **From** and **Message**
JBoss Messaging	*Keep listening/Sleeping time*	Select this check box to keep listening the MOM server for fetching any new message. Set the frequency of verification in seconds.
	Message From	Type in the message source, exactly as expected by the server; this must include the type and name of the source. e.g.: queue/A or topic/testtopic Note that the field is case-sensitive.
	Message Type	Select the message type, either: **topic** or **queue**.
Websphere	*Channel*	Value by default is Channel
	Queue Manager	Fill in the server driver details
	Message Queue	Source of the message
	Is using message id to fetch	Select this check box to fetch messages according to their ids.
	Message id	Enter the relevant message id.
Usage	This component is generally used as a start component. It requires to be linked to an output component.	
Limitation	Make sure the relevant JBoss or Websphere server is launched.	

Scenario: asynchronous communication via a MOM server

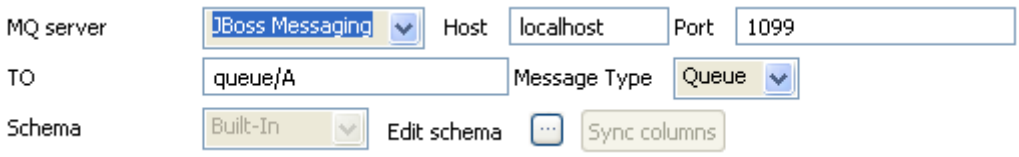
This scenario is made of two Jobs. The first Job aims at posting messages onto a JBoss server queue and the second Job fetches the message from the server.

In the first Job, a string message is created using a **tRowGenerator** and put on a JBoss server using a **tMomOutput**. An intermediary **tLogRow** component displays the flow being passed.

- Drop the three components of this first Job from the **Palette** to the design workspace and right-click to connect them using a **Main** row link.
- Double-click on the **tRowGenerator** to set the schema to be randomly generated.



- Set only one column called *message*. This is the message to be put on the MOM queue.
- This column is of **String** type and is nullable.To produce the data, use a preset function which concatenates randomly chosen ascii characters to form a 6-char string. This function is `getAsciiRandomString`. (Java version). Click the Preview button to view a random sample of data generated.
- Set the **Number of rows to be generated** to *10*.
- Click **OK** to validate.
- The **tLogRow** is only used to display a intermediary state of the data to be handled. In this example, it doesn't require any specific configuration.
- Then select the **tMomOutput** component.



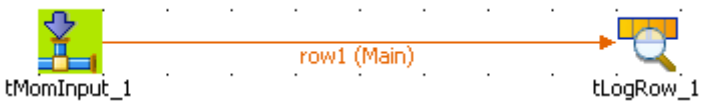
- In this use case, the **MQ server** to be used is **JBoss**.
- In **Host** and **Port** fields, fill in the relevant connection information.
- Select the **Message type** in the list. The message can be of **Queue** or **Topic** type. In this example, select the **Queue** type on the list.
- In the **To** field, type in the message source information strictly as expected by the server. This should match the Message Type you selected, such as: `queue/A`.

The message name is case-sensitive, therefore `queue/A` and `Queue/A` are different.

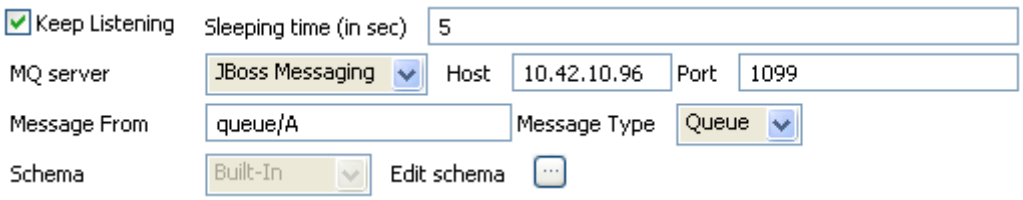
- Then click **Sync Columns** to pass on the schema from the preceding component. The schema being read-only, it cannot be changed. The data posted onto the MQ come from the first encountered column of the schema.
- Press **F6** to execute the Job and see the console the data flow being passed on thanks to the **tLogRow** component.

```
Starting job Mominput at 17:46 14/09/2007.
c8X5GC
lEhC41
opGYqP
x6gOt8
ESknZp
tXU2ET
6H7Nwl
8UA9eM
2sebwV
rfPZAP
Job Mominput ended at 17:46 14/09/2007. [exit code=0]
```

Then set the second Job in order to fetch the queuing messages from the MOM server.



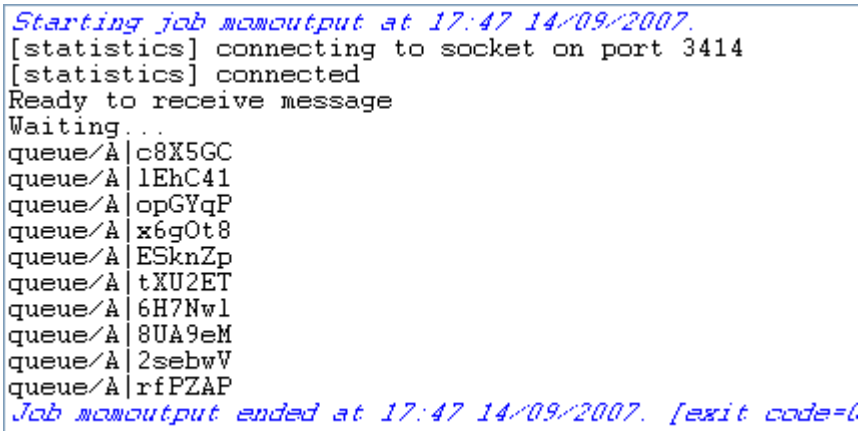
- Drop the **tMomInput** component and a **tLogRow** from the **Palette** to the design workspace.
- Select the **tMomInput** to set the parameters.



- Select the **MQ server** on the list. In this example, a JBoss messaging server is used.
- Set the server **Host** and **Port** information.
- Set the **Message From** and the **Message Type** to match the source and type expected by the messaging server.
- The **Schema** is read-only and is made of two columns: **From** and **Message**.
- Select the **Keep listening** check box and set the frequency of verification to 5 seconds.

When using the **Keep Listening** option, you'll need to kill the Job to end it.

- No need to change any default setting from the **tLogRow**.
- Save the Job and run it (when launching for the first time or if you killed it on a previous run).



The messages fetched on the server are displayed on the console.

tMomMessageIdList

tMomMessageIdList Properties

Component family	Internet	
Function	**tMomMessageIdList** fetches a message ID list from a queue on a Message-Oriented middle ware system and passes it to the next component.	
Purpose	**tMomMessageIdList** makes it possible to iterate on certain message IDs. It is usually used with **tMomInput**, for more information, see *tMomInput Properties on page 560*.	
Basic settings	*MQ Server*	Select in the list the MOM server to be used. According to the server selected, the parameters required slightly differ.
	Host/Port	Fill in the Host name or IP address of the MOM server and of the Port.
Websphere	*Channel*	Channel on the queue.
	Queue Manager	Fill in the server driver details.
	Message Queue	Source of the message.
Usage	This component is generally used as a start component.	
Limitation	Make sure the relevant Websphere server is launched.	

Related scenario

For a related scenario, see *tMomInput on page 560*.

tMomOutput

tMomOutput Properties

Component family	Internet	
Function	Puts a message in a queue of a Message-Oriented middle ware system in order for it to be fetched asynchronously.	
Purpose	**tMomOutput** makes it possible to set up asynchronous communications via a MOM server.	
Basic settings	*MQ Server*	Select in the list the MOM server to be used. According to the server selected, the parameters required slightly differ.
	Host/Port	Fill in the Host name or IP address of the MOM server as well as the Port.
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. In the context of **tMomOutput** usage, the schema is read-only but will change according to the incoming schema. Only one-column schema is expected by the server to contain the **Messages**
JBoss Messaging	*To*	Type in the message destination, exactly as expected by the server; this must include the type and name of the target folder. e.g.: queue/A or topic/testtopic Note that the field is case-sensitive.
	Message Type	Select the message type, either: **topic** or **queue**.
Websphere	*Channel*	Value by default is Channel
	Queue Manager	Fill in the server driver details
	Message Queue	Destination of the message
	Is using message id to set	Select this check box to set messages according to their ids
Usage	This component requires to be linked to an input or intermediary component.	
Limitation	Make sure the relevant JBoss or Websphere server is launched.	

Related scenario

For a related scenario, see *tMomInput on page 560.*

tPOP

tPOP properties

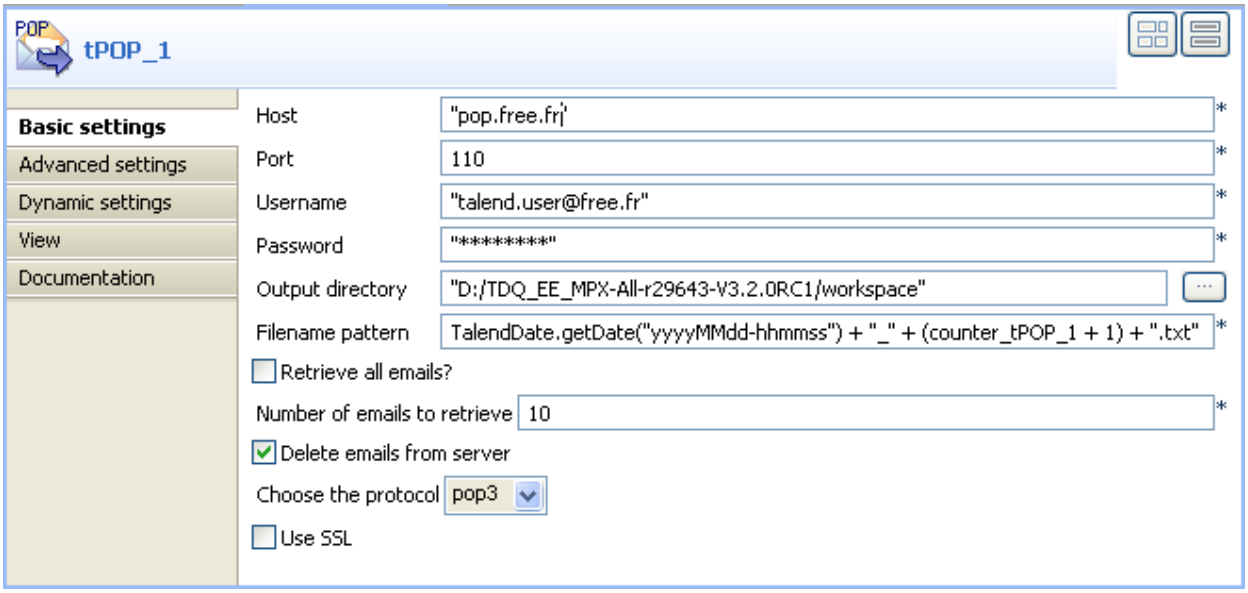
Component family	Internet	
Function	The **tPOP** component fetches one or more email messages from a server using the POP3 or IMAP protocol.	
Purpose	The **tPOP** component uses the POP or IMAP protocol to connect to a specific email server. Then it fetches one or more email messages and write the recovered information in defined files. Parameters in the **Advanced settings** view allows to use filters on your selection.	
Basic settings	*Host*	IP address of the email server you want to connect to.
 *Java-only field*	*Port*	Port number of the email server.
	Username and *Password*	User authentication data for the email server. **Username**: enter the username you use to access your email box. **Password**: enter the password you use to access your email box.
	Output directory	Enter the path to the file in which you want to store the email messages you retrieve from the email server, or click the three-dot button next to the field to browse to the file.
	Filename pattern	Define the syntax of the names of the files that will hold each of the email messages retrieved from the email server, or press **Ctrl+Space** to display the list of predefined patterns.
	Retrieve all emails?	By default, all email messages present on the specified server are retrieved. To retrieve only a limited number of these email messages, clear this check box and in the **Number of emails to retrieve**.field, enter the number of messages you want to retrieve. email messages are retrieved starting from the most recent.
	Delete emails from server	Select this check box if you do not want to keep the retrieved email messages on the server.
 *Java-only field*	*Choose the protocol*	Select in the list the protocol to use for retrieving the email messages from the server. This protocol is the one used by the email server.
 *Java-only field*	*Use SSL*	Select this check box if your email server uses this protocol for authentication and communication confidentiality.  This option is obligatory for users of the Gmail email.
Advanced settings	*tStatCatcher Statistics*	Select this check box to gather the job processing metadata at a job level as well as at each component level.

Java-only field	*Filter*	Click the plus button to add as many lines as needed to filter email messages and retrieve only a specific selection:
		Filter item: select one of the following filter types from the list: **From**: email messages are filtered according to the sender email address. **To**: email messages are filtered according to the recipient email address. **Subject**: email messages are filtered according to the message subject matter. **Before date**: email messages are filtered by the sending or receiving date. All messages before the set date are retrieved. **After date**: email messages are filtered by the sending or receiving date. All messages after the set date are retrieved.
		Pattern: press **Ctrl+Space** to display the list of available values. Select the value to use for each filter.
Java-only field	*Filter condition relation*	Select the type of logical relation you want to use to combine the specified filters: **and**: the conditions set by the filters are combined together, the research is more restrictive. **or**: the conditions set by the filters are independent, the research is large.
Usage	This component does not handle data flow, it can be used alone.	
Limitation	n/a	

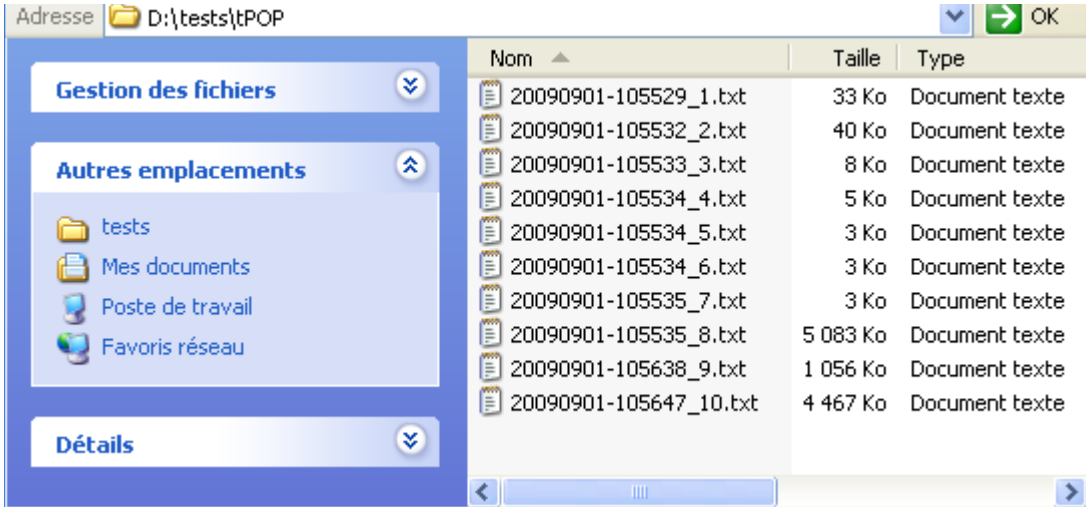
Scenario: Retrieving a selection of email messages from an email server

This Java scenario is a one-component Job that retrieves a predefined number of email messages from an email server.

- Drop the **tPOP** component from the **Palette** to the design workspace.
- Double click **tPOP** to display the **Basic settings** view and define the component properties.
- Enter the email server IP address and port number in the corresponding fields.
- Enter the username and password for your email account in the corresponding fields. In this example, the email server is called *Free*.



- In the **Output directory** field, enter manually the path to the output directory or click the three-dot button next to the field and browse to the output directory where to store the email messages retrieved from the email server.
- In the **Filename pattern** field, define the syntax you want to use to name the output files that will hold the messages retrieved from the email server, or press **Ctrl+Space** to display a list of predefined patterns. The syntax used in this example is the following:
  `TalendDate.getDate("yyyyMMdd-hhmmss") + "_" + (counter_tPOP_1 + 1) + ".txt"`
  The output files will be stored as .txt files and are defined by date, time and arrival chronological order.
- Clear the **Retrieve all emails?** field and in the **Number of emails to retrieve** field, enter the number of email messages you want to retrieve, 10 in this example.
- Select the **Delete emails from server** check box to delete the email messages from the email server once they are retrieved and stored locally.
- In the **Choose the protocol** field, select the protocol type you want to use. This depends on the protocol used by the email server. Certain email suppliers, like *Gmail*, use both protocols. In this example, the protocol used is *pop3*.
- Save your Job and press **F6** to execute it.



The **tPOP** component retrieves the 10 recent messages from the specified email server.

In the *tPOP* directory stored locally, a .txt file is created for each retrieved message. Each file holds the metadata of the email message headings (sender's address, recipient's address, subject matter) in addition to the message content.

tRSSInput

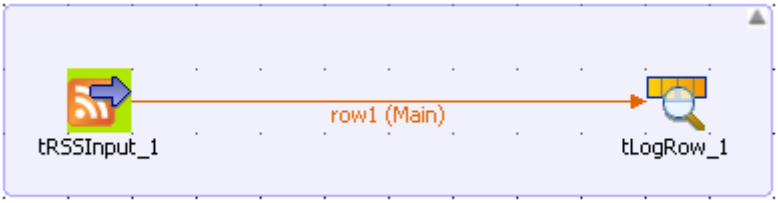
tRSSInput Properties

Component family	Internet	
Function	**tRSSInput** reads RSS-Feeds using URLs.	
Purpose	**tRSSInput** makes it possible to keep track of blog entries on websites to gather and organize information for you quickly and easily.	
Basic settings	*Schema type* and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the Repository. In the context of **tRSSInput** usage, the schema is made of four columns: **TITLE**, **DESCRIPTION**, **PUBDATE**, and **Link**. The parameter titles are read-only while their type and length are not.
	RSS URL	Enter the URL for the RSS_Feed to read.
	Read articles from	If selected, **tRSSInput** reads articles on the RSS_Feed from the date set through the three-dot [...] button next to the **date time** field.
	Max number of articles	If selected, **tRSSInput** reads as many articles as the number entered in the **max amount** field.
	Die on error	Clear this check box to skip errors and complete the process.
Usage	This component is generally used as a start component. It requires to be linked to an output component.	
Limitation	n/a.	

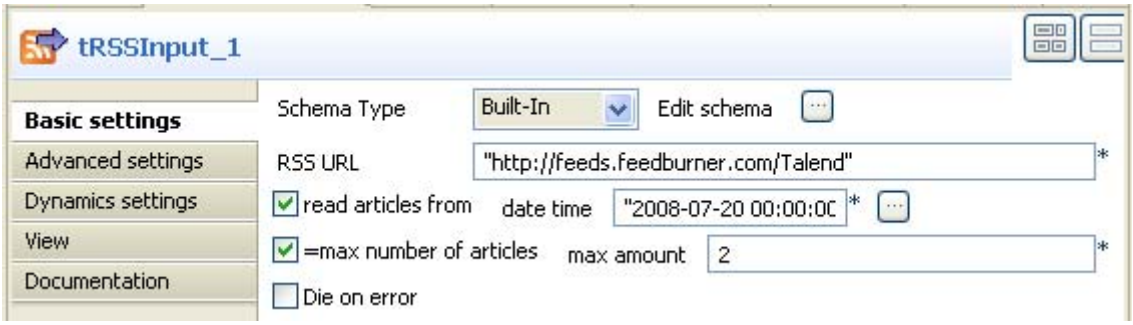
Scenario: Fetching frequently updated blog entries.

This two-component Java scenario aims at retrieving frequently updated blog entries from a **Talend** local news RSS feed using the **tRSSInput** component.

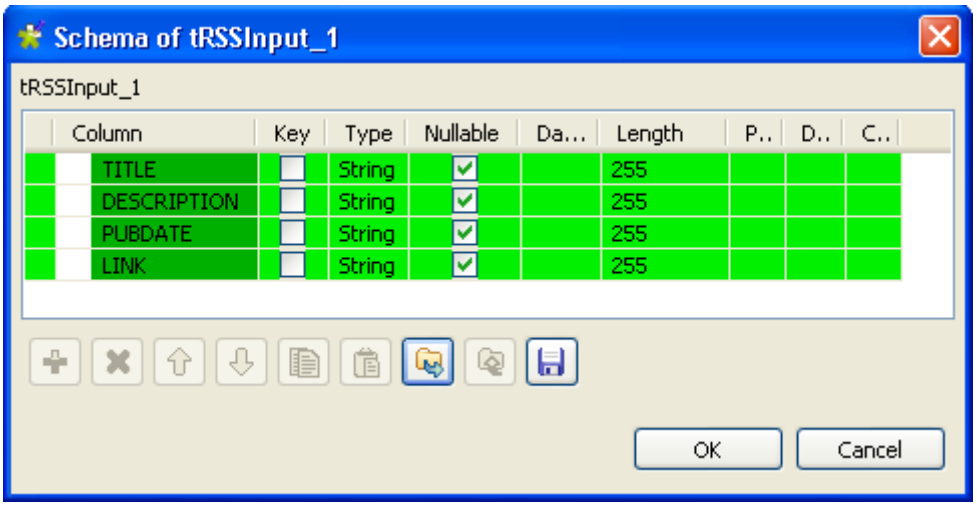
- Drop the following components from the **Palette** onto the design workspace: **tRSSInput** and **tLogRow**.
- Right-click to connect them using a **Row Main** link.



- In the design workspace, select **tRSSInput**.
- Click the **Component** tab to define the basic settings for **tRSSInput**.



- Set the **Schema Type** to **Built-In** and click the three-dot [...] button next to **Edit Schema** to change the type and length of the schema parameters if necessary.
- Click **OK** to close the dialog box.



The scheme for **tRSSInput** is made up of four columns, *TITLE*, *Description*, *PUBDATE*, and LINK, and it is read-only apart from the type and length of parameters.

- In the **Basic settings** view of **tRSSInput**, enter the URL for the RSS_Feed to access. In this scenario, **tRSSInput** links to the **Talend** RSS_Feed: *http://feeds.feedburner.com/Talend*.
- Select/clear the other check boxes as needed. In this scenario, we want to display the information about two articles dated from July 20, 2008.

- In the design workspace, select **tLogRow** and click the **Component** tab to define its basic settings. For more information about **tLogRow** properties, see *tLogRow properties on page 628*.
- Save the Job and press **F6** to execute it.

```
Starting job RSSInput at 15:17 07/08/2008.
.-------------------------------------------------------------------------------
|                                        #1. tLogRow_1
+-------------+-----------------------------------------------------------------
| key         | value
+-------------+-----------------------------------------------------------------
| TITLE       | Welcoming Jean-Luc Solans
| DESCRIPTION | Jean-Luc Solans joins Talend as VP of Strategy and Busines
| PUBDATE     | 24 Jul 2008 19:40:11 GMT
| LINK        | http://feeds.feedburner.com/~r/Talend/~3/344920575/
+-------------+-----------------------------------------------------------------

.-------------------------------------------------------------------------------
|                                    #2. tLogRow_1
+-------------+-----------------------------------------------------------------
| key         | value
+-------------+-----------------------------------------------------------------
| TITLE       | Talend Open Profiler gets rave reviews
| DESCRIPTION | An interview and a product review of Talend Open Profiler.
| PUBDATE     | 23 Jul 2008 21:12:53 GMT
| LINK        | http://feeds.feedburner.com/~r/Talend/~3/343928056/
+-------------+-----------------------------------------------------------------

Job RSSInput ended at 15:17 07/08/2008. [exit code=0]
```

The **tRSSInput** component accessed the RSS feed of **Talend** website on your behalf and organized the information for you.

Two blog entries are displayed on the console. Each entry has its own title, description, publication date, and the corresponding RSS feed URL address. Blogs show the last entry first, and you can scroll down to read earlier entries.

tRSSOutput

tRSSOutput Properties

Component family	Internet	
Function	**TRSSOutput** writes RSS_Feed XML files.	
Purpose	**tRSSOutput** makes it possible to create XML files that hold RSS feeds.	
Basic settings	*File name*	Name or path to the output XML file. Related topic: *Defining variables from the Component view* of **Talend Open Studio** User Guide.
	Encoding type	Select an encoding type from the list, or select **Custom** and define it manually. This field is compulsory for DB data handling.
	Channel	**Note:** the information to type in here is about your entire input data, site...etc, rather than about a particular item. **Title**: Enter a meaningful title. **Description**: Enter a description that you think will describe your content. **Publication date**: Enter the relevant date. **Link**: Enter the relevant UR.
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the Repository. In the context of **tRSSInput** usage, the schema is made of four columns: **TITLE**, **DESCRIPTION**, **PUBDATE**, and **Link**. The parameter titles are read-only while their type and length are not.
Usage	This component requires to be linked to an input or intermediary component.	
Limitation	n/a	

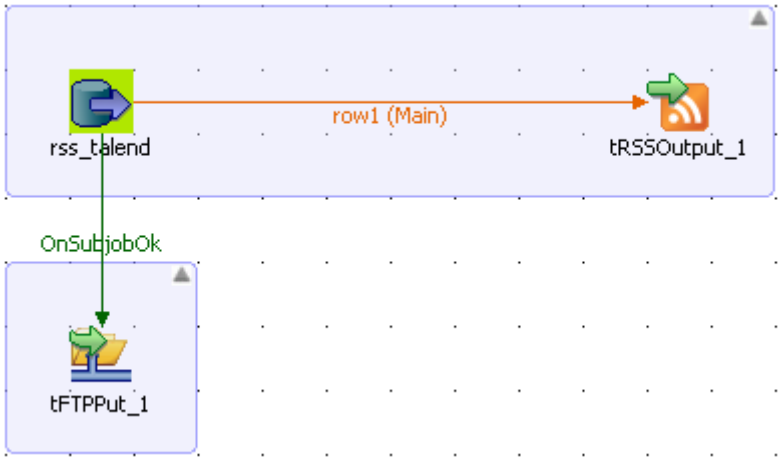
Scenario: Creating an RSS flow.

This Java scenario aims at:

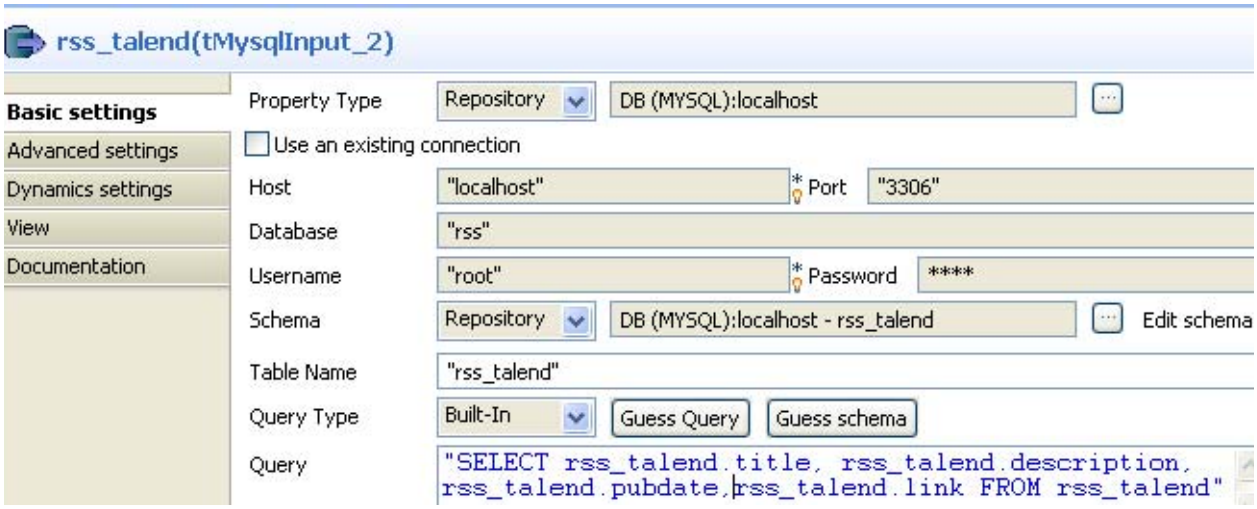
- creating an RSS flow for files that you would like to share with other people, and
- storing the complete files on an FTP server.

This scenario writes an RSS feed XML file about a Mysql table holding information about books. It adds links to the files stored on an FTP server in case users want to have access to the complete files.

- Drop the following components from the **Palette** onto the design workspace: **tMysqlInput**, **tRSSOutput**, and **tFTPPut**.
- Right-click **tMysqlInput** and connect it to **tRSSOutpu** using a **Row Main** link.
- Right-click **tMysqlInput** and connect it to **tFTPPut** using an **OnSubjobOk** link.

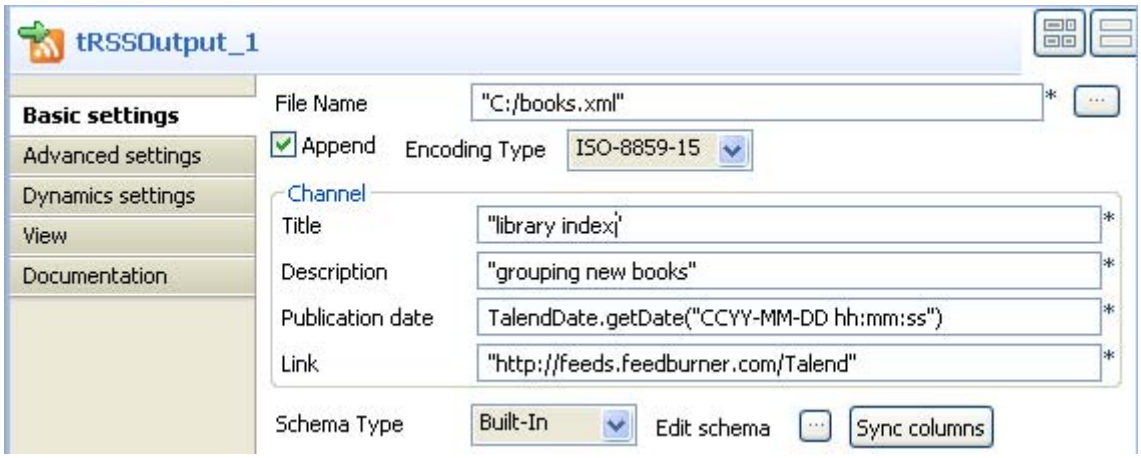


- In the design workspace, select **tMysqlInput**.
- click the **Component** tab to define the basic settings for **tMysqlInput**.



- Set the **Property type** to **Repository** and click the three-dots button [...] to select the relevant DB entry from the list. The connection details along with the schema get filled in automatically.
- In the **Table Name** field, either type your table name or click the three dots button [...] and select your table name from the list. In this scenario, the Mysql input table is called “rss_talend” and the schema is made up of four columns, *TITLE*, *Description*, *PUBDATE*, and LINK.
- In the **Query** field, enter your DB query paying particular attention to properly sequence the fields in order to match the schema definition, or click **Guess Query**.
- In the design workspace, select **tRSSOutput**.

- click the **Component** tab to define the basic settings for **tRSSOutput**.



- In the **File name** field, use the par default file name and path, or browse to set your own for the output XML file.
- Select the encoding type on the **Encoding Type** list.
- On the **Channel** panel, enter a title, a description, a publication date, and a link to define your input data as a whole.
- Select your schema type on the **Schema Type** list and click **Edit Schema** to modify the schema if necessary.

  You can click **Sync Column** to retrieve the generated schema from the preceding component.

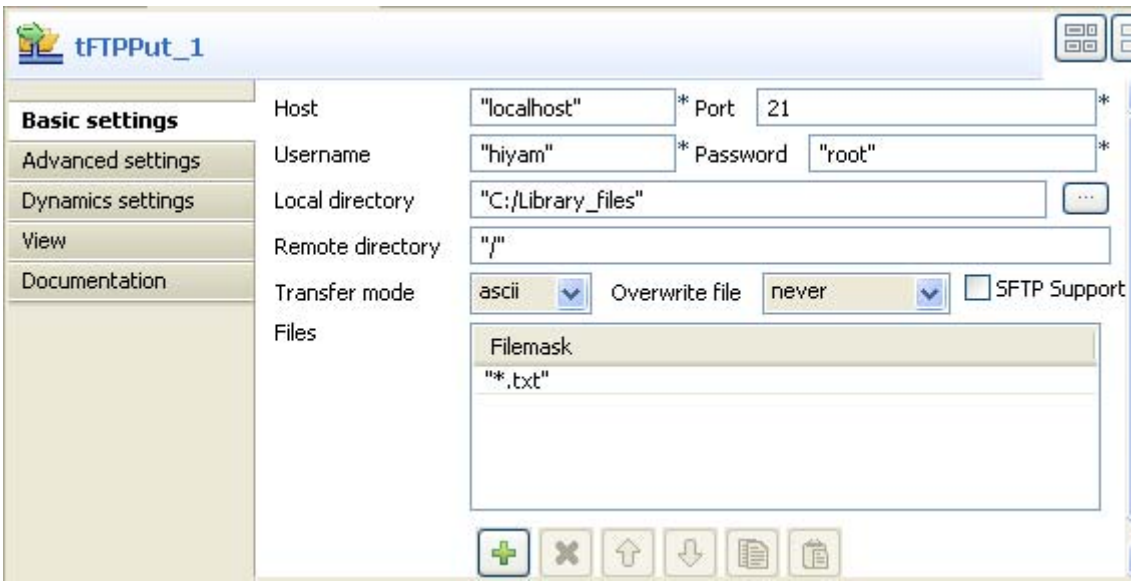
- Save your Job and press **F5** to execute this first part.

```
<?xml version="1.0" encoding="UTF-8" ?>
- <rss version="2.0">
  - <channel>
      <title>library index</title>
      <description>grouping new books</description>
      <pubdate>2008-08-16 14:55:29</pubdate>
      <link>http://feeds.feedburner.com/Talend</link>
    - <item>
        <title>Book 1</title>
        <description>Summary of book1</description>
        <pubdate>2008-01-02</pubdate>
        <link>ftp://localhost/file_1.txt</link>
      </item>
    - <item>
        <title>Book 2</title>
        <description>Summary of book2</description>
        <pubdate>2008-01-02</pubdate>
        <link>ftp://localhost/file_2.txt</link>
      </item>
    - <item>
        <title>Book 3</title>
        <description>Summary of book3</description>
        <pubdate>2008-01-02</pubdate>
        <link>ftp://localhost/file_3.txt</link>
      </item>
  </channel>
</rss>
```

The **tRSSOutput** created an output RSS flow in an XML format for the defined files.

To store the complete files on an FTP server:

- In the design workspace, select **FTPPut**.
- Click the **Component** tab to define the basic settings for **tFTPPut**.



- Enter the host name and the port number in their corresponding fields.
- Enter your connection details in the corresponding **Username** and **Password** fields.
- Browse to the local directory, or enter it manually in the **Local directory** field.
- Enter the details of the remote server directory.
- Select the transfer mode from the **Transfer mode** list.
- On the **Files** panel, click on the plus button to add new lines and fill in the filemasks of all files to be copied onto the remote directory. In this scenario, the files to be saved on the FTP server are all text files.
- Save your Job and press **F6** to execute it.

Files defined in the Filemask are copied on the remote server.

tSCPConnection

tSCPConnection properties

Component family	Internet/SCP	
Function	**tSCPConnection** opens an SCP connection for the current transaction.	
Purpose	**tSCPConnection** allows to open an SCP connection to transfer files in one transaction.	
Basic settings	*Host*	IP address of the SCP server.
	Port	Number of listening port of the SCP server.
	Username	User name for the SCP server.
	Authentication method	SCP authentication method.
	Password	User password for the SCP server.
Usage	This component is typically used as a single-component sub-job. It is used along with other SCP components.	
Limitation	n/a	

Related scenarios

For a related scenario, see *Scenario: Putting files on a remote FTP server on page 557*.

For a related scenario using a different protocol, see *Scenario: Getting files from a remote SCP server on page 582*.

tSCPDelete

tSCPDelete properties

Component family	Internet/SCP	
Function	This component deletes files from remote hosts over a fully encrypted channel.	
Purpose	**tSCPDelete** allows to remove a file from the defined SCP server.	
Basic settings	*Host*	SCP IP address.
	Port	Listening port number of the SCP server.
	Username	SCP user name.
	Authentication method	SCP authentication method.
	Password	SCP password.
	Filelist	File name or path to the files to be deleted.
Usage	This component is typically used as a single-component sub-job but can also be used with other components.	
Limitation	n/a	

Related scenario

For **tSCPDelete** related scenario, see *Scenario: Getting files from a remote SCP server on page 582*.

For **tSCPDelete** related scenario using a different protocol, see *Scenario: Putting files on a remote FTP server on page 557*.

tSCPFileExists

tSCPFileExists properties

Component family	Internet/SCP	
Function	This component checks, over a fully encrypted channel, if a file exists on a remote host.	
Purpose	**tSCPFileExists** allows to verify the existence of a file on the defined SCP server.	
Basic settings	*Host*	SCP IP address.
	Port	Listening port number of the SCP server.
	Username	SCP user name.
	Authentication method	SCP authentication method.
	Password	SCP password.
	Remote directory	File path on the remote directory.
	Filename	Name of the file to check.
Usage	This component is typically used as a single-component sub-job but can also be used with other components.	
Limitation	n/a	

Related scenario

For **tSCPFileExists** related scenario, see *Scenario: Getting files from a remote SCP server on page 582*.

For **tSCPFileExists** related scenario using a different protocol, see *Scenario: Putting files on a remote FTP server on page 557*.

tSCPFileList

tSCPFileList properties

Component family	Internet/SCP	
Function	This component iterates, over a fully encrypted channel, on files of a given directory on a remote host.	
Purpose	**tSCPFileList** allows to list files from the defined SCP server.	
Basic settings	*Host*	SCP IP address.
	Port	Listening port number of the SCP server.
	Username	SCP user name.
	Authentication method	SCP authentication method.
	Password	SCP password.
	Command separator	The character used to separate multiple commands.
	*Filelis*t	Directory name or path to the directory holding the files to list.
Usage	This component is typically used as a single-component sub-job but can also be used with other components.	
Limitation	n/a	

Related scenario

For **tSCPFileList** related scenario, see *Scenario: Getting files from a remote SCP server on page 582*.

For **tSCPFileList** related scenario using a different protocol, see *Scenario: Putting files on a remote FTP server on page 557*.

tSCPGet

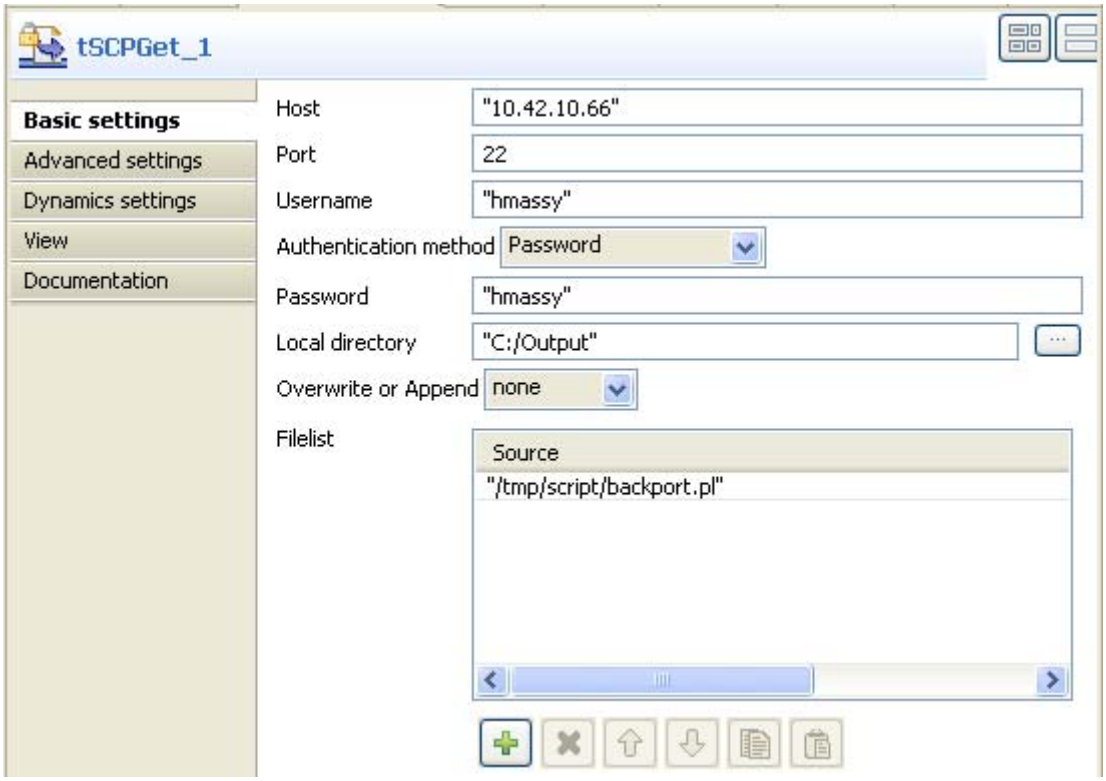
tSCPGet properties

Component family	Internet/SCP	
Function	This component transfers defined files via an SCP connection over a fully encrypted channel.	
Purpose	**tSCPGet** allows to copy files from the defined SCP server.	
Basic settings	*Host*	SCP IP address.
	Port	Listening port number of the SCP server.
	Username	SCP user name.
	Authentication method	SCP authentication method.
	Password	SCP password.
	Local directory	Path to the destination folder.
	Overwrite or *Append*	List of available options for the transferred files.
	Filelist	File name or path to the file(s) to copy.
Usage	This component is typically used as a single-component sub-job but can also be used with other components.	
Limitation	n/a	

Scenario: Getting files from a remote SCP server

This Java scenario creates a single-component Job which gets the defined file from a remote SCP server.

- Drop a **tSCPGet** component from the **Palette** onto the design workspace.
- In the design workspace, select **tSCPGet** and click the **Component** tab to define its basic settings.



- Fill in the **Host** IP address, the listening **Port** number, and the user name in the corresponding fields.
- On the **Authentication method** list, select the appropriate authentication method.
  Note that the field to follow changes according to the selected authentication method. The authentication form used in this scenario is password.
- Fill in the local directory details where you want to copy the fetched file.
- On the **Overwrite or Append** list, select the action to be carried out.
- In the **Filelist** area, click the plus button to add a line in the **Source** list and fill in the path to the given file on the remote SCP server.
  In this scenario, the file to copy from the remote SCP server to the local disk is *backport*.
- Save the Job and press **F6** to execute it.

The given file on the remote server is copied on the local disk.

tSCPPut

tSCPPut properties

Component family	Internet/SCP	
Function	This component copies defined files to a remote SCP server over a fully encrypted channel.	
Purpose	**tSCPPut** allows to copy files to the defined SCP server.	
Basic settings	*Host*	SCP IP address.
	Port	Listening port number of the SCP server.
	Username	SCP user name.
	Authentication method	SCP authentication method.
	Password	SCP password.
	Remote directory	Path. to the destination folder.
	Filelist	File name or path to the file(s) to copy.
Usage	This component is typically used as a single-component sub-job but can also be used with other components.	
Limitation	n/a	

Related scenario

For **tSCPPut** related scenario, see *Scenario: Getting files from a remote SCP server on page 582*.

For **tSCPut** related scenario using a different protocol, see *Scenario: Putting files on a remote FTP server on page 557*.

tSCPRename

tSCPRename properties

Component family	Internet/SCP	
Function	This component renames files on a remote SCP server.	
Purpose	**tSCPRename** allows to rename file(s) on the defined SCP server.	
Basic settings	*Host*	SCP IP address.
	Port	Listening port number of the SCP server.
	Username	SCP user name.
	Authentication method	SCP authentication method.
	Password	SCP password.
	File to rename	Enter the name or path to the file you want to rename.
	Rename to	Enter the file new name.
Usage	This component is typically used as a single-component sub-job but can also be used with other components.	
Limitation	n/a	

Related scenario

For **tSCPRename** related scenario, see *Scenario: Getting files from a remote SCP server on page 582*.

tSCPTruncate

tSCPRename properties

Component family	Internet/SCP	
Function	This component removes all the data from a file via an SCP connection.	
Purpose	**tSCPTruncate** allows to remove data from file(s) on the defined SCP server.	
Basic settings	*Host*	SCP IP address.
	Port	Listening port number of the SCP server.
	Username	SCP user name.
	Authentication method	SCP authentication method.
	Password	SCP password.
	Remote directory	Path. to the destination file.
	Filelist	File name or path to the file(s) to truncate.
Usage	This component is typically used as a single-component sub-job but can also be used with other components.	
Limitation	n/a	

Related scenario

For **tSCPTruncate** related scenario, see *Scenario: Getting files from a remote SCP server on page 582*.

tSendMail

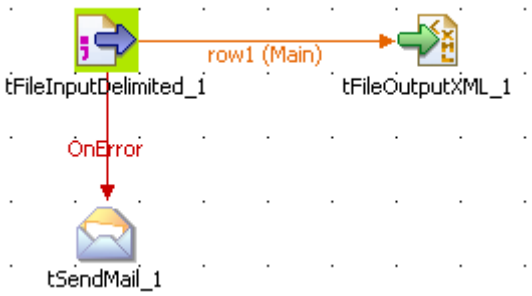
tSendMail Properties

Component family	Internet	
Function	**tSendMail** sends emails and any attachments to defined recipients.	
Purpose	**tSendMail** purpose is to notify recipients about a particular state of a Job or possible errors.	
Basic settings	*To*	Main recipient email address.
	From	Sending server email address.
	Show sender's name	Select this check box if you want the sender name to show in the messages.
	Cc	Email addresses of secondary recipients of the email message directed to another.
	Bcc	Email addresses of secondary recipients of the email message. Recipients listed in the **Bcc** field receive a copy of the message but are not shown on any other recipient's copy.
	Subject	Heading of the mail.
	Message	Body message of the email. Press **Ctrl+Space** to display the list of available variables.
	Die if the attachment file doesn't exist	This check box is selected by default. Clear this check box if you want the message to be sent even if there are no attachments.
	Attachments	Click the plus button to add as many lines as needed where you can put Filemask or path to the file to be sent along with the mail, if any.
	Other Headers	Click the plus button to add as many lines as needed where you can type the key and the corresponding value of any header information that does not belong to the standard header.
	SMTP Host and Port	IP address of SMTP server used to send emails.
	SSL Support	Select this check box to authenticate the server at the client side via an SSL protocol.
	STARTTLS Support	Select this check box to authenticate the server at the client side via a STARTTLS protocol.
	Importance	Select in the list the priority level of your messages.
	Need authentication / Username and Password	Select this check box and enter a username and a password in the corresponding fields if this is necessary to access the service.
	Die on error	Clear this check box to skip the row on error and complete the process for error-free rows.

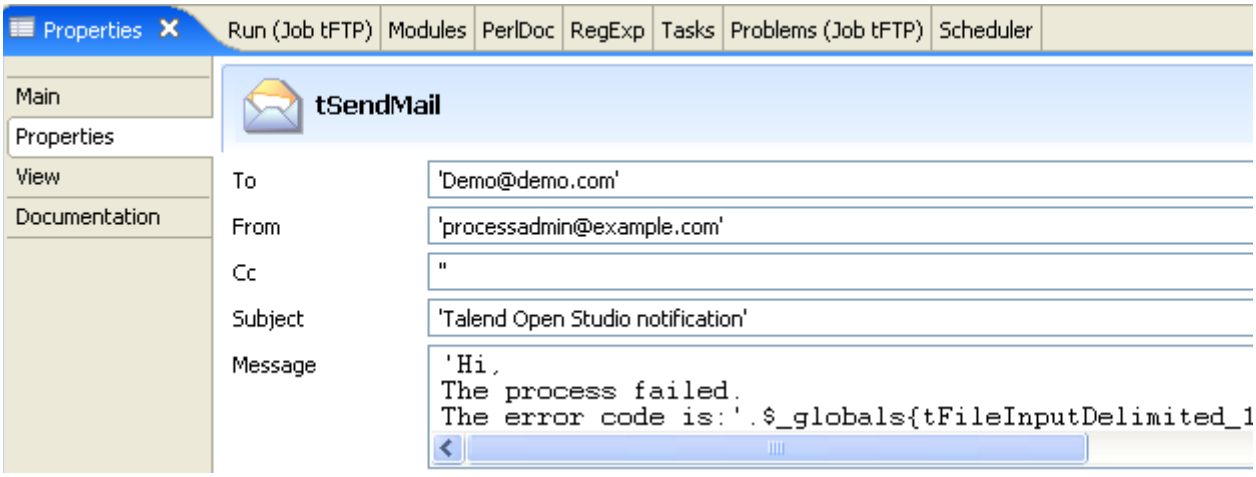
Advanced settings	*MIME subtype from the 'text' MIME type*	Select in the list the structural form for the text of the message.
	Encoding type	Select the encoding from the list or select **Custom** and define it manually.
	tStatCatcher Statistics	Select this check box to gather the job processing metadata at a job level as well as at each component level.
Usage	This component is typically used as one sub-job but can also be used as output or end object. It can be connected to other components with either **Row** or **Iterate** links.	
Limitation	Note that email with or without attachment require two different perl modules.	

Scenario: Email on error

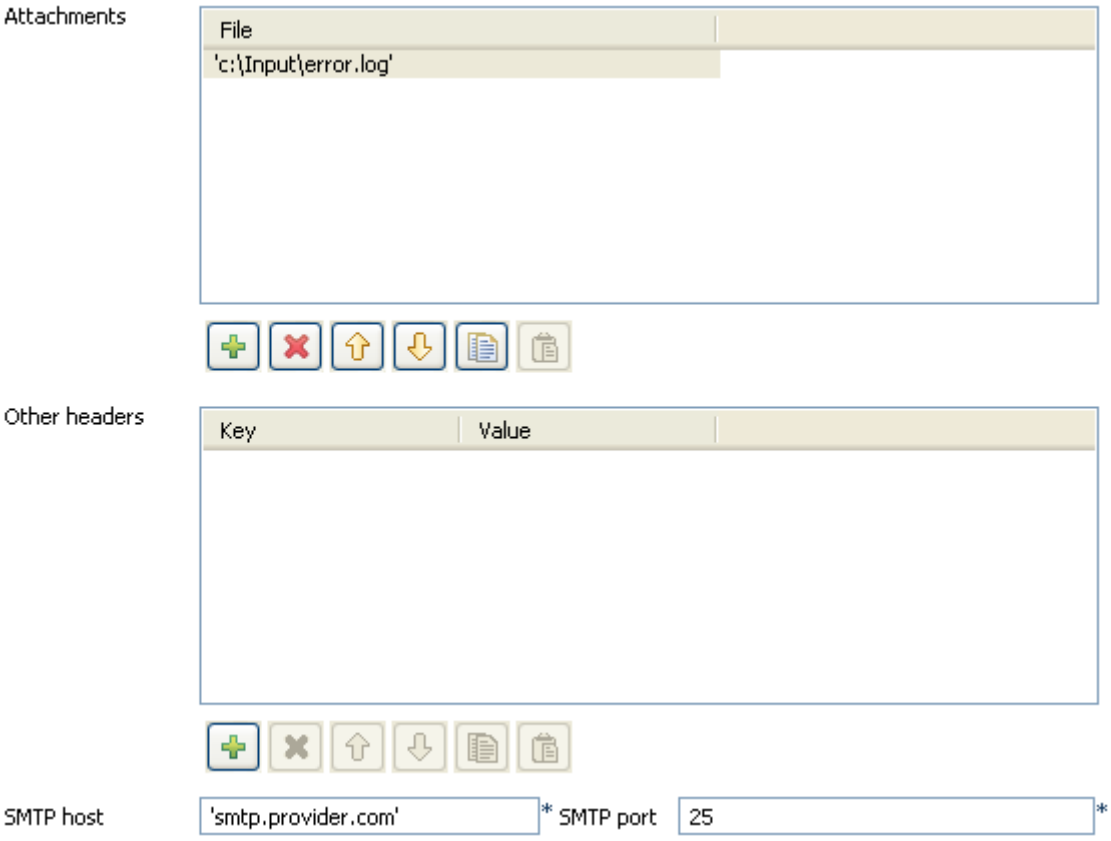
This scenario creates a three-component Job which sends an email to defined recipients when an error occurs.



- Drop the following components from your **Palette** to the design workspace: **tFileInputDelimited**, **tFileOutputXML**, **tSendMail**.
- Define **tFileInputdelimited** properties. Related topic: *tFileInputDelimited on page 494*.
- Right-click on the **tFileInputDelimited** component and select *Row* > *Main*. Then drag it onto the **tFileOutputXML** component and release when the plug symbol shows up.
- Define **tFileOutputXML** properties.
- Drag a **Run on Error** link from **tFileDelimited** to **tSendMail** component.
- Define the **tSendMail** component properties:



- Enter the recipient and sender email addresses, as well as the email subject.
- Enter a message containing the error code produced using the corresponding global variable. Access the list of variables by pressing **Ctrl+Space**.
- Add attachments and extra header information if any. Type in the SMTP information.



In this scenario, the file containing data to be transferred to XML output cannot be found. **tSendmail** runs on this error and sends a notification email to the defined recipient.

tSocketInput

tSocketInput properties

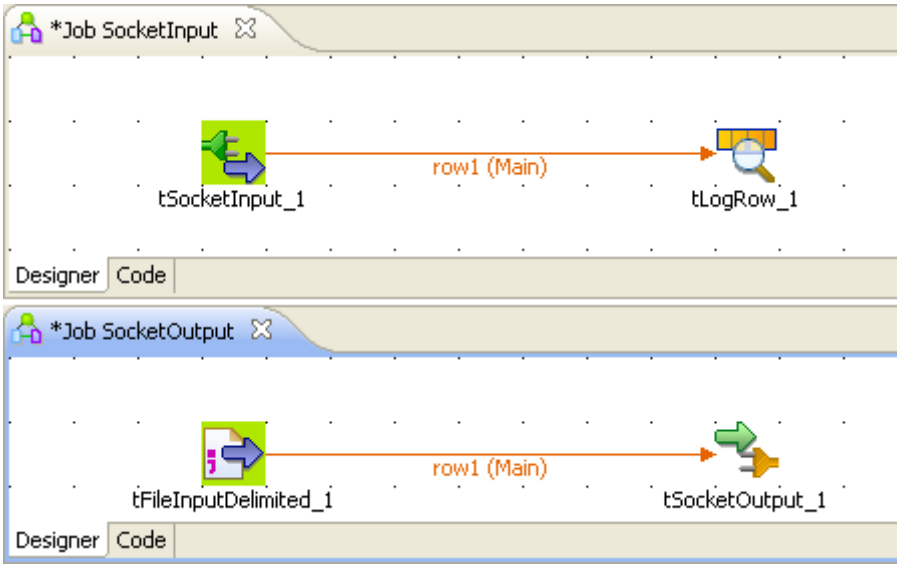
Component family	Internet	
Function	**tSocketInput** component opens the socket port and listens for the incoming data.	
Purpose	**tSocketInput** component is a listening component, allowing to pass data via a defined port	
JAVA Basic settings	*Host name*	Name or IP address of the Host server
	Port	Listening port to open
	Timeout	Number of seconds for the port to listen before closing.
	Uncompress	Select this check box to unzip the data if relevant
	Die on error	Clear this check box to skip the row on error and complete the process for error-free rows.
	Field separator	Character, string or regular expression to separate fields.
	Row separator	String (ex: “\n”on Unix) to distinguish rows.
	Escape Char	Character of the row to be escaped
	Text enclosure	Character used to enclose text.
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the Repository. Click **Edit Schema** to make changes to the schema. Note that if you make changes, the schema automatically becomes built-in.
		Built-in: The schema will be created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused in various projects and job flowcharts. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	Encoding type	Select the encoding from the list or select Custom and define it manually. This field is compulsory for DB data handling.
Usage	This component opens a point of access to a workstation or server. This component starts a Job and only stops after the time goes out.	
Limitation	n/a	

The Perl properties being slightly different from the Java properties, they are described in a separate table below.

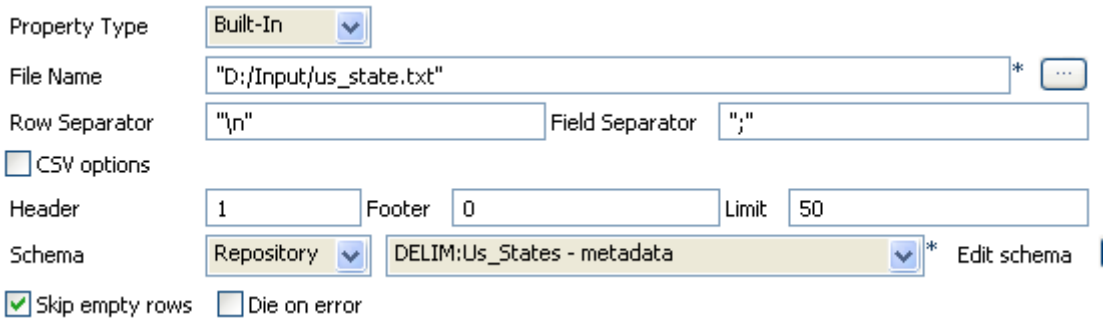
PERL basic settings	*Schema type* and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the Repository. Click **Edit Schema** to make changes to the schema. Note that if you make changes, the schema automatically becomes built-in.
		Built-in: The schema will be created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused in various projects and job flowcharts. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	Host name	Name or IP address of the Host server
	Port	Listening port to open
	Field separator	Character, string or regular expression to separate fields.
	End of Line separator	String (ex: “\n”on Unix) to distinguish row.
	End of data	Character, string or regular expression that points out the end of the data section
	Opening message /Message /Acknowledge message /Closing message	Description of the message if relevant.
Usage	This component opens a point of access to a workstation or server. This component starts a Job and only stops after it receives a closing message.	

Scenario: Passing on data to the listening port (Java)

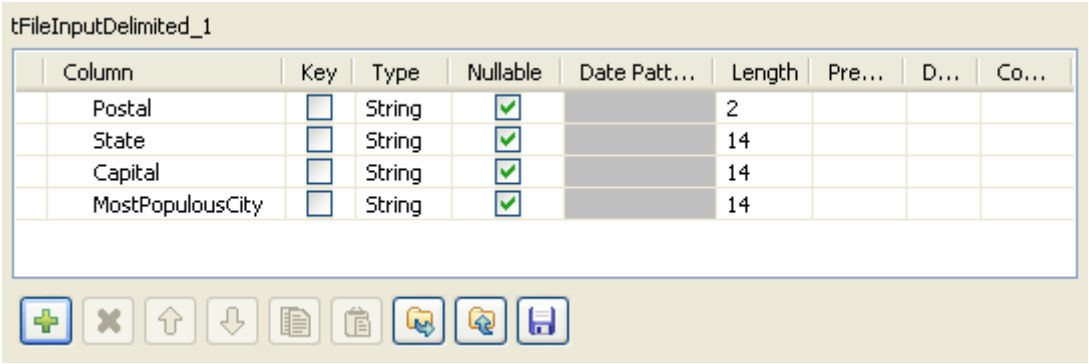
The following scenario describes a double Job aiming at passing data via a listening port.Another application for the Socket components would be to allow controlled communication between servers which cannot communicate directly.



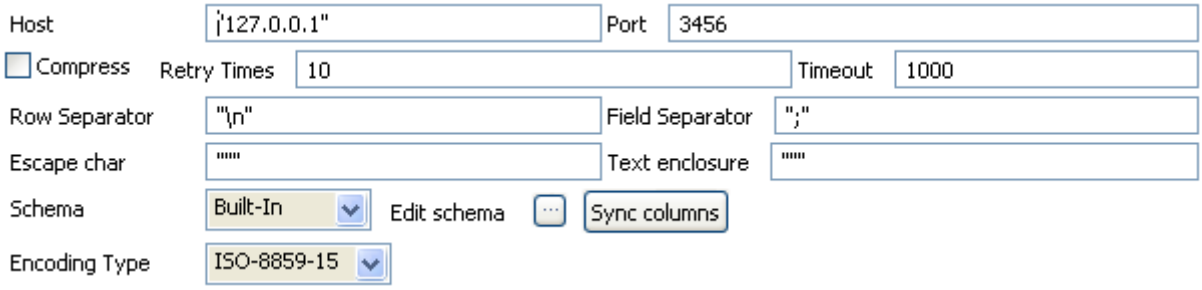
- Create two Jobs: a first Job (*SocketInput*) opens the listening port and waits for the data to be sent over. The second Job (*SocketOutput*) passes delimited data from a file to a defined port number corresponding to the listening port.
- On the first Job, Drop the following components: **tSocketInput** and **tLogRow** from the **Palette** to the design workspace.
- On the second Job, Drop the following components from the **Palette** to the design workspace: **tFileInputDelimited** and **tSocketOutput**.
- Let's set the parameters of the second Job first...
- Select the **tFileInputDelimited** and on the **Basic Settings** tab of the **Component** view, set the access parameters to the input file.



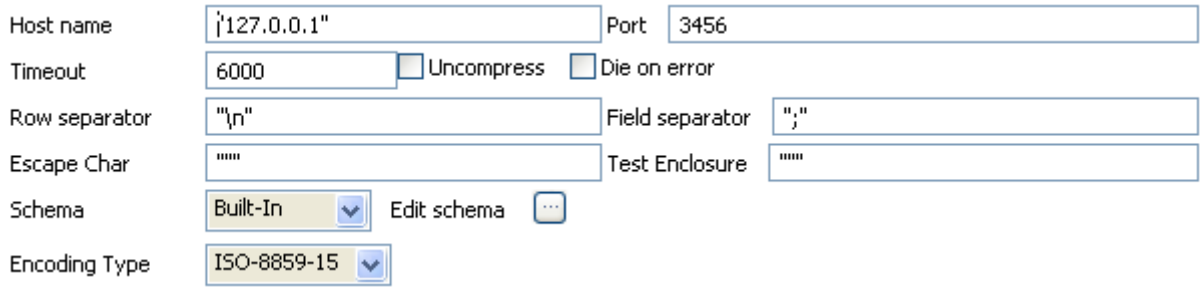
- In **File Name**, browse to the file.
- Define the **Row** and **Field separators**, as well as the **Header**.



- Describe the **Schema** of the data to be passed on to the **tSocketOutput** component.
- Select the **tSocketOutput** component and set the parameters on the **Basic Settings** tab of the **Component** view.



- Define the **Host** IP address and the **Port** number where the data will be passed on to.
- Set the number of retries in the **Retry** field and the amount of time (in seconds) after which the Job will time out.
- Define the rest of elements if need be.
- The schema should be propagated from the preceding component.
- Now on the other Job (*SocketInput*) design, define the parameters of the **tSocketInput** component.



- Define the **Host** IP address and the listening **Port** number where the data are passed on to.
- Set the amount of time (in seconds) after which the Job will time out.
- Define the rest of elements if need be.
- Edit the schema and set it to reflect the whole or part of the other Job's schema.

- The **tLogRow** does not require any particular setting for this Job.
- Press **F6** to execute this Job (SocketInput) first, in order to open the listening port and prepare it to receive the passed data.
- Before the time-out, launch the other Job (SocketOutput) to pass on the data.

The result displays on the **Run** view, along with the opening socket information.

```
Starting job SocketInput at 17:53 04/02/2008.
socket connected
AL|Alabama|Montgomery|Birmingham
AK|Alaska|Juneau|Anchorage
AZ|Arizona|Phoenix|Phoenix
AR|Arkansas|Little Rock|Little Rock
CA|California|Sacramento|Los Angeles
CO|Colorado|Denver|Denver
CT|Connecticut|Hartford|Bridgeport
DE|Delaware|Dover|Wilmington
FL|Florida|Tallahassee|Jacksonville
GA|Georgia|Atlanta|Atlanta
HI|Hawaii|Honolulu|Honolulu
ID|Idaho|Boise|Boise
IL|Illinois|Springfield|Chicago
IN|Indiana|Indianapolis|Indianapolis
```

tSocketOutput

tSocketOutput properties

Component family	Internet	
Function	**tSocketOutput** component writes data to a listening port.	
Purpose	**tSocketOutput** sends out the data from the incoming flow to listening socket port.	
Basic settings	*Host name*	Name or IP address of the Host server
	Port	Listening port to open
	Compress	Select this check box to zip the data if relevant.
	Retry times	Number of retries before the Job fails.
	Timeout	Number of seconds for the port to listen before closing.
	Die on error	Clear this check box to skip the row on error and complete the process for error-free rows.
	Field separator	Character, string or regular expression to separate fields.
	Row separator	String (ex: “\n”on Unix) to distinguish rows.
	Escape Char	Character of the row to be escaped
	Text enclosure	Character used to enclose text.
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the Repository. Click **Edit Schema** to make changes to the schema. Note that if you make changes, the schema automatically becomes built-in.
		Built-in: The schema will be created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused in various projects and job flowcharts. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	Encoding type	Select the encoding from the list or select Custom and define it manually. This field is compulsory for DB data handling.
Usage	This component opens a point of access to a workstation or server. This component starts a Job and only stops after the time goes out.	
Limitation	n/a	

Related Scenario

For use cases in relation with **tSocketOutput**, see *Scenario: Passing on data to the listening port (Java) on page 591*.

tWebServiceInput

tWebServiceInput Properties

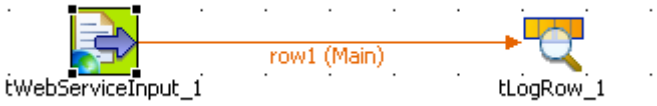
Component family	Internet	
Function	Calls the defined method from the invoked Web service, and returns the class as defined, based on the given parameters.	
Purpose	Invokes a Method through a Web service.  To handle complex hierarchical data, use the advanced features of **tWebServiceInput** and provide Java code directly in the **Code** field of the **Advanced Settings** view.	
Basic settings	Property type	Either **Built-in** or **Repository**.
		Built-in: No property data stored centrally.
		Repository: Select the Repository file where properties are stored. The fields that come after are pre-filled in using fetched data.
		Click this icon to open a WSDL schema wizard and store your WSDL connection in the **Repository** tree view. For more information about setting up and storing database connection parameters, see *Setting up a WSDL schema* of **Talend Open Studio** **User Guide**.
 *Perl only field*	*Encoding type*	Select the encoding type from the list or select **Custom** and define the type manually. This field is obligatory to manipulate data in a database.
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the Repository. Click **Edit Schema** to make changes to the schema. Note that if you make changes, the schema automatically becomes built-in. Click **Sync columns** to retrieve the schema from the previous component in the Job.
		Built-in: You create the schema and store it locally for the relevant component. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: You have already created the schema and stored it in the Repository. You can reuse it in various projects and job flowcharts. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
 *Perl only field*	*End Point URI*	Resource identifier of the Web service.
	WSDL	Description of Web service bindings and configuration.

Java only field	*Need authentication / Username and Password*	Select this check box and enter a username and a password in the corresponding fields if this is necessary to access the service.
Java only field	*Use http proxy*	Select this check box if you are using a proxy server and fill in the necessary information.
Java only field	*Trust server with SSL*	Select this check box to validate the server certificate to the client via an SSL protocol and fill in the corresponding fields: **TrustStore file**: enter the path (including filename) to the certificate TrustStore file that contains the list of certificates that the client trusts. **TrustStore password**: enter the password used to check the integrity of the TrustStore data.
Java only field	*Time out (second)*	Set a value in seconds for Web service connection time out.
	Method Name	Enter the exact name of the Method to be invoked. The Method name MUST match the corresponding method described in the Web Service. The Method name is also case-sensitive.
	Parameters	Enter the parameters expected and the sought values to be returned. Make sure that the parameters entered fully match the names and the case of the parameters described in the method.
Advanced settings	*Advanced Use* *Java only field*	Select this check box to display the fields dedicated for the advanced use of **tWebServiceInput**: **WSDL2java**: click the three-dot button to generate **Talend** routines that hold the Java code necessary to connect and query the Web service. **Code**: replace the generated model Java code with the code necessary to connect and query the specified Web service using the code in the generated **Talend** routines. **Match Brackets**: select the number of brackets to be used to close the `for` loop based on the number of open brackets.
	tStatCatcher Statistics	Select this check box to gather the job processing metadata at a job level as well as at each component level.
Usage	This component is generally used as a Start component. It requires to be linked to an output component.	
Limitation	n/a	

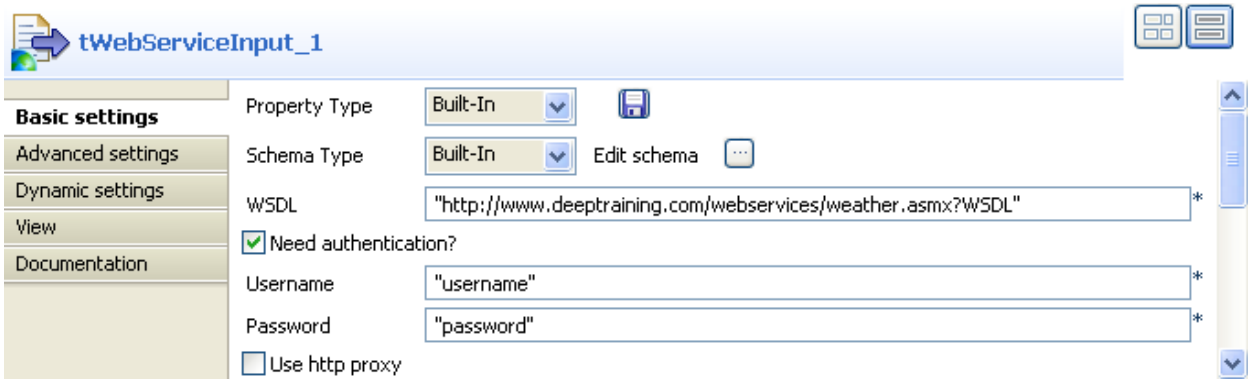
Scenario 1: Extracting images through a Web service

This scenario describes a two-component Job aiming at using a Web service method and displaying the output on the **Run** console view.

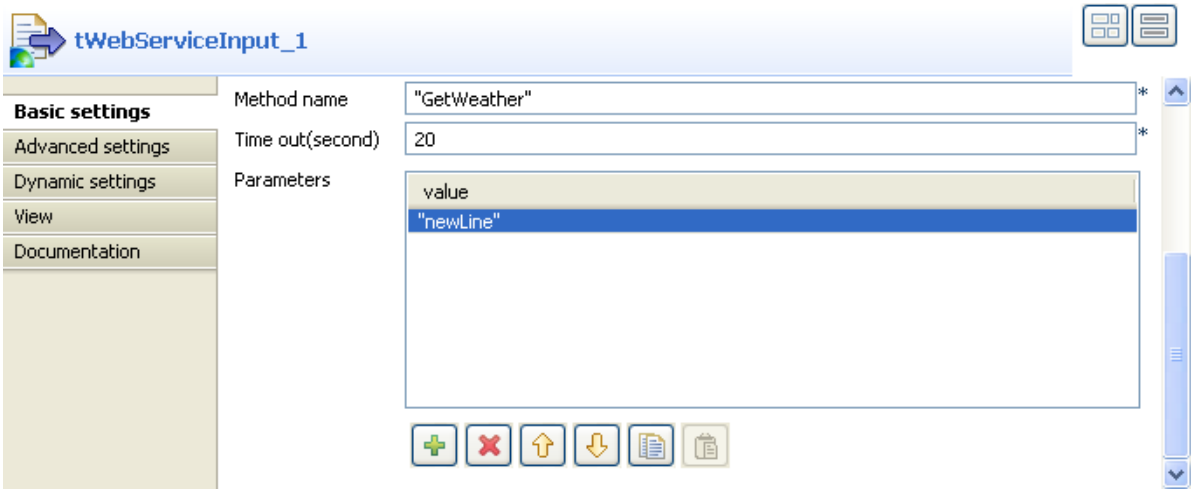
The method takes a full url as an input string and returns a string array of images from a given Web page.



- Drop a **tWebServiceInput** component and a **tLogRow** component from the **Palette** to the design workspace.
- On the **Component** view of the **tWebServiceInput** component, define the WSDL specifications, such as **End Point URI**, **WSDL** and **SOAPAction URI** where required.
- If the Web service you invoke requires authentication details, select the **Need authentication** check box and provide the relevant authentication information.



- If you are using a proxy server, select the **Use http proxy** check box and fill in the necessary connection information.
- In the **Method Name** field, type in the method name as defined in the Web Service description. The name and the case of the method entered must match exactly the corresponding Web service method.



- In the **Parameters** area, click the plus (+) button to add a line to the table.
- In the **Value** column, type in the URL of the Website, the images are to be extracted from.
- Link the **tWebServiceInput** component to the standard output component, **tLogRow**.
- Then save your Job and press **F6** to execute it.

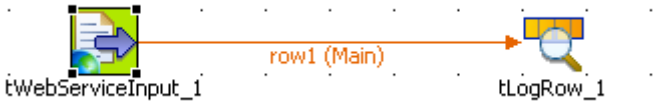
```
Starting job WebService at 16:30 06/06/2007.
<img src="http://us.a1.yimg.com/us.yimg.com/i/ww/beta/y3.gif"></img>
<img src="http://us.i1.yimg.com/us.yimg.com/i/ww/tbl/webs.gif"></img>
<img src="http://us.i1.yimg.com/us.yimg.com/i/ww/tbl/fdbg.gif"></img>
<img src="http://us.i1.yimg.com/us.yimg.com/i/ww/tbl/fdbg.gif"></img>
<img src="http://us.i1.yimg.com/us.yimg.com/i/ww/tbl/fdbg.gif"></img>
<img src="http://us.i1.yimg.com/us.yimg.com/i/ww/tbl/fdbg.gif"></img>
<img src="http://us.i1.yimg.com/us.yimg.com/i/ww/tbl/fdbg.gif"></img>
<img src="http://us.i1.yimg.com/us.yimg.com/i/ww/tbl/fdbg.gif"></img>
<img src="http://us.i1.yimg.com/us.yimg.com/i/ww/tbl/allys.gif"></img>
Job WebService ended at 16:30 06/06/2007. [exit code=0]
```

All images extracted from the given Web site are returned as a list of URLs on the **Run** view.

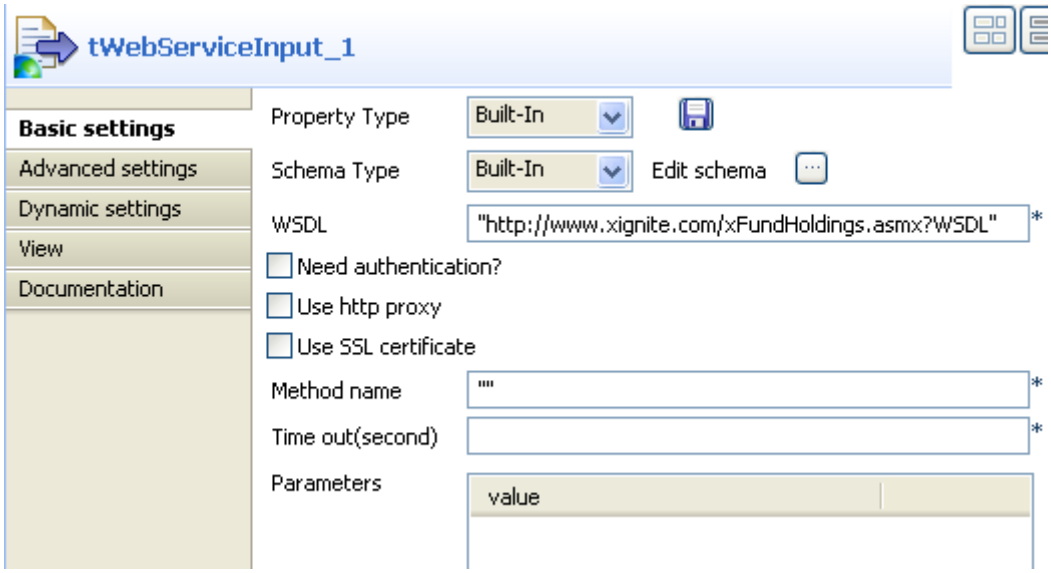
Scenario 2: Reading the data published on a Web service using the tWebServiceInput advanced features (Java only)

This Java scenario describes a two-component Job that aims at fetching a list of funds published by a financial Web service (distributed by www.xignite.com) and displaying the output on the standard console (the **Run** view).

This scenario is designed for advanced users with Java basics. Since the aim of this Job is to fetch complex hierarchical data, you need to code in Java the necessary functions.

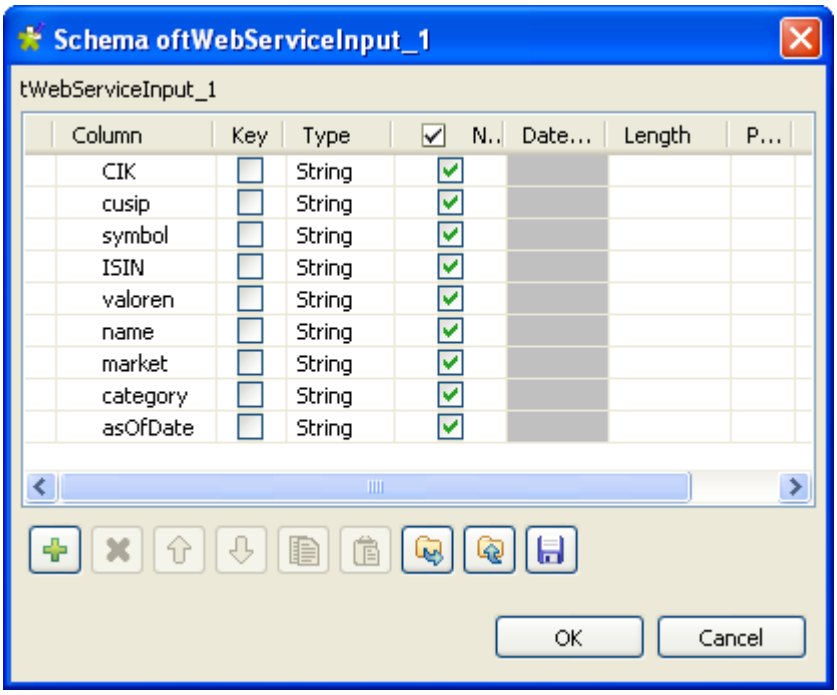


- Drop the following components from the **Palette** onto the design workspace: **tWebServiceInput** and **tLogRow**.
- Connect the two components together using the **Row Main** link.
- Double-click **tWebServiceInput** to show the **Component** view and set the component properties:

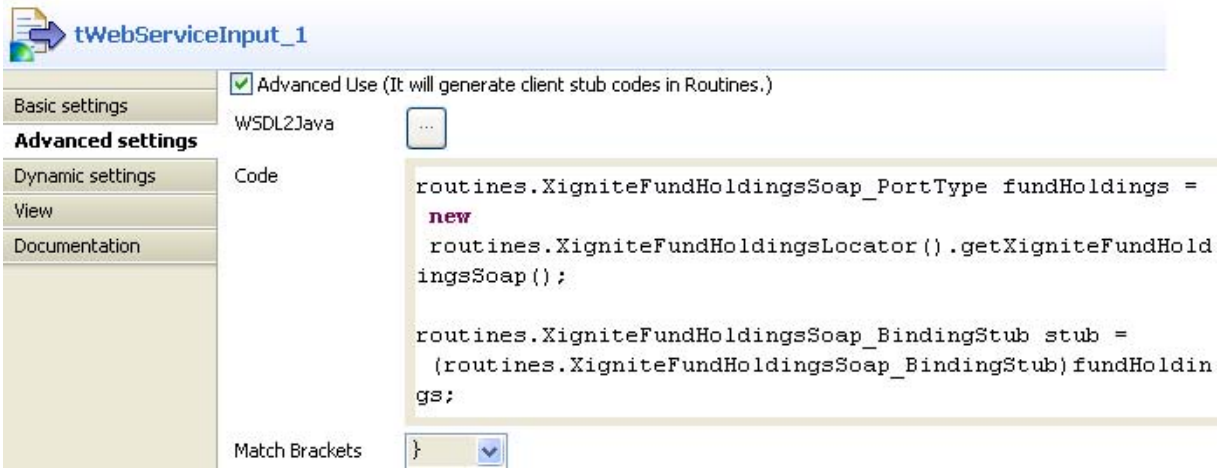


In the **Basic settings** view:

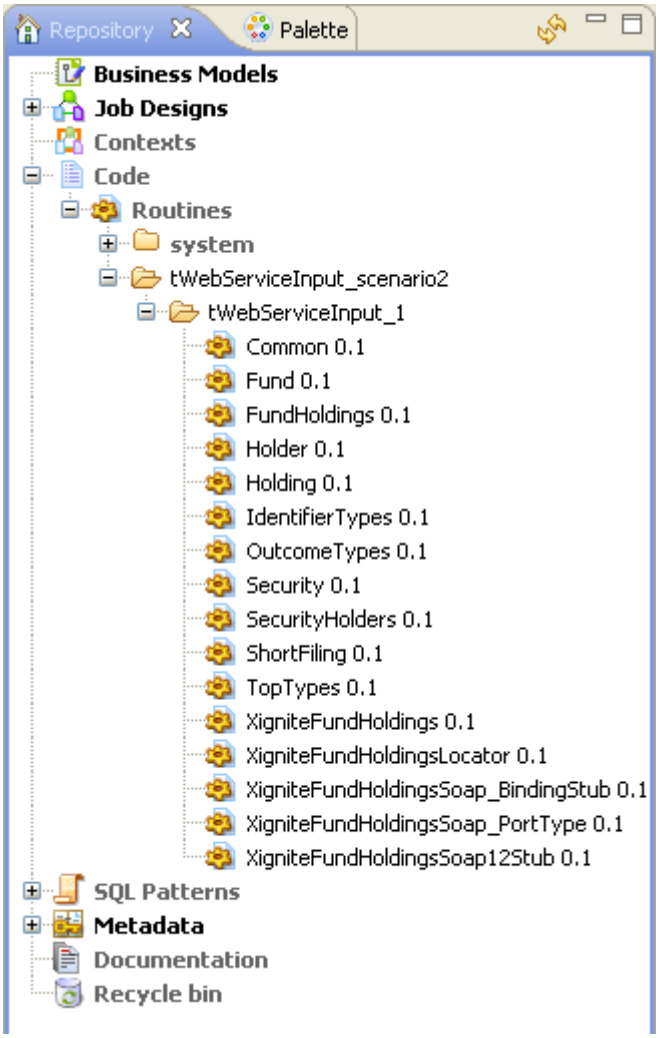
- In the **Property Type** list, select the option **Built-in** and fill in manually the fields that follow.
- In the **Schema Type** list, select **Built-in** and click the three-dot button to configure manually the data structure (schema) as shown by the figure below:



- In the **WSDL** field, type in the URL where to get the WSDL.
- In the **Time out** field, type in the duration in seconds of the connection to the Web Service.
- Click the **Advanced settings** tab to display the corresponding view where you can set the **tWebServiceInput** advanced features:



- Select the check box next to **Advanced Use** to display the advanced configuration fields.
- Click the three-dot [...] button next to the field **WSDL2Java** in order to generate automatically a number of routines from the WSDL information provided.



The generated routines display automatically under **Code** > **Routines** in the **Repository** tree view. These routines can thus easily be called in the code to build the function required to fetch complex hierarchical data from the Web Service.

- Type in the relevant function in the **Code** field. By default two code examples are provided in the Code field. The first example returns one piece of data, and the second example returns several data.
- In this scenario, several data are to be returned. Therefore, remove the first example of code and use the second example of code to build the function.
- Replace the pieces of code provided as examples with the relevant routines that have been automatically generated from the WSDL.
- Change `TalendJob_PortType` to the routine name ending with **_Port_Type**, such as: `XigniteFundHoldingsSoap_PortType`.
- Replace the various instances of `TalendJob` with a more relevant name such as the name of the method in use. In this use case: *fundHolding*
- Replace `TalendJobServiceLocator` with the name of the routine ending with **Locator**, such as: `XigniteFundHoldingLocator`.

- Replace both instances of `TalendJobSoapBindingStub` with the routine name ending with **BindingStub**, such as: `XigniteFundHoldingsSoap_BindingStub`.
- Within the brackets corresponding to the pieces of code: stub.setUsername and stub.setPassword, type in respectively your username and password between quotes. For the sake of confidentiality or maintenance, you can store your username and password in context variables.
- The list of funds provided by the *Xignite* Web service is identified using so-called "*symbols*", which are of string type. In this example, we intend to fetch the list of funds whom symbol is between "I" and "J". To do so, define the following statements: `string startSymbol="I"` and `string endSymbol="J"`.
- Then enter the piece of code to create the result table showing the list of funds (**listFunds**) of funds holdings using the statements defined earlier on: `routines.Fund[] result = fundHoldings.listFunds(startSymbol, endSymbol);`
- Loop on the fund list to fetch the funds ranging from "I" to "J": `for(int i = 0; i < result.length; i++) {`.
- Define the results to return, for example: fetch the **CIK** data from the **Security** schema using the code getSecurity().getCIK(), then pass them on to the **CIK** output schema.

The function that operates the Web service should read as follows:

```
routines.XigniteFundHoldingsSoap_PortType fundHoldings = new
routines.XigniteFundHoldingsLocator().getXigniteFundHoldingsSoap(
);

routines.XigniteFundHoldingsSoap_BindingStub stub =
(routines.XigniteFundHoldingsSoap_BindingStub)fundHoldings;

stub.setUsername("identifiant");
Stub.setPassword("mot de passe");

String startSymbol="I";
String endSymbol="J";

routines.Fund[ ] result = fundHoldings.listFunds(startSymbol,
endSymbol); for(int i = 0; i < result.length; i++) {

output_row.CIK = (result[i]).getSecurity().getCIK();
output_row.cusip = (result[i]).getSecurity().getCusip();
output_row.symbol = (result[i]).getSecurity().getSymbol();
output_row.ISIN = (result[i]).getSecurity().getISIN();
output_row.valoren = (result[i]).getSecurity().getValoren();
output_row.name = (result[i]).getSecurity().getName();
output_row.market = (result[i]).getSecurity().getMarket();
output_row.category =
(result[i]).getSecurity().getCategoryOrIndustry();
output_row.asOfDate = (result[i]).getAsOfDate();
```

The outputs defined in the Java function `output_row.output` should match exactly the columns defined in the component schema. Indeed, the case need to match in order for the data to be fetched.

- In the **Match Brackets** field, select the number of brackets to use to end the For loop, based on the number of open brackets. For this scenario, select one bracket only as only one bracket has been opened in the function.
- Double-click the component **tLogRow** to display the **Component** view and set its parameters.
- Click the three-dot [...] button next to the field **Edit Schema**, in order to check that the preceding component schema was properly propagated to the output component. If needed, click the **Sync Columns** button to retrieve the schema.
- Save your Job and press **F6** to run it.

```
Starting job tWebServiceInput_scenario2 at 13:59 27/07/2009.

[statistics] connecting to socket on port 3922
[statistics] connected

|893957753|IAAAX|||Ta Idex Asset Allocation Growth |FUNDS|Large
Blend|7/27/2009
|893957746|IAABX|||Ta Idex Asset Allocation Growth |FUNDS|Large
Blend|7/27/2009
|893957720|IAALX|||Ta Idex Asset Allocation Growth |FUNDS|Large
Blend|7/27/2009
0000895430|44980Q724|IACAX|||ING International Capital Appre
|FUNDS|Foreign Large Growth|7/27/2009
N/A|893958454|IACBX|||Ta Idex Salomon All Cap Class B |FUNDS|Large
Blend|7/27/2009
N/A|00141T122|IACFX|||Aim China Institutional Class
|FUNDS|Pacific/Asia ex-Japan Stk|7/27/2009
```

The funds comprised between “I” and “J” are returned and displayed in the console of **Talend Open Studio**.

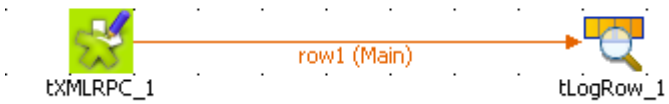
tXMLRPC

tXMLRPC Properties

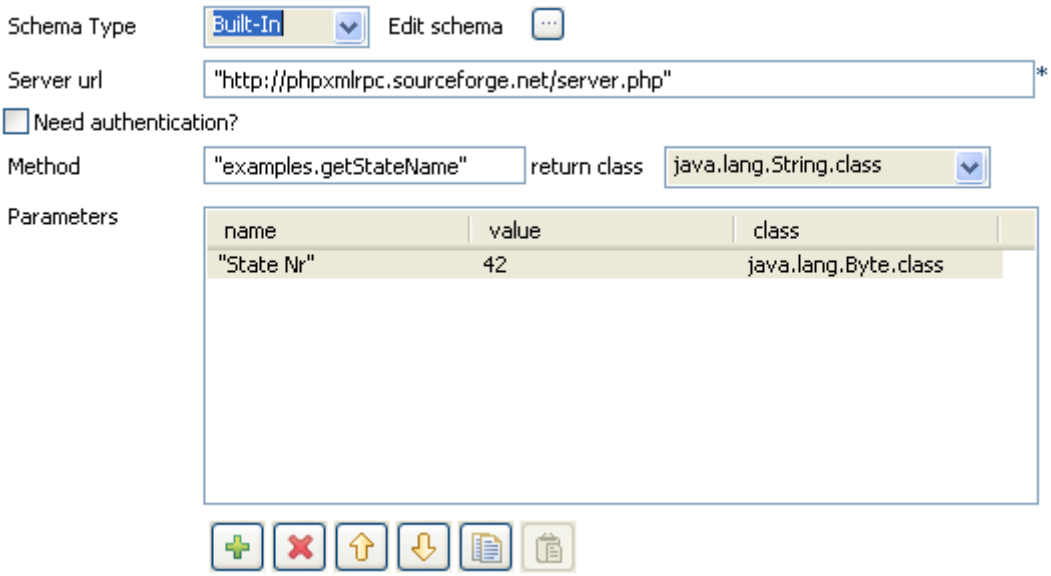
Component family	Internet	
Function	Calls the defined method from the invoked RPC service, and returns the class as defined, based on the given parameters.	
Purpose	Invokes a Method through a Web service and for the described purpose	
Basic settings	*Schema type* and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the Repository. Click **Edit Schema** to make changes to the schema. Note that if you make changes, the schema automatically becomes built-in. Click **Sync columns** to retrieve the schema from the previous component connected in the Job. In the RPC context, the schema corresponds to the output parameters. If two parameters are meant to be returned, then the schema should contain two columns.
	Server URL	URL of the RPC service to be accessed
	Need authentication / *Username and Password*	Select this check box and fill in a username and password if required to access the service.
	Method Name	Enter the exact name of the Method to be invoked. The Method name MUST match the corresponding method described in the RPC Service. The Method name is also case-sensitive.
	Return class	Select the type of data to be returned by the method. Make sure it fully matches the one defined in the method.
	Parameters	Enter the parameters expected by the method as input parameters.
Usage	This component is generally used as a Start component. It requires to be linked to an output component.	
Limitation	n/a	

Scenario: Guessing the State name from an XMLRPC

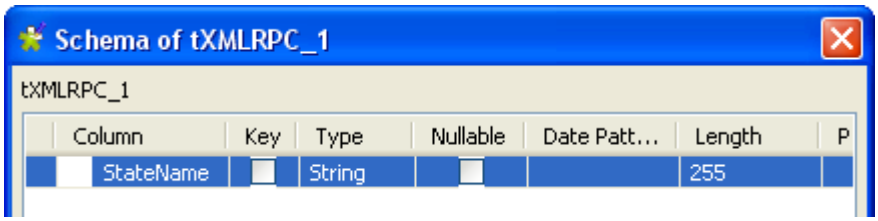
This scenario describes a two-component Job aiming at using a RPC method and displaying the output on the console view.



- Drop the **tXMLRPC** and a **tLogRow** components from the **Palette** to the design workspac.
- Set the **tXMLRPC** basic settings.

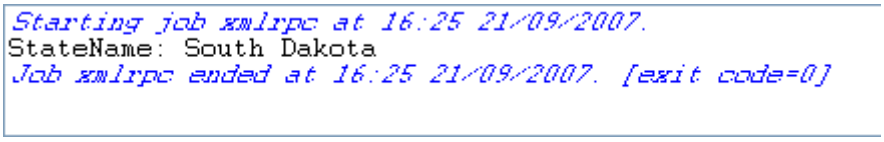


- Define the **Schema type** as **Built-in** for this use case.
- Set a single-column schema as the expected output for the called method is only one parameter: *StateName*.



- Then set the **Server url**. For this demo, use: *http://phpxmlrpc.sourceforge.net/server.php*
- No authentication details are required in this use case.
- The **Method** to be called is: *examples.getStateName*
- The **return class** is not compulsory for this method but might be strictly required for another. Leave the default setting for this use case.
- Then set the input **Parameters** required by the method called. The **Name** field is not used in the code but the value should follow the syntax expected by the method. In this example, the Name used is *State Nr* and the value randomly chosen is *42*.
- The class has not much impact using this demo method but could have with another method, so leave the default setting.

- On the **tLogRow** component **Component** view, check the box: **Print schema column name in front of each value**.
- Then save the Job and press **F6** to execute it.



South Dakota is the state name found using the `GetStateName` RPC method and corresponds the 42nd State of the United States as defined as input parameter.

Logs & Errors components

This chapter details the major components that you can find in **Logs & Errors** group of the **Palette** of **Talend Open Studio**.

The Logs & Errors family groups the component dedicated to log information catching and job error handling.

tChronometerStart

tChronometerStart Properties

Component family	Log/Error	
Function	Starts measuring the time a subjob takes to be executed.	
Purpose	Operates as a chronometer device that starts calculating the processing time of one or more subjobs in the main Job, or that starts calculating the processing time of part of your subjob.	
Usage	You can use **tChronometerStart** as a start or middle component. It can precede one or more processing tasks in the subjob. It can precede one or more subjobs in the main Job.	
Limitation	n/a	

Related scenario

For related scenario, see *Scenario: Measuring the processing time of a subjob and part of a subjob on page 611*.

tChronometerStop

tChronometerStop Properties

Component family	Log/Error	
Function	Measures the time a subjob takes to be executed.	
Purpose	Operates as a chronometer device that stops calculating the processing time of one or more subjobs in the main Job, or that stops calculating the processing time of part of your subjob. **tChronometerStop** displays: the total execution time, number of runs, number of rows processed per second, and the average, minimum and maximum processing time of a row.	
Basic settings	*Since options*	Select either check box to select measurement starting point: **Since the beginning**: stops time measurement launched at the beginning of a subjob. **Since a tChronometerStart**: stops time measurement launched at one of the **tChronometerStart** components used on the data flow of the subjob.
	Display duration in console	When selected, it displays subjob execution information on the console.
	Display component name	When selected, it displays the name of the component on the console.
	Caption	Enter desired text, to identify your subjob for example.
	Display human readable duration	When selected, it displays subjob execution information in readable time unites.
Usage	Cannot be used as a start component.	
Limitation	n/a	

Scenario: Measuring the processing time of a subjob and part of a subjob

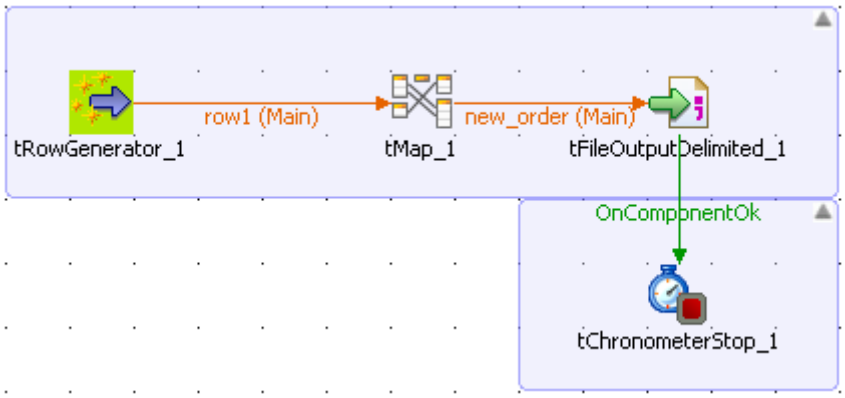
This scenario is a Perl subjob that does the following in a sequence:

- generates 1000 000 rows of first and last names,
- replaces first names with last names,
- stores the output data in a delimited file,
- measures the duration of the subjob as a whole,
- measures the duration of the name replacement operation,
- displays the gathered information about the processing time on the **Run** log console.

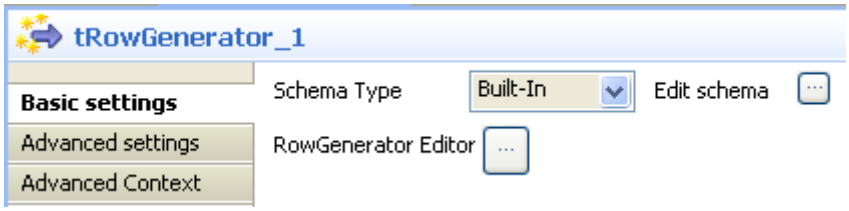
To measure the processing time of the subjob:

- Drop the following components from the **Palette** onto the design workspace: **tRowGenerator**, **tMap**, **tFileOutputDelimited**, and **tChronometerStop**.
- Connect the first three components using **Main Row** links.

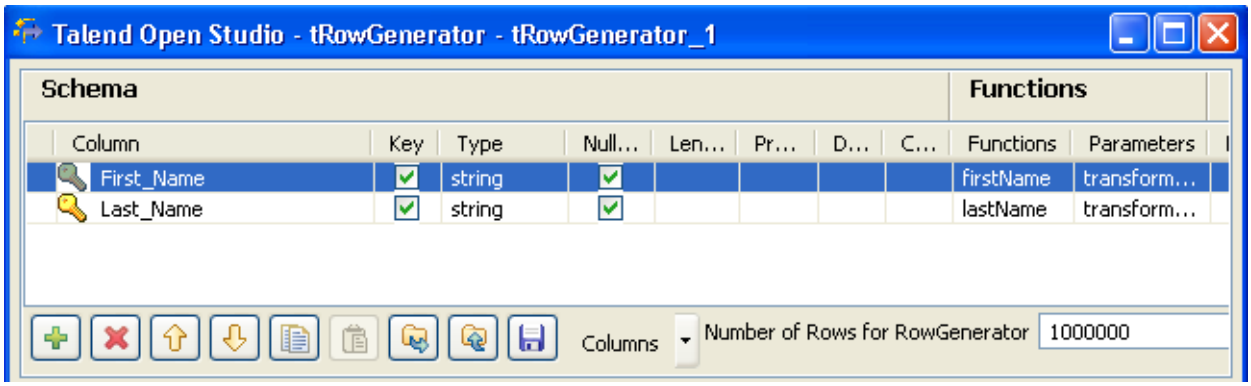
When connecting **tMap** to **tFileOutputDelimited**, you will be prompted to name the output table. The name used in this example is “new-order”.



- Connect **tFileOutputDelimited** to **tChronometerStop** using an **OnComponentOk** link.
- Select **tRowGenerator** and click the **Component** tab to display the component view.
- In the component view, click **Basic settings**.The **Component** tab opens on the **Basic settings** view by default.

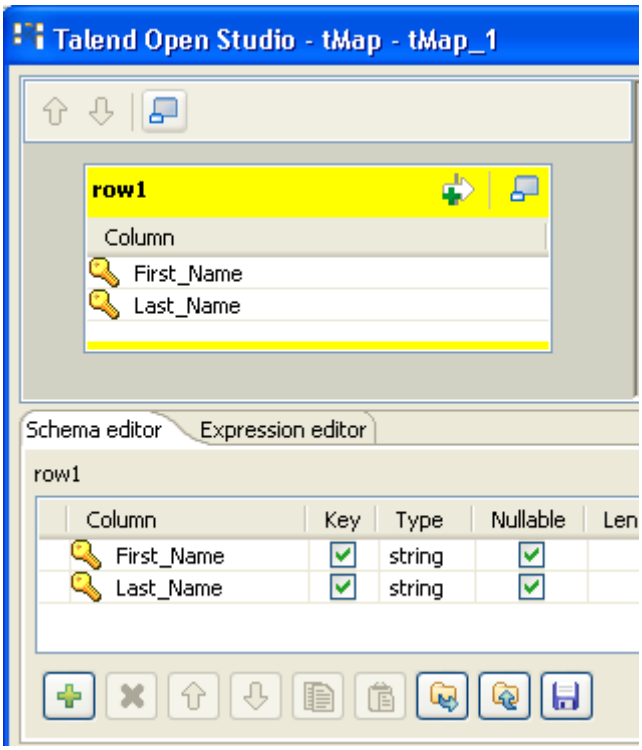


- Click **Edit schema** to define the schema of the **tRowGenerator**. For this Job, the schema is composed of two columns: *First_Name* and *Last_Name*, generated using Perl script.
- Click the **RowGenerator Editor** three-dot button to open the editor and define the data to be generated.

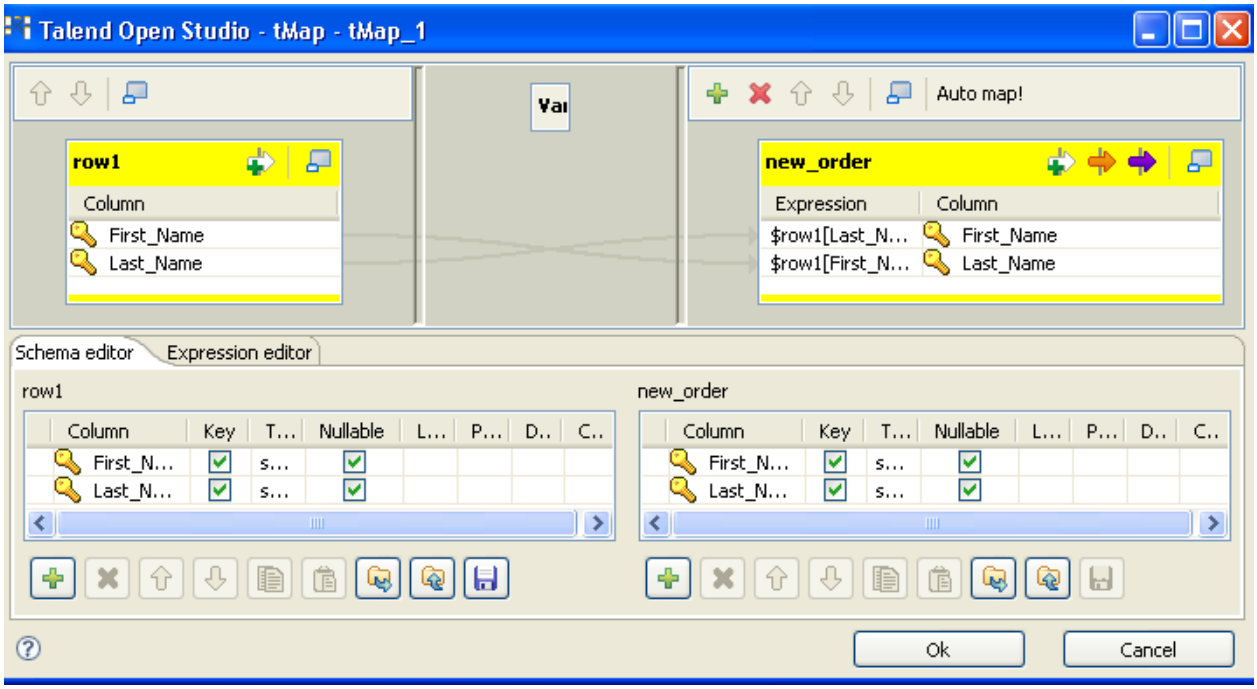


- In the **RowGenerator Editor**, specify the number of rows to be generated in the **Number of Rows for RowGenerator** field and click **OK**. The **RowGenerator Editor** closes.

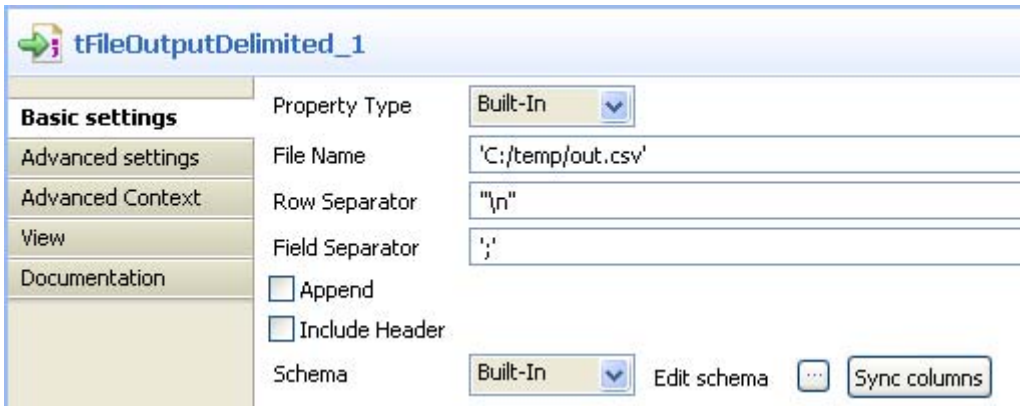
- Double-click on the **tMap** component to open the Map editor. The Map editor opens displaying the input metadata of the **tRowGenerator** component.



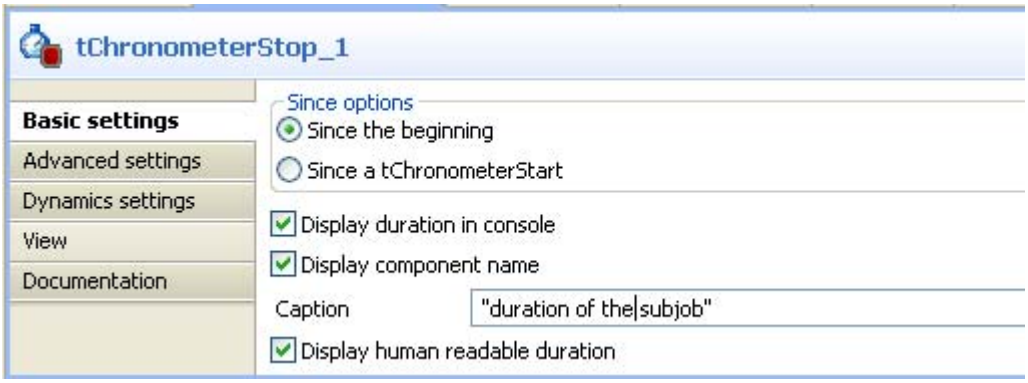
- In the **Schema editor** panel of the Map editor, click the plus button of the output table to add two rows and define them.
- In the Map editor, drag the *First_Name* row from the input table to the *Last_Name* row in the output table and drag the *Last_Name* row from the input table to the *First_Name* row in the output table.
- Click **OK** to close the editor.



- Select **tFileOutputDelimited** and click the **Component** tab to display the component view.
- In the Basic settings view, set **tFileOutputDelimited** properties as needed.



- Select **tChronometerStop** and click the **Component** tab to display the component view.
- In the **Since options** panel of the **Basic settings** view**,** select **Since the beginning check box** to measure the duration of the subjob as a whole.



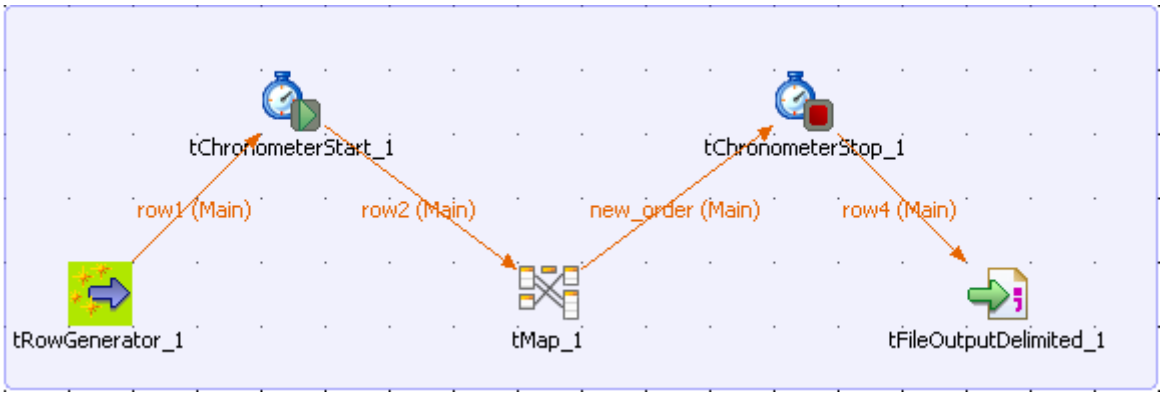
- Select/clear the other check boxes as needed. In this scenario, we want to display the subjob duration on the console preceded by the component name.
- If needed, enter a text in the **Caption** field.
- Save your Job and press **F6** to execute it.

```
Starting job test at 14:18 06/08/2008.
[tChronometerStop_1] duration of the subjob: 70093 ms (1 minute and
10 seconds), 1 run, average : 70093750 microseconds, min : 70093750
microseconds, max: 70093750 microseconds, speed: 0 rows/second
Job test ended at 14:19 06/08/2008. [exit code=0]
```

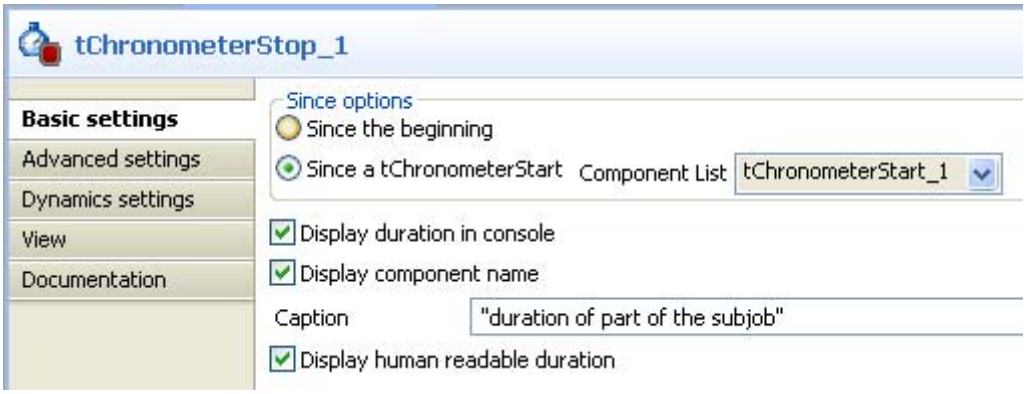
You can measure the duration of the subjob the same way by placing **tChronometerStop** below **tRowGenerator**, and connecting the latter to **tChronometerStop** using an **OnSubjobOk** link.

To measure only part of the subjob, the duration of the replacement task processed by **tMap** in this scenario:

- Drop **tChronometerStart** from the **Palette** to the design workspace.
- Connect **tRowGenerator**, **tChronometerStart**, and **tMap** using **Main Row** links via a right-click on each component



- Place **tChronometerStop** between **tMap** and **tFileOutputDelimited**.
- Connect **tMap** to **tChronometerStop** using the **Main Row** link *new_order*, and **tChronometerStop** to **tFileOutputDelimited** using a **Main Row** link.
- In the **Basic settings** of the **tChronometerStop** component, select the **Since a tChronometerStart** check box and select **tChronometerStart_1** from the component list.



- Select/clear the other check boxes as needed.
- If needed, enter a text in the **Caption** field.
- Press **F6** to execute the Job and display its result

```
Starting job chrono at 14:43 06/08/2008.
[tChronometerStop_1] duration of part of a subjob 10742 ms (10
seconds), 1000000 runs, average : 10 microseconds, min : 0
microseconds, max: 31253 microseconds, speed: 93092 rows/second
Job chrono ended at 14:44 06/08/2008. [exit code=0]
```

Such usage of **tChronometerStart** and **tChronometerStop** can display the performance rate of desired components in your subjob.

tDie

tDie properties

Both **tDie** and **tWarn** components are closely related to the **tLogCatcher** component.They generally make sense when used alongside a tLogCatcher in order for the log data collected to be encapsulated and passed on to the output defined.

Component family	Log & Error	
Function	Kills the current Job. Generally used with a **tCatch** for log purpose.	
Purpose	Triggers the **tLogCatcher** component for exhaustive log before killing the Job.	
Basic settings	*Die message*	Enter the message to be displayed before the Job is killed.
	Error code	Enter the error code if need be, as an integer
	Priority	Set the level of priority, as an integer
Usage	Cannot be used as a start component.	
Limitation	n/a	

Related scenarios

For uses cases in relation with **tDie**, see **tLogCatcher** scenarios:

- *Scenario1: warning & log on entries on page 624*
- *Scenario 2: log & kill a Job on page 626*

tFlowMeter

tFlowMeter Properties

Component family	Log/Error	
Function	Counts the number of rows processed in the defined flow.	
Purpose	The number of rows is then meant to be caught by the **tFlowMeterCatcher** for logging purpose.	
Basic settings	*Use input connection name as label*	Select this check box to reuse the name given to the input main row flow as label in the logged data.
	Mode	Select the type of values for the data measured: **Absolute**: the actual number of rows is logged **Relative**: a ratio (%) of the number of rows is logged. When selecting this option, the reference
	Thresholds	Adds a threshold to watch proportions in volumes measured. you can decide that the normal flow has to be between low and top end of a row number range, and if the flow is under this low end, there is a bottleneck.
Usage	Cannot be used as a start component as it requires an input flow to operate.	
Limitation	n/a	

If you have a need of log, statistics and other measurement of your data flows*Automating statistics & logs use* in **Talend Open Studio** User Guide.

Related scenario

For related scenario, see *Scenario: Catching flow metrics from a Job on page 620.*

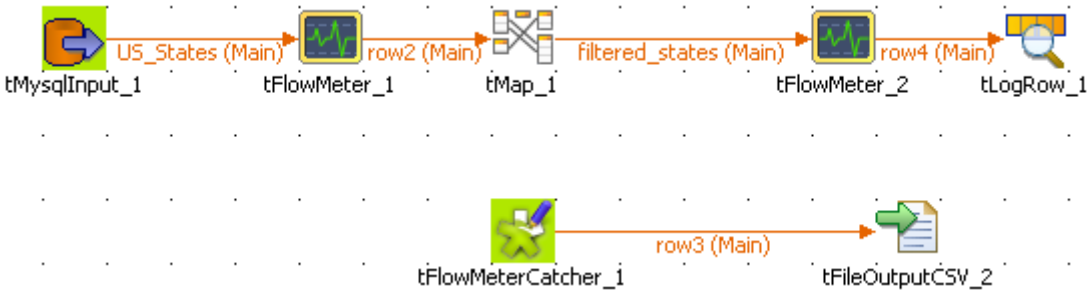
tFlowMeterCatcher

tFlowMeterCatcher Properties

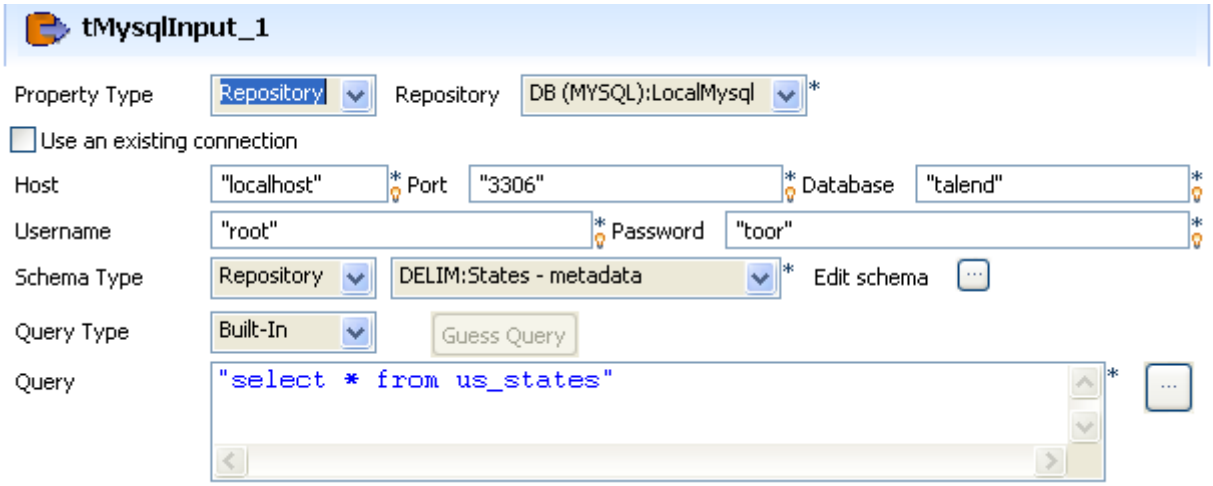
Component family	Log & Error	
Function	Based on a defined sch.ema, the **tFlowMeterCatcher** catches the processing volumetric from the **tFlowMeter** component and passes them on to the output component.	
Purpose	Operates as a log function triggered by the use of a **tFlowMeter** component in the Job.	
Basic settings	*Schema type*	A schema is a row description, i.e., it defines the fields to be processed and passed on to the next component. In this particular case, the schema is read-only, as this component gathers standard log information including:
		Moment: Processing time and date
		Pid: Process ID
		Father_pid: Process ID of the father Job if applicable. If not applicable, Pid is duplicated.
		Root_pid: Process ID of the root Job if applicable. If not applicable, pid of current Job is duplicated.
		System_pid: Process id generated by the system
		Project: Project name, the Job belongs to.
		Job: Name of the current Job
		Job_repository_id: ID generated by the application.
		Job_version: Version number of the current Job
		Context: Name of the current context
		Origin: Name of the component if any
		Label: Label of the row connection preceding the **tFlowMeter** component in the Job, and that will be analyzed for volumetrics.
		Count: Actual number of rows being processed
		Reference: Name of the reference row as defined in the **tFlowMeter** component for relative counting mode.
		Thresholds: Only used when the relative mode is selected in the **tFlowMeter** component.
Usage	This component is the start component of a secondary Job which triggers automatically at the end of the main Job.	
Limitation	The use of this component cannot be separated from the use of the **tFlowMeter**. For more information, see *tFlowMeter on page 618*.	

Scenario: Catching flow metrics from a Job

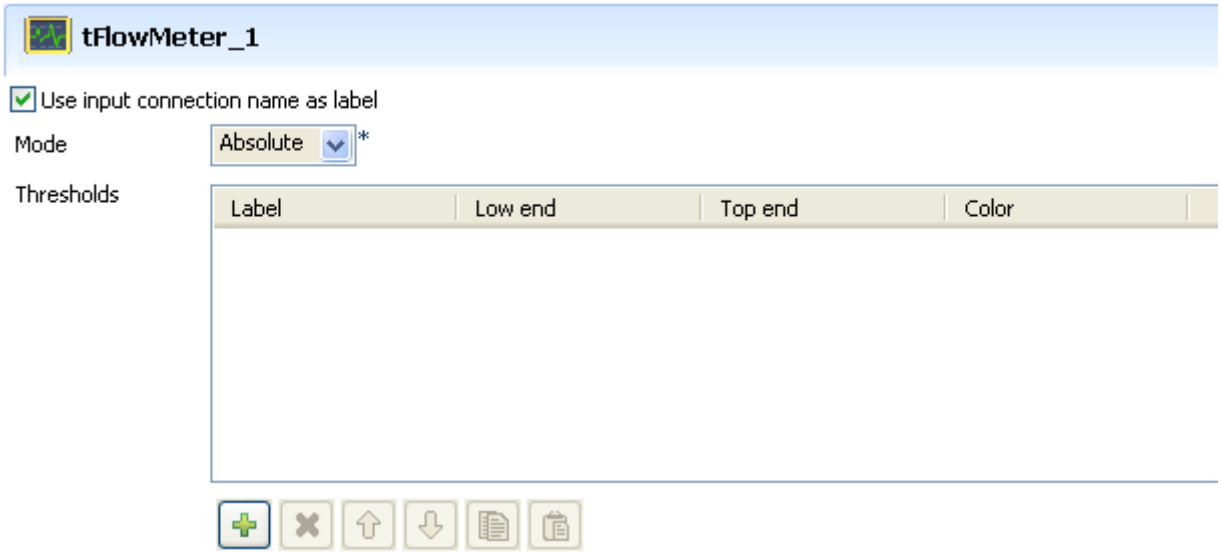
The following basic Job aims at catching the number of rows being passed in the flow processed. The measures are taken twice, once after the input component, that is, before the filtering step and once right after the filtering step, that is, before the output component.



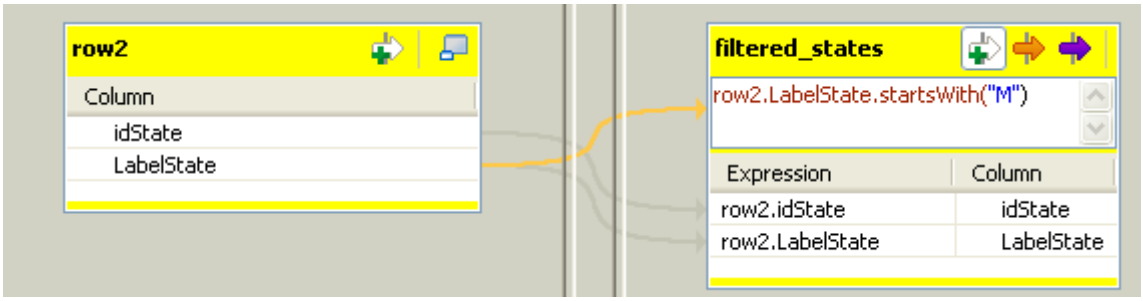
- Drop the following components from the **Palette** to the design workspace: **tMysqlInput**, **tFlowMeter** (x2), **tMap**, **tLogRow**, **tFlowMeterCatcher** and **tFileOutputCSV**.
- Link components using row main connections and click on the label to give consistent name throughout the Job, such as *US_States* from the input component and *filtered_states* for the output from the **tMap** component, for example.
- Link the **tFlowMeterCatcher** to the **tFileOutputCSV** component using a row main link also as data is passed.
- On the **tMysqlInput** Component view, configure the connection properties as **Repository**, if the table metadata are stored in the Repository. Or else, set the Type as **Built-in** and configure manually the connection and schema details if they are built-in for this Job.



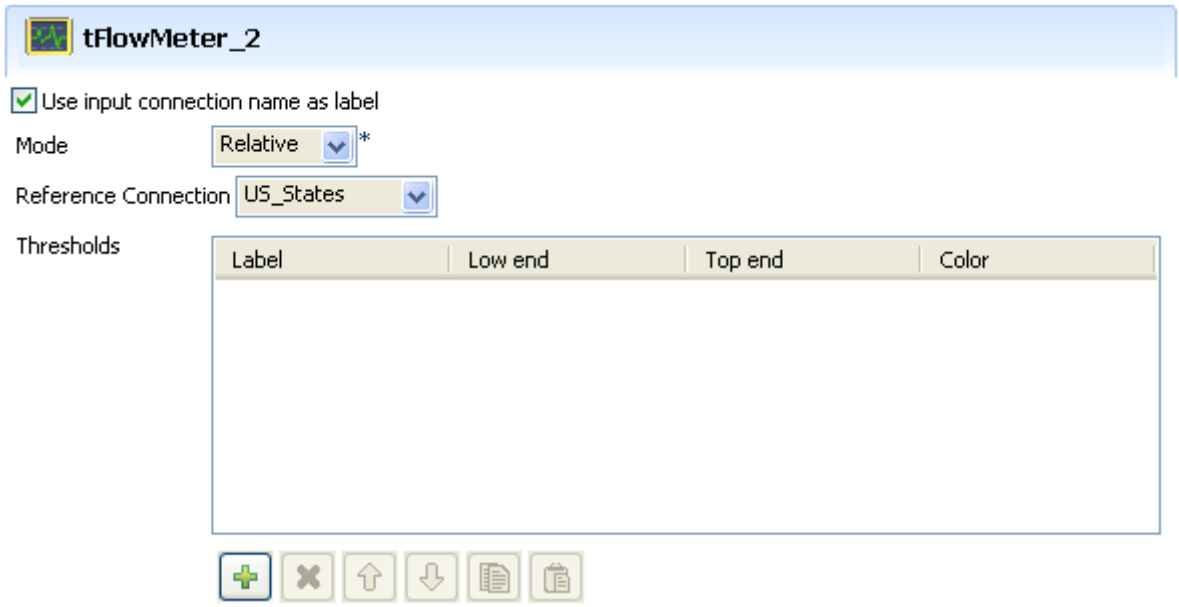
- The **Schema** is simply made of two columns: *idState* and *LabelState*.
- The **Query type** is Built-in for this Job example.
- The 50 States of the USA are recorded in the table *us_states*. In order for all 50 entries of the table to get selected, the query to run onto the Mysql database is as follows:
  `select * from us_states`.
- Select the relevant **Encoding type** in the list.
- Then select the following component which is a **tFlowMeter** and set its properties.



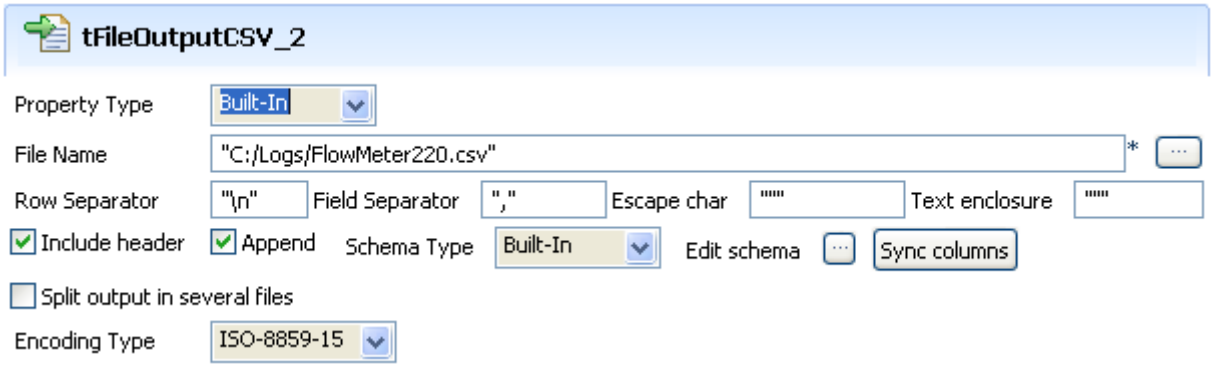
- Select the check box **Use input connection name as label**, in order to reuse the label you chose in the log output file (**tFileOutputCSV**).
- The mode is **Absolute** as there is no reference flow to meter against, also no **Threshold** is to be set for this example.
- Then launch the **tMap** editor to set the filtering properties.
- For this use case, drag and drop the ID and States columns from the Input area of the tMap towards the Output area. No variable is used in this example.



- On the Output flow area (labelled *filtered_states* in this example), click the arrow & plus button to activate the expression filter field.
- Drag the *LabelState* column from the Input area (*row2*) towards the expression filter field and type in the rest of the expression in order to filter the state labels starting with the letter *M*. The final expression looks like: `row2.LabelState.startsWith("M")`
- Click **OK** to validate the setting.
- Then select the second **tFlowMeter** component and set its properties.



- Select the check box **Use input connection name as label**.
- Select **Relative** as **Mode** and in the **Reference connection** list, select *US_States* as reference to be measured against.
- Once again, no threshold is used for this use case.
- No particular setting is required in the **tLogRow**.
- Neither does the **tFlowMeterCatcher** as this component's properties are limited to a preset schema which includes typical log information.
- So eventually set the log output component (**tFileOutputCSV**).



- Select the **Append** check box in order to log all **tFlowMeter** measures.
- Then save your Job and press **F6** to execute it.

```
Starting job FlowMeterCatcher at 17:56 29/08/2007.
19|Maine
20|Maryland
21|Massachusetts
22|Michigan
23|Minnesota
24|Mississippi
25|Missouri
26|Montana
Job FlowMeterCatcher ended at 17:56 29/08/2007. [exit code=0]
```

The **Run** view shows the filtered state labels as defined in the Job.

	K	L	M	N	O
	origin	label	count	reference	thresholds
1	tFlowMeter_1	US_States	50		
1	tFlowMeter_2	filtered_states	8	50	

In the delimited csv file, the number of rows shown in column **count** varies between **tFlowMeter1** and **tFlowMeter2** as the filtering has then been carried out. The **reference** column shows also this difference.

tLogCatcher

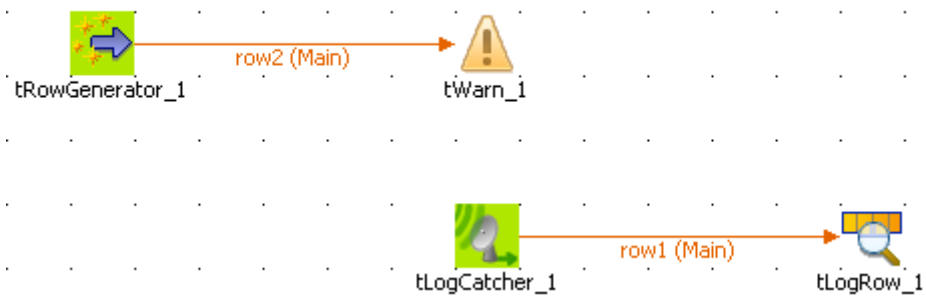
tLogCatcher properties

Both **tDie** and **tWarn** components are closely related to the **tLogCatcher** component.They generally make sense when used alongside a **tLogCatcher** in order for the log data collected to be encapsulated and passed on to the output defined.

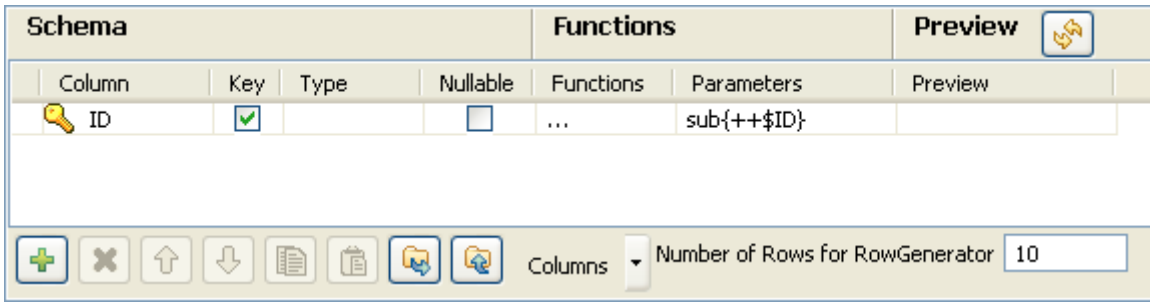
Component family	Log & Error	
Function	Fetches set fields and messages from Java Exception/PerlDie, **tDie** and/or **tWarn** and passes them on to the next component.	
Purpose	Operates as a log function triggered by one of the three: Java exception/PerlDie, **tDie** or **tWarn**, to collect and transfer log data.	
Basic settings	*Schema type* and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the Repository.
		Built-in: The schema will be created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused in various projects and job flowcharts. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide
	Catch PerlDie *Catch Java Exception*	Select this check box to trigger the **tCatch** function when a PerlDie/Java Exception occurs in the Job
	Catch tDie	Select this check box to trigger the **tCatch** function when a **tDie** is called in a Job
	Catch tWarn	Select this check box to trigger the tCatch function when a tWarn is called in a Job
Usage	This component is the start component of a secondary Job which automatically triggers at the end of the main Job	
Limitation	n/a	

Scenario1: warning & log on entries

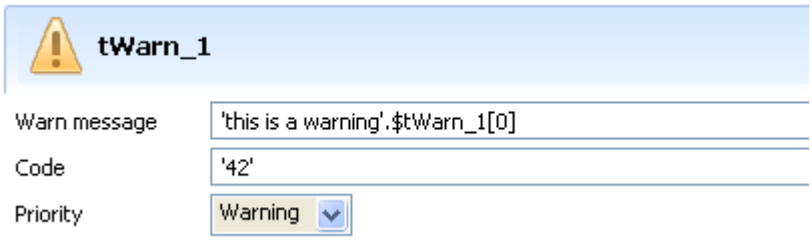
In this basic scenario made of three components, a **tRowGenerator** creates random entries (id to be incremented). The input hits a **tWarn** component which triggers the **tLogCatcher** subjob. This subjob fetches the warning message as well as standard predefined information and passes them on to the **tLogRow** for a quick display of the log data.



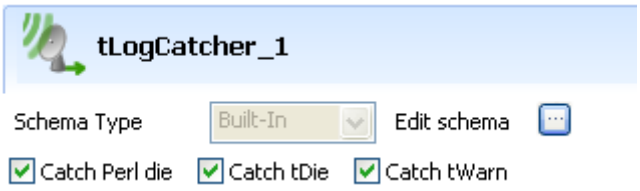
- Drop a **tRowGenerator**, a **tWarn**, a **tLogCatcher** and a **tLogRow** from the **Palette**, on your design workspace
- Connect the **tRowGenerator** to the **tWarn** component.
- Connect separately the **tLogCatcher** to the **tLogRow**.
- On the **tRowGenerator** editor, set the random entries creation using a basic Perl function:



- On the **tWarn** Component view, set your warning message, the code the priority level. In this case, the message is “this is a warning’.
- For this scenario, we will concatenate a Perl function to the message above, in order to collect the first value from the input table.



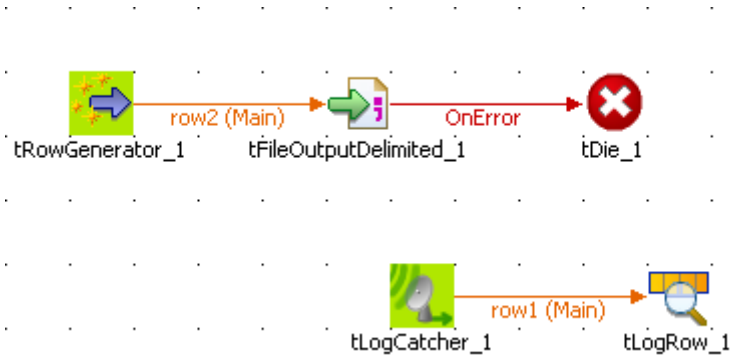
- On the **Basic settings** view of **tLogCatcher**, select the **tWarn** check box in order for the message from the latter to be collected by the subjob.
- Click **Edit Schema** to view the schema used as log output. Notice that the log is comprehensive.



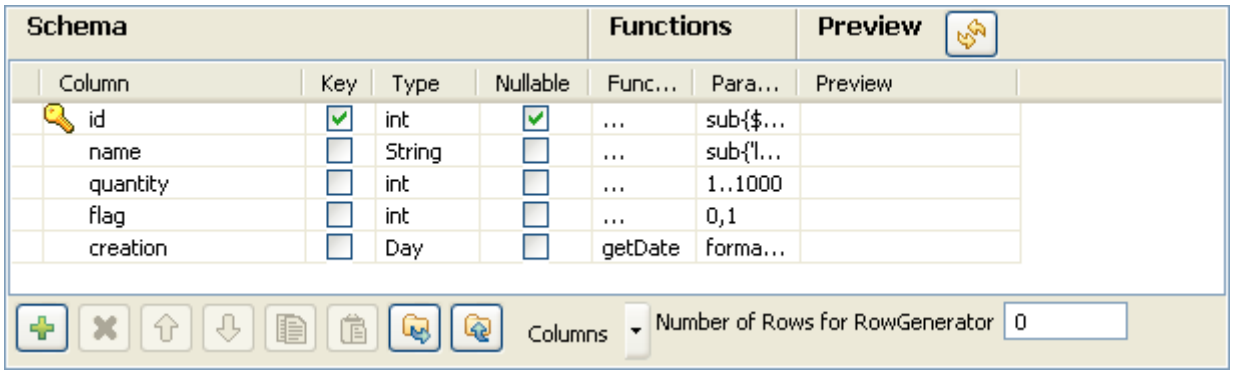
Press **F6** to execute the Job. Notice that the Log produced is exhaustive.

Scenario 2: log & kill a Job

This scenario uses a **tLogCatcher** and a **tDie** component. A **tRowGenerator** is connected to a **tFileOutputDelimited** using a Row link. On error, the **tDie** triggers the catcher subjob which displays the log data content on the **Run** console.

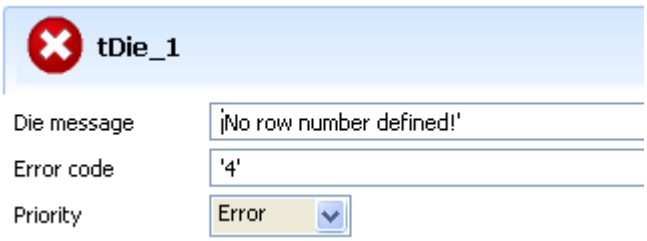


- Drop all required components from various folders of the **Palette** to the design workspace: **tRowGenerator**, **tFileOutputDelimited**, **tDie**, **tLogCatcher**, **tLogRow**.
- On the tRowGenerator Component view, define the setting of the input entries to be handled.

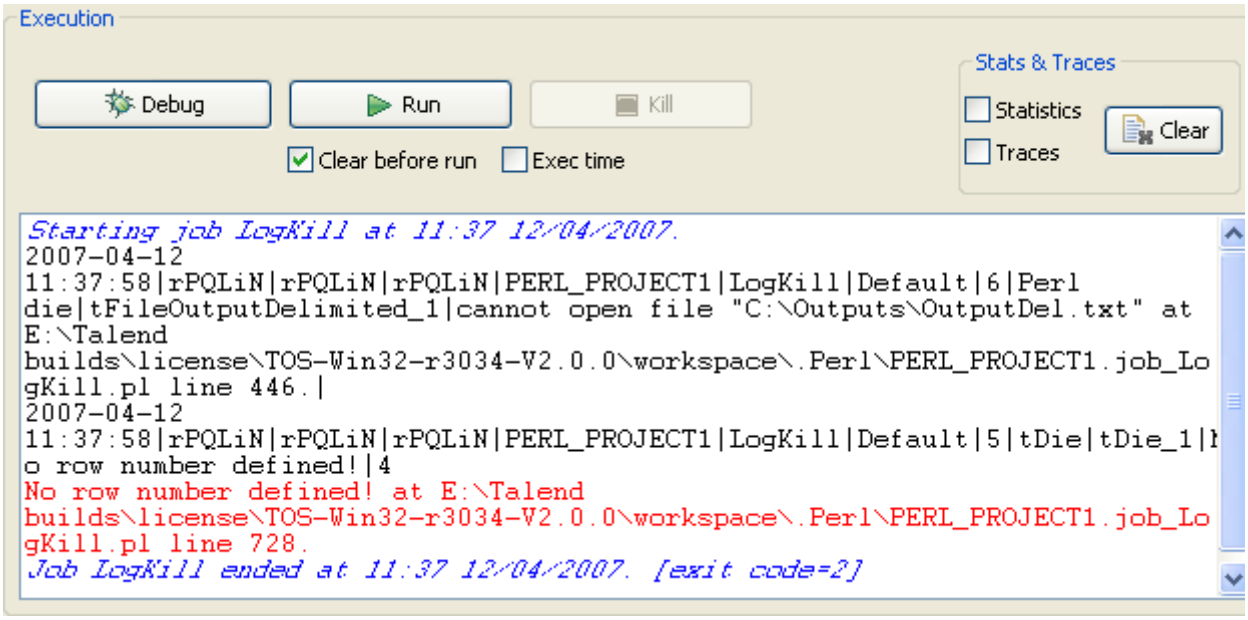


- Edit the schema and define the following columns as random input examples: *id, name, quantity, flag and creation*.
- Set the **Number of rows** onto 0. This will constitute the error which the Die operation is based on.
- On the **Values** table, define the **Perl array** functions to feed the input flow.
- Define the **tFileOutputDelimited** to hold the possible output data. The row connection from the tRowGenerator feeds automatically the output schema. The separator is a simple semi-colon.

- Connect this output component to the tDie using a **Trigger > If** connection. Double-click on the newly created connection to define the if:
  `$_globals{tRowGenerator_1}{NB_LINE} <= 0`
- Then double-click to select and define the **Basic settings** of the **tDie** component.



- Enter your **Die** message to be transmitted to the **tLogCatcher** before the actual kill-job operation happens.
- Next to the Job but not physically connected to it, drop a **tLogCatcher** from the **Palette** to the design workspace and connect it to a **tLogRow** component.
- Define the tLogCatcher Basic settings. Make sure the **tDie** box is selected in order to add the Die message to the Log information transmitted to the final component.



- Press **F6** to run the Job and notice that the log contains a black message and a red one.
- The black log data come from the **tDie** and are transmitted by the **tLogCatcher**. In addition the normal PerlDie message in red displays as a Job abnormally died.

tLogRow

tLogRow properties

Component family	Log & Error	
Function	Displays data or results in the **Run** console	
Purpose	**tLogRow** helps monitoring data processed.	
Basic settings	*Print values in table cells*	The output flow displays in table cells.
	Separator	Enter the separator which will delimit data on the Log display
	Print component unique name in front of each output row	Select this check box in case several LogRow components are used. Allows to differentiate outputs
	Print schema column name in front of each value	Select this check box to retrieve column labels from output schema.
	Use fixed length for values	Select this check box to set a fixed width for the value display.
Usage	This component can be used as intermediate step in a data flow or as a n end object in the job flowchart.	
Limitation	n/a	

Scenario: Delimited file content display

Related topics using a **tLogRow** component:

- **tFileInputDelimited** *Scenario: Delimited file content display on page 495*.
- **tContextLoad** *Scenario: Dynamic context use in MySQL DB insert on page 652*
- **tWarn, tDie, tLogCatcher** *Scenario1: warning & log on entries on page 624* and *Scenario 2: log & kill a Job on page 626*

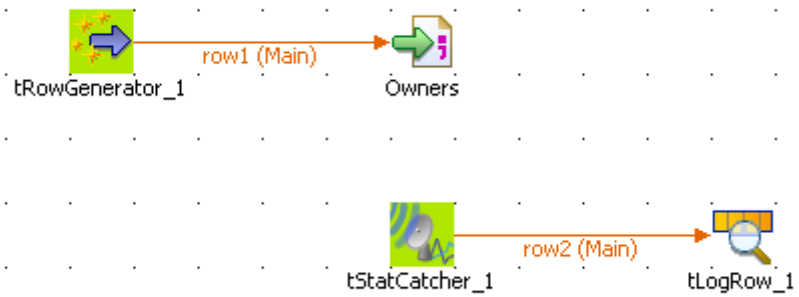
tStatCatcher

tStatCatcher Properties

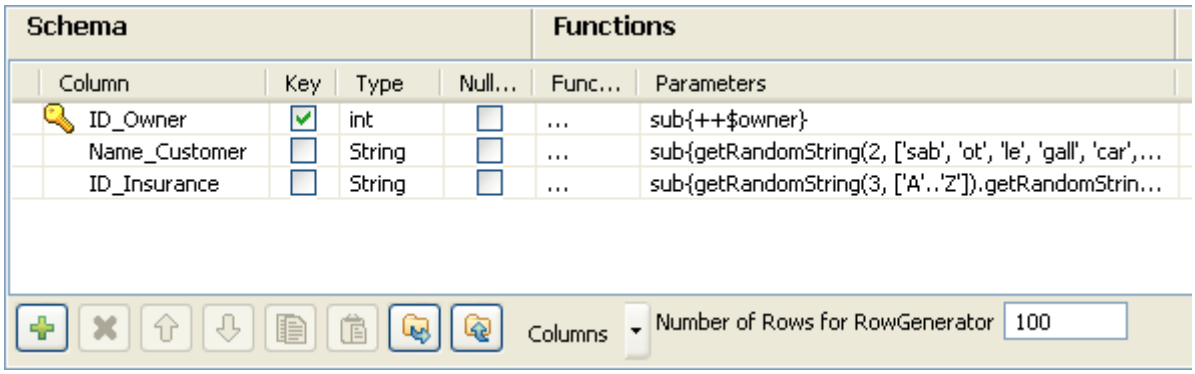
Component family	Log & Error	
Function	Based on a defined sch.ema, gathers the job processing metadata at a job level as well as at each component level.	
Purpose	Operates as a log function triggered by the **StatsCatcher Statistics** check box of individual components, and collects and transfers this log data to the output defined.	
Basic settings	*Schema type*	A schema is a row description, i.e., it defines the fields to be processed and passed on to the next component. In this particular case, the schema is read-only, as this component gathers standard log information including:
		Moment: Processing time and date
		Pid: Process ID
		Father_pid: Process ID of the father Job if applicable. If not applicable, Pid is duplicated.
		Root-pid: Process ID of the root Job if applicable. If not applicable, pid of current Job is duplicated.
		Project: Project name, the Job belongs to.
		Job: Name of the current Job
		Context: Name of the current context
		Origin: Name of the component if any
		Message: Begin or End.
Usage	This component is the start component of a secondary Job which triggers automatically at the end of the main Job. The processing time is also displayed at the end of the log.	
Limitation	n/a	

Scenario: Displaying job stats log

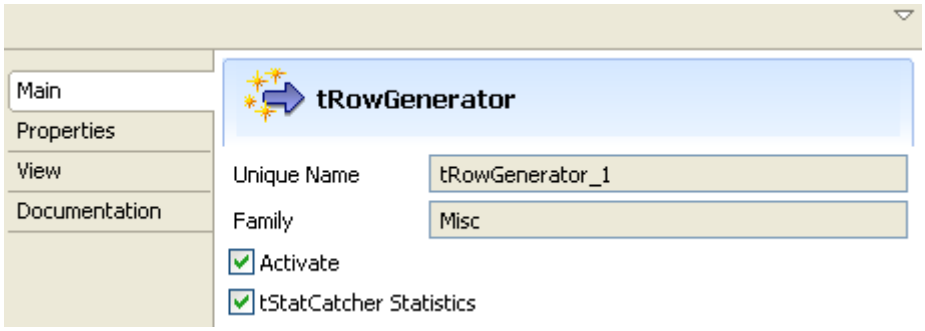
This scenario describes a four-component Job, aiming at displaying on the **Run** console the statistics log fetched from the file generation through the tStatCatcher component.



- Drop the required components: **tRowGenerator**, **tFileOutputDelimited**, **tStatCatcher** and **tLogRow** from the **Palette** to the design workspace.
- In the **Basic settings** panel of **tRowGenerator**, define the data to be generated. For this Job, the schema is composed of three columns: *ID_Owners*, *Name_Customer* and *ID_Insurance*, generated using Perl script.

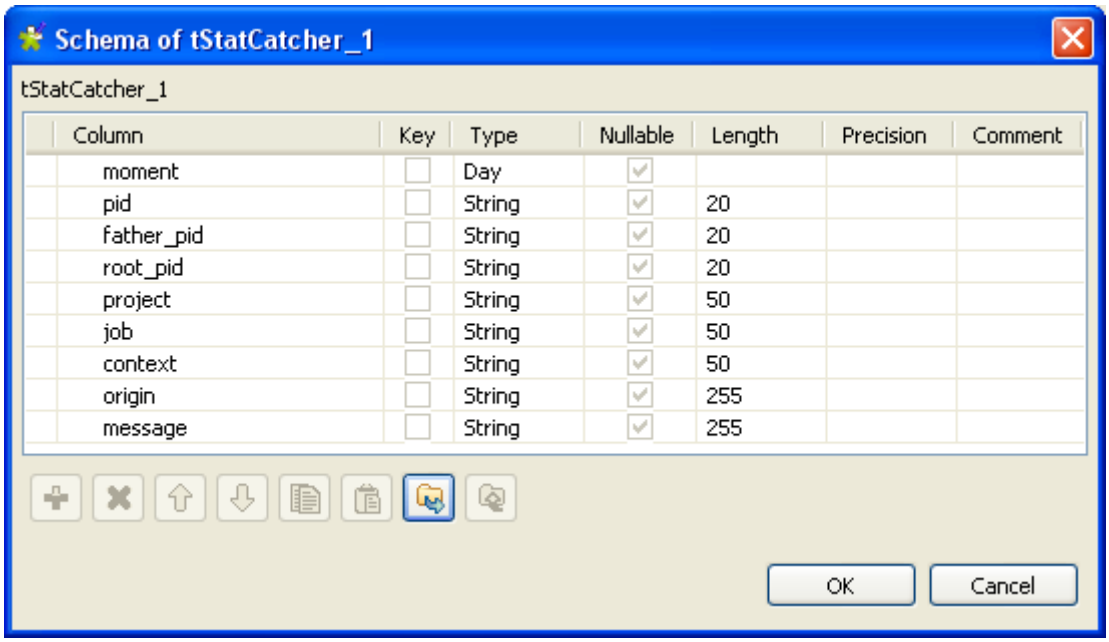


- The number of rows can be restricted to 100.
- Click on the **Main** tab of the Component view.



- And select the **tStatCatcher Statistics** check box to enable the statistics fetching operation.
- Then, define the output component's properties. In the **tFileOutputDelimited** Component view, browse to the output file or enter a name for the output file to be created. Define the delimiters, such as semi-colon, and the encoding.
- Click on **Edit schema** and make sure the schema is recollected from the input schema. If need be, click on **Sync Columns**.
- Then click on the **Basic settings** tab of the **Component** view, and select here as well the **tStatCatcher Statistics** check box to enable the processing data gathering.

- In the secondary Job, double-click on the **tStatCatcher** component. Note that the Properties are provided for information only as the schema representing the processing data to be gathered and aggregated in statistics, is defined and read-only.



- Define then the **tLogRow** to set the delimiter to be displayed on the console.
- Eventually, press **F6** to run the Job and display the job result.

```
Starting job StatsCatch at 15:10 23/02/2007.
2007-02-23 15:10:30|3656|StatsCatch|Default||begin
2007-02-23
15:10:30|3656|StatsCatch|Default|tFileOutputDelimited_1|begin
2007-02-23 15:10:30|3656|StatsCatch|Default|tRowGenerator_1|begin
2007-02-23 15:10:30|3656|StatsCatch|Default|tRowGenerator_1|end
2007-02-23 15:10:30|3656|StatsCatch|Default|tRowGenerator_1|0.0 seconds
2007-02-23 15:10:30|3656|StatsCatch|Default|tFileOutputDelimited_1|end
2007-02-23 15:10:30|3656|StatsCatch|Default|tFileOutputDelimited_1|0.0
seconds
2007-02-23 15:10:30|3656|StatsCatch|Default||end
2007-02-23 15:10:30|3656|StatsCatch|Default||0.0 seconds
Job StatsCatch ended at 15:10 23/02/2007. [exit code=0]
```

The log shows the Begin and End information for the Job itself and for each of the component used in the Job.

tWarn

tWarn Properties

Both **tDie** and **tWarn** components are closely related to the **tLogCatcher** component.They generally make sense when used alongside a **tLogCatcher** in order for the log data collected to be encapsulated and passed on to the output defined.

Component family	Log/Error	
Function	Provides a priority-rated message to the next component	
Purpose	Triggers a warning often caught by the **tLogCatcher** component for exhaustive log.	
Basic settings	*Warn message*	Type in your warning message
	Code	Define the code level
	Priority	Enter the priority level as an integer
Usage	Cannot be used as a start component. If an output component is connected to it, an input component should be preceding it.	
Limitation	n/a	

Related scenarios

For uses cases in relation with **tWarn**, see **tLogCatcher** scenarios:

- *Scenario1: warning & log on entries on page 624*
- *Scenario 2: log & kill a Job on page 626*

Misc group components

This chapter details the major components that you can find in **Misc** group of the **Palette** of **Talend Open Studio**.

The **Misc** family gathers miscellaneous components covering needs such as creating set of dummy data rows, buffering data, loading context variables.

tAddLocationFromIP

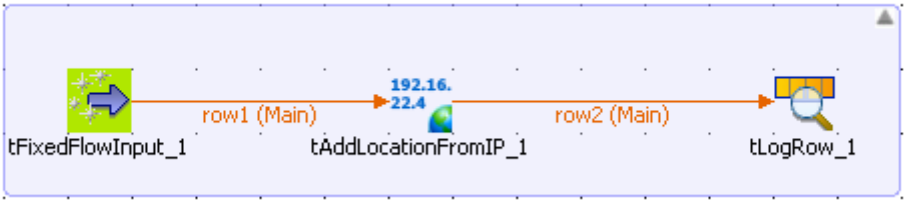
tAddLocationFromIP Properties

Component family	Misc	
Function	**tAddLocationFromIP** replaces IP addresses with geographical locations.	
Purpose	**tAddLocationFromIP** helps you to geolocate visitors through their IP addresses. It identifies visitors' geographical locations i.e. country, region, city, latitude, longitude, ZIP code...etc.using an IP address lookup database file.	
Basic settings	*Schema type* and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either built-in or remotely stored in the Repository.
		Built-in: You create and store the schema locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: Select the Repository file where Properties are stored. When selected, the fields that follow are pre-defined using fetched data.
	Database Filepath	The path to the IP address lookup database file.
	Input parameters	**Input column:** Select the input column from which the input values are to be taken.
		input value is a hostname: Check if the input column holds hostnames.
		input value is an IP address: Check if the input column holds IP addresses.
	Location type	**Country code**: Check to replace IP with country code.
		Country name: Check to replace IP with country name.
Usage	This component is an intermediary step in the data flow allowing to replace IP with geolocation information. It can not be a start component as it requires an input flow. It also requires an output component.	
Limitation	n/a	

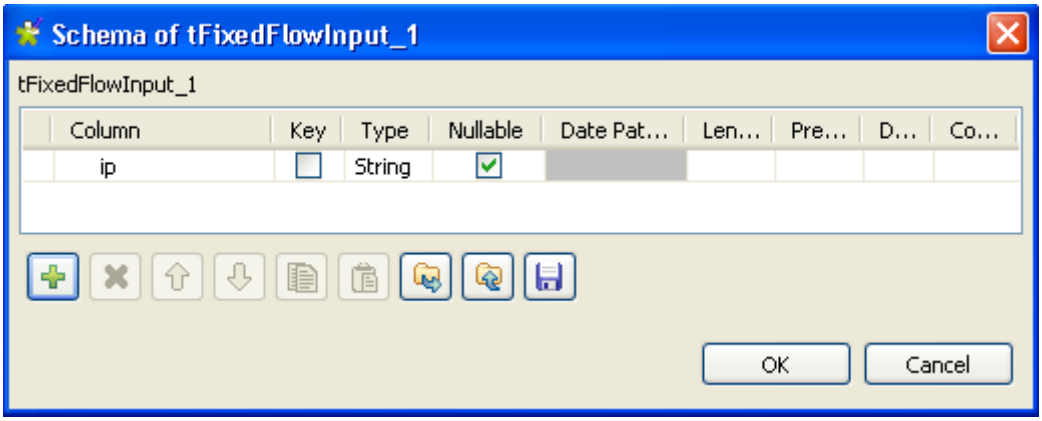
Scenario: Identifying a real-world geographic location of an IP

The following Java scenario creates a three-component Job that associates an IP with a geographical location. It obtains a site visitor's geographical location based on its IP.

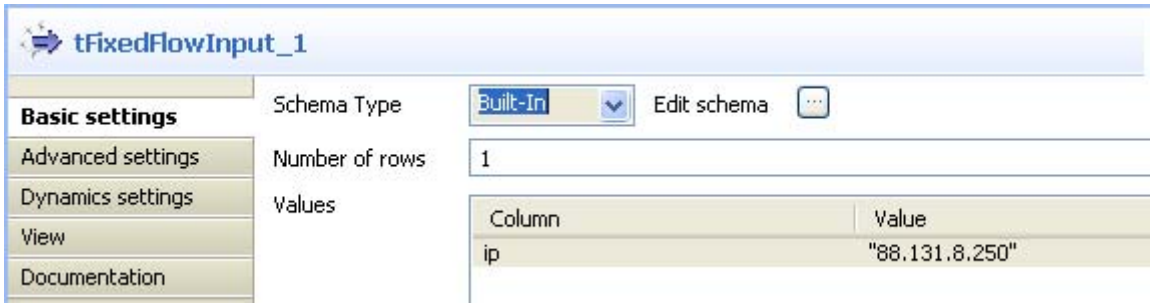
- Drop the following components from the **Palette** onto the design workspace: **tFixedFlowInput**, **tAddLocationFromIP**, and **tLogRow**.
- Connect the three components using **Row Main** links.



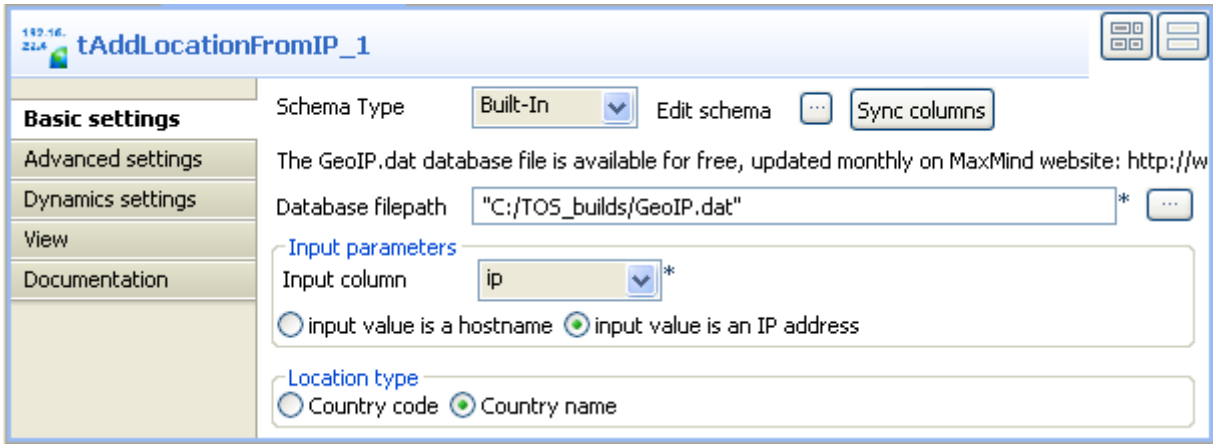
- In the design workspace, select **tFixedFlowInput**.
- Click the **Component** tab to define the basic settings for **tFixedFlowInput**.
- Set the **Schema Type** to **Built-In** and click the three-dot [...] button next to **Edit Schema** to define the data you want to use as input. In this scenario, the schema is made of one column that holds an IP address.



- Click **OK** to close the dialog box, and accept propagating the changes when prompted by the system. The defined column displays in the **Values** panel of the **Basic settings** view.
- Click in the **Value** cell and set the value for the IP address.



- In the **Number of rows** field, enter the number of rows to be generated.
- In the design workspace, select **tAddLocationFromIP**.
- Click the **Component** tab to define the basic settings for **tAddLocationFromIP**.



- Click the **Sync columns** button to synchronize the schema with the input schema set with **tFixedFlowInput**.
- Browse to the GeoIP.dat file to set its path in the **Database filepath** field.

  Make sure to download the latest version of the IP address lookup database file from the relevant site as indicated in the **Basic settings** view of **tAddLocationFromIp**.
- In the **Input parameters** panel, set your input parameters as needed. In this scenario, the input column is the *ip* column defined earlier that holds an IP address.
- In the **Location type** panel, set location type as needed. In this scenario, we want to display the country name.
- In the design workspace, select **tLogRow**.
- Click the **Component** tab and define the basic settings for **tLogRow** as needed. In this scenario, we want to display values in cells of a table.
- Save your Job and press **F6** to execute it.

```
Starting job add_location at 11:00 06/08/2008.
.------------+--------.
|      tLogRow_1       |
|=-----------+-------=|
|ip          |location|
|=-----------+-------=|
|88.131.8.250|Sweden  |
'------------+--------'

Job add_location ended at 11:00 06/08/2008. [exit code=0]
```

The only row is generated to display the country name that is associated with the set IP address.

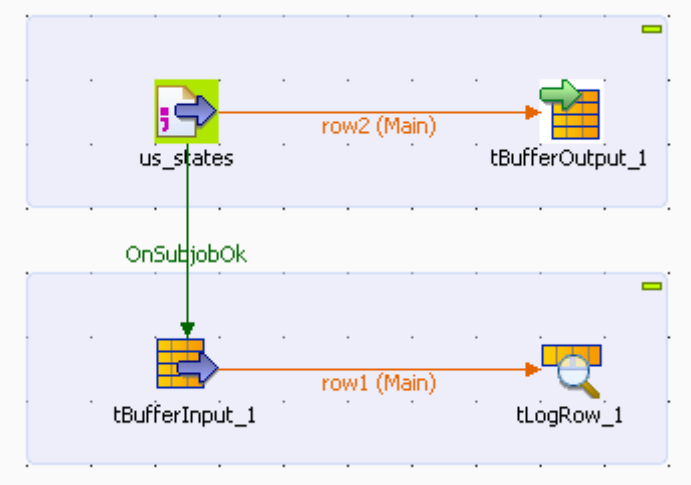
tBufferInput

tBufferInput properties

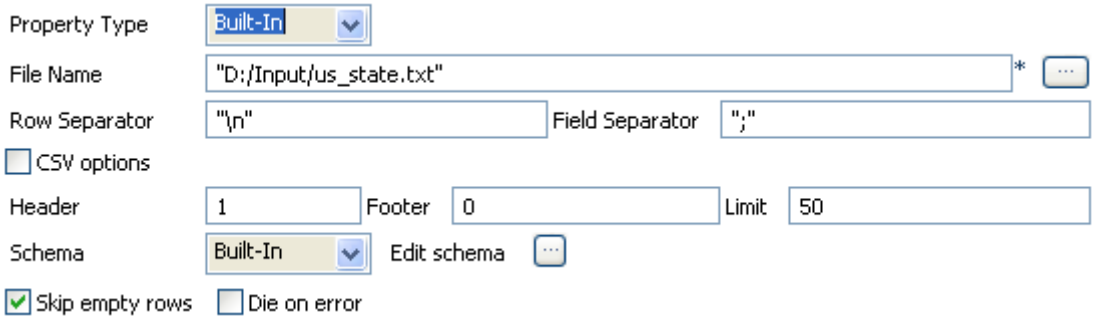
Component family	Misc	
Function	This component retrieves bufferized data in order to process it in a second subjob.	
Purpose	The **tBufferInput** component retrieves data bufferized via a **tBufferOutput** component, for example, to process it in another subjob.	
Basic settings	*Schema type* and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the Repository. In the case of **tBufferInput**, the column position is more important than the column label as this will be taken into account.
		Built-in: You create the schema and store it locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: You have already created the schema and stored it in the Repository, hence can be reused in various projects and job designs. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
Usage	This component is the start component of a secondary Job which is triggered automatically at the end of the main Job.	

Scenario: Retrieving bufferized data (Java)

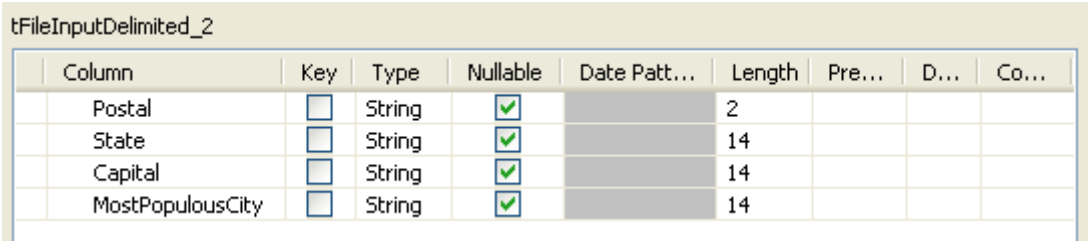
This scenario describes a Job that retrieves bufferized data from a subjob and displays it on the console.



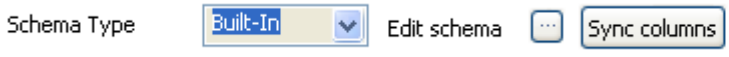
- Drop the following components from the **Palette** onto the design workspace: **tFileInputDelimited** and **tBufferOutput**.
- Select the **tFileInputDelimited** and on the **Basic Settings** tab of the **Component** view, set the access parameters to the input file.



- In the **File Name** field, browse to the delimited file holding the data to be bufferized.
- Define the **Row** and **Field separators**, as well as the **Header**.
- Click [...] next to the **Schema type** field to describe the structure of the file.

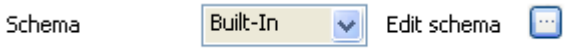


- Describe the **Schema** of the data to be passed on to the **tBufferOutput** component.
- Select the **tBufferOutput** component and set the parameters on the **Basic Settings** tab of the **Component** view.



Generally speaking, the schema is propagated from the input component and automatically fed into the **tBufferOutput** schema. But you can also set part of the schema to be bufferized if you want to.

- Drop the **tBufferInput** and **tLogRow** components from the **Palette** onto the design workspace below the subjob you just created.
- Connect **tFileInputDelimited** and **tBufferInput** via a **Trigger** > **OnSubjobOk** link and connect **tBufferInput** and **tLogRow** via a **Row** > **Main** link.
- Double-click **tBufferInput** to set its **Basic settings** in the **Component** view.
- In the **Basic settings** view, click [...] next to the **Edit Schema** field to describe the structure of the file.



- Use the schema defined for the **tFileInputDelimited** component and click **OK**.
- The schema of the **tBufferInput** component is automatically propagated to the **tLogRow**. Otherwise, double-click **tLogRow** to display the **Component** view and click **Sync column**.
- Save your Job and press **F6** to execute it.

```
Starting job BufferFatherJob at 10:41 05/02/2008.
AL|Alabama|Montgomery|Birmingham
AK|Alaska|Juneau|Anchorage
AZ|Arizona|Phoenix|Phoenix
AR|Arkansas|Little Rock|Little Rock
CA|California|Sacramento|Los Angeles
CO|Colorado|Denver|Denver
CT|Connecticut|Hartford|Bridgeport
DE|Delaware|Dover|Wilmington
FL|Florida|Tallahassee|Jacksonville
GA|Georgia|Atlanta|Atlanta
HI|Hawaii|Honolulu|Honolulu
ID|Idaho|Boise|Boise
IL|Illinois|Springfield|Chicago
IN|Indiana|Indianapolis|Indianapolis
IA|Iowa|Des Moines|Des Moines
KS|Kansas|Topeka|Wichita
KY|Kentucky|Frankfort|Louisville
```

The standard console returns the data retrieved from the buffer memory.

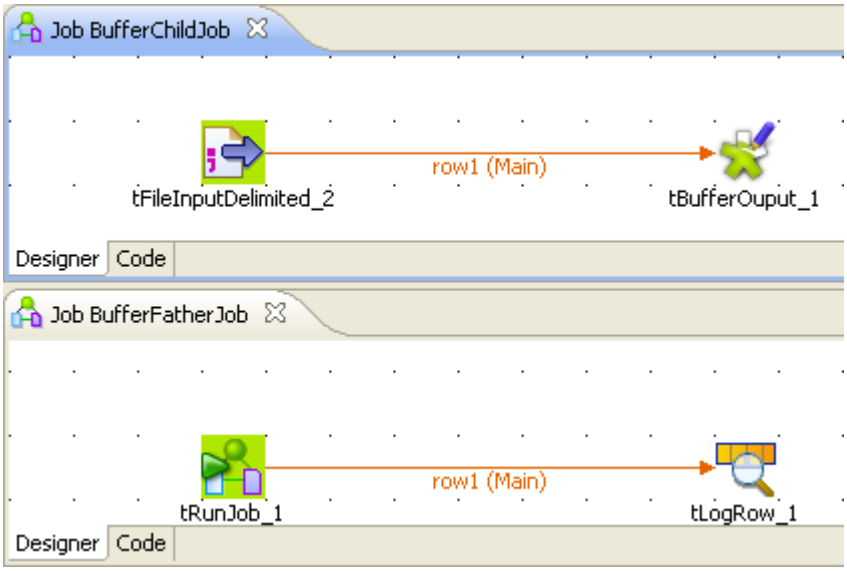
tBufferOutput

tBufferOutput properties

Component family	Misc	
Function	This component collects data in a buffer in order to access it later via webservice for example.	
Purpose	This component allows a Webservice to access data. Indeed it had been designed to be exported as Webservice in order to access data on the web application server directly. For more information, see *Exporting Jobs as Webservice* in **Talend Open Studio** User Guide.	
Basic settings	*Schema type* and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the Repository. In the case of the **tBufferOutput**, the column position is more important than the column label as this will be taken into account.
		Built-in: The schema will be created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused in various projects and job designs. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
Usage	This component is not startable (green background) and it requires an output component.	

Scenario1: Buffering data (Java)

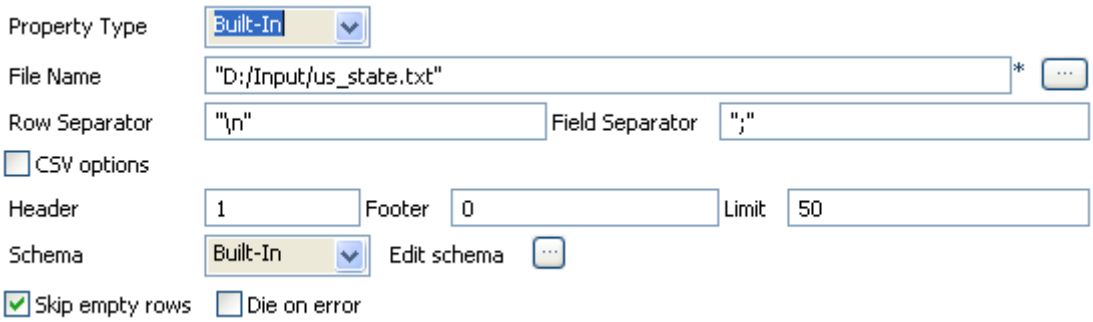
This scenario describes an intentionally basic Job that bufferizes data in a child job while a parent Job simply displays the bufferized data onto the standard output console. For an example of how to use **tBufferOutput** to access output data directly on the Web application server, see *Scenario 2: Buffering output data on the webapp server on page 643*.



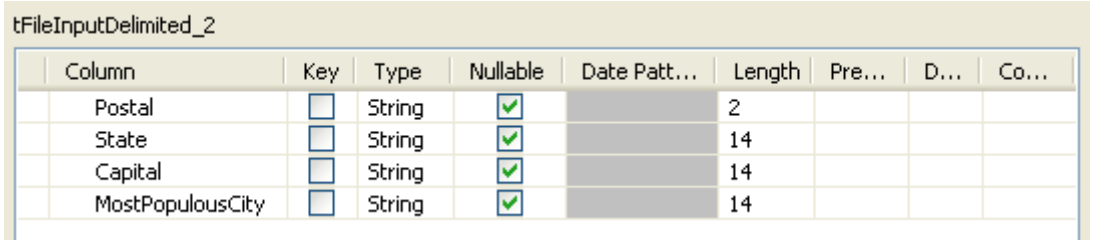
- Create two Jobs: a first Job (*BufferFatherJob*) runs the second Job and displays its content onto the **Run** console. The second Job (*BufferChildJob*) stores the defined data into a buffer memory.
- On the first Job, drop the following components: **tRunJob** and **tLogRow** from the **Palette** to the design workspace.
- On the second Job, drop the following components: **tFileInputDelimited** and **tBufferOutput** the same way.

Let's set the parameters of the second Job first:

- Select the **tFileInputDelimited** and on the **Basic Settings** tab of the **Component** view, set the access parameters to the input file.

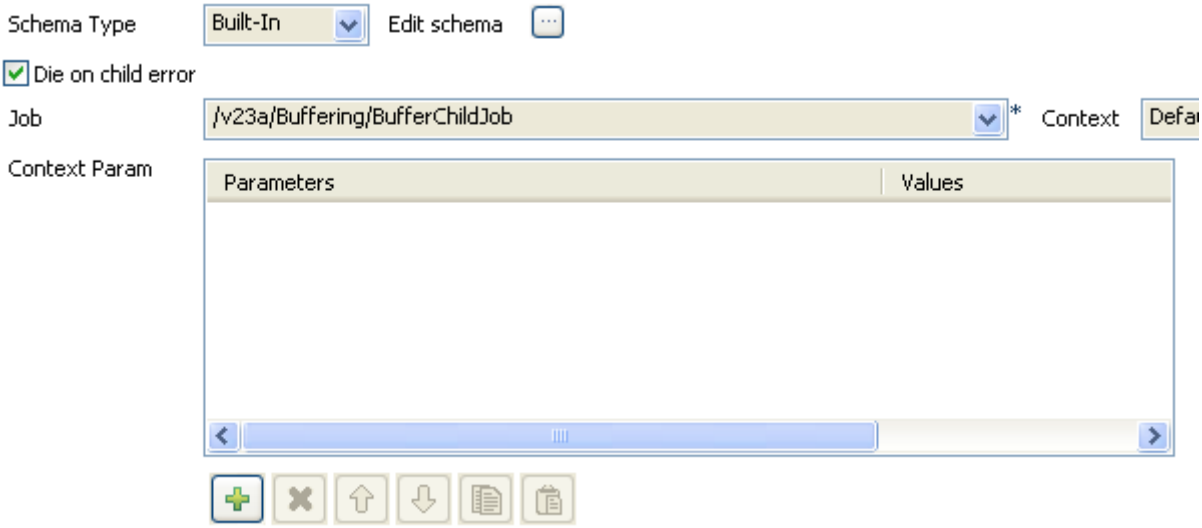


- In **File Name**, browse to the delimited file whose data are to be bufferized.
- Define the **Row** and **Field separators**, as well as the **Header**.



- Describe the **Schema** of the data to be passed on to the **tBufferOutput** component.
- Select the **tBufferOutput** component and set the parameters on the **Basic Settings** tab of the **Component** view.

- Generally the schema is propagated from the input component and automatically fed into the **tBufferOutput** schema. But you could also set part of the schema to be bufferized if you want to.
- Now on the other Job (*BufferFatherJob*) Design, define the parameters of the **tRunJob** component.



- Edit the Schema if relevant and select the column to be displayed. The schema can be identical to the bufferized schema or different.
- You could also define context parameters to be used for this particular execution. To keep it simple, the default context with no particular setting is used for this use case.

Press **F6** to execute the parent Job. The **tRunJob** looks after executing the child Job and returns the data onto the standard console:

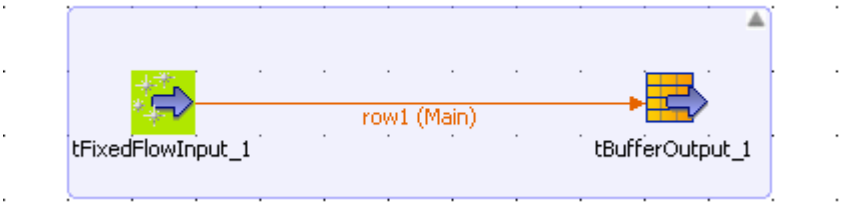
```
Starting job BufferFatherJob at 10:41 05/02/2008.
AL|Alabama|Montgomery|Birmingham
AK|Alaska|Juneau|Anchorage
AZ|Arizona|Phoenix|Phoenix
AR|Arkansas|Little Rock|Little Rock
CA|California|Sacramento|Los Angeles
CO|Colorado|Denver|Denver
CT|Connecticut|Hartford|Bridgeport
DE|Delaware|Dover|Wilmington
FL|Florida|Tallahassee|Jacksonville
GA|Georgia|Atlanta|Atlanta
HI|Hawaii|Honolulu|Honolulu
ID|Idaho|Boise|Boise
IL|Illinois|Springfield|Chicago
IN|Indiana|Indianapolis|Indianapolis
IA|Iowa|Des Moines|Des Moines
KS|Kansas|Topeka|Wichita
KY|Kentucky|Frankfort|Louisville
```

Scenario 2: Buffering output data on the webapp server

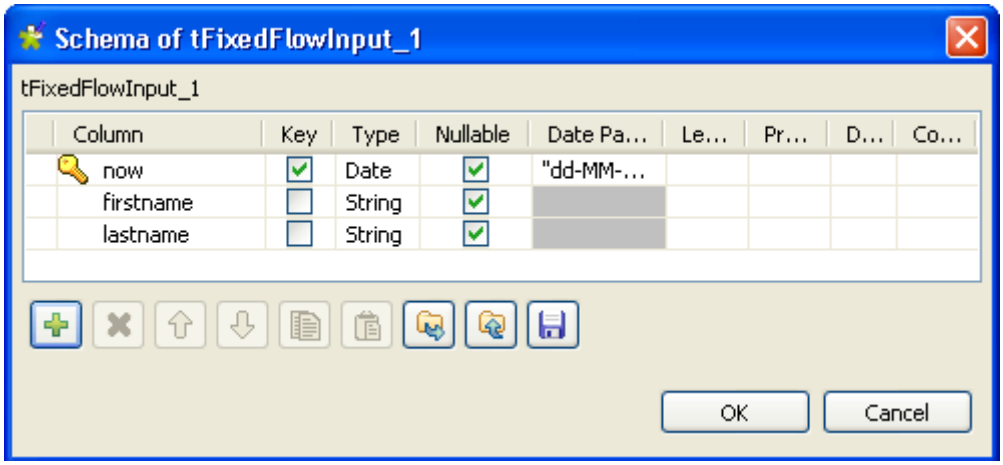
This scenario describes a Job that is called as a Webservice and stores the output data in a buffer directly on the server of the Web application. This scenario creates first a Webservice oriented Job with context variables, and next exports the Job as a Webservice.

Creating a Webservice oriented Job with context variables:

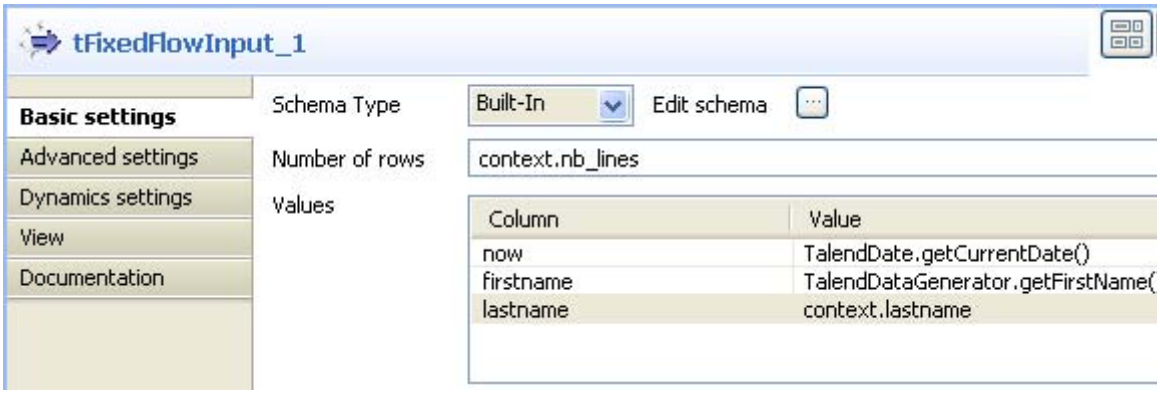
- Drop the following components from the **Palette** onto the design workspace: **tFixedFlowInput** and **tBufferOutput**.
- Connect **tFixedFlowInput** to **tBufferOutput** using a **Row Main** link.



- In the design workspace, select **tFixedFlowInput**.
- Click the **Component** tab to define the basic settings for **tFixedFlowInput**.
- Set the **Schema Type** to **Built-In** and click the three-dot [...] 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*.



- Click the plus button to add the three parameter lines and define your variables.
- 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** panel of the **Basic settings** view of **tFixedFlowInput**.

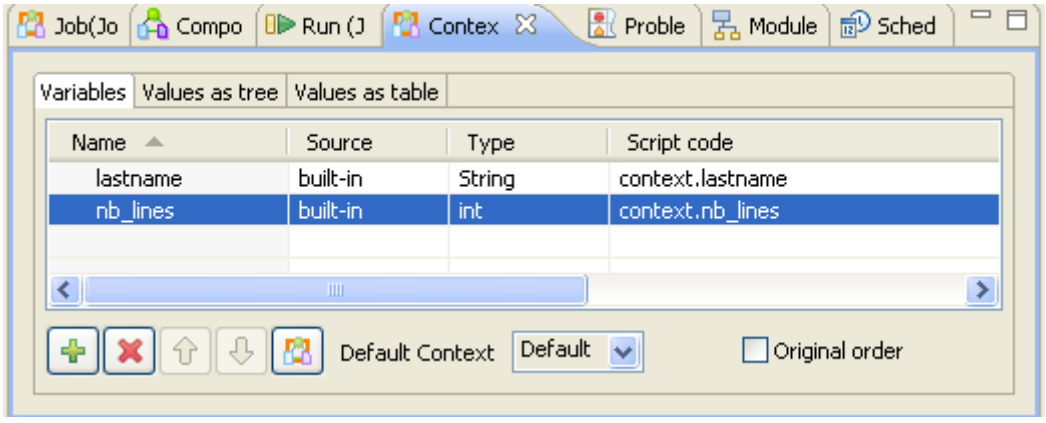


- Click in the **Value** cell of each of the first two defined columns and press **Ctrl+Space** to access the global variable list.
- From the global variable list, select *TalendDate.getCurrentDate()* and *talendDatagenerator.getFirstName*, for the *now* and *firstname* columns respectively.

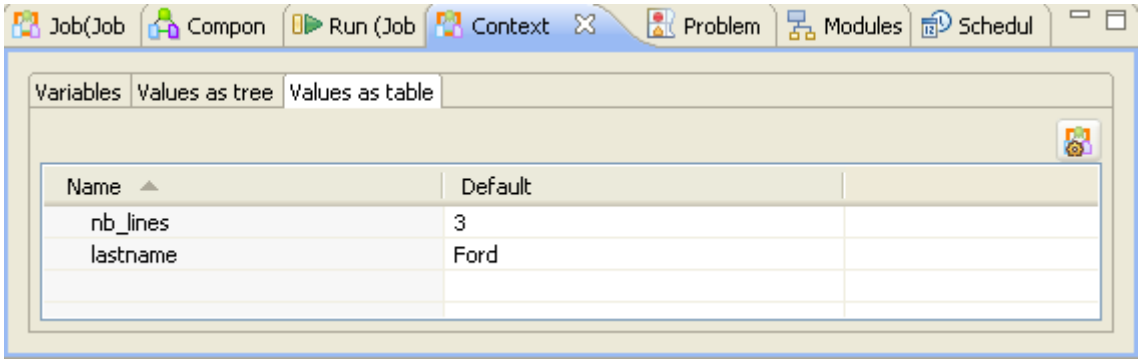
For this scenario, we want to define two context variables: *nb_lines* and *lastname*. In the first we set the number of lines to be generated, and in the second we set the last name to display in the output list. The **tFixedFlowInput** component will generate the number of lines set in the context variable with the three columns: *now*, *firstname* and *lastname*. For more information about how to create and use context variables, see *Defining variables on the Contexts view* of **Talend Open Studio** User Guide.

To define the two context variables:

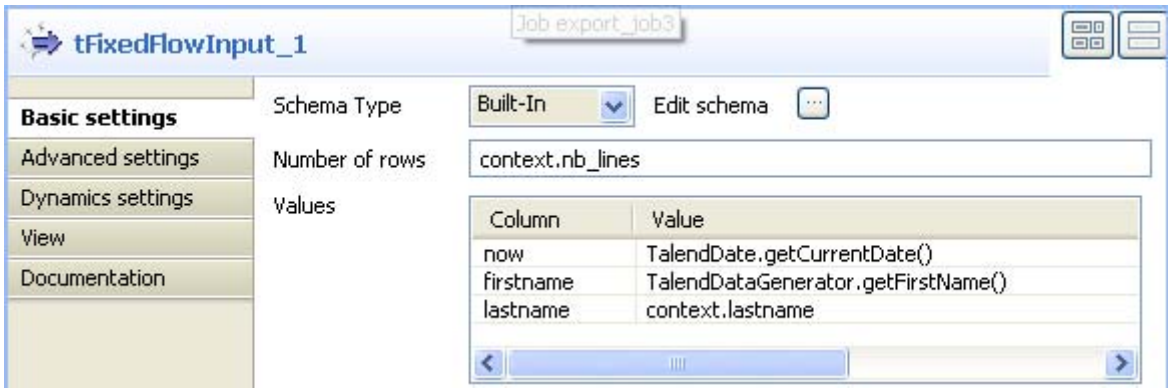
- Select **tFixedFlowInput** and click the **Contexts** tab.
- In the **Variables** view, click the plus button to add two parameter lines and define them.



- Click the **Values as table** tab and define the first parameter to set the number of lines to be generated and the second to set the last name to be displayed.



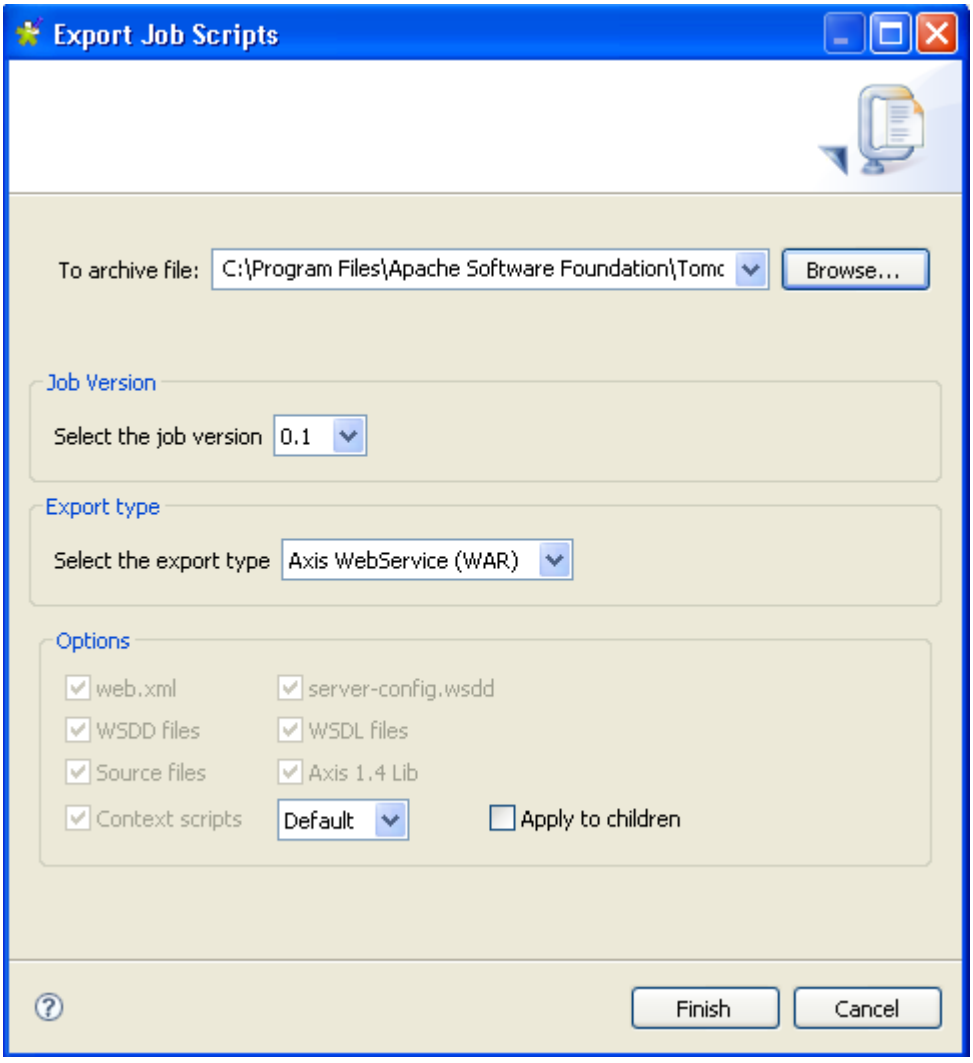
- Click the **Component** tab to go back to the **Basic settings** view of **tFixedFlowInput**.
- Click in the **Value** cell of *lastname* column and press **Ctrl+Space** to access the global variable list.
- From the global variable list, select *context.lastname*, the context variable you created for the last name column.



Exporting your Job as a Webservice:

Before exporting your Job as a Web service, see *Exporting job scripts* in **Talend Open Studio** User Guide for more information.

- In the Repository view, right-click on the above created Job and select **Export Job Scripts**. The **[Export Job Scripts]** dialog box displays.

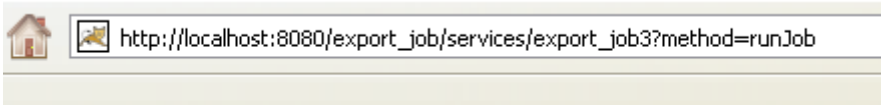


- Click the **Brows...** button to select a directory to archive your Job in.
- In the **Export type** panel, select the export type you want to use in the Tomcat webapp directory (WAR in this example) and click **Finish**. The **[Export Job Scripts]** dialog box disappears.
- Copy the War folder and paste it in a Tomcat webapp directory.

Scenario 3: Calling a Job with context variables from a browser

This scenario describes how to call the Job you created in scenario 2 from your browser with/without modifying the values of the context variables.

Type the following URL into your browser:
http://localhost:8080//export_job/services/export_job3?method=runJob where “export_job” is the name of the webapp directory deployed in Tomcat and “export_job3” is the name of the Job.



Click **Enter** to execute your Job from your browser.

```
- <soapenv:Envelope>
  - <soapenv:Body>
    - <runJobReturn xsi:type="ns1:runJobReturn">
      - <ns1:item xsi:type="ns1:ArrayOf_xsd_string">
          <ns1:item xsi:type="xsd:string">31-07-2008</ns1:item>
          <ns1:item xsi:type="xsd:string">William</ns1:item>
          <ns1:item xsi:type="xsd:string">Ford</ns1:item>
        </ns1:item>
      - <ns1:item xsi:type="ns1:ArrayOf_xsd_string">
          <ns1:item xsi:type="xsd:string">31-07-2008</ns1:item>
          <ns1:item xsi:type="xsd:string">Millard</ns1:item>
          <ns1:item xsi:type="xsd:string">Ford</ns1:item>
        </ns1:item>
      - <ns1:item xsi:type="ns1:ArrayOf_xsd_string">
          <ns1:item xsi:type="xsd:string">31-07-2008</ns1:item>
          <ns1:item xsi:type="xsd:string">James</ns1:item>
          <ns1:item xsi:type="xsd:string">Ford</ns1:item>
        </ns1:item>
      </runJobReturn>
    </soapenv:Body>
  </soapenv:Envelope>
```

The Job uses the default values of the context variables: *nb_lines* and *lastname*, that is it generates three lines with the current date, first name and Ford as a last name.

You can modify the values of the context variables directly from your browser. To call the Job from your browser and modify the values of the two context variables, type the following URL:

http://localhost:8080//export_job/services/export_job3?method=runJob&arg1=--context_param%20lastname=MASSY&arg2=--context_param%20nb_lines=2.

%20 stands for a blank space in the URL language. In the first argument “arg1”, you set the value of the context variable to display “MASSY” as last name. In the second argument “arg2”, you set the value of the context variable to “2” to generate only two lines.

Click **Enter** to execute your Job from your browser.

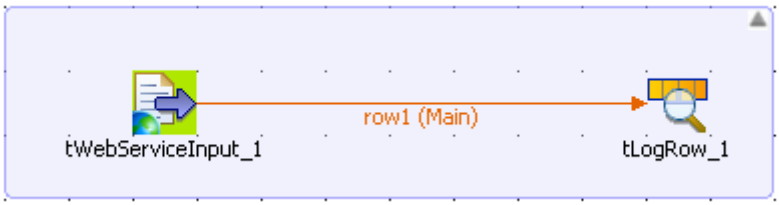
```
- <soapenv:Envelope>
  - <soapenv:Body>
    - <runJobReturn xsi:type="ns1:runJobReturn">
      - <ns1:item xsi:type="ns1:ArrayOf_xsd_string">
          <ns1:item xsi:type="xsd:string">31-07-2008</ns1:item>
          <ns1:item xsi:type="xsd:string">Richard</ns1:item>
          <ns1:item xsi:type="xsd:string">MASSY</ns1:item>
        </ns1:item>
      - <ns1:item xsi:type="ns1:ArrayOf_xsd_string">
          <ns1:item xsi:type="xsd:string">31-07-2008</ns1:item>
          <ns1:item xsi:type="xsd:string">Theodore</ns1:item>
          <ns1:item xsi:type="xsd:string">MASSY</ns1:item>
        </ns1:item>
      </runJobReturn>
    </soapenv:Body>
  </soapenv:Envelope>
```

The Job generates two lines with MASSY as last name.

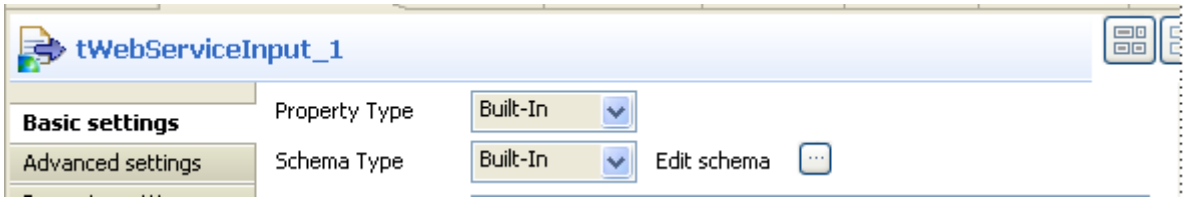
Scenario 4: Calling a Job exported as Webservice in another Job

This scenario describes a Job that calls another Job exported as a Webservice using the **tWebServiceInput**. This scenario will call the Job created in scenario 2.

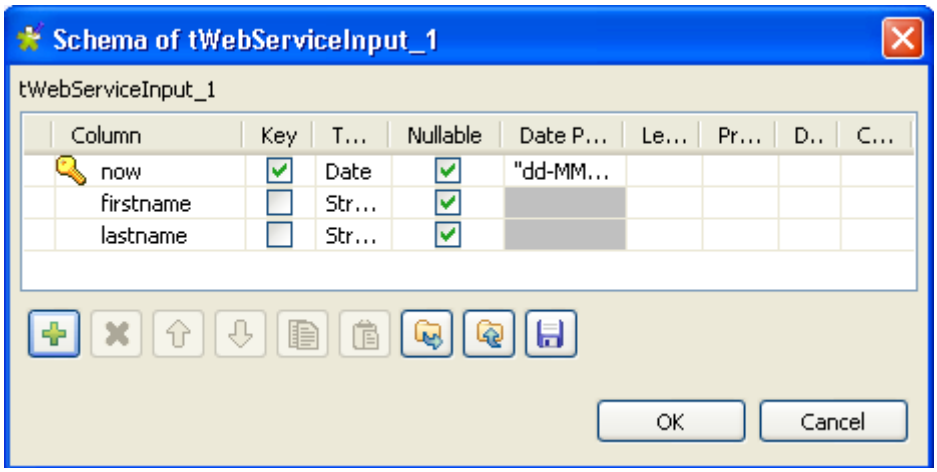
- Drop the following components from the **Palette** onto the design workspace: **tWebServiceInput** and **tLogRow**.
- Connect **tWebserviceInput** to **tLogRow** using a **Row Main** link.



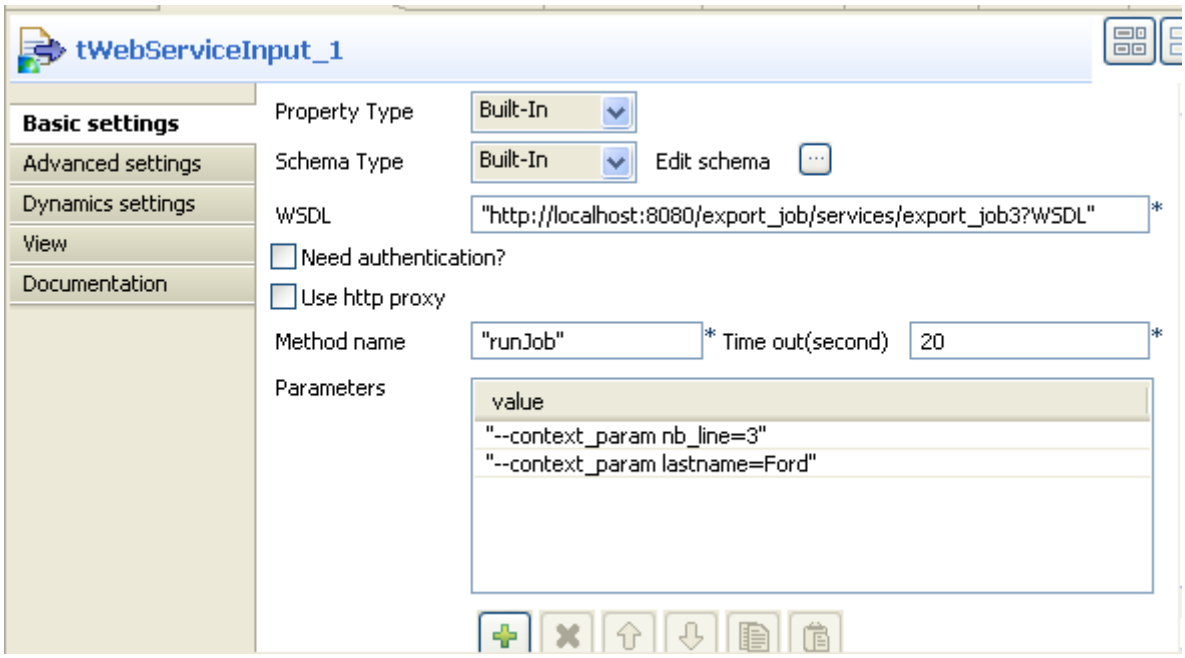
- In the design workspace, select **tWebServiceInput**.
- Click the **Component** tab to define the basic settings for **tWebServiceInput**.



- Set the **Schema Type** to **Built-In** and click the three-dot [...] button next to **Edit Schema** to describe the data structure you want to call from the exported Job. In this scenario, the schema is made of three columns, *now*, *firstname*, and *lastname*.



- Click the plus button to add the three parameter lines and define your variables.Click **OK** to close the dialog box.
- In the **WSDL** field of the **Basic settings** view of **tWebServiceInput**, enter the URL *http://localhost:8080/export_job/services/export_job3?WSDL* where “export_job” is the name od the webapp directory where the Job to call is stored and “export_job3” is the name of the Job itself.



- In the **Method name** field, enter *runJob*.
- In the **Parameters** panel, Click the plus button to add two parameter lines to define your context variables.
- Click in the first **Value** cell to enter the parameter to set the number of generated lines using the following syntax: *--context_param nb_line=3*.

- Click in the second **Value** cell to enter the parameter to set the last name to display using the following syntax: *--context_param lastname=Ford.*
- Select **tLogRow** and click the **Component** tab to display the component view.
- Set the **Basic settings** for the **tLogRow** component to display the output data in a tabular mode. For more information, see *tLogRow on page 628*.
- Save your Job and press **F6** to execute it.

```
Starting job Call_Webservice_In_Job at 14:12 01/08/2008.
.----------+---------+--------.
|           tLogRow_1           |
|=---------+---------+-------=|
|now       |firstname|lastname|
|=---------+---------+-------=|
|01-08-2008|Ronald   |Ford    |
|01-08-2008|Woodrow  |Ford    |
|01-08-2008|Franklin |Ford    |
'----------+---------+--------'

Job Call_Webservice_In_Job ended at 14:12 01/08/2008. [exit code=0]
```

The system generates three columns with the current date, first name, and last name and displays them onto the log console in a tabular mode.

tContextDump

tContextDump properties

Component family	Misc	
Function	**tContextDump** makes a dump copy the values of the active job context.	
Purpose	**tContextDump** can be used to transform the current context parameters into a flow that can then be used in a **tContextLoad**. This feature is very convenient in order to define once only the context and be able to reuse it in numerous Jobs via the **tContextLoad**.	
Basic settings	*Schema type* and *Edit Schema*	In the **tContextDump** use, the schema is read only and made of two columns, Key and Value, corresponding to the parameter name and the parameter value to be copied. A schema is a row description, i.e., it defines the fields that will be processed and passed on to the next component. The schema is either built-in or remote in the Repository. Click **Edit Schema** to make changes to the schema. Note that if you make changes, the schema automatically becomes built-in.
		Built-in: The schema will be created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused in various projects and job flowcharts. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	Print operations	Select this check box to display the context parameters set in the **Run** view.
Usage	This component creates from the current context values, a data flow, therefore it requires to be connected to an output component.	
Limitation	**tContextDump** does not create any non-defined context variable.	

Related Scenario

No scenario is available for this component yet.

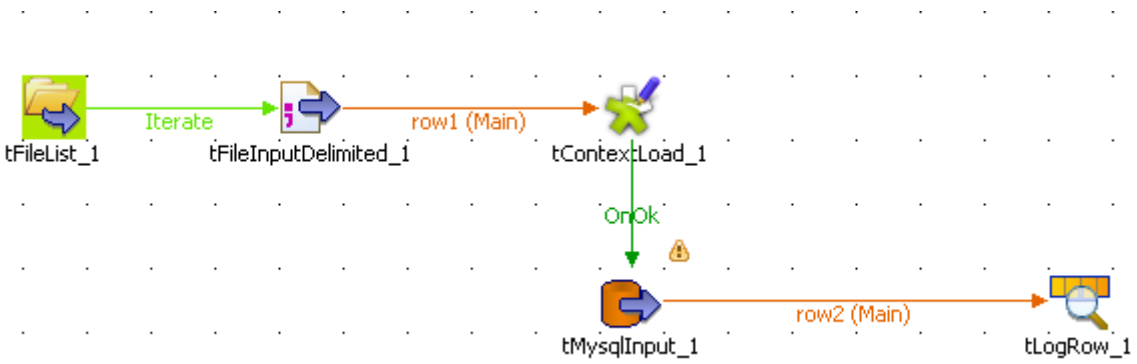
tContextLoad

tContextLoad properties

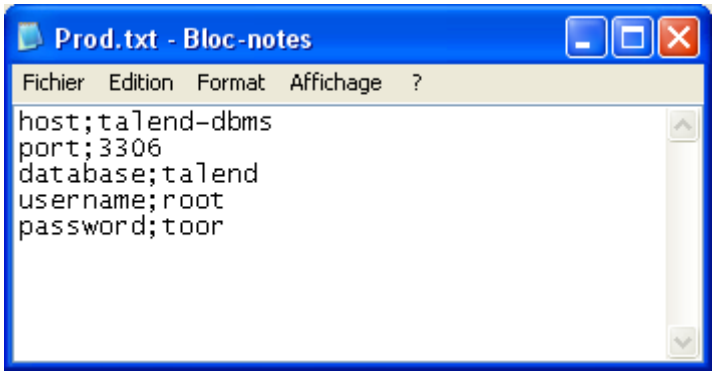
Component family	Misc	
Function	**tContextLoad** modifies dynamically the values of the active context.	
Purpose	**tContextLoad** can be used to load a context from a flow. This component performs also two controls. It warns when the parameters defined in the incoming flow are not defined in the context, and the other way around, it also warns when a context value is not initialized in the incoming flow. But note that this does not block the processing.	
Basic settings	*Schema type* and *Edit Schema*	In the tContextLoad use, the schema must be made of two columns, including the parameter name and the parameter value to be loaded. A schema is a row description, i.e., it defines the fields that will be processed and passed on to the next component. The schema is either built-in or remote in the Repository. Click **Edit Schema** to make changes to the schema. Note that if you make changes, the schema automatically becomes built-in.
		Built-in: The schema will be created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused in various projects and job flowcharts. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	Print operations	Select this check box to display the context parameters set in the **Run** view.
Usage	This component relies on the data flow to load the context values to be used, therefore it requires a preceding input component and thus cannot be a start component.	
Limitation	**tContextLoad** does not create any non-defined variable in the default context.	

Scenario: Dynamic context use in MySQL DB insert

This scenario is made of two subjobs. The first subjob aims at dynamically load the context parameters, and the second subjob uses the loaded context to display the content of a DB table.

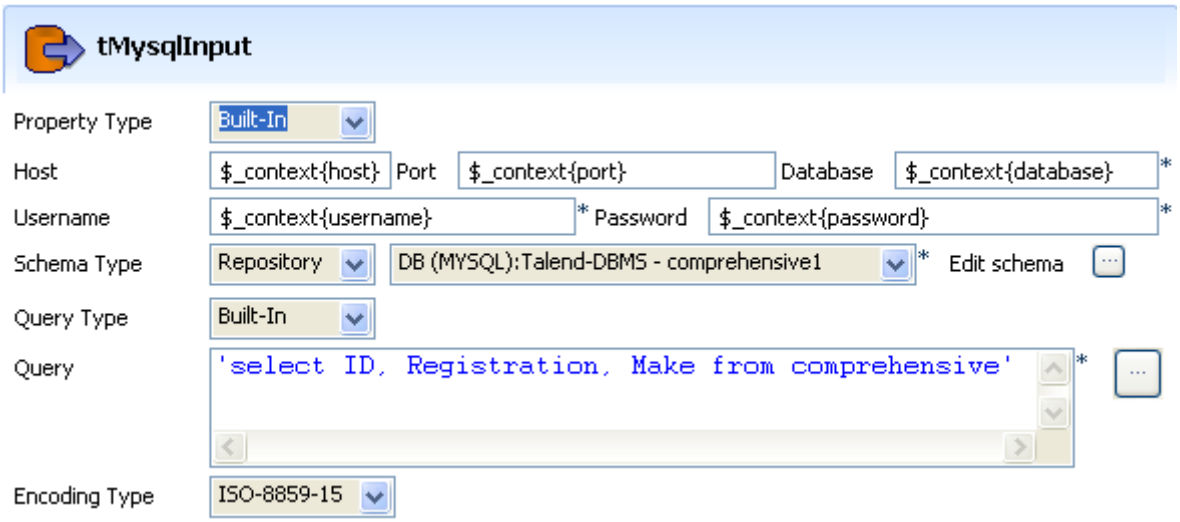


- For the first subjob, drop a **tFilelist**, **tFileInputDelimited**, **tContextLoad** from the **Palette** to the design workspace.
- Drop **tMysqlInput** and a **tLogRow** the same way for the second subjob.
- Connect all the components together.
- Create as many delimited files as there are different contexts and store them in a specific directory, named *Contexts*. In this scenario, *test.txt* contains the local database connection details for testing purpose. And *prod.txt* holds the actual production db details.
- Each file is made of two fields, contain the parameter name and the corresponding value, according to the context.

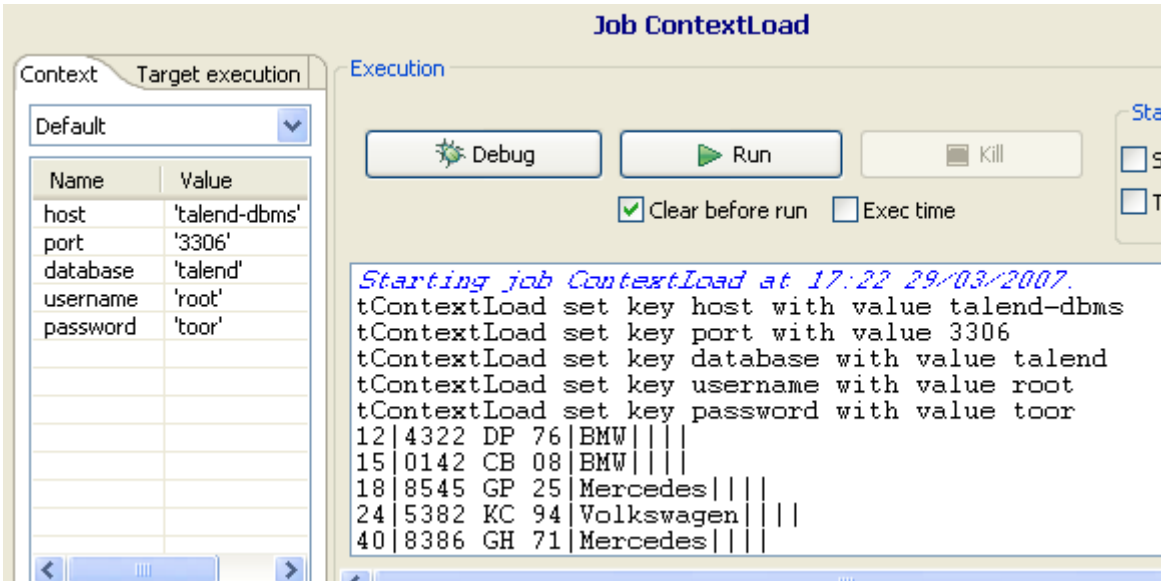


- In the **tFileList** component **Basic settings** panel, select the directory where both context files, *test* and *prod*, are held.
- In the **tFileInputDelimited** component **Basic settings** panel, press **Ctrl+Space bar** to access the global variable list. Select $_globals{tFileList_1}{CURRENT_FILEPATH} to loop on the context files' directory.
- Define the schema manually (Built-in). It contains two columns defined as: *Key* and *Value*.
- Accept the defined schema to be propagated to the next component (**tContextLoad**).
- For this scenario, select the **Print operations** check box in order for the context parameters in use to be displayed on the **Run** panel.
- Then double-click to open the **tMySQLInput** component **Basic settings**.

- For each of the field values being stored in a context file, press F5 and define the user-defined context parameter. For example: The **Host** field has for value parameter *$_context{host}* (in Perl), as the parameter name is *host* in the context file. Its actual value being *talend-dbms*.



- Then fill in the Schema information. If you store the schema in the Repository Metadata, then you can retrieve by selecting **Repository** and the relevant entry in the list.
- And type in the SQL **Query** to be executed on the DB table specified. In this case, a simple select of three columns of the table, which will be displayed on the **Run** tab, through the **tLogRow** component.
- Eventually, press **F6** to run the Job.



The context parameters as well as the select values from the DB table are all displayed on the **Run** view.

tMsgBox

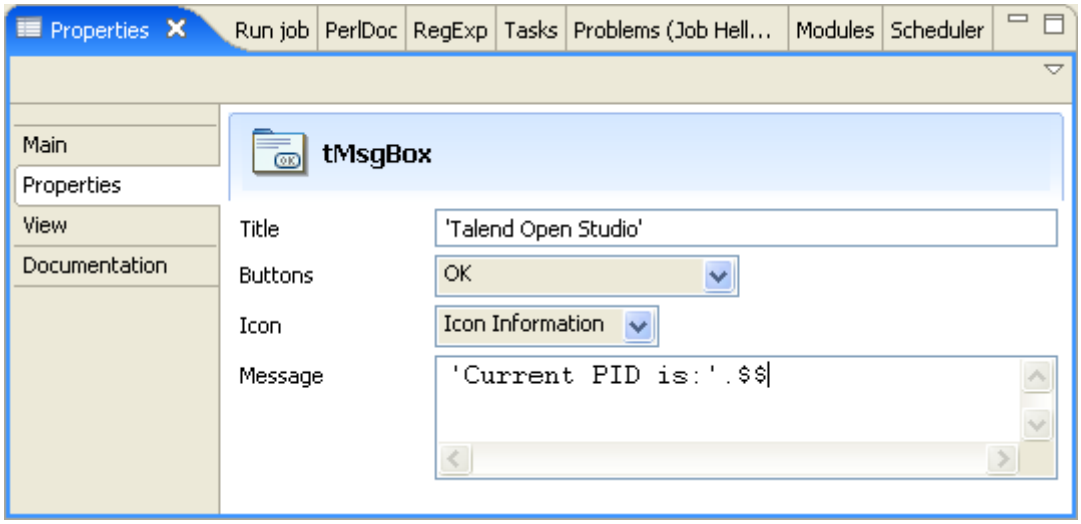
tMsgBox properties

Component family	Misc	
Function	Opens a dialog box with an OK button requiring action from the user.	
Purpose	**tMsgBox** is a graphical break in the job execution progress.	
Basic settings	*Title*	Text entered shows on the title bar of the dialog box created.
	Buttons	Listbox of buttons you want to include in the dialog box. The button combinations are restricted and cannot be changed.
	Icon	Icon shows on the title bar of the dialog box.
	Message	Free text to display as message on the dialog box. Text can be dynamic (for example: retrieve and show a file name).
Usage	This component can be used as intermediate step in a data flow or as a start or end object in the job flowchart. It can be connected to the next/previous component using either a Row or Iterate link.	
Limitation	For Perl users: Make sure the relevant package is installed.	

Scenario: ‘Hello world!’ type test

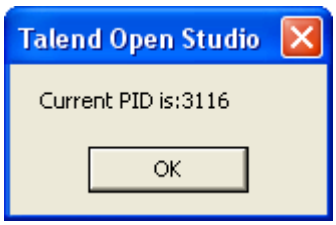
The following scenario creates a single-component Job, where **tMsgBox** is used to display the pid (process id) in place of the traditional “Hello World!” message.

- Drop a **tMsgBox** component from the **Palette** to the design workspace.
- Define the dialog box display properties:



- 'My Title' is the message box title, it can be any variable.
- In the Message field comes the message text in quotes concatenated with the Perl scalar variable ($$) containing the "pid" for this example.
- Switch to the **Run** tab to execute the Job defined.

The Message box displays the message and requires the user to click **OK** to go to the next component or end the Job.



After the user clicked **OK**, the **Run** log is updated accordingly.

Related topic: *Running a Job* of **Talend Open Studio** User Guide.

tRowGenerator

tRowGenerator properties

Component family	Misc	
Function	**tRowGenerator** generates as many rows and fields as needed using random values taken in a list.	
Purpose	Can be used to create an input flow in a Job for testing purpose in particular for boundary test sets	
Basic settings	Row generation editor	The editor allows you to define precisely the columns and nature of data to be generated. You can use predefined routines or type in yourself the function to be used to generate the data specified
Usage	The **tRowGenerator** Editor's ease of use allows users without any Perl or Java knowledge to generate random data for test purpose.	
Limitation	n/a	

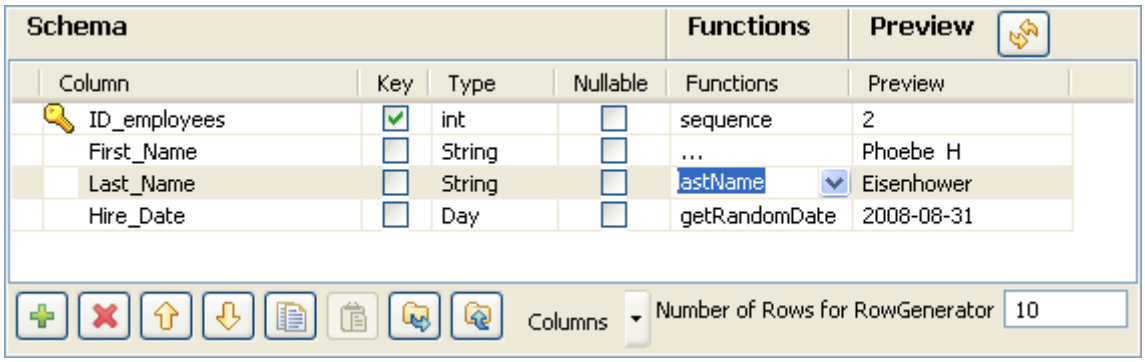
The **tRowGenerator** Editor opens up on a separate window made of two parts:

- a **Schema** definition panel at the top of the window
- and a **Function** definition and preview panel at the bottom.

Defining the schema

First you need to define the structure of data to be generated.

- Add as many columns to your schema as needed, using the **plus (+)** button.
- Type in the names of the columns to be created in the **Columns** area and select the **Key** check box if required
- Make sure you define then the nature of the data contained in the column, by selecting the **Type** in the list. According to the type you select, the list of **Functions** offered will differ. This information is therefore compulsory.



- Some extra information, although not required, might be useful such as **Length**, **Precision** or **Comment**. You can also hide these columns, by clicking on the **Columns** drop-down button next to the toolbar, and unchecking the relevant entries on the list.

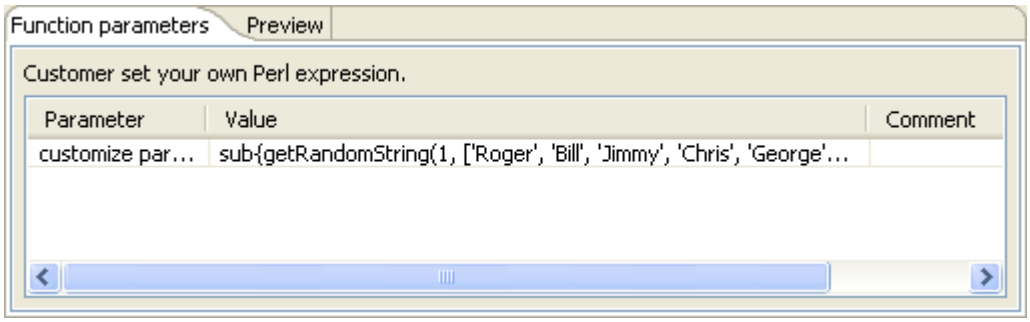
- In the **Function** area, you can select the predefined routine/function if one of them corresponds to your needs.You can also add to this list any routine you stored in the **Routine** area of the **Repository**. Or you can type in the function you want to use in the **Function** definition panel. Related topic: *Defining the function on page 658* of **Talend Open Studio** User Guide
- Click **Refresh** to have a preview of the data generated.
- Type in a number of rows to be generated. The more rows to be generated, the longer it'll take to carry out the generation operation.

The functions list differs from Perl to Java.

Defining the function

You selected the three dots **[...]** as **Function** in the Schema definition panel, as you want to customize the function parameters.

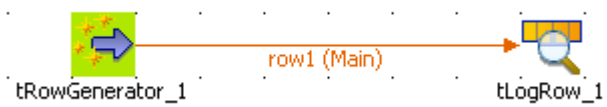
- Select the **Function parameters** tab
- The **Parameter** area displays **Customized parameter** as function name (read-only)



- In the **Value** area, type in the Perl or Java function to be used to generate the data specified.
- Click on the **Preview** tab and click **Preview** to check out a sample of the data generated.

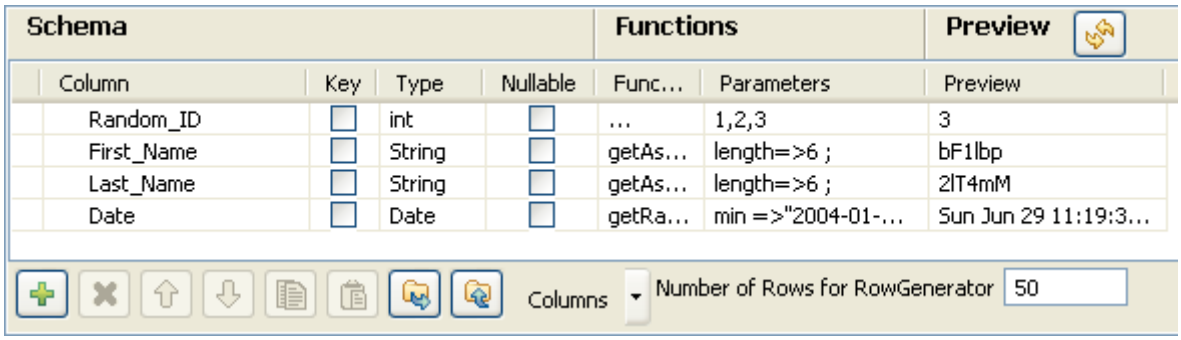
Scenario: Generating random java data

The following scenario creates a two-component Job made in Java, generating 50 rows structured as follows: a randomly picked-up ID in a 1-to-3 range, a random ascii First Name and Last Name generation and a random date taken in a defined range.



- Drop a **tRowGenerator** and a **tLogRow** component from the **Palette** to the design workspace.
- Right-click on the **tRowGenerator** component and select *Row* > *Main*. Drag this main row link onto the **tLogRow** component and release when the plug symbol displays.

- Double-click on the **tRowGenerator** component to open the Editor.
- Define the fields to be generated.



- The random ID column is of integer type, the First and Last names are of string type and the Date is of date type.
- In the **Function** list, select the relevant function or set on the three dots for custom function.
- On the **Function parameters** tab, define the Values to be randomly picked up.

- *First_Name* and *Last_Name* columns are to be generated using the getAsciiRandomString function that is predefined in the system routines. By default the length defined is 6 character-long. But you can change it if need be.
- The *Date* column calls the also predefined getRandomDate function. You can edit the parameter values in the **Function parameters** tab.
- Set the **Number of Rows** to be generated to 50.
- Click **OK** to validate the setting.
- Double-click on the **tLogRow** component to view the Basic settings. The default setting is retained for this Job.
- Press **F6** to run the Job.

```
Starting job JavaRowGenerate at 11:44 10/04/2007.
Running process with context: Default
1|iH6mtj|y3hSXH|Wed Apr 16 11:44:15 CEST 2008
3|cityeQ|uvnKkO|Sun Jan 06 11:44:16 CET 2008
2|X2O0DP|1SVzKT|Wed Oct 26 11:44:16 CEST 2005
1|DSLuE1|S8u15i|Sat Mar 04 11:44:16 CET 2006
3|cc4znX|yuc9cf|Thu Jan 20 11:44:16 CET 2005
3|NwK3PN|lnNyDU|Mon Jun 06 11:44:16 CEST 2005
3|CtO6Ba|pCQgwp|Sat Aug 07 11:44:16 CEST 2004
3|XMt7KN|SUIzFn|Sun May 09 11:44:16 CEST 2004
1|jiPor7|l45rp7|Wed Aug 25 11:44:16 CEST 2004
1|OBN4TX|hywNdP|Sat Feb 09 11:44:16 CET 2008
2|kbVUuE|s21FA0|Tue Jan 27 11:44:16 CET 2004
2|zQlteU|VNHb5w|Sun Oct 05 11:44:16 CEST 2008
3|4nrniu|5Tbnxd|Fri Nov 14 11:44:16 CET 2008
2|7A0qqu|s3jEzJ|Tue Feb 13 11:44:16 CET 2007
3|FgVoTg|uYhJmF|Thu Mar 01 11:44:16 CET 2007
1|Sp4e9P|zXcTK5|Thu Apr 08 11:44:16 CEST 2004
1|E3A4Q3|vKNocj|Wed Jan 11 11:44:16 CET 2006
1|pOMGpG|koklW2|Thu Apr 22 11:44:16 CEST 2004
3|9XOgm8|CCbFya|Sat Feb 18 11:44:16 CET 2006
2|itZXWx|LhQBx1|Tue Apr 18 11:44:16 CEST 2006
1|On1dwv|E7Yaqn|Fri Dec 10 11:44:16 CET 2004
1|UPTZ9M|8s902c|Fri Nov 11 11:44:16 CET 2005
1|WhIT0u|KWn8hO|Mon May 07 11:44:16 CEST 2007
1|LbbeFy|Evn1lc|Thu Mar 16 11:44:16 CET 2006
```

The 50 rows are generated following the setting defined in the **tRowGenerator** editor and the output is displayed in the **Run** console.

MultiSchema components

This chapter details the major components that you can find in **MultiSchema** group of the **Palette** of **Talend Open Studio**.

The **MultiSchema** family groups components that read and write multiple schemas in delimited, positional, and XML file types.

tFileInputMSDelimited

tFileInputMSDelimited properties

Component family	MultiSchema or File/Input	
Function	**tFileInputMSDelimited** reads a complex multi-structured delimited file.	
Purpose	**tFileInputMSDelimited** opens a complex multi-structured file, reads its data structures (schemas) and then uses **Row** links to send fields as defined in the different schemas to the next job components.	
Basic settings	*Multi Schema Editor*	The **[Multi Schema Editor]** helps to build and configure the data flow in a multi-structure delimited file to associate one schema per output. For more information, see *The Multi Schema Editor on page 662*.
	Output	Lists all the schemas you define in the **[Multi Schema Editor]**, along with the related record type and the field separator that corresponds to every schema, if different field separators are used.
Usage	Use this component to read multi-structured delimited files and separate fields contained in these files using a defined separator.	

The Multi Schema Editor

The **[Multi Shema Editor]** enables you to:

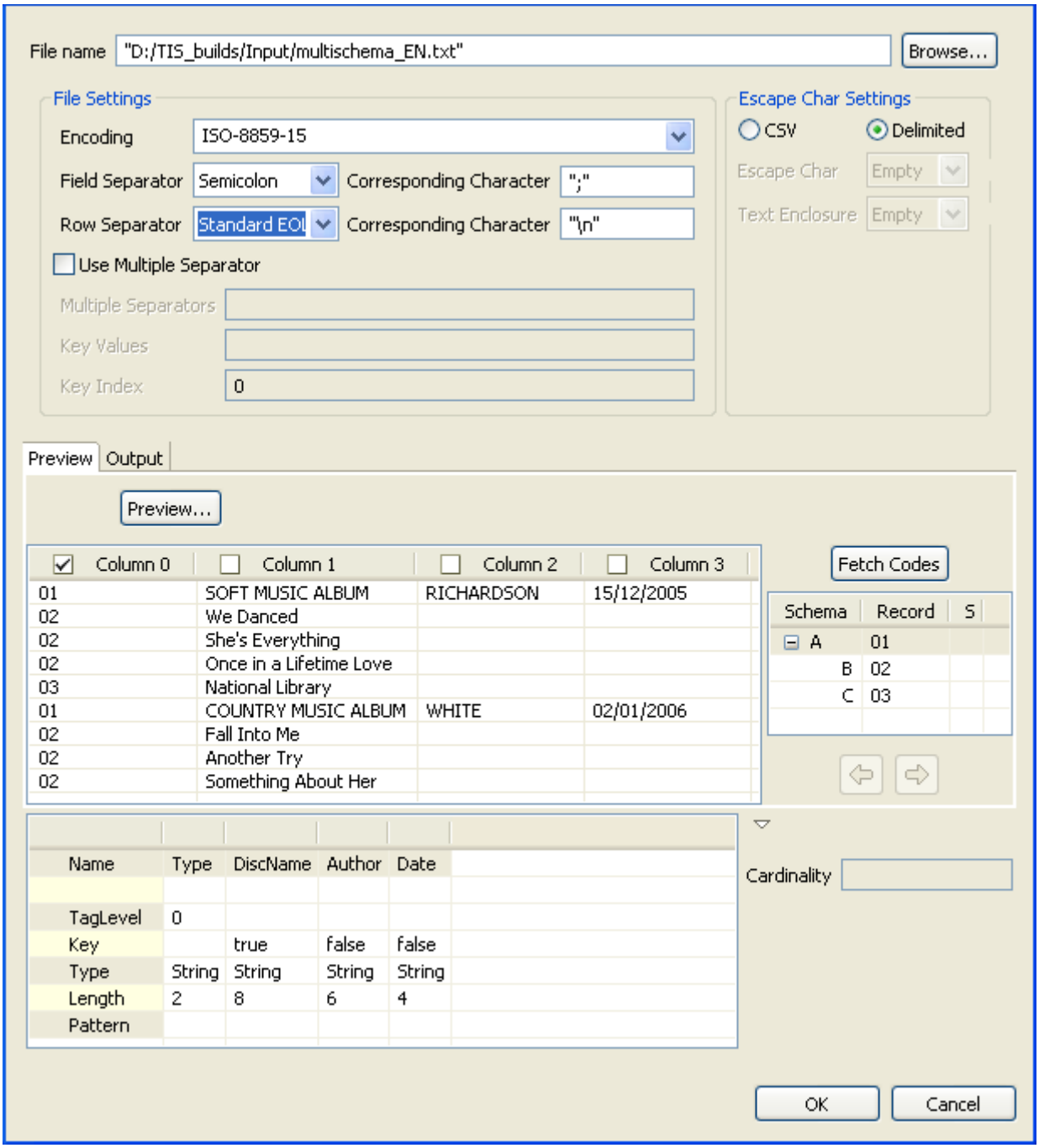
- set the path to the source file,
- define the source file properties,
- define data structure for each of the output schemas.

When you define data structure for each of the output schemas in the **[Multi Schema Editor]**, column names in the different data structures automatically appear in the input schema lists of the components that come after **tFileInputMSDelimited**. However, you can still define data structures directly in the **Basic settings** view of each of these components.

The **[Multi Shema Editor]** also helps to declare the schema that should act as the source schema (primary key) from the incoming data to insure its unicity.The editor uses this mapping to associate all schemas processed in the delimited file to the source schema in the same file.

The editor opens with the first column, that usually holds the record type indicator, selected by default. However, once the editor is open, you can select the check box of any of the schema columns to define it as a primary key.

The below figure illustrates an example of the **[Multi Schema Editor]**.



For detailed information about the usage of the **Multi Schema Editor**, see *Scenario: Reading a multi structure delimited file on page 663*.

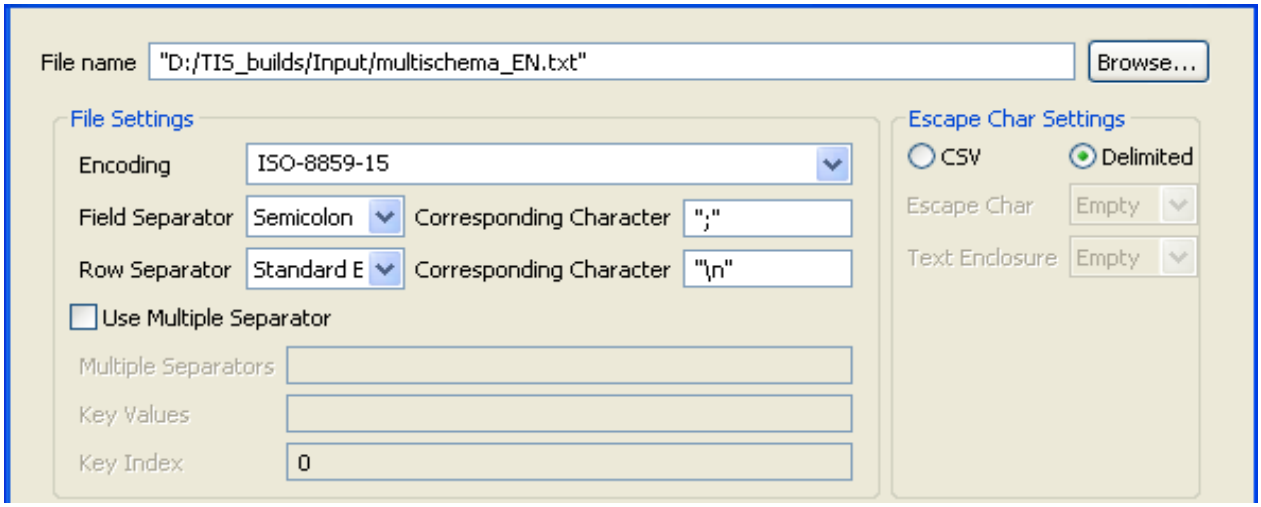
Scenario: Reading a multi structure delimited file

The following scenario creates a Java Job which aims at reading three schemas in a delimited file and displaying their data structure on the **Run Job** console.

The delimited file processed in this example looks like the following:

```
01;SOFT MUSIC ALBUM;RICHARDSON;15/12/2005
02;We Danced
02;She's Everything
02;Once in a Lifetime Love
03;National Library
01;COUNTRY MUSIC ALBUM;WHITE;02/01/2006
02;Fall Into Me
02;Another Try
02;Something About Her
```

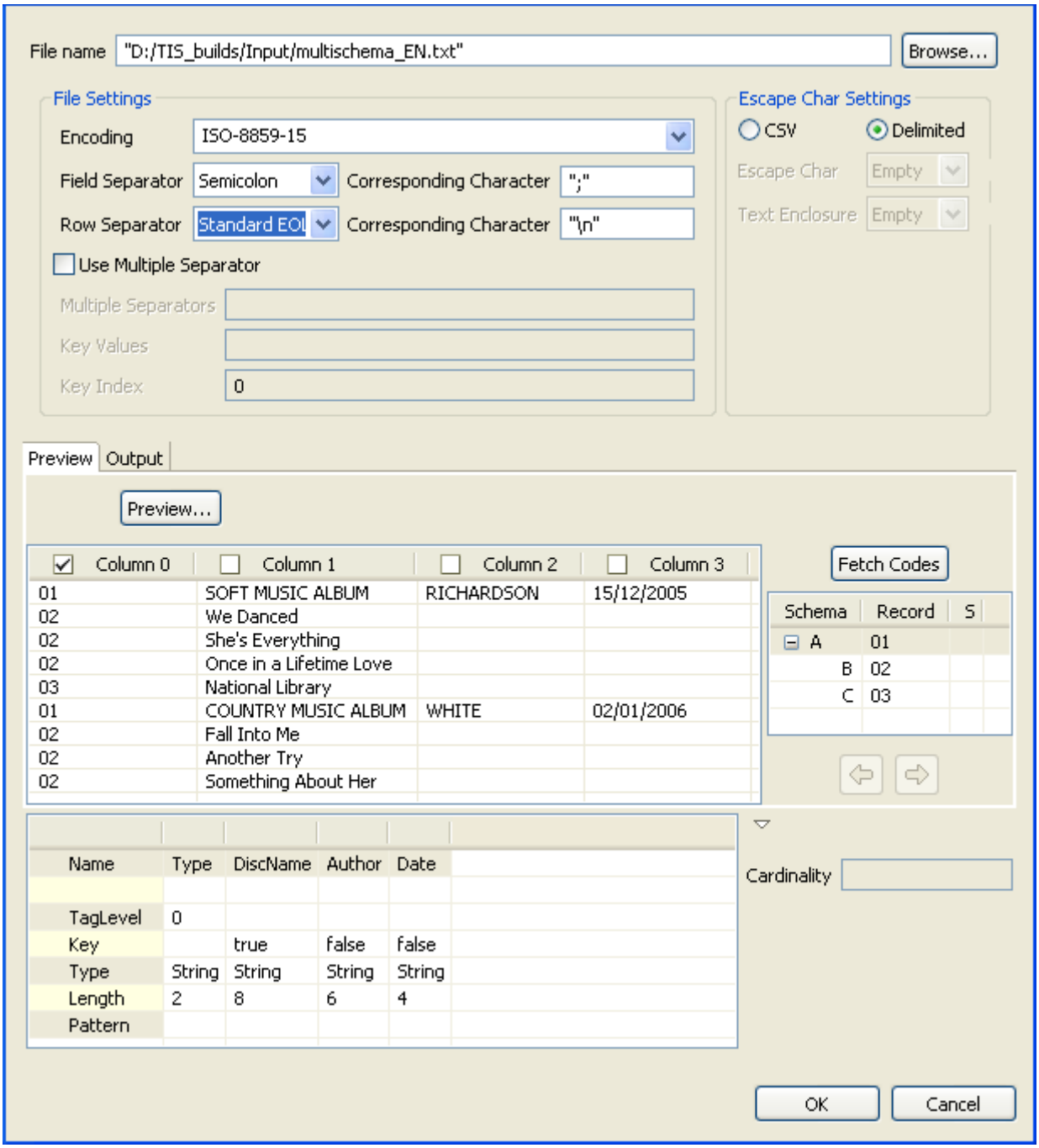
- Drop a **tFileInputMSDelimited** and **tLogRow** (X3) components from the **Palette** onto the design workspace.
- Double-click **tFileInputMSDelimited** to open the **Multi Schema Editor**.



- Click **Browse...** next to the **File name** field to locate the multi schema delimited file you need to process.
- In the **File Settings** area:
  -Select from the list the encoding type the source file is encoded in. This setting is meant to ensure encoding consistency throughout all input and output files.
  -Select the field and row separators used in the source file.

Select the **Use Multiple Separator** check box and define the fields that follow accordingly if different field separators are used to separate schemas in the source file.

A preview of the source file data displays automatically in the **Preview** panel.

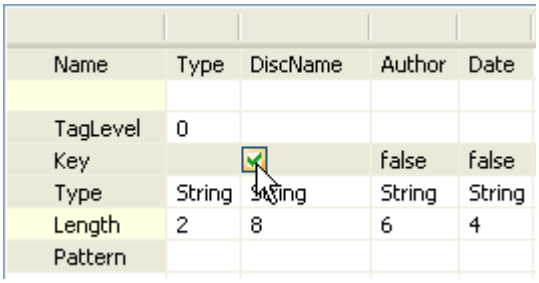


Column 0 that usually holds the record type indicator is selected by default. However, you can select the check box of any of the other columns to define it as a primary key.

- Click **Fetch Codes** to the right of the **Preview** panel to list the type of schema and records you have in the source file. In this scenario, the source file has three schema types (*A*, *B*, *C*).
- Click each schema type in the **Fetch Codes** panel to display its data structure below the **Preview** panel.
- Click in the name cells and set column names for each of the selected schema.
  In this scenario, column names read as the following:
  -Schema *A*: *Type*, *DiscName*, *Author*, *Date*,
  -Schema *B*: *Type*, *SongName*,
  -Schema *C*: *Type*, *LibraryName*.

You need now to set the primary key from the incoming data to insure its unicity (*DiscName* in this scenario). To do that:

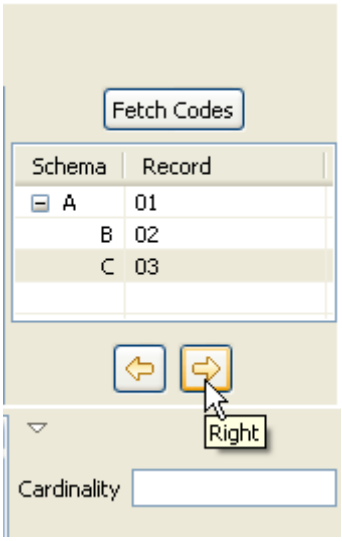
- In the **Fetch Codes** panel, select the schema holding the column you want to set as the primary key (schema *A* in this scenario) to display its data structure.
- Click in the *Key* cell that corresponds to the *DiscName* column and select the check box that displays.



- Click anywhere in the editor and the *false* in the *Key* cell will become *true*.

You need now to declare the parent schema by which you want to group the other “children” schemas (*DiscName* in this scenario). To do that:

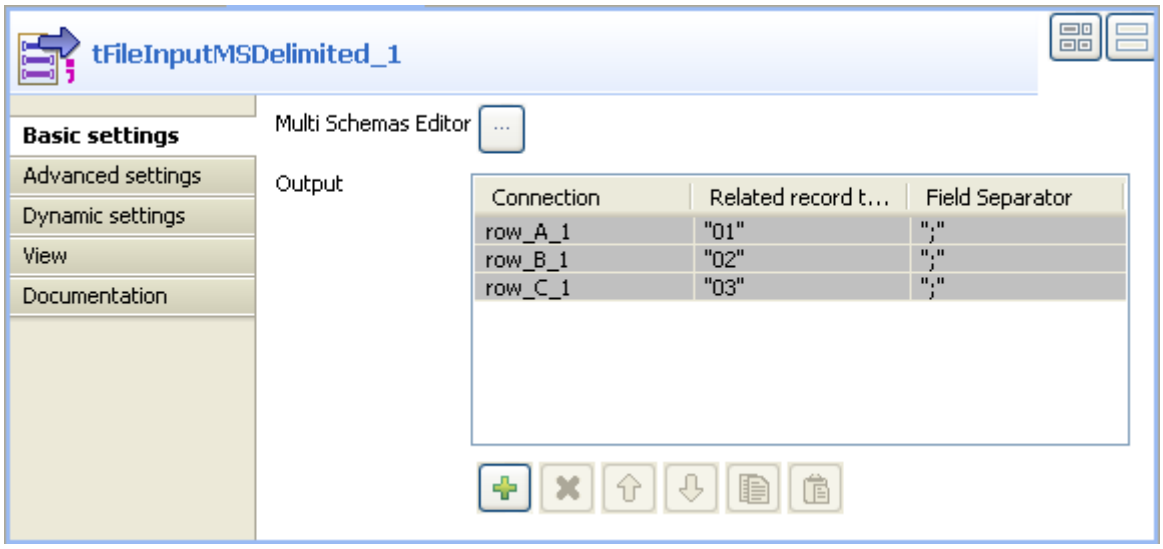
- In the **Fetch Codes** panel, select schema *B* and click the right arrow button to move it to the right.
- Do the same with schema *C*.



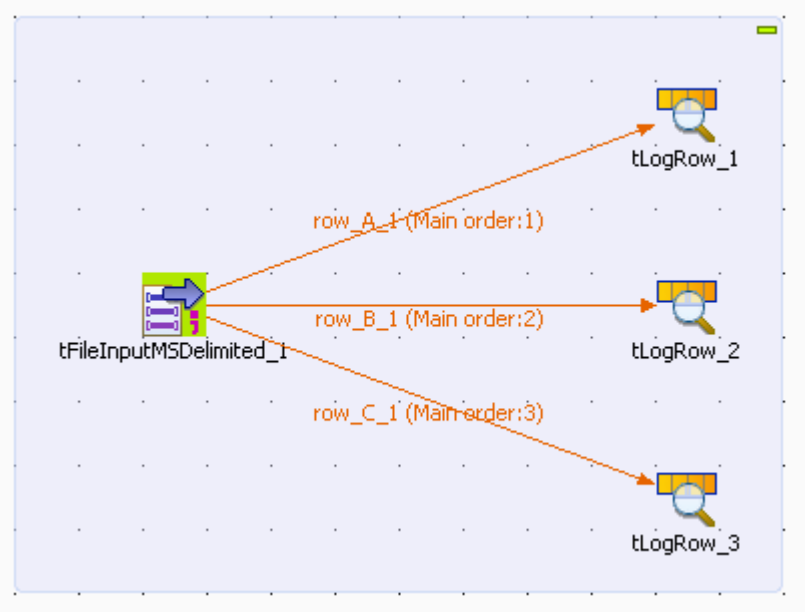
The **Cardinality** field is not compulsory. It helps you to define the number (or range) of fields in “children” schemas attached to the parent schema. However, if you set the wrong number or range and try to execute the Job, an error message will display.

- In the **[Multi Schema Editor]**, click **OK** to validate all the changes you did and close the editor.

The three defined schemas along with the corresponding record types and field separators display automatically in the **Basic settings** view of **tFileInputMSDelimited**.

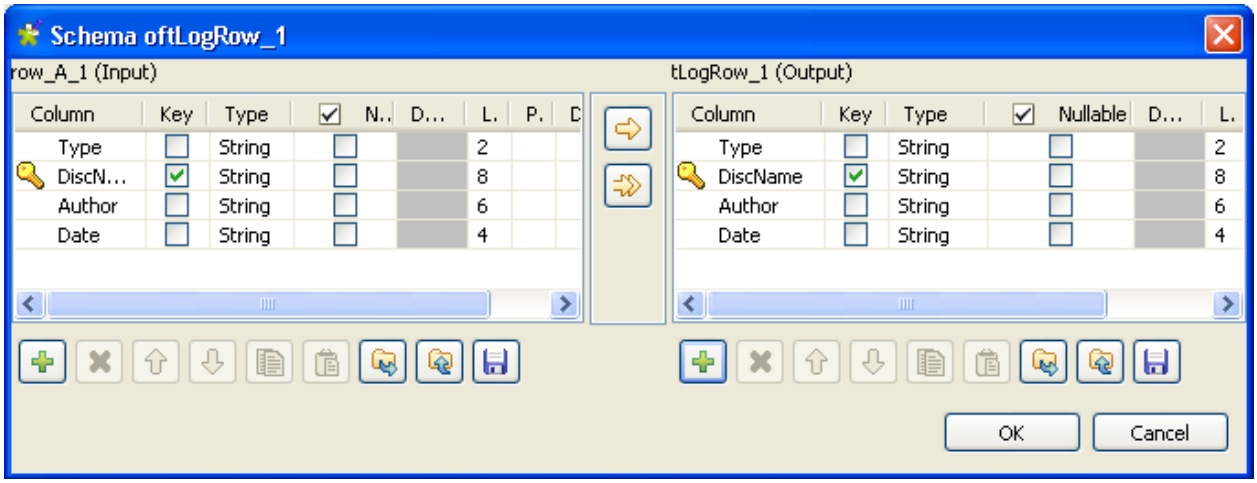


- In the design workspace, right-click **tFileInputMSDelimited** and connect it to **tLogRow1**, **tLogRow2**, and **tLogRow3** using the **row_A_1**, **row_B_1**, and **row_C_1** links respectively.



The three schemas you defined in the **[Multi Schema Editor]** are automatically passed to the three **tLogRow** components.

- If needed, click the **Edit schema** button in the **Basic settings** view of each of the **tLogRow** components to view the input and output data structures you defined in the **Multi Schema Editor** or to modify them.



- Save your Job and click **F6** to execute it.

The multi schema delimited file is read row by row and the extracted fields are displayed on the **Run Job** console as defined in the [**Multi Schema Editor**].

```
Starting job tFileInputMSDelimited at 14:56 13/10/2009.

01|SOFT MUSIC ALBUM|RICHARDSON|15/12/2005
02|We Danced|1
02|She's Everything|1
02|Once in a Lifetime Love|1
03|National Library|1
01|COUNTRY MUSIC ALBUM|WHITE|02/01/2006
02|Fall Into Me|2
02|Another Try|2
02|Something About Her|2
Job tFileInputMSDelimited ended at 14:56 13/10/2009. [e
```

tFileInputMSPositional

tFileInputMSPositional properties

Component family	MultiSchema or File/Input	
Function	**tFileInputMSPositional** reads multiple schemas from a positional file.	
Purpose	tFileInputMSPositional opens a complex multi-structured file, reads its data structures (schemas) and then uses **Row** links to send fields as defined in the different schemas to the next job components.	
Basic settings	*Property type*	Either **Built-in** or **Repository**.
		Built-in: No property data stored centrally.
		Repository: Select the repository file where properties are stored. The fields that come after are pre-filled in using fetched data.
	File Name	Name of the file to be processed. Related topic:*Defining variables from the Component view* of **Talend Open Studio** User Guide.
	Row separator	String (ex: “\n”on Unix) to distinguish rows.
	Header Field Position	Character, string or regular expression to separate fields.
	Records	**Schema**: define as many schemas as needed. **Header value**: enter the value of the Header **Pattern**: set the pattern corresponding to the field separator position. **Reject incorrect row size**: select the check boxes of the schemas where to reject incorrect row size.
	Skip from header	Number of rows to be skipped in the beginning of file.
	Skip from footer	Number of rows to be skipped at the end of the file.
	Limit	Maximum number of rows to be processed. If Limit = 0, no row is read or processed.
	Die on parse error	Let the component die if an parsing error occurs.
	Die on unknown header type	Length values separated by commas, interpreted as a string between quotes. Make sure the values entered in this fields are consistent with the schema defined.
Usage	Use this component to read a multi schemas positional file and separate fields using a position separator value.	

Related scenario

For related use case, see **tFileInputMSDelimited** *Scenario: Reading a multi structure delimited file on page 663*.

tFileOutputMSDelimited

tFileOutputMSDelimited properties

Component family	MultiSchema	
Function	**tFileOutputMSDelimited** writes multiple schema in a delimited file.	
Purpose	**tFileOutputMSDelimited** creates a complex multi-structured delimited file, using data structures (schemas) coming from several incoming **Row** flows.	
Basic settings	*File Name*	Path to the file to be created. Related topic:*Defining variables from the Component view* of **Talend Open Studio** User Guide.
	Row Separator	String (ex: “\n”on Unix) to distinguish rows.
	Field Separator	Character, string or regular expression to separate fields.
	Use Multi Field Separators	Select this check box to set a different field separator for each of the schemas using the **Field separator** field in the **Schemas** area.
	Schemas	The table gets automatically populated by schemas coming from the various incoming rows connected to **tFileOutputMSDelimited**. Fill out the dependency between the various schemas: **Parent row**: Type in the parent flow name (based on the **Row** name transferring the data). **Parent key column**: Type in the key column of the parent row. **Key column**: Type in the key column for the selected row.
Advanced settings	*Advanced separator (for number)*	Select this check box to modify the separators used for numbers: **Thousands separator**: define separators for thousands. **Decimal separator**: define separators for decimals.
	CSV options	Select this check box to take into account all parameters specific to CSV files, in particular **Escape char** and **Text enclosure** parameters.
	Create directory if not exists	This check box is selected by default. It creates the directory that holds the output delimited file, if it does not already exist.
	Encoding type	Select the encoding from the list or select **Custom** and define it manually. This field is compulsory for DB data handling.
	Don’t generate empty file	Select this check box if you do not want to generate empty files.
	tStatCatcher Statistics	Select this check box to gather the job processing metadata at a job level as well as at each component level.
Usage	Use this component to write a multi-schema delimited file and separate fields using a field separator value.	

Related scenarios

No scenario is available for this component yet.

tFileOutputMSPositional

tFileOutputMSPositional properties

Component family	MultiSchema or File/Output	
Function	**tFileOutputMSPositional** writes multiple schemas in a positional file.	
Purpose	**tFileOutputMSPositional** creates a complex multi-structured file, using data structures (schemas) coming from several incoming **Row** flows.	
Basic settings	*File Name*	Path to the file to be created. Related topic:*Defining variables from the Component view* of **Talend Open Studio** User Guide.
	Row separator	String (ex: “\n”on Unix) to distinguish rows.
	Schemas	The table gets automatically populated by schemas coming from the various incoming rows connected to **tFileOutputMSPositional**. Fill out the dependency between the various schemas: **Parent row**: Type in the parent flow name (based on the Row name transferring the data). **Parent key column**: Type in the key column of the parent row **Key column**: Type in the key column for the selected row. **Pattern**: Type in the pattern that positions the fields separator for each incoming row. **Padding char**: type in the padding character to be used **Alignment**: Select the relevant alignment parameter
Usage	Use this component to write a multi-schema positional file and separate fields using a position separator value.	

Related scenario

No scenario is available for this component yet.

tFileInputMSXML

tFileInputMSXML Properties

Component family	MultSchema or XML or File/Input	
Function	**tFileInputMSXML** reads and outputs multiple schema within an XML structured file.	
Purpose	**tFileInputMSXML** opens a complex multi-structured file, reads its data structures (schemas) and then uses **Row** links to send fields as defined in the different schemas to the next job components.	
Basic settings	File Name	Name of the file to be processed. Related topic:*Defining variables from the Component view* of **Talend Open Studio** User Guide.
	Root XPath query	The root of the XML tree, which the query is based on.
	Outputs	**Schema**: define as many schemas as needed. **Schema XPath loop**: node of the tree which the loop is based on. **XPath Queries**: Enter the fields to be extracted from the structured input. **Create empty row**: select the check boxes of the schemas where you want to create empty rows.
Limitation	n/a	

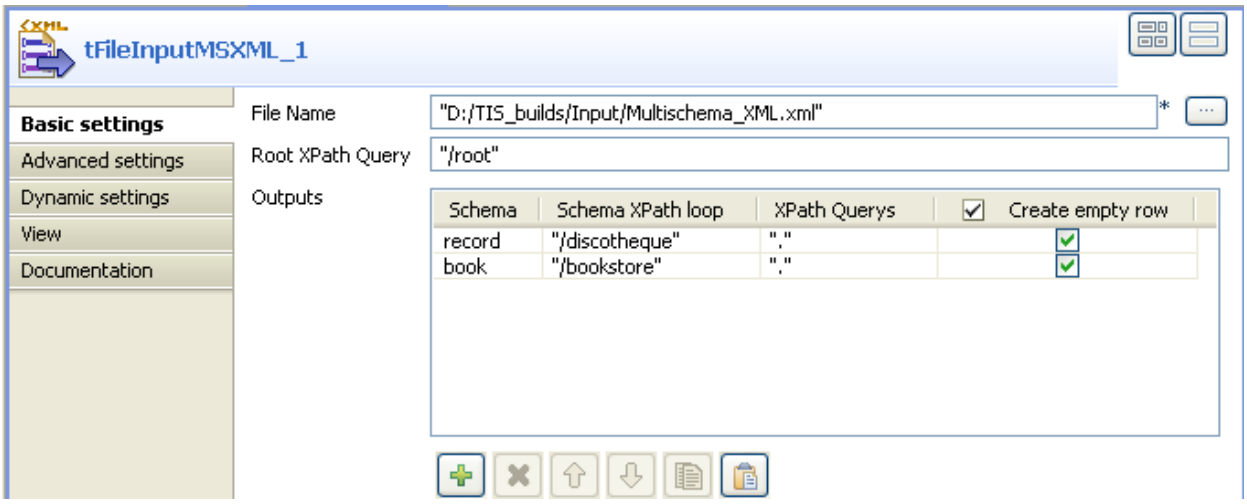
Scenario: Reading a multi structure XML file

The following scenario creates a Java Job which aims at reading a multi schema XML file and displaying data structures on the **Run Job** console.

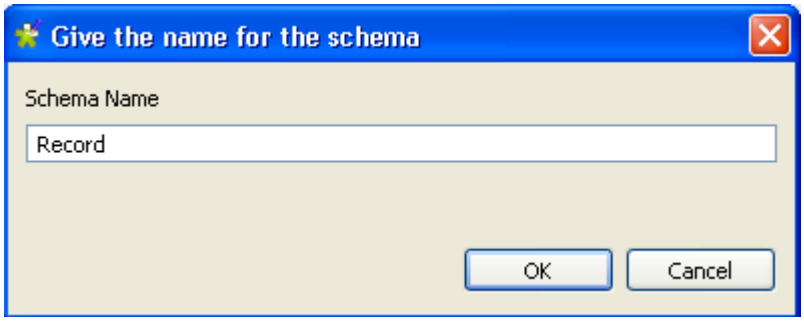
The XM file processed in this example looks like the following:

```
- <root>
  - <discotheque>
      <record>Something About Her</record>
      <record>Fall Into Me</record>
      <record>Once In A Liftime</record>
    </discotheque>
  - <bookstore>
      <book>Another Try</book>
      <book>By Myself</book>
      <book>null</book>
    </bookstore>
  </root>
```

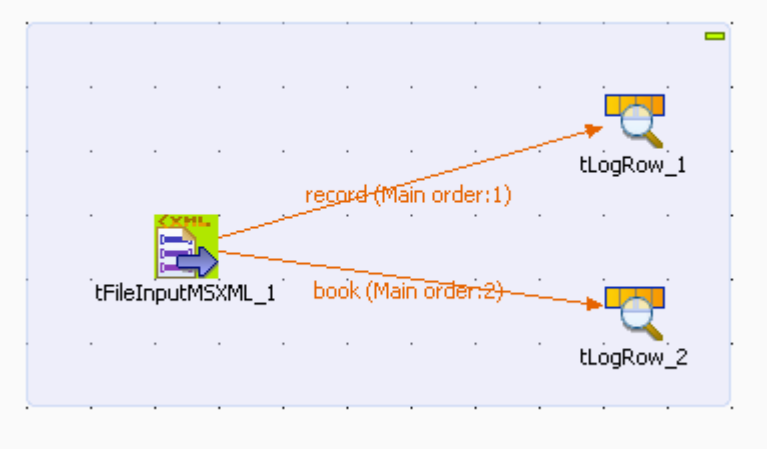
- Drop a **tFileInputMSXML** and two **tLogRow** components from the **Palette** onto the design workspace.
- Double-click **tFileInputMSXML** to open the component **Basic settings** view.



- Browse to the XML file you want to process.
- In the **Root XPath query** field, enter the root of the XML tree, which the query will be based on.
- Click the plus button to add lines in the **Outputs** table where you can define the output schema, two lines in this scenario: *record* and *book*.
- In the **Outputs** table, click in the **Schema** cell and then click a three-dot button to display a dialog box where you can define the schema name.



- Enter a name for the output schema and click **OK** to close the dialog box.
  The **tFileInputMSXML** schema editor displays.
- Define the schema you previously defined in the **Outputs** table.
- Do the same for all the output schemas you want to define.
- In the design workspace, right-click **tFileInputMSXML** and connect it to **tLogRow1**, and **tLogRow2** using the **record** and **book** links respectively.



- In the **Basic settings** view and in the **Schema XPath loop** cell, enter the node of the XML tree, which the loop is based on.
- In the **XPath Queries** cell, enter the fields to be extracted from the structured XML input.
- Select the check boxes next to schemas' names where you want to create empty rows.
- Save your Job and press **F6** to execute it. The defined schemas are extracted from the multi schema XML structured file and displayed on the console.

The multi schema XML file is read row by row and the extracted fields are displayed on the **Run Job** console as defined.

```
Starting job tFileInputXML_ at 15:15 21/07/2009.

.-------------------------------------------------------------.
|                           tLogRow_1                         |
|=-----------------------------------------------------------=|
|recordname                                                   |
|=-----------------------------------------------------------=|
|
    Something About Her
    Fall Into Me
    Once In A Liftime
  |
'-------------------------------------------------------------'

.---------------------------------------.
|               tLogRow_2               |
|=--------------------------------------=|
|bookname                               |
|=--------------------------------------=|
||
    Another Try
    By Myself
      null
  |
'---------------------------------------'
Job tFileInputXML_ ended at 15:15 21/07/2009. [exit code=0]
```

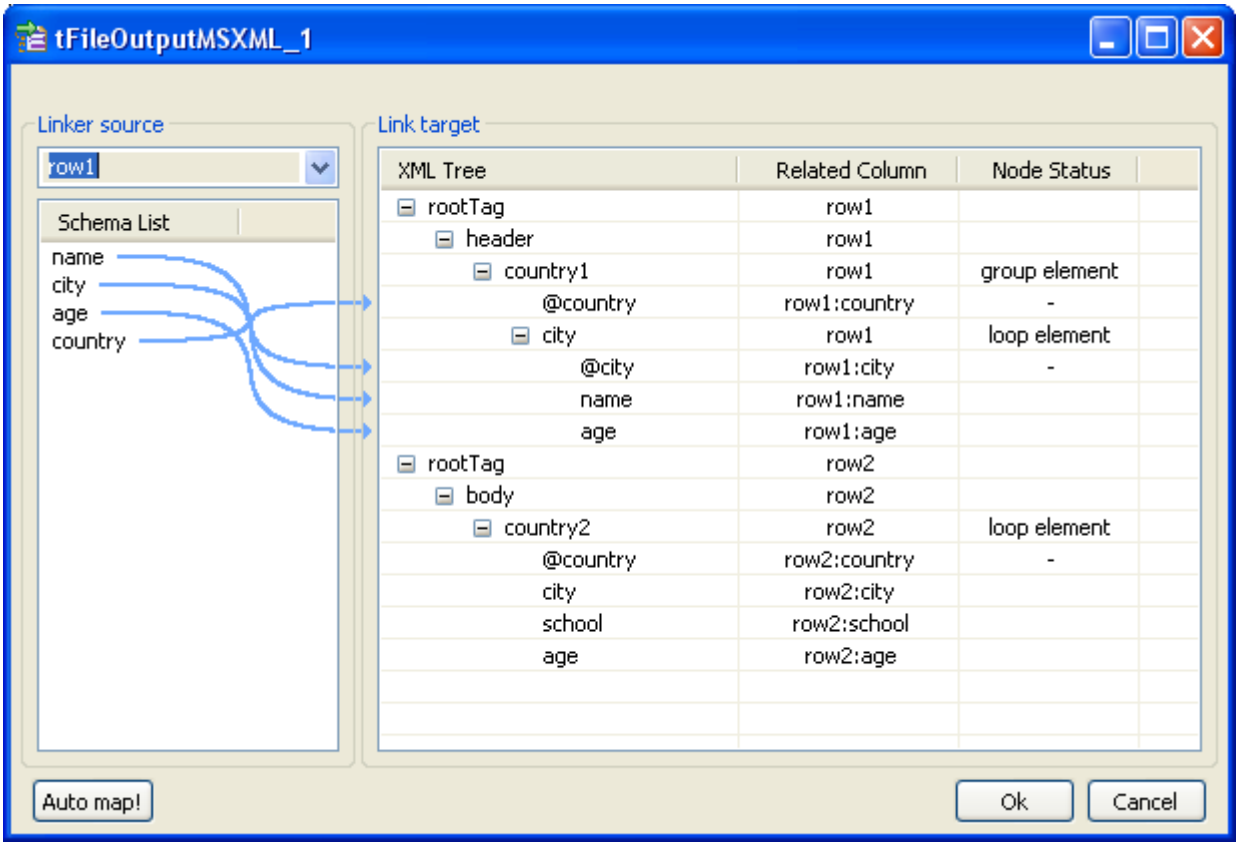
tFileOutputMSXML

tFileOutputMSXML Properties

Component family	MultiSchema or File/Output	
Function	**tFileOutputMSXML** writes multiple schema within an XML structured file.	
Purpose	**tFileOutputMSXML** creates a complex multi-structured XML file, using data structures (schemas) coming from several incoming **Row** flows.	
Basic settings	File Name	Path to the file to be created. Related topic:*Defining variables from the Component view* of **Talend Open Studio** User Guide.
	Configure XML tree	Opens the dedicated interface to help you set the XML mapping. For details about the interface*Defining the MultiSchema XML tree on page 676*.
Limitation	n/a	

Defining the MultiSchema XML tree

Double-click on the **tFileOutputMSXML** component to open the dedicated interface or click on the three-dot button on the **Basic settings** vertical tab of the **Component** tab.



To the left of the mapping interface, under **Linker source**, the drop-down list includes all the input schemas that should be added to the multi-schema output XML file (on the condition that more than one input flow is connected to the **tFileOutputMSXML** component).

And under **Schema List**, are listed all columns retrieved from the input data flow in selection.

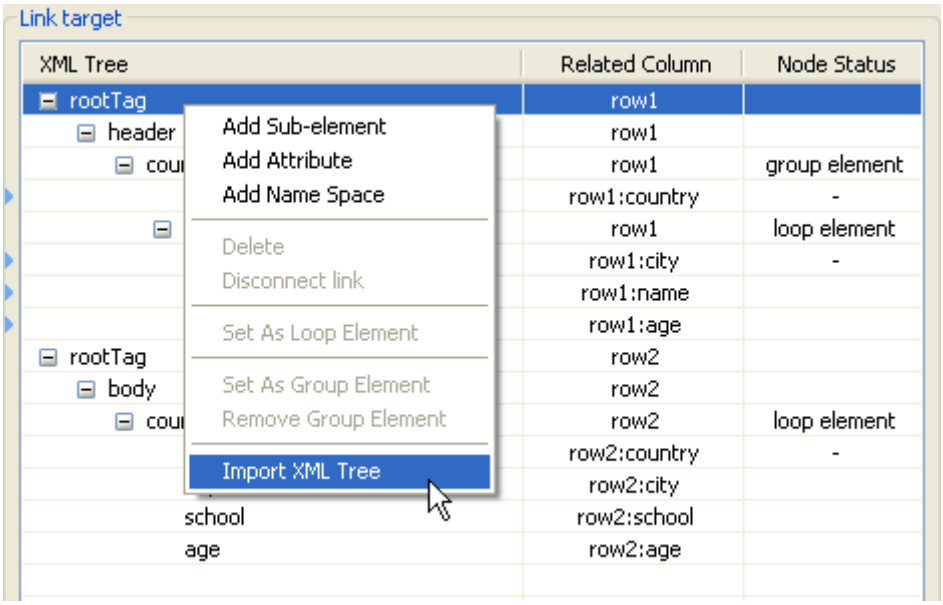
To the right of the interface, are expected all XML structures you want to create in the output XML file.

You can create manually or easily import the XML structures. Then map the input schema columns onto each element of the XML tree, respectively for each of the input schemas in selection under **Linker source**.

Importing the XML tree

The easiest and most common way to fill out the XML tree panel, is to import a well-formed XML file.

- Rename the **root tag** that displays by default on the **XML tree** panel, by clicking on it once.
- Right-click on the root tag to display the contextual menu.
- On the menu, select **Import XML tree**.
- Browse to the file to import and click **OK**.



The **XML Tree** column is hence automatically filled out with the correct elements. You can remove and insert elements or sub-elements from and to the tree:

- Select the relevant element of the tree.
- Right-click to display the contextual menu
- Select Delete to remove the selection from the tree or select the relevant option among: **Add sub-element**, **Add attribute**, **Add namespace** to enrich the tree.

Creating manually the XML tree

If you don't have any XML structure already defined, you can manually create it.

- Rename the **root tag** that displays by default on the **XML tree** panel, by clicking on it once.
- Right-click on the root tag to display the contextual menu.
- On the menu, select **Add sub-element** to create the first element of the structure.

You can also add an attribute or a child element to any element of the tree or remove any element from the tree.

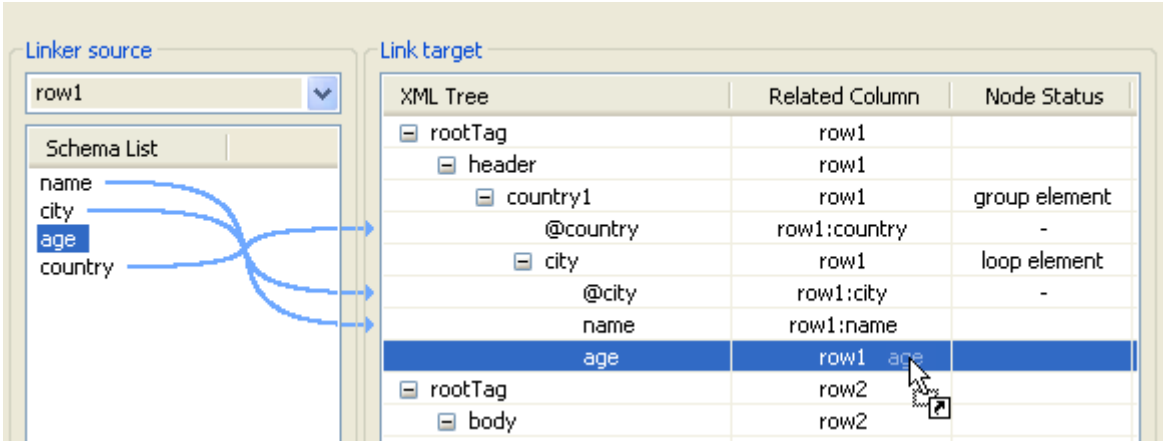
- Select the relevant element on the tree you just created.
- Right-click to the left of the element name to display the contextual menu.
- On the menu, select the relevant option among: **Add sub-element**, **Add attribute**, **Add namespace** or **Delete**.

Mapping XML data from multiple schema sources

Once your XML tree is ready, select the first input schema that you want to map.

You can map each input column with the relevant XML tree element or sub-element to fill out the **Related Column**:

- Click on one of the **Schema column name**.
- Drag it onto the relevant sub-element to the right.
- Release the mouse button to implement the actual mapping.



A light blue link displays that illustrates this mapping. If available, use the **Auto-Map** button, located to the bottom left of the interface, to carry out this operation automatically.

You can disconnect any mapping on any element of the XML tree:

- Select the element of the XML tree, that should be disconnected from its respective schema column.
- Right-click to the left of the element name to display the contextual menu.

- Select **Disconnect link**.

The light blue link disappears.

Defining the node status

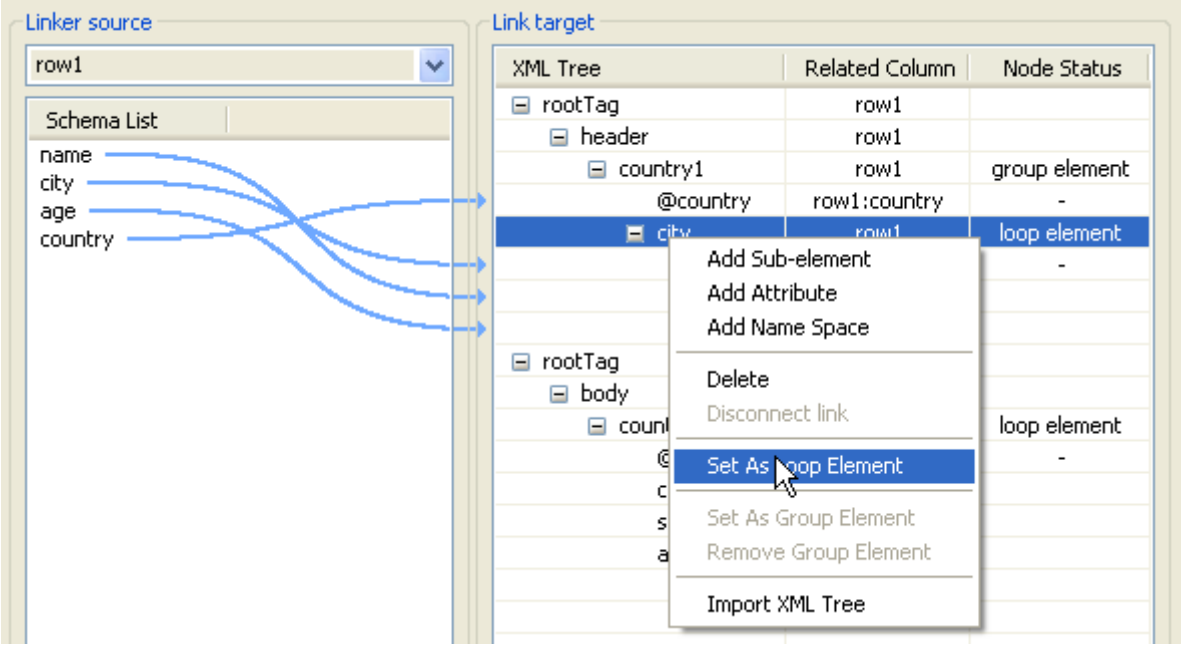
Defining the XML tree and mapping the data is not sufficient. You also need to define the loop elements **for each of the source in selection** and if required the group element.

Loop element

The loop element allows you to define the iterating object. Generally the Loop element is also the row generator.

To define an element as loop element:

- Select the relevant element on the XML tree.
- Right-click to the left of the element name to display the contextual menu.
- Select **Set as Loop Element**.



The **Node Status** column shows the newly added status.

There can only be one loop element at a time.

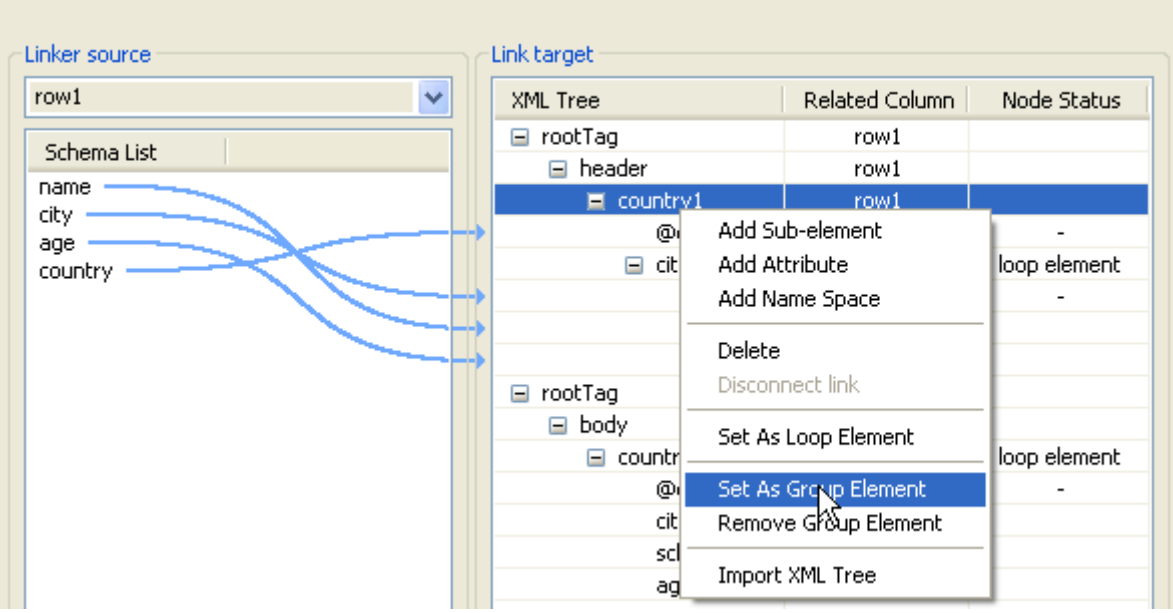
Group element

The group element is optional, it represents a constant element where the Groupby operation can be performed. A group element can be defined on the condition that a loop element was defined before.

When using a group element, the rows should be sorted, in order to be able to group by the selected node.

To define an element as group element:

- Select the relevant element on the XML tree.
- Right-click to the left of the element name to display the contextual menu.
- Select **Set as Group Element**.



The **Node Status** column shows the newly added status and any group status required are automatically defined, if needed.

Click **OK** once the mapping is complete to validate the definition for this source and perform the same operation for the other input flow sources.

Related scenario

No scenario is available for this component yet.

Orchestration components

This chapter details the major components that you can find in **Orchestration** group of the **Palette** of **Talend Open Studio**.

The Orchestration family groups components that help you sequence or orchestrate tasks or processings in your Jobs and subjobs and so on.

tFileList

tFileList belongs to two component families: File and Orchestration. For more information on **tFileList**, see *tFileList on page 527*.

tFlowToIterate

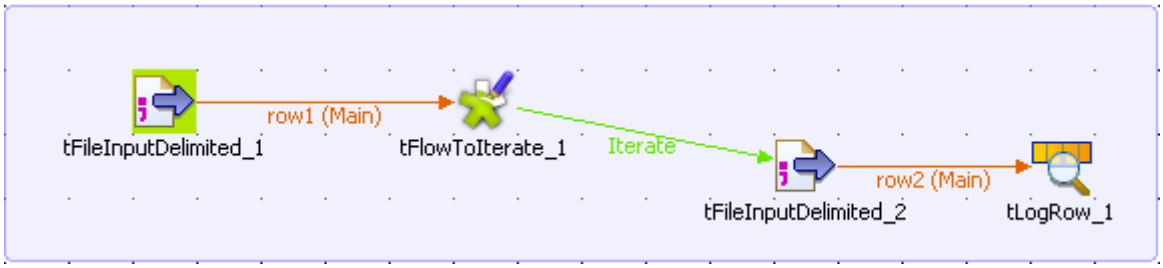
tFlowToIterate Properties

Component family	Orchestration	
Function	**tFlowToIterate** transforms data flow into a list.	
Purpose	Allows to transform processable flow into non processable data.	
Basic settings	*Use the default (key, value) in global variables*	When selected, the system uses the default value of the global variable in the current Job.
	Customize	**key**: Type in a name for the new global variable. Press **Ctrl+Space** to access all available variables either global or user-defined.
		value: Click in the cell to access a list of the columns attached to the defined global variable.
Usage	You can not use this component as a start component. **tFlowToIterate** requires an output component.	

Scenario: Transforming data flow to a list

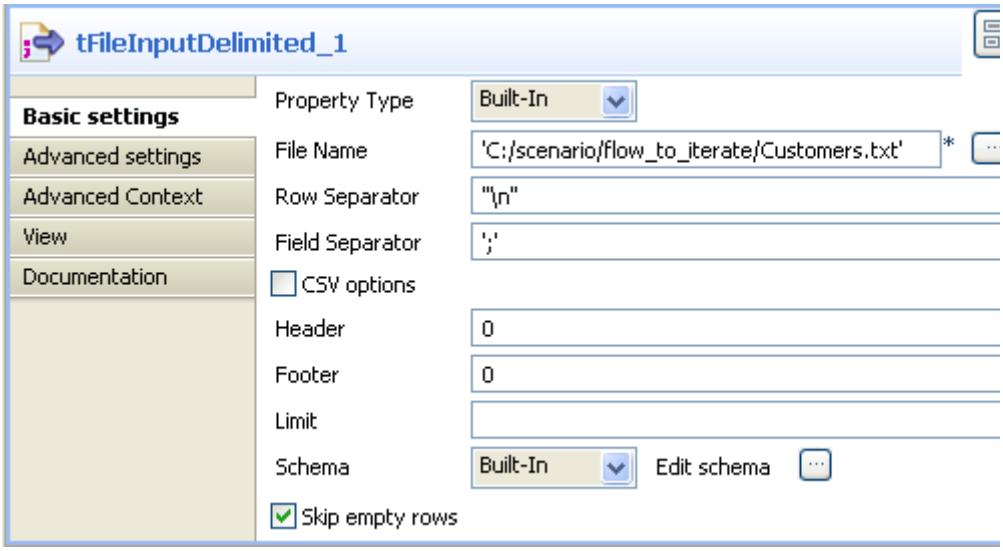
The following scenario describes a Job that reads a list of files from a defined input file, iterates on each of the files, selects input data and displays the output on the **Run** log console.

- Drop the following components from the **Palette** onto the design workspace: **tFileInputDelimited** (x2), **tFlowToIterate**, and **tLogRow**.
- Via a right-click on each of the components, connect the first **tFileInputdelimited** to **tFlowToIterate** using a **Row Main** link, **tFlowToIterate** to the second **tFileInputdelimited** using an **Iterate** link, and the second **tFileInputdelimited** to **tLogRow** using a **Row Main** link.



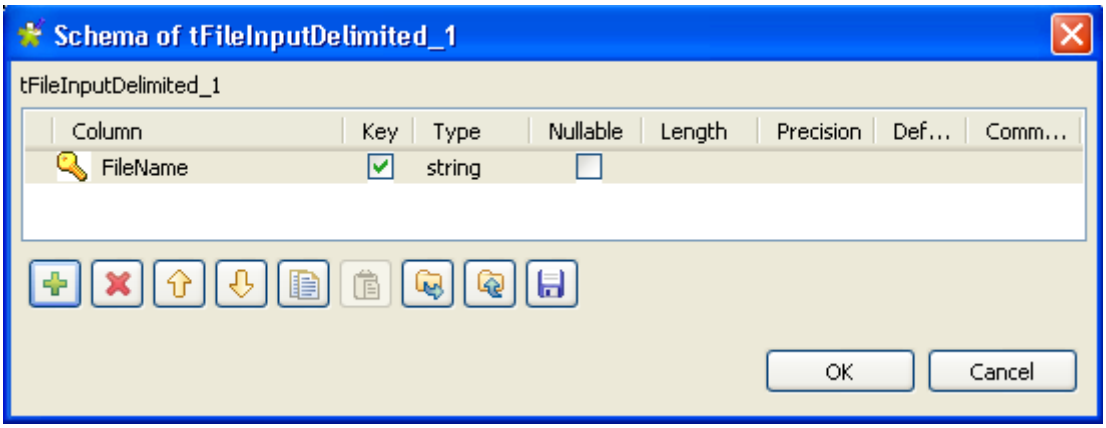
- In the design workspace, select the first **tFileInputDelimited**.
- Click the **Component** tab to display the relevant view where you can define the basic settings for **tFileInputDelimited**.
- In the **Basic settings** view, click the three-dot [...] button next to the **File Name** field to select the path to the input file.

The **File Name** field is mandatory.

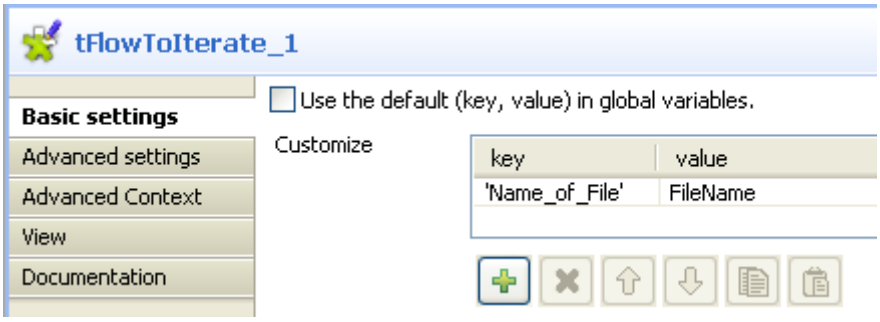


The input file used in this scenario is called *Customers*. It is a text file that holds three other simple text files: *Name*, *E-mail* and *Address*. The first text file, *Name*, is made of one column holding customers' names. The second text file, *E-mail*, is made of one column holding customers' e-mail addresses. The third text file, *Address*, is made of one column holding customers' postal addresses.

- Fill in all other fields as needed. For more information, see *tFileInputDelimited properties on page 494*. In this scenario, the header and the footer are not set and there is no limit for the number of processed rows
- Click **Edit schema** to describe the data structure of this input file. In this scenario, the schema is made of one column, *FileName*.

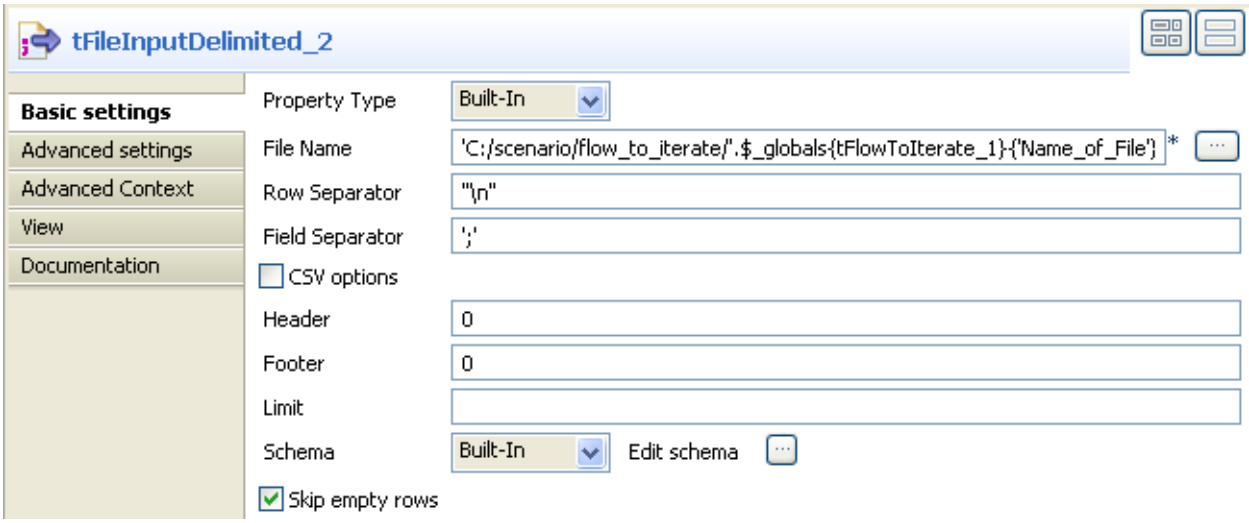


- In the design workspace, select **tFlowToIterate**.
- Click the **Component** tab to define the basic settings for **tFlowToIterate**.

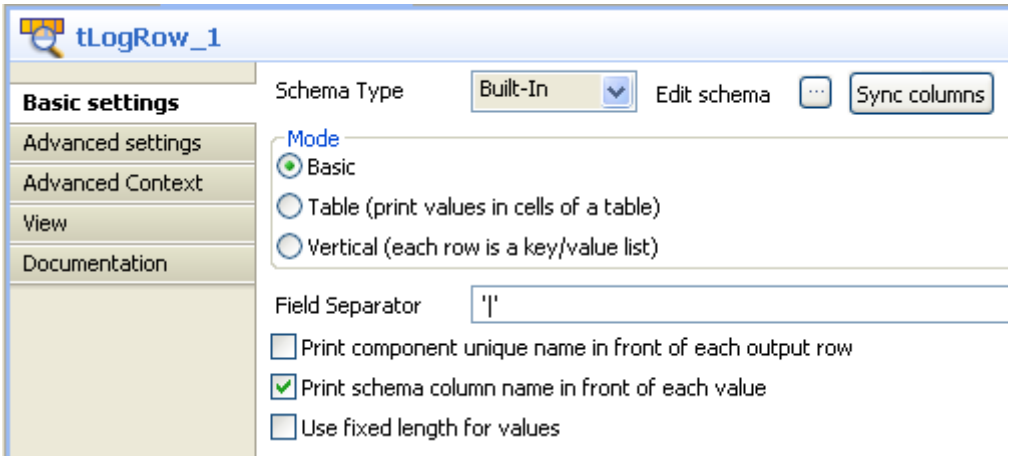


- If needed, select the **Use the default (key, value) in global variables** check box to use the default value of the global variable.
- Click the plus button to add new parameter lines and define your variables.
- Click in the **key** cell to modify the variable name as desired.

  You can press **Ctrl+Space** in the **key** cell to access the list of global and user-specific variables.
- In the design workspace, select the second **tFileInputDelimited**.
- Click the **Component** tab to define the basic settings for the second **tFileInputDelimited**.



- In the **File Name** field, enter the file name using the variable containing the name of the file. You must use the correct syntax according to the language used, Perl or Java. In Perl, the relevant syntax is *.$_globals{tFlowToIterate}{'Name_of_File}.*In java, the relevant syntax is *+globalMap.get("file").*
- Fill in all other fields as needed. For more information, see *tFileInputDelimited properties on page 494*.
- In the design workspace, select the last component, **tLogRow**.
- Click the **Component** tab to define the basic settings for **tLogRow**.



- Define your settings as needed. For more information, see *tLogRow properties on page 628*.
- Save your Job and press **F6** to execute it

```
Starting job flow_to_iterate at 16:32 25/07/2008.
col1: Madison Moore
col1: Andrew Taylor
col1: Christopher Anderson
col1: madison_moor@hotmail.com
col1: Andrew_taylor@usamail1.com
col1: Madison Moore
col1:     100 MAIN ST
col1:     PO BOX 1022
col1:     SEATTLE WA 98104
col1:     USA
col1: Andrew Taylor
col1:     300 BOYLSTON AVE E
col1:     SEATTLE WA 98102
col1:     USA
Job flow_to_iterate ended at 16:32 25/07/2008. [exit code=0]
```

Customers' names, customers' e-mails, and customers' postal addresses display on the console preceded by the schema column name.

tIterateToFlow

tIterateToFlow Properties

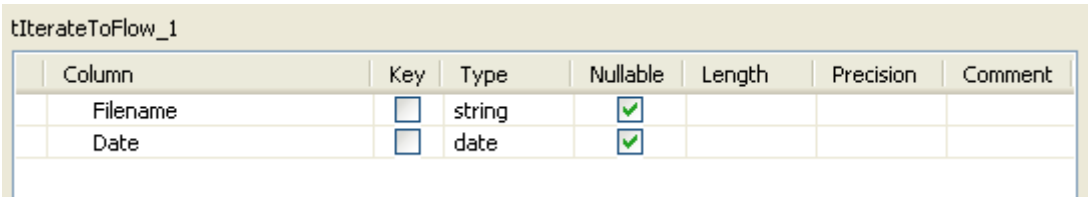
Component family	Orchestration	
Function	**tIterateToFlow** transforms a list into a data flow that can be processed.	
Purpose	Allows to transform non processable data into processable flow.	
Basic settings	*Schema type* and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the Repository. In the case of **tIterateToFlow**, the schema is to be defined
		Built-in: The schema will be created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused in various projects and job designs. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	Column	Type in a name for the columns to be created
	Value	Press **Ctrl+Space** bar to access all available variables either global or user-defined.
Usage	This component is not startable (green background) and it requires an output component.	

Scenario: Transforming a list of files as data flow

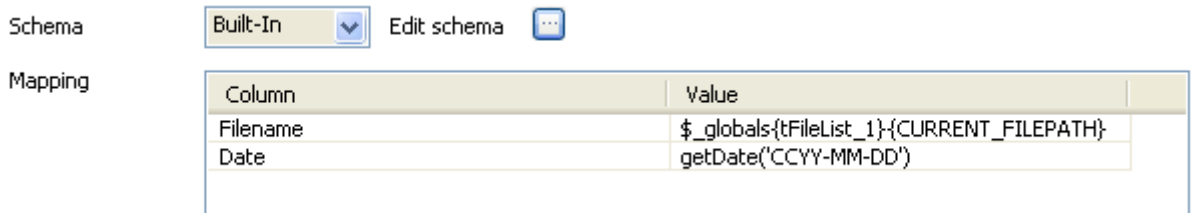
The following scenario describes a Job that iterates on a list of files, picks up the filename and current date and transforms this into a flow, that gets displayed on the console.

- Drop the following components: **tFileList**, **tIterateToFlow** and **tLogRow** from the **Palette** to the design workspace.
- Connect the **tFileList** to the **tIterateToFlow** using an **iterate** link and connect the Job to the **tLogRow** using a **Row main** connection.
- In the **tFileList Component** view, set the directory where the list of files is stored.

- In this example, the files are three simple .txt files held in one directory: *Countries*.
- No need to care about the case, hence clear the **Case sensitive** check box.
- Leave the **Include Subdirectories** check box unchecked.
- Then select the **tIterateToFlow** component et click **Edit Schema** to set the new schema



- Add two new columns: *Filename* of **String** type and *Date* of **date** type. Make sure you define the correct pattern in Java.
- Click **OK** to validate.
- Notice that the newly created schema shows on the **Mapping** table.



- In each cell of the **Value** field, press **Ctrl+Space bar** to access the list of global and user-specific variables.
- For the *Filename* column, use the global variable: (`$_globals{tFileList_1}{CURRENT_FILEPATH}`. It retrieves the current filepath in order to catch the name of each file, the Job iterates on.
- For the *Date* column, use the Talend routine: `Date.GetDate` (Perl) or `TalendDate.getCurrentDate()` (in Java)
- Then on the **tLogRow** component view, select the **Print values in cells of a table** check box.
- Save your Job and press **F6** to execute it.

```
Starting job Iteratetoflow at 12:04 03/10/2007.
.-------------------------------------------------.
|                    tLogRow_1                    |
+------------------------------+------------+
| Filename                     | Date       |
+------------------------------+------------+
| D:\Input\Countries\in-01.txt | 2007-10-03 |
| D:\Input\Countries\in-02.txt | 2007-10-03 |
| D:\Input\Countries\in-03.txt | 2007-10-03 |
'------------------------------+------------'
Job Iteratetoflow ended at 12:04 03/10/2007. [exit code=0]
```

The filepath displays on the *Filename* column and the current date displays on the *Date* column.

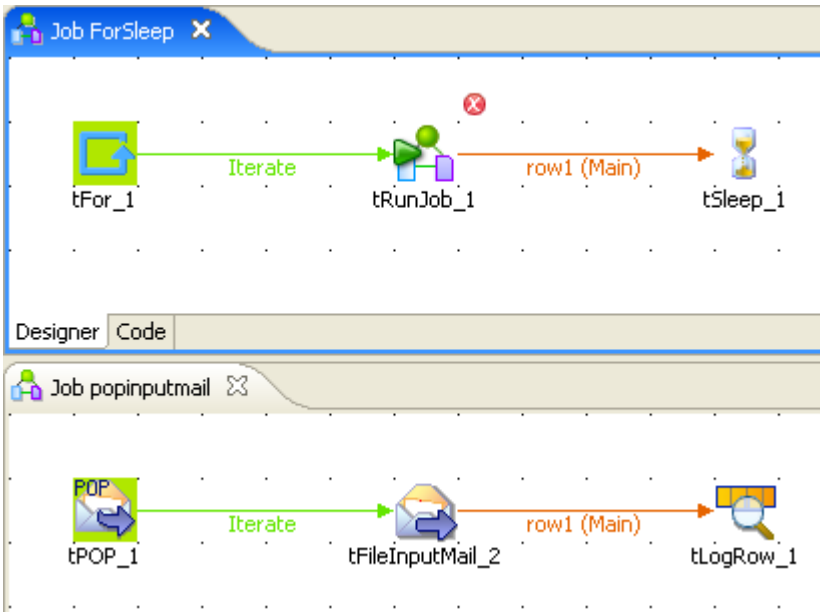
tLoop

tLoop Properties

Component family	Orchestration	
Function	**tLoop** iterates on a task execution.	
Purpose	**tLoop** allows to automatically execute a task or a Job based on a loop	
Basic settings	*Loop Type*	Select a type of loop to be carried out: either **For** or **While**. **For**: The task or Job is carried out for the defined number of iteration **While**: The task or Job is carried until the condition is met.
For	*From*	Type in the first instance number which the loop should start from. A start instance number of 2 with a step of 2 means the loop takes on every even number instance.
	To	Type in the last instance number which the loop should finish with.
	Step	Type in the step the loop should be incremented of. A step of 2 means every second instance.
While	*Declaration*	Type in an expression initiating the loop.
	Condition	Type in the condition that should be met for the loop to end.
	Iteration	Type in the expression showing the operation to be performed at each loop.
Usage	**tLoop** is to be used as a start component and can only be used with an iterate connection to the next component.	
Limitation	n/a	

Scenario: Job execution in a loop

This scenario describes a Job composed of a parent Job and a child Job. The parent Job implements a loop which executes n times a child Job, with a pause between each execution.



- In the parent Job, drop a **tLoop**, a **tRunJob** and a **tSleep** component from the **Palette** to the design workspace.
- Connect the **tLoop** to the **tRunJob** using an **Iterate** connection.
- Then connect the **tRunJob** to a **tSleep** component using a **Row** connection.
- On the child Job, drop the following components: **tPOP**, **tFileInputMail** and **tLogRow** the same way.
- On the **Basic settings** panel of the **tLoop** component, type in the instance number to start from (*1*), the instance number to finish with (*5*) and the step (*1*)
- On the **Basic settings** panel of the **tRunJob** component, select the child Job in the list of stored Jobs offered. In this example: *popinputmail*
- Select the context if relevant. In this use case, the context is *default* with no variables stored.
- In the **tSleep Basic settings** panel, type in the time-off value in second. In this example, type in *3 seconds* in the **Pause** field.
- Then in the child Job, define the connection parameters to the pop server, on the **Basic settings** panel.
- In the **tFileInputMail Basic settings** panel, select a global variable as **File Name**, to collect the current file in the directory defined in the **tPOP** component. Press **Ctrl+Space bar** to access the variable list. In this example, the variable to be used is:
  `$_globals{tPOP_1}{CURRENT_FILEPATH}` (for Perl)
  `((String)globalMap.get("tPOP_1_CURRENT_FILEPATH"))` (for Java)
- Define the **Schema**, for it to include the mail element to be processed, such as *author*, *topic*, *delivery date* and *number of lines*.
- In the **Mail Parts** table, type in the corresponding **Mail part** for each column defined in the schema. ex: *author* comes from the *From* part of the email file.
- Then connect the **tFileInputMail** to a **tLogRow** to check out the execution result on the **Run** view.

- Press **F6** to run the Job.

tPostjob

tPostjob Properties

Component family	Orchestration	
Function	**tPostjob** starts the execution of a postjob.	
Purpose	**tPostjob** triggers a task required after the execution of a Job	
Usage	**tPostjob** is a start component and can only be used with an iterate connection to the next component.	
Limitation	n/a	

For more information about the **tPostjob** component, see *Prejob and postjob parts* of **Talend Open Studio User Guide**.

Related scenario

No scenario is available for this component yet.

tPrejob

tPrejob Properties

Component family	Orchestration	
Function	**tPrejob** starts the execution of a prejob.	
Purpose	**tPrejob** triggers a task required for the execution of a job	
Usage	**tPrejob** is a start component and can only be used with an iterate connection to the next component.	
Limitation	n/a	

For more information about the **tPrejob** component, see *Prejob and postjob parts* of **Talend Open Studio** User Guide.

Related scenario

No scenario is available for this component yet.

tReplicate

tReplicate Properties

Component family	Orchestration	
Function	Duplicate the incoming schema into two identical output flows.	
Purpose	Allows to perform different operations on the same schema.	
Basic settings	*Schema type* and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the Repository.
		Built-in: The schema will be created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused in various projects and job designs. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
Usage	This component is not startable (green background), it requires an Input component and an output component.	

Related scenario

For use case showing this component in use, see *tReplaceList on page 117*.

tSleep

tSleep Properties

Component family	Orchestration	
Function	**tSleep** implements a time off in a job execution.	
Purpose	Allows to identify possible bottlenecks using a time break in the Job for testing or tracking purpose. In production, it can be used for any needed pause in the Job to feed input flow for example.	
Basic settings	*Pause (in second)*	Time in second the job execution is stopped for.
Usage	**tSleep** component is generally used as a middle component to make a break/pause in the Job, before resuming the Job.	
Limitation	n/a	

Related scenarios

For use cases in relation with **tSleep**, see **tLoop** *Scenario: Job execution in a loop on page 690*.

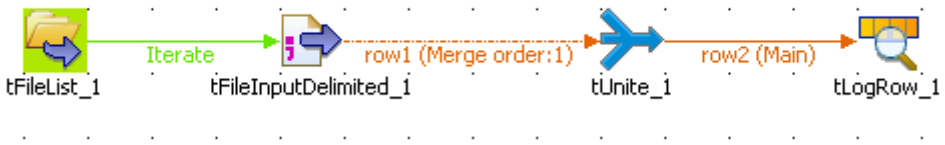
tUnite

tUnite Properties

Component family	Orchestration	
Function	Merges data from various sources, based on a common schema.	
Purpose	Centralize data from various and heterogeneous sources.	
Basic settings	*Schema type* and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the Repository.
		Built-in: The schema will be created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused in various projects and job designs. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
Usage	This component is not startable and requires one or several input components and an output component.	

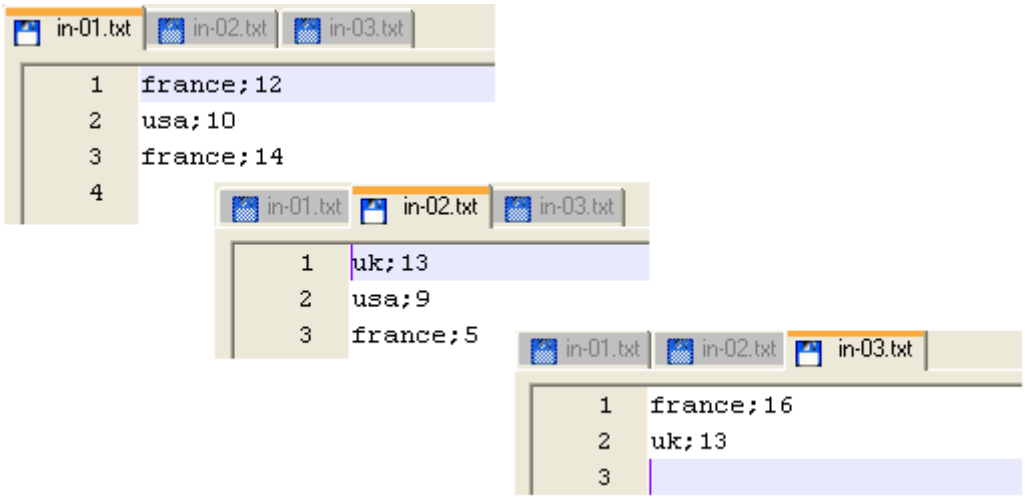
Scenario: Iterate on files and merge the content

The following Job iterates on a list of files then merges their content and diplays the final 2-column content on the console.

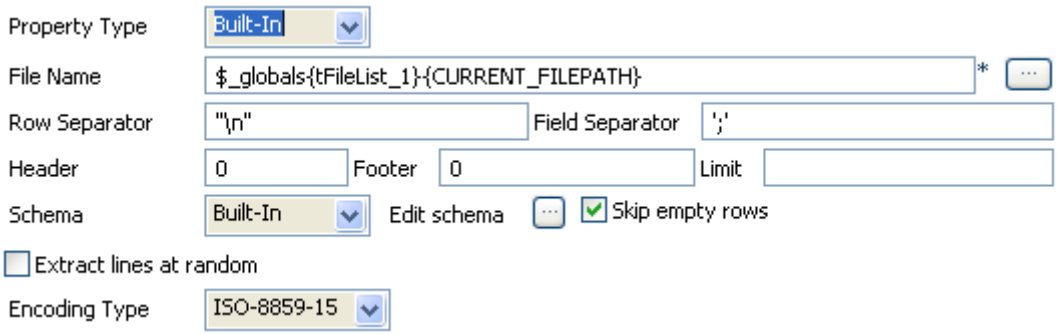


- Drop the following components onto the design workspace: **tFileList**, **tFileInputDelimited**, **tUnite** and **tLogRow**.
- Connect the **tFileList** to the **tFileInputDelimited** using an **iterate** connection and connect the other component using a **row main** link.
- In the **tFileList Basic settings** view, browse to the directory, where the files to merge are stored.

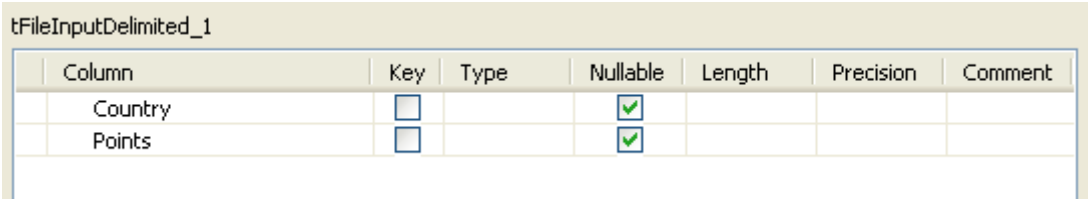
- As **Filemask**, type in `*.txt` as all files to be merged are of this type.
- The **Case Sensitive** box is selected by default. No need to clear it.
- The files are pretty basic and contain a list of countries and their respective score.



- Select the **tFileInputdelimited** component, and display this component's **Basic settings** view.
- In this use case, the input files' connection properties are not centrally stored in the **Repository**, therefore select **Built-In** as **Property type** and set every single field manually.



- To fill in the **File Name** field, use the **Ctrl+Space bar** combination to access the variable completion list. To process all files from the directory defined in the **tFileList**, select `$_globals{tFileList_1}{CURRENT_FILEPATH}` on the global variable list (in Perl).
- Keep the default setting for the **Row** and **Field separators** as well as the other fields.
- Click the **Edit Schema** button and set manually the 2-column schema to reflect the input files' content.



- For this example, the 2 columns are *Country* and *Points* .They are both **nullable**.
- Click **OK** to validate the setting and accept to propagate the schema throughout the Job.
- Then select the **tUnite** component and display the **Component** view. Notice that the output schema strictly reflects the input schema and is read-only.
- In the **tLogRow** Component view, select the **Print values in cells of the table** check box to display properly the output values.
- Save the Job and press **F6** to execute it.

```
Starting job Unite at 11:16 02/10/2007.
.-------------------.
|     tLogRow_1     |
+---------+---------+
| Country | Points  |
+---------+---------+
| france  |      12 |
| usa     |      10 |
| france  |      14 |
| uk      |      13 |
| usa     |       9 |
| france  |       5 |
| france  |      16 |
| uk      |      13 |
'---------+---------'
Job Unite ended at 11:16 02/10/2007. [exit code=0]
```

The console shows the data from the various files, merged into one single table. This uniformized output can then be aggregated to set

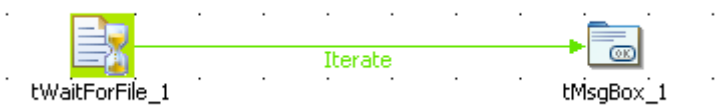
tWaitForFile

tWaitForFile properties

Component family	Orchestration	
Function	**tWaitForFile** component iterates on a given folder for file insertion or deletion then triggers a subjob to be executed when the condition is met.	
Purpose	This component allows a subjob to be triggered given a condition linked to file presence or removal.	
Basic settings	*Wait at each iteration (in seconds)*	Set the time interval in seconds between each check for the file.
	Max. iterations (infinite if empty)	Number of checks for file before the jobs times out.
	Directory to scan	Name of the folder to be checked for insert or removal
	File mask	Mask of the file to be searched for insertion or removal.
	Trigger action when rowcount is	Select the condition to be met for the action to be carried out: **A file is created** A file is deleted
	Then	Select the action to be carried out: either stop the iterations when the condition is met or continue the loop until the end of the max iteration number.
Usage	This component plays the role of the start (or trigger) component of the subjob which gets executed under the condition described. Therefore this component requires a subjob to be connected to via an Iterate link.	
Limitation	n/a	

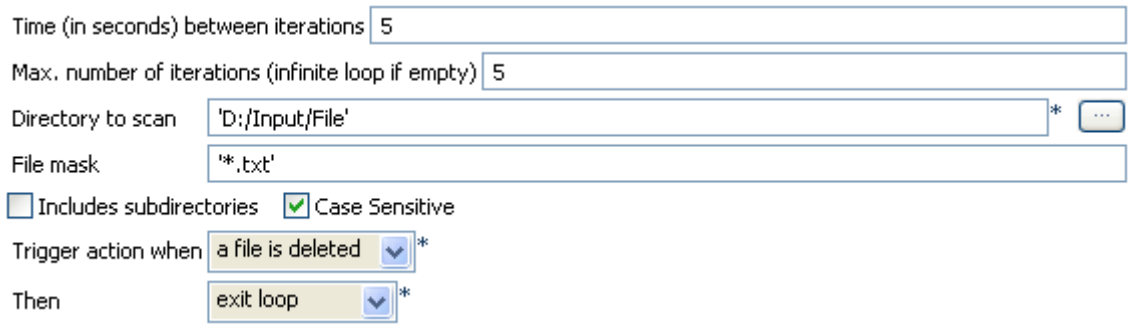
Scenario: Waiting for a file to be removed

This scenario describes a Job scanning a directory and waiting for a file to be removed from this directory, in order for a subjob to be executed. When the condition of file removal is met, then the subjob simply displays a message box showing the file being removed.

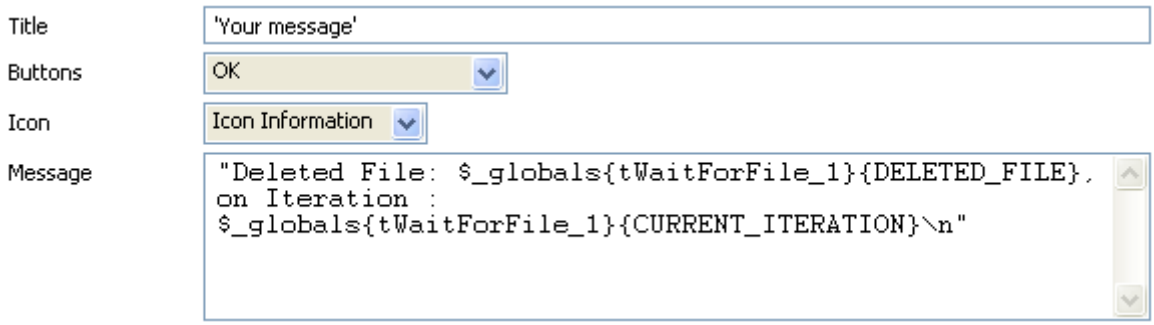


- This use case only requires two components from the **Palette**: **tWaitForFile** and **tMsgbox**

- Click and place these components on the design workspace and connect them using an **Iterate** link to implement the loop.
- Then select the **tWaitForFile** component, and on the Basic Settings view of the Components tab, set the condition and loop properties:

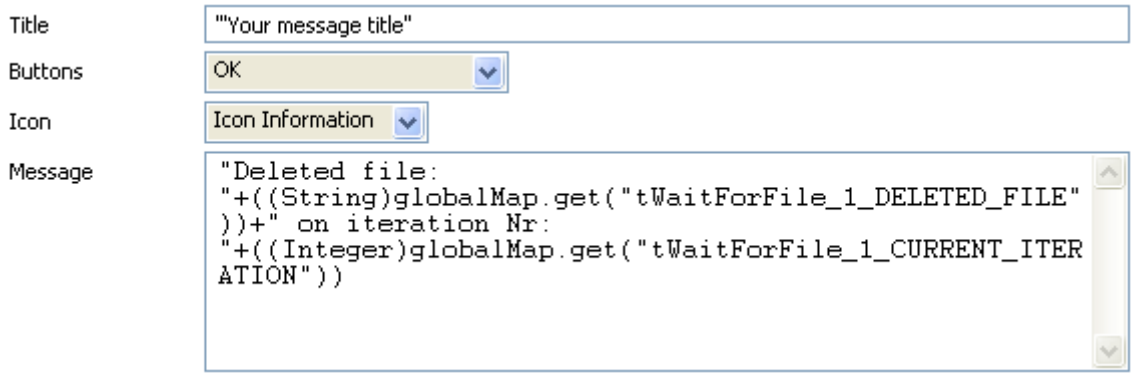


- In the **Wait at each iteration** field, set the time in seconds you want to wait before the next iteration starts. In this example, the directory will be scanned every *5* seconds.
- In the **Max iterations** field, fill out the number of iterations max you want to have before the whole Job is forced to end. In this example, the directory will be scanned a maximum of 5 times.
- In the **Directory to scan** field, type in the path to the folder to scan.
- In the **Trigger action when** field, select the condition to be met, for the subjob to be triggered. In this use case, the condition is a file is deleted (or moved) from the directory.
- In the **Then** field, select the action to be carried out when the condition is met before the number of iteration defined is reached. In this use case, as soon as the condition is met, the loop should be ended.
- Then set the subjob to be executed when the condition set is met. In this use case, the subjob simply displays a message box.
- Select the **tMsgBox** component, and on the **Basic Setting** view of the **Component** tab, set the message to be displayed.
- Fill out the **Title** and **Message** fields.
- Select the type of **Buttons** and the **Icon**

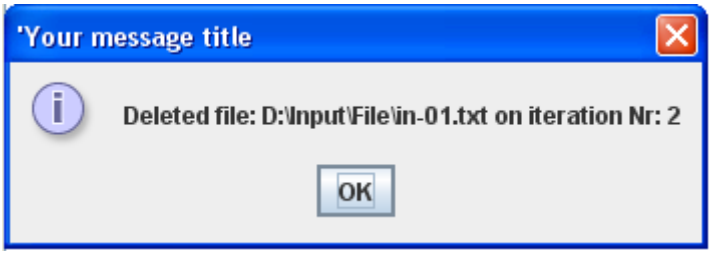


- In the **Message** field, you can write any type of message you want to display and use global variables available in the auto-completion list via Ctrl+Space combination.

- For example, in Perl, the message used for this use case is: `"Deleted File: $_globals{tWaitForFile_1}{DELETED_FILE}, on Iteration : $_globals{tWaitForFile_1}{CURRENT_ITERATION}\n"`
- The equivalent Java message is: `"Deleted file: "+((String)globalMap.get("tWaitForFile_1_DELETED_FILE"))+" on iteration Nr:"+((Integer)globalMap.get("tWaitForFile_1_CURRENT_ITERATION"))`



Then execute the Job via the **F6** key. While the loop is executing, remove a file from the location defined. The message pops up and shows the defined message.



tWaitForSqlData

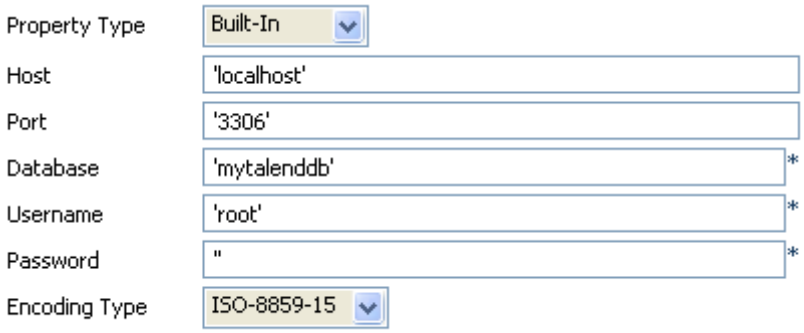
tWaitForSqlData properties

Component family	Orchestration	
Function	**tWaitForSqlData** component iterates on a given connection for insertion or deletion of rows and triggers a subjob to be executed when the condition is met.	
Purpose	This component allows a subjob to be triggered given a condition linked to sql data presence.	
Basic settings	*Wait at each iteration (in seconds)*	Set the time interval in seconds between each check for the sql data.
	Max. iterations (infinite if empty)	Number of checks for sql data before the Jobs times out.
	Use an existing connection/Component List	A connection needs to be open to allow the loop to check for sql data on the defined DB.
	Table to scan	Name of the table to be checked for insert or deletion
	Trigger action when rowcount is	Select the condition to be met for the action to be carried out: **Equal to** **Not Equal to** **Greater than** **Lower than** **Greater or equal to** **Lower or equal to**
	Then	Select the action to be carried out: either stop the iterations when the condition is met or continue the loop until the end of the max iteration number.
Usage	Although this component requires a Connection component to open the DB access, it plays also the role of the start (or trigger) component of the subjob which gets executed under the condition described. Therefore this component requires a subjob to be connected to via an Iterate link.	
Limitation	n/a	

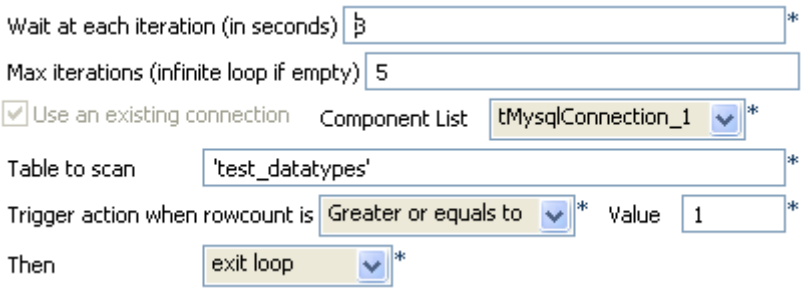
Scenario: Waiting for insertion of rows in a table

This scenario describes a Job reading a DB table and waiting for data to be put in this table in order for a subjob to be executed. When the condition of the data insertion in the table is met, then the subjob performs a Select* on the table and simply displays the content of the inserted data onto the standard console. This use case is presented in Perl, but there are no difference in setting if you implement it in Java.

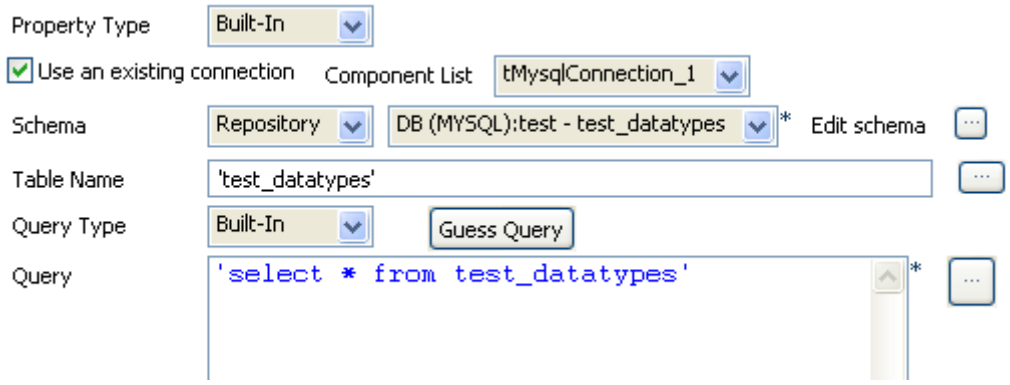
- Drop the following components from the **Palette** onto the design workspace: **tMySqlConnection**, **tWaitForSqlData**, **tMysqlInput**, **tLogRow**.
- Connect the **tMysqlConnection** component to the **tWaitforSqlData** using an **OnSubjobOK** link, available on the right-click menu.
- Then connect the **tWaitForSqlData** component to the subjob using an **Iterate** link as no actual data is transferred in this part. Indeed, simply a loop is implemented by the **tWaitForSqlData** until the condition is met.
- On the subjob to be executed if the condition is met, a **tMysqlInput** is connected to the standard console component, **tLogRow**. As the connection passes on data, use a Row main link.
- Now, set the connection to the table to check at regular intervals. On the **Basic Settings** view of the **tMySqlConnection Component** tab, set the DB connection properties



- Fill out the **Host**, **Port**, **Database**, **Username**, **Password** fields to open the connection to the Database table.
- Select the relevant **Encoding** if needed.
- Then select the **tWaitForSqlData** component, and on the **Basic Setting** view of the **Component** tab, set its properties.
- In the **Wait at each iteration** field, set the time in seconds you want to wait before the next iteration starts.

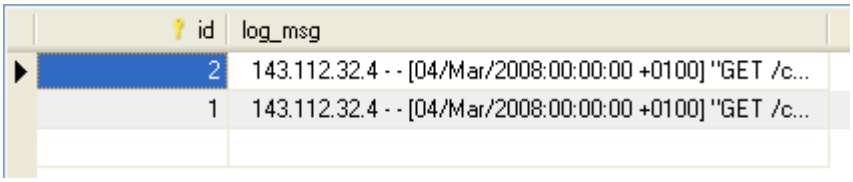


- In the **Max iterations** field, fill out the number of iterations max you want to have before the whole Job is forced to end.
- The **tWaitForSqlData** component requires a connection to be open in order to loop on the defined number of iteration. Select the relevant connection (if several) in the **Component List** combo box.
- In the **Table to scan** field, type in the name of the table in the DB to scan.In this example: *test_datatypes*.
- In the **Trigger action when rowcount is** and **Value** fields, select the condition to be met, for the subjob to be triggered. In this use case, the number of rows in the scanned table should *be greater or equal to 1*.
- In the **Then** field, select the action to be carried out when the condition is met before the number of iteration defined is reached. In this use case, as soon as the condition is met, the loop should be ended.
- Then set the subjob to be executed when the condition set is met. In this use case, the subjob simply selects the data from the scanned table and displays it on the console.
- Select the **tMySqlInput** component, and on the **Basic Setting** view of the **Component** tab, set the connection to the table.



- If the connection is set in the Repository, select the relevant entry on the list. Or alternatively, select the **Use an existing connection** check box and select the relevant connection component on the list.
- In this use case, the schema corresponding to the table structure is stored in the **Repository**.
- Fill out the **Table Name** field with the table the data is extracted from, *Test_datatypes*.
- Then in the **Query** field, type in the Select statement to extract the content from the table.
- No particular setting is required in the **tLogRow** component for this use case.

Then before executing the Job, make sure the table to scan (*test_datatypes*) is empty, in order for the condition (greater or equal to 1) to be met. Then execute the Job by pressing the **F6** key on your keyboard. Before the end of the iterating loop, feed the test_datatypes table with one or more rows in order to meet the condition.



id	log_msg
2	143.112.32.4 - - [04/Mar/2008:00:00:00 +0100] "GET /c...
1	143.112.32.4 - - [04/Mar/2008:00:00:00 +0100] "GET /c...

The Job ends when this table insert is detected during the loop, and the table content is thus displayed on the console.

```
Starting job tWaitForSql at 16:55 06/03/2008.
2| 143.112.32.4 - - [04/Mar/2008:00:00:00 +0100] "GET
/components/none HTTP/1.1" 404 354 "-" "Mozilla/4.0 (compatible
1| 143.112.32.4 - - [04/Mar/2008:00:00:00 +0100] "GET
/components/none HTTP/1.1" 404 354 "-" "Mozilla/4.0 (compatible
Job tWaitForSql ended at 16:55 06/03/2008. [exit code=0]
```


Processing components

This chapter details the major components that you can find in **Processing** group of the **Palette** of **Talend Open Studio**.

The Processing family gathers components that help you to perform all types of processing tasks on data flows, including aggregation, mapping, transformation, denormalizing, filtering and so on.

tAggregateRow

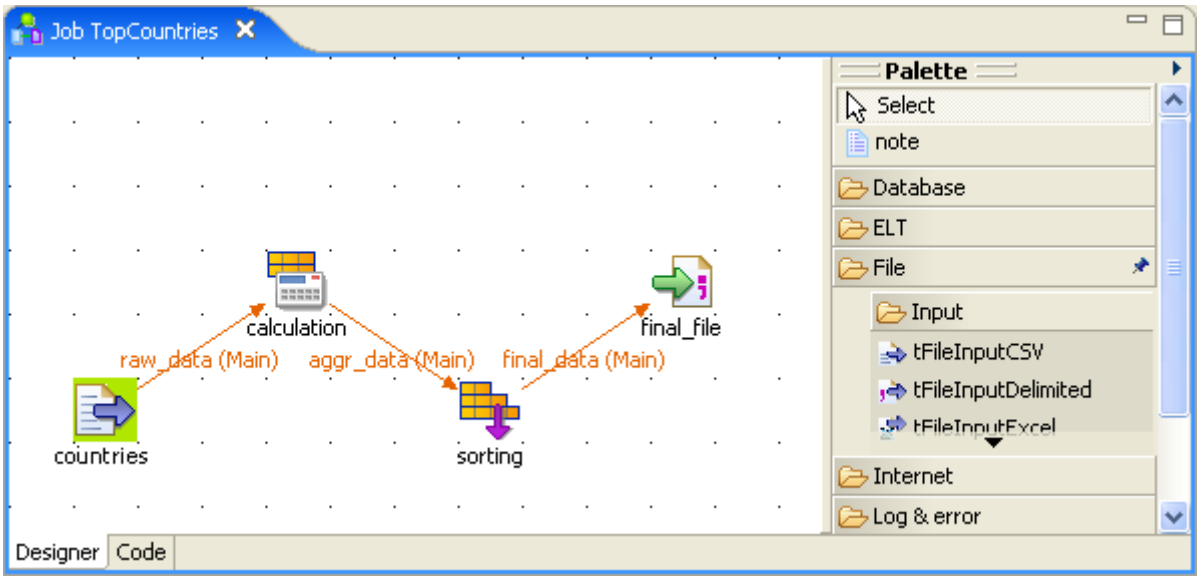
tAggregateRow properties

Component family	Processing	
Function	**tAggregateRow** receives a flow and aggregates it based on one or more columns. For each output line, are provided the aggregation key and the relevant result of set operations (min, max, sum...).	
Purpose	Helps to provide a set of metrics based on values or calculations.	
Basic settings	*Schema type* and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the Repository. Click **Edit Schema** to make changes to the schema. Note that if you make changes, the schema automatically becomes built-in.
		Built-in: The schema will be created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused in various projects and job flowcharts. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	Group by	Define the aggregation sets, the values of which will be used for calculations.
		Output Column: Select the column label in the list offered based on the schema structure you defined. You can add as many output columns as you wish to make more precise aggregations. Ex: Select Country to calculate an average of values for each country of a list or select Country and Region if you want to compare one country's regions with another country' regions.
		Input Column: Match the input column label with your output columns, in case the output label of the aggregation set needs to be different.
	Operations	Select the type of operation along with the value to use for the calculation and the output field.
		Output Column: Select the destination field in the list.
		Function: Select the operator among: count, min, max, avg, sum, first, last, list, list(objects), count(distinct), standard deviation.
		Input column: Select the input column from which the values are taken to be aggregated.

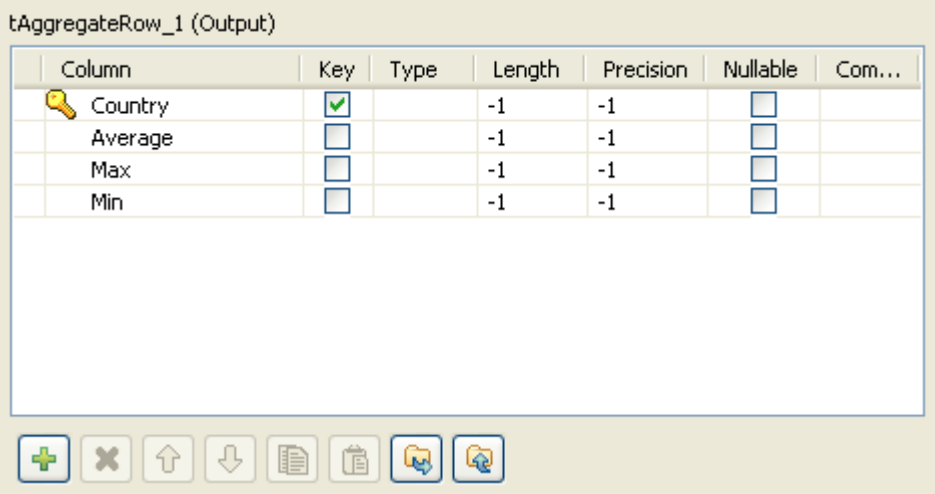
		Ignore null values: Select the check boxes corresponding to the names of the columns for which you want the NULL value to be ignored.
Usage	This component handles flow of data therefore it requires input and output, hence is defined as an intermediary step. Usually the use of **tAggregateRow** is combined with the **tSortRow** component.	
Limitation	n/a	

Scenario: Aggregating values and sorting data

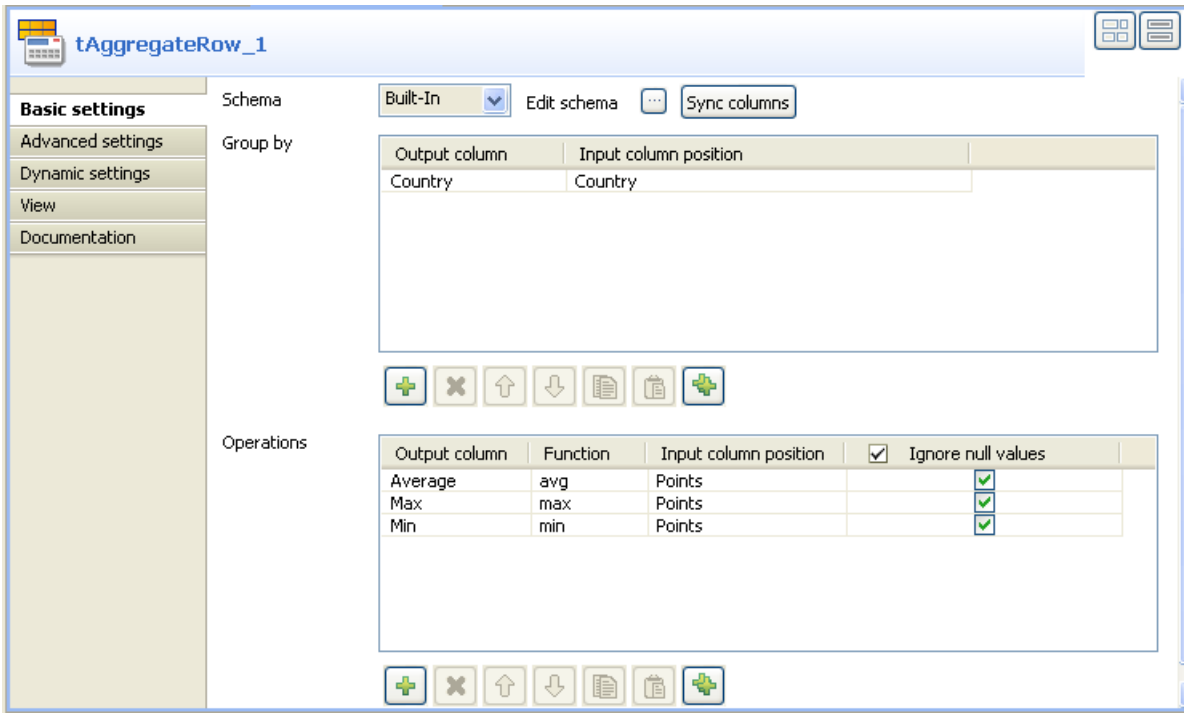
The following scenario describes a four-component Job. As input component, a CSV file contains countries and notation values to be sorted by best average value. This component is connected to a **tAggregateRow** operator, in charge of the average calculation then to a **tSortRow** component for the ascending sort. The output flow goes to the new csv file.



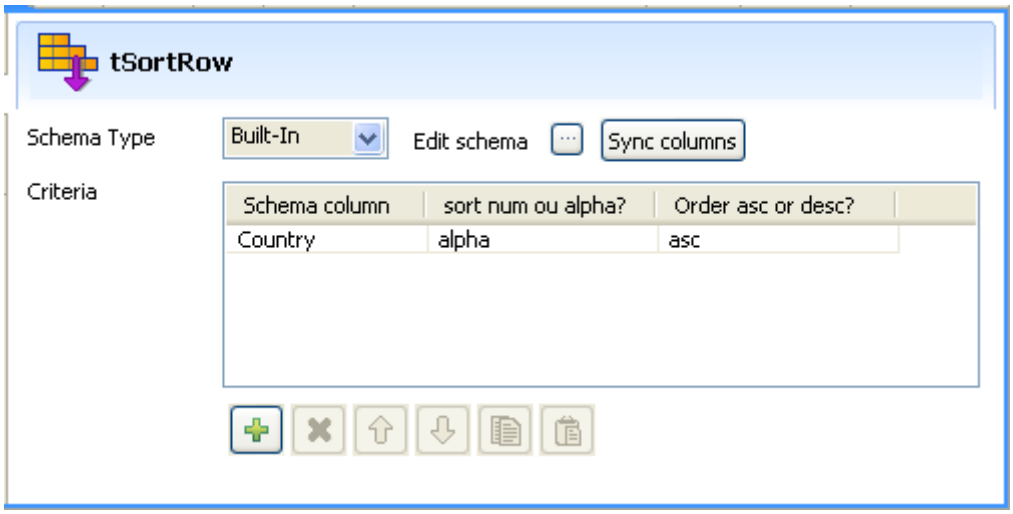
- From the **File** folder in the **Palette**, drop a **tFileInputCSV** component to the design workspace.
- Click on the label and rename it as *Countries*. Or rename it from the **View** tab panel
- In the **Basic settings** tab panel of this component, define the filepath and the delimitation criteria. Or select the metadata file in the repository if it exists.
- Click on **Edit schema...** and set the columns: *Countries* and *Points* to match the file structure. If your file description is stored in the Metadata area of the Repository, the schema is automatically uploaded when you click on **Repository** in **Schema type** field.
- Then from the **Processing** folder in the **Palette**, drop a **tAggregateRow** component to the design workspace. Rename it as *Calculation*.
- Connect *Countries* to *Calculation* via a right-click and select **Row > Main**.
- Double-click on *Calculation* (**tAggregateRow** component) to set the properties. Click on **Edit schema** and define the output schema. You can add as many columns as you need to hold the set operations results in the output flow.



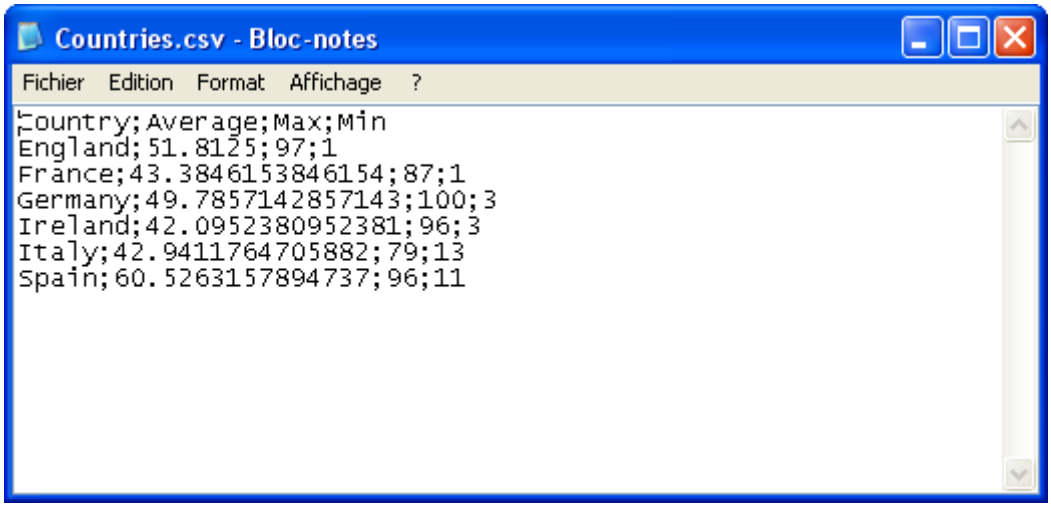
- In this example, we'll calculate the average notation value and we will display the max and the min notation for each country, given that each country holds several notations. Click OK when the schema is complete.
- To carry out the various set operations, back in the **Basic settings** panel, define the sets holding the operations in the **Group By** area. In this example, select **Country** as group by column. Note that the output column needs to be defined a key field in the schema. The first column mentioned as output column in the Group By table is the main set of calculation. All other output sets will be secondary by order of display.
- Choose the input column which the values will be taken from.
- Then fill in the various operations to be carried out. The functions are average, min, max for this use case. Select the input columns, where the values are taken from and select the check boxes in the **Ignore null values** list as needed.



- Drop a **tSortRow** component from the **Palette** onto the design workspace. For more information regarding this component , see *tSortRow properties on page 788.*
- Connect the **tAggregateRow** to this new component using a row main link.
- On the **Component** view of the **tSortRow** component, define the column the sorting is based on, the sorting type and order.



- In this case, the column to be sorted by is Country, the sort type is alphabetical and the order is ascending.
- Drop a **tFileOutputDelimited** from the **Palette** to the design workspace and define it to set the output flow.
- Connect the tSortRow component to this output component.
- In the Component view, enter the output filepath. Edit the schema if need be. In this case the delimited file is of csv type. And select the **Include Header** check box to reuse the schema column labels in your output flow.
- Press **F6** to execute the Job. The csv file thus created contains the aggregating result.



tAggregateSortedRow

tAggregateSortedRow properties

Component family	Processing	
Function	**tAggregateSortedRow** receives a sorted flow and aggregates it based on one or more columns. For each output line, are provided the aggregation key and the relevant result of set operations (min, max, sum...).	
Purpose	Helps to provide a set of metrics based on values or calculations. As the input flow is meant to be sorted already, the performance are hence greatly optimized.	
Basic settings	*Schema type* and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the Repository. Click **Edit Schema** to make changes to the schema. Note that if you make changes, the schema automatically becomes built-in.
		Built-in: The schema will be created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused in various projects and job flowcharts. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	Group by	Define the aggregation sets, the values of which will be used for calculations.
		Output Column: Select the column label in the list offered based on the schema structure you defined. You can add as many output columns as you wish to make more precise aggregations. Ex: Select Country to calculate an average of values for each country of a list or select Country and Region if you want to compare one country's regions with another country' regions.
		Input Column: Match the input column label with your output columns, in case the output label of the aggregation set needs to be different.
	Operations	Select the type of operation along with the value to use for the calculation and the output field.
		Output Column: Select the destination field in the list.
		Function: Select the operator among: count, min, max, avg, first, last.

		Input column: Select the input column from which the values are taken to be aggregated.
		Ignore null values: Select the check boxes corresponding to the names of the columns for which you want the NULL value to be ignored.
Usage	This component handles flow of data therefore it requires input and output, hence is defined as an intermediary step.	
Limitation	n/a	

Related scenario

For related use case, see **tAggregateRow** *Scenario: Aggregating values and sorting data on page 711*.

tConvertType

tConvertType properties

Component family	Processing	
Function	**tConvertType** allows specific conversions at run time from one **Talend** java type to another.	
Purpose	Helps to automatically convert one **Talend** java type to another and thus.avoid compiling errors.	
Basic settings	*Schema type* and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the Repository. Click **Edit Schema** to make changes to the schema. Note that if you make changes, the schema automatically becomes built-in.
		Built-in: You create and store the schema locally for only the current component. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: You have already created the schema and stored it in the Repository, and thus you can reuse it in various projects and job flowcharts. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	Auto Cast	This check box is selected by default. It performs an automatic java type conversion.
Usage	This component cannot be used as a start component as it requires an input flow to operate.	
Limitation	n/a	

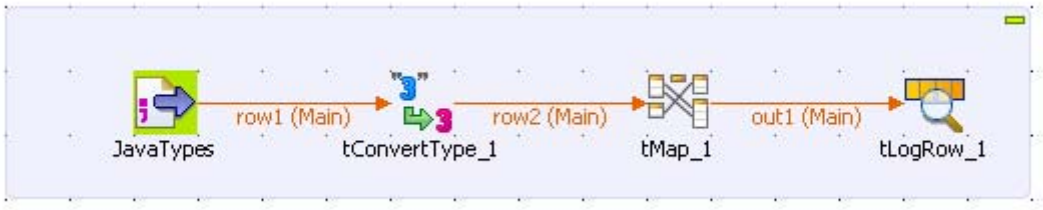
Scenario: Converting java types

This Java scenario describes a four-component Job where the **tConvertType** component is used to convert Java types in three columns, and a **tMap** is used to adapt the schema and have as an output the first of the three columns and the sum of the two others after conversion.

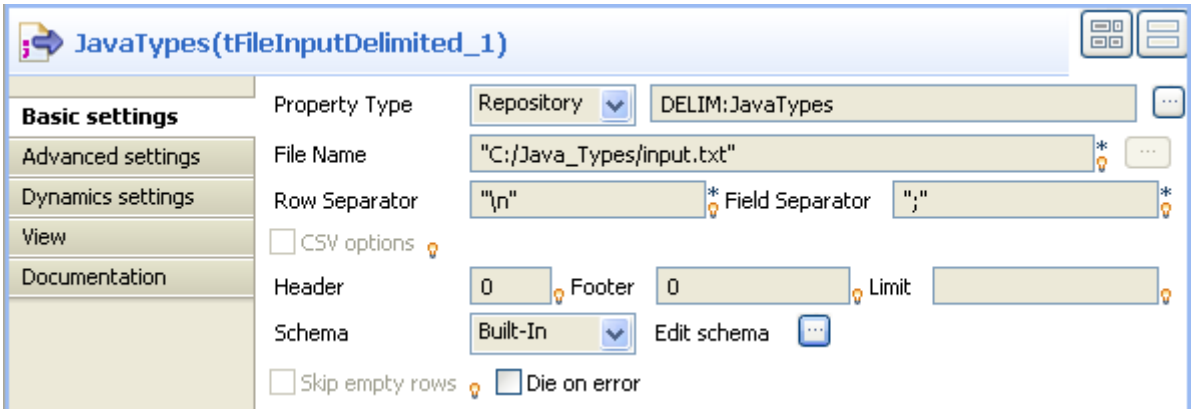
In this scenario, the input schemas for the input delimited file are stored in the repository, you can simply drag and drop the relevant file node from **Repository** - **Metadata** - **File delimited** onto the design workspace to automatically retrieve the **tFileInputDelimited** component's setting. For more information, see *Drop components from the Metadata node* in **Talend Open Studio** User Guide.

- Drop the following components from the **Palette** onto the design workspace: **tConvertType**, **tMap**, and **tLogRow**.

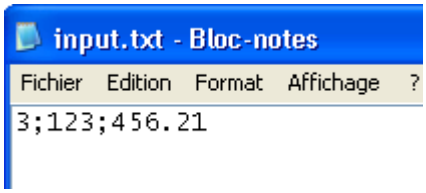
- In the Repository tree view, expand **Metadata** and from **File delimited** drag the relevant node, *JavaTypes* in this scenario, to the design workspace.
  The **[Components]** dialog box displays.
- From the component list, select **tFileInputDelimited** and click **Ok**.
  A **tFileInputComponent** called *Java types* displays in the design workspace.
- Connect the components using **Row Main** links.



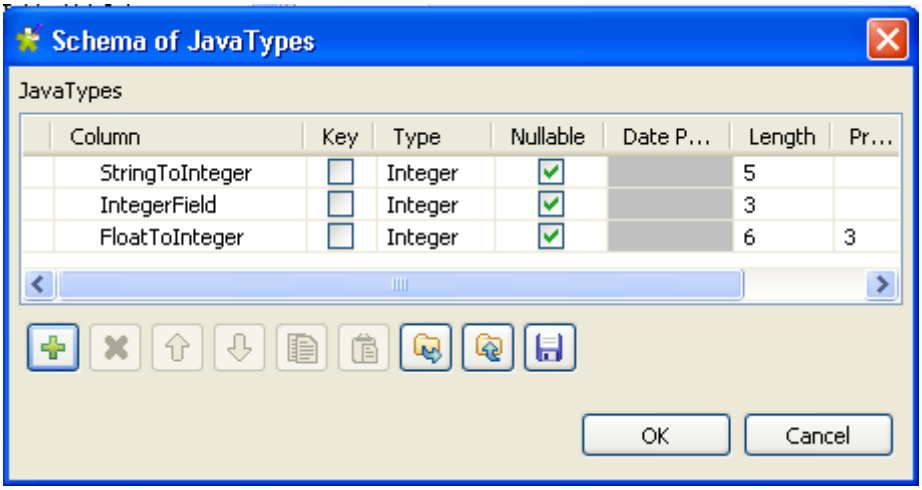
- In the design workspace, select **tFileInputDelimited** and click the **Component** tab to define its basic settings.
- In the **Basic settings** view, set **Property Type** to **Repository** since the file details are stored in the repository. The fields to follow are pre-defined using the fetched data.



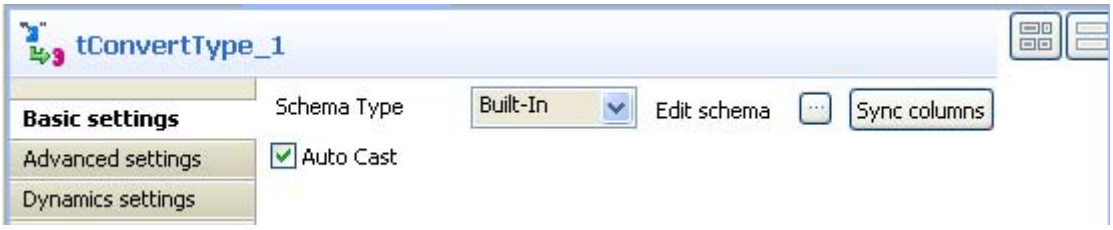
The input file used in this scenario is called *input*. It is a text file that holds string, integer, and float java types.



- In the **Basic settings** view, fill in all other fields as needed. For more information, see *tFileInputDelimited properties on page 494*. In this scenario, the header and the footer are not set and there is no limit for the number of processed rows.
- Click **Edit schema** to describe the data structure of this input file. In this scenario, the schema is made of three columns, *StringtoInteger, IntegerField*, and *FloatToInteger*.

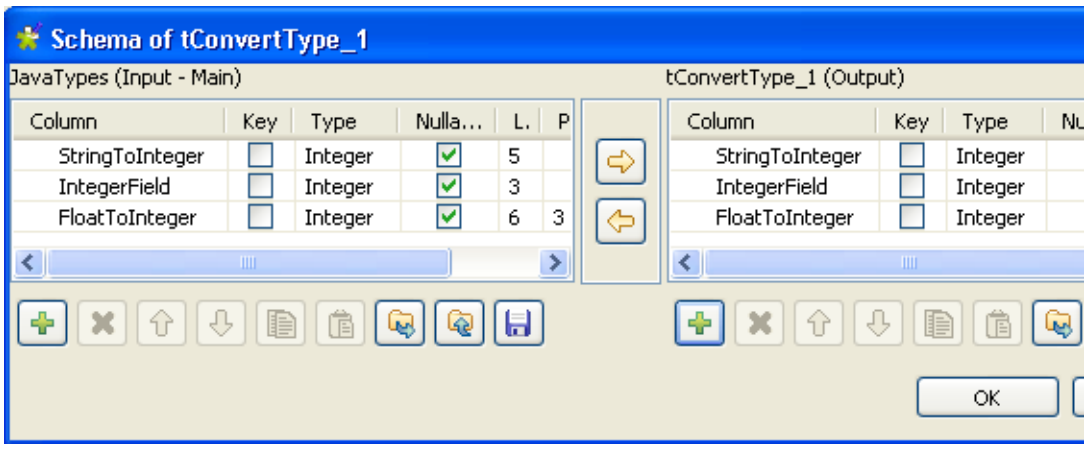


- Click **Ok** to close the dialog box.
- In the design workspace, select **tConvertType** and click the **Component** tab to define its basic settings.



- Set **Schema Type** to **Built in**, and click **Sync columns** to automatically retrieve the columns from the **tFileInputDelimited** component.
- If needed, click **Edit schema** to describe manually the data structure of this processing component.

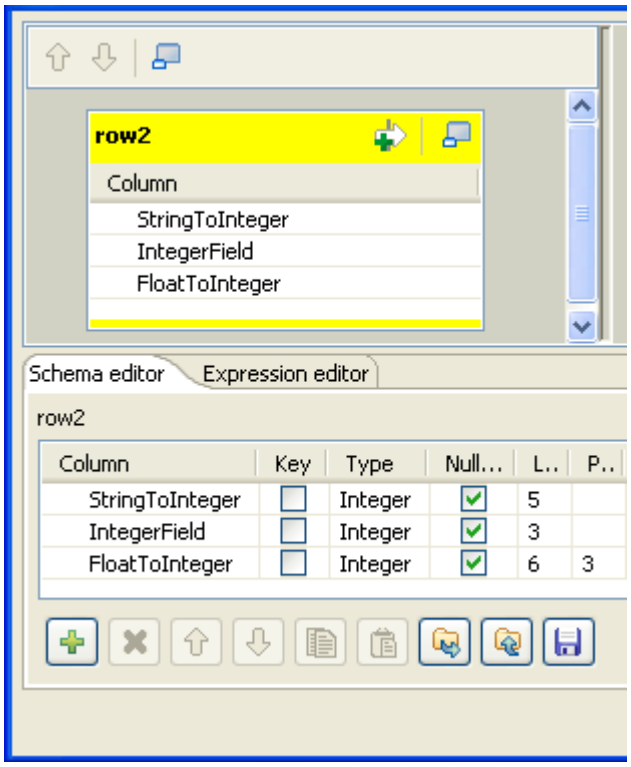
Usually, the **Auto Cast** check box is selected by default.



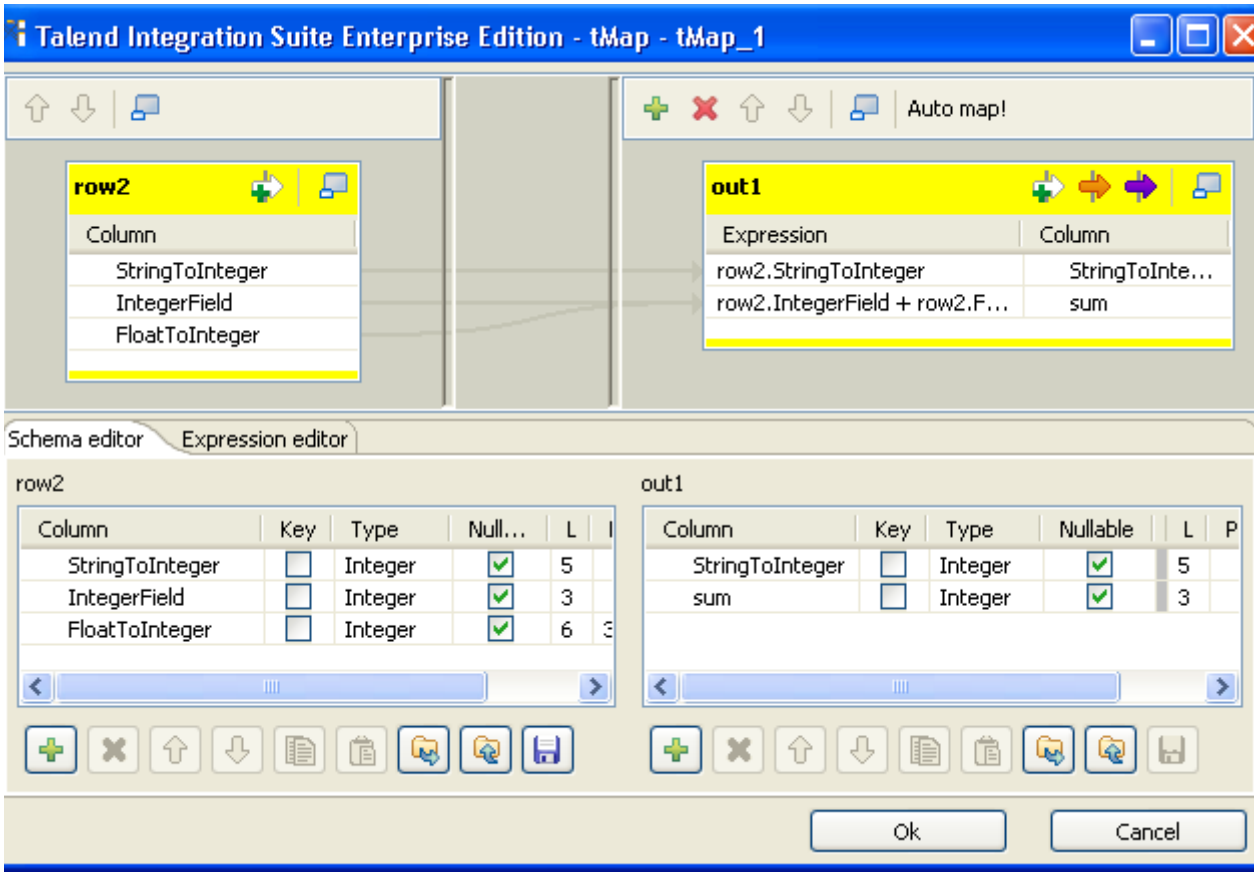
In this scenario, we want to convert a string type data into an integer type and a float type data into an integer type.

- Click **OK** to close the **[Schema of tConvertType]** dialog box.

- In the design workspace, double-click **tMap** to open the Map editor.
  The Map editor opens displaying the input metadata of the **tFileInputDelimited** component



- In the **Schema editor** panel of the Map editor, click the plus button of the output table to add two rows and define them as *StringToInteger* and *Sum*.
- In the Map editor, drag the *StringToInteger* row from the input table to the *StringToInteger* row in the output table.
- In the Map editor, drag each of the *IntegerField* and the *FloatToInteger* rows from the input table to the *Sum* row in the output table and click **OK** to close the Map editor.



- In the design workspace, select **tLogRow** and click the **Component** tab to define its basic settings. For more information, see *tLogRow on page 628*.
- Save your Job and press **F6** to execute it.

```
Starting job tConvertType at 18:03 19/11/2008.
.---------------+---.
|      tLogRow_1      |
|=---------------+---=|
|StringToInteger|sum|
|=---------------+---=|
|3              |579|
'---------------+---'

Job tConvertType ended at 18:03 19/11/2008. [ex
```

The string type data is converted into an integer type and displayed in the *StringToInteger* column on the console. The float type data is converted into an integer and added to the *IntegerField* value to give the addition result in the *Sum* column on the console.

tDenormalize

tDenormalize Properties

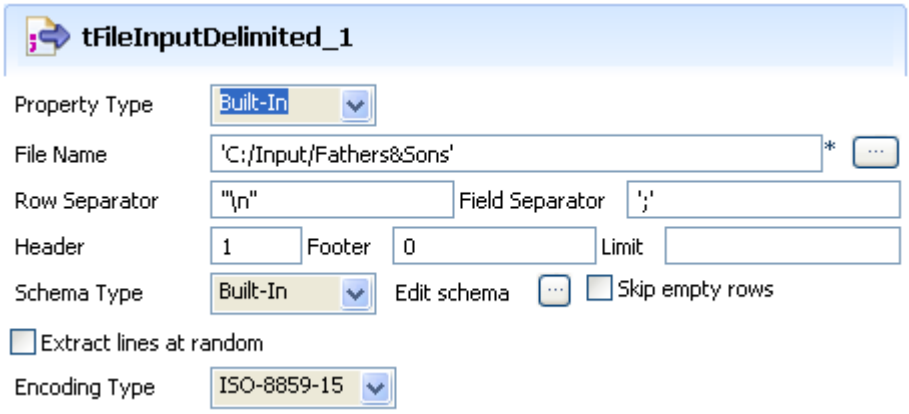
Component family	Processing/Fields	
Function	Denormalizes the input flow based on one column.	
Purpose	**tDenormalize** helps synthesize the input flow.	
Basic settings	*Schema type* and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the Repository. In this component, the schema is read-only.
		Built-in: The schema will be created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
Perl only	*Column to denormalize*	Select the column from the input flow which the normalization is based on (included in key)
Java only	*Group by*	Select one or several columns to be grouped. We recommend to remove unused columns from the schema before processing.
	Separator	Enter the separator which will delimit data in the denormalized flow.
Advanced settings		
Perl only	*Deduplicate items*	Removes duplicates when concatenating denormalized values.
Java only	*Merge the same value when concatenating*	Removes duplicates when concatenating denormalized values.
Usage	This component can be used as intermediate step in a data flow.	
Limitation	Note that this component may change the order of the input flow data in Java, and in Perl, when the **Deduplicate** check box is selected.	

Scenario 1: Denormalizing on one column in Perl

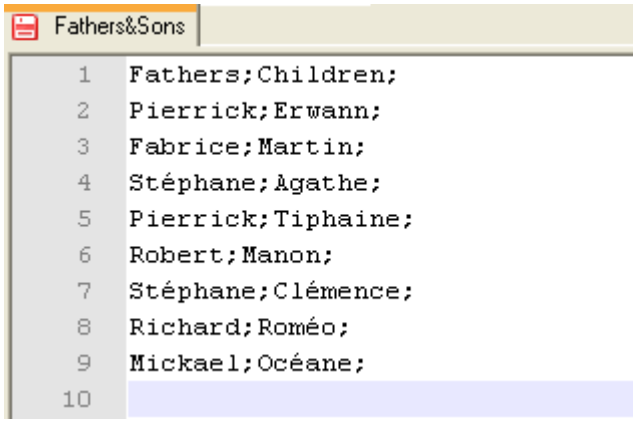
This scenario illustrates a Perl Job denormalizing one column in a delimited file.

- Drop the following components: **tFileInputDelimited**, **tDenormalize**, **tLogRow** from the **Palette** to the design workspace.

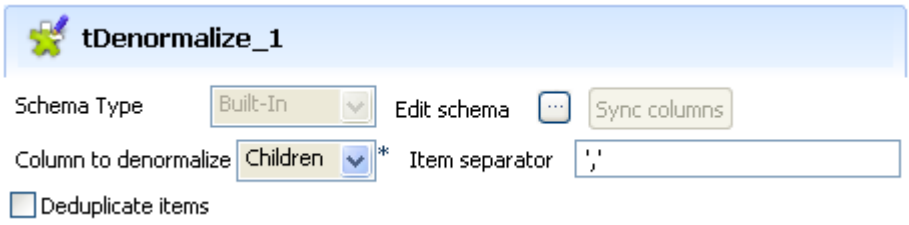
- Connect the components using **Row main** connections.
- On the **tFileInputDelimited** Component view, set the filepath to the file to be denormalized.



- Define the **Header**, **Row Separator** and **Field Separator** parameters.
- The input file schema is made of two columns, *Fathers* and *Children*.



- In the **Basic settings** of **tDenormalize**, define the column that contains multiple values to be grouped.
- In this use case, the column to denormalize is *Children*.



- Set the **Item Separator** to separate the grouped values. Beware as only one column can be denormalized.
- Select the **Deduplicate items** check box, if you know that some values to be grouped are strictly identical.
- Save your Job and press **F6** to execute it.

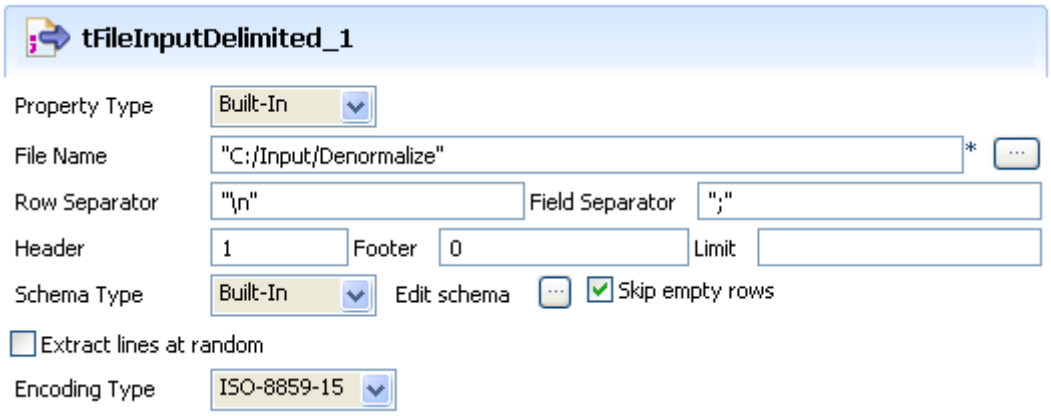
```
Starting job Denormalize at 20:39 03/07/2007.
Richard| Roméo
Stéphane| Agathe, Clémence
Mickael| Océane
Pierrick| Erwann, Tiphaine
Robert| Manon
Fabrice| Martin
Job Denormalize ended at 20:39 03/07/2007. [exi
```

All values from the column *Children* (set as column to denormalize) are grouped by their *Fathers* column. Values are separated by a comma.

Scenario 2: Denormalizing on multiple columns in Java

This scenario illustrates a Java Job denormalizing two columns from a delimited file.

- Drop the following components: **tFileInputDelimited**, **tDenormalize**, **tLogRow** from the **Palette** to the design workspace.
- Connect all components using a **Row main** connection.
- On the **tFileInputDelimited Basic settings** panel, set the filepath to the file to be denormalized.

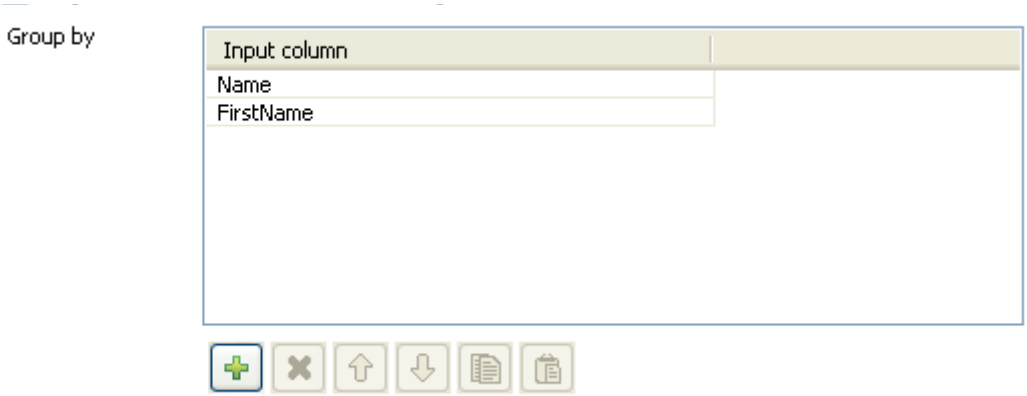


- Define the **Row** and **Field** separators, the **Header** and other information if required.
- The file schema is made of four columns including: **Name**, **FirstName**, **HomeTown**, **WorkTown**.

Denormalize

```
1   Name;FirstName;HomeCity;WorkCity
2   Pitt;Brad;Beverly Hills;Los Angeles
3   Pitt;Brad;Paris;London
4   Joli;Angelina;Berlin;Berlin
5   Joli;Angelina;Berlin;Los Angeles
6   Joli;Angelina;Los Angeles;Los Angeles
7   Willis;Bruce;Paris;Los Angeles
8   Willis;Bruce;Paris;Madrid
9   Willis;Bruce;Madrid;Madrid
10  Willis;Bruce;Roma;Dublin
11  Moore;Demi;New York;Paris
12  Moore;Demi;Rio de Janeiro;Los Angeles
13
```

- In the **tDenormalize** component **Basic settings**, select the columns that contain the repetition. These are the column which are meant to occur multiple times in the document. In this use case, *FirstName* and *Name* are the columns against which the denormalization is performed.
- Add as many line to the table as you need using the plus button. Then select the relevant columns in the drop-down list.



- Define the delimiter for concatenated values. In this case, the comma is used.
- Save your Job and press **F6** to execute it.

```
Starting job denormalize at 21:03 03/07/2007.
Pitt|Brad|Beverly Hills,Paris|Los Angeles,London
Willis|Bruce|Paris,Paris,Madrid,Roma|Los
Angeles,Madrid,Madrid,Dublin
Moore|Demi|New York,Rio de Janeiro|Paris,Los Angeles
Joli|Angelina|Berlin,Berlin,Los Angeles|Berlin,Los Angeles,Los
Angeles
Job denormalize ended at 21:03 03/07/2007. [exit code=0]
```

- The result shows the denormalized values concatenated using a comma.
- Back to the **tDenormalize** components **Basic settings**, select the **Deduplicate** check box to remove the duplicate occurrences.
- Save your Job again and press **F6** to execute it.

```
Starting job denormalize at 21:20 03/07/2007.
Pitt|Brad|Paris,Beverly Hills|Los Angeles,London
Willis|Bruce|Madrid,Paris,Roma|Madrid,Los Angeles,Dublin
Moore|Demi|Rio de Janeiro,New York|Los Angeles,Paris
Joli|Angelina|Los Angeles,Berlin|Los Angeles,Berlin
Job denormalize ended at 21:20 03/07/2007. [exit code=0]
```

This time, the console shows the results with no duplicate instances.

tDenormalizeSortedRow

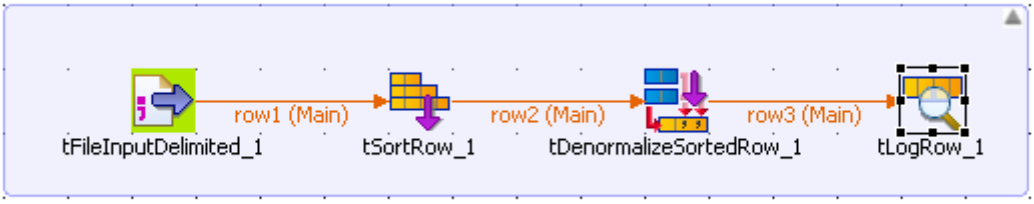
tDenormalizeSortedRow properties

Component family	Processing/Fields	
Function	**tDenormalizeSortedRow** combines in a group all input sorted rows. Distinct values of the denormalized sorted row are joined with item separators.	
Purpose	**tDenormalizeSortedRow** helps synthesizing sorted input flow to save memory.	
Basic settings	*Schema type* and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the Repository. Click **Edit Schema** to make changes to the schema. Note that if you make changes, the schema automatically becomes built-in. Click **Sync columns** to retrieve the schema from the previous component in the Job.
		Built-in: You create the schema and store it locally for the relevant component. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: You have already created the schema and stored it in the Repository. You can reuse it in various projects and job flowcharts. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	Input rows count	Enter the number of input rows.
	To denormalize	Enter the name of the column to denormalize.
Usage	This component handles flows of data therefore it requires input and output components.	
Limitation	n/a	

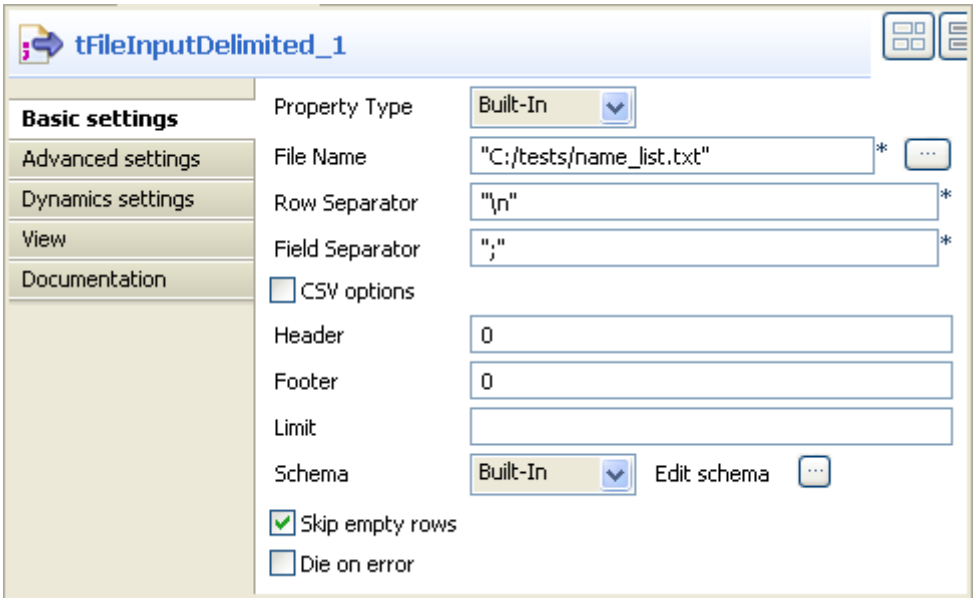
Scenario: Regrouping sorted rows

This Java scenario describes a four-component Job. It aims at reading a given delimited file row by row, sorting input data by sort type and order, denormalizing all input sorted rows and displaying the output on the **Run** log console.

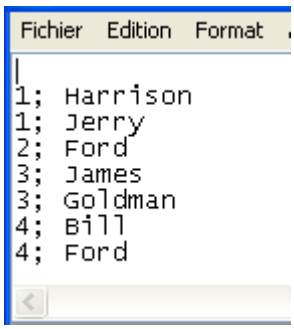
- Drop the following components from the **Palette** onto the design workspace: **tFileInputDelimited**, **tSortRow**, **tDenormalizeSortedRow**, and **tLogRow**.
- Connect the four components using **Row Main** links.



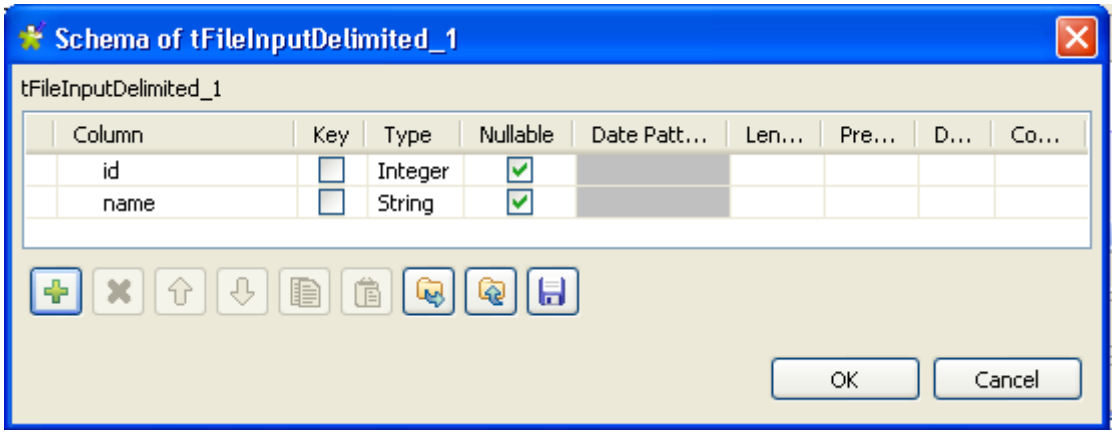
- In the design workspace, select **tFileInputDelimited**.
- Click the **Component** tab to define the basic settings for **tFileInputDelimited**.



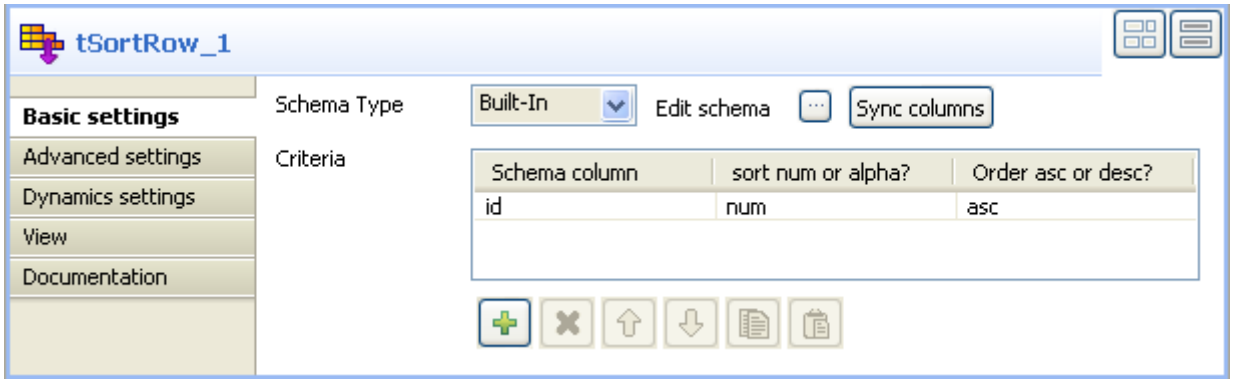
- Set **Property Type** to **Built-In**.
- Fill in a path to the processed file in the **File Name** field. The *name_list* file used in this example holds two columns, id and first name.



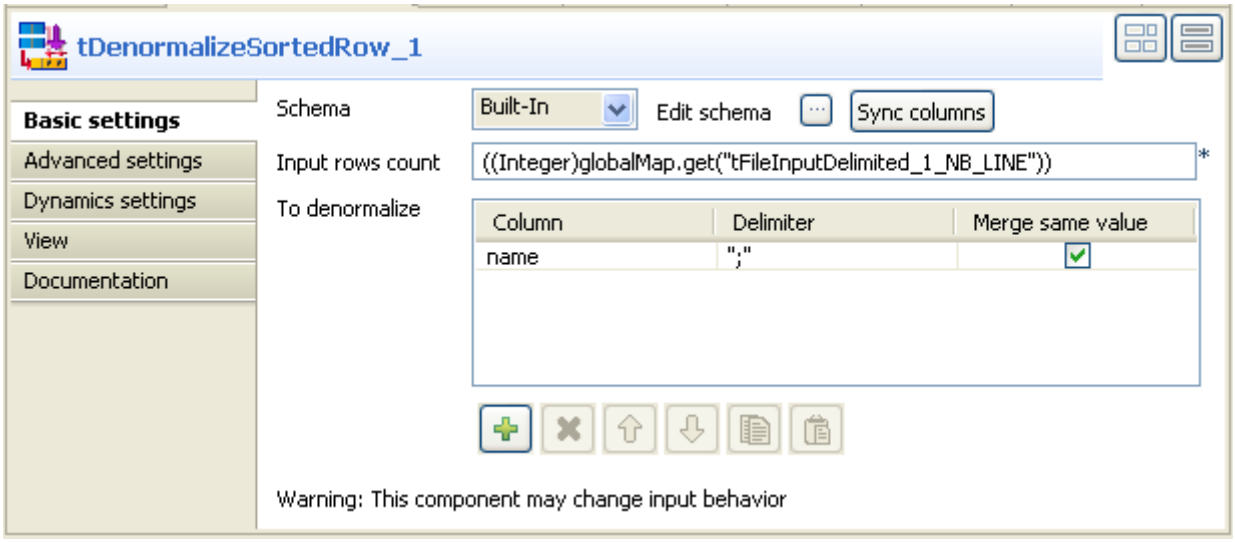
- If needed, define row and field separators, header and footer, and the number of processed rows.
- Set **Schema** to **Built in** and click the three-dot button next to **Edit Schema** to define the data to pass on to the next component. The schema in this example consists of two columns, *id* and *name*.



- In the design workspace, select **tSortRow**.
- Click the **Component** tab to define the basic settings for **tSortRow**.



- Set the **Schema Type** to **Built-In** and click **Sync columns** to retrieve the schema from the **tFileInputDelimited** component.
- In the **Criteria** panel, use the plus button to add a line and set the sorting parameters for the schema column to be processed. In this example we want to sort the *id* columns in ascending order.
- In the design workspace, select **tDenormalizeSortedRow**.
- Click the **Component** tab to define the basic settings for **tDenormalizeSortedRow**.



- Set the **Schema Type** to **Built-In** and click **Sync columns** to retrieve the schema from the **tSortRow** component.
- In the **Input rows count**field, enter the number of the input rows to be processed or press **Ctrl+Space** to access the context variable list and select the variable: `tDenormalize`SortedRow_1.NB_LINE`.`
- In the **To denormalize** panel, use the plus button to add a line and set the parameters to the column to be denormalize. In this example we want to denormalize the *name* column.
- In the design workspace, select **tLogRow** and click the **Component** tab to define its basic settings. For more information about **tLogRow**, see *tLogRow on page 628*.
- Save your Job and press **F6** to execute it.

```
Starting job denormalize1 at 14:42 01/09/2008.
bufferSize=100000 objects
Sorting buffer...
0 milliseconds for 9 objects to sort in memory. 2147483647
items/s
Writing ordered buffer in file...
16 milliseconds for 9 objects to write in file. 562  items/s
.--+----------------------.
|        tLogRow_1         |
|=-+----------------------=|
|id|name                   |
|=-+----------------------=|
|1 | Harrison; Jerry       |
|2 | Ford                  |
|3 | James; Goldman        |
|4 | Bill; Ford; Harry; Sue|
'--+----------------------'

Job denormalize1 ended at 14:42 01/09/2008. [exit code=0]
```

The result displayed on the console shows how the *name* column was denormalize.

tEmptyToNull

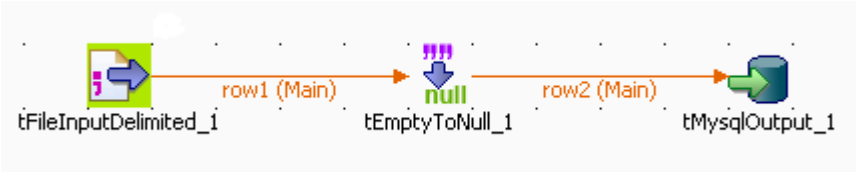
tEmptyToNull properties

Component family	Processing	
Function	**tEmptyToNull** transforms empty fields in a file or a table to NULL fields in a database table.	
Purpose	**tEmptyToNull** allows to replace empty fields by undefined fields that give the NULL value (unknown value) in the output component;	
Basic settings	This component does not need any configuration. It executes automatically.	
Advanced settings	*tStatCatcher Statistics*	Select this check box to gather the job processing metadata at a job level as well as at each component level.
Usage	This component is generally used as an intermediate step in a data flow. It needs then an input and output components. The output component should be a database output component.	
Limitation	n/a	

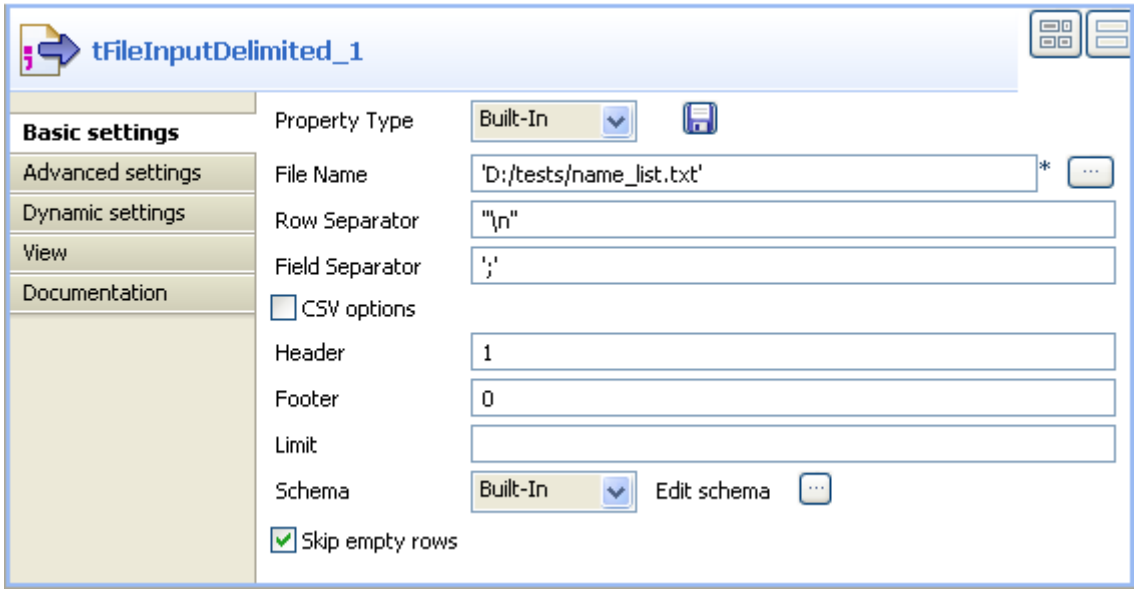
Scenario: Replacing empty fields by NULL fields (fields of unknown value)

This Perl scenario describes a three-component Job that makes it possible to replace input fields without character strings by undefined fields in order to generate Null values (unknown values) in the output fields.

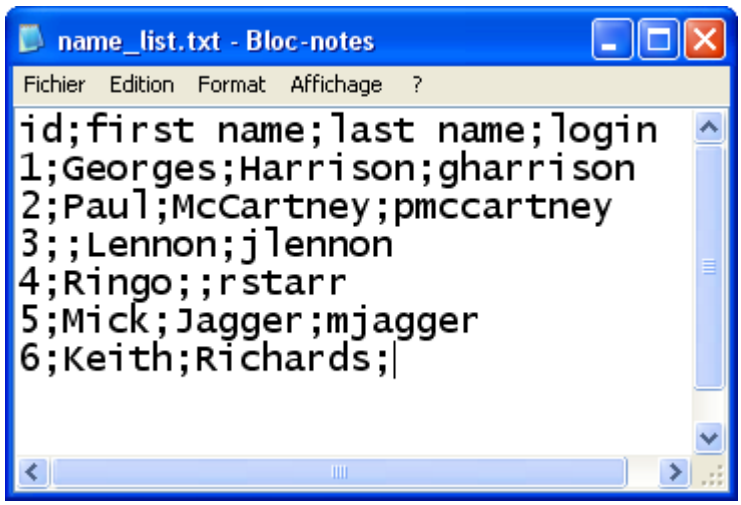
- Drop the following components from the **Palette** onto the design workspace: **tFileInputDelimited**, **tEmptyToNull** and **tMysqlOutput**.
- Connect the three components using **Row Main** links.



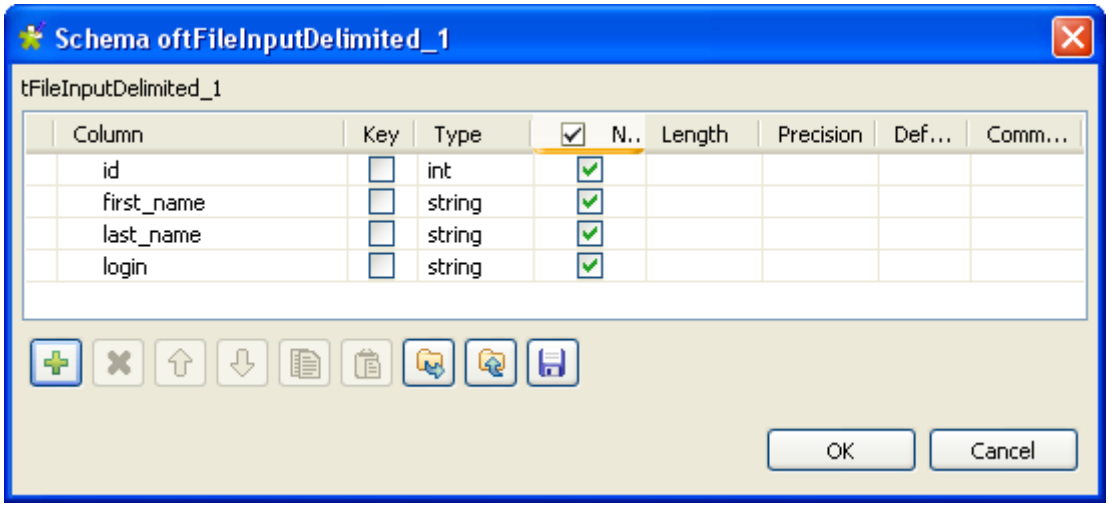
- In the design workspace, double-click **tFileInputDelimited** to display its **Basic settings** view where you can define the component properties.



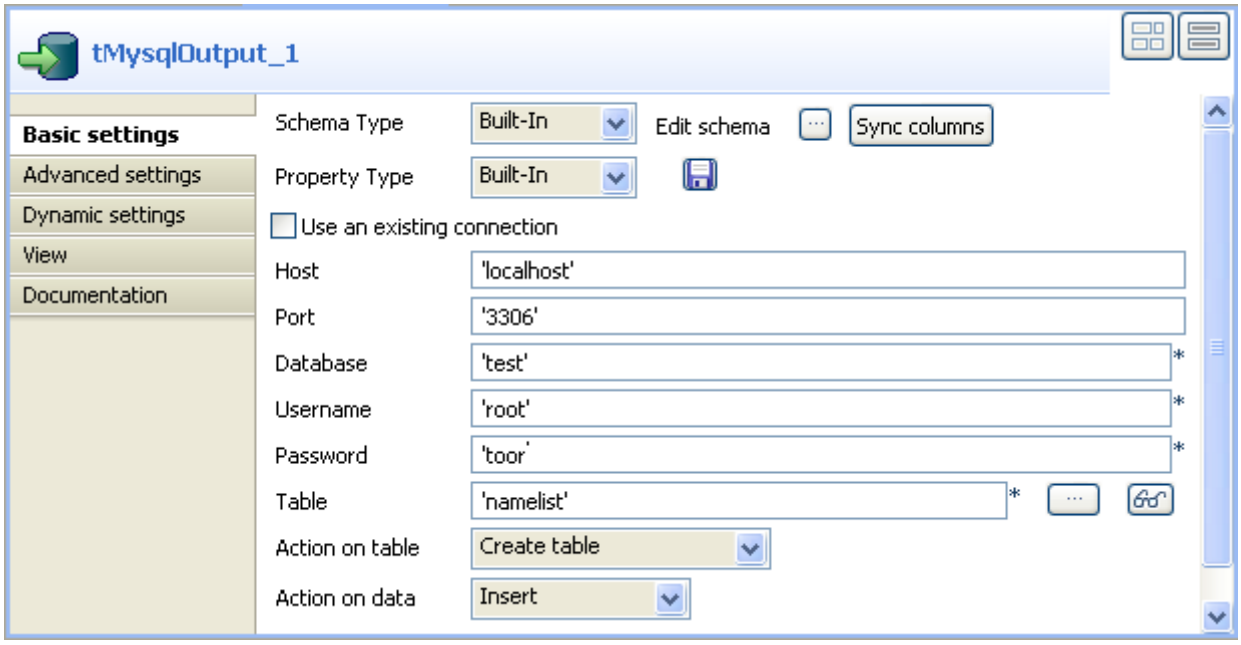
- From the **Property Type** list, select:
  -**Repository** if you have already stored the metadata of your input file in the Repository, the fields that follow are filled in with the stored information automatically, or
  -select **Built-In** and fill in the fields that follow manually.
  For this example, we use the **Built-In** mode.
- Click the three-dot button next to the **File Name** field and browse to the input file. In this example, our source file is *name_list* and it holds four columns: *id*, *first name*, *last name* and *login*.



- In the **Basic settings** view of **tFileInputDelimited**, define in the corresponding fields the row and field separators used in the source file.
- If needed, set **Header**, **Footer** and **Limit**.
  In this example, set **Header** to 1 since the first row that holds columns' names is to be ignored. **Footer** and **Limit** for the number of processed rows are not set.
- In the **Schema** field, set schema to **Built in** then click the three-dot button next to the **Edit Schema** field to define the data to be passed to the following component. In this example, the source schema consists of four columns: *id*, *first_name*, *last_name* and *login*.



- In the design workspace, double-click **tMysqlOutput** to open its **Basic Settings** view where you can define the component properties.



- Click **Sync columns** to retrieve the schema of the preceding component.

You can click the three-dot button next to **Edit schema** to check the retrieved schema.

- From the **Property Type** list, select:
-**Repository** if you have already stored the metadata of your database connection in the Repository, the fields that follow are filled in with the stored information automatically, or
-select **Built-In** and fill in the connection information manually.
For more information about **tMysqlOutput** properties, see *tMysqlOutput on page 281*.
- In the **Table** field, enter the name of the table that will hold the data extracted from the source delimited file.
- From the **Action on table** list, select the operation you want to carry out on the defined table. In this example, we select **Create table** to create the defined table.

- From the **Action on data** list, select the operation you want to carry out on the data. In this example, we select **Insert**.
- Save your Job and press **F6** to execute it.

id	first_name	last_name	login
1	Georges	Harrison	gharrison
2	Paul	McCartney	pmccartney
3	NULL	Lennon	jlennon
4	Ringo	NULL	rstarr
5	Mick	Jagger	mjagger
6	Keith	Richards	NULL

Through your data explorer, you can check that the output database table *namelist* is created with the defined columns and that the empty fields in the source file have been replaced by the NULL values (unknown values).

tExternalSortRow

tExternalSortRow properties

Component family	Processing	
Function	Uses an external sort application to sort input data based on one or several columns, by sort type and order	
Purpose	Helps creating metrics and classification table.	
Basic settings	*Schema type* and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the Repository. Click **Edit Schema** to make changes to the schema. Note that if you make changes, the schema automatically becomes built-in. Click **Sync columns** to retrieve the schema from the previous component connected in the Job.
		Built-in: The schema will be created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused in various projects and job flowcharts. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	File Name	Name of the file to be processed. Related topic: *Defining variables from the Component view* of **Talend Open Studio**. User Guide.
	Field separator	Character, string or regular expression to separate fields.
	External command "sort" path	Enter the path to the external file containing the sorting algorithm to use.
	Criteria	Click the plus button to add as many lines as required for the sort to be complete. By default the first column defined in your schema is selected.
		Schema column: Select the column label from your schema, which the sort will be based on. Note that the order is essential as it determines the sorting priority.
		Sort type: Numerical and Alphabetical order are proposed. More sorting types to come.
		Order: Ascending or descending order.

Advanced settings	*Maximum memory*	Type in the size of physical memory you want to allocate to sort processing.
	Temporary directory	Set the location where the temporary files should be stored.
	Add a dummy EOF line	Select this check box when using the **tAggregateSortedRow** component.
	tStatCatcher Statistics	Select this check box to gather the job processing metadata at the Job level as well as at each component level.
Usage	This component handles flow of data therefore it requires input and output, hence is defined as an intermediary step.	
Limitation	n/a	

Related scenario

For related use case, see **tSortRow** *Scenario: Sorting entries on page 789*.

tExtractDelimitedFields

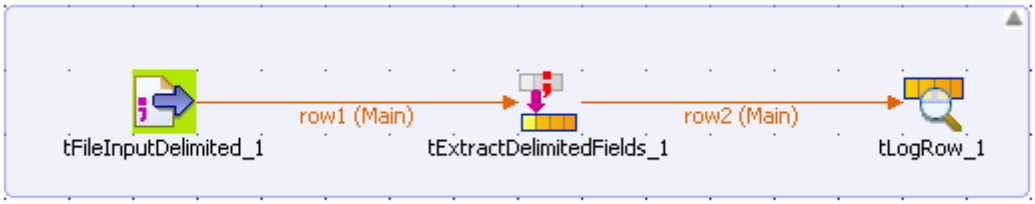
tExtractDelimitedFields properties

Component family	Processing/Fields	
Function	**tExtractDelimitedFields** generates multiple columns from a given column in a delimited file.	
Purpose	**tExtractDelimitedFields** helps to extract 'fields' from within a string to write them elsewhere for example.	
Basic settings	*Field to split*	Select from the list the field to split.
	Field separator	Set field separator.  Since this component uses regex to split a filed and the regex syntax uses special characters as operators, make sure to precede the regex operator you use as a field separator by a double backslash. For example, you have to use "\\|" instead of "|".
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the Repository. Click **Edit schema** to make changes to the schema. Note that if you make changes, the schema automatically becomes built-in. Click **Sync columns** to retrieve the schema from the previous component connected to **t**ExtractDelimitedFields.
		Built-in: You create the schema and store it locally for the component. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: You have already created the schema and stored it in the Repository. You can use it in various projects and job flowcharts. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
Usage	This component handles flow of data therefore it requires input and output components.	
Limitation	n/a	

Scenario: Extracting fields from a comma-delimited file

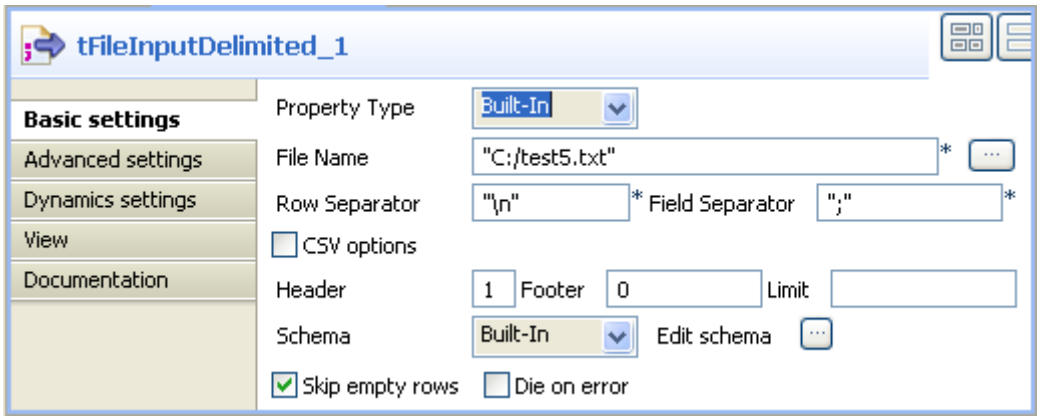
This Java scenario describes a three-component Job where the **tExtractdelimitedFields** component is used to extract two columns from a comma-delimited file.

- Drop the following components from the **Palette** onto the design workspace: **tFileInputDelimited**, **tExtractDelimitedFields**, and **tLogRow**.
- Via a right-click on each of the three components, connect them using **Row Main** links.



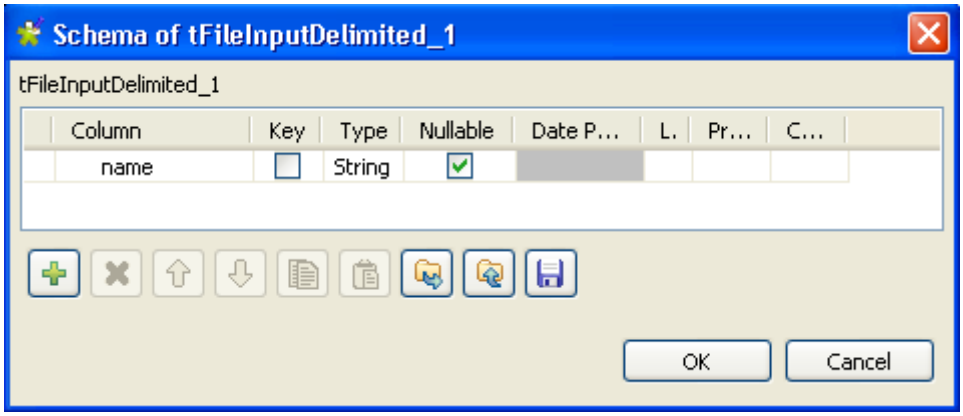
- In the design workspace, select **tFileInputDelimited**.
- Click the **Component** tab to define the basic settings for **tFileInputDelimited**.
- In the **Basic settings** view, set **Property Type** to **Built-In**.
- Click the three-dot [...] button next to the **File Name** field to select the path to the input file.

The **File Name** field is mandatory.

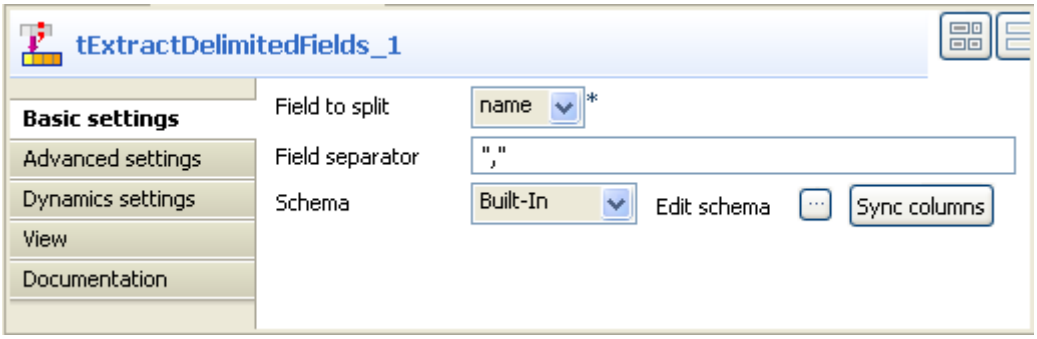


The input file used in this scenario is called *test5*. It is a text file that holds comma-delimited data.

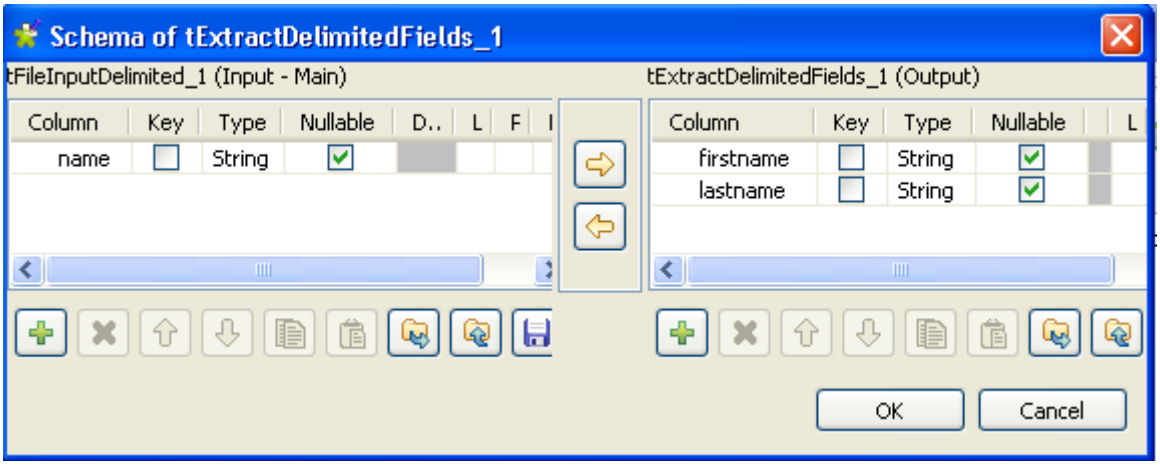
- In the **Basic settings** view, fill in all other fields as needed. For more information, see *tFileInputDelimited properties on page 494*. In this scenario, the header and the footer are not set and there is no limit for the number of processed rows
- Click **Edit schema** to describe the data structure of this input file. In this scenario, the schema is made of one column, *name*.



- In the design workspace, select **tExtractDelimitedFields**.
- Click the **Component** tab to define the basic settings for **tExtractDelimitedFields**.



- From the **Field to split** list, select the column to split, *name* in this scenario.
- In the **Field separator** field, enter the corresponding separator.
- Click **Edit schema** to describe the data structure of this processing component.
- In the output panel of the **[Schema of tExtractRegexFields]** dialog box, click the plus button to add two columns for the output schema, *firstname* and *lastname*.



I this scenario, we want to split the *name* column into two columns in the output flow, *firstname* and *lastname*.

- Click **OK** to close the **[Schema of tExtractDelimitedFields]** dialog box.

- In the design workspace, select **tLogRow** and click the **Component** tab to define its basic settings. For more information, see *tLogRow on page 628*.
- Save your Job and press **F6** to execute it.

```
Starting job extract_delimited at 16:39 25/08/2008.
.---------+--------.
|    tLogRow_1       |
|=---------+-------=|
|firstname|lastname|
|=---------+-------=|
|Janet    |Anderson|
|Martin   |Chairman|
|Lily     |Massy   |
'---------+--------'

Job extract_delimited ended at 16:39 25/08/2008. [exit
```

First names and last names are extracted and displayed in the corresponding defined columns on the console.

tExtractPositionalFields

tExtractPositionalFields properties

Component family	Processing/Fields	
Function	**tExtractPositionalFields** generates multiple columns from one column using positional fields.	
Purpose	**tExtractPositionalFields** allows to use a positional pattern to extract data from a formatted string.	
Basic settings	*Field*	Select from the list the field to extract from.
	Customize	Select this check box to customize the data format of the positional file and define the table columns: **Column**: Select the column you want to customize. **Size**: Enter the column size. **Padding char**: Type in between quotes the padding characters used. A space by default. **Alignment**: Select the appropriate alignment parameter.
	Pattern	Enter the pattern to use as basis for the extraction. A pattern is length values separated by commas, interpreted as a string between quotes. Make sure the values entered in this fields are consistent with the schema defined.
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the Repository. Click **Edit Schema** to make changes to the schema. Note that if you make changes, the schema automatically becomes built-in. Click **Sync columns** to retrieve the schema from the previous component connected to **t**PositionalFields.
		Built-in: You create the schema and store it locally for the component. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: You have already created the schema and stored it in the Repository. You can reuse it in various projects and job flowcharts. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
Usage	This component handles flow of data therefore it requires input and output components.	
Limitation	n/a	

Related scenario

For related use case, see *tExtractRegexFields on page 742*.

tExtractRegexFields

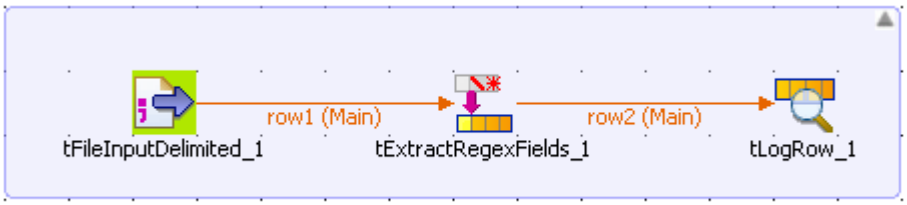
tExtractRegexFields properties

Component family	Processing/Fields	
Function	**tExtractRegexFields** generates multiple columns from a given column using regex matching.	
Purpose	**tExtractRegexFields** allows to use regular expressions to extract data from a formatted string.	
Basic settings	*Field to split*	Select on the list the field (column) to split.
	Regex	Enter a regular expression according to the programming language you are using.
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the Repository. Click **EditSchema** to make changes to the schema. Note that if you make changes, the schema automatically becomes built-in. Click **Sync columns** to retrieve the schema from the previous component connected to **t**ExtractRegexFields.
		Built-in: You create and store the schema locally for the component. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: You have already created and stored the schema in the Repository. You can reuse it in various projects and job flowcharts. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
Usage	This component handles flow of data therefore it requires input and output components.	
Limitation	n/a	

Scenario: Extracting name, domain and TLD from e-mail addresses

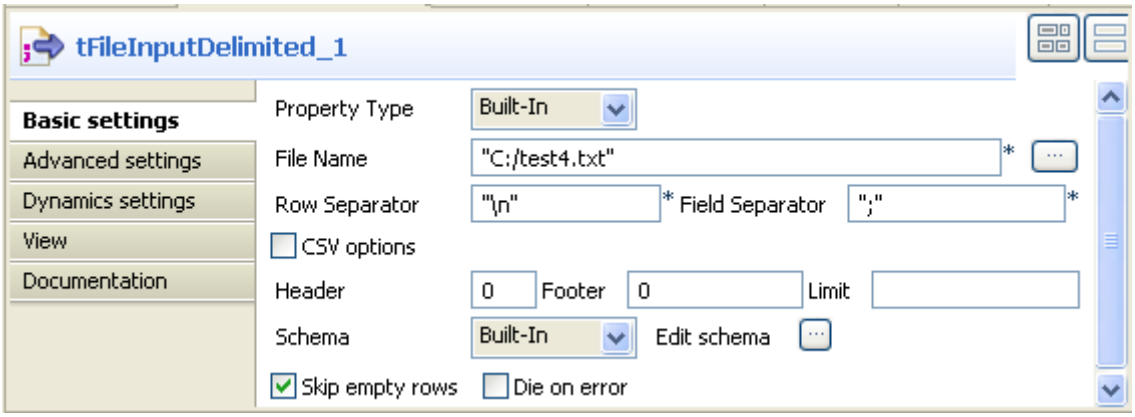
This Java scenario describes a three-component Job where **tExtractRegexFields** is used to specify a regular expression that corresponds to one column in the input data, *email*. The **tExtractRegexFields** component is used to perform the actual regular expression matching. This regular expression includes field identifiers for user name, domain name and Top-Level Domain name portions in each e-mail address. If the given e-mail address is valid, the name, domain and TLD are extracted and displayed on the console in three separate columns. Data in the other two input columns, *id* and *age* is extracted and routed to destination as well.

- Drop the following components from the **Palette** onto the design workspace: **tFileInputDelimited**, **tExtractRegexFields**, and **tLogRow**.
- Connect the three components using **Row Main** links.



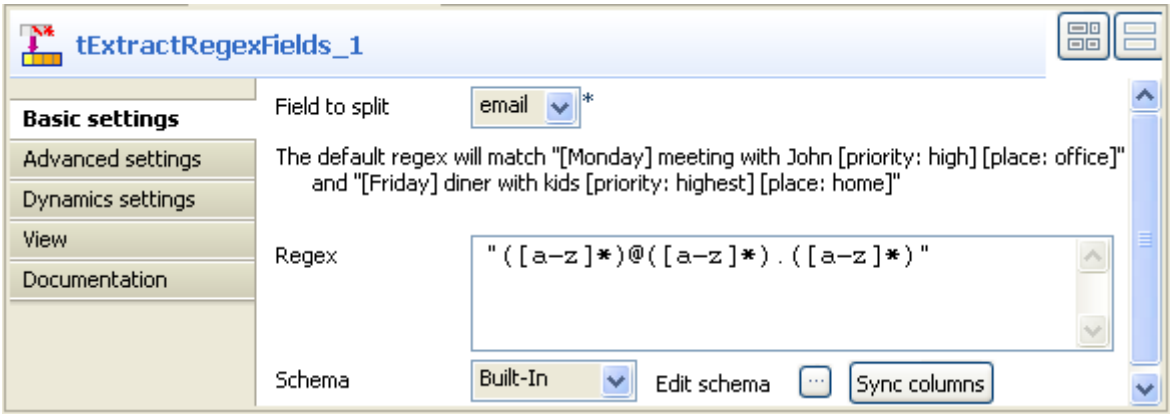
- In the design workspace, select **tFileInputDelimited**.
- Click the **Component** tab to define the basic settings for **tFileInputDelimited**.
- In the **Basic settings** view, set **Property Type** to **Built-In**.
- Click the three-dot [...] button next to the **File Name** field to select the path to the input file.

The **File Name** field is mandatory.

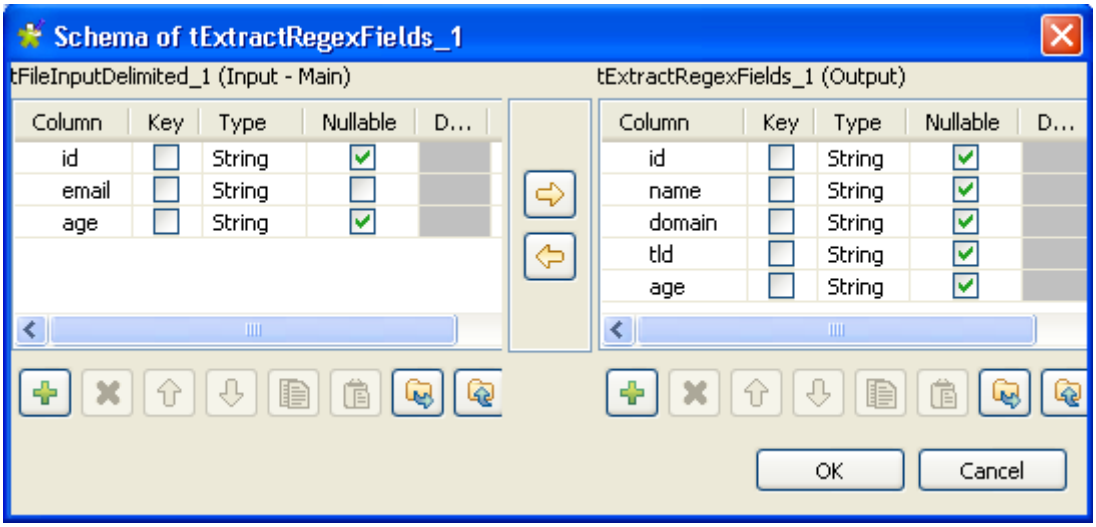


The input file used in this scenario is called *test4*. It is a text file that holds three columns: *id*, *email*, and *age*.

- Fill in all other fields as needed. For more information, see *tFileInputDelimited properties on page 494*. In this scenario, the header and the footer are not set and there is no limit for the number of processed rows
- Click **Edit schema** to describe the data structure of this input file. In this scenario, the schema is made of the three columns, *id*, *email* and *age*.
- In the design workspace, select **tExtractRegexFields**.
- Click the **Component** tab to define the basic settings for **tExtractRegexFields**.
- From the **Field to split** list, select the column to split, *email* in this scenario.
- In the **Regex** panel, enter the regular expression you want to use to perform data matching, java regular expression in this scenario.



- Click **Edit schema** to describe the data structure of this processing component.
- In the output panel of the **[Schema of tExtractRegexFields]** dialog box, click the plus button to add five columns for the output schema.



In this scenario, we want to split the input *email* column into three columns in the output flow, *name*, *domain*, and *tld*. The two other input columns will be extracted as they are.

- Click **OK** to close the **[Schema of tExtractRegexFields]** dialog box.
- In the design workspace, select **tLogRow** and click the **Component** tab to define its basic settings. For more information, see *tLogRow on page 628*.
- Save your Job and press **F6** to execute it.

```
Starting job extract_regex_mine at 10:46 29/08/2008.
.--+----+------+---+---.
|       tLogRow_1        |
|=-+----+------+---+--=|
|id|name|domain|tld|age|
|=-+----+------+---+--=|
|1 |name|domain|com|24 |
|2 |name|domain|net|31 |
|3 |name|domain|org|13 |
'--+----+------+---+---'

Job extract_regex_mine ended at 10:46 29/08/2008. [exit code=0]
```

The **tExtractRegexFields** component matches all given e-mail addresses with the defined regular expression and extracts the name, domain, and TLD names and displays them on the console in three separate columns. The two other columns, *id* and *age*, are extracted as they are.

tFilterColumn

tFilterColumn Properties

Component family	Processing	
Function	Makes specified changes to the schema defined, based on column name mapping.	
Purpose	Helps homogenizing schemas either on the columns order or by removing unwanted columns or adding new columns.	
Basic settings	*Schema type* and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the Repository.
		Built-in: The schema will be created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused in various projects and job designs. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
Usage	This component is not startable (green background) and it requires an output component.	

Related Scenario

For more info regarding the **tFilterColumn** component in use, see **tReplace** *Scenario: multiple replacements and column filtering on page 781*

tFilterRows

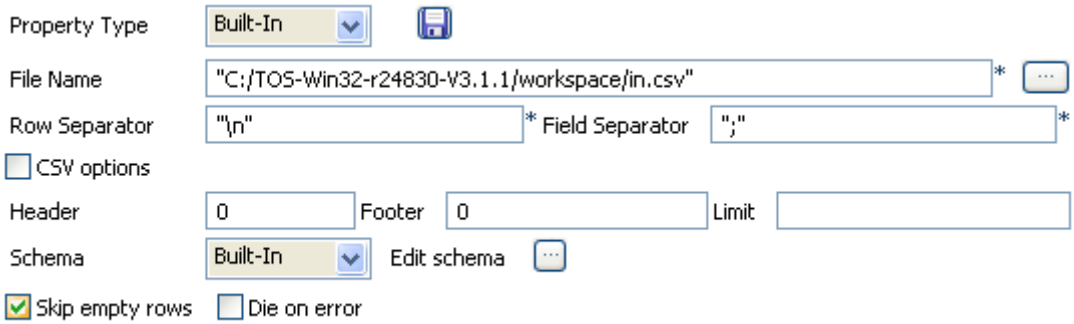
tFilterRows Properties

Component family	Processing	
Function	Compares a column from the main flow with a reference column from the lookup flow and outputs the main flow data displaying the distance	
Purpose	Helps ensuring the data quality of any source data against a reference data source.	
Basic settings	*Schema type* and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the Repository.
		Built-in: The schema will be created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused in various projects and job designs. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	Logical operator used to combine conditions	In the case you want to combine simple filtering and advanced mode, select the operator to combine both modes.
	Conditions	Click the plus button to add as many conditions as needed. The conditions are performed one after the other for each row. **Function**: Select the function on the list **Input column**: Select the column of the schema the function is to be operated on **Operator**: Select the operator to bind the input column with the value **Value**: Type in the filtered value, between quotes if need be.
	Use advanced mode	Select this check box when the operation you want to perform cannot be carried out through the standard functions offered. In the text field, type in the regular expression as required.
Usage	This component is not startable (green background) and it requires an output component.	

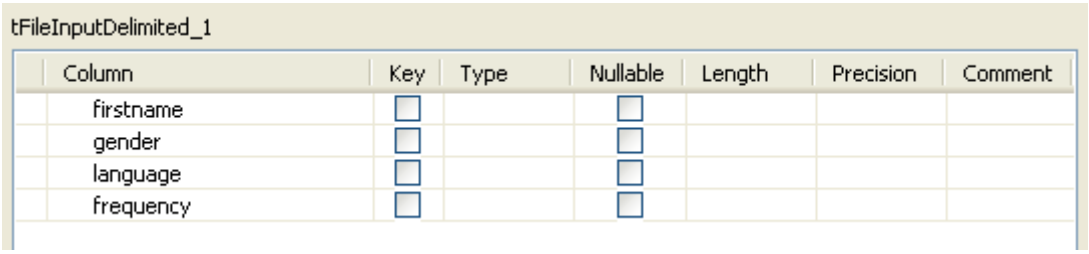
Scenario: Filtering and searching a list of names

The following scenario filters a list of first names based on the name gender. Then using a regular expression, the first names starting with ‘rom’ are listed.

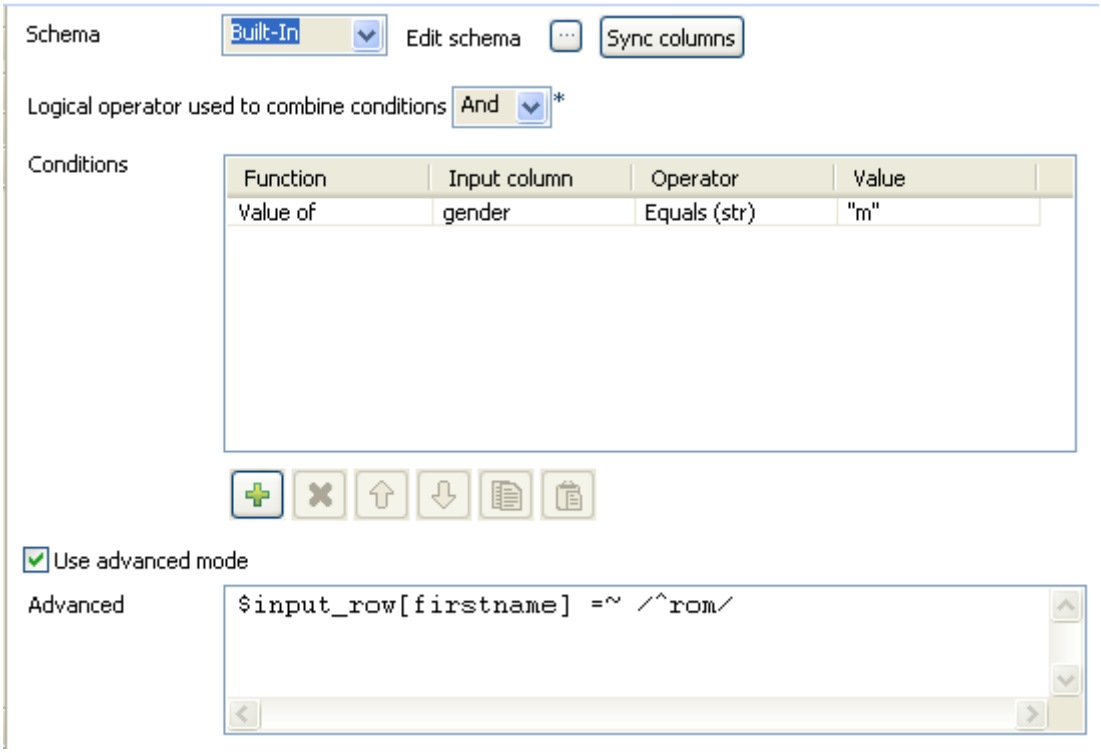
- Drop **tFileInputDelimited**, **tFilterRows** and **tLogRow** from the **Palette** onto the design workspace.
- Connect the three components using a **Row > Main** link.
- Double-click **tFileInputDelimited** to display its **Basic settings** view and define its properties.



- Set the file path and the row and field separators in the corresponding fields. The row separator is a carriage return and the field separator is a tabulation.
- From the **Property Type** list, select **Built-in**. The properties and schema are **Built-in** for this Job. This means, the schema is not stored in the **Repository**.
- Click the three-dot button next to **Edit schema** to define the schema for the input file. In this example, the schema is made of the following four columns: *firstname*, *gender*, *language* and *frequency*.



- Click the **Advanced settings** tab to display the corresponding view then from the **Encoding type** list, select the type according to your file.
- Double-click **tFilterRow**s to display its **Basic settings** view and define its properties.



- In the **Conditions** table, fill in the filtering parameters based on the *gender* column.
- In **Function**, select **value of,** as **Input column**, select *gender* and as operator, select Equals (Str) as the expected values are of string type.
- In the **Value** column, type in *m* between double quotes to filter only the male names.
- Then to implement the search on first names starting with the *rom* syllable, select the **Use advanced mode** check box and type in the following regular expression (in Perl) that includes the name of the column to be searched: `$input_row[firstname] =~ /^rom/`
- To combine both conditions (simple and advanced), select **And** as logical operator for this example.
- The **tLogRow** component does not require any particular setting for this example.
- Save your Job and press **F6** to execute it.

```
Starting job FilterNames at 16:51 24/09/2007.
romain|m|french|15.32
roman|m|russian, polish, czech|53.65
romano|m|italian|0.41
romeo|m|italian|0.29
romolo|m|italian|0
romulus|m|roman mythology|0.45
Job FilterNames ended at 16:51 24/09/2007. [exit code=0
```

Only the male names starting with the *rom* syllable are listed on the console.

tMap

tMap properties

Component family	Processing	
Function	**tMap** is an advanced component, which integrates itself as plugin to **Talend Open Studio**.	
Purpose	**tMap** transforms and routes data from single or multiple sources to single or multiple destinations.	
Basic settings	Preview	The preview is an instant shot of the Mapper data. It becomes available when Mapper properties have been filled in with data. The preview synchronization takes effect only after saving changes.
	Mapping links display as	**Auto**: the default setting is curves links **Curves**: the mapping display as curves **Lines**: the mapping displays as straight lines. This last option allows to slightly enhance performance.
	Map editor	It allows you to define the **tMap** routing and transformation properties.
Usage	Possible uses are from a simple reorganization of fields to the most complex jobs of data multiplexing or demultiplexing transformation, concatenation, inversion, filtering and more...	
Limitation	The use of **tMap** supposes minimum Perl or Java knowledge in order to fully exploit its functionalities. This component is a junction step, and for this reason cannot be a start nor end component in the Job	

For further information, see *Mapping data flows* in **Talend Open Studio** User Guide.

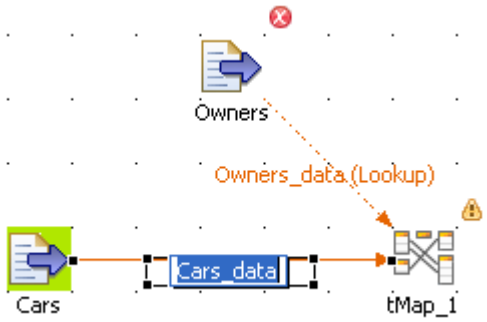
Scenario 1: Mapping with filter and simple explicit join (Perl)

The Job described below aims at reading data from a csv file stored in the Repository, looking up at a reference file also stored remotely, then extracting data from these two files based on defined filters to an output file or a reject file.

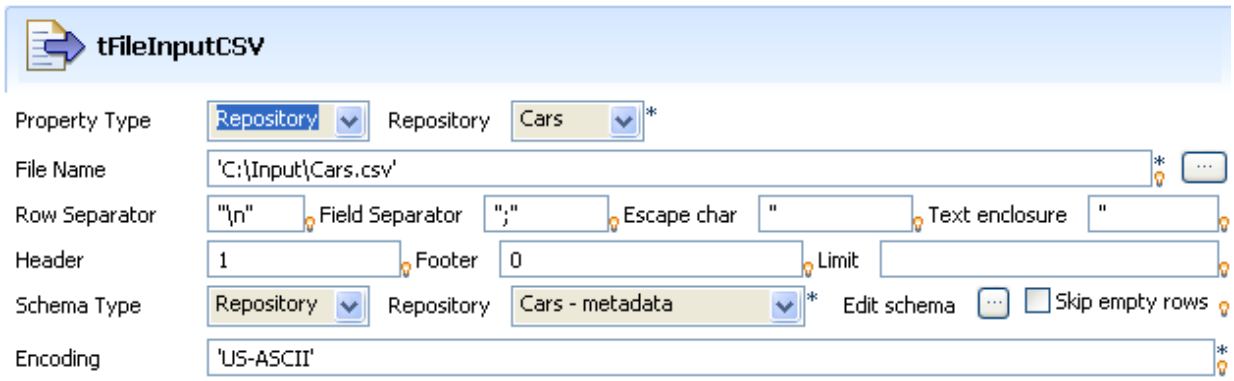
This Job is presented in Perl but could also be carried out in Java.

- Click on **File** in the **Palette** of components, select **tFileInputCSV** and drop it on the design area. Rename the Label to *Cars*, either by double-clicking on the label in the design workspace or via the **View** tab of the Component view.
- Repeat this operation, and rename this second input component: *Owners*.
- Click on **Processing** in the **Palette** of components, select **tMap** and drop it on the design area.

- Connect the two Input components to the mapping component and customize the row connection labels.



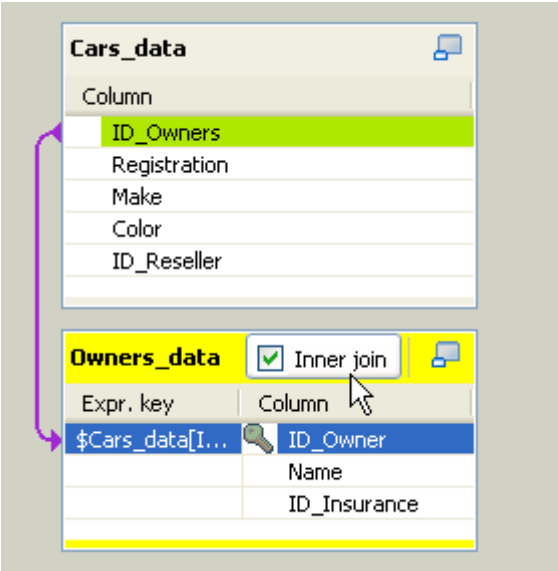
- The Cars and Owners delimited files metadata are defined in the Metadata area of the repository. You can hence use their Repository reference in the Component view.
- Double-click on *Cars*, to set the Component view.



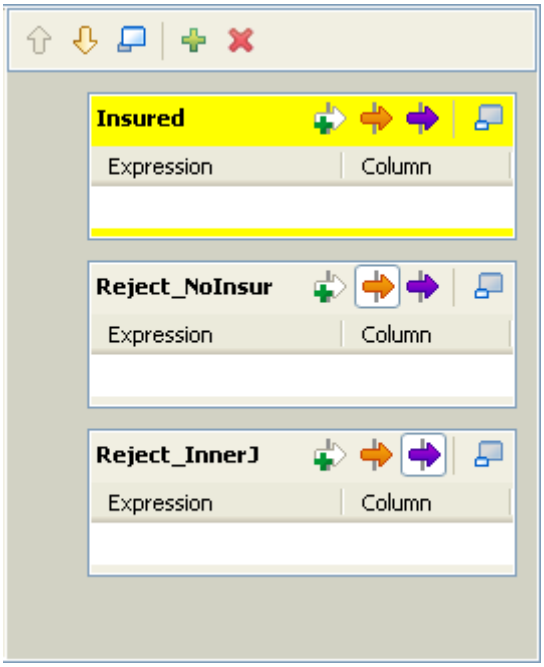
- Select *Repository* to retrieve **Property type** as well as **Schema type**. The rest of the fields gets automatically filled in appropriately when you select the relevant Metadata entry on the list.
- Double-click on the Owners component and repeat the setting operation. Select the corresponding Metadata entry.

For further information regarding Metadata creation in the Repository, see *Defining Metadata items* in **Talend Open Studio** User Guide.

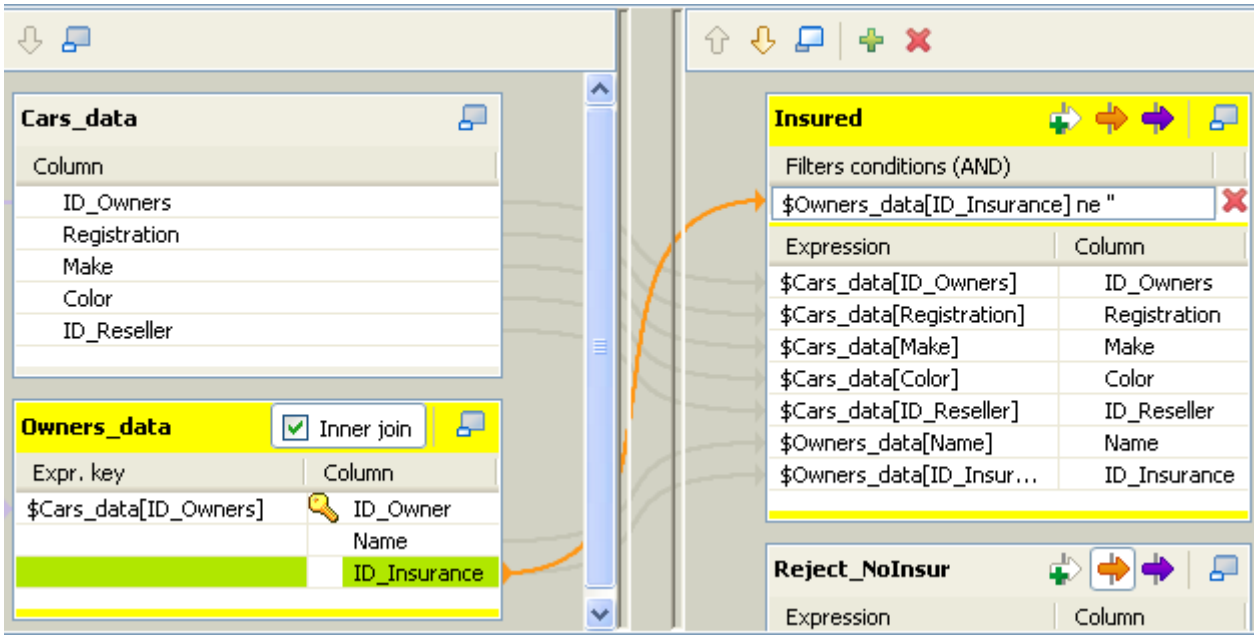
- Then double-click on the **tMap** component to open the Mapper. Notice that the Input area is already filled with the Input component metadata tables and that the top table is the Main flow table.
- Notice also that the respective row connection labels display on the top bar of the tables.
- Create a Join between the two tables on the *ID_Owner* field by simply dragging the top table *ID_Owner* field to the lookup input table.



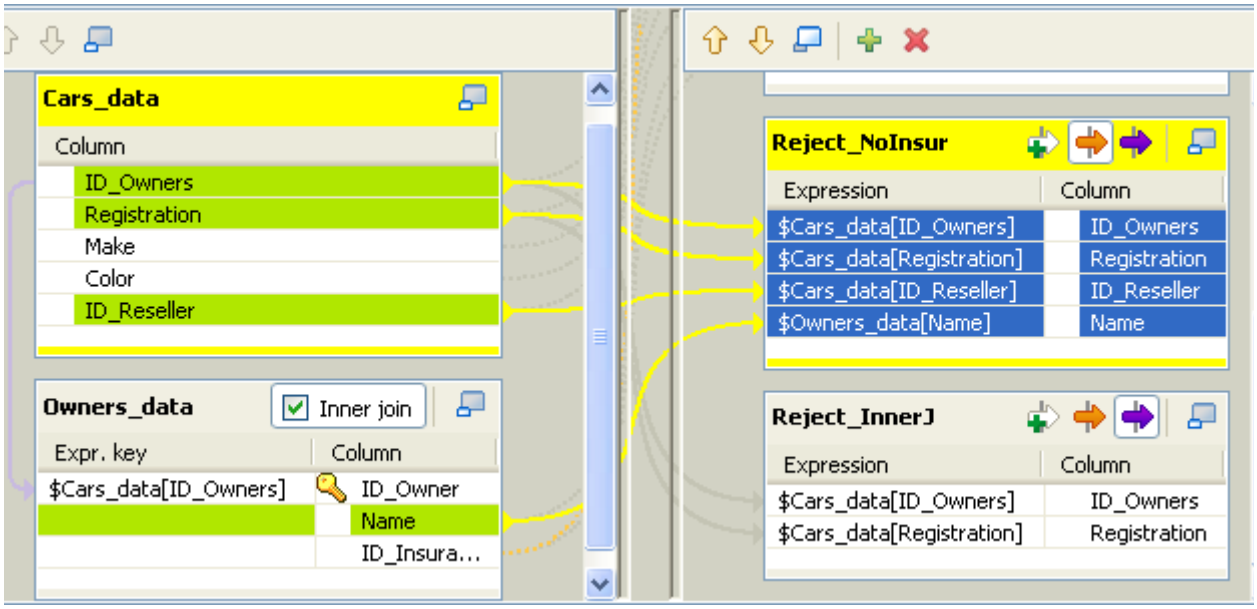
- Define this link as an inner join by selecting the corresponding check box.
- Click on the **Plus** button on the Output area of the Mapper to add three Output tables



- Drag and drop Input content to fill in the first output schema. For more information regarding data mapping, see *Mapping data flows* in **Talend Open Studio** User Guide.
- Click on the plus arrow button to add a filter row. In the *Insured* table, will be gathered cars and owners data which include an *Insurance ID*.

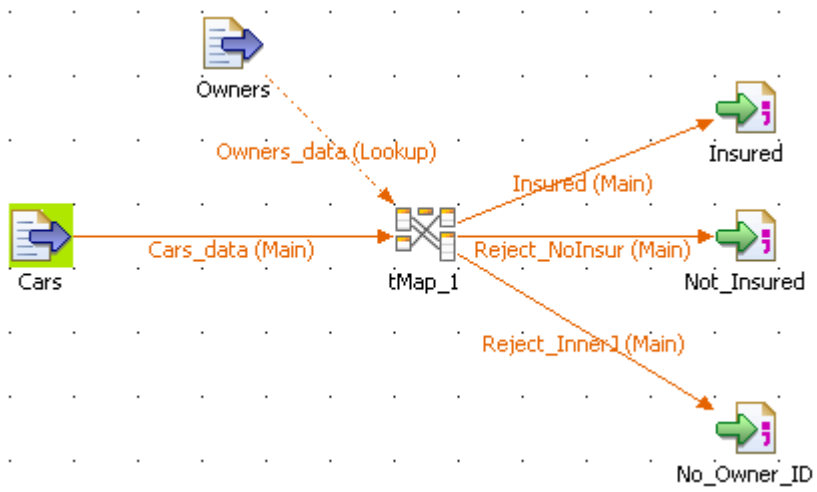


- Therefore drag the *ID_Insurance* field to the Filter condition area and enter the formula used meaning 'not undefined': `$Owners_data[ID_Insurance] ne ''`
- The Reject_NoInsur table is a standard reject output flow containing all data that do not satisfy the required filter condition. Click the orange arrow to set the table as Reject Output.

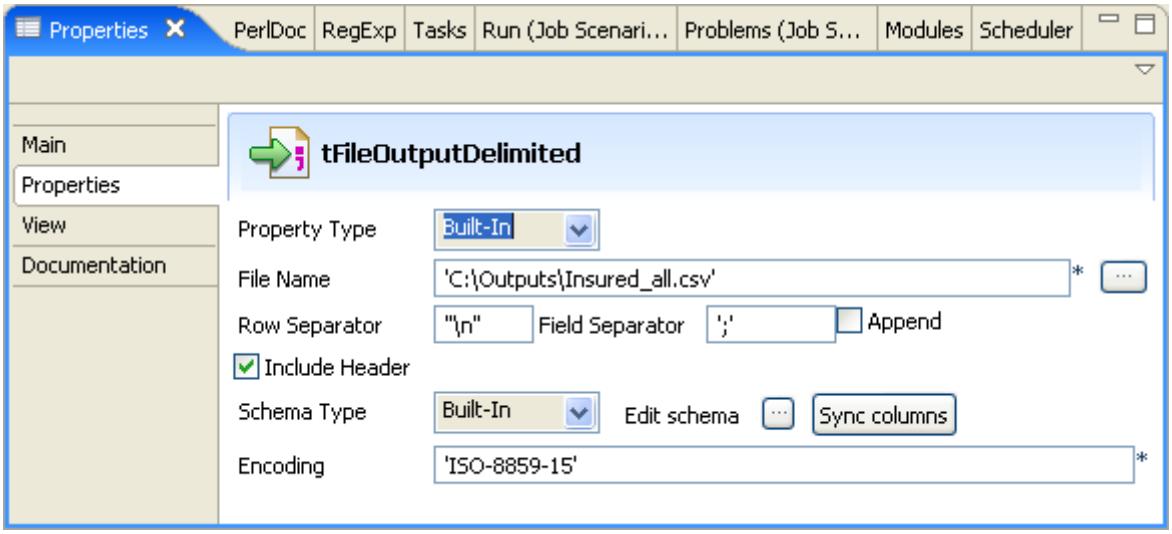


- The third and last table gathers the schema entries whose Inner Join could not be established. One *Owners_ID* from the *Car* database does not match any *Owner_ID* from the *Owner* file.
- Click the violet arrow button to set the last table as the Inner Join Reject output flow.
- Click **OK** to validate and come back to the design area.
- Add three **tFileOutputDelimited** components to the design workspace and right-click on the **tMap** to connect the Mapper with all three output components, using the relevant Row connection.

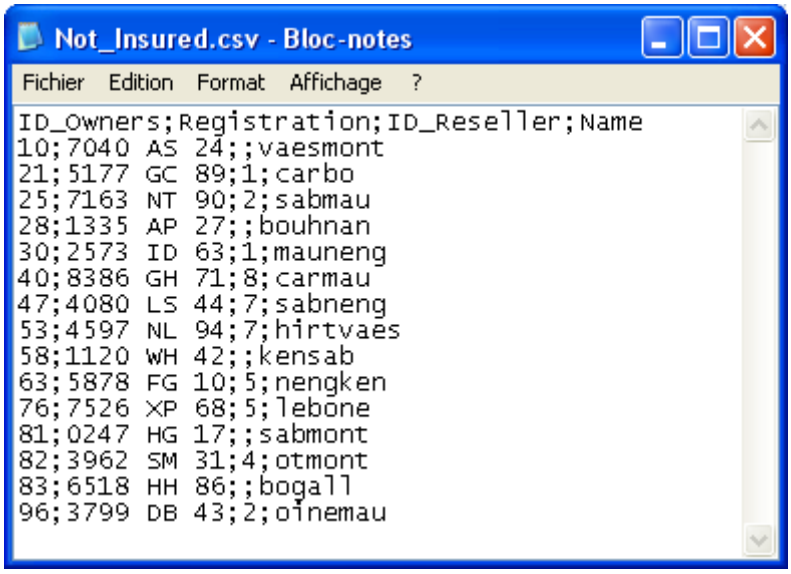
- Relabel the three output components accordingly.



- Then double-click on each of them, one after the other, in order to define their respective output filepath. If you want a new file to be created, browse to the destination output folder, and type in a file name including the extension.
- Select the **Include header** check box to reuse the column labels from the schema as header row in the output file.



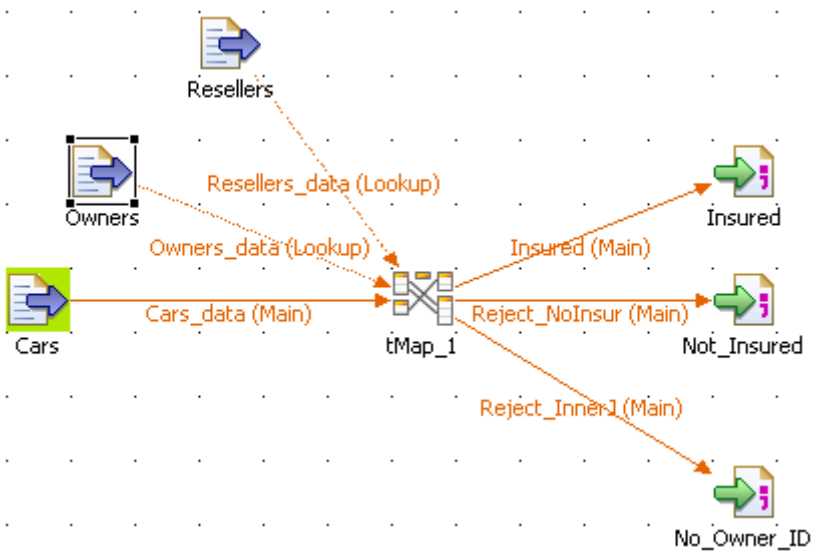
- Run the Job using the **F6** keystroke or via the **Run** panel.
- Output files have been created if need be.



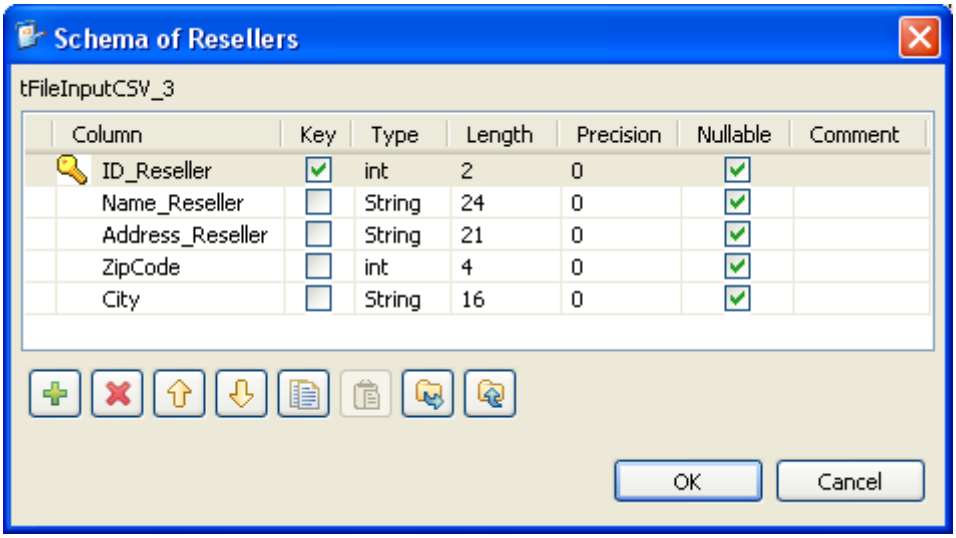
Scenario 2: Mapping with Inner join rejection (Perl)

This scenario, based on scenario 1, adds one input file containing Resellers details and extra fields in the main Output table. Two filters on Inner Joins are added to gather specific rejections.

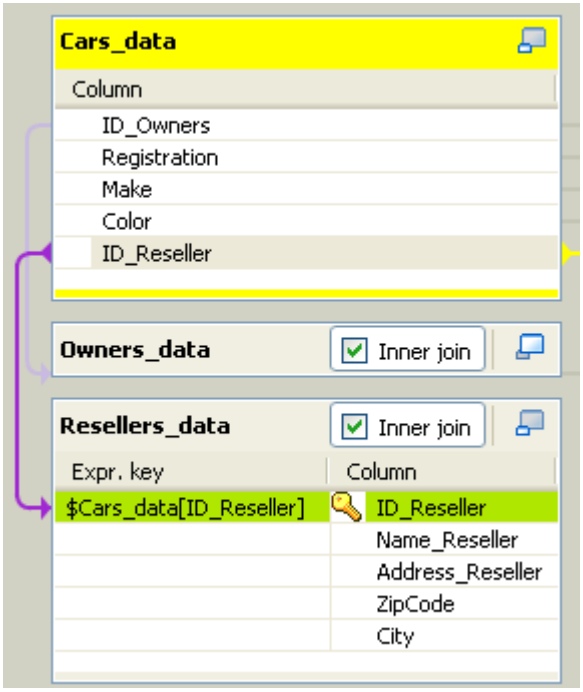
- Click on **File** in the **Palette** of Components, and drop a **tFileInputCSV** component to the design workspace.
- Connect it to the Mapper and put a label on the component and the connection.



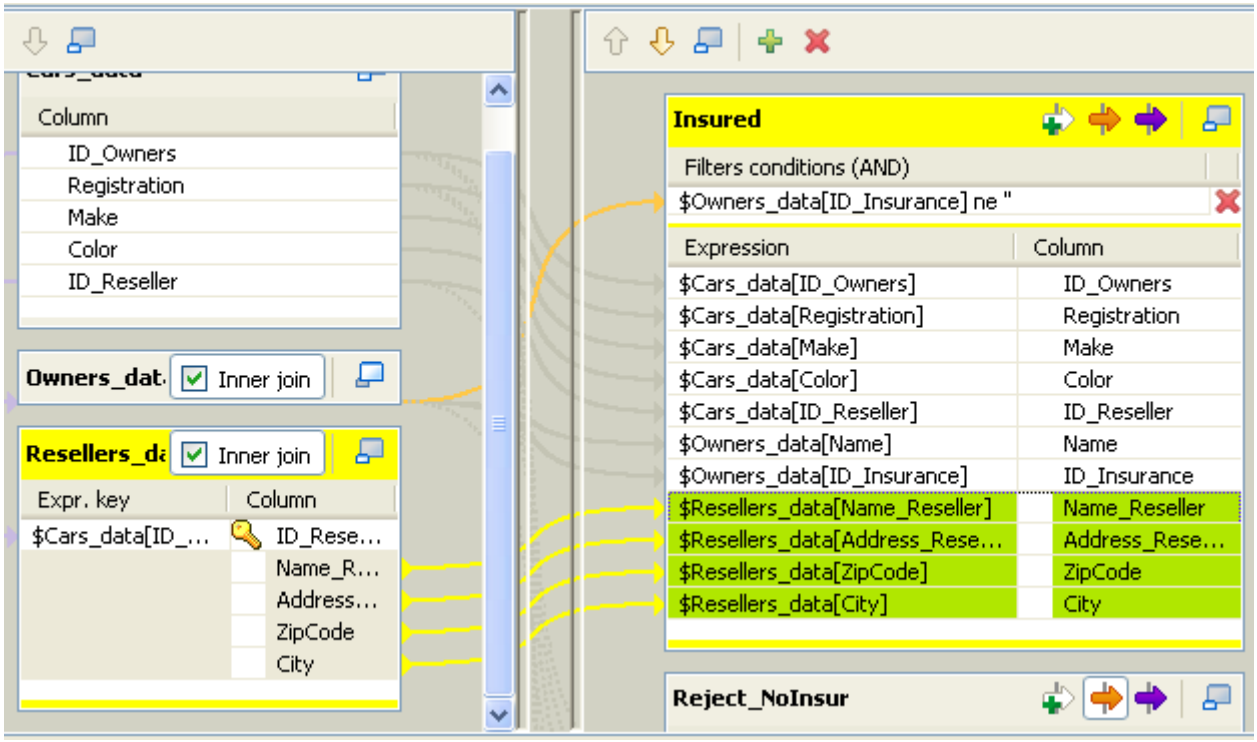
- Double-click on the *Resellers* component, to define the *Reseller* input properties.
- Browse to the *Resellers.csv* file. Edit the schema and add the columns as needed to match the file structure.



- You could also create a metadata entry for this file description and select Repository as properties and schema type. For further information, see *Setting up a File Delimited schema* in **Talend Open Studio** User Guide.
- Double-click on the **tMap** component and notice that the schema is added on the Input area.

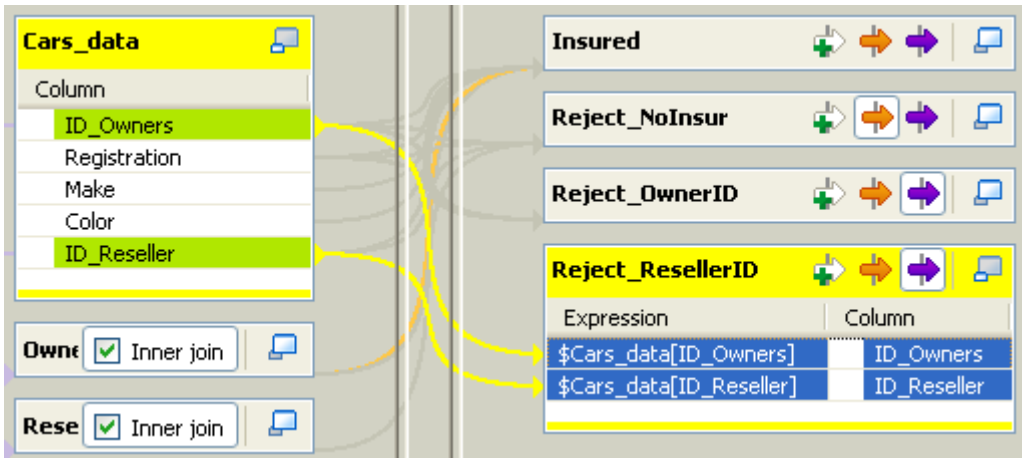


- Create a join between the main input flow and the resellers input. Select the **Inner Join** check box to define that an Inner Join Reject output is to be created.
- Drag the fields from the *Resellers* table to the main Output table.



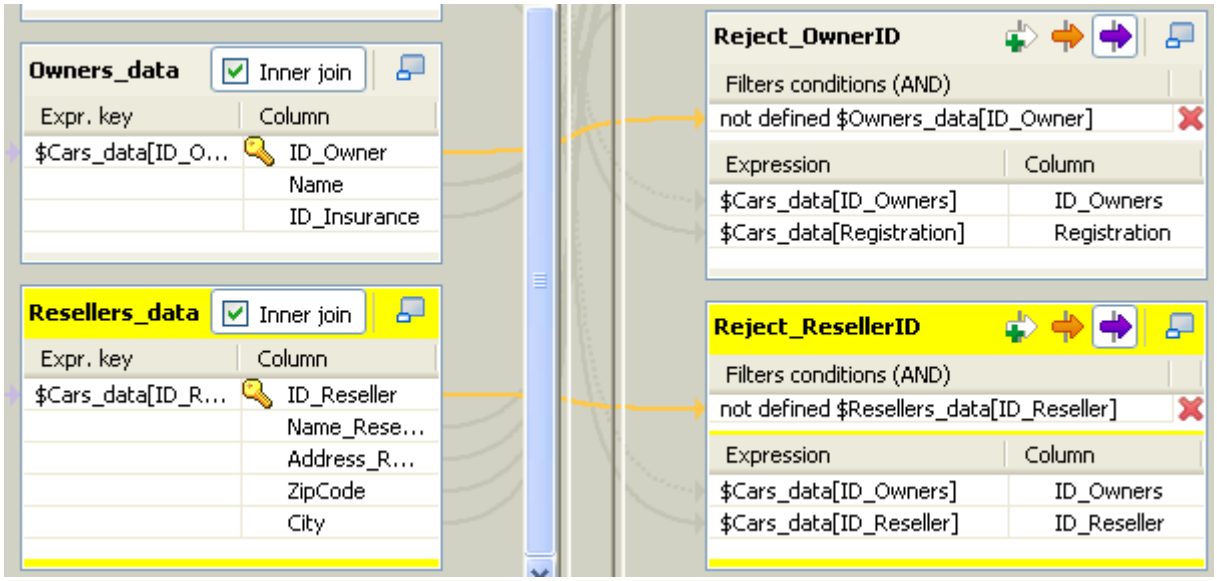
When two inner joins are defined, you either need to define two different inner join reject tables to differentiate both rejections or if there is only one Inner Join reject output, both Inner Join rejections will be stored in the same output.

- On the output area, click on the plus button to add a new output table.
- Give a name to this new Output connection: *Reject_ResellerID*
- Click the Inner Join reject button to define this new output table as Inner Join Reject output.
- Drag & drop two fields from the main input flow (*Cars*) to the new reject output table. In this case, if the Inner Join cannot be established, these data (*ID_Cars* & *ID_resellers*) will be gathered in the output file and will help identify the bottleneck.

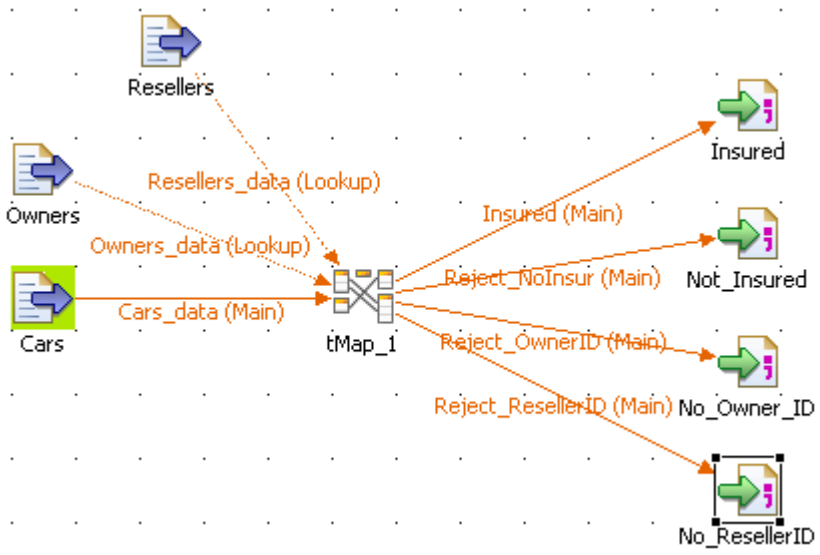


- Now apply filters on the two Inner Join reject outputs, in order for to distinguish both types of rejection.

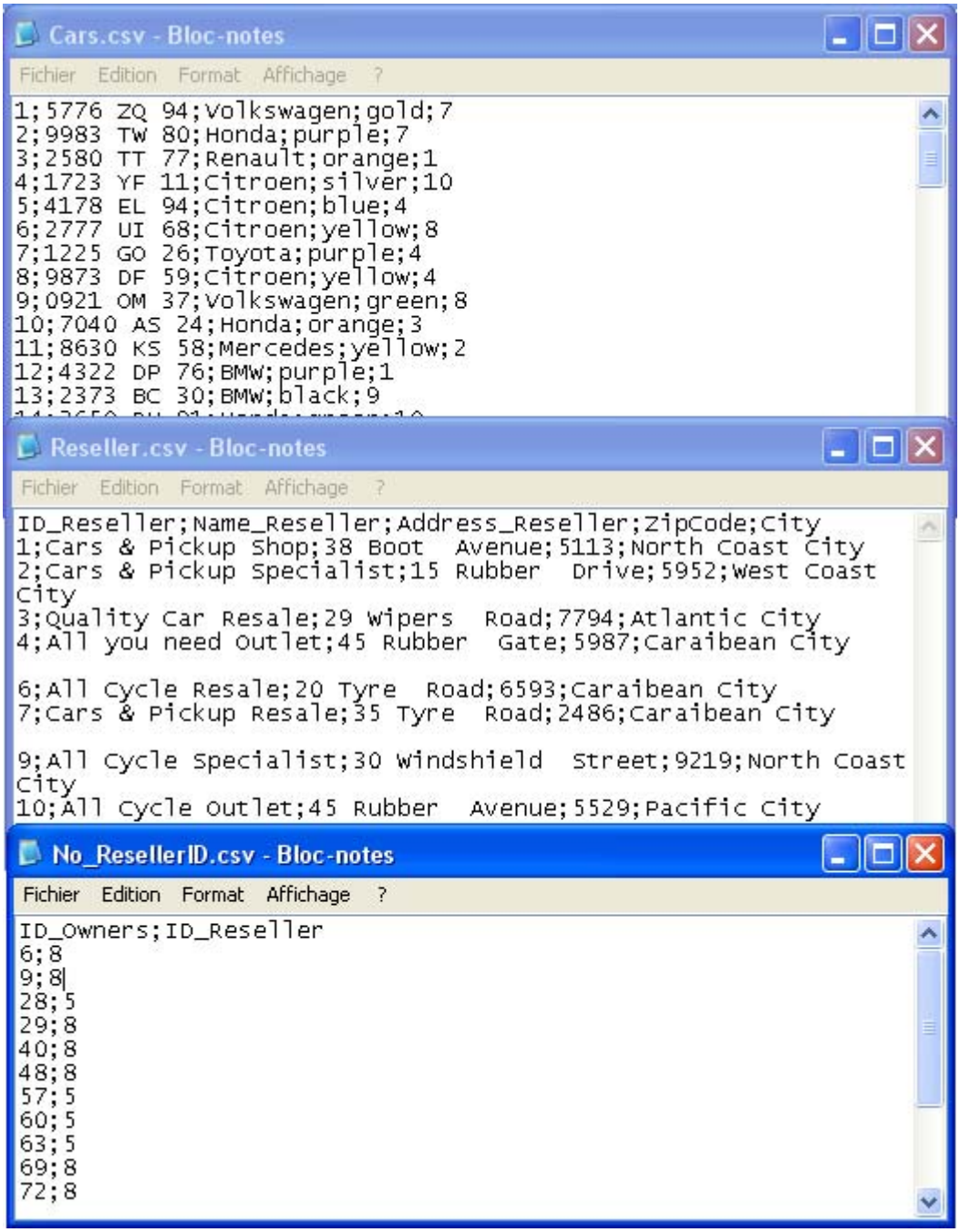
- In the first Inner Join output table (*Reject_OwnerID*), click the plus arrow button to add a filter line and fill it in with the following formula to gather only OwnerID-related rejection: `not defined $Owners_data[ID_Owner]`
- In the second Inner Join output table (*Reject_ResellerID*), repeat the same operation using the following formula: `not defined $Resellers_data[ID_Reseller]`



- Click OK to validate and close the Mapper editor.
- Right-click on the **tMap** component, click on **Row** and select *Reject_ResellerID* in the list.
- Connect the main row from the Mapper to the Reseller Inner Rejection output component



- For this scenario, remove from the Resellers.csv the rows corresponding to Reseller ID 5 and 8.
- Then run the Job by pressing **F6** or from the **Run** panel.



- The four output files are all created in the defined folder (Outputs).
- Notice in the Inner Join reject output file, *NoResellerID.csv*, that the ID_Owners field values matching the Reseller ID 5 and 8 were rejected from the cars file to this file.

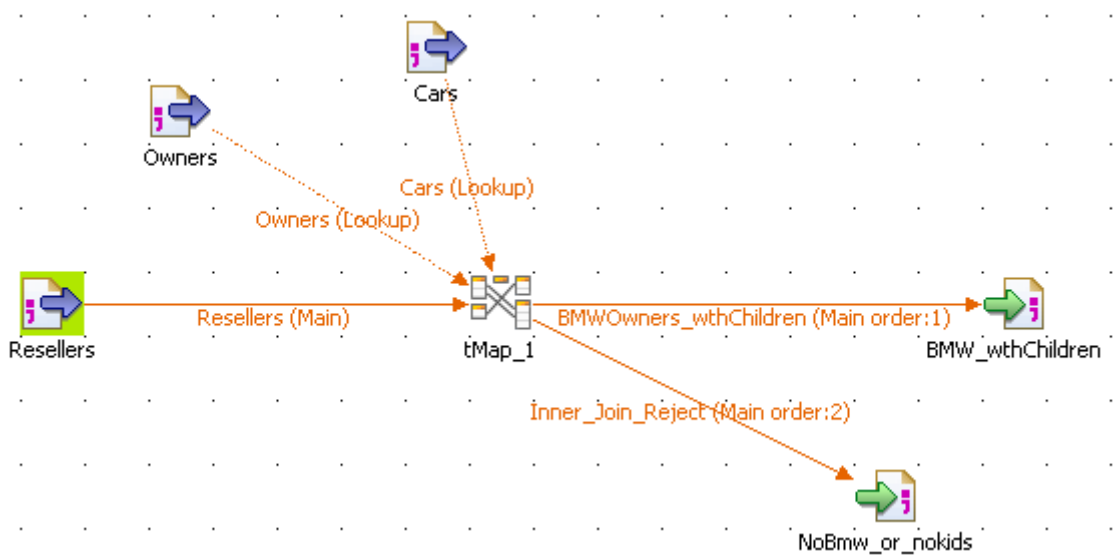
Scenario 3: Cascading join mapping

As third advanced use scenario, based on the scenario 2, add a new Input table containing Insurance details for example.

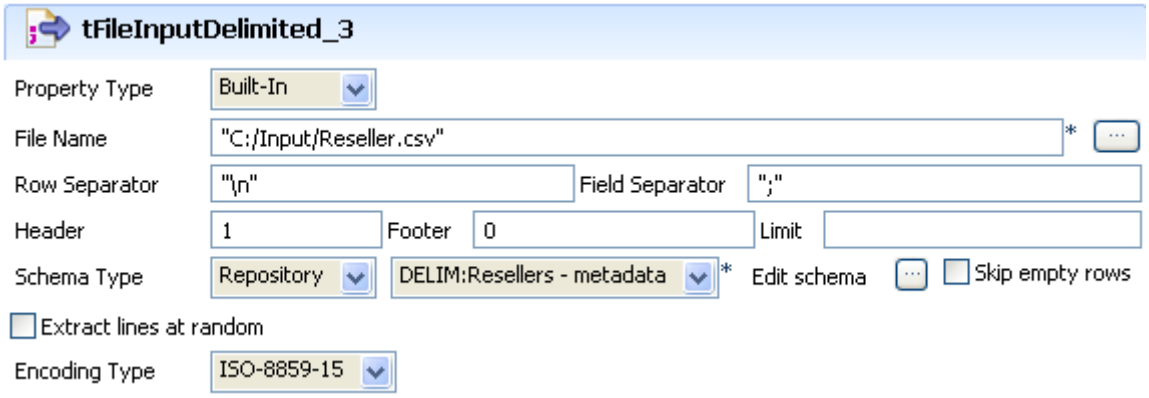
Set up an Inner Join between two lookup input tables (Owners and Insurance) in the Mapper to create a cascade lookup and hence retrieve Insurance details via the Owners table data.

Scenario 4: Advanced mapping with filters, explicit joins and Inner join rejection

This scenario introduces a Job (in Java) **tMap** which allows to find the reseller's customer leads who are owners of a defined make, and have between 2 and 6 children (inclusive), for upsale purpose for example.

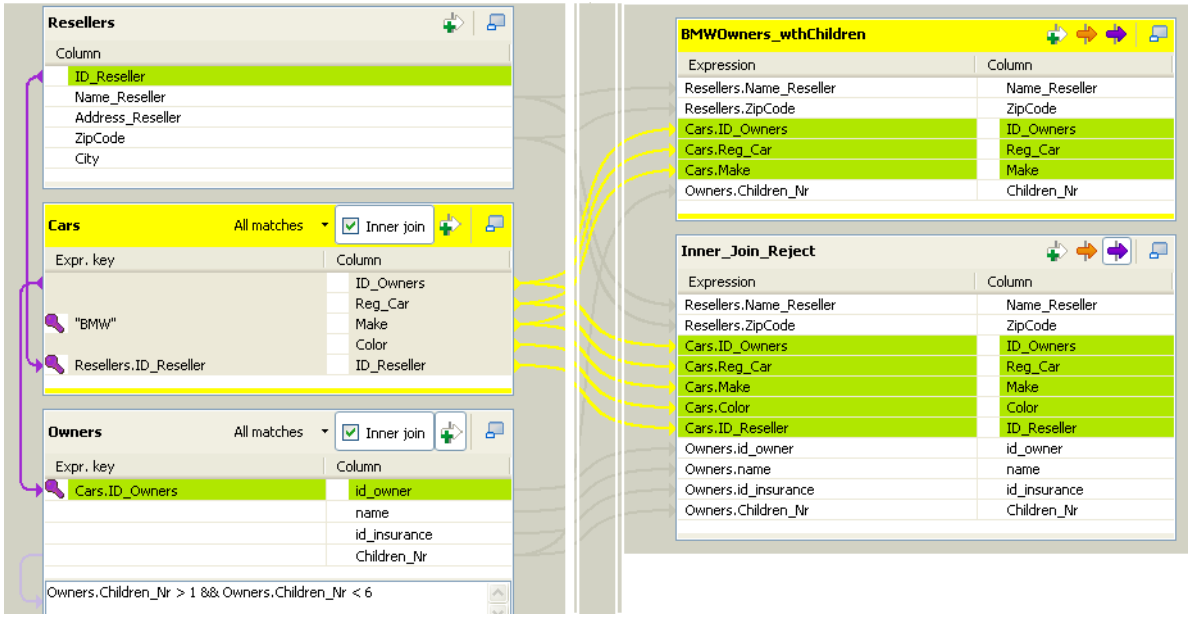


- Drag and drop the following components from the **Palette**: **tFileInputDelimited** (x3), **tMap**, **tFileOutputDelimited** (x2)
- Connect the input flow components to the **tMap** using a **Main row** connection. Pay attention to the file you connect first as it will automatically be set as **Main** flow. And all other connections will thus become **Lookup** flows.
- Define the **Basic settings** of each of the Input components. For example, define the *Resellers* file path used as **Main** flow in the Job.

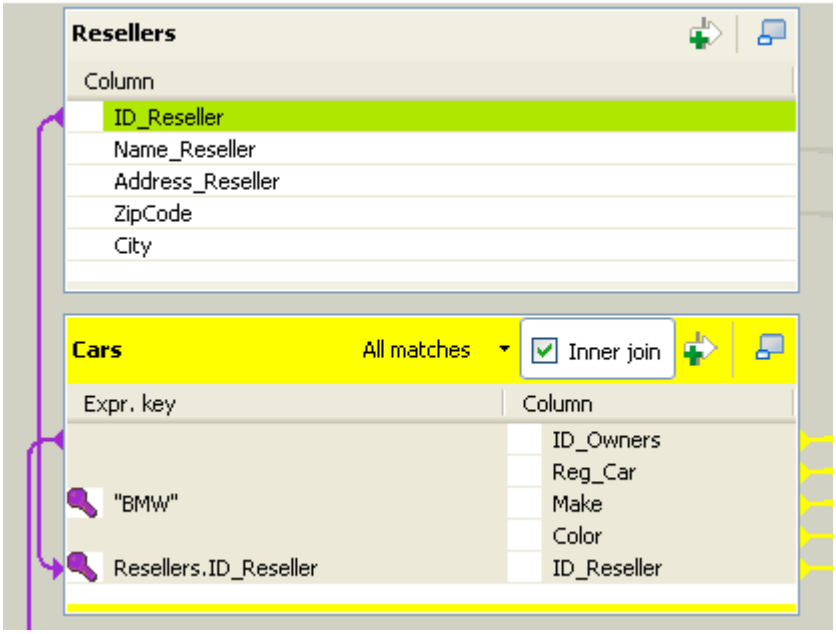


- Define the delimited file to be used, the Row and the Field Separator, the Header and Footer rows if any.
- **Edit the Schema** if it hasn't been stored in the **Repository**. You will retrieve this schema in the **Main** table at the top of the **Input** area of the mapper.

- Carry out these previous steps for the other Input components: *Cars* and *Owners*. These two **Lookup** flows will fill in secondary (lookup) tables in the Input area of the Mapper.
- Then double-click on the **tMap** component to launch the Mapper and define the mapping and filters.

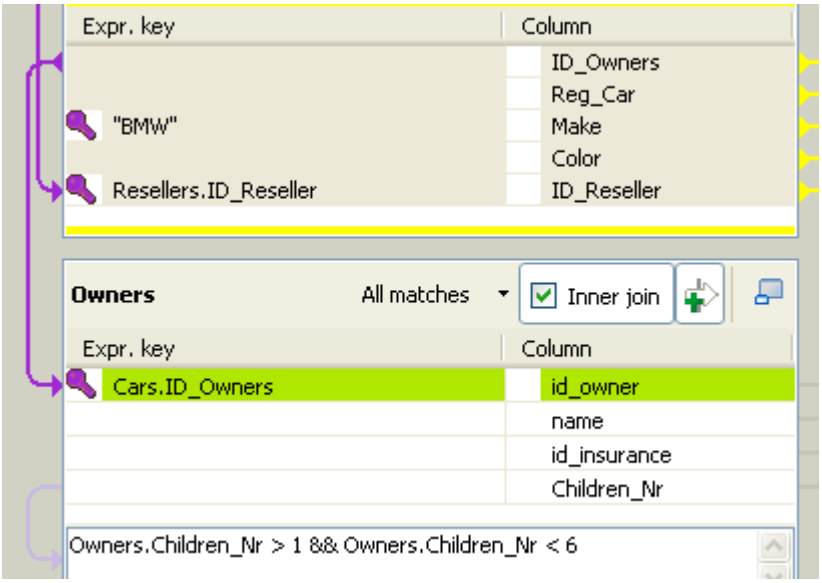


- First set the explicit joins between the **Main** flow and the **Lookup** flows.
- Simply drag & drop the *ID_Resellers* **column** towards the corresponding column and this way fill in the **Expression key** field of the **Lookup** table.

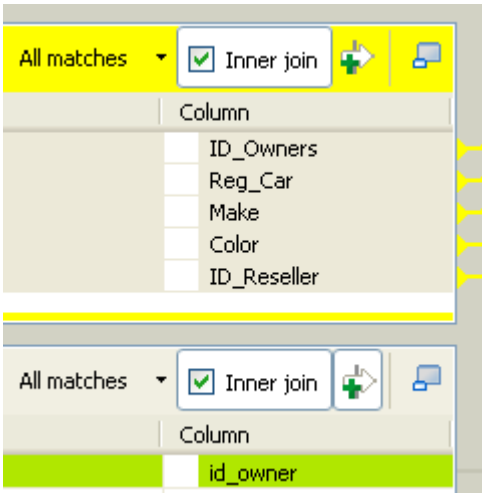


- The explicit join displays in color along with a hash key.
- Then in the **Expr. Key** of the *Make* column, type in (in Java) the filter. In this use case, simply type in *"BMW"* as the search is focused on the Owners of this particular Make.

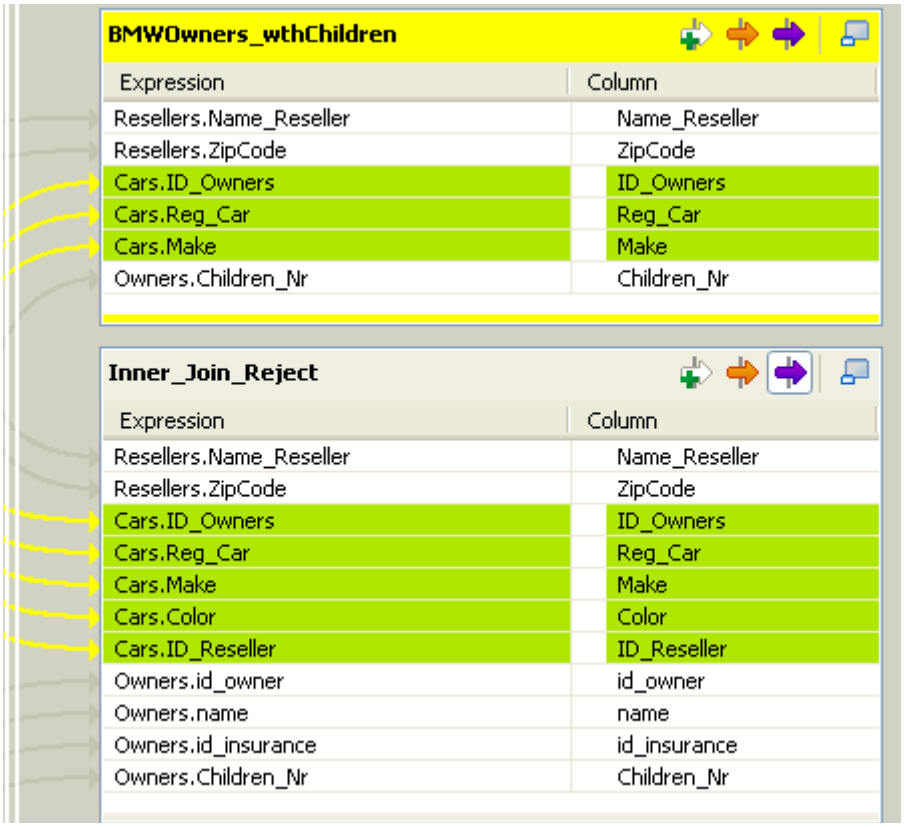
- Implement a cascading join between the two lookup tables *Cars* and *Owners*, in order to retrieve owners information regarding the number of children they have.



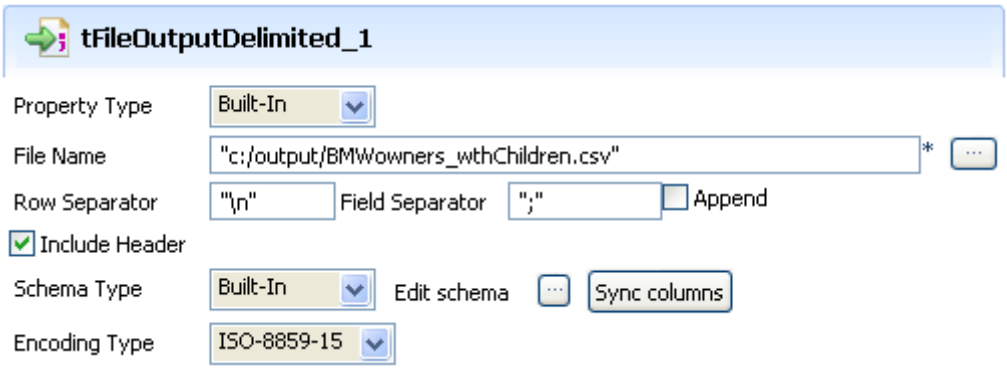
- Simply drag and drop the *ID_Owners* column from the *Cars* table towards the **Expr. Key** field of the *id_owner* column from the *Owners* table.
- Click the **Filter** button next to the **Inner Join** button to display the **Filter** expression area.
- Type in the **Filter** statement to narrow down the number of rows loaded in the **Lookup** flows. In this use case, the statement reads: `Owners.Children_Nr > 1 && Owners.Children_Nr < 6`
- Then, as you want to reject the null values into a separate table and exclude them from the standard output, select the **Inner Join** check box for each of the filtered **Lookup** tables.



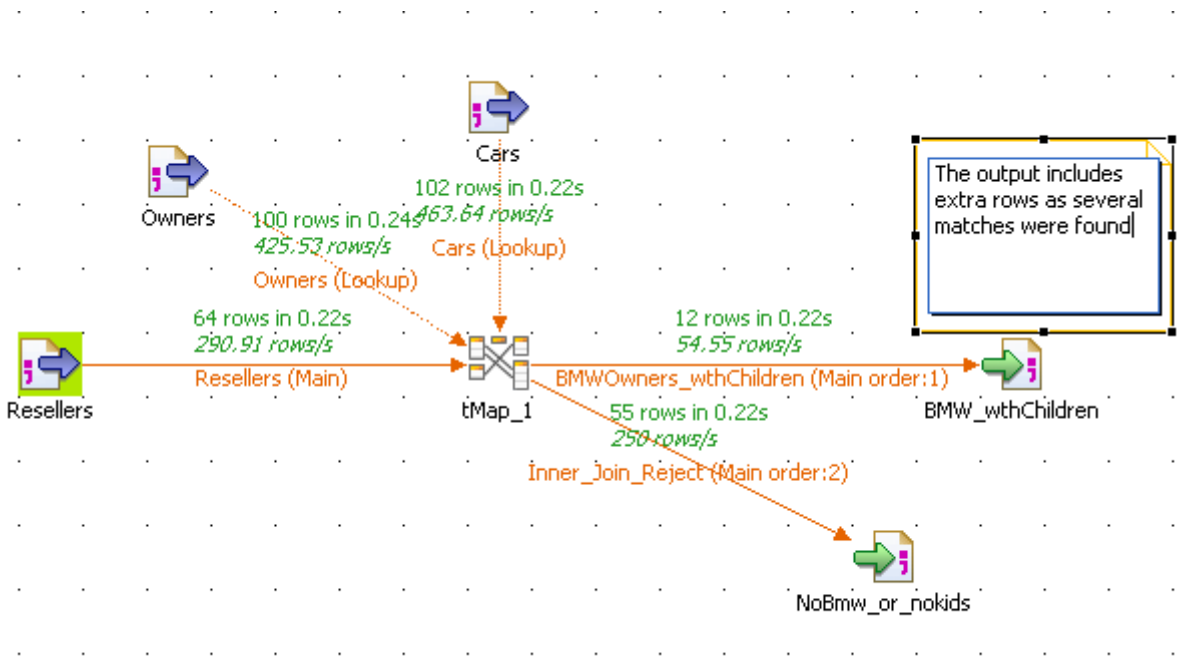
- In the Inner join, you can then choose to include only a **Unique match, the First match or All Matches**. In this use case, the **All matches** option is selected. Thus if several matches are found in the Inner Join, i.e. rows matching the explicit join as well as the filter, all of them will be added to the output flow (either in Rejection or the regular output).
- Then on the **Output** area of the **Mapper**, add two tables, one for the full matches and one for the rejections.
- Click the plus button to add the tables and give a name to the outputs.



- Drag and drop data from the **Main** and **Lookup** tables in the Input area, towards the respective output tables, following the type of information you want to fetch.
- In the rejection table used to direct the non-matches from the external join or the filter, click the **Inner Join Reject** button (violet arrow) to activate it.
- On the **design** space, right-click on the **tMap** and pull the respective output link to the relevant components.
- Define the **Basic settings** of the **Output** components.



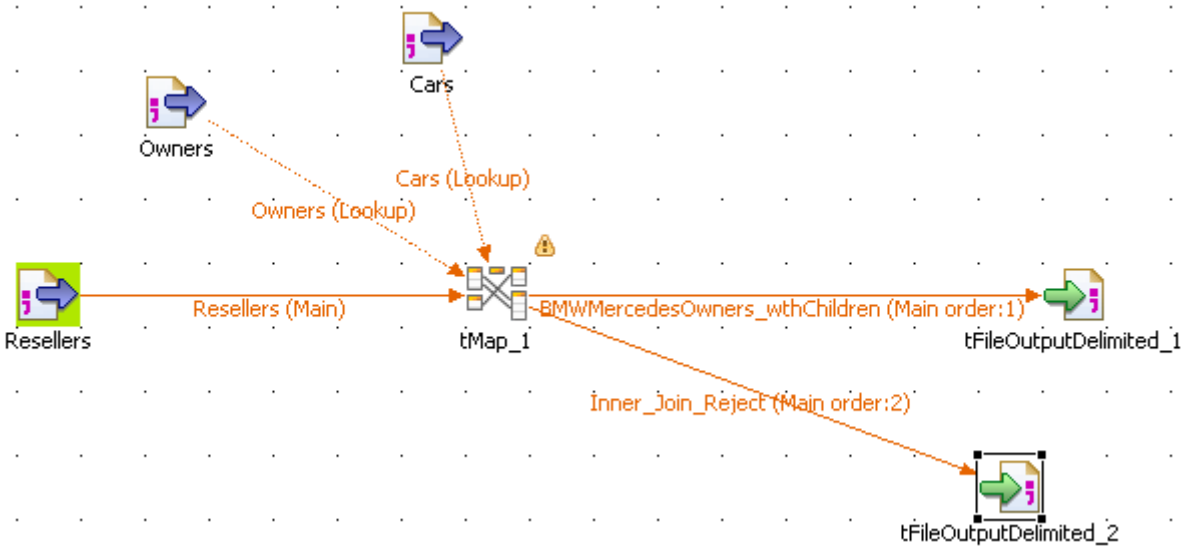
- Define the filepath, the expected Row and Field separator. And for this use case, select the **Include Header** check box.
- The **Schema** should be automatically propagated from the **Mapper**.
- Save you Job, then go to the **Run** tab and select the **Statistics** check box to follow the processing thread.



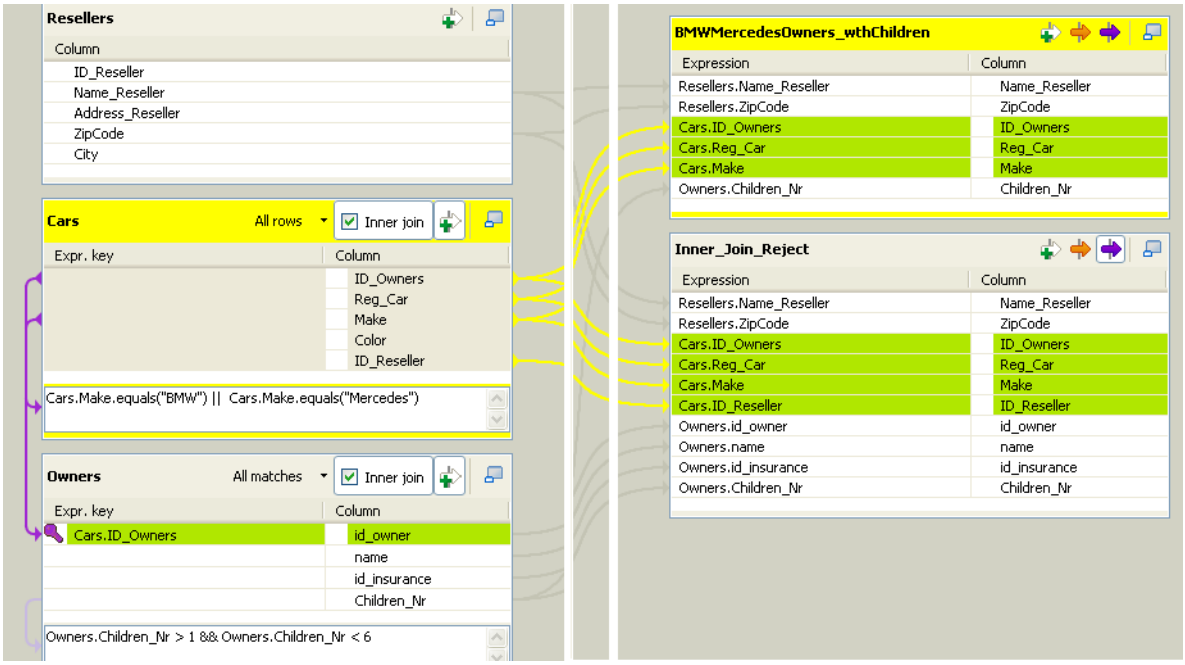
The statistics show that several matches were found and therefore the sum of the output rows (Main + rejected) exceeds the **Main** flow input rows.

Scenario 5: Advanced mapping with filters and a check of all rows

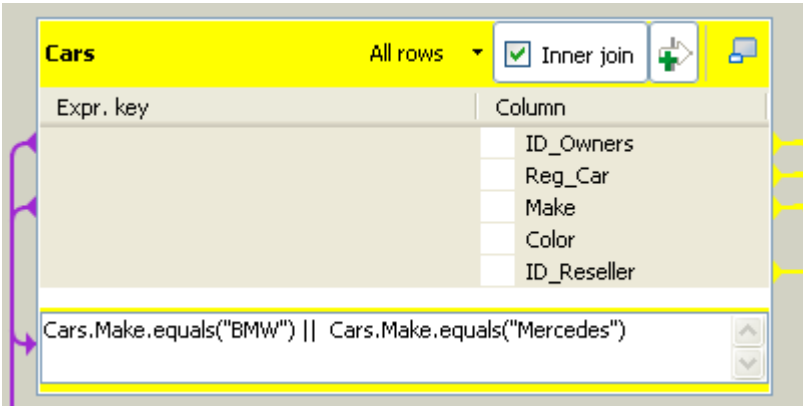
This scenario is a modified version of the preceding scenario. It describes a Job that applies filters and then checks each row of loaded lookup rows.



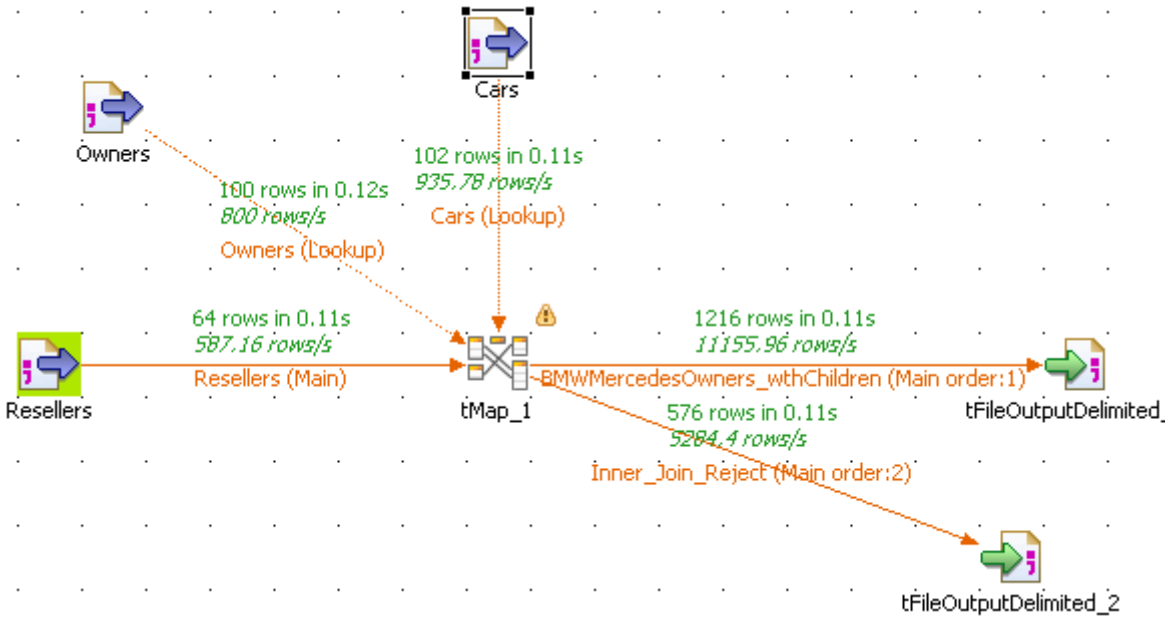
- Take the same Job as in *Scenario 4: Advanced mapping with filters, explicit joins and Inner join rejection on page 760*.
- No changes are required in the Input delimited files.
- Launch the Mapper to change the mapping and the filters.



- Remove all explicit joins between the **Main** table and the *Cars* **Lookup** table.
- Notice that the **All Matches** setting changes automatically to **All Rows**. In fact, as no explicit join is declared (no hash keys), all lookup rows need to be loaded and checked against all main flow rows.
- Remove the **Expr. key** filter (*"BMW"*) from the *Cars* table.



- And click the **Filters** button to display the **Filter** area. Then type in the new filter to narrow down the search to *BMW* or *Mercedes* car makes. The statement reads as follows:
  `Cars.Make.equals("BMW") || Cars.Make.equals("Mercedes")`
- The filter on the *Owners* **Lookup** table doesn't change from the previous scenario.
- Define new file paths for the respective outputs.
- Save the Job and enable the **Statistics** on the **Run** tab before executing the Job.



The Statistics show that a cartesian product has been carried out between the Main flow rows with the filtered Lookup rows.

BMWMercedes_wthChildren.csv

```
34  Cars & Pickup Specialist;5952;76;2251 JG 82;Mercedes;5
35  Cars & Pickup Specialist;5952;79;2930 CP 77;Mercedes;5
36  Cars & Pickup Specialist;5952;96;8506 ZQ 08;BMW;4
37  Cars & Pickup Specialist;5952;97;7757 KQ 65;BMW;5
38  Cars & Pickup Specialist;5952;99;9162 MC 60;Mercedes;4
39  Cars & Pickup Specialist;5952;100;0146 DA 20;BMW;5
40  Quality Car Resale;7794;9;9939 CJ 88;Mercedes;3
41  Quality Car Resale;7794;15;4563 ZB 33;BMW;3
42  Quality Car Resale;7794;20;3408 EW 35;Mercedes;2
43  Quality Car Resale;7794;27;5792 QT 18;BMW;3
44  Quality Car Resale;7794;33;8253 DP 32;BMW;3
45  Quality Car Resale;7794;44;3748 NN 21;BMW;2
46  Quality Car Resale;7794;45;4065 EA 69;Mercedes;3
```

The content of the main output flow shows that the filtered rows have correctly been passed on.

MWMercedes_wthChildren.csv | BMWMercedes_wthchildren_InnerReject.csv

```
1  Name_Reseller;ZipCode;ID_Owners;Reg_Car;Make;ID_Reseller;id_owner;name;
   id_insurance;Children_Nr
2  Cars & Pickup Shop;5113;16;6709 YE 10;BMW;27;16;bobouh;QYW1412;1
3  Cars & Pickup Shop;5113;37;5898 EB 09;BMW;54;37;hirtgall;MFR4898;1
4  Cars & Pickup Shop;5113;65;0439 XF 39;BMW;32;65;mauvaes;FBG6516;6
5  Cars & Pickup Shop;5113;68;8147 RS 83;Mercedes;33;68;bouhle;NGT4401;6
6  Cars & Pickup Shop;5113;80;1359 DY 17;Mercedes;8;80;gallgall;GAC9240;6
7  Cars & Pickup Shop;5113;86;0094 SH 41;BMW;26;86;otbo;OFU7978;6
8  Cars & Pickup Shop;5113;92;6544 LF 76;BMW;50;92;otmau;XNH2512;6
```

Whereas, the Reject result clearly shows the rows that didn't match one of the filter.

Scenario 6: Advanced mapping with lookup reload at each row (java)

The following scenario describes a Job that retrieves people details from a lookup database, based on a join on the age. The main flow source data is read from a MySQL database table called *people_age* that contains people details such as numeric id, alphanumeric first name and last name and numeric age. The people age is either 40 or 60. The number of records in this table is intentionally restricted.

The reference or lookup information is also stored in a MySQL database table called *large_data_volume*. This lookup table contains a number of records including the city where people from the main flow have been to. For the sake of clarity, the number of records is restricted but, in a normal use, the usefulness of the feature described in the example below is more obvious for very large reference data volume.

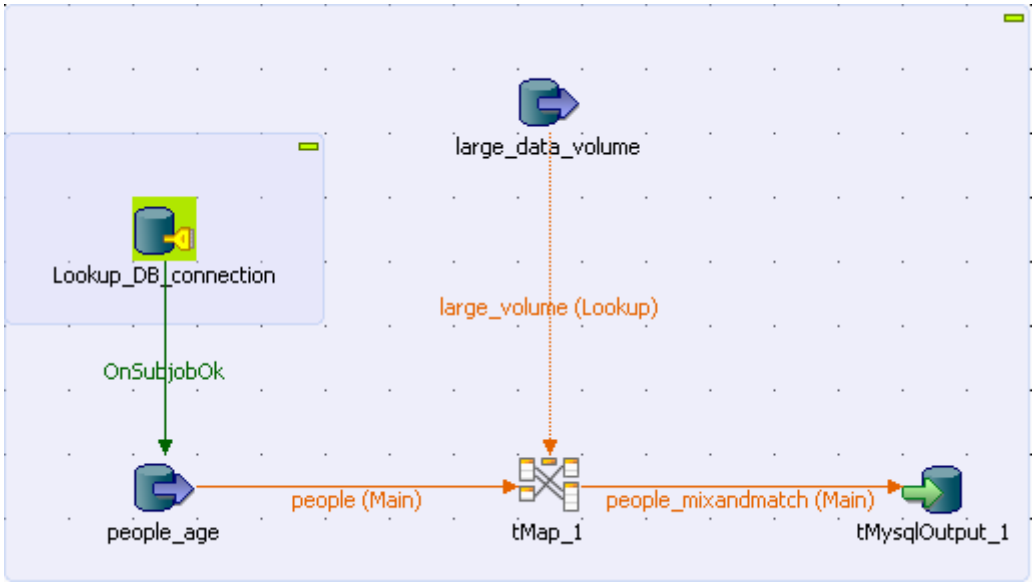
To optimize performance, a database connection component is used in the beginning of the Job to open the connection to the lookup database table in order not to do that every time we want to load a row from the lookup table.

An Expression Filter is applied to this lookup source flow, in order to select only data from people whose age is equal to 60 or 40. This way only the relevant rows from the lookup database table are loaded for each row from the main flow.

Therefore this Job shows how, from a limited number of main flow rows, the lookup join can be optimized to load only results matching the expression key.

Generally speaking, as the lookup loading is performed for each main flow row, this option is mainly interesting when a limited number of rows is processed in the main flow while a large number of reference rows are to be looked up to.

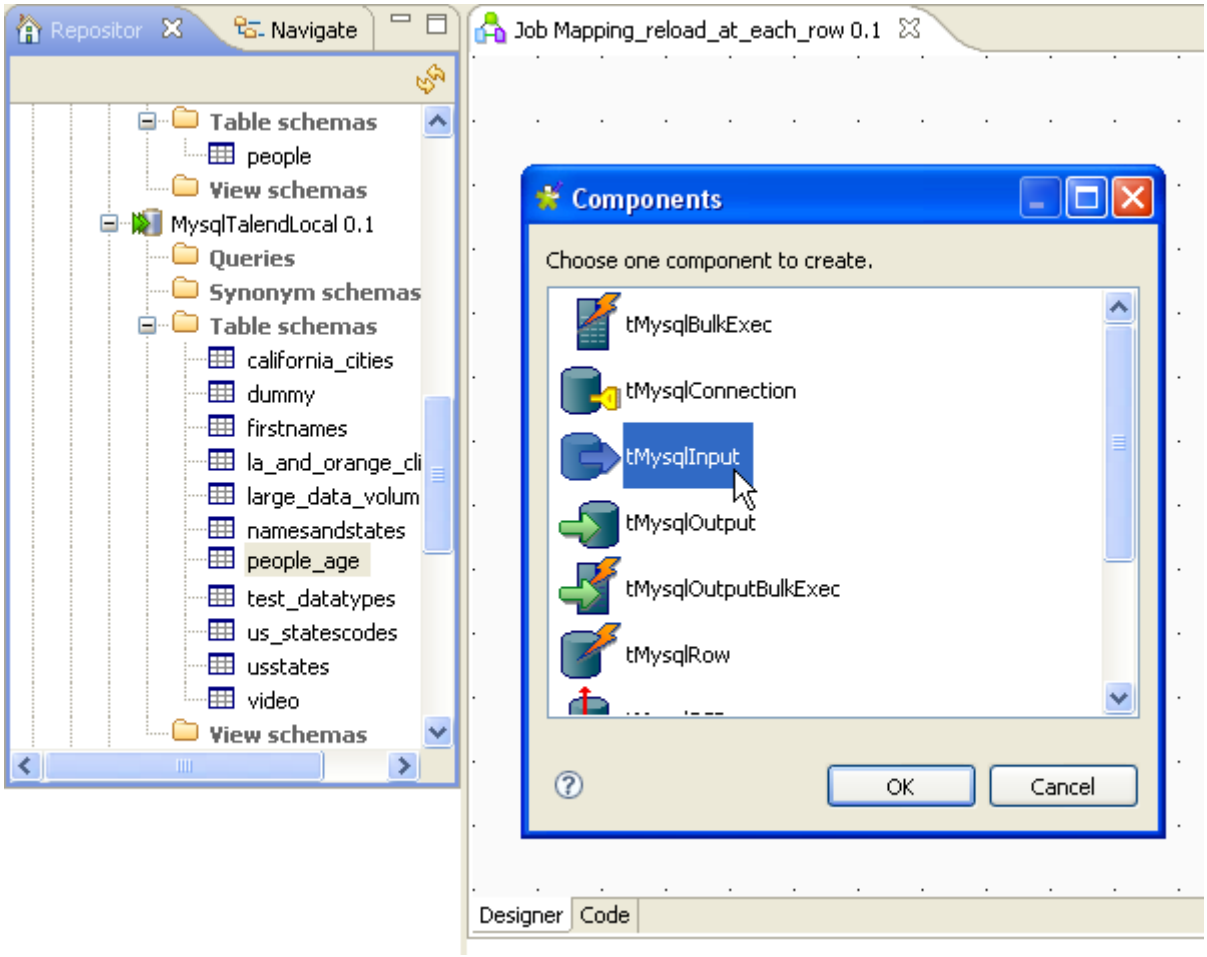
The join is solved on the *age* field. Then, using the relevant loading option in the **tMap** component editor, the lookup database information is loaded for each main flow incoming row.



For this Job, the metadata has been prepared for the source and connection components. For more information on how to set up the DB connection schema metadata, see the relevant section in the **Talend Open Studio** User Guide.

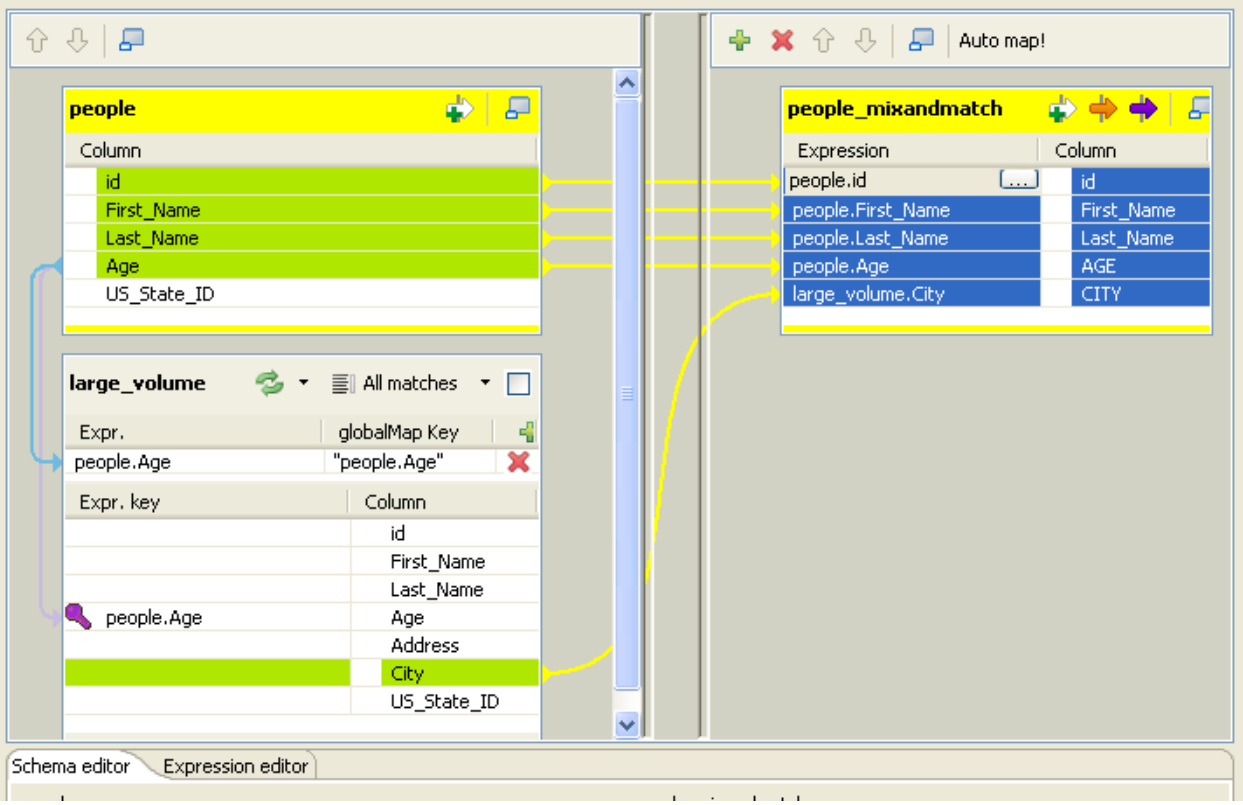
This Java Job is formed with five components, four database components and a mapping component.

- Drop the DB Connection under the **Metadata** node of the **Repository** to the design workspace. In this example, the source table is called *people_age*.
- Select **tMySQLInput** in the list that pops up when dropping the component.



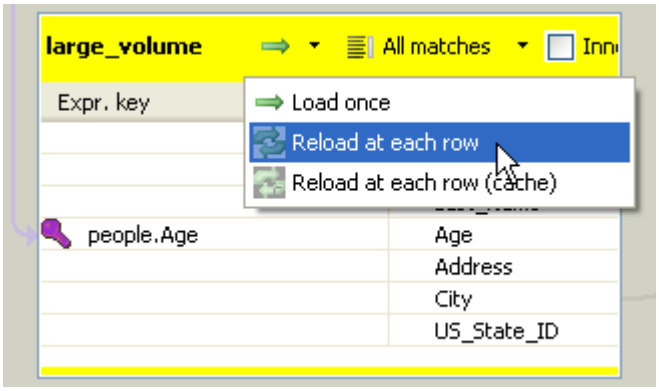
- Drop the lookup DB connection table from the **Metadata** node to the design workspace selecting **tMysqlInput** from the list that pops up. In this Job, the lookup is called *large_data_volume*.
- The same way, drop the DB connection from the **Metadata** node to the design workspace selecting **tMysqlConnection** from the list that pops up. This component creates a permanent connection to the lookup database table in order not to do that every time we want to load a row from the lookup table.
- Then pick the **tMap** connector from the **Processing** family and the **tMySQLOutput** connector from the **Database** family in the **Palette** to the right hand side of the editor.
- Now connect all the components together. To do so, right-click on the **tMySQLInput** component corresponding to the *people* table and drag the link towards **tMap**.
- Release the link over the **tMap** component, the main row flow is automatically set up.
- Rename the **Main** row link to *people*, to identify more easily the main flow data.
- Perform the same operation to connect the lookup table (*large_data_volume*) to the **tMap** component and the **tMap** to the **tMySQLOutput** component.

- A dialog box prompts for a name to the output link. In this example, the output flow is named: *people_mixandmatch*
- Rename also the lookup row connection link to *large_volume*, to help identify the reference data flow.
- Connect **tMysqlConnection** to **tMysqlInput** using the trigger link **OnSubjobOk**.
- Then double-click the **tMap** component to open the graphical mapping editor.



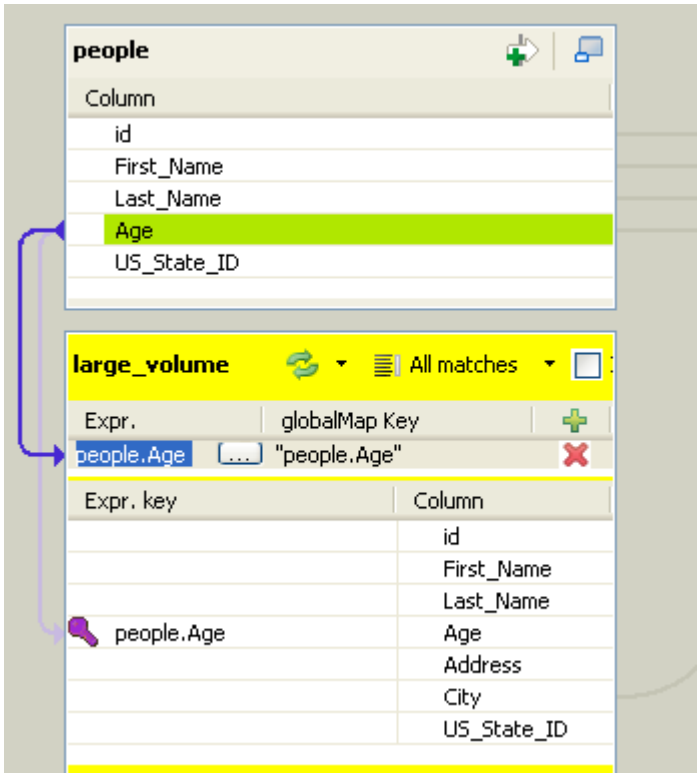
- The **Output** table (that was created automatically when you linked the **tMap** to the **tMySQLOutput** will be formed by the matching rows from the lookup flow (*large_data_volume*) and the main flow (*people_age*).
- Select the main flow rows that are to be passed on to the output and drag them over to paste them in the Output table (to the right hand side of the mapping editor).
- In this example, the selection from the main flow include the following fields: *id*, *first_name*, *last_Name* and *age*.
- From the lookup table, the following column is selected: *city*.
- Drop the selected columns from the input tables (*people* and *large_volume*) to the output table.
- Now set up the join between the main and lookup flows.
- Select the *age* column of the main flow table (on top) and drag it towards the *age* column of the lookup flow table (*large_volume* in this example).
- A key icon appears next to the linked expression on the lookup table. The join is now established.

- Now select the **Reload at each row** option in order for the lookup to be reloaded for each row being processed.



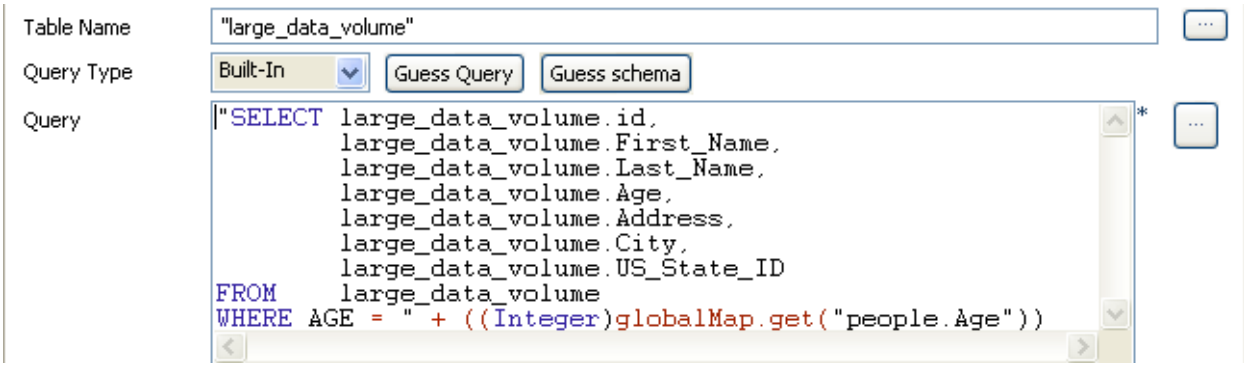
- Click the green arrow to display the pop-up menu and select the relevant option.
- In this use case you also need to select the **All matches** option in the Lookup table, in order to gather all instances of *age* matches in the output flow.
- Now implement the filtering, based on the *age* column, in the Lookup table. The **GlobalMapKey** field is automatically created when you selected the **Reload at each row** option. Indeed you can use this expression to dynamically filter the reference data in order to load only the relevant information when joining with the main flow.

As mentioned in the introduction of the scenario, the main flow data contains only people whose age is either 40 or 60. To avoid the pain of loading all lookup rows, including ages that are different from 40 and 60, you can use the main flow age as global variable to feed the lookup filtering.

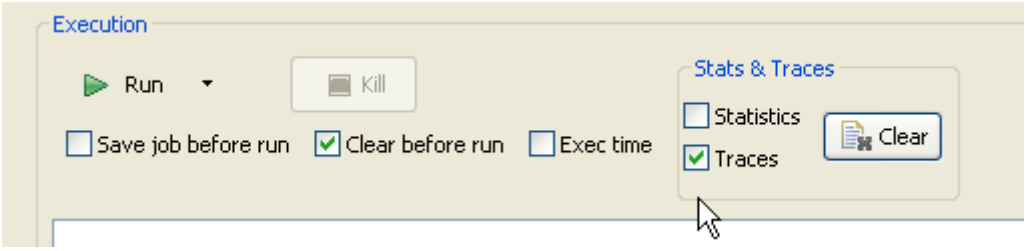


- Drop the *Age* column from the main flow table to the **Expr.** field of the lookup table.

- Then in the **globalMap Key** field, put in the variable name, using the expression. In this example, it reads: `“people.Age”`
- Click **OK** to save the mapping setting and go back to the design workspace.
- To finalize the implementation of the dynamic filtering of the lookup flow, you need now to add a WHERE clause in the query of the database input.

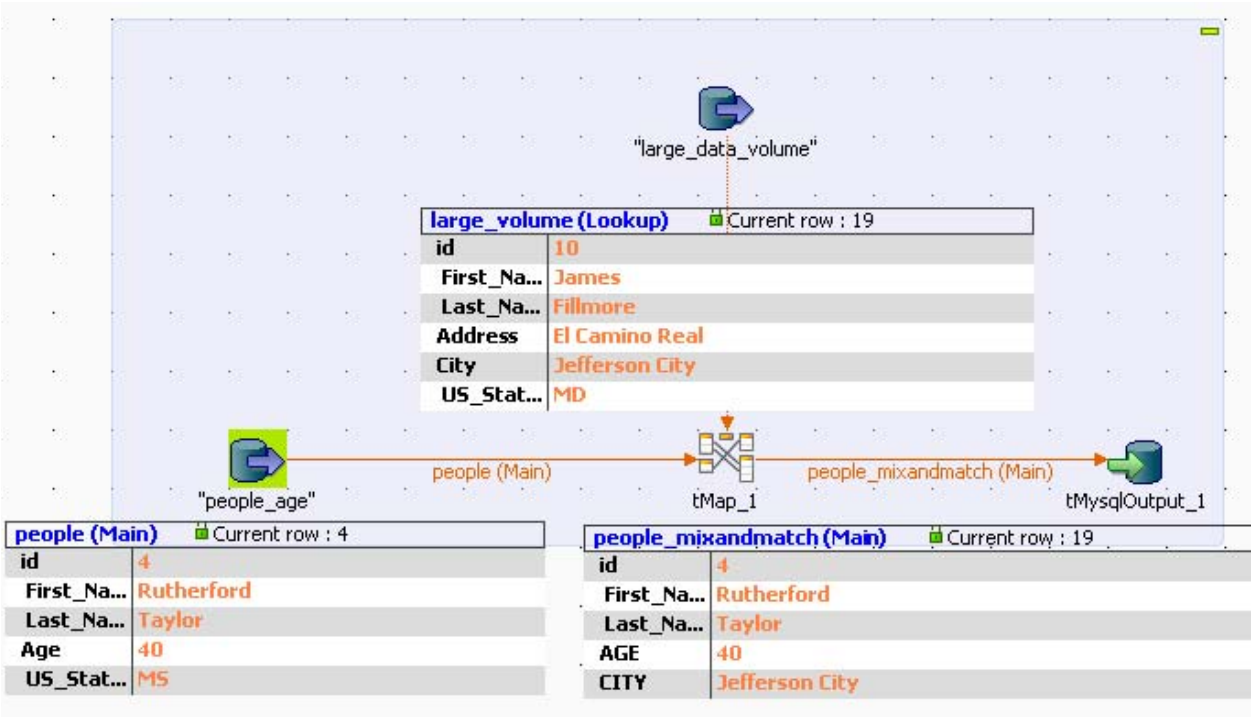


- At the end of the Query field, following the Select statement, type in the following where clause: `WHERE AGE = " + ((Integer)globalMap.get("people.Age"))`
- Make sure that the type corresponds to the column used as variable. In this use case, *Age* is of `Integer` type. And use the variable the way you set in the **globalMap key** field of the map editor.
- Then double-click on the **tMySQLoutput** component and check that the schema corresponds to the mapping setting. You are now ready to execute the Job.
- Click the **Run** view of the tab system located at the bottom of the design workspace, to display the Job execution tab.



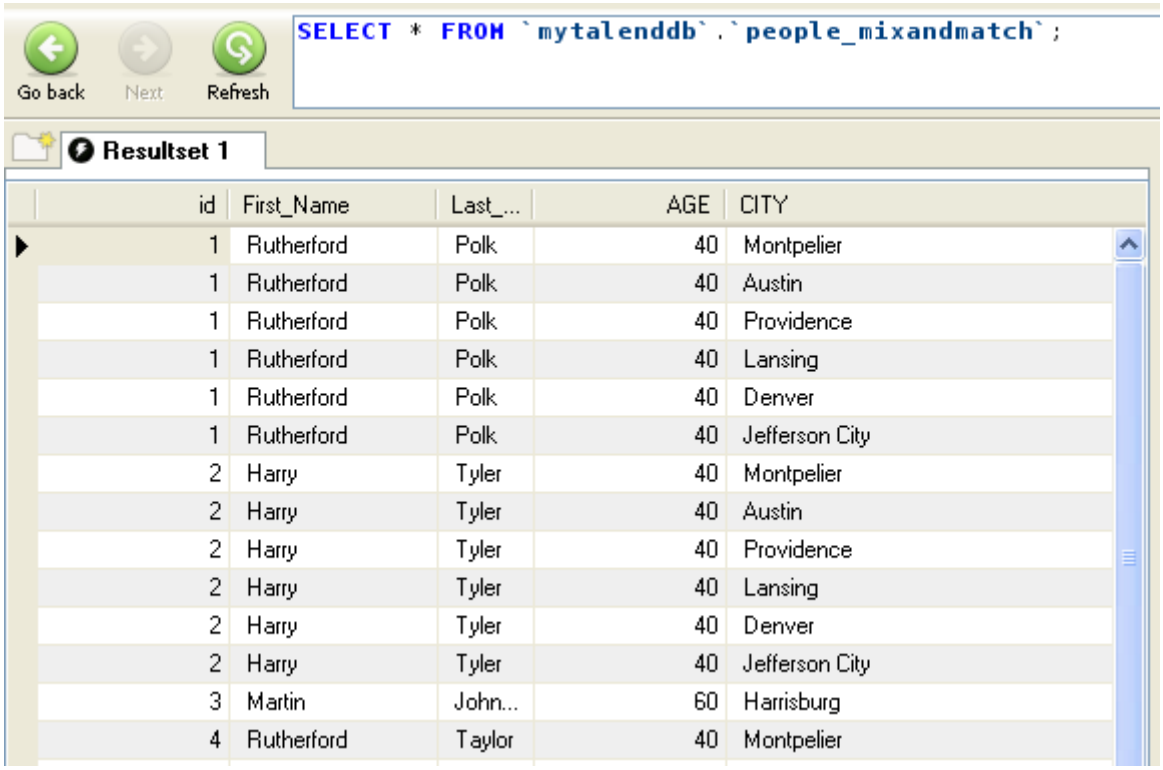
- Before running the Job, select the **Traces** check box to view the data processing progress.
- Make sure that the database types are respected (string, integer...) to avoid execution errors.
- Then press **F6** or click the **Run** button to execute the Job.

For more comfort, you can maximize the Job design view while executing by simply double-clicking on the Job name tab.



The lookup data is reloaded for each of the main flow's rows, corresponding to the age constraint. All *age* matches are retrieved in the lookup rows and grouped together in the output flow.

Therefore if you check out the data contained in the newly created *people_mixandmatch* table, you will find all the *age* duplicates corresponding to different individuals whose age equals to 60 or 40 and the city where they have been to.



id	First_Name	Last_...	AGE	CITY
1	Rutherford	Polk	40	Montpelier
1	Rutherford	Polk	40	Austin
1	Rutherford	Polk	40	Providence
1	Rutherford	Polk	40	Lansing
1	Rutherford	Polk	40	Denver
1	Rutherford	Polk	40	Jefferson City
2	Harry	Tyler	40	Montpelier
2	Harry	Tyler	40	Austin
2	Harry	Tyler	40	Providence
2	Harry	Tyler	40	Lansing
2	Harry	Tyler	40	Denver
2	Harry	Tyler	40	Jefferson City
3	Martin	John...	60	Harrisburg
4	Rutherford	Taylor	40	Montpelier

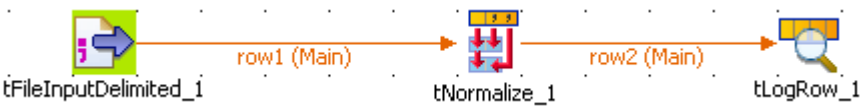
tNormalize

tNormalize Properties

Component family	Processing/Fields	
Function	Normalizes the input flow following SQL standard.	
Purpose	tNormalize helps improve data quality and thus eases the data update.	
Basic settings	*Schema type* and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the Repository. In this component, the schema is read-only.
		Built-in: The schema will be created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused in various projects and job designs. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
Java only	*Get rid of duplicated rows from output*	Select this check box to deduplicate rows in the data of the output flow.
Java only	*Use CSVparameters*	Select this check box to include CSV specific parameters such as **escape mode** and **enclosure** character.
	Column to normalize	Select the column from the input flow which the normalization is based on
	Item separator	Enter the separator which will delimits data in the input flow.
Usage	This component can be used as intermediate step in a data flow.	
Limitation	n/a	

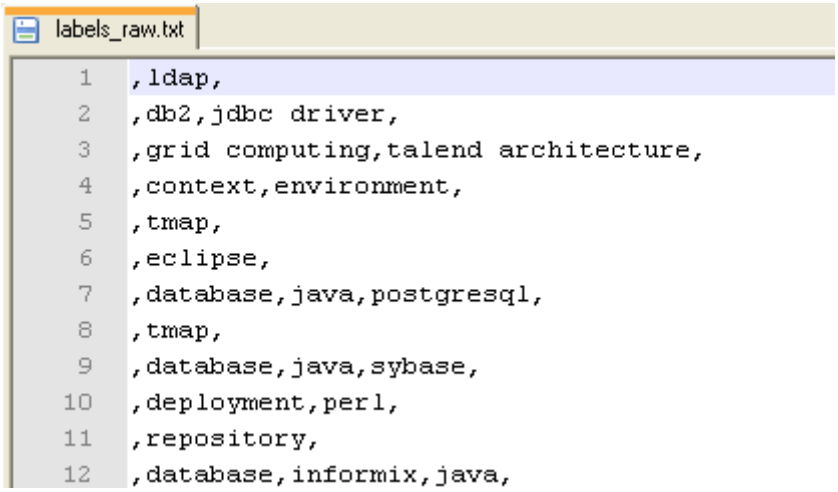
Scenario: Normalizing data (in Perl)

This simple scenario illustrates a Job that normalizes a list of tags for Web forum topics and outputs them into a table in the standard output console (**Run** tab).

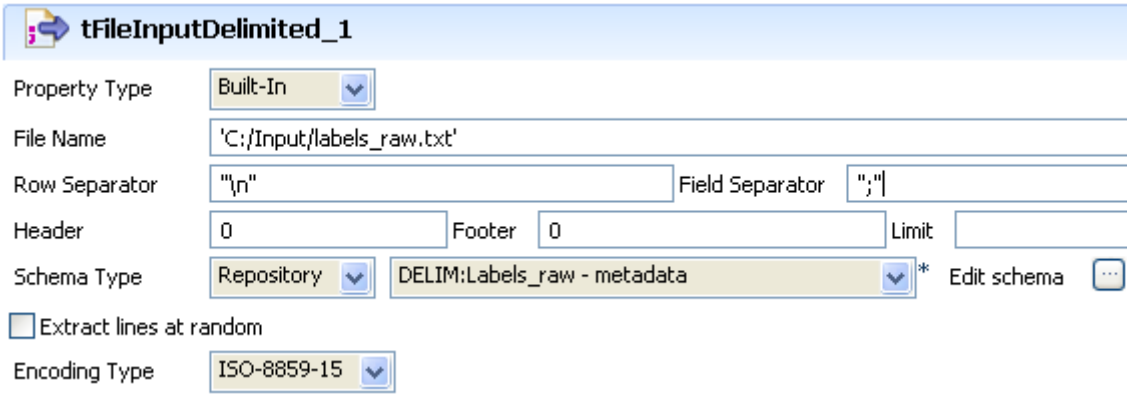


- Drop the following components from the **Palette** to the design workspace: **tFileInputDelimited**, **tNormalize**, **tLogRow**.

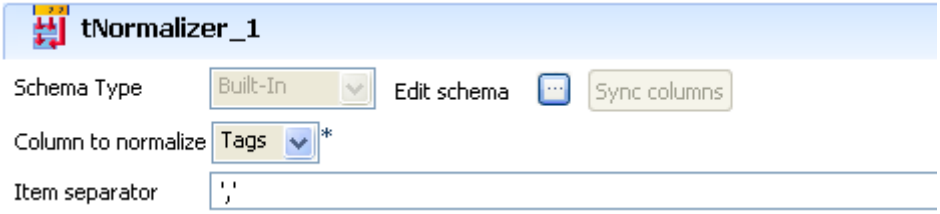
- In the **tFileInputDelimited** Basic settings, set the input file to be normalized.



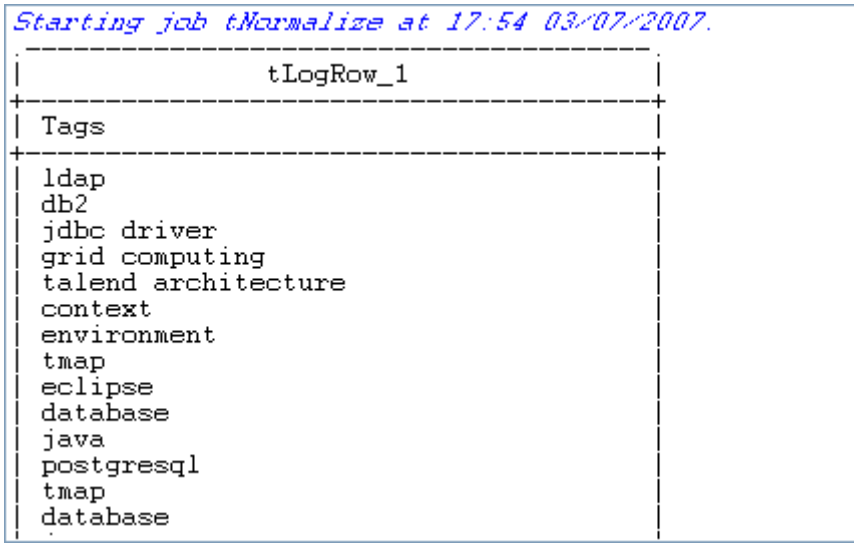
- The file schema is stored in the repository for ease of use. It is made of one column, called *Tags*, containing rows with one or more keywords.
- Set the **Row Separator** and the **Field Separator**.



- On the **tNormalize Basic settings** panel, define the column the normalization operation is based on.
- In this use case, the column to normalize is *Tags*.



- The **Item separator** is the comma, surrounded here by single quotes as the Job is done in Perl.
- In the **tLogRow** component, select the **Print values in the cells of table** check box.
- Save the Job and press **F6** to execute it.



The values are normalized and displayed in a table cell on the console.

tPivotToRows

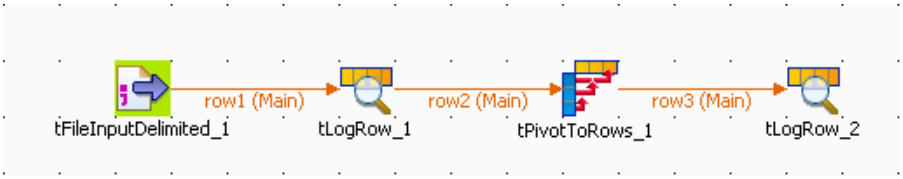
tPivotToRows properties

Component family	Processing	
Function	**tPivotToRows** transforms multiple columns into multiple key/value lines.	
Purpose	**tPivotToRows** makes it possible to choose a list of columns from the input flow and transforms it into lines in the output flow.	
Basic settings	*Schema type* and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the Repository. Click **Edit Schema** to make changes to the schema. Note that if you make changes, the schema automatically becomes built-in. Click **Sync columns** to retrieve the schema from the previous component in the Job.
		Built-in: You create the schema and store it locally for the relevant component. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: You have already created the schema and stored it in the Repository. You can reuse it in various projects and job flowcharts. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	Row keys	Select the list of the columns of the input schema which you want to display as a unique output line. The non-selected columns will constitute the pivot. Click on the plus button to add as many lines as of columns to concatenate. click in each of the lines of the **Input column** list and select the name of the column.
	Row key concatenate delimiter	Set separators for concatenated values.
Advanced settings	*tStatCatcher Statistics*	Select this check box to gather the job processing metadata at the job level as well as at each component level.
Usage	This component is generally used as an intermediate step in a data flow. It needs then an input and output components.	

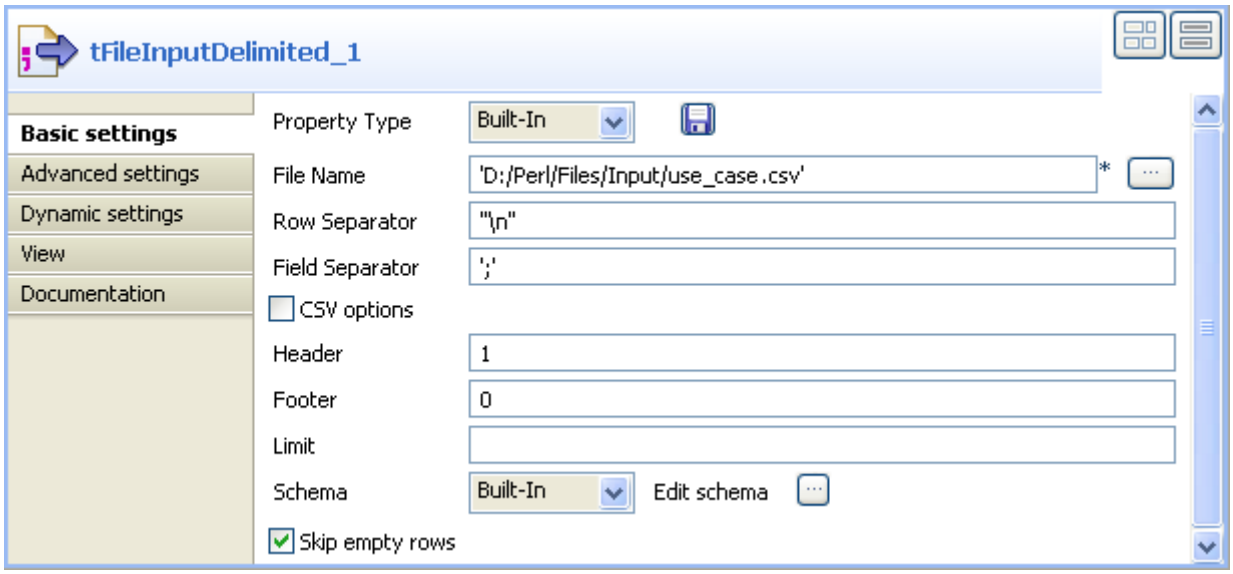
Scenario: Concatenating a list of columns in a table by using the other table columns as pivot

This Perl scenario describes a four-component Job that allows to concatenate in a single line the information distributed in a list of columns by using the other columns in the table as pivot.

- Drop the following components from the **Palette** onto the design workspace: **tFileInputDelimited**, **tPivotToRows** and **tLogRow** (x2).
- Connect the four components using **Row Main** links.



- In the design workspace, double-click **tFileInputDelimited** to display its **Basic settings** view where you can define its properties.



- From the **Property Type** list, select:
  -**Repository** if you have already stored the metadata of your input file in the Repository, the fields that follow are filled in with the stored information automatically, or
  -select **Built-In** and fill in the fields that follow manually.
  For this example, we use the **Built-In** mode.
- Click the three-dot button next to the **File Name** field and browse to the input file. In this example, our source file is *use_case_tunpivotrow* and it holds eight columns. The five columns: *id*, *CustomerName*, *CustomerAddress*, *id2* and *RegisterTime* are to be concatenated, while the other three columns: *Sum1*, *Sum2* and *Sum3* are to be used as pivot.
- In the **Basic settings** view of **tFileInputDelimited**, define in the corresponding fields the row and field separators used in the source file.
- If needed, set **Header**, **Footer** and **Limit**.
  In this example, set **Header** to 1 since the first row that holds columns' names is to be ignored. **Footer** and **Limit** for the number of processed rows are not set.

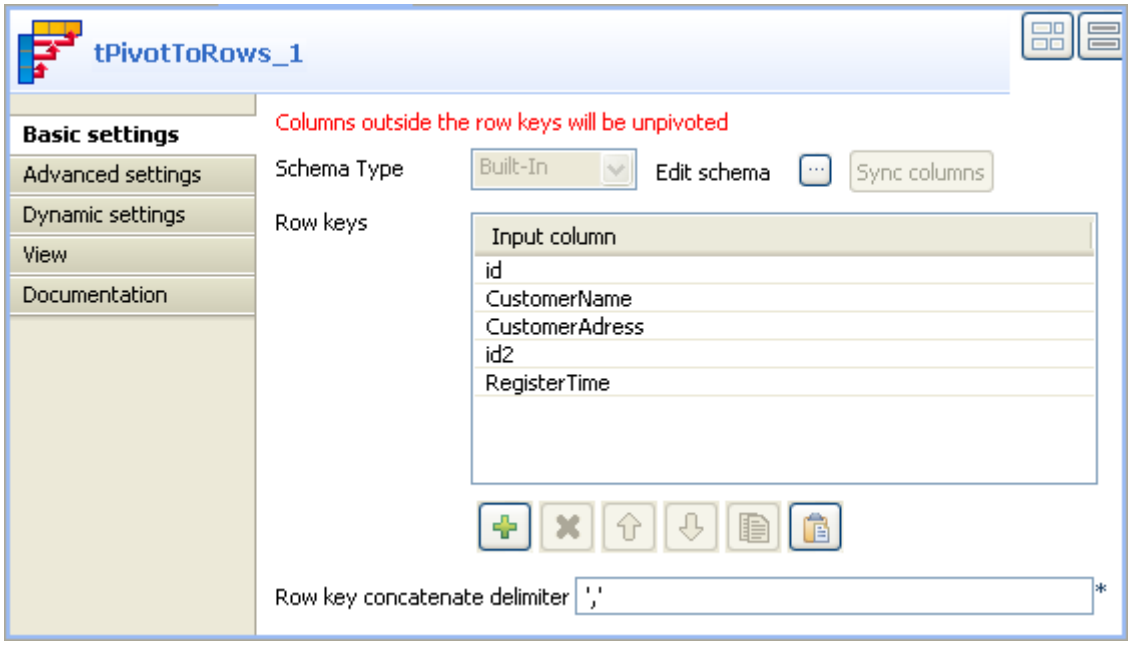
- In the **Schema** field, set schema to **Built in** then click the three-dot button next to the **Edit Schema** field to define the data to be passed to the following component. In this example, the source schema consists of the eight columns of the source file *use_case_tunpivotrow.*

Column	Key	Type	Nullable
id	☐	int	☑
CustomerName	☐	string	☑
CustomerAdress	☐	string	☑
id2	☐	int	☑
RegisterTime	☐	string	☑
Sum1	☐	int	☑
Sum2	☐	int	☑
Sum3	☐	int	☑

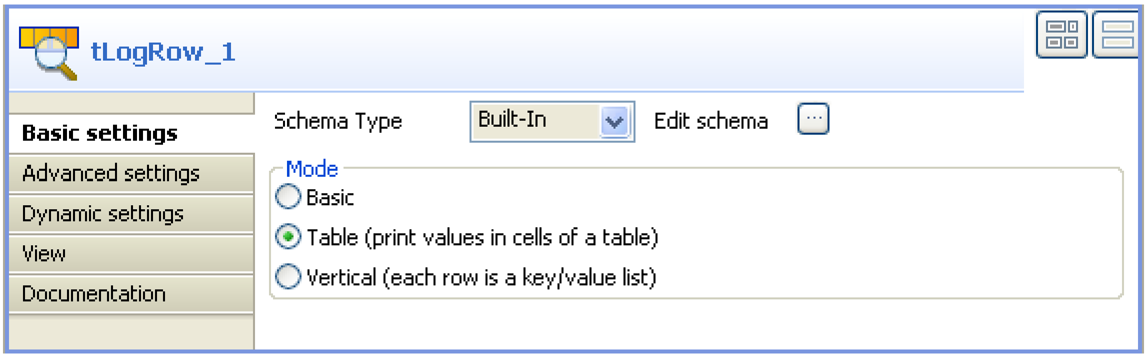
- In the design workspace, double-click **tPivotToRows** to display its **Basic settings** view where you can define the component properties.
- Click **Sync columns** to retrieve the schema from the preceding component.

You can click the three-dot button next to **Edit schema** to check the retrieved schema.

- Click the plus button to add in the **Row keys** area as many lines as the columns to concatenate. In this example, we add five lines.



- Click in each of the lines of the **Input column** list and select the name of the column to be concatenated. The input schema columns that are not selected will be used as pivot.
- In the **Row key concatenate delimiter**field, define a character to separate the data of the various columns once the concatenation is completed.
- Double-click the first **tLogRow** component to display its **Basic settings** view and define its properties.



- In the mode area, select **Table** to display the source file and the **tPivotToRows** results together to be able to compare them.
- Do the same for the second **tLogRow** component
- save the Job and press **F6** to execute it.

```
Starting job test2 at 14:29 30/07/2009.

[stat] connecting to socket on port 4008 ...
[stat] connected
.------------------------------------------------------------------------------------------------------------------------
|                                                      tLogRow_1
+----+---------------------------------+--------------------------+-----+-------------------------+-----------+------------+--
| id | CustomerName                    | CustomerAdress           | id2 | RegisterTime            | Sum1      | Sum2       | S
+----+---------------------------------+--------------------------+-----+-------------------------+-----------+------------+--
|  1 | Griffith Paving and Sealcoatin  | talend@apres91           |  41 | 2001-01-17 06:26:40.000 | 67852.0   | 61521.484  |
|  2 | Bill's Dive Shop                | 511 Maple Ave.  Apt. 1B  |   5 | 2002-06-07 09:40:00.000 | 88792.0   |   15434.1  |
|  3 | Childress Child Day Care        | 662 Lyons Circle         |  28 | 1990-04-01 21:00:00.000 | 35340.0   |  17856.88  |
|  4 | Facelift Kitchen and Bath       | 220 Vine Ave.            |  15 | 1972-04-23 18:00:00.000 |  6097.0   |  55560.24  | 1
'----+---------------------------------+--------------------------+-----+-------------------------+-----------+------------+--
.----------------------------------------------------------------------------------------------------------.
|                                              tLogRow_2                                                   |
+----------------------------------------------------------------------------+-----------+-----------------+
| row_key                                                                    | pivot_key | pivot_value     |
+----------------------------------------------------------------------------+-----------+-----------------+
| 1,Griffith Paving and Sealcoatin,talend@apres91,41,2001-01-17 06:26:40.000 | Sum1      |         67852.0 |
| 1,Griffith Paving and Sealcoatin,talend@apres91,41,2001-01-17 06:26:40.000 | Sum2      |       61521.484 |
| 1,Griffith Paving and Sealcoatin,talend@apres91,41,2001-01-17 06:26:40.000 | Sum3      |          4949.0 |
| 2,Bill's Dive Shop,511 Maple Ave.  Apt. 1B,5,2002-06-07 09:40:00.000       | Sum1      |         88792.0 |
| 2,Bill's Dive Shop,511 Maple Ave.  Apt. 1B,5,2002-06-07 09:40:00.000       | Sum2      |         15434.1 |
| 2,Bill's Dive Shop,511 Maple Ave.  Apt. 1B,5,2002-06-07 09:40:00.000       | Sum3      |          6068.0 |
| 3,Childress Child Day Care,662 Lyons Circle,28,1990-04-01 21:00:00.000     | Sum1      |         35340.0 |
| 3,Childress Child Day Care,662 Lyons Circle,28,1990-04-01 21:00:00.000     | Sum2      |        17856.88 |
| 3,Childress Child Day Care,662 Lyons Circle,28,1990-04-01 21:00:00.000     | Sum3      |          4029.0 |
| 4,Facelift Kitchen and Bath,220 Vine Ave.,15,1972-04-23 18:00:00.000       | Sum1      |          6097.0 |
| 4,Facelift Kitchen and Bath,220 Vine Ave.,15,1972-04-23 18:00:00.000       | Sum2      |        55560.24 |
| 4,Facelift Kitchen and Bath,220 Vine Ave.,15,1972-04-23 18:00:00.000       | Sum3      |         10862.0 |
'----------------------------------------------------------------------------+-----------+-----------------'
Job test2 ended at 14:29 30/07/2009. [exit code=0]
```

The console shows the results of the two **tLogRow** components. Table *tLogRow_1* gives an outline of the source file and table *tLogRow_2* shows the concatenation of the columns *id*, *CustomerName*, *CustomerAddress*, *id2* and *RegisterTime* as well as the transformation of the columns *Sum1*, *Sum2* and *Sum3* as pivot.

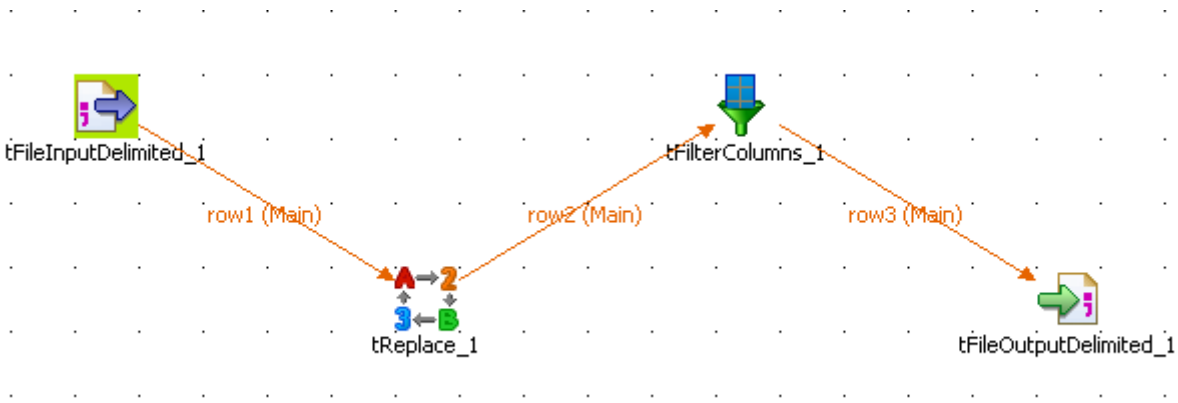
tReplace

tReplace Properties

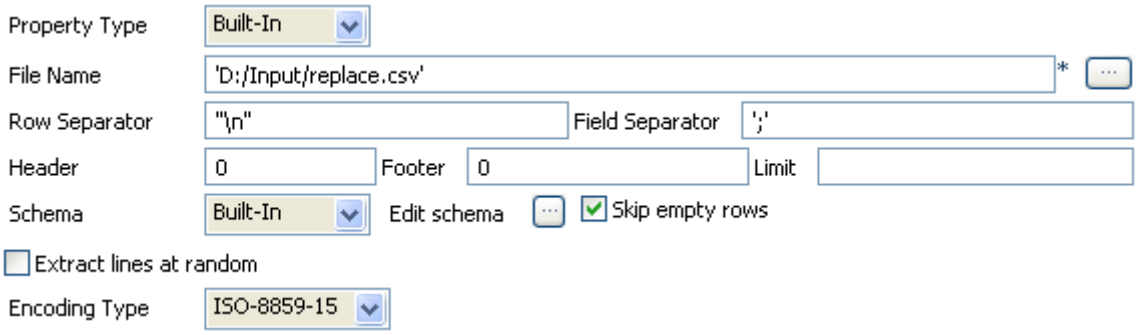
Component family	Processing	
Function	Carries out a Search & Replace operation in the input columns defined.	
Purpose	Helps to cleanse all files before further processing.	
Basic settings	*Schema type* and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the Repository. Two read-only columns, Value and Match are added to the output schema automatically.
		Built-in: The schema will be created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused in various projects and job designs. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	Simple Mode *Search / Replace*	Click Plus to add as many conditions as needed. The conditions are performed one after the other for each row. **Input column**: Select the column of the schema the search & replace is to be operated on **Search**: Type in the value to search in the input column **Replace with**: Type in the substitution value. **Whole word**: Select this check box if the searched value is to be considered as whole. **Case sensitive**: Select this check box to care about the case. Note that you cannot use regular expression in these columns.
	Use advanced mode	Select this check box when the operation you want to perform cannot be carried out through the simple mode. In the text field, type in the regular expression as required.
Usage	This component is not startable as it requires an input flow. And it requires an output component.	

Scenario: multiple replacements and column filtering

This following Job (made in Perl) searches and replaces various typos and defects in a csv file then operates a column filtering before producing a new csv file with the final output.



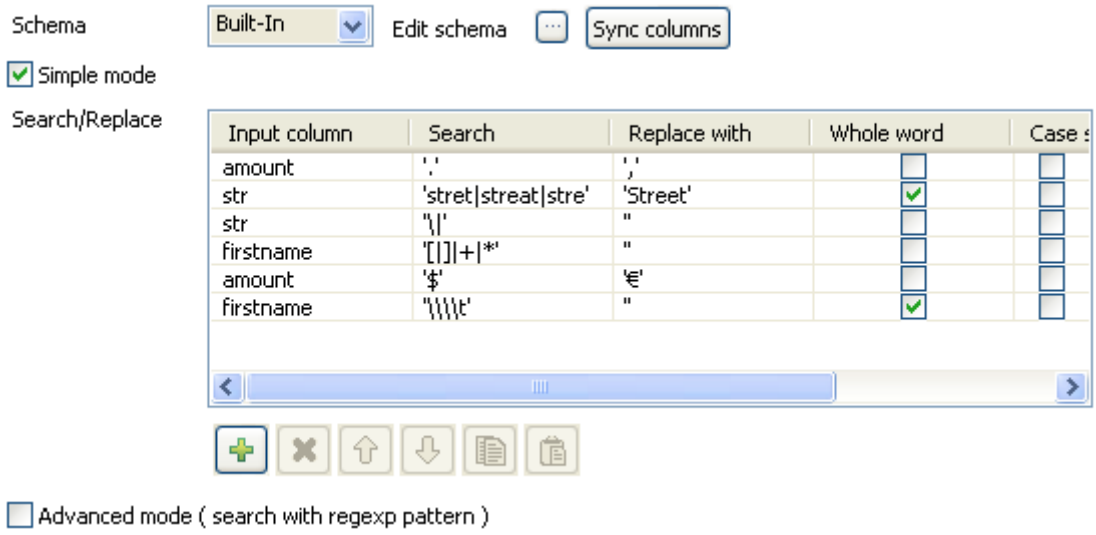
- Click & drop the following components from the **Palette**: **tFileInputDelimited**, **tReplace**, **tFilterColumn** and **tFileOutputDelimited**.
- Connect the components using **Main Row** connections via a right-click on each component.
- Select the **tFileInputDelimited** component and set the input flow parameters.



- The **Property type** for this scenario is **Built-in**. Therefore the following fields are to be set manually unlike the Properties stored centrally in the repository, that are retrieved automatically.
- The **File** is a simple csv file stored locally. The **Row Separator** is a carriage return and the **Field Separator** is a semi-colon. In this example no **Header**, no **Footer** and no **Limit** are to be set.
- The file contains characters such as: `\t, |||, [d]` or `*d` which should not be interpreted as special characters or wild card.

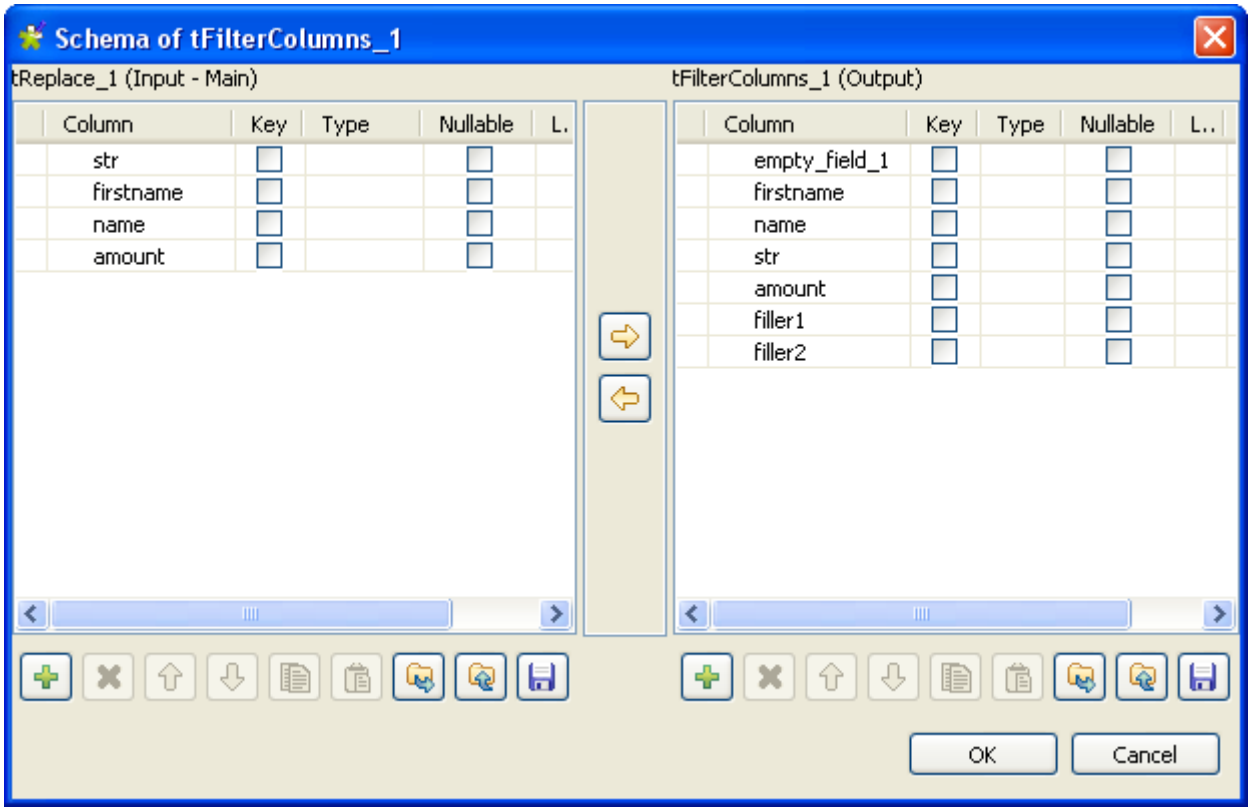
|||	Bill	Clinton	100
stre	John	Kennedy	98.30 $
Streat	Richar[d]	Nixon	78.23$
stret	Jimmy\t	Carter	38.54$
Streat	Richar+d	Nixon	78.2 $
Streat	toto	Nixon	78.23
Streat	Richar*d	Nixon	78.23$
street	Georges	bush	99.99$

- The schema for this file is built in also and made of four columns of various types (string or int).
- Now select the **tReplace** component to set the search & replace parameters.

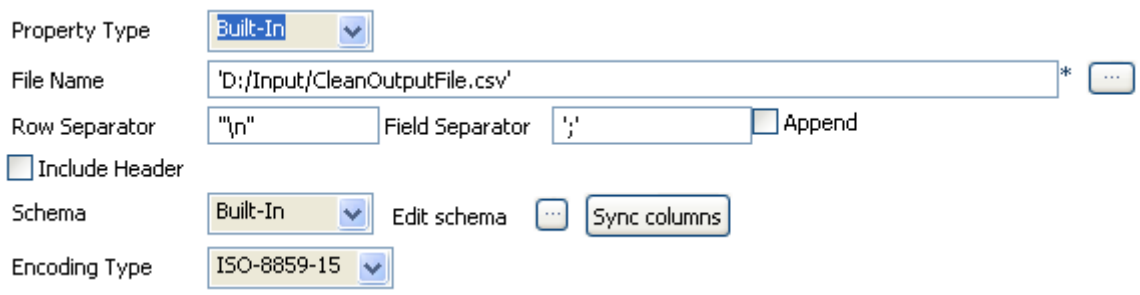


- The schema can be synchronized with the incoming flow.
- Select the **Simple mode** check box as the search parameters can be easily set without requiring the use of regexp.
- Click the plus sign to add some lines to the parameters table.
- On the first parameter line, select *amount* as **input column**. In the **search** field look for the decimal *dot* separator and **replace** it with a *comma*, in between single quotes.
- On the second parameter line, select *str* as **input column**. I**n the search field, look for** *stret* or *streat* or *stre*. Note that these values are separated by a pipe that means `or` in Perl language. Replace them by *Street*. Select the **whole word** check box.
- On the third parameter line, select again *str* as **input column**, **search** the pipe character using a backslash in front, to differentiate it from the “or” in Perl language. and replace it with nothing between single quotes (‘’).
- On the fourth parameter line, select *firstname* as **input column**. In the **Search** field, look for the following characters: `[, ], +, *`. Note that these values are separated by a pipe that means `or` in Perl language. Replace them with nothing between single quotes (‘’).

- On the fifth parameter line, select *amount* as **input column**. In the Search field, type in the dollar sign between single quotes and In the Replace field, type in the Euro sign.
- On the last parameter line, select *firstname* as **input column**. Search the string: `\t`. To differenciate it from the tabulation, add as many backslashes in front of it as there are parsing, in other words, two backlashes are used to avoid misinterpreting and two extra backslashes constitute part of the character being looked for. In total four backslahes including the one in the character it self are being searched. Replace them with nothing between single quotes (‘’). And select the **whole word** check box.
- The advanced mode isn’t used in this scenario.
- Select the next component in the Job, **tFilterColumn**.



- The **tFilterColumn** component holds a schema editor allowing to build the output schema based on the column names of the input schema. In this use case, change the order of the input schema columns and add 3 new columns, to obtain a schema as follows: *empty_field, firstname, name, str, amount, filler1, filler2*.
- Click **OK** to validate.



- Set the **tFileOutputDelimited** properties manually.
- The schema is built-in for this scenario, and comes from the preceding component in the Job.
- Save the Job and press **F6** to execute it.

	Bill	Clinton		100
	John	Kennedy	Street	98,30 €
	Richard	Nixon	Street	78,23 €
	Jimmy	Carter	Street	38,54 €
	Richard	Nixon	Street	78,20 €
	toto	Nixon	Street	78,23
	Richard	Nixon	Street	78,23 €
	Georges	bush	street	99,99 €

The first column is empty and the rest of the columns have been cleaned up from the parasitical characters. The street column was moved. And the decimal delimiter has been changed from a dot to a comma, along with the currency sign.

tSampleRow

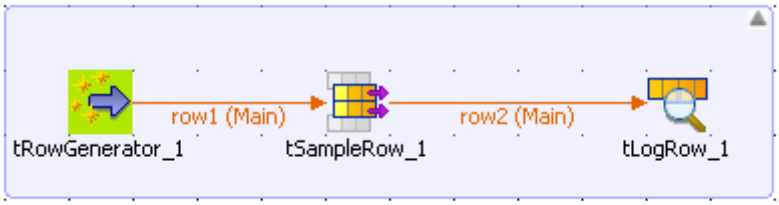
tSampleRow properties

Component family	Processing	
Function	**tSampleRow** filters rows according to line numbers.	
Purpose	**tSampleRow** helps to select rows according to a list of single lines and/or a list of groups of lines.	
Basic settings	*Schema type* and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the Repository. Click **Edit Schema** to make changes to the schema. Note that if you make changes, the schema automatically becomes built-in. Click **Sync columns** to retrieve the schema from the previous component in the Job.
		Built-in: You create the schema and store it locally for the relevant component. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: You have already created the schema and stored it in the Repository. You can reuse it in various projects and job flowcharts. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	Range	Enter a range using the relevant syntax to choose a list of single lines and/or a list of groups of lines.
Usage	This component handles flows of data therefore it requires input and output components.	
Limitation	n/a	

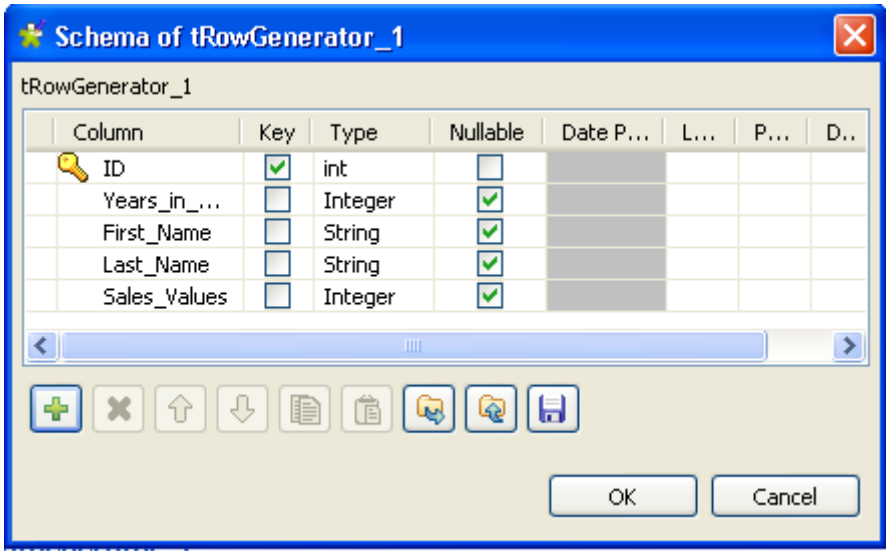
Scenario: Filtering rows and groups of rows

This Java scenario describes a three-component Job. A **tRowGenerator** is used to create random entries which are directly sent to a **tSampleRow** where they will be filtered according to a defined range. In this scenario, we suppose the input flow contains names of salespersons along with their respective number of sold products and their years of presence in the enterprise. The result of the filtering operation is displayed on the **Run** console.

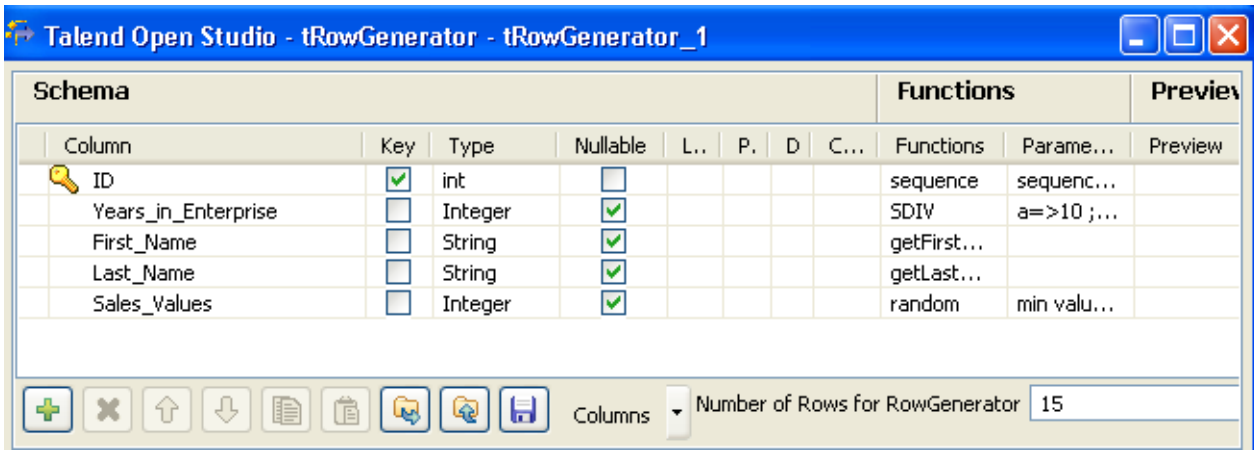
- Drop the following components from the **Palette** onto the design workspace: **tRowGenerator**, **tSampleRow**, and **tLogRow**.
- Connect the three components using **Row Main** links.



- In the design workspace, select **tRowgenerator**.
- Click the **Component** tab to define the basic settings for **tRowGenerator**.
- In the **Basic settings** view, set the **Schema Type** to **Built-In** and click the three-dot [...] button next to **Edit Schema** to define the data you want to use as input. In this scenario, the schema is made of five columns.

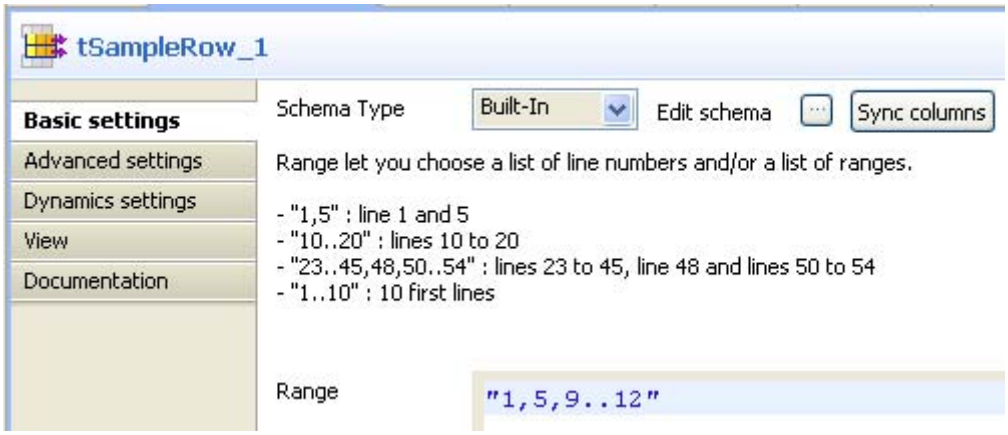


- In the **Basic settings** view, click **RowGenerator Editor** to define the data to be generated.
- In the **RowGenerator Editor**, specify the number of rows to be generated in the **Number of Rows for RowGenerator** field and click **OK**. The **RowGenerator Editor** closes.



- In the design workspace, select **tSampleRow**.

- Click the **Component** tab to define the basic settings for **tSamleRow**.



- n the **Basic settings** view, set the **Schema Type** to **Built-In** and click **Sync columns** to retrieve the schema from the **tRowGenerator** component.
- In the **Range** panel, set the filter to select your rows using the correct syntax as explained. In this scenario, we want to select the first and fifth lines along with the group of lines between 9 and 12.
- In the design workspace, select **tLogRow** and click the **Component** tab to define its basic settings. For more information about **tLogRow**, see *tLogRow on page 628*.
- Save your Job and press **F6** to execute it.

```
Starting job sample_Row at 11:16 20/08/2008.
.--+-------------------+----------+---------+-----------.
|                        tLogRow_1                        |
|=-+-------------------+----------+---------+-----------=|
|ID|Years_in_Enterprise|First_Name|Last_Name|Sale_Values|
|=-+-------------------+----------+---------+-----------=|
|1 |1                  |Martin    |Taft     |16         |
|5 |1                  |Franklin  |Adams    |45         |
|9 |1                  |William   |Madison  |15         |
|10|1                  |George    |Kennedy  |47         |
|11|1                  |George    |Van Buren|16         |
|12|1                  |Theodore  |Grant    |68         |
'--+-------------------+----------+---------+-----------'

Job sample_Row ended at 11:16 20/08/2008. [exit code=0]
```

The filtering result displayed on the console shows the first and fifth rows and the group of rows between 9 and 12.

tSortRow

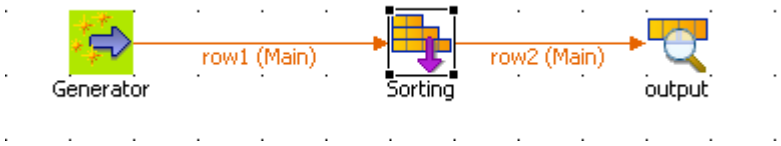
tSortRow properties

Component family	Processing	
Function	Sorts input data based on one or several columns, by sort type and order	
Purpose	Helps creating metrics and classification table.	
Basic settings	*Schema type* and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the Repository. Click **Edit Schema** to make changes to the schema. Note that if you make changes, the schema automatically becomes built-in. Click **Sync columns** to retrieve the schema from the previous component connected in the Job.
		Built-in: The schema will be created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused in various projects and job flowcharts. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	Criteria	Click + to add as many lines as required for the sort to be complete. By default the first column defined in your schema is selected.
		Schema column: Select the column label from your schema, which the sort will be based on. Note that the order is essential as it determines the sorting priority.
		Sort type: Numerical and Alphabetical order are proposed. More sorting types to come.
		Order: Ascending or descending order.
Advanced settings	*Sort on disk*	Customize the memory used to temporarily store output data. **Temp data directory path**: Set the location where the temporary files should be stored. **Buffer size of external sort**: Type in the size of physical memory you want to allocate to sort processing.
	tStatCatcher Statistics	Select this check box to gather the job processing metadata at the job level as well as at each component level.

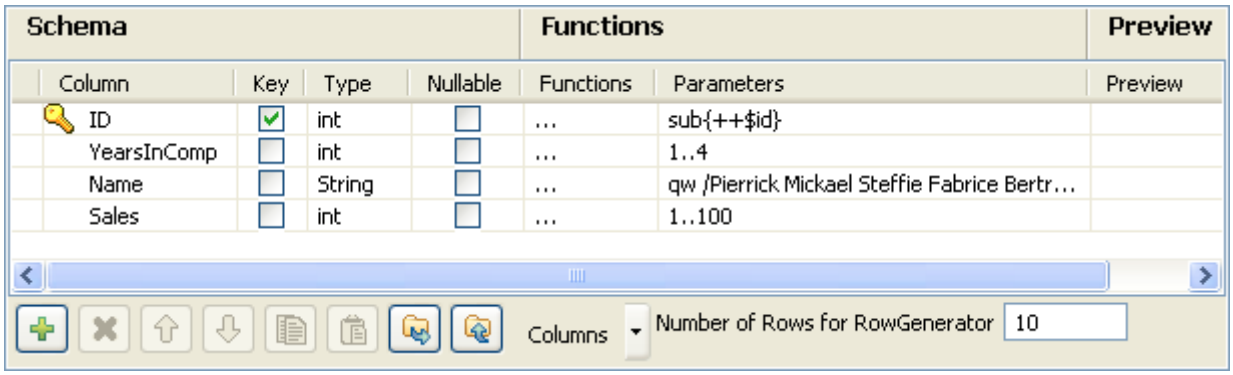
Usage	This component handles flow of data therefore it requires input and output, hence is defined as an intermediary step.
Limitation	n/a

Scenario: Sorting entries

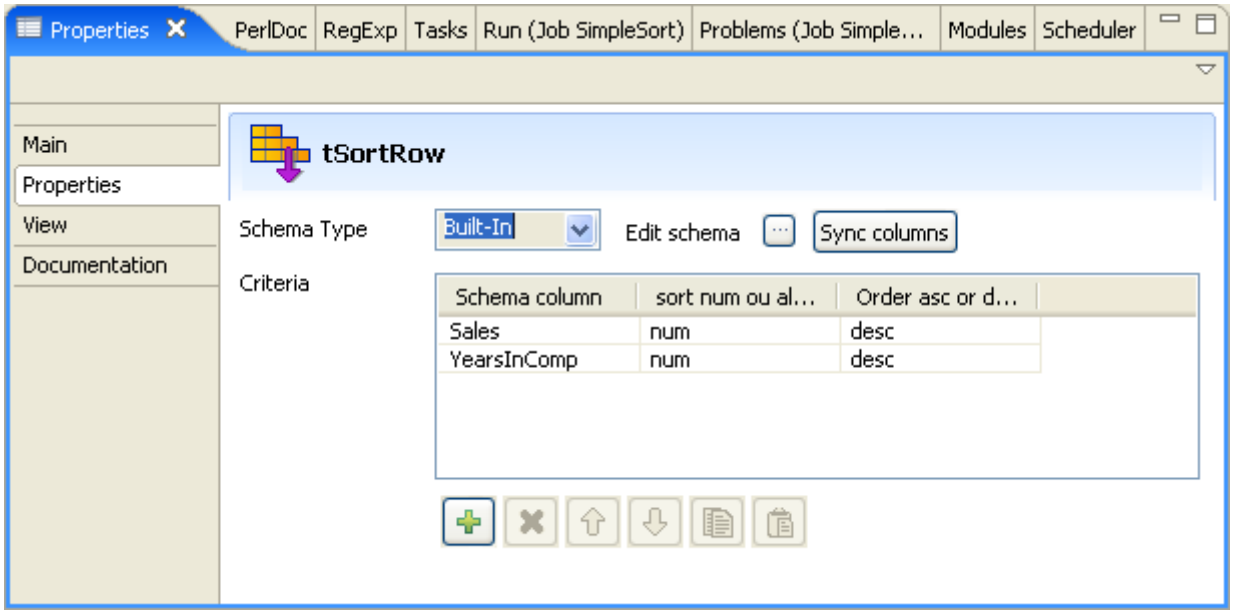
This scenario describes a three-component Job. A **tRowGenerator** is used to create random entries which are directly sent to a **tSortRow** to be ordered following a defined value entry. In this scenario, we suppose the input flow contains names of salespersons along with their respective sales and their years of presence in the company. The result of the sorting operation is displayed on the **Run** console.



- Drop the three components required for this use case: **tRowGenerator**, **tSortRow** and **tLogRow** from the **Palette** to the design workspace.
- Connect them together using **Row main** links.
- On the **tRowGenerator** editor, define the values to be randomly used in the Sort component. For more information regarding the use of this particular component, see *tRowGenerator on page 657*.



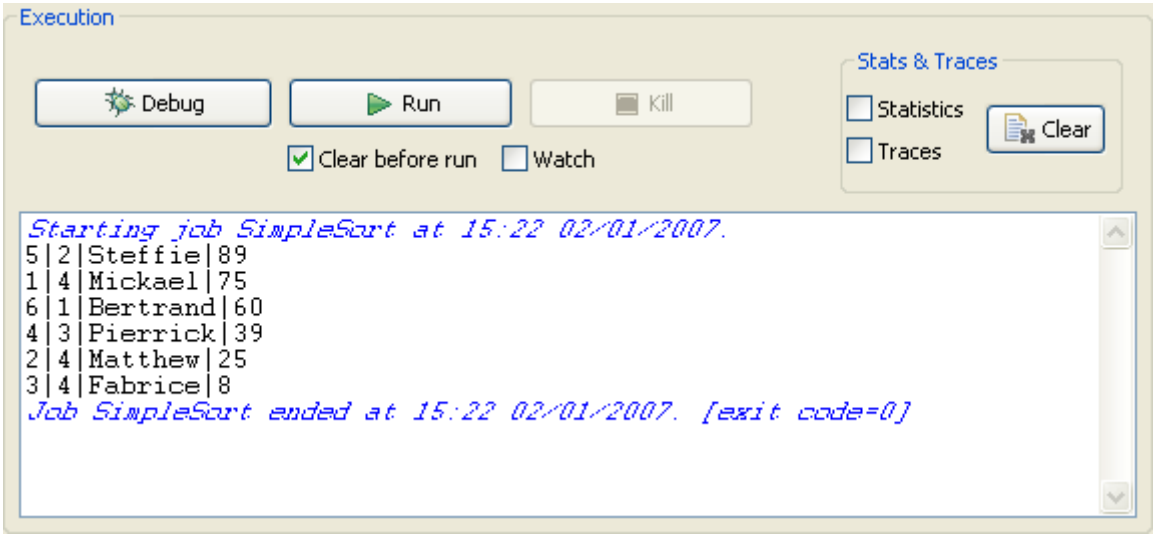
- In this scenario, we want to rank each salesperson according to its *Sales* value and to its number of years in the company.
- Double-click on **tSortRow** to display the **Basic settings** tab panel. Set the sort priority on the Sales value and as secondary criteria, set the number of years in the company.



- Use the plus button to add the number of rows required. Set the type of sorting, in this case, both criteria being integer, the sort is numerical. At last, given that the output wanted is a rank classification, set the order as descending.
- Display the **Advanced Settings** tab and select the **Sort on disk** check box to modify the temporary memory parameters. In the **Temp data directory path** field, type the path to the directory where you want to store the temporary data. In the **Buffer size of external sort** field, set the maximum buffer value you want to allocate to the processing.

The default buffer value is 1000000 but the more rows and/or columns you process, the higher the value needs to be to prevent the Job from automatically stopping. In that event, an "out of memory" error message displays.

- Make sure you connected this flow to the output component, **tLogRow**, to display the result in the job console.
- Press **F6** to run the Job or go to the **Run** panel and click **Run**.The ranking is based first on the Sales value and second on the number of years of experience.



System components

This chapter details the major components that you can find in **System** group of the **Palette** of **Talend Open Studio**.

The System family groups components that help you interact with the operating system.

tRunJob

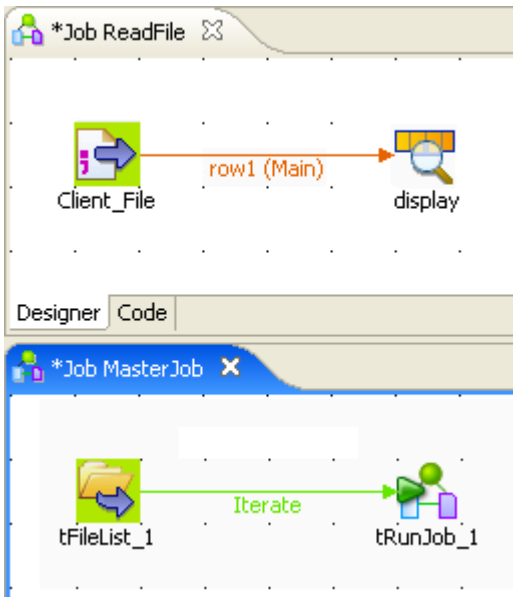
tRunJob Properties

Component family	System	
Function	**tRunJob** executes the Job called in the component's properties, in the frame of the context defined.	
Purpose	**tRunJob** helps mastering complex job systems which need to execute one Job after another.	
Basic settings	*Schema type* and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the Repository. Click **Edit Schema** to make changes to the schema. Note that if you make changes, the schema automatically becomes built-in.
		Built-in: You create and store the schema locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: You have already created the schema and stored it in the Repository. You can reuse it in various projects and job flowcharts. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide
	CopyChild Job Schema	Click to fetch the child job schema.
	Job	Select the Job to be called in and processed. Make sure you already executed once the Job called, beforehand, in order to ensure a smooth run through **tRunJob**.
	Version	Select the child Job version that you want to use.
	Context	If you defined contexts and variables for the Job to be run by the **tRunJob**, select the applicable context entry on the list.
	Context Param	You can change the selected context parameters. Click the plus button to add the parameters as defined in the Context of the child Job. For more information on context parameters, see *Context settings* in **Talend Open Studio** User Guide.
	Die on child error	Clear this check box to execute the parent Job even though there is an error when executing the child Job.
	Transmit whole context	Select this check box to transfer all the context variables from the child Job to the parent Job.

Usage	This component can be used as a standalone Job or can help clarifying complex Job by avoiding having too many sub-jobs all together in one Job.
Limitation	n/a

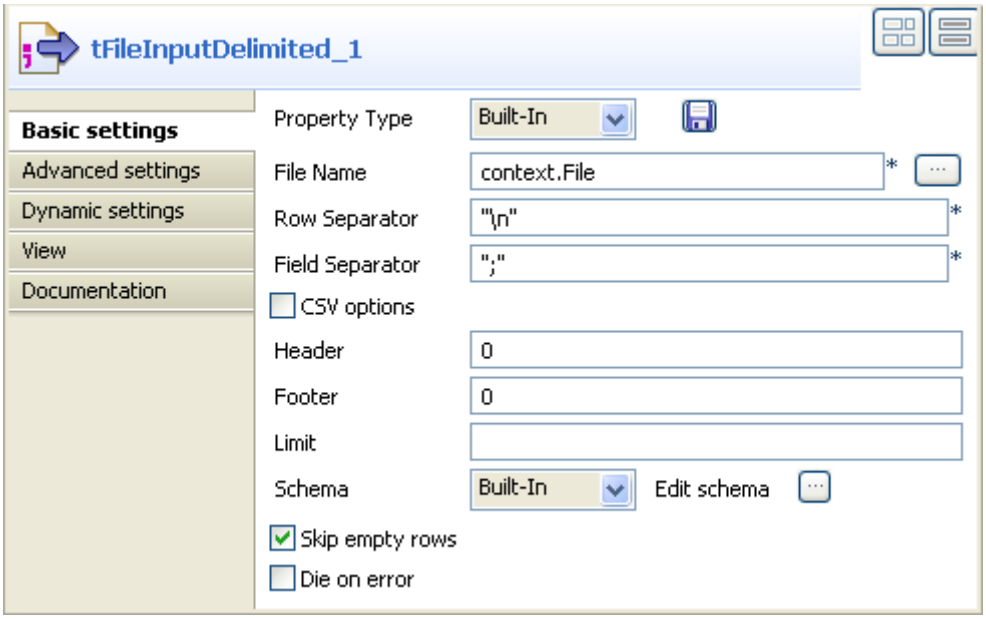
Scenario: Executing a child Job

This scenario describes a single-component Job calling in and executing another Job. The Job to be executed reads a basic delimited file and simply displays its content on the **Run** log console. The particularity of this Job lies in the fact that this latter Job is executed from a separate Job and uses a context variable to prompt for the input file to be processed.

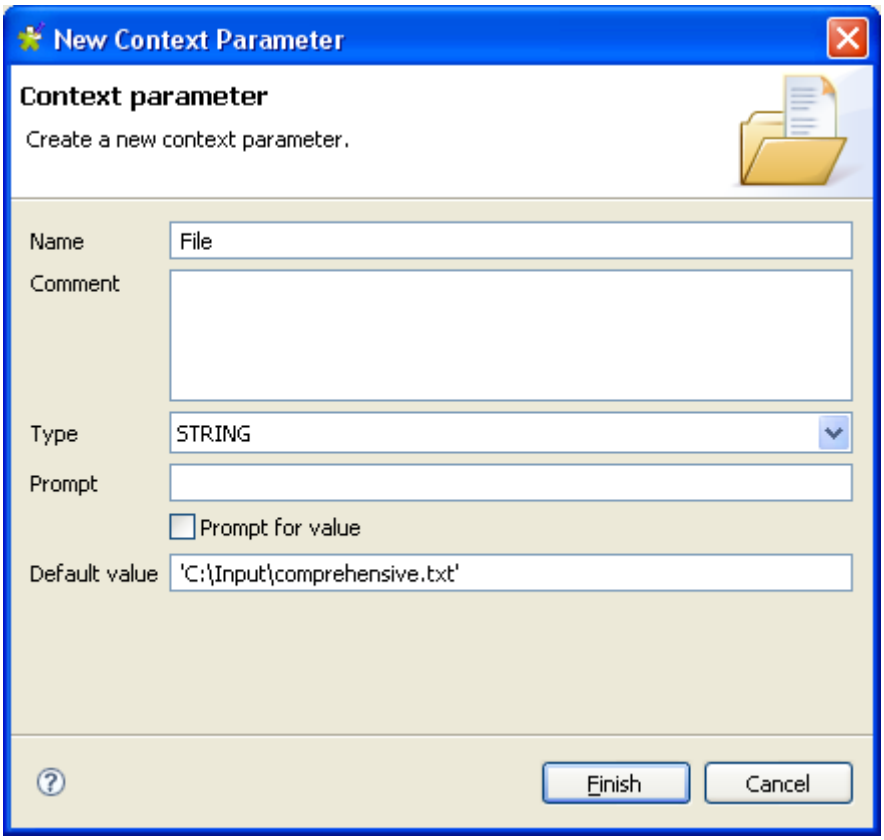


Create the first Job reading the delimited file.

- Drop a **tFileInputDelimited** and a **tLogRow** from the **Palette** to the design workspace.
- Connect the two components together using a **Row Main** link.
- Double-click **tFileInputDelimited** to open its **Basic settings** view and define its properties.



- Set the **Property type** to **Built-In** for this Job.
- Click in the **File Name** field and then press **F5** to open the **[New Context Parameter]** dialog box and configure the context variable.



- In the **Name** field, enter a name for this new context variable, *File* in this example.

In this example, there is no need to either select the **Prompt for value** check box or to set a prompt message, as the default parameter value can be used.

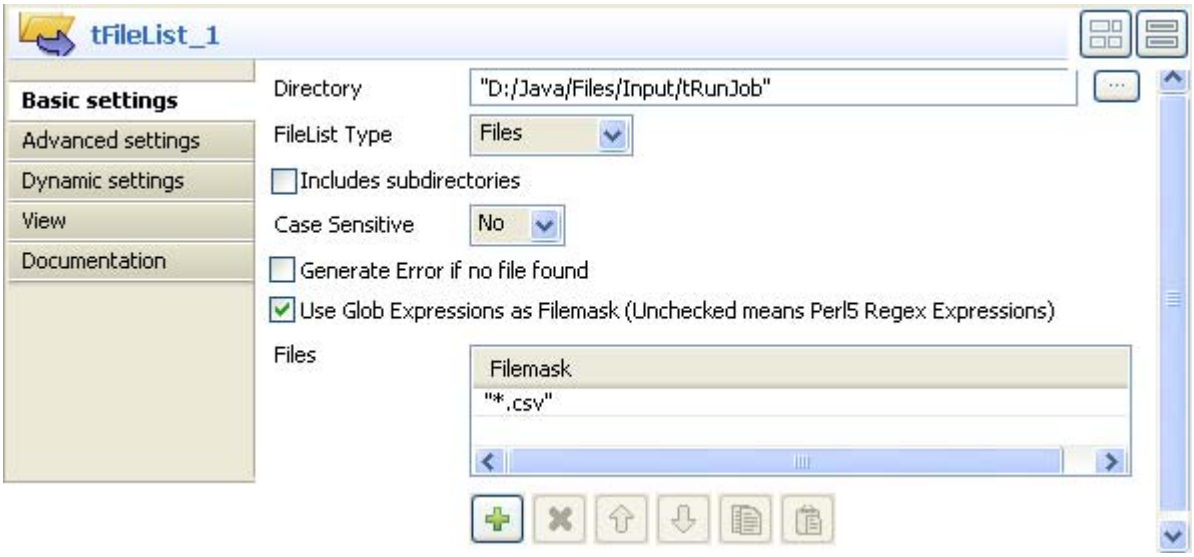
- Click **Finish** to validate the modification and press **Enter** on your keyboard to make sure the new context variable is stored the **File Name** field.
- In the **Basic settings** view, type in the field and row separators used in the input file.
- If needed, set **Header**, **Footer**, and **Limit**. In this example, no header or footer are used and no limit for the number of processed rows is set.
- Set **Schema type** to **Built-in** for this example. Click the three-dot button next to the field name to open the schema dialog box where you can configure the schema manually.
- In the dialog box, click the plus button to add two columns and name them following the first and second column names of your input file, *username* and *age* in this example.

If you store your schema in the **Repository** tree view, you only need to select the relevant metadata entry corresponding to your input file structure.

- Double-click **tLogRow** to display its **Basic settings** view and define its properties.
- Click **Sync columns** to retrieve the schema of the input component and then set other options according to your needs.
- save your Job and press **F6** to make sure that it executes without error.

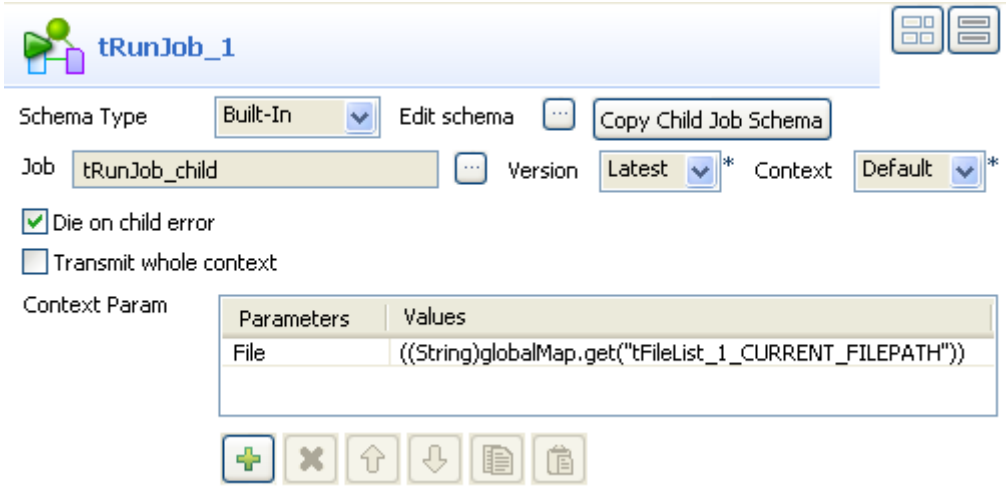
Create the second Job that will be the parent Job.

- Drop a **tFileList** and a **tRunJob** from the **Palette** to the design workspace.
- Connect the two components together using an **Iterate** link.
- Double-click **tFileList** to open its **Basic settings** view and define its properties.

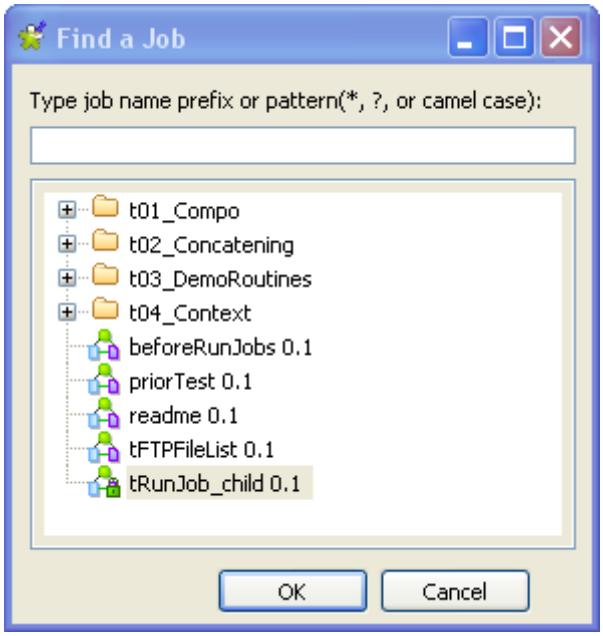


- In the **Directory** field, set the path to the directory that holds the files to be processed, or click the three-dot button next to the field to browse to the directory. In this example, the directory is called *tRunJob* and it holds three delimited files.
- In the **FileList Type** list, select **Files**.
- select the **Use Glob Expressions as Filemask** check box to be able to use regular expressions in your file masks.

- In the **Files** area, click the plus button to add a line where you can set the filter to apply. In this example, we want only to retrieve delimited files “**.csv*”.
- Double-click **tRunJob** to display its **Basic settings** view and define its properties.



- Click the three-dot button next to the **Job** field to open the **[Find a Job]** dialog box.



- Select the child Job you want to execute and click **OK** to close the dialog box. The name of the selected Job displays in the **Job** field in the **Basic settings** view of **tRunJob**.
- Click **Copy Child Job Schema** to retrieve the schema from the child Job.
- In the **Context Param** area, click the plus button to add a line and define the context parameter.
- Click in the **Values** cell and then press **Ctrl+Espace** on your keyboard to access the list of context variables.

- In the list, select *tFileList-1.CURRENT_FILEPATH*. The corresponding context variable displays in the **Values** cell:
  `((String)globalMap.get("tFileList-1.CURRENT_FILEPATH"))`.
  For more information on context variables, see *Context settings* in **Talend Open Studio** User Guide.
- save your Job and press **F6** to execute it.

```
Starting job tRunJob_father at 12:17 06/10/2009.

[statistics] connecting to socket on port 4043
[statistics] connected
clevenes|25
elsabot|32
mlelandais|26
hmassy|31
abibota|24
ychen|27
plegall|28
smallet|34
scorreia|37
[statistics] disconnected
Job tRunJob_father ended at 12:17 06/10/2009. [exit code=0]
```

The called-in Job reads the data contained in the input file, as defined by the input schema, and the result of this Job is displayed directly in the **Run** console.

Related topic: *tLoop on page 690*, and *Scenario1: Buffering data (Java) on page 640* of the **tBufferOutput** component.

tSetEnv

tSetEnv Properties

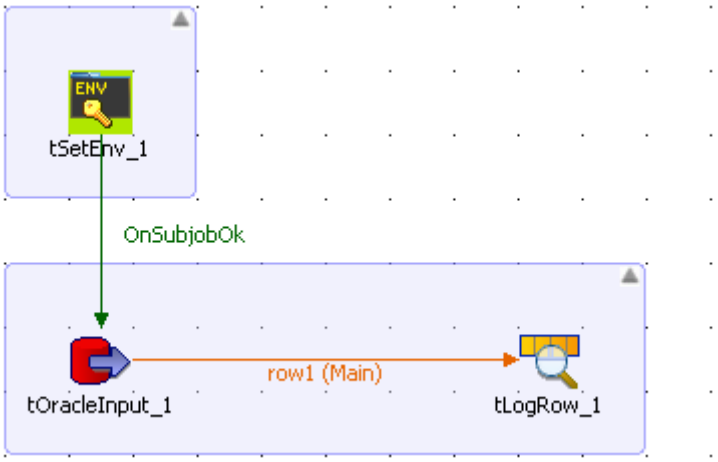
Component family	System	
Function	**tSetEnv** adds variables temporarily to system environment during the execution of a Job.	
Purpose	**tSetEnv** allows to create variables and execute a job script through communicating the information about the newly created variables between subjobs. After job execution, the newly created variables are deleted.	
Basic settings	*Parameters*	**Click the plus button to add the variables needed for the job.** **name**: Enter the syntax for the new variable. **value**: Enter a parameter value according to the context. **append**: Select this check box to add the new variable at the end.
Usage	**tSetEnv** can be used as a start or an intermediate component.	
Limitation	n/a	

Scenario: Modifying the *Date* variable during the execution of a Job

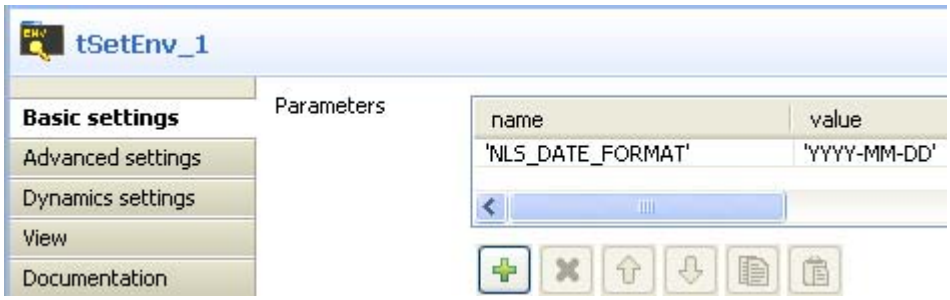
The following scenario is a Perl Job that reads a column in an Oracle DB, retrieves the current date from the column using a DB query, creates a new variable using **tSetEnv** to modify the date format, and outputs date, in the modified format, on the console.

To modify the date format using a new variable created by **tSetEnv**:

- Drop the following components from the **Palette** onto the design workspace: **tSetEnv**, **tOracleInput**, and **tLogRow**.
- Connect **tSetEnv** to **tOracleInput** using an **OnSubjobOk** link.
- Connect **tOracleInput** to **tLogRow** using a **Row Main** link.

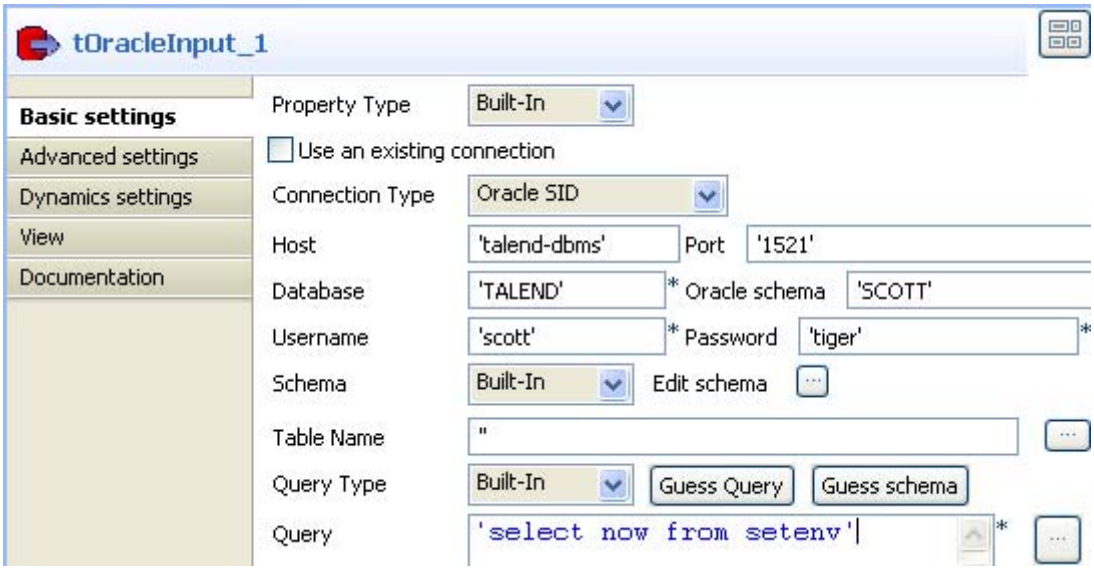


- Select **tSetEnv** and click the **Component** tab to display the component view.
- In the **Basic settings** panel, click the plus button to add a new parameter line and define your new variable.
- Click in the **name** cell and enter the syntax for your date variable. In this example, we use `NLS_DATE_FORMAT`.
- Click in the **value** cell and enter the desired value for your new date variable.



In this example, we want to modify the date format `DD-MMM-YY` predefined in the system to display as `YYYY-MM-DD`.

- Select **tOracleInput** and click the **Component** tab to display the component view.
- Set the **Basic settings** for the **tOracleInput** component. For more information, see *tOracleInput on page 336*.



In this example, we query an Oracle database to extract the data held in the “now” column from the “setenv” table.

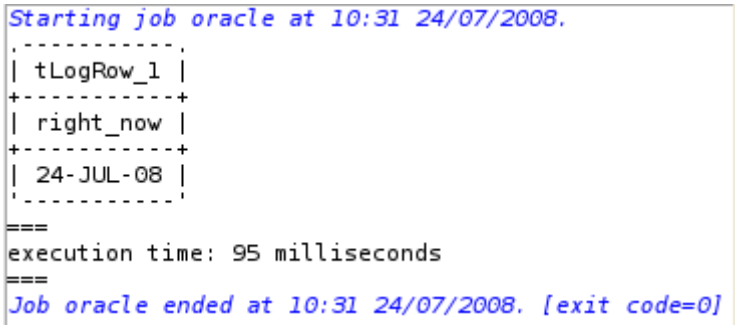
- Select **tLogRow** and click the **Component** tab to display the component view.
- Set the **Basic settings** for the **tLogRow** component as needed. For more information, see *tLogRow on page 628*.
- Save your Job and press **F6** to execute it.

```
Starting job oracle at 10:30 24/07/2008.
.------------.
|  tLogRow_1 |
+------------+
| right_now  |
+------------+
| 2008-07-24 |
'------------'
===
execution time: 85 milliseconds
===
Job oracle ended at 10:30 24/07/2008. [exit code=0]
```

The date displays on the console in the format `YYYY-MM-DD` set with the **tSetEnv** component.

To display the date in the above Job in the format pre-defined in the system:

- In the design workspace, right-click the **tSetEnv** component and select **Deactivate tSetEnv_1** from the drop-down list.
- Press **F6** to execute your Job.



The date displays on the console in the format `DD-MMM-YY` pre-defined in the system.

tSSH

tSSH Properties

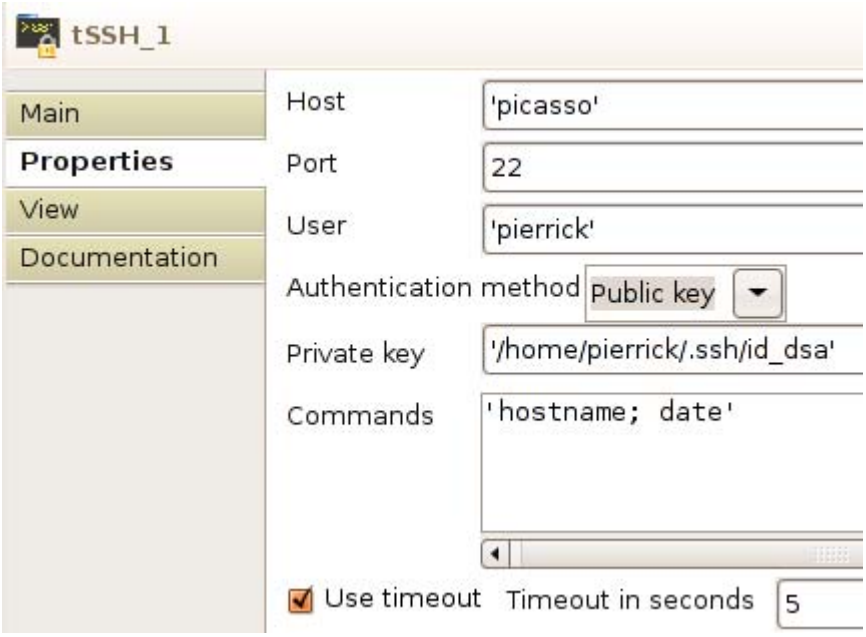
Component family	System	
Function	Returns data from a remote computer, based on the secure shell command defined.	
Purpose	Allows to establish a communication with distant server and return securely sensible information.	
Basic settings	*Host*	IP address
	Port	Listening port number
	User	User authentication information
	Public Key/Password	Select the relevant option. In case of Public Key, make sure the key is added to the agent or that no passphrase is required.
	Password/Private Key	**Password**: Type in the password required. **Private key**: browse to the relevant key location.
	Commands	Type in the command for the relevant information to be returned from the remote computer.
	Use timeout/timeout in seconds	Define the timeout time period. A timeout message will be generated if the actual response time exceeds this expected processing time.
Usage	This component can be used as standalone component.	
Limitation	The component use is optimized for Unix-like systems.	

Scenario: Remote system information display via SSH

The following use case describes a basic Job that uses SSH command to display the hostname of the distant server being connected to, and the current date on this remote system.

The **tSSH** component is sufficient for this Job. Drop it from the **Palette** to the desihn workspace.

Double-click on the **tSSH** component and select the **Basic settings** view tab.



- Type in the name of the **Host** to be accessed through SSH as well as the **Port** number.
- Fill in the **User** identification name on the remote machine.
- Select the **Authentication method** on the list. For this use case, the authentication method used is the public key.
- Thus fill in the corresponding **Private key**.
- On the Command field, type in the following command. For this use case, type in `hostname; date` between single quotes (as the Job is generated in Perl.).
- Select the **Use timeout** check box and set the time before falling in error to 5 seconds.

```
Starting job uniteElisa at 16:26 26/09/2007.
picasso
Wed Sep 26 14:24:15 CEST 2007
Job uniteElisa ended at 16:26 26/09/2007. [exit code=0]
```

The remote machine returns the host name and the current date and time as defined on its system.

tSystem

tSystem Properties

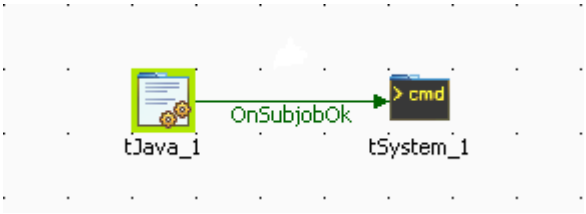
Component family	System	
Function	**tSystem** executes one or more system commands.	
Purpose	**tSystem** can call other processing commands, already up and running in a larger Job.	
Basic settings	*Use home directory*	Select this check box to change the name and path of a dedicated directory.
	Command	Enter the system command. Note that the syntax is not checked.
	Standard Output and *Error Output*	Select the type of output for the processed data to be transferred to.
		to console: data is passed on to be viewed in the **Run** view.
		to global variable: data is passed on to an output variable linked to the **tSystem** component.
		to console and to global variable: data is passed on to the **Run** view and to an output variable linked to the **tSystem** component.
		normal: data is passed on to the component that comes next.
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the Repository. Click **Edit Schema** to make changes to the schema. Note that if you make changes, the schema automatically becomes built-in. Click **Sync columns** to retrieve the schema from the preceding component in the Job.
		Built-in: You create and store the schema locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: You have already created the schema and stored it in the Repository. You can reuse it in various projects and job flowcharts. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide
	Parameters	Click the plus button to add as many global variables as needed. **name**: Enter the syntax of the new variable. **value**: Enter a value for this variable according to the context.

Usage	This component can typically used for companies which already implemented other applications that they want to integrate into their processing flow through Talend.
Limitation	n/a

Scenario: Echo 'Hello World!'

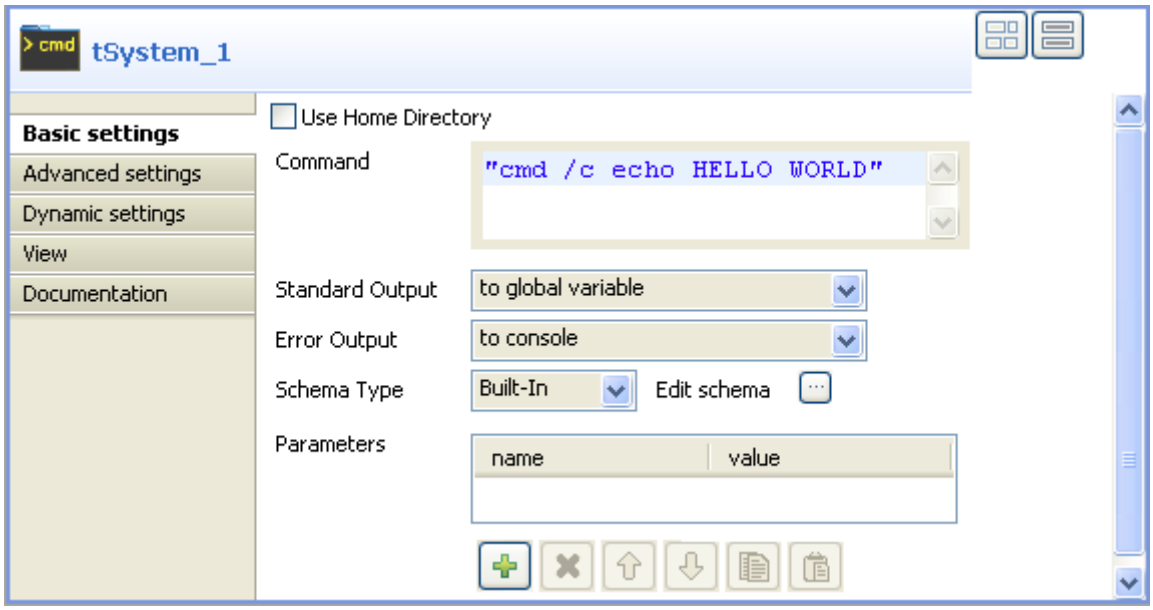
This Java scenario is a two-component Job that executes a system command and shows the results in the **Run** view "console".

- Drop a **tJava** and a **tSystem** components from the **Palette** to the design workspace.
- Connect the two components together using a **Trigger** > **OnSubjobOk** link between the two components.

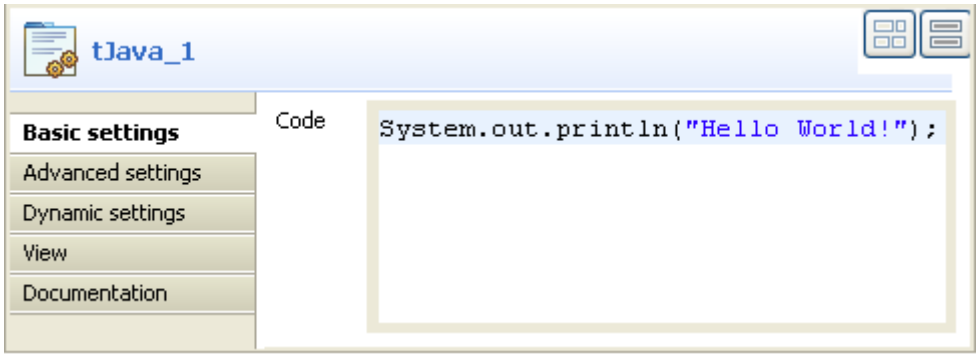


When executing the Job, the first component triggers the second one.

- Double-click **tSystem** to open the **Basic settings** view and display the component properties.



- In the **Command** field, enter the `echo` command followed by the string to display, "Hello World!" in this example.
- In the **Standard Output** field, select **to a global variable** to send the output to a global variable.
- Keep the by-default parameters in the other fields.
- Double-click **tJava** to open the **Basic settings** view and define the component properties.



- Enter the following Java command to display the **tSystem** output variable in the console: `System.out.println("Hello World!");`
- Save your Job and press **F6** to execute it.

```
Starting job tSystem_scenario at 17:26 08/10/2009.

[statistics] connecting to socket on port 3961
[statistics] connected
Hello World!
[statistics] disconnected
Job tSystem_scenario ended at 17:26 08/10/2009. [exit code=0]
```

The Job executes an echo command and shows the output in the Console of the **Run** view using a `Println` command in the **tJava** component.

XML components

This chapter details the major components that you can find in **XML** group of the **Palette** of **Talend Open Studio**.

The XML family groups the components dedicated to XML related tasks such as parsing, validation, XML structure creation and so on.

tAdvancedFileOutputXML

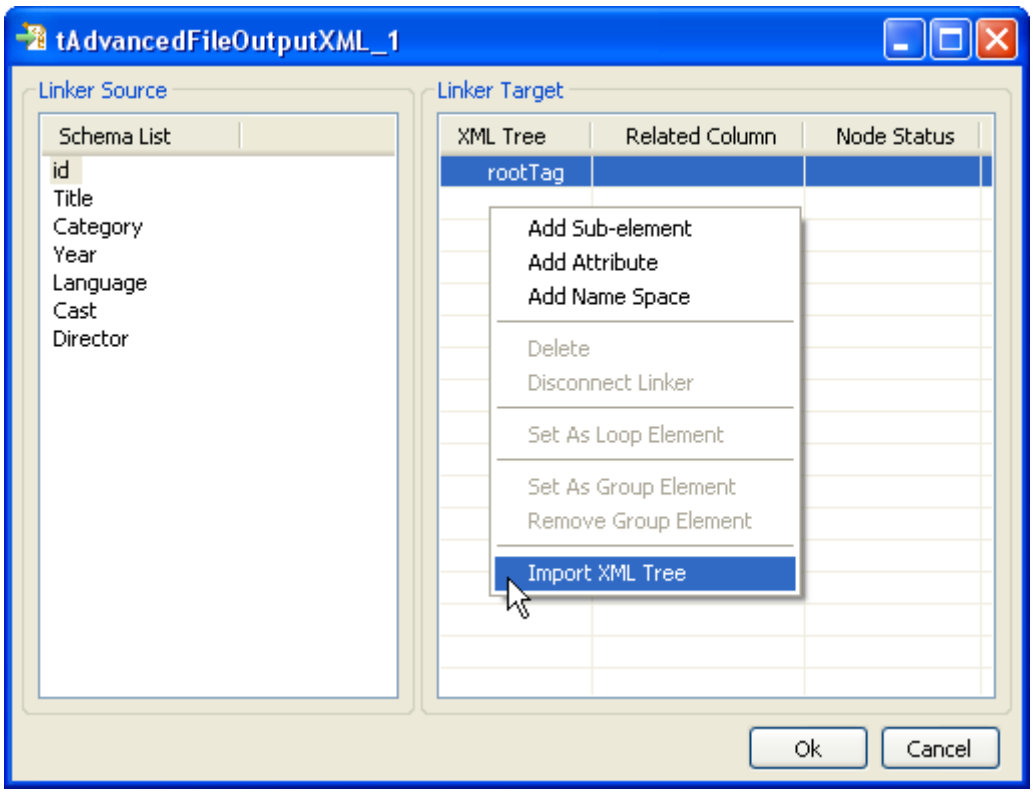
tAdvancedFileOutputXML properties

Component family	XML or File/Output	
Function	**tAdvancedFileOutputXML** outputs data to an XML type of file and offers an interface to deal with loop and group by if needed.	
Purpose	**tAdvancedFileOutputXML** writes an XML file with separated data value according to a XML tree schema.	
Basic settings	*File name*	Name or path to the output file. Related topic: *Defining variables from the Component view* of **Talend Open Studio** User Guide
	Configure XML tree	Opens the dedicated interface to help you set the XML mapping. For details about the interface, see *Defining the XML tree on page 809*.
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the Repository.
		Built-in: The schema will be created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused in various projects and job designs. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	Sync columns	Click to synchronize the output file schema with the input file schema. The Sync function only displays once the Row connection is linked with the Output component.
	Append the source xml file	Select this check box to add the new lines at the end of your source XML file.
	Generate compact file	Select this check box to generate a file that does not have any empty space or line separators. All elements then are presented in a unique line and this will reduce considerably file size.
Advanced settings	*Split output in several files*	If the XML file output is big, you can split the file every certain number of rows.
	Create directory only if not exists	This check box is selected by default. It creates a directory to hold the output XML files if required.
	Create empty element if needed	This box is selected by default. If the **Associated Column** content is null or if no column is associated to an XML node, this option will create an open/close tag in place of the expected tag.

	Create associated XSD file	If one of the XML elements is defined as a Namespace element, this option will create the corresponding XSD file. To use this option, you are required to select **Dom4J** as generation mode.
	Advanced separator	Select this check box to change the expected data separator.
	Generation mode	Select the faster or slower generation mode according to the memory usage requirements you have.
	Encoding type	Select the encoding from the list or select Custom and define it manually. This field is compulsory for DB data handling.
Usage	Use this component to write an XML file with data passed on from other components using a Row link.	
Limitation	n/a	

Defining the XML tree

Double-click on the **tAdvancedFileOutputXML** component to open the dedicated interface or click on the three-dot button on the **Basic settings** vertical tab of the **Component Settings** tab.



To the left of the mapping interface, under **Schema List**, are listed all columns retrieved from the incoming data flow (on the condition that an input flow is connected to the **tAdvancedFileOutputXML** component).

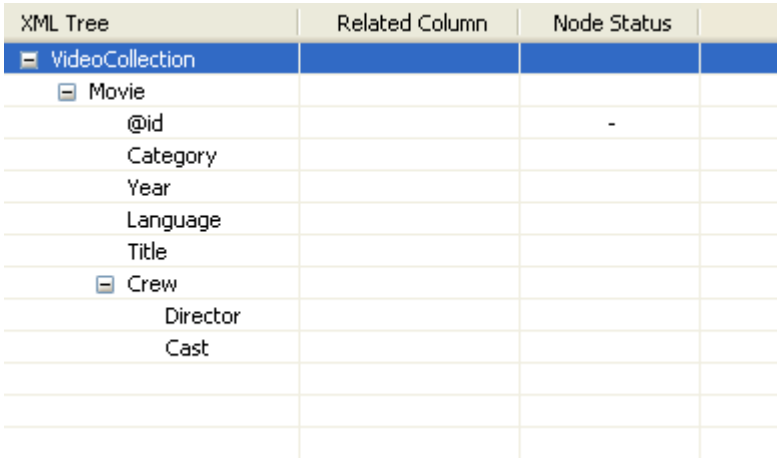
To the right of the interface, is expected the XML structure you want to obtain as output.

You can create manually or easily import the XML structure. Then map the input schema columns onto each element of the XML tree.

Importing the XML tree

The easiest and most common way to fill out the XML tree panel, is to import a well-formed XML file.

- Rename the **root tag** that displays by default on the **XML tree** panel, by clicking on it once.
- Right-click on the root tag to display the contextual menu.
- On the menu, select **Import XML tree**.
- Browse to the file to import and click **OK**.



The **XML Tree** column is hence automatically filled out with the correct elements. You can remove and insert elements or sub-elements from and to the tree:

- Select the relevant element of the tree.
- Right-click to display the contextual menu
- Select Delete to remove the selection from the tree or select the relevant option among: **Add sub-element**, **Add attribute**, **Add namespace** to enrich the tree.

Creating manually the XML tree

If you don't have any XML structure already defined, you can manually create it.

- Rename the **root tag** that displays by default on the **XML tree** panel, by clicking on it once.
- Right-click on the root tag to display the contextual menu.
- On the menu, select Add sub-element to create the first element of the structure.

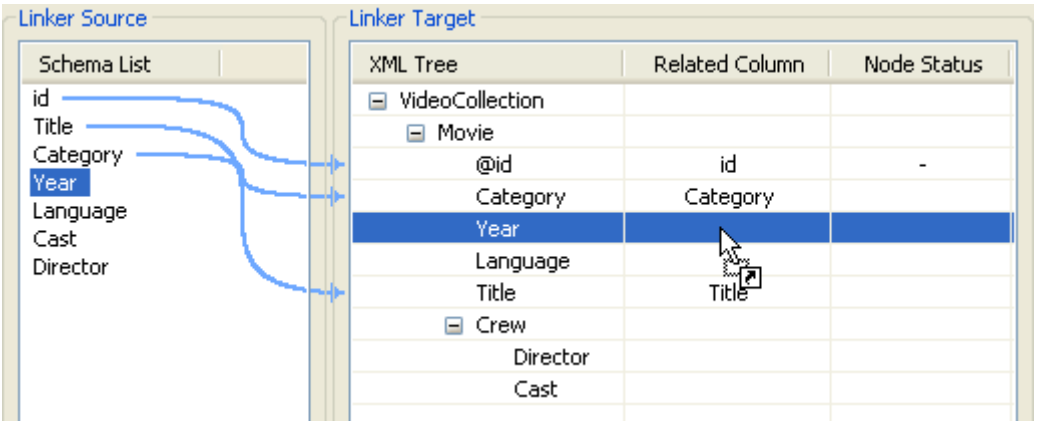
You can also add an attribute or a child element to any element of the tree or remove any element from the tree.

- Select the relevant element on the tree you just created.
- Right-click to the left of the element name to display the contextual menu.
- On the menu, select the relevant option among: **Add sub-element**, **Add attribute**, **Add namespace** or **Delete**.

Mapping XML data

Once your XML tree is ready, you can map each input column with the relevant XML tree element or sub-element to fill out the **Related Column**:

- Click on one of the **Schema column name**.
- Drag it onto the relevant sub-element to the right.
- Release to implement the actual mapping.



A light blue link displays that illustrates this mapping. If available, use the **Auto-Map** button, located to the bottom left of the interface, to carry out this operation automatically.

You can disconnect any mapping on any element of the XML tree:

- Select the element of the XML tree, that should be disconnected from its respective schema column.
- Right-click to the left of the element name to display the contextual menu.
- Select **Disconnect linker**.

The light blue link disappears.

Defining the node status

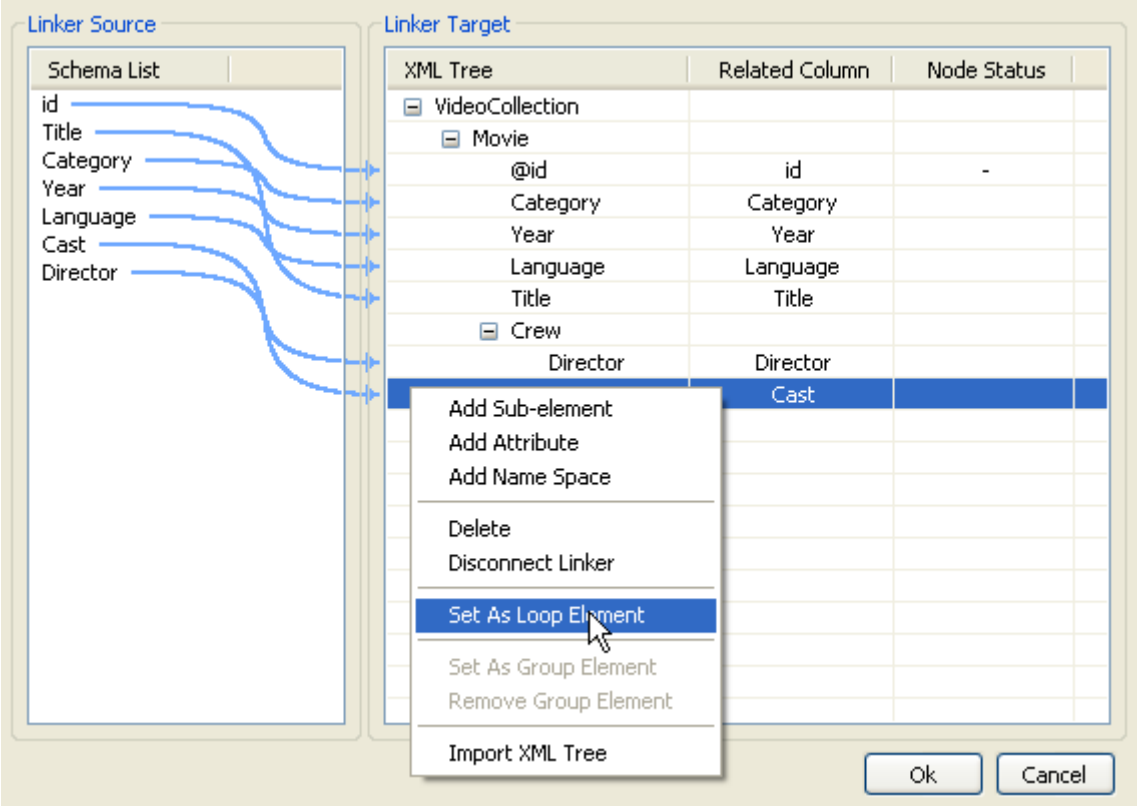
Defining the XML tree and mapping the data is not sufficient. You also need to define the loop element and if required the group element.

Loop element

The loop element allows you to define the iterating object. Generally the Loop element is also the row generator.

To define an element as loop element:

- Select the relevant element on the XML tree.
- Right-click to the left of the element name to display the contextual menu.
- Select **Set as Loop Element**.



The **Node Status** column shows the newly added status.

There can only be one loop element at a time.

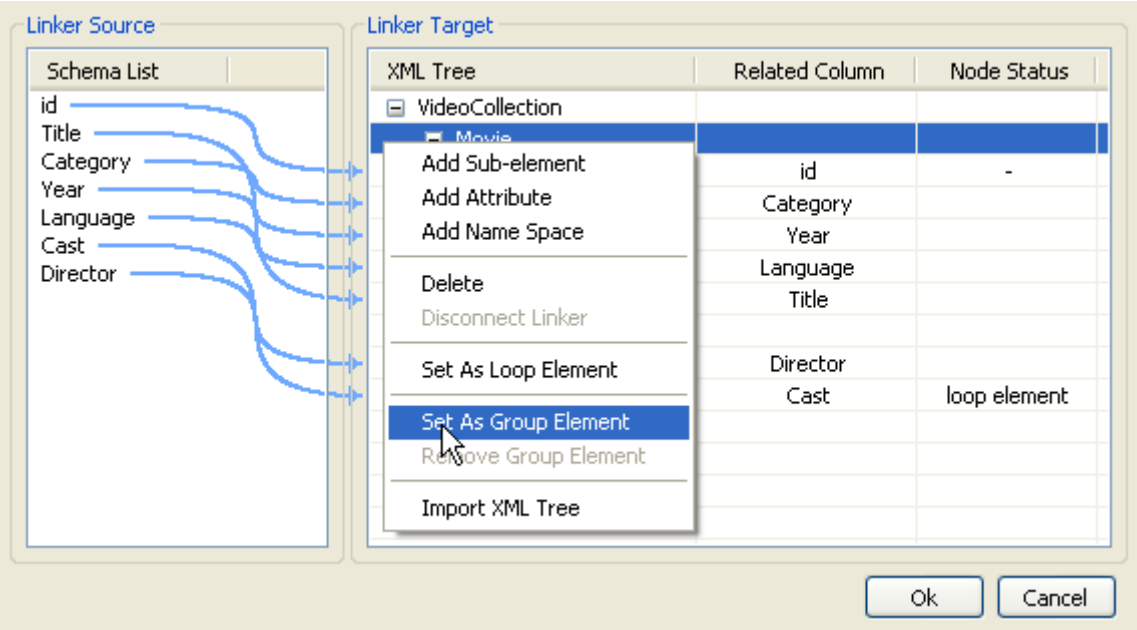
Group element

The group element is optional, it represents a constant element where the groupby operation can be performed. A group element can be defined on the condition that a loop element was defined before.

When using a group element, the rows should sorted, in order to be able to group by the selected node.

To define an element as group element:

- Select the relevant element on the XML tree.
- Right-click to the left of the element name to display the contextual menu.
- Select **Set as Group Element**.



The **Node Status** column shows the newly added status and any group status required are automatically defined, if needed.

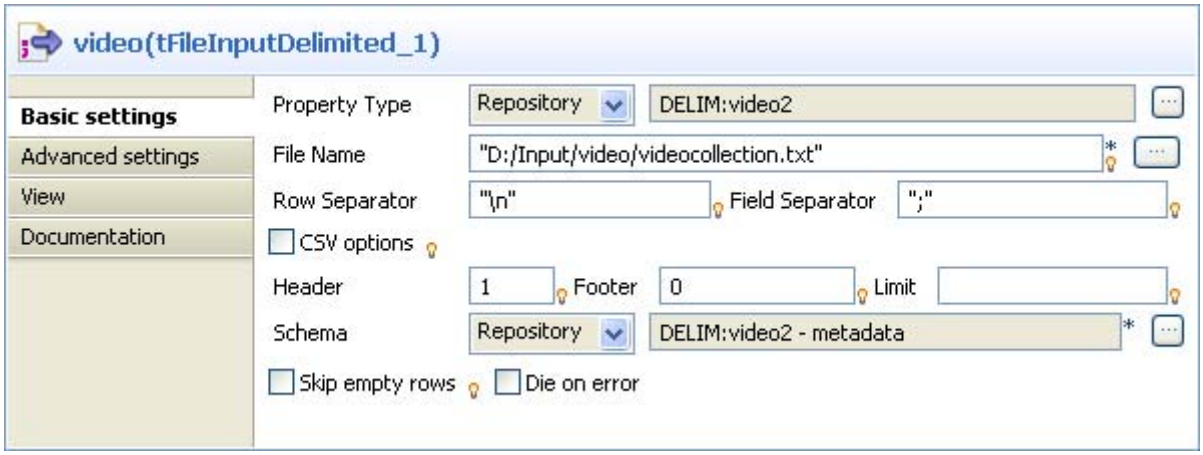
Click **OK** once the mapping is complete to validate the definition and continue the job configuration where needed.

Scenario: Creating an XML file using a loop

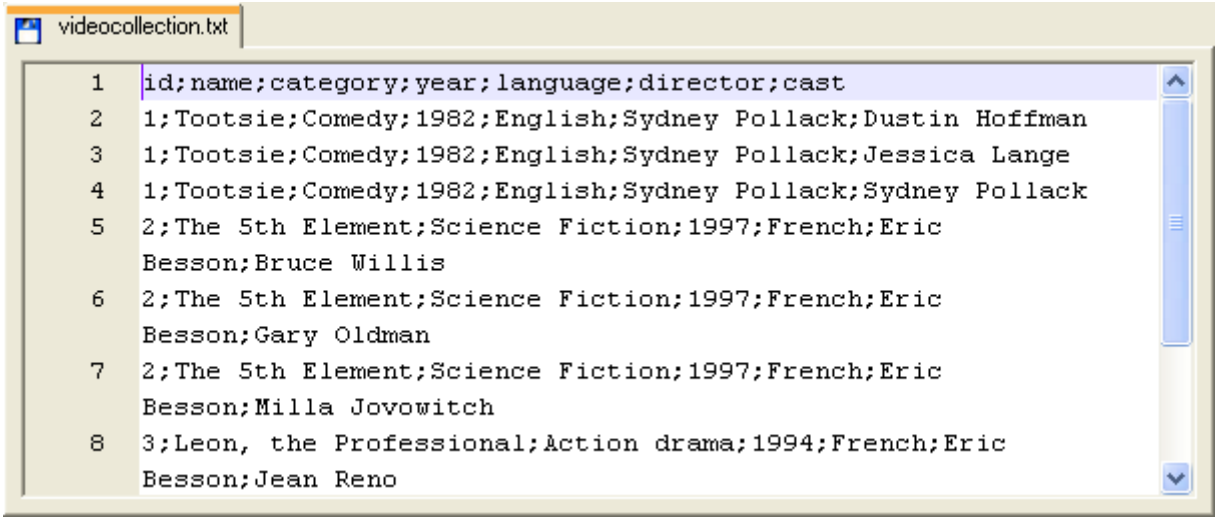
The following scenario describes the creation of an XML file from a sorted flat file gathering a video collection.

- Drop a **tFileInputDelimited** and a **tAdvancedFileOutputXML** from the **Palette** to the design workspace.
- Alternatively, if you configured a description for the input delimited file in the **Metadata** area of the **Repository**, then you can directly drag & drop the metadata entry onto the editor, to set up automatically the input flow.
- Right-click on the input component and drag a row main link towards the **tAdvancedFileOutputXML** component to implement a connection.

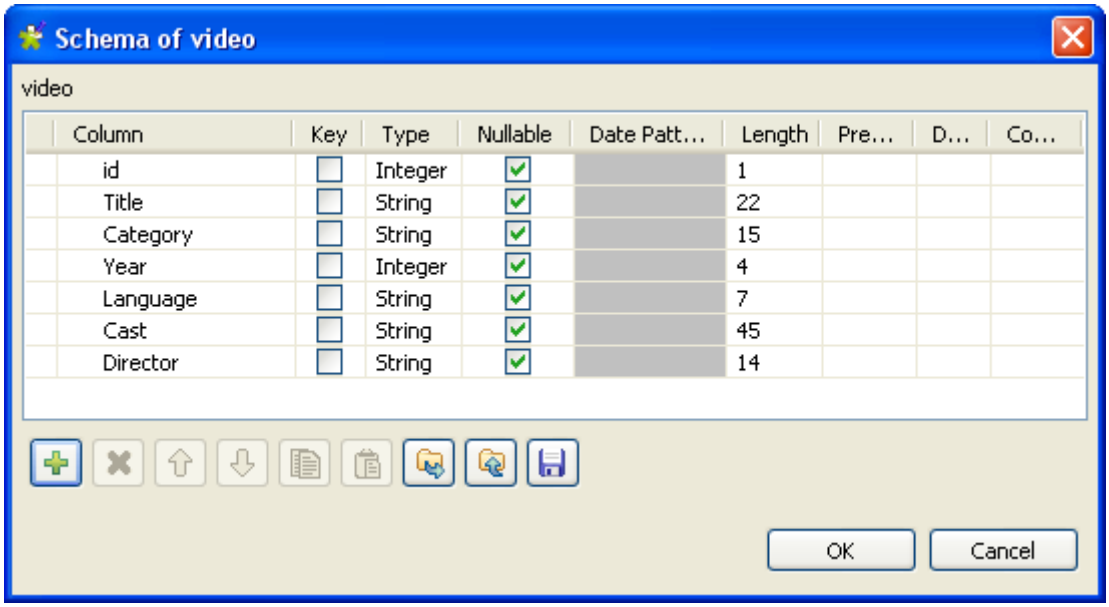
- Select the **tFileInputDelimited** component and display the **Component settings** tab located in the tab system at the bottom of the Studio.



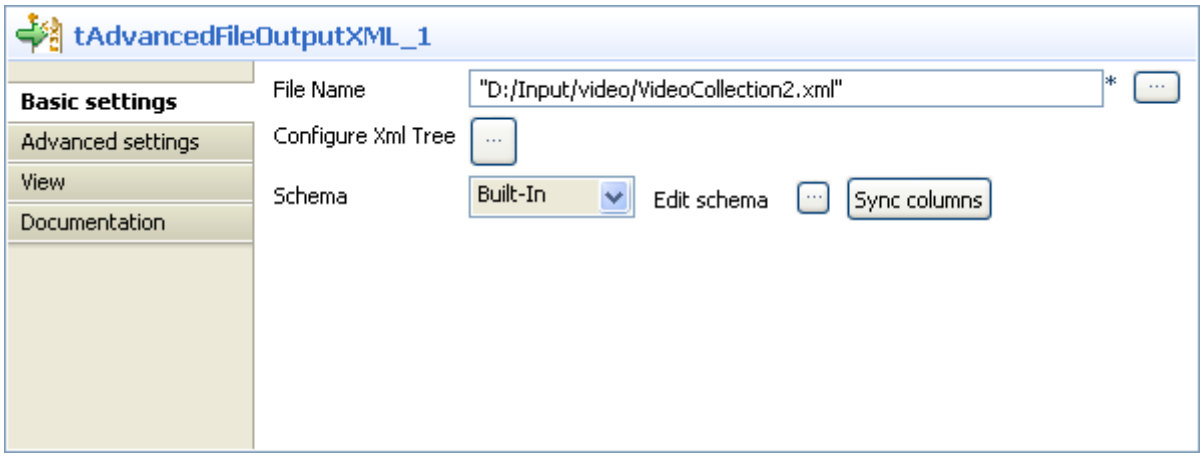
- Select the **Property type**, according to whether you stored the file description in the Repository or not. If you dragged & dropped the component directly from the Metadata, no changes to the setting should be needed.
- If you didn't setup the file description in the **Repository**, then select **Built-in** and manually fill out the fields displayed on the **Basic settings** vertical tab.
- The input file contains the following type of columns separated by semi-colons: *id*, *name*, *category*, *year*, *language*, *director* and *cast*.



- In this simple use case, the **Cast** field gathers different values and the id increments when changing movie.
- If needed, define the **tFileDelimitedInput** schema according to the file structure.

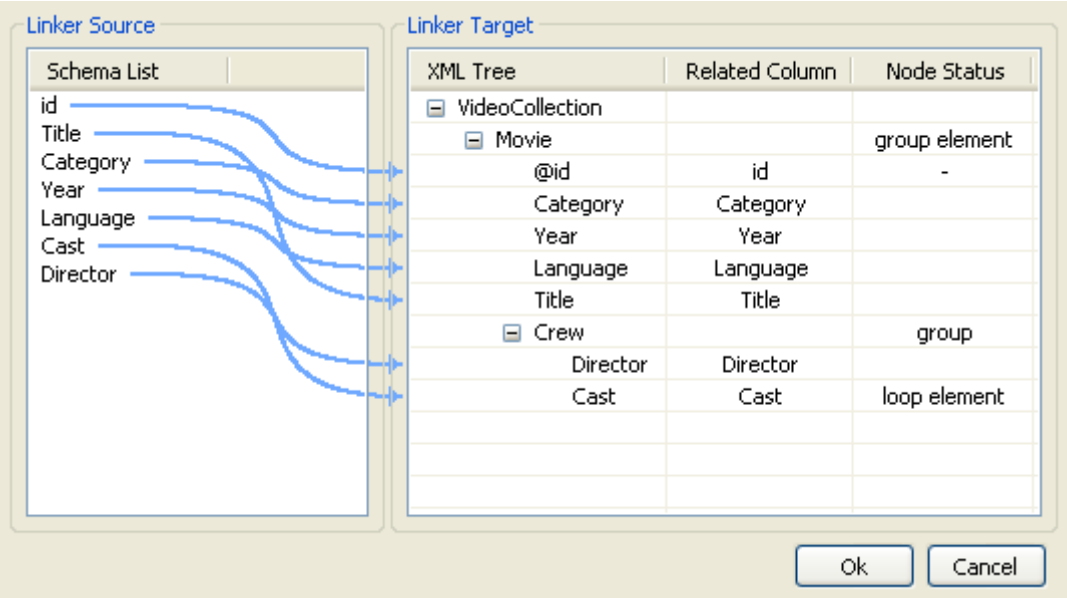


- Once you checked that the schema of the input file meets your expectation, click on **OK** to validate.
- Then select the **tAdvancedFileOutputXML** component and click on the **Component settings** tab to configure the basic settings as well as the mapping. Note that a double-click on the component will open directly the mapping interface.



- In the **File Name** field, browse to the file to be written if it exists or type in the path and file name that needs to be created for the output.
- By default, the schema (file description) is automatically propagated from the input flow. But you can edit it if you need.
- Then click on the three-dot button or double-click on the **tAdvancedFileOutputXML** component on the design workspace to open the dedicated mapping editor.
- To the left of the interface, are listed the columns from the input file description.
- To the right of the interface, set the XML tree panel to reflect the expected XML structure output.

- Either create node by node the structure. For more information about the manual creation of an XML tree, see *Defining the XML tree on page 809*.
- In this use case, an XML template is used to populate the XML tree automatically.
- Right-click on the **root tag** displaying by default and select **Import XML tree** at the end of the contextual menu options.
- Browse to the XML file to be imported and click **OK** to validate the import operation.
- Then drag & drop each column name from the **Schema List** to the matching (or relevant) **XML tree** elements as described in *Mapping XML data on page 811*.
- The mapping is shown as blue links between the left and right panels.



- At last, define the node status where the loop should take place. In this use case, the *Cast* being the changing element on which the iteration should operate, this element will be the loop element.
- Right-click on the Cast element on the XML tree, and select **Set as loop element**.
- To group by movie, this use case needs also a group element to be defined.
- Right-click on the Movie parent node of the XML tree, and select **Set as group element**.
- The newly defined node status show on the corresponding element lines.
- Click **OK** to validate the configuration.
- Press **F6** to execute the Job.

VideoCollection2.xml

```
1  <?xml version="1.0" encoding="ISO-8859-15"?>
2
3  <VideoCollection>
4    <Movie id="1">
5      <Category>Comedy</Category>
6      <Year>1982</Year>
7      <Language>English</Language>
8      <Title>Tootsie</Title>
9      <Crew>
10       <Director>Sydney Pollack</Director>
11       <Cast>Dustin Hoffman</Cast>
12       <Cast>Jessica Lange</Cast>
13       <Cast>Sydney Pollack</Cast>
14     </Crew>
15   </Movie>
16   <Movie id="2">
17     <Category>Science Fiction</Category>
18     <Year>1997</Year>
19     <Language>French</Language>
20     <Title>The 5th Element</Title>
21     <Crew>
22       <Director>Eric Besson</Director>
23       <Cast>Bruce Willis</Cast>
```

The output XML file shows the structure as defined.

tDTDValidator

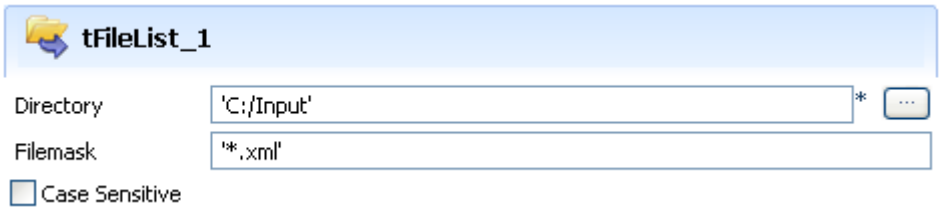
tDTDValidator Properties

Component family	XML	
Function	Validates the XML input file against a DTD file and sends the validation log to the defined output.	
Purpose	Helps at controlling data and structure quality of the file to be processed	
Basic settings	*Schema type* and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either built-in or remotely stored in the Repository but in this case, the schema is read-only. It contains standard information regarding the file validation.
	DTD file	Filepath to the reference DTD file.
	XML file	Filepath to the XML file to be validated.
	If XML is valid, display *If XML is not valid detected, display*	Type in a message to be displayed in the **Run** console based on the result of the comparison.
	Print to console	Select this check box to display the validation message.
Usage	This component can be used as standalone component but it is usually linked to an output component to gather the log data.	
Limitation	n/a	

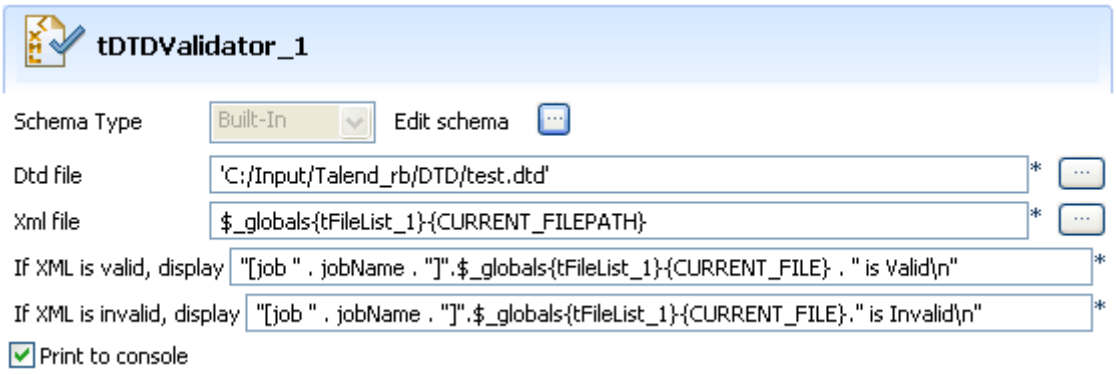
Scenario: Validating xml files

This scenario describes a Job that validates several files from a folder and outputs the log information for the invalid files into a delimited file.

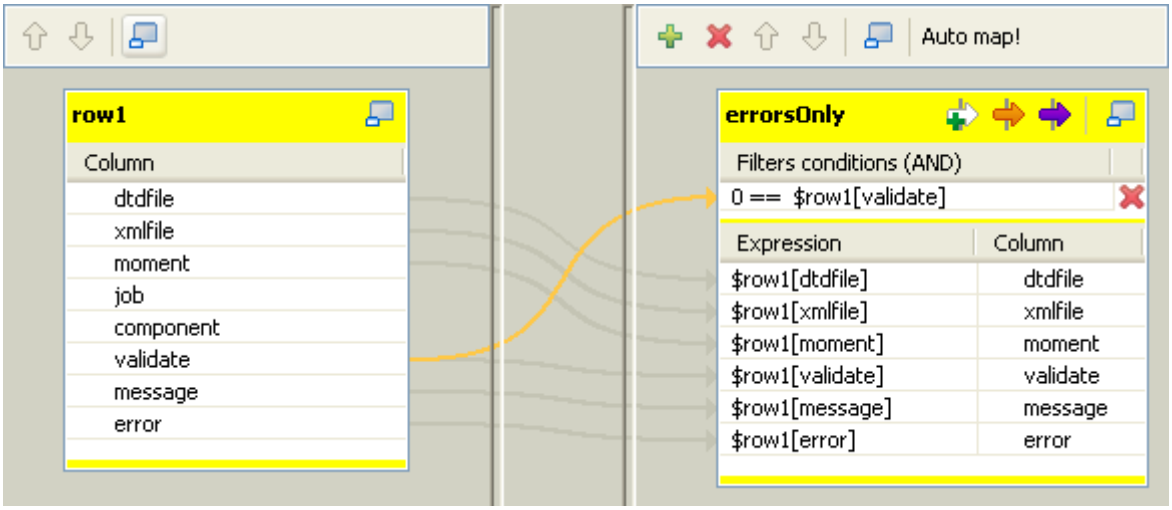
- Drop the following components from the **Palette** to the design workspace: **tFileList**, **tDTDValidator**, **tMap**, **tFileOutputDelimited**.
- Connect the **tFileList** to the **tDTDValidator** with an **Iterate** link and the remaining component using a **main** row.
- Set the **tFileList** component properties, to fetch an XML file from a folder.



- Change the **Filemask** to **.xml*. Mind the quotes depending on the Perl or Java version you are using.
- Clear the **Case Sensitive** check box.
- In the **tDTDValidate Component** view, the schema is read-only as it contains standard log information related to the validation process.
- Set the **DTD file** to be used as reference.



- Press **Ctrl+Space bar** to access the variable list. In the XML file field, select the current filepath global variable: $_globals{tFileList_1}{CURRENT_FILEPATH} (in Perl)
- In the various messages to display in the **Run** tab console, use the jobName to recall the job name tag. Recall the filename using the relevant global variable: $_globals{tFileList_1}{CURRENT_FILE}. Mind the Perl or Java operators such as the dot or the plus sign to build your message.
- Select the **Print to Console** check box.
- In the **tMap** component, drag and drop the information data from the standard schema that you want to pass on to the output file.



- Once the Output schema is defined as required, add a filter condition to only select the log information data when the XML file is invalid.
- Follow the best practice by typing first the wanted value for the variable, then the operator based on the type of data filtered then the variable that should meet the requirement. In this case (in Java and Perl): 0 == $row1[validate]
- Then connect (if not already done) the **tMap** to the **tFileOutputDelimited** component using a main row. Name it as relevant, in this example: *errorsOnly*.
- In the **tFileOutputDelimited Basic settings**, Define the destination filepath, the field delimiters and the encoding.
- Save your Job and press **F6** to run it.

```
Starting job DTDValidate at 15:29 21/06/2007.
[job jobName]test.xml is Invalid
[job jobName]test2.xml is Invalid
[job jobName]test3.xml is Invalid
[job jobName]test4.xml is Invalid
[job jobName]test5.xml is Invalid
[job jobName]test6.xml is Invalid
Job DTDValidate ended at 15:29 21/06/2007. [exit code=0]
```

On the **Run** console the messages defined display for each of the invalid files. At the same time the output file is filled with log data.

tExtractXMLField

tExtractXMLField properties

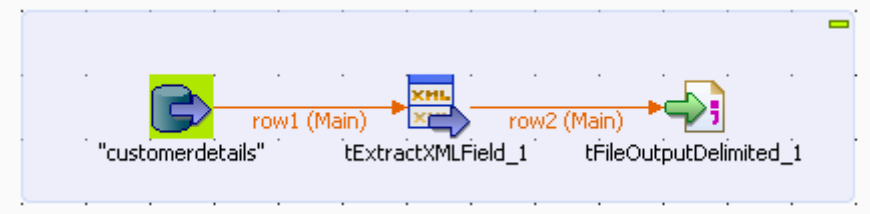
Component family	XML	
Function	**tExtractXMLField** reads an input XML field of a file or a database table and extracts desired data.	
Purpose	**tExtractXMLField** opens an input XML field, reads the XML structured data directly without having first to write it out to a temporary file, and finally sends data as defined in the schema to the following component via a **Row** link.	
Basic settings	*Property type*	Either **Built-in** or **Repository**.
		Built-in: No property data is stored centrally.
		Repository: Properties are stored in a repository file. When this file is selected, the fields that follow are pre-filled in using fetched data.
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the Repository.
		Built-in: You create the schema and store it locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide
		Repository: You already created the schema and stored it in the Repository, hence can be reused in various projects and job flowcharts. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	XML field	Name of the XML field to be processed. Related topic:*Defining variables from the Component view* of **Talend Open Studio** User Guide
	Loop XPath query	Node of the XML tree, which the loop is based on.
	Mapping	**Column**: reflects the schema as defined by the Schema type field. **XPath Query**: Enter the fields to be extracted from the structured input. **Get nodes**: Select this check box to recuperate the XML content of all current nodes specified in the **Xpath query** list or select the check box next to specific XML nodes to recuperate only the content of the selected nodes. The **Get Nodes** option is available only in **DOM4j** mode.
	Limit	Maximum number of rows to be processed. If Limit is 0, no rows are read or processed.

Usage	This component is an intermediate component. It needs an input and an output components.
Limitation	n/a

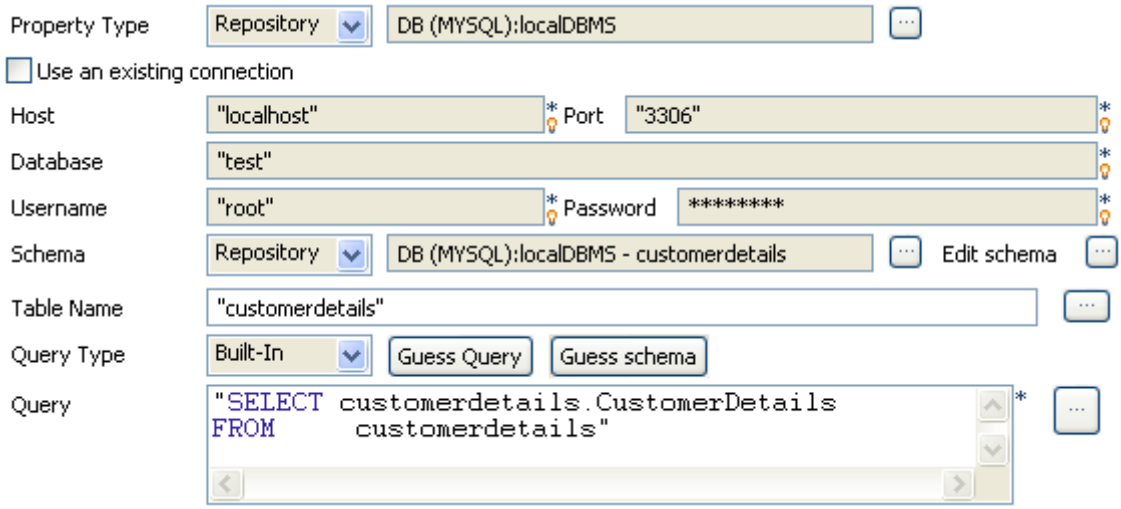
Scenario: Extract XML data from a field in a database table

This three-component Java scenario allows to read the XML structure included in the fields of a database table and then extracts the data.

- Drop the following components from the **Palette** onto the design workspace: **tMysqlInput**, **tExtractXMLField**, and **tFileOutputDelimited**.

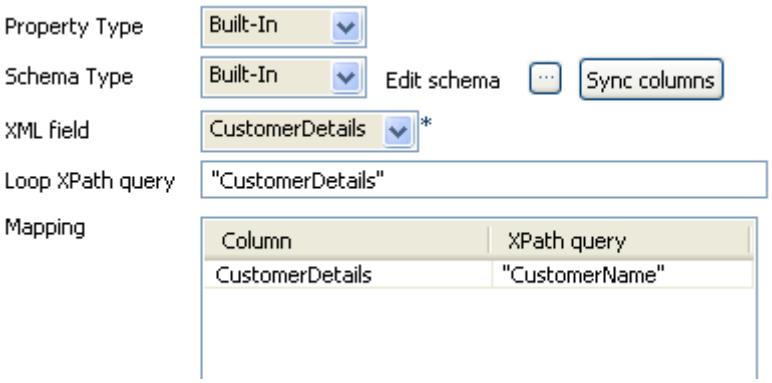


- Connect the three components using **Main** links.
- Double-click **tMysqlInput** to display its **Basic settings** view and define its properties.

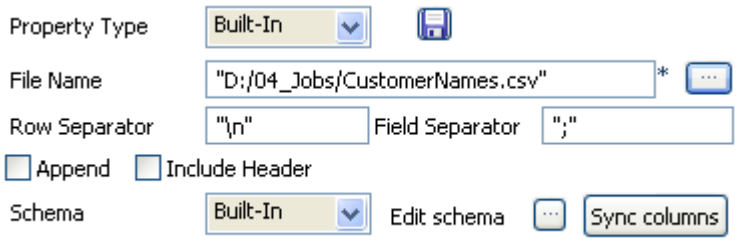


- If you have already stored the input schema in the **Repository** tree view, select **Repository** first from the **Property Type** list and then from the **Schema** list to display the **[Repository Content]** dialog box where you can select the relevant metadata.
  For more information about storing schema metadata in the **Repository** tree view, see *Setting up a DB connection* and *Drop components from the Metadata node* in **Talend Open Studio User Guide**.
- If you have not stored the input schema locally, select **Built-in** in the **Property Type** and **Schema** fields and enter the database connection and the data structure information manually. For more information about **tMysqlInput** properties, see *tMysqlInput on page 274*.

- In the **Table Name** field, enter the name of the table holding the XML data, *customerdetails* in this example.
- Click **Guess Query** to display the query corresponding to your schema.
- Double-click **tExtractXMLField** to display its **Basic settings** view and define its properties.



- In the **Property type** list, select **Repository** if you have already stored the description of your file in the **Repository** tree view. The fields that follow are filled in automatically with the stored data.
- If not, select **Built-in** and fill in the fields that follow manually.
- Click **Sync columns** to retrieve the schema from the preceding component. You can click the three-dot button next to **Edit schema** to view/modify the schema.
  Column in the **Mapping** table will be automatically populated with the defined schema.
- In the **Xml field** list, select the column from which you want to extract the XML data. In this example, the filed holding the XML data is called *CustomerDetails*.
- In the **Loop XPath query** field, enter the node of the XML tree on which to loop to retrieve data.
- In the **Xpath query** column, enter between inverted commas the node of the XML field holding the data you want to extract, *CustomerName* in this example.
- Double-click **tFileOutputDelimited** to display its **Basic settings** view and define its properties.



- In the **File Name** field, define or browse to the path of the output file you want to write the extracted data in.
- Click **Sync columns** to retrieve the schema from the preceding component. If needed, click the three-dot button next to **Edit schema** to view the schema.

- Save your Job and click **F6** to execute it.

```
1  Griffith Paving and Sealcoatin
2  Bill's Dive Shop
3  Childress Child Day Care
4  Facelift Kitchen and Bath
5  Terrinni & Son Auto and Truck
6  Kermit the Pet Shop
7  Tub's Furniture Store
8  Toggle & Myerson Ltd
9  Childress Child Day Care
10 Elle Hypnosis and Therapy Cent
11 Lennox Air Pollution Control
```

tExtractXMLField read and extracted the clients names under the node `CustomerName` of the *CustomerDetails* field of the defined database table.

tFileInputXML

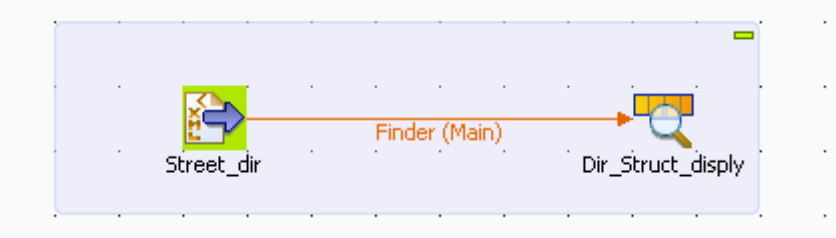
tFileInputXML Properties

Component family	XML or File/Input	
Function	**tFileInputXML** reads an XML structured file and extracts data row by row.	
Purpose	Opens an XML structured file and reads it row by row to split them up into fields then sends fields as defined in the Schema to the next component, via a **Row** link.	
Basic settings	*Property type*	Either Built-in or Repository.
		Built-in: No property data stored centrally.
		Repository: Select the Repository file where Properties are stored. The following fields are pre-filled in using fetched data.
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the Repository.
		Built-in: The schema will be created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide
		Repository: The schema already exists and is stored in the Repository, hence can be reused in various projects and job flowcharts. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	File Name/Input Stream	Name of the file to be processed. Related topic:*Defining variables from the Component view* of **Talend Open Studio** User Guide.
	Loop XPath query	Node of the tree, which the loop is based on.
	Mapping	**Column** reflects the schema as defined by the Schema type field. **XPath Query**: Enter the fields to be extracted from the structured input. **Get nodes**: Select this check box to recuperate the XML content of all current nodes specified in the **Xpath query** list or select the check box next to specific XML nodes to recuperate only the content of the selected nodes. The **Get Nodes** option is available only in **DOM4j** mode. See **Advanced settings** for the **Generation modes** available.
	Limit	Maximum number of rows to be processed. If Limit = 0, no row is read nor processed.

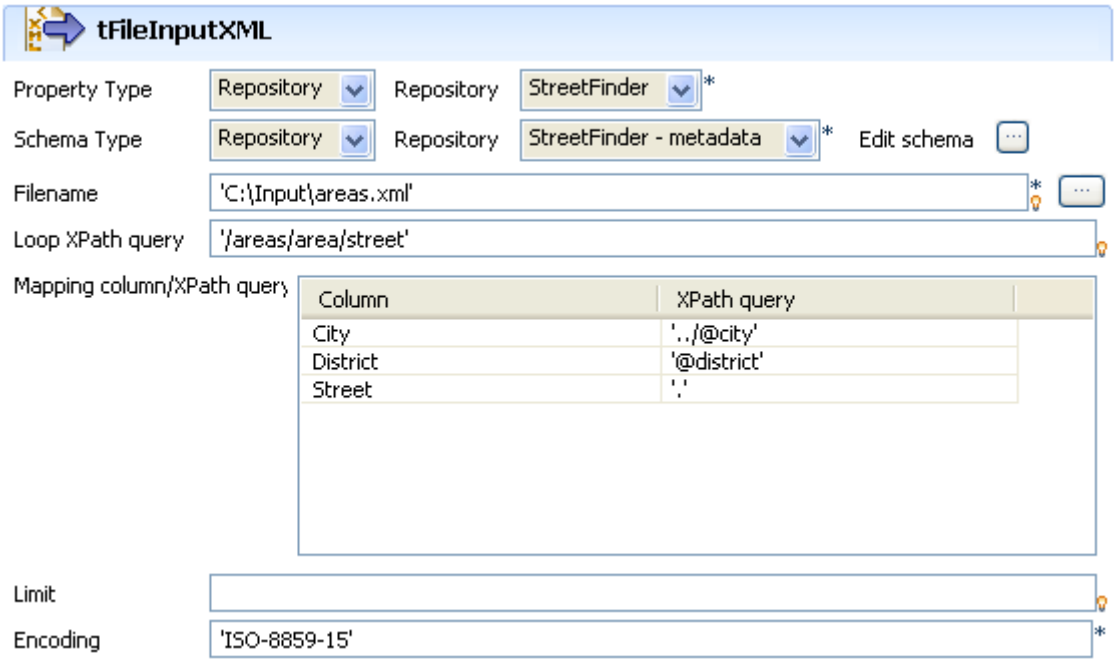
Advanced settings	*Advanced separator (for number)*	Select this check box to change data separator for numbers: **Thousands separator**: define the separators to use for thousands. **Decimal separator**: define the separators to use for decimals.
	Ignore the namespaces	Select this check box to ignore name spaces. **Generate a temporary file**: click the three-dot button to browse to the XML temporary file and set its path in the field.
	Use Separator for mode Xerces	Select this check box if you want to separate concatenated children node values. This field can only be used if the selected **Generation mode** is Xerces. The following field displays: **Field separator**: Define the delimiter to be used to separate the children node values.
	Encoding Type	Select the encoding type from the list or select **Custom** and define it manually. This field is compulsory for DB data handling.
	Generation mode	Select from the list the generation mode for the XML file according to available memory and desired speed.
	tStatCatcher Statistics	Select this check box to gather the job processing metadata at a job level as well as at each component level.
Limitation	n/a	

Scenario: XML street finder

This very basic scenario is made of two components. A **tFileInputXML** component extracts from the defined street directory file and the output is displayed on the **Run** console via a **tLogRow** component.



- Drop **tFileInputXML** and **tLogRow** from the **Palette** to the design workspace
- Connect both components together using a **Main** link.
- On the **Basic settings** panel of the **tFileInputXML**, define the properties:



- As the street dir file used as input file has been previously defined in the Metadata area, select **Repository** as **Property type**. This way, the properties are automatically leveraged and the rest of the properties fields are filled in (apart from Schema). For more information regarding the metadata creation wizards*Defining Metadata items* in **Talend Open Studio** User Guide.
- Select the same way the relevant schema in the Repository metadata list. **Edit schema** if you want to make any change to the schema loaded.
- The **Filename** shows the structured file to be used as input
- In **Loop XPath query**, change if needed the node of the structure where the loop is based.
- On the **Mapping** table, fill the fields to be extracted and displayed in the output.
- If the file size is consequent, fill in a **Limit** of rows to be read.
- Enter the encoding if needed then double-click on **tLogRow** to define the separator character.
- At last, press **F6** or go to the **Run** view and click **Run** to execute the Job. On the console, the fields defined in the input properties are extracted from the XML structured and displayed.

```
Starting job XMLStreetFinder at 12:42 05/01/2007.
Paris|2eme arrondissement|Rue de la Paix
Paris|8eme arrondissement|Champs Elysees
New York City|Manhattan|Madison avenue
New York City|Brooklyn|Washington heights
Job XMLStreetFinder ended at 12:42 05/01/2007. [exit code
```

tFileOutputXML

tFileOutputXML properties

Component family	XML or File/Output	
Function	**tFileOutputXML** outputs data to an XML type of file.	
Purpose	**tFileOutputXML** writes an XML file with separated data value according to a defined schema.	
Basic settings	*File name*	Name or path to the output file. Related topic: *Defining variables from the Component view* of **Talend Open Studio** User Guide
	Root tag	Wraps the whole output file structure and data.
	Row tag	Wraps data and structure per row
	Column name as tag name	Select this check box to leverage the column labels from the input schema, as data wrapping tag.
	Split output in files	If the XML file output is big, you can split the file every certain number of rows.
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the Repository.
		Built-in: The schema will be created and stored locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: The schema already exists and is stored in the Repository, hence can be reused in various projects and job designs. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	Sync columns	Click to synchronize the output file schema with the input file schema. The Sync function only displays once the Row connection is linked with the Output component.
	Encoding	Select the encoding from the list or select Custom and define it manually. This field is compulsory for DB data handling.
Usage	Use this component to write an XML file with data passed on from other components using a Row link.	
Limitation	n/a	

Scenario: From Positional to XML file

Find a scenario using **tFileOutputXML** component in the scenario of *tFileInputPositional on page 512*.

tWriteXMLField

tWriteXMLField properties

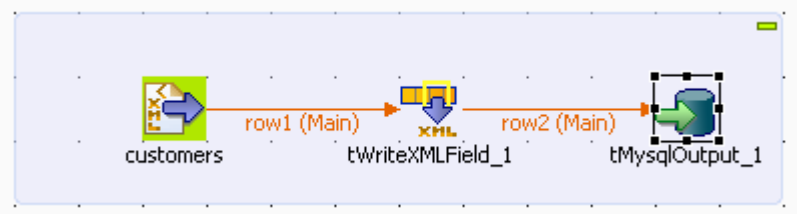
Component family	XML	
Function	**tWriteXMLField** outputs data to defined fields of the output XML file.	
Purpose	**tWriteXMLField** reads an input XML file and extracts the structure to insert it in defined fields of the output file.	
Basic settings	*Output Column*	Select the destination field in the output component where you want to write the XML structure.
	Configure Xml Tree	Opens the interface that supports the creation of the XML structure you want to write in a field. For more information about the interface, see *Defining the XML tree on page 809*.
	Schema type and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields that will be processed and passed on to the next component. The schema is either built-in or remote in the Repository.
		Built-in: You create the schema and store it locally for this component only. Related topic: *Setting a built-in schema* of **Talend Open Studio** User Guide.
		Repository: You already created the schema and stored it in the Repository, hence can be reused in various projects and job flowcharts. Related topic: *Setting a Repository schema* of **Talend Open Studio** User Guide.
	Sync columns	Click to synchronize the output file schema with the input file schema. The Sync function only displays once the Row connection is linked with the output component.
	Group by	Define the aggregation set, the columns you want to use to regroup the data.
Advanced settings	*Remove the xml declaration*	Select this check box if you do not want to include the XML header.
	Create empty element if needed	This check box is selected by default. If the **Related Column** in the interface that supports the creation of the XML structure has null values, or if no column is associated with the XML node, this option creates an open/close tag in the expected place.
	Create associated XSD file	If one of the XML elements is defined as a Namespace element, this option will create the corresponding XSD file. To use this option, you are required to select **Dom4J** as generation mode.

	Advanced separator (for number)	Select this check box if you want to modify the separators used by default for numbers. **Thousands separator**: enter between brackets the separators to use for thousands. **Decimal separator**: enter between brackets the separators to use for decimals.
	Generation mode	Select in the list the faster or slower generation mode according to the available memory in your system: **Fast but memory-consuming - Dom4J**, **Slow with no memory consumed**.
	Encoding Type	Select the encoding type in the list or select **Custom** and define it manually. This field is compulsory when working with databases.
	tStatCatcher Statistics	Select this check box to gather the job processing metadata at a job level as well as at each component level.
Usage	This component can be used as intermediate step in a data flow.	
Limitation	n/a	

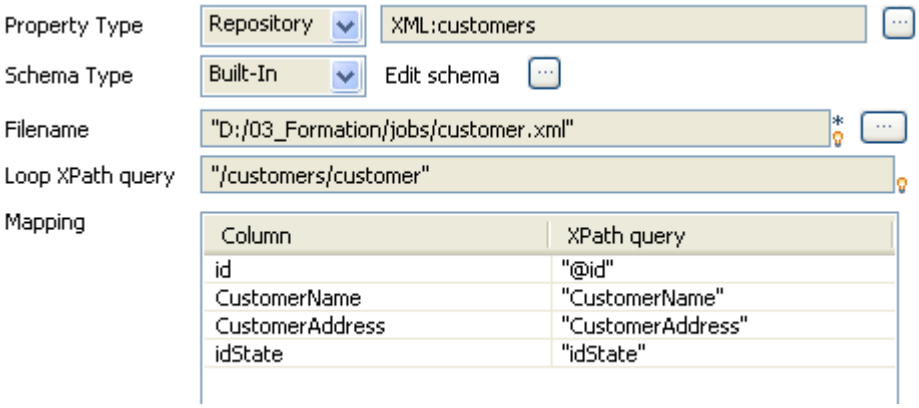
Scenario: Extracting the structure of an XML file and inserting it into the fields of a database table

This three-component Java scenario allows to read an XML file, extract the XML structure, and finally outputs the structure to the fields of a database table.

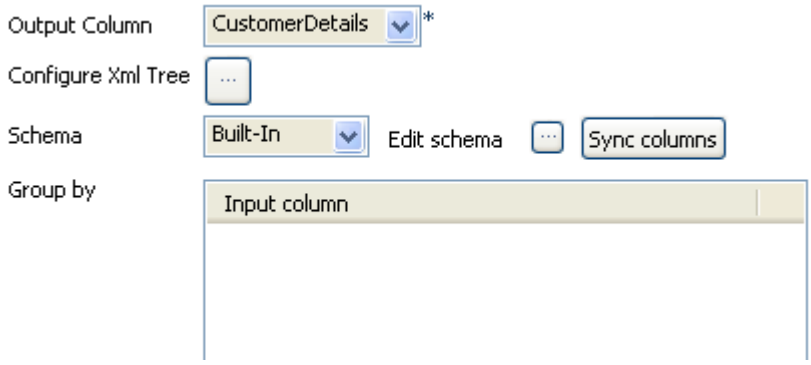
- Drop the following components from the **Palette** onto the design workspace: **tFileInputXml**, **tWriteXMLField**, and **tMysqlOutput**.



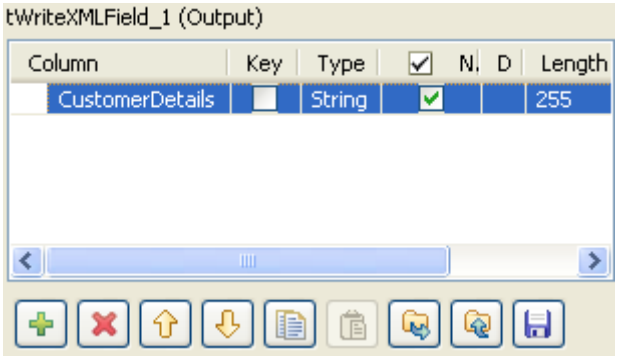
- Connect the three components using **Main** links.
- Double-click **tFileInputXml** to open its **Basic settings** view and define its properties.



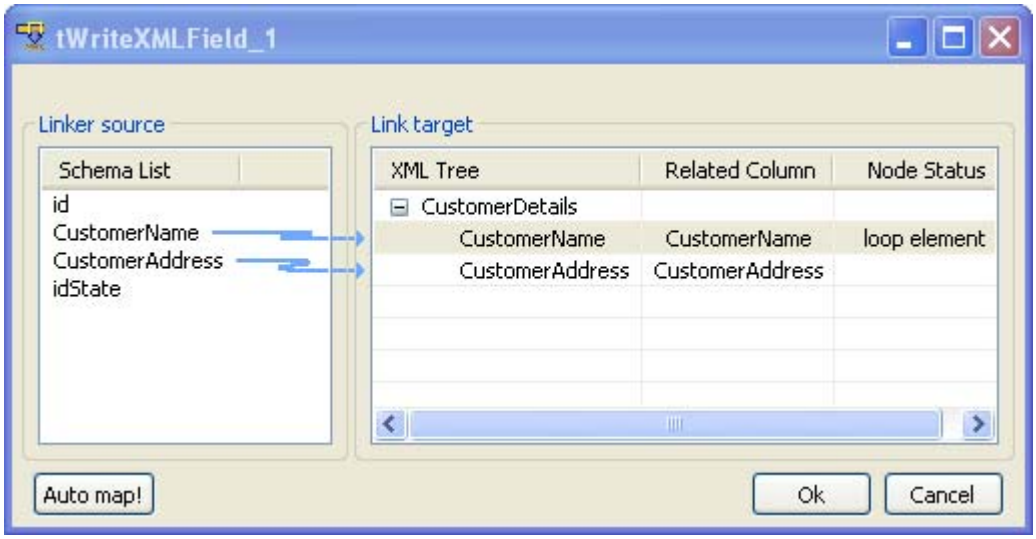
- If you have already stored the input schema in the **Repository** tree view, select **Repository** first from the **Property Type** list and then from the **Schema** list to display the **[Repository Content]** dialog box where you can select the relevant metadata.
  For more information about storing schema metadata in the **Repository** tree view, see *Setting up a File XML schema* and *Drop components from the Metadata node* in **Talend Open Studio User Guide**.
- If you have not stored the input schema locally, select **Built-in** in the **Property Type** and **Schema** fields and fill in the fields that follow manually. For more information about **tFileInputXML** properties, see *tFileInputXML on page 825*.
- If you have selected **Built-in**, click the three-dot button next to the **Edit schema** field to open a dialog box where you can manually define the structure of your file.
- In the **Look Xpath query** field, enter the node of the structure where the loop is based. In this example, the loop is based on the *customer* node. **Column** in the **Mapping** table will be automatically populated with the defined file content.
- In the **Xpath query** column, enter between inverted commas the node of the XML file that holds the data corresponding to each of the **Column** fields.
- In the design workspace, click **tWriteXMLField** and then in the **Component** view, click **Basic settings** to open the relevant view where you can define the component properties.



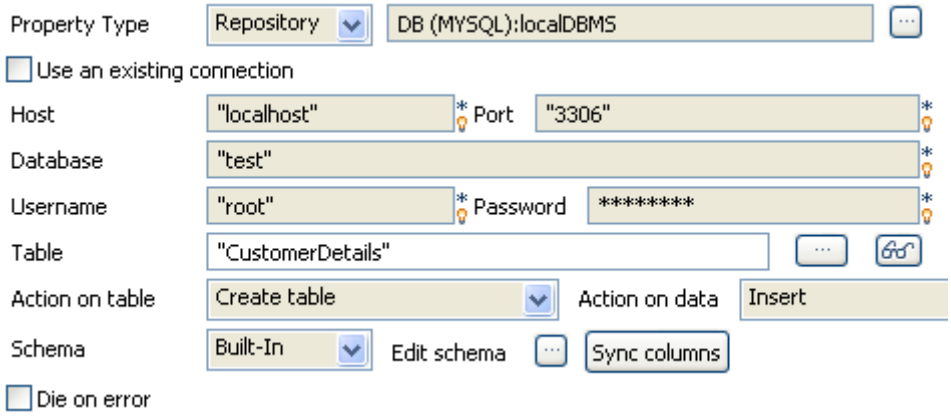
- Click the three-dot button next to the **Edit schema** field to open a dialog box where you can add a line by clicking the plus button.



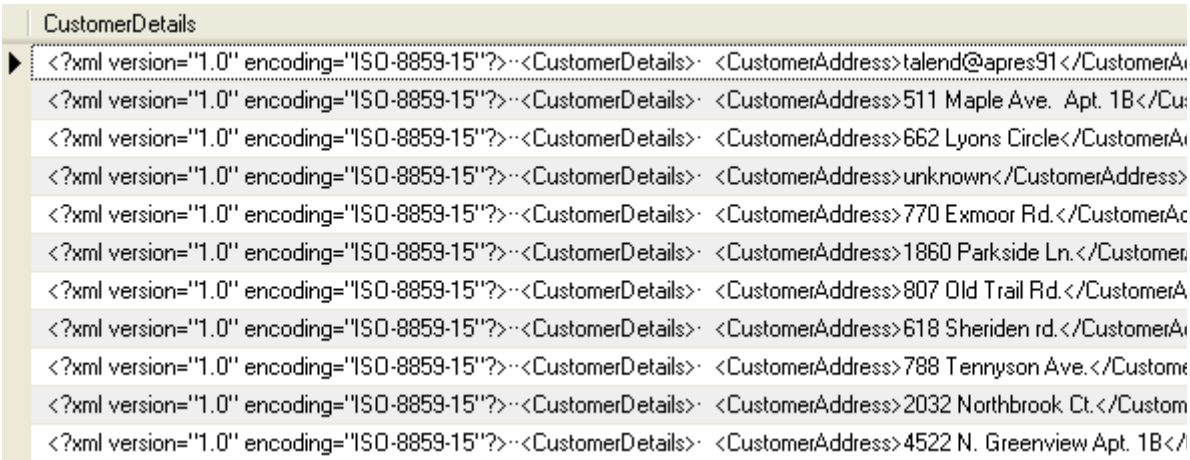
- Click in the line and enter the name of the output column where you want to write the XML content, *CustomerDetails* in this example.
- Define the type and length in the corresponding fields, *String* and *255*in this example.
- Click **Ok** to validate your output schema and close the dialog box.
- In the **Basic settings** view and from the **Output Column** list, select the column you already defined where you want to write the XML content.
- Click the three-dot button next to **Configure Xml Tree** to open the interface that helps to create the XML structure.



- In the **Link Target** area, click *rootTag* and rename it as *CustomerDetails*.
- In the **Linker source** area, drop *CustomerName* and *CustomerAddress* to *CustomerDetails*. A dialog box displays asking what type of operation you want to do.
- Select **Create as sub-element of target node** to create a sub-element of the *CustomerDetails node*.
- Right-click *CustomerName* and select from the contextual menu **Set As Loop Element**.
- Click **OK** to validate the XML structure you defined.
- Double-click **tMysqlOutput** to open its **Basic settings** view and define its properties.



- If you have already stored the schema in the **DB Connection** node in the **Repository** tree view, select **Repository** from the **Schema** list to display the **[Repository Content]** dialog box where you can select the relevant metadata.
  For more information about storing schema metadata in the **Repository** tree view, see *Setting up a DB connection* and *Drop components from the Metadata node* in **Talend Open Studio User Guide**.
- If you have not stored the schema locally, select **Built-in** in the **Property Type** and **Schema** fields and enter the database connection and data structure information manually. For more information about **tMysqlOutput** properties, see *tMysqlOutput on page 281*.
- In the **Table** field, enter the name of the database table to be created, where you want to write the extracted XML data.
- From the **Action on table** list, select **Create table** to create the defined table.
- From the **Action on data** list, select Insert to write the data.
- Click **Sync columns** to retrieve the schema from the preceding component. You can click the three-dot button next to **Edit schema** to view the schema.
- Save your Job and click **F6** to execute it.



tWriteXMLField fills every field of the *CustomerDetails* column with the XML structure of the input file: the XML processing instruction `<?xml version=""1.0"" encoding=""ISO-8859-15""?>`, the first node that separates each client

<CustomerDetails> and finally customer information <CustomerAddress> and <CustomerName>.

tXSDValidator

tDTDValidator Properties

Component family	XML	
Function	Validates the XML input file against a XSD file and sends the validation log to the defined output.	
Purpose	Helps at controlling data and structure quality of the file to be processed	
Basic settings	*Schema type* and *Edit Schema*	A schema is a row description, i.e., it defines the number of fields to be processed and passed on to the next component. The schema is either built-in or remotely stored in the Repository but in this case, the schema is read-only. It contains standard information regarding the file validation.
	XSD file	Filepath to the reference DTD file.
	XML file	Filepath to the XML file to be validated.
	If XML is valid, display *If XML is not valid detected, display*	Type in a message to be displayed in the **Run** console based on the result of the comparison.
	Print to console	Select this check box to display the validation message.
Usage	This component can be used as standalone component but it is usually linked to an output component to gather the log data.	
Limitation	n/a	

Related scenario

For related **tXSDValidator** use cases, see *Scenario: Validating xml files on page 818.*

tXSLT

tXSLT Properties

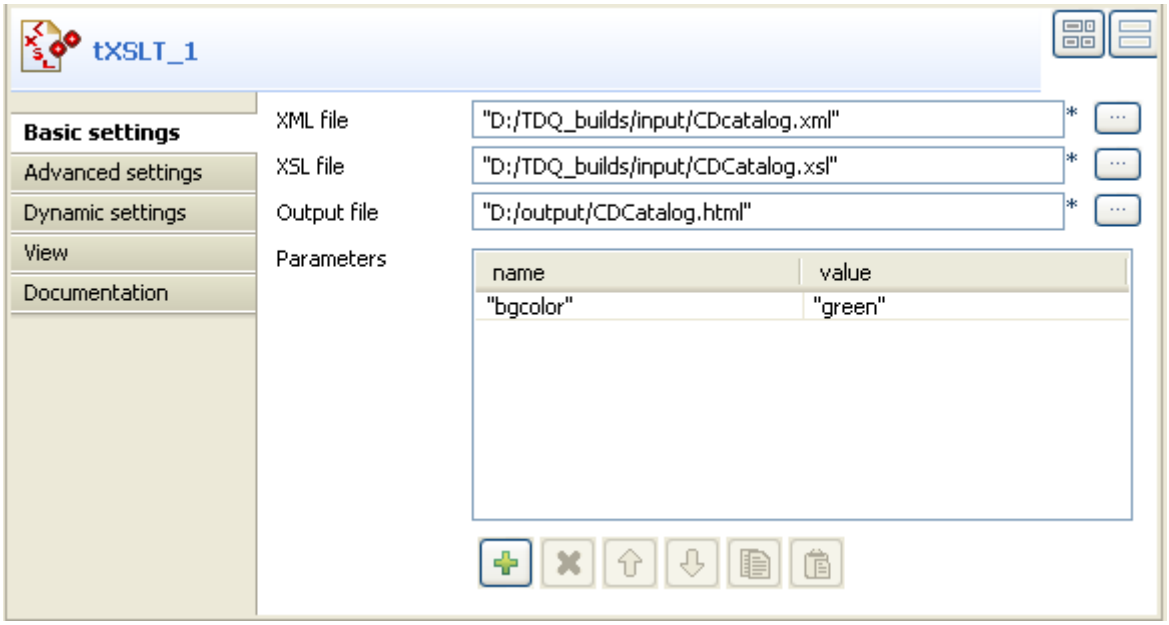
Component family	XML	
Function	Refers to an XSL stylesheet, to transform an XML source file into a defined output file.	
Purpose	Helps to transform data structure to another structure.	
Basic settings	*XML file*	File path to the XML file to be validated.
	XSL file	File path to the reference XSL transformation file.
	Output file	File path to the output file. If the file does not exist, it will be created. The output file can be any structured or unstructured file such as html, xml, txt or even pdf or edifact depending on your xsl.
	Parameters	Click the plus button to add new lines in the **Parameters** list and define the transformation parameters of the XSLT file. Click in each line and enter the key in the *name* list and its associated value in the *value* list.
Usage	This component can be used as standalone component.	
Limitation	n/a	

Scenario: Transforming XML to html using an XSL stylesheet

This Java scenario describes a two-component Job that converts xml data into an html document using an xsl stylesheet. It as well defines a transformation parameter of the xsl stylesheet to change the background color of the header of the created html document.

- Drop the **tXSLT** and **tMsBox** components from the **Palette** to the design workspace.

- Double-click **tXSLT** to open its **Basic settings** view where you can define the component properties.



- In the **XML file** field, set the path or browse to the xml file to be transformed. In this example, the xml file holds a list of MP3 song titles and related information including artist names, company etc.

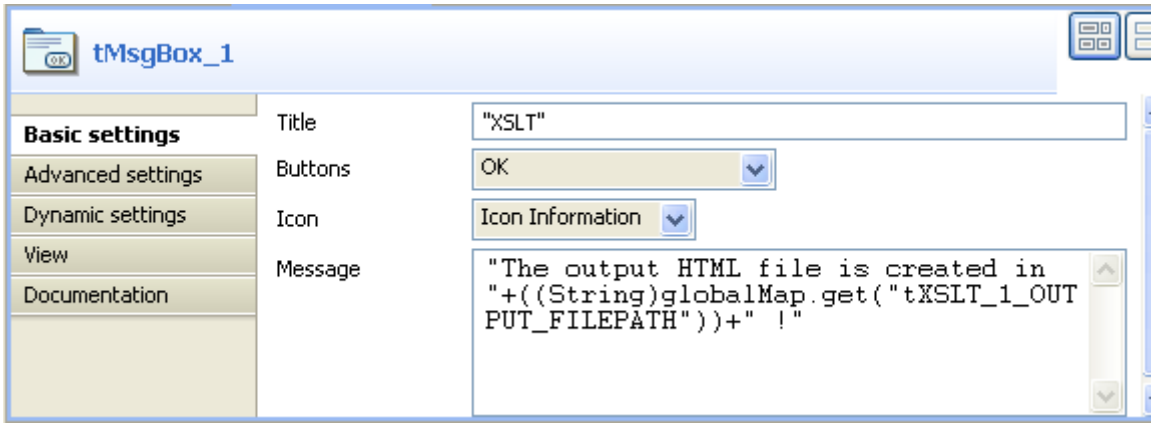
```
1  <?xml version="1.0" encoding="ISO-8859-1"?>
2  <catalog>
3    <cd>
4      <title>Empire Burlesque</title>
5      <artist>Bob Dylan</artist>
6      <country>USA</country>
7      <company>Columbia</company>
8      <price>10.90</price>
9      <year>1985</year>
10   </cd>
11   .
12   .
13
14 </catalog>
```

- In the **XSL file** field in the **Basic settings** view, set the path or browse to the relevant xsl file.
- In the **Output file** field, set the path or browse to the output html file.

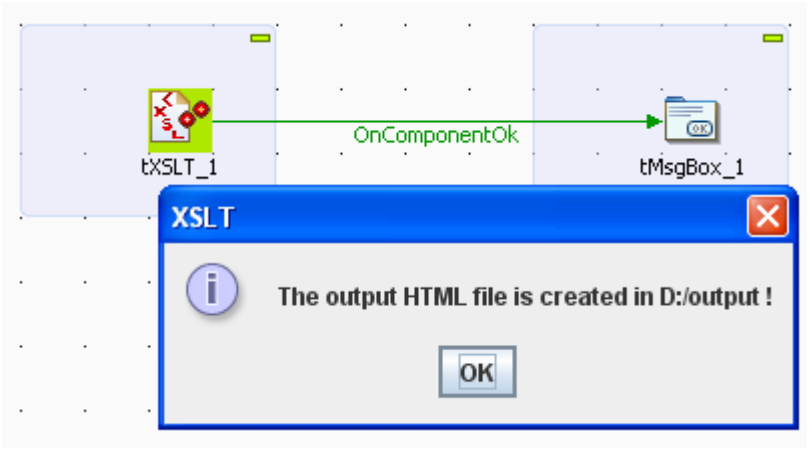
In this example, we want to convert the xml data into an html file holding a table heading followed by a table listing artists' names next to song titles.

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:param name="bgcolor" />

<xsl:template match="/">
  <html>
  <body>
  <h2>My CD Collection</h2>
  <table border="1">
    <tr bgcolor="{$bgcolor}">
      <th>Title</th>
      <th>Artist</th>
    </tr>
    <xsl:for-each select="catalog/cd">
    <tr>
      <td><xsl:value-of select="title"/></td>
      <td><xsl:value-of select="artist"/></td>
    </tr>
    </xsl:for-each>
  </table>
  </body>
  </html>
</xsl:template>

</xsl:stylesheet>
```

- In the **Parameters** area of the **Basic settings** view, click the plus button to add a line where you can define the name and value of the transformation parameter of the xsl file. In this example, the name of the transformation parameter we want to use is *bgcolor* and the value is *green*.
- Double-click the **tMsgBox** to display its **Basic settings** view and define its display properties as needed.



- Save the Job and press **F6** to execute it. The message box displays confirming that the output html file is created and stored in the defined path.



- Click **Ok** to close the message box.

You can now open the output html file to check the transformation of the xml data and that of the background color of the table heading.

My CD Collection

Title	Artist
Empire Burlesque	Bob Dylan
Hide your heart	Bonnie Tyler
Greatest Hits	Dolly Parton
Still got the blues	Gary Moore

V

X