talend | Connecting the Data-Driven Enterprise

Participant Guide

DI Basics

Version 6.3.1

Talend Inc.
800 Bridge Parkway, Suite 200
Redwood City, CA 94065
United States
+1 (650) 539 3200

Welcome to Talend Training

Congratulations on choosing a Talend training course.

Working through the course

You will develop your skills by working through use cases and practice exercises using live software. Completing the exercises is critical to learning!

If you are following a self-paced, on-demand training (ODT) module, and you need an answer to proceed with a particular exercise, use the help suggestions on your image desktop. If you can't access your image, contact customercare@talend.com.

Exploring

You will be working in actual Talend software, not a simulation. We hope you have fun and get lots of practice using the software! However, if you work on tasks beyond the scope of the training, you could run out of time with the environment, or you could mess up data or Jobs needed for subsequent exercises. We suggest finishing the course first, and if you have remaining time, explore as you wish. Keep in mind that our technical support team can't assist with your exploring beyond the course materials.

For more information

Talend product documentation (help.talend.com)

Talend Community (community.talend.com)

Sharing

Intentionally blank

CONTENTS

Intentionally blank

LESSON 1

Getting Started with Talend Studio

This chapter discusses:

Getting Started

Lesson Overview

During this training, you will be assigned a **Talend Data Fabric** training environment. The purpose of this lesson is to get familiarized with the Talend Studio by covering the essentials.

First, you will start Talend Studio and you will explore the different sections of the Main Window.

Then you will create a first Job by adding components, connecting and configuring them.

This Job is very simple: it will display a *"Hello World"* message in the execution console.

Finally, you will execute the Job in the development environment to test it.

Objectives

After completing this lesson, you will be able to:

- Start the Talend Studio
- Create a new Job
- Find and Add components
- Connect and configure components
- Find Help on components
- Run a Job

Next Step

The first step is to open the existing training project in Talend Studio and explore the interface: Start Talend Studio

Starting Talend Studio

Overview

Before starting to develop using Talend Studio, you need to get familiarized with the environment. The first step is to start the software and open an existing project. Then you will discover the Talend Studio main window and tool bars.

Open a Talend Studio Project

1. START TALEND STUDIO

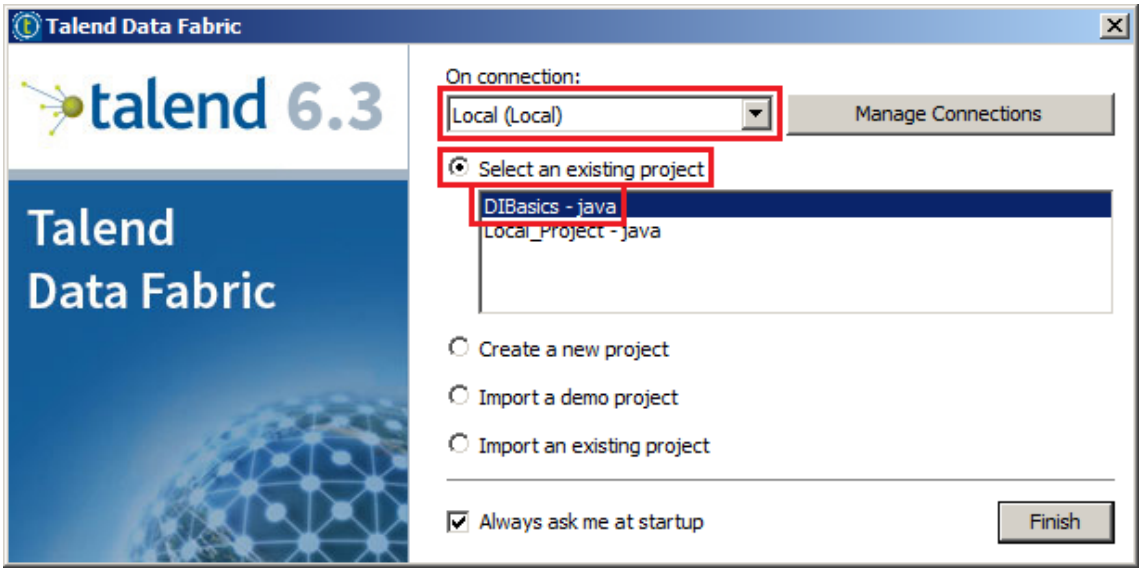
   Start Talend Studio by double-clicking the following icon on your desktop:

2. SELECT A PROJECT

   The **Talend Data Fabric** window opens. Using the **Local** connection, click **Select an existing project** and then select the **DIBasics** project, which has already been created for you.

   Click **Finish** and the main project window will open.

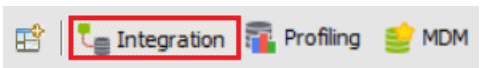


NOTE:

- If prompted to connect to the TalendForge community, select **Skip this Step**.
- If presented with the **Welcome** page, click **Start now!**

3. SWITCH TO THE INTEGRATION PERSPECTIVE

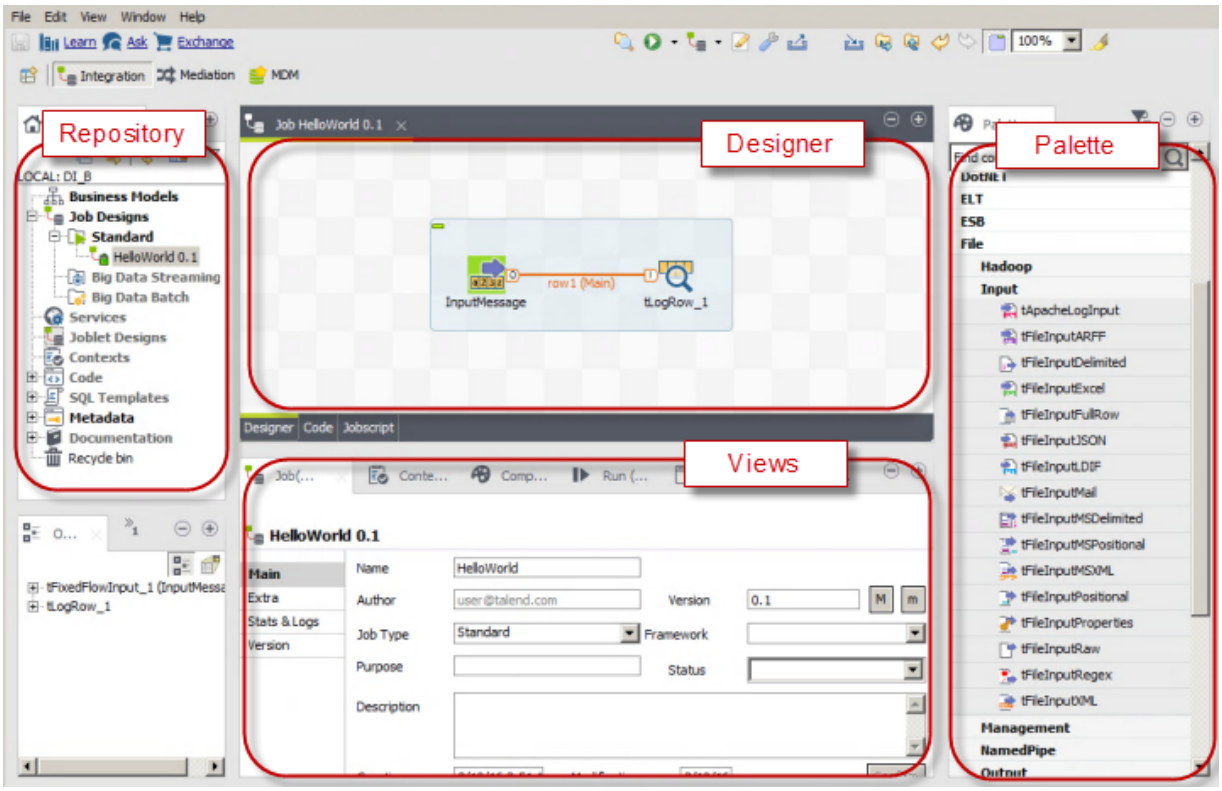
   Select the **Integration** perspective, if it is not already selected. This is the perspective you will be using for this entire course.



Understand the Talend Studio Main Window

Before starting to build your first Job, take a look at the different sections from the **Talend Studio** interface.

For now, the panels and views in your environment are empty as there is no Job defined.
This is how your environment will look after building your first Job in the next section.



The main panels and views of Studio are as follows:

1. REPOSITORY

   On the left is the **Repository** view, containing the project structure with several nodes:

   - **Job Designs** contains all your developed Jobs, which can also be organized into folders. A Job Design is a graphical design consisting of one or more components connected together. This allows you to set up and run data flow management processes. A Job Design translates business needs into code, routines, and programs. In other words, it implements your data flow.
   - The **Contexts** folder groups files holding context variables that you can reuse across different Jobs.
   - The **Metadata** folder stores reusable information on files, databases, and systems that you need to access from within your Jobs.

2. PALETTE

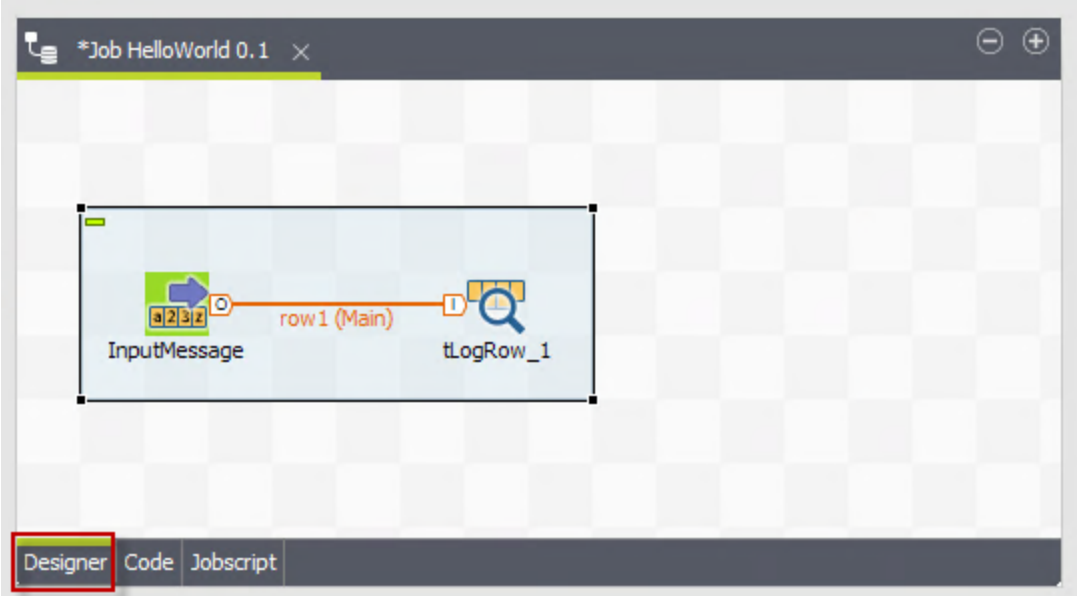
   On the right, the **Palette** provides a collection of components that you can add to your Job design.

3. DESIGNER

   In the middle is the **Designer** view. This is the main area where you will graphically design your Jobs.

   Underneath are several tabs:

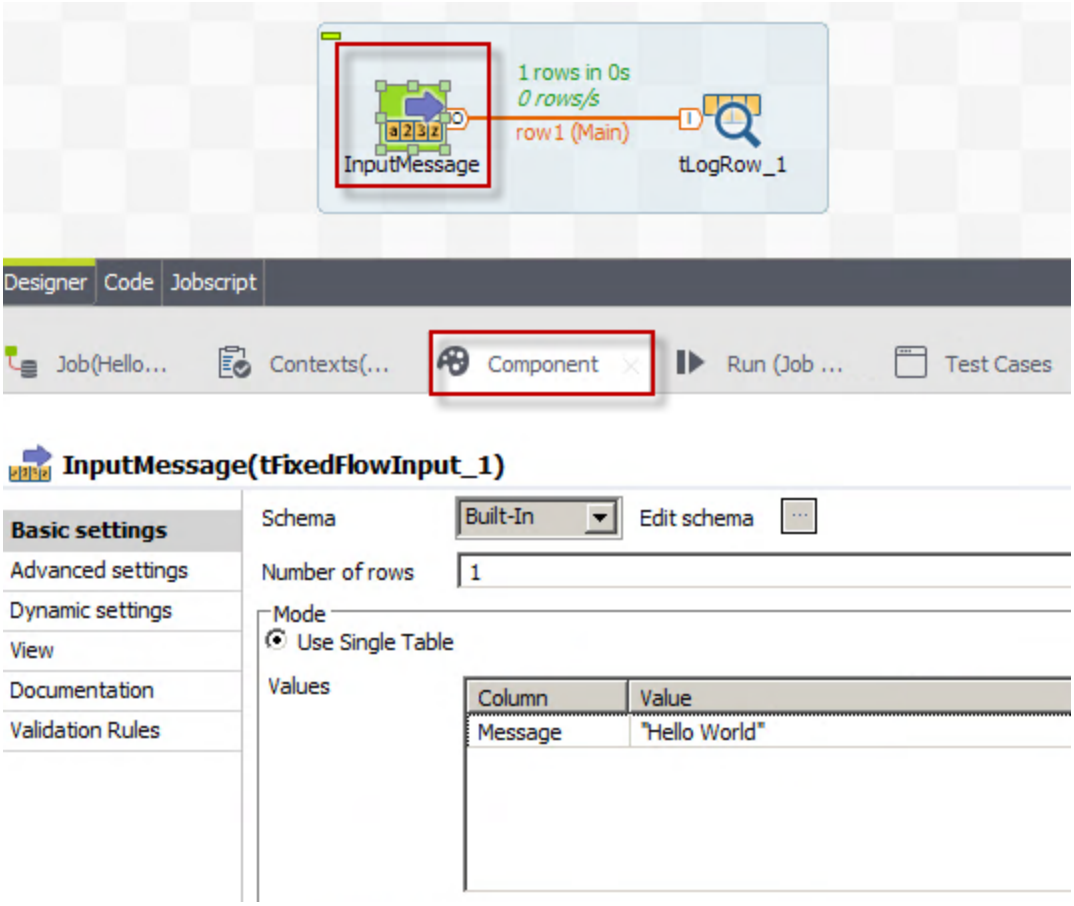
   - The **Designer** tab is the default view that displays the Job in a graphical mode.
   - The **Code** tab enables you to view the underlying code being generated. This can be helpful in pinpointing design errors or incorrect expressions.
   - The **Jobscript** tab enables you to visualize and edit the Job script. Job script is a JSON-like application programming interface that enables you to textually define Jobs.



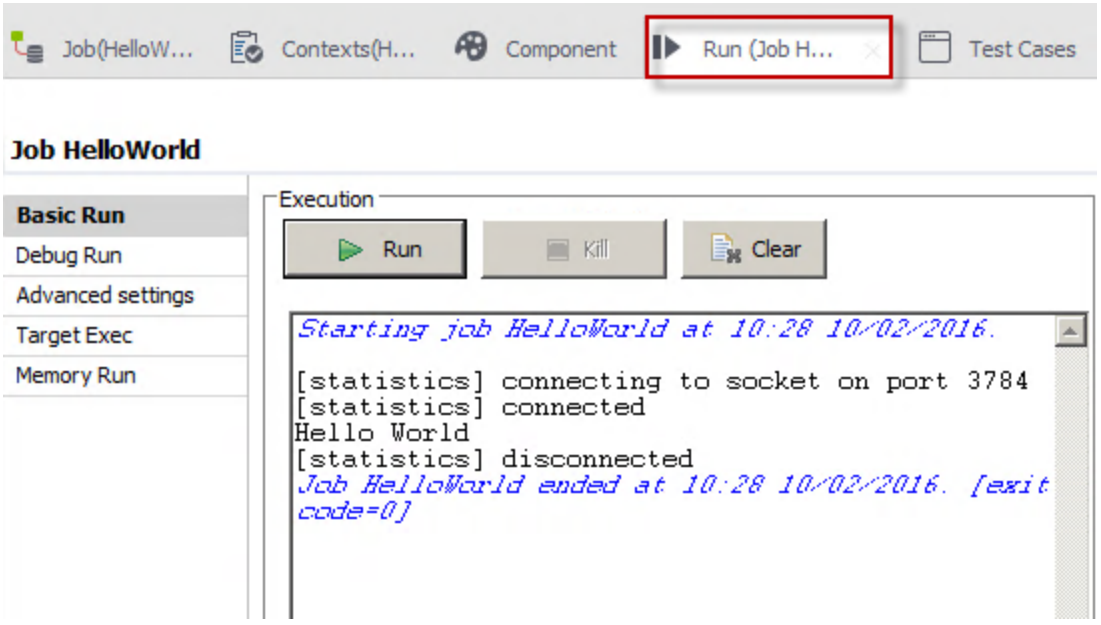
4. ADDITIONAL VIEWS

   In the lower portion of the workspace, you will find several other views.

   The **Component** view displays the properties of the selected element in the **Designer**. These properties can be edited to change or set the parameters related to a particular component.



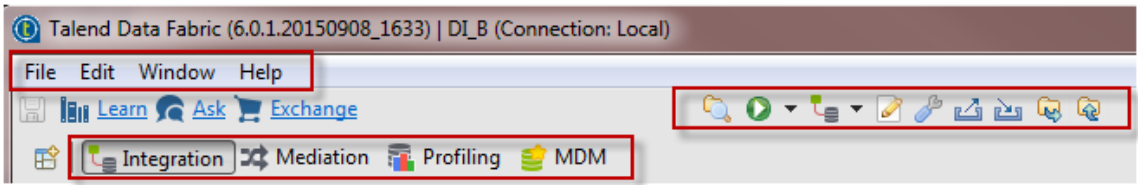
The **Run** view is used to execute Jobs, configure settings for the execution environment, and explore the results.



5. TOOLBARS

At the top of the **Talend Data Fabric** main window, several tool bars group the most commonly-used features:

- The main menubar
- The quick access toolbar
- The perspectives switcher. Depending on the size of your Talend Studio window, the perspectives switcher toolbar will be displayed on the top right corner, just after the quick access toolbar or on the left side, below the quick access toolbar.



Perspectives

A Perspective defines the initial set and layout of views in the Studio. The selected Perspective is always the active one.

The **Talend Data Fabric** Studio will be used to create data integration projects in this course, but it can also be used to create other types of projects, such as: big data, application integration, master data management, and data profiling. Each of these project types provides a unique perspective for managing them.

In this course, only the **Integration** perspective will be used, which is the default perspective when opening the **DIBasics** project.

If you click and open another perspective, you can always come back to the **Integration** perspective using the perspective switcher.

Next

Now that you have opened the training project and you have taken a quick tour through the Talend Studio interface, you can start to build your First Job.

Creating a First Job

Overview

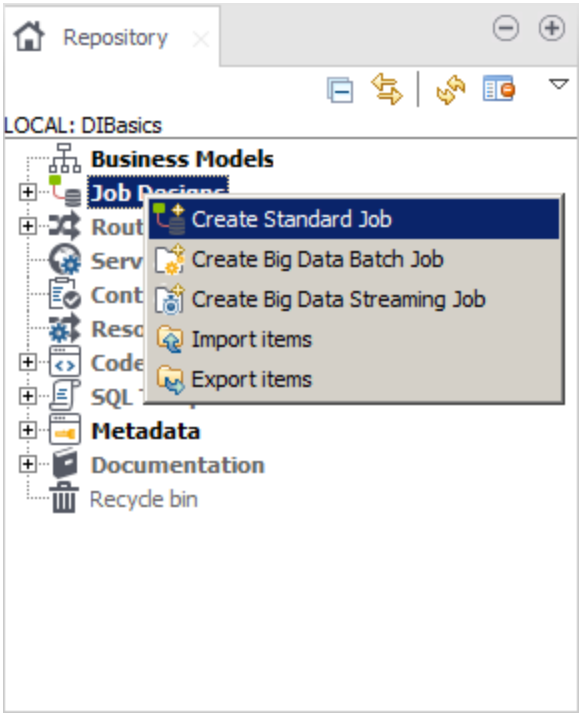
In this section, you will create your first Job and learn how to add and connect components. You will make use of the **Palette** in order to search for components. You will also configure the components in the corresponding **Views** and finally you will execute the Job by using the **Run** View .

The Job is very simple: it will define a *"Hello World"* message that will be displayed in the execution console.

Create a Job

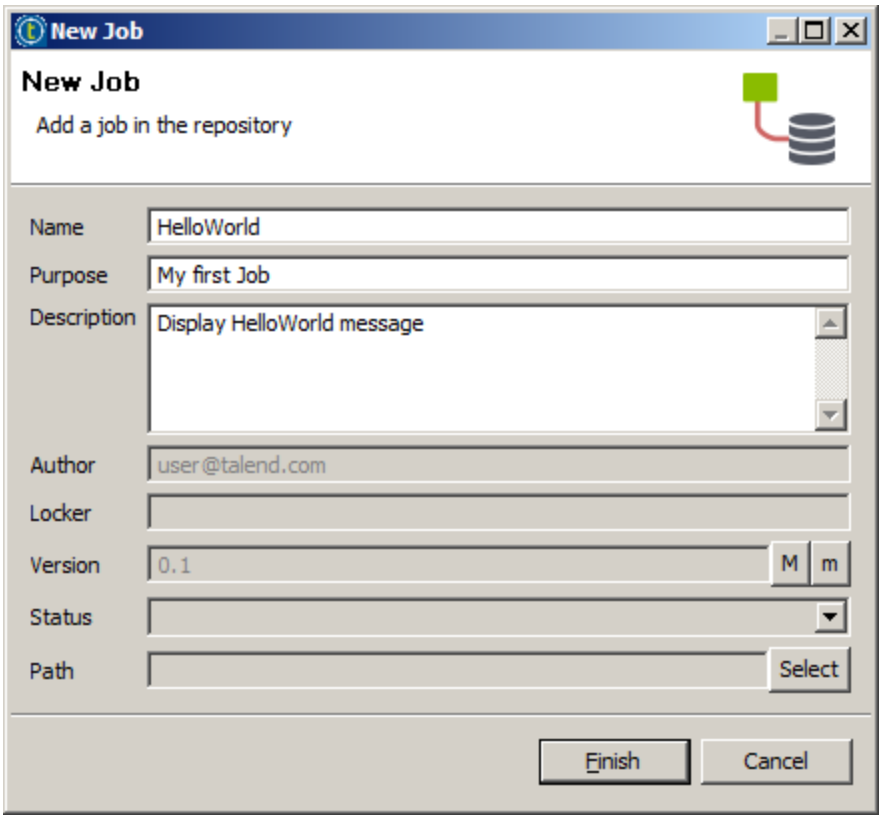
1. CREATE A STANDARD JOB

   Create a new Job by right-clicking on **Repository > Job Designs**, then selecting **Create Standard Job**.



2. ENTER JOB DETAILS

   Enter *HelloWorld* for the **Name**, then populate the **Purpose** and **Description**. Click **Finish** when done.

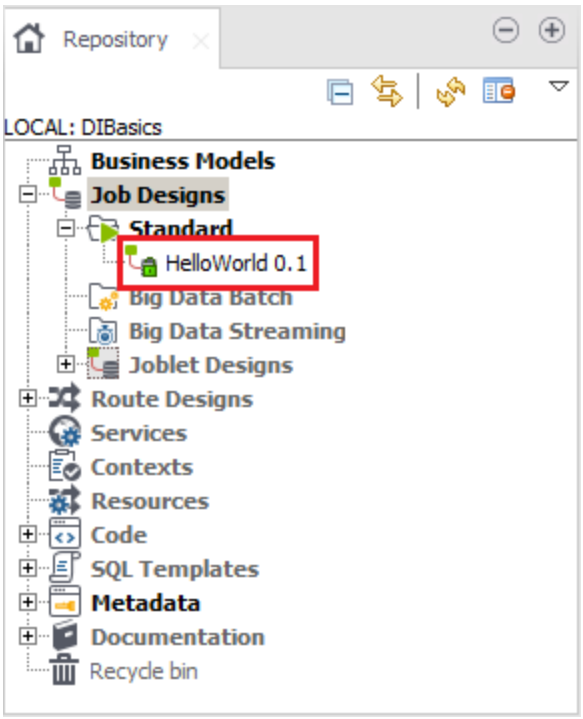


WARNING:
If you try to include illegal characters in the Job name, you will be notified in the Studio. For example, *HelloWorld* is legal, but *Hello World* will not be accepted due to the space in the Job name.

3. OBSERVE THE CHANGES

   Notice the changes in the **Designer**, the **Palette**, and the rest of the views.

   The newly created **Job** is opened in the **Designer** and components are now available in the **Palette**.

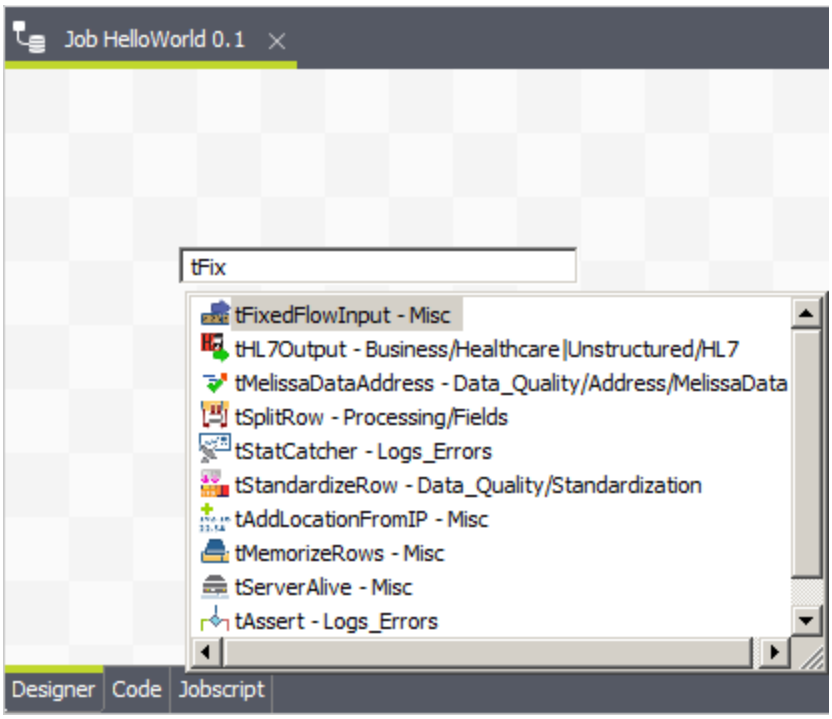


Add Components

You are going to add two components to your newly created Job. You need a component to input the message, and a component to output the message.

1. FIND THE INPUT COMPONENT

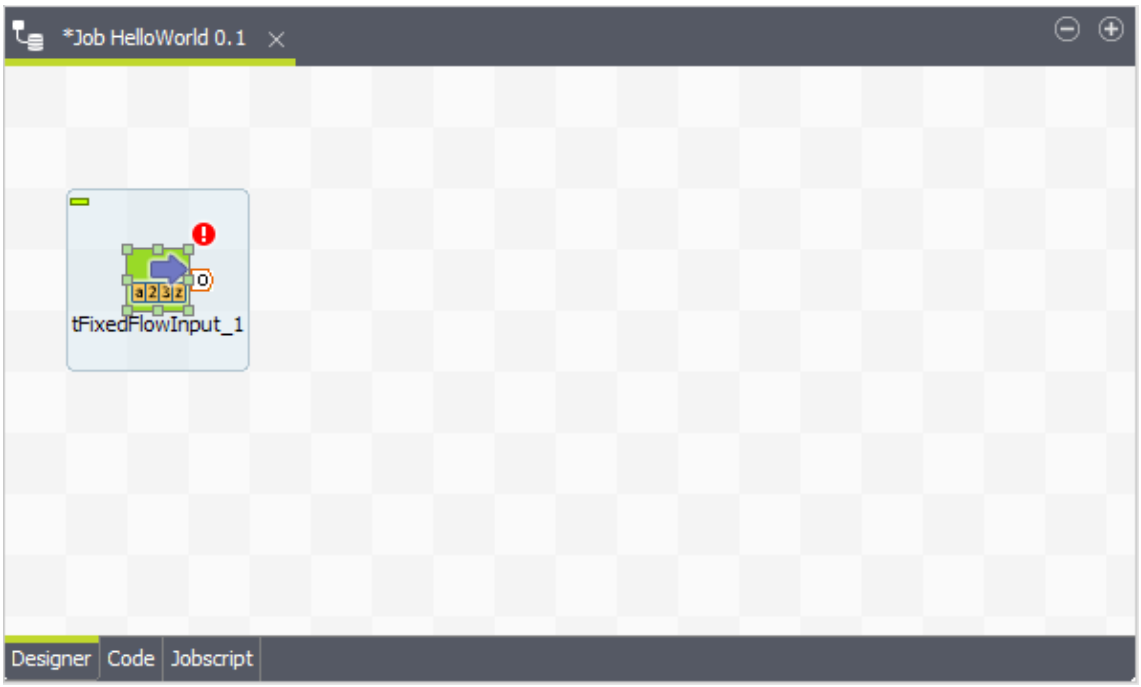
   Start adding a component to the Job by clicking anywhere in the left-hand portion of the **Designer**. Begin typing the name of the component, in this case *tFixedFlowInput*, and a list of matching components will appear. Search the list for the **tFixedFlowInput** component. Notice the suffix **Misc**. This indicates the folder within the **Palette** in which the component can also be found.



TIP:
The position of the components does not affect the resulting code or Jobscript, but best practices dictate a scheme for positioning components. In this scheme, data flow generally proceeds from left to right and top to bottom, so input components are typically placed in the top-left portion of the Designer.

2. ADD THE COMPONENT

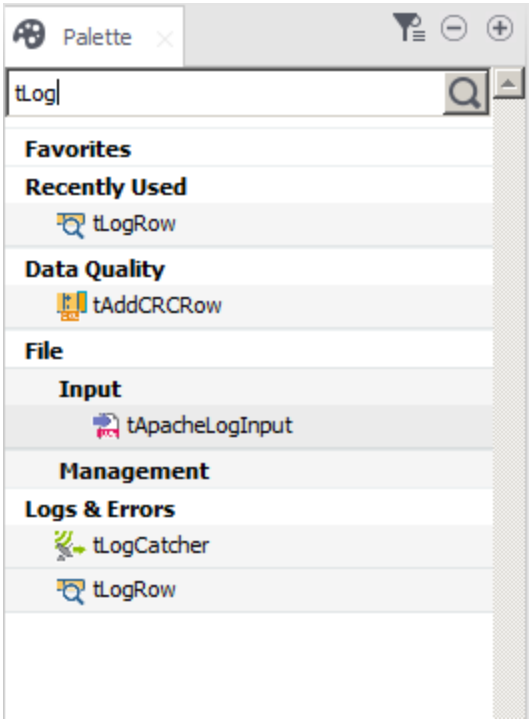
   Double-click the **tFixedFlowInput** component in the list to plant the component at the selected spot in the **Designer**.



The component is now added to the Job. This "click and type" method represents a quick and easy way to add components to your Jobs. In the next step, you will add a second component using an alternate method.

3. FIND ANOTHER COMPONENT USING THE PALETTE

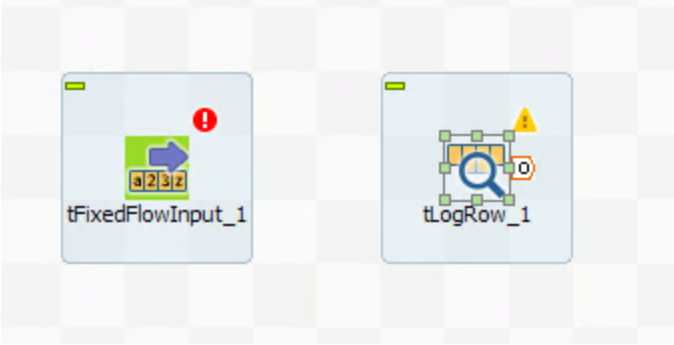
In the **Palette** view, begin typing the name of another component, in this case *tLogRow*, into the **Find Component** box and then click the **Search** button ( ). You can also press the **Enter** key.



The component has been found under the **Logs & Errors** folder.

4. ADD THE COMPONENT

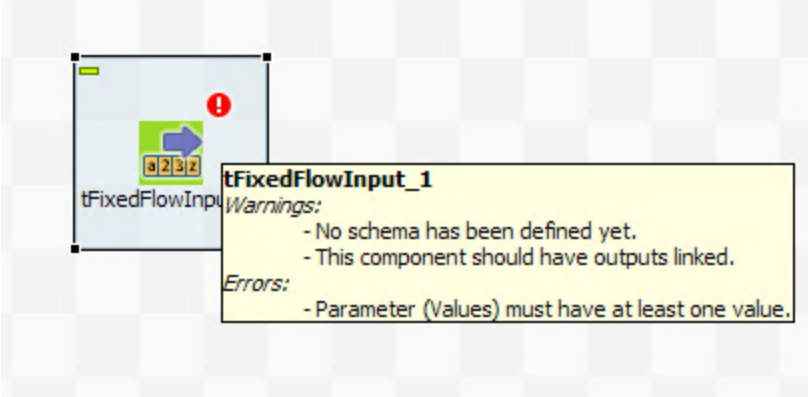
   Select the **tLogRow** component in the **Palette** view, then click on a spot to the right of **tFixedFLowInput_1** in the **Designer** to place the new component.



   The component is now added to the Job. This approach represents another way you can browse components and add them to your Job. Note that it is also possible to simply drag and drop the component from the **Palette** to your Job's design space.

5. INSPECT WARNINGS

   Notice the markings on the top-right corners of the components. Hover the mouse over the marking for **tFixedFlowInput_1** to see the list of issues.



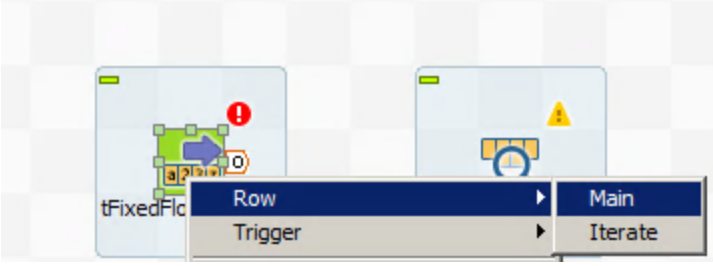
   These warnings tell you that the component needs to be configured with a schema and also must be connected to an output component.

   Do the same for **tLogRow_1** and you will see that the issue is also related to the required connection.

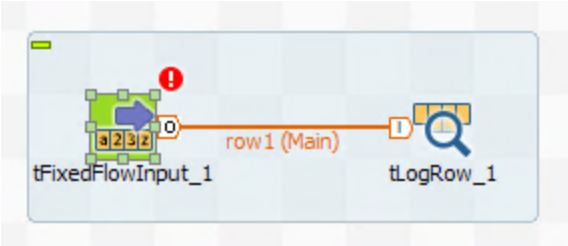
Connect Components

1. CONNECT THE COMPONENTS

   Right-click on the **tFixedFlowInput** component, then select **Row > Main**.



Move the mouse over top of **tLogRow_1** component and click on it.



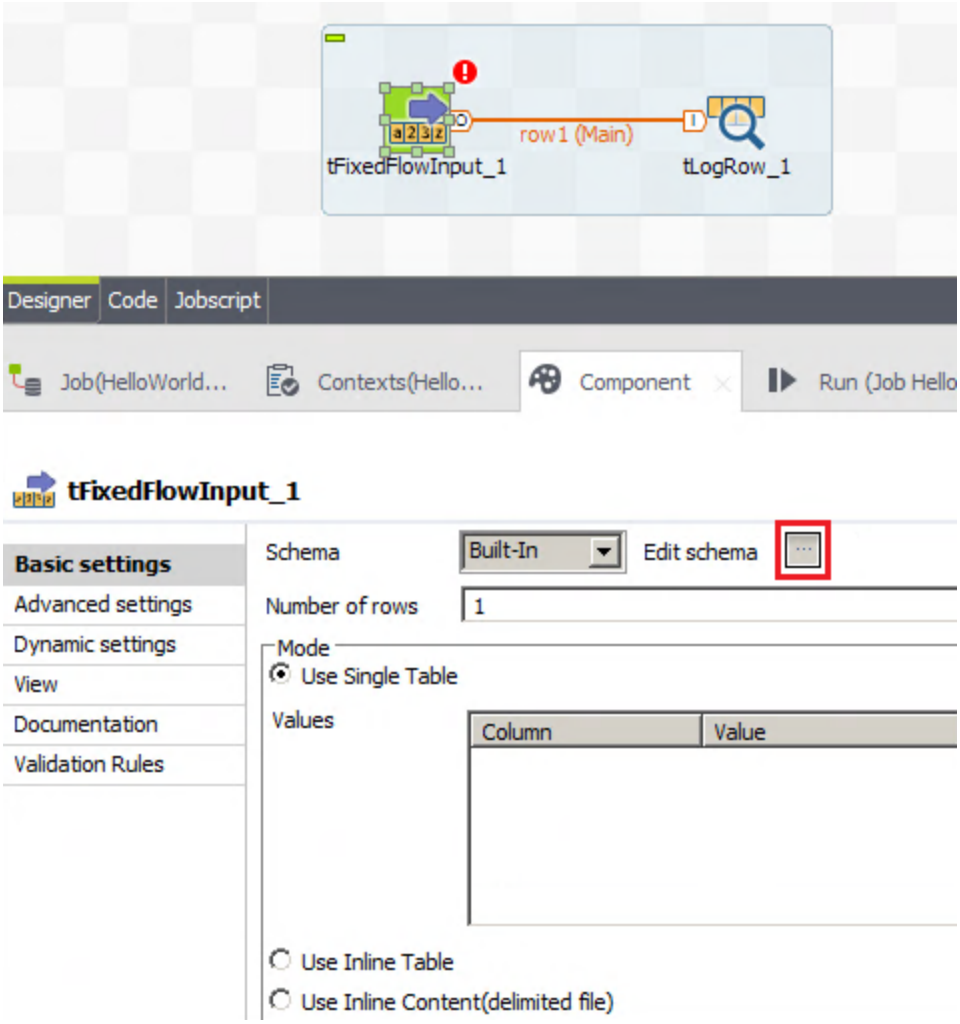
The link will be created and displayed. A row (or link) allows data to flow from one component to another. The arrow indicates the direction of the flow.

Configure Components

1. OPEN THE COMPONENT VIEW

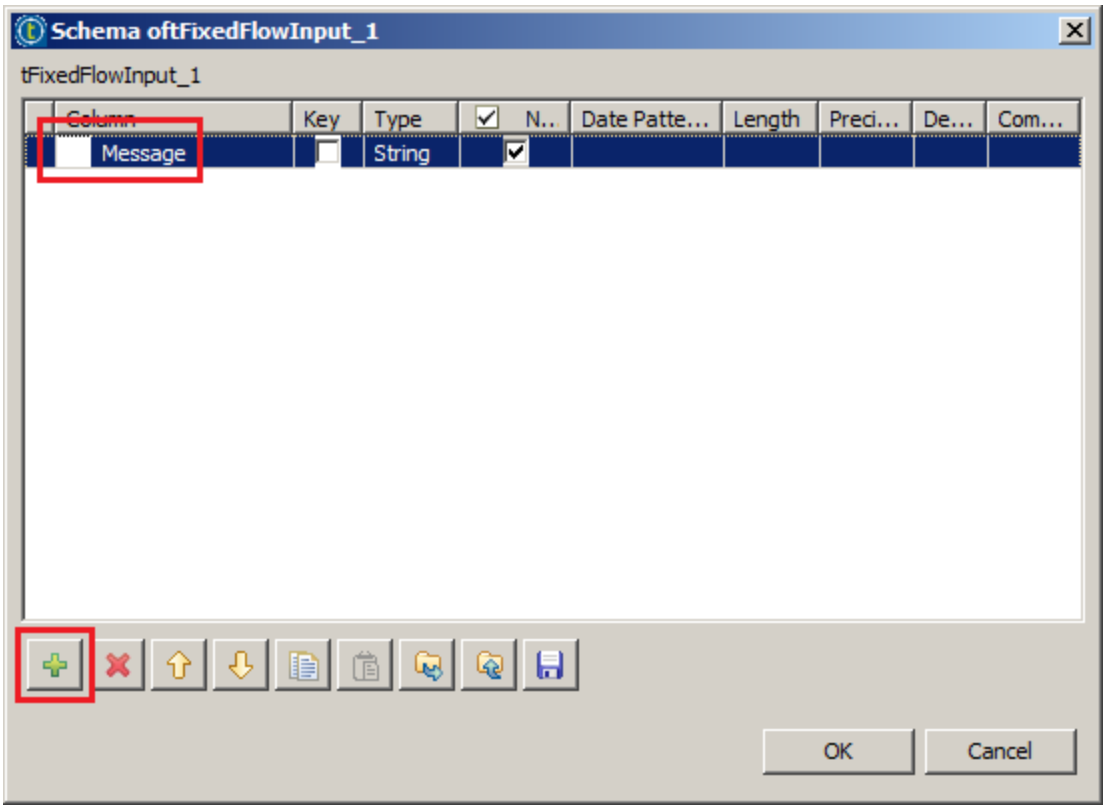
   Double-click **tFixedFlowInput_1** to open the **Component** view, which is located in the lower portion of the main window.

   Then click on the **[...]** button near **Edit schema** to configure the schema.

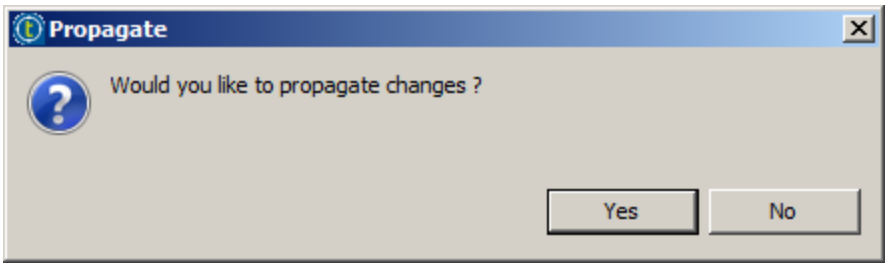


2. ADD NEW COLUMN

   Click the Add button (+) to add a new column. Change the Column name to *Message*, then click **OK**.



Answer **Yes** when asked if you would like to propagate changes.



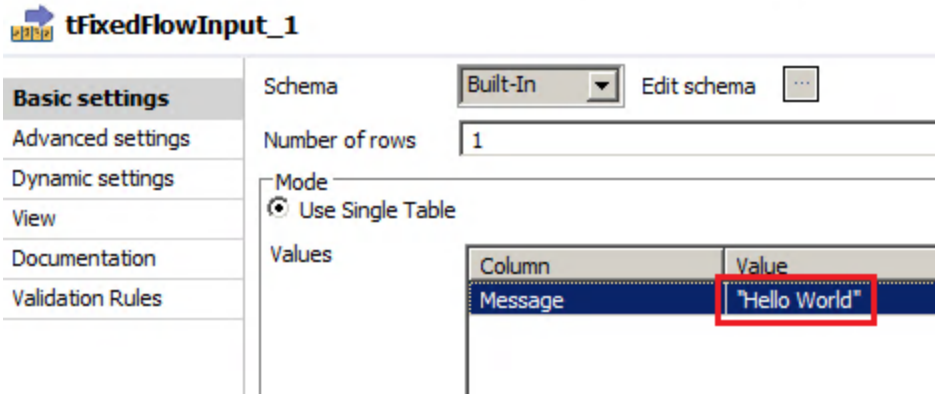
NOTE:
You will get this question every time you update a schema. If you have more than one Job using the schema, it will be propagated to all the Jobs using it.

Note that the **tFixedFlowInput_1** component shows no more errors, as the input schema has been defined and the component has an output linked.

3. PROVIDE A MESSAGE

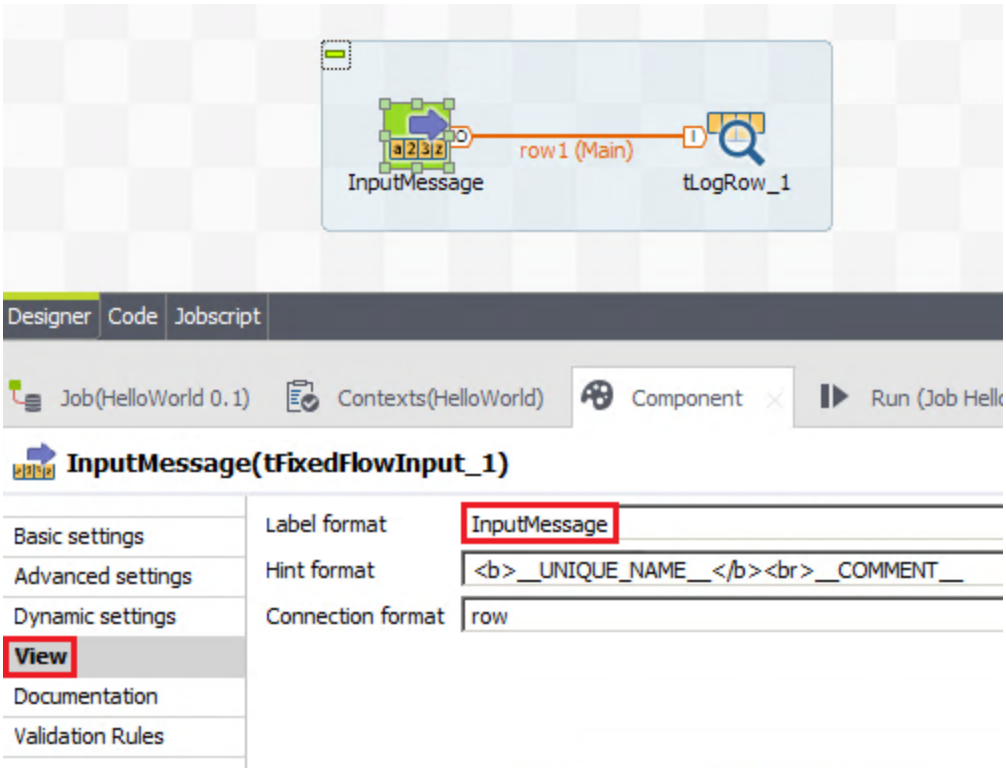
Notice that the **Message** column is added to the schema. Now, set the **Value** of the **Message** Column to *"Hello World"*.

WARNING:
Be sure to include the opening and closing quotation marks.



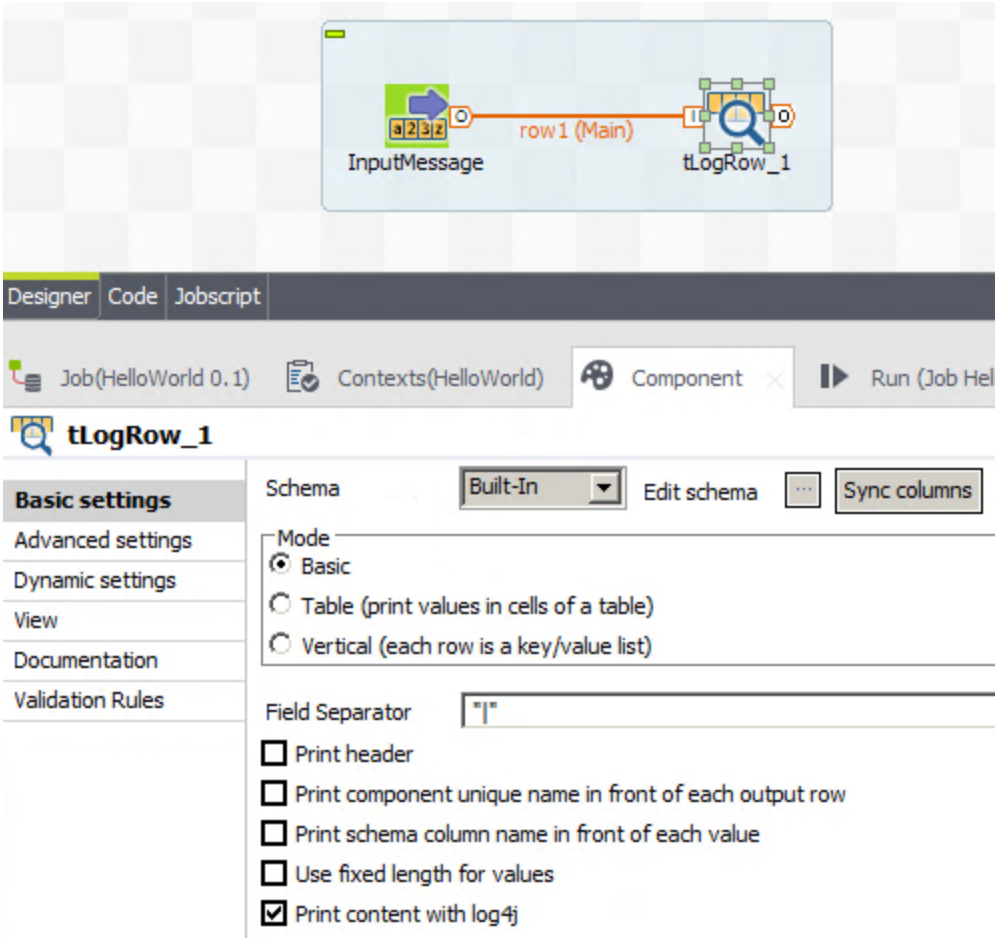
4. NAME THE COMPONENT

   Click the **View** tab in the **Component** view on the left and then replace the text in the **Label format** box with *InputMessage*. This updates the component's name in the **Designer**.



5. CHECK tLogRow SETTINGS

   Double-click **tLogRow_1** and check the settings in the **Components** view:.



For now, leave the settings as they are.

6. SAVE THE JOB

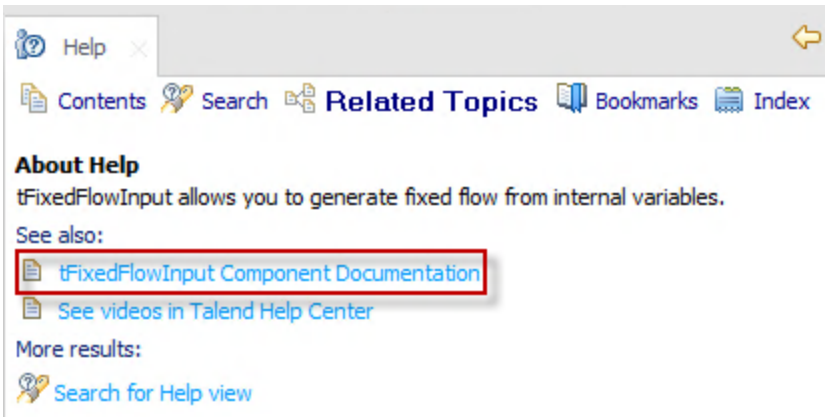
   Note there is an asterisk (*) preceding the name of your Job in the **Designer** tab. This means the Job has changes which have not been saved yet. Click the **Save** button in the main toolbar ( ) and the * disappears.

   TIP:
   You can also press **Ctrl + S** to save the Job.

Help

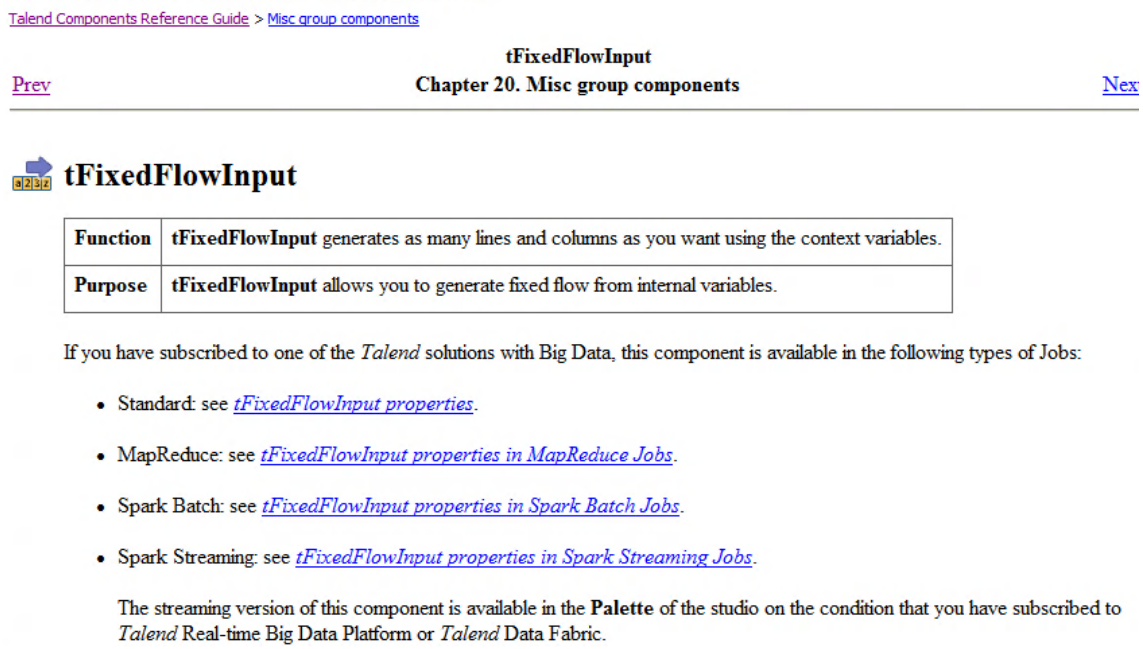
1. OPEN tFixedFlowInput HELP PAGE

   Click on the **InputMessage** component and press the **F1** key. This opens the **Help** page for the selected component. Click on the Component Documentation link highlighted.



2. EXPLORE THE DOCUMENTATION

Double-click on the **Help** tab to enlarge it so it is more readable.



Talend Components Reference Guide > Misc group components

tFixedFlowInput

Prev | Chapter 20. Misc group components | Nex

tFixedFlowInput

Function	**tFixedFlowInput** generates as many lines and columns as you want using the context variables.
Purpose	**tFixedFlowInput** allows you to generate fixed flow from internal variables.

If you have subscribed to one of the *Talend* solutions with Big Data, this component is available in the following types of Jobs:

- Standard: see *tFixedFlowInput properties*.
- MapReduce: see *tFixedFlowInput properties in MapReduce Jobs*.
- Spark Batch: see *tFixedFlowInput properties in Spark Batch Jobs*.
- Spark Streaming: see *tFixedFlowInput properties in Spark Streaming Jobs*.

The streaming version of this component is available in the **Palette** of the studio on the condition that you have subscribed to *Talend* Real-time Big Data Platform or *Talend* Data Fabric.

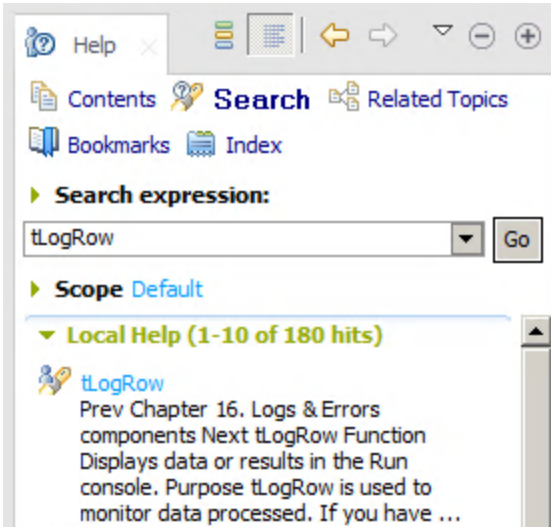
Take a moment to explore the characteristics of the component. Note you can read about different types of Jobs in which the component can be used and you can understand the respective properties. For example, by clicking the **tFixedFlowInput** properties, you will find information about the **Component Family** as well as how to configure the **Schema**.

tFixedFlowInput properties

Component family	Misc	
Basic settings	*Schema* and *Edit Schema*	A schema is a row description, 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 the current schema is of the **Repository** type, three options are available: • **View schema**: choose this option to view the schema only. • **Change to built-in property**: choose this option to change the schema to **Built-in** for local changes. • **Update repository connection**: choose this option to change the schema stored in the repository and decide whether to propagate the changes to all the Jobs upon completion. If you just want to propagate the changes to the current Job, you can select **No** upon completion and choose this schema metadata again in the **[Repository Content]** window.
		Built-in: The schema will be created and stored locally for this component only. Related topic: see *Talend Studio User Guide*.

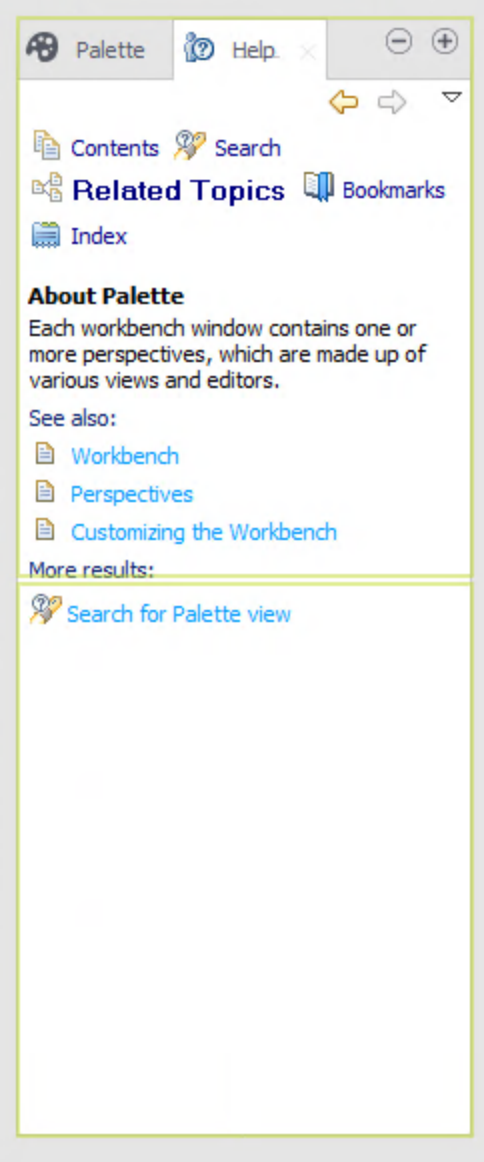
3. OPEN DOCUMENTATION FOR tLogRow

   Click on the **Search** button and find the documentation for the **tLogRow** component.



4. RESTORE THE SIZE OF THE HELP PAGE

   Double-click the **Help** tab to reset the size of the page. You can also drag and drop it to any section of the main window to rearrange the layout. These principles can be applied to any views to customize the layout.



TIP:
If you want to get back to the initial layout, select **Window > Reset Perspective** from the main menu bar.

Next

You have now created your first Talend Data Integration Job. It's now time to Run it .

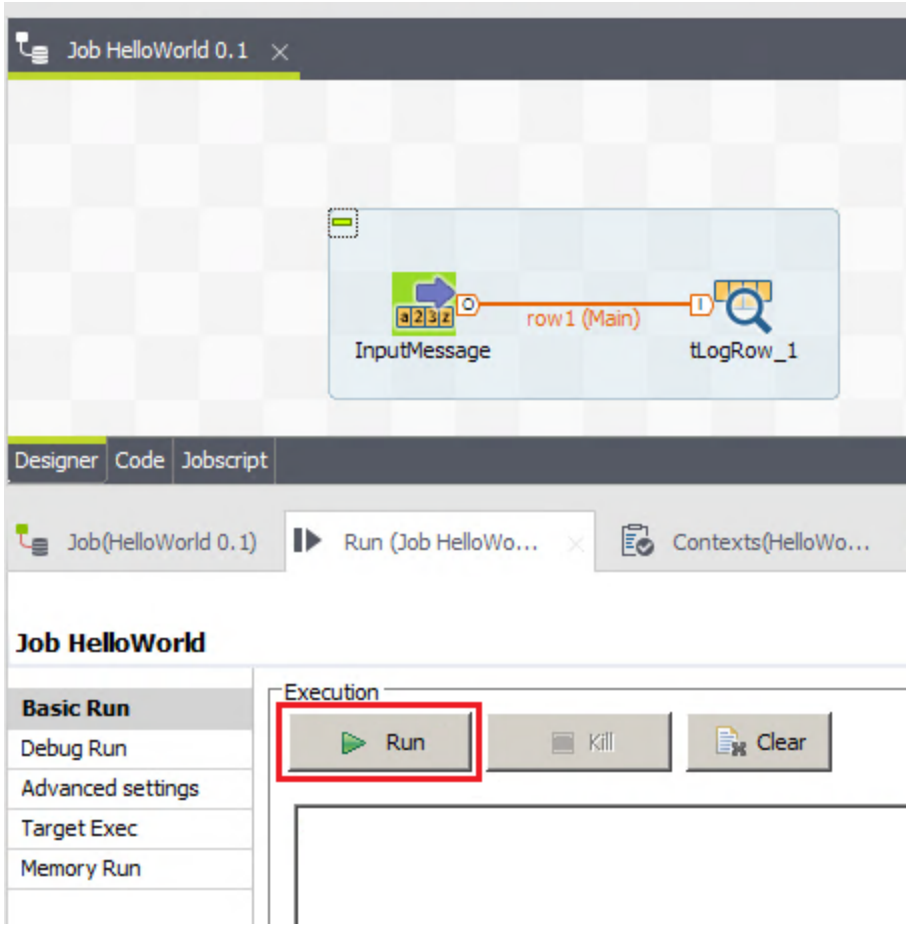
Running a Job

Overview

Running a Job

1. OPEN THE RUN VIEW

   To run the Job, open the **Run** view (near the **Component** tab in the lower portion of the window) and click **Run**.



TIP:

You can also use the **Run** icon ( ) in the main toolbar to run the Job.

2. REVIEW JOB OUTPUT

   Talend Studio saves the Job, builds it, and then runs it. This may take a few moments. Once complete, the Job runs and finishes quickly, displaying messages in the **Run** view.

```
Starting job HelloWorld at 08:41 10/02/2016.

[statistics] connecting to socket on port 3909
[statistics] connected
Hello World
[statistics] disconnected
Job HelloWorld ended at 08:41 10/02/2016. [exit code=0]
```

You can see the *Hello World* message is then displayed in the execution console.

Next

Congratulations! You have now built and run your first Talend Data Integration Job. It's now time to Wrap-Up.

Wrap-Up

In this lesson, you covered the basics required to build data integration Jobs in the Talend Studio.

You started the **Talend Data Fabric** Studio and you explored different views, including the **Repository**, the **Designer**, the **Palette**, and the **Component** view for configuring components.

You then built your first Job using two components. You experienced different methods to add and connect the components and you learned how to configure them. You used the **Palette** to search for the components and you opened the component's documentation from the **Help** page.

While building your first Job you learned that **tFixedFlowInput** component is used to assign fixed flow for internal variables and that the **tLogRow** component is used to monitor the processed data.

You will have the chance to use other components and create other Jobs in the following lessons.

Next step

Congratulations! You have successfully completed this lesson. To save your progress, click **Check your status with this unit** below. To go to the next lesson, on the next screen, click **Completed. Let's continue >**.

Intentionally blank

LESSON 2

Working with Files

This chapter discusses:

Working with Files

Lesson Overview

In this lesson you will start working with files.

First you will read a delimited file storing customer information from your local file system. Then you will use a data transformation component to apply a basic change to the data: transform the customer's state name to upper case. In order to test your transformation, you will write the transformed data to an output file.

Finally, you will consolidate your knowledge of data transformation by combining the first name and last name columns into one column containing the full name of the customer.

Here is high-level view of the Job you will build in this lesson.

Objectives

After completing this lesson, you will be able to:

- Read and write a delimited text file
- Read warning and error messages
- Transform data using the **tMap** component
- Build simple expressions
- Explore data using the data viewer in the Studio
- Duplicate an existing Job as the basis for a new Job

Next Step

The first step is to create a new Job that reads a delimited file

Reading an Input File

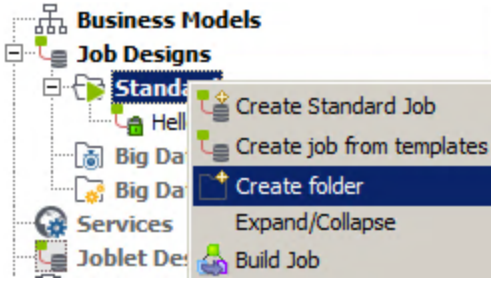
Overview

In this exercise, you will create a new Job that reads data from a comma-separated-value (CSV) file containing customer data. The Job will be created in a specific folder that will contain all Jobs related to the Customer use case.

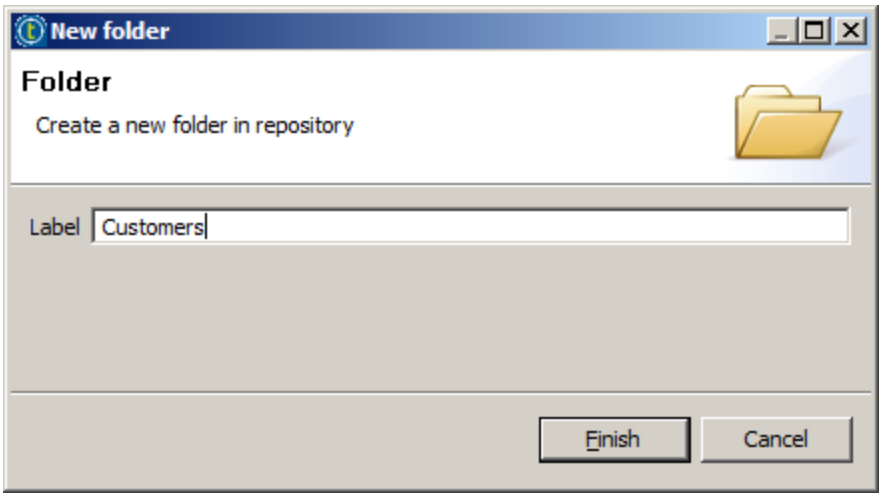
Create a New Job

1. CREATE A FOLDER FOR NEW JOBS

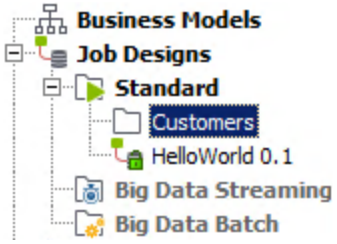
   Create a new folder for your Jobs. Right-click on **Repository > Job Designs > Standard**, then select **Create folder**.



   Enter *Customers* for the **Label**, then click **Finish**.

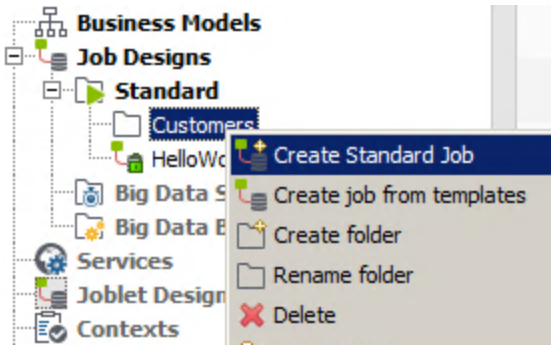


   The new folder appears under **Repository > Job Designs > Standard**.

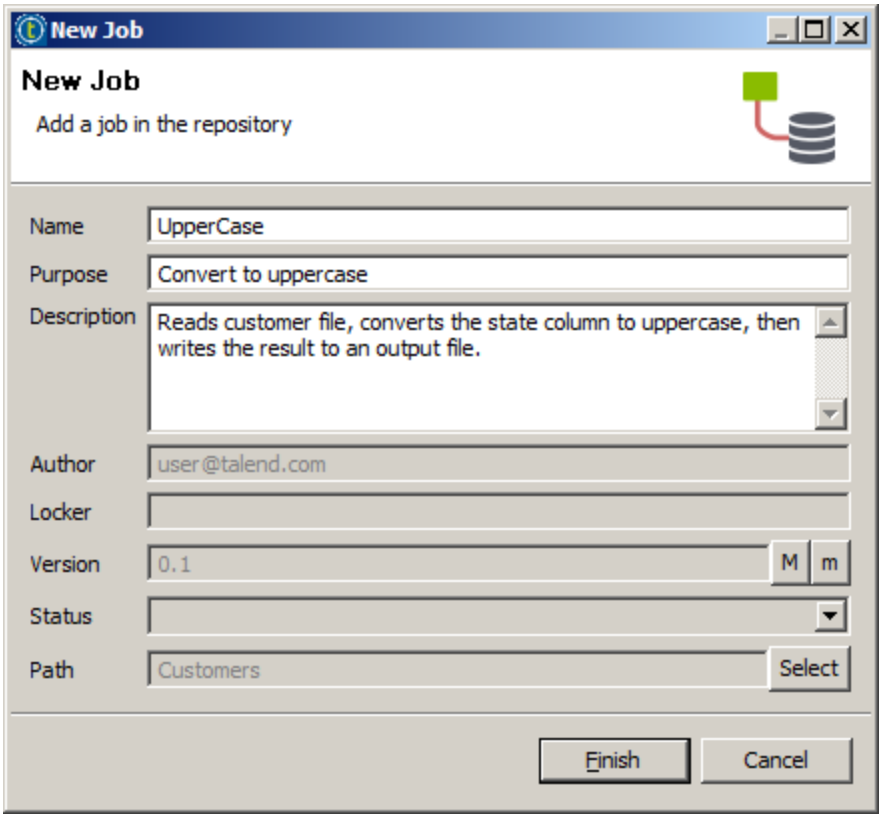


2. CREATE A NEW JOB

   Right-click on **Repository > Job Designs > Standard > Customers**, then select **Create Standard Job**.



Enter *UpperCase* into the **Name** box, then fill in the **Purpose** and **Description** boxes as shown in the figure below. When done, click **Finish**.



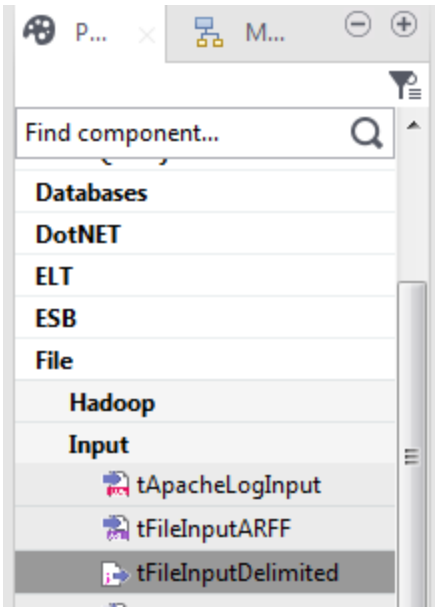
The new Job opens in the **Designer**.

TIP:
When creating a new Job, only the name is mandatory, but it is good practice to complete the remaining fields to help document the Job.

Add a File Reader

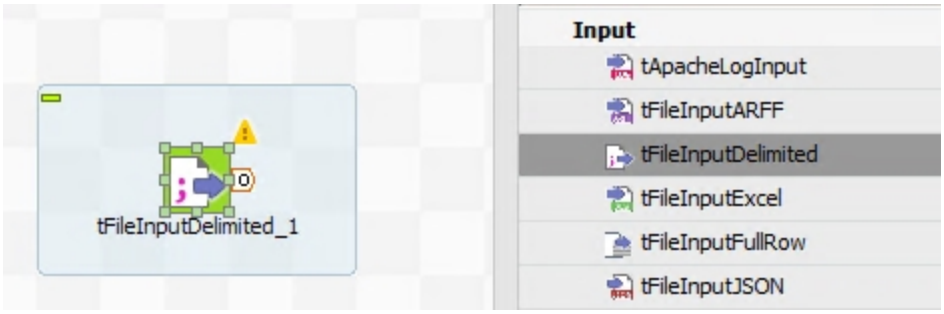
1. ADD A COMPONENT TO READ A CSV FILE

   In the **Palette**, click **File > Input**, then scroll down until you locate the component called **tFileInputDelimited**.



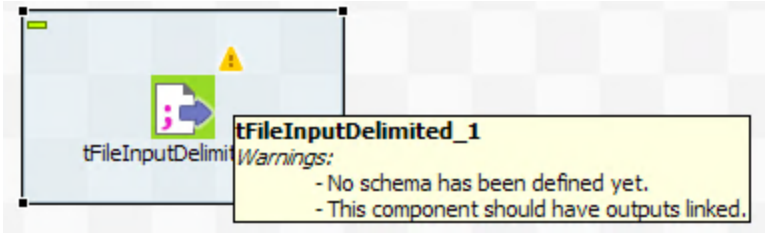
NOTE:
Components in the **Palette** are grouped by function. This particular component reads input from a delimited file.

Select **tFileInputDelimited** in the **Palette**, and then click in the **Designer** to place the component.



2. CHECK THE WARNINGS

Notice the marking above the new component, indicating a warning. Hover the mouse over the component to read the warnings.



The warnings indicate that the component requires a schema, and should be attached to another component.

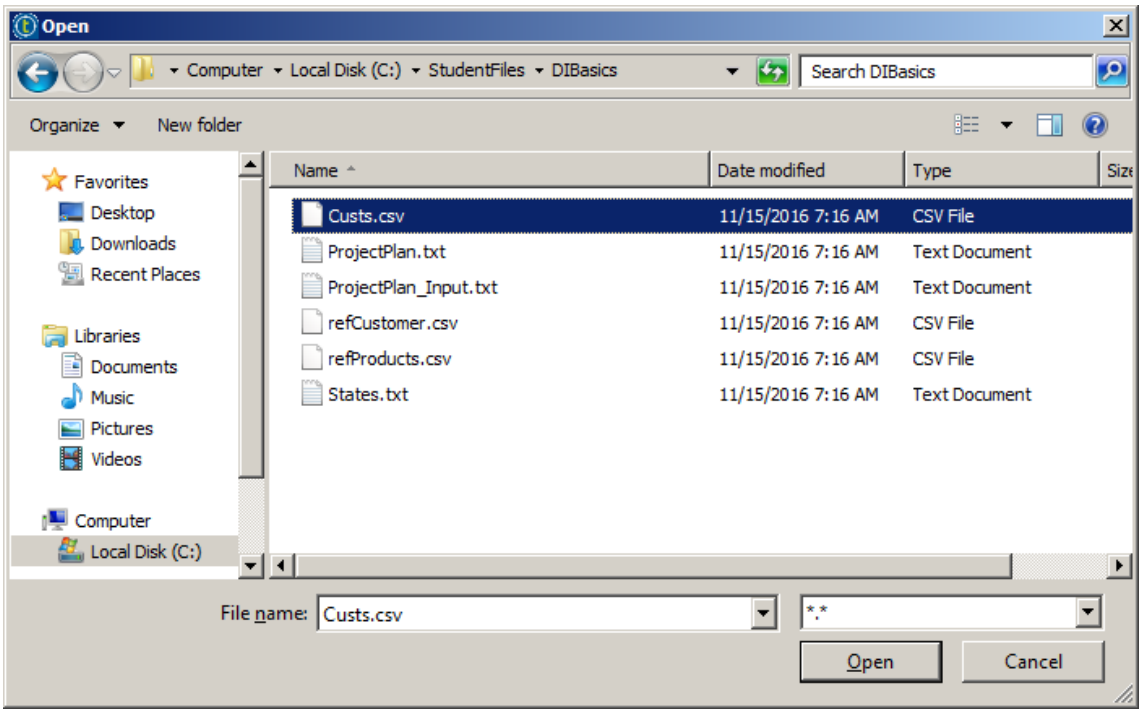
Configure the File Reader

1. CONFIGURE THE INPUT FILE

   Double-click **tFileInputDelimited_1** to open the **Component** view.

   Click the button marked with an ellipsis **[...]** next to the **File Name / Stream** field to browse for the input file.

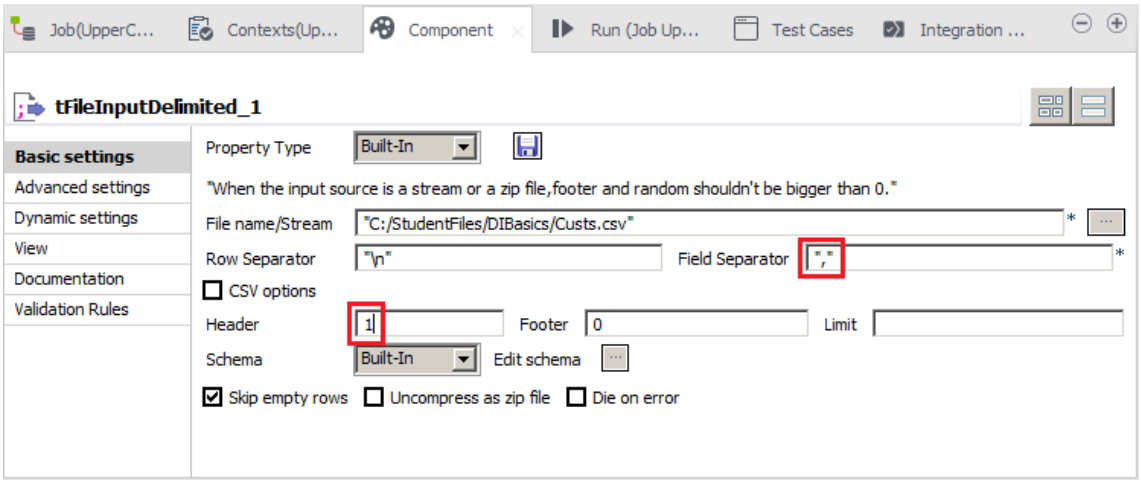
   Locate the file *C:\StudentFiles\DIBasics\Custs.csv* and then click **Open**.



2. CONFIGURE FIELD SEPARATOR AND HEADER PROPERTIES

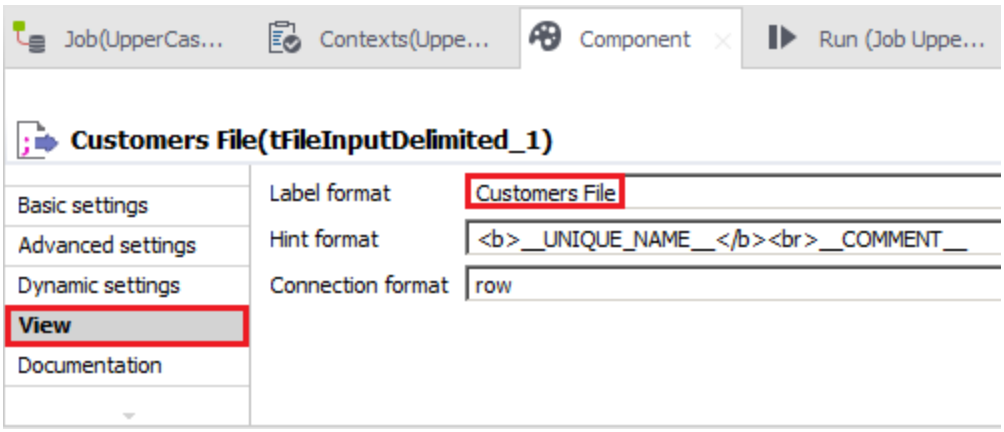
Still in the **Component** view, change the value in the **Field Separator** box to "," to indicate that a comma separates column values in the file, rather than the default semicolon.

Then, change the value in the **Header** box to *1* to specify that the first row of the input file contains column names.



3. LABEL THE COMPONENT

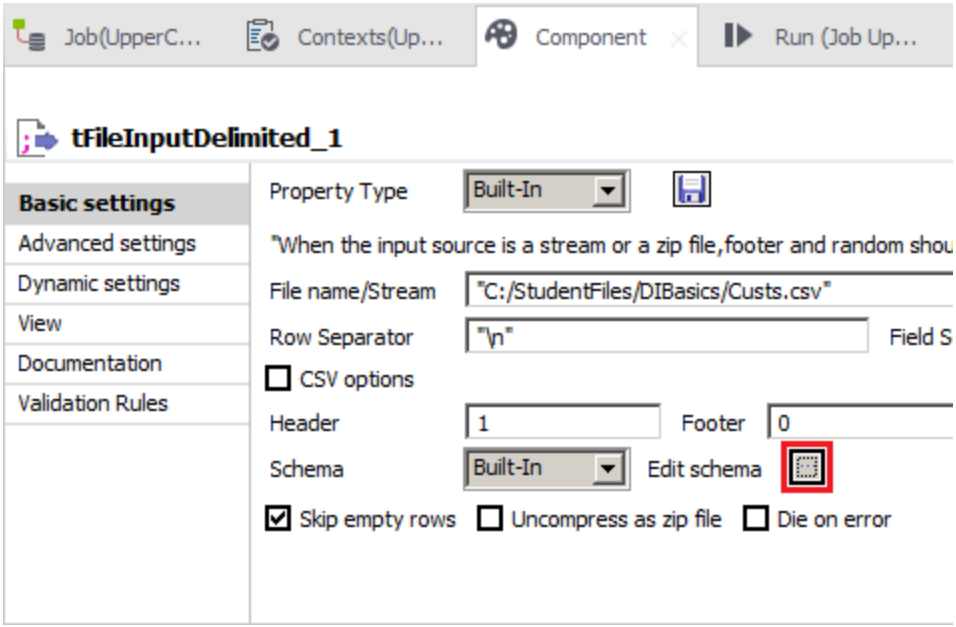
Click the **View** tab, then enter *Customers File* into the **Label format** box.



Defining a Schema

1. OPEN THE SCHEMA EDITOR

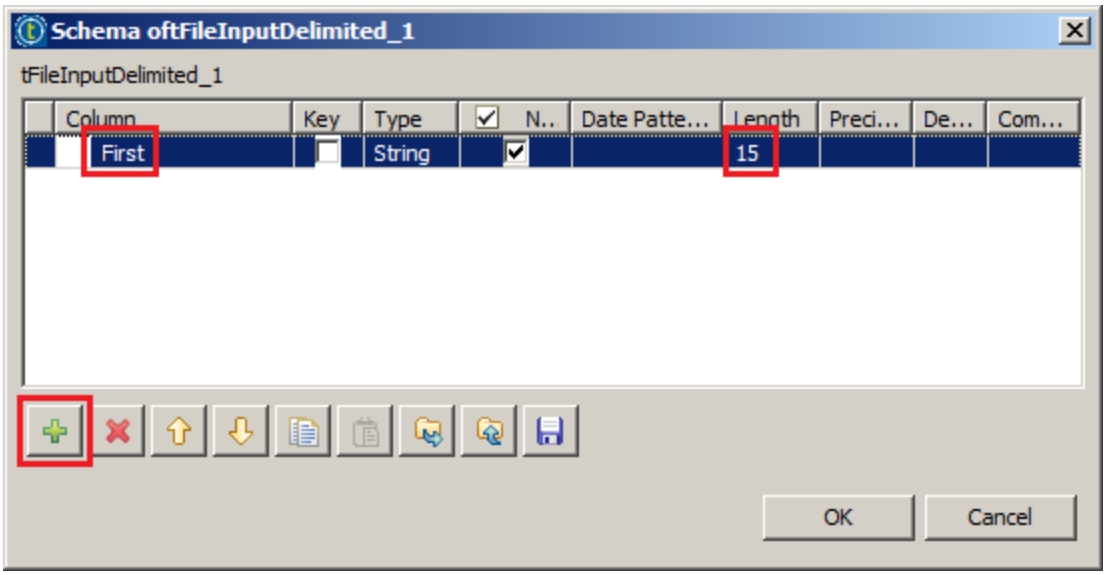
   Click the **Basic settings** tab. Near the bottom of the **Component** view, click the **Edit schema** button marked with an ellipsis **[...]**.



   The **Schema** window appears, where you specify the format of the data from the input file.

2. ADD THE FIRST COLUMN

   Click the **Add** button (+) to add a new column to the schema. Enter *First* for the **Column** name, then enter *15* for the **Length**.



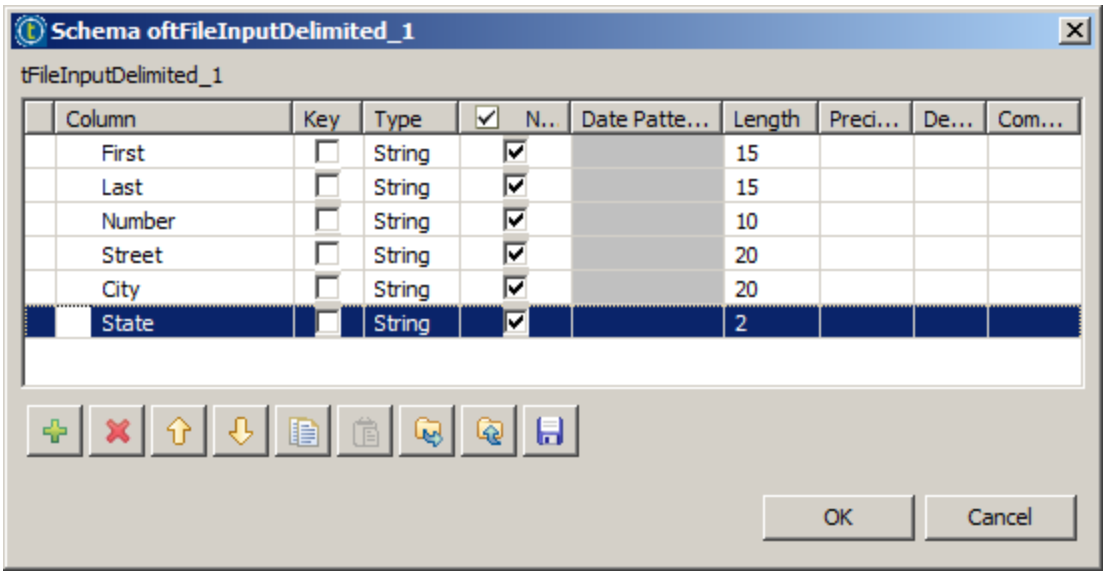
This specifies that the first column of each row of data in the *Custs.csv* file contains a 15-character string value representing the first name of the customer.

3. COMPLETE THE SCHEMA

   Add five more columns of type **String** and set the **Length** as follows:

   - Last, 15
   - Number, 10
   - Street, 20
   - City, 20
   - State, 2

   When finished, your schema should resemble the figure below. Click **OK** when finished.

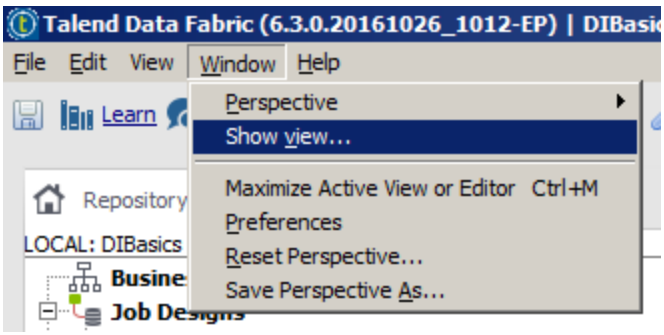


As you can see, the *Custs.csv* file contains basic identification and address information for a group of customers. The information is organized into six columns.

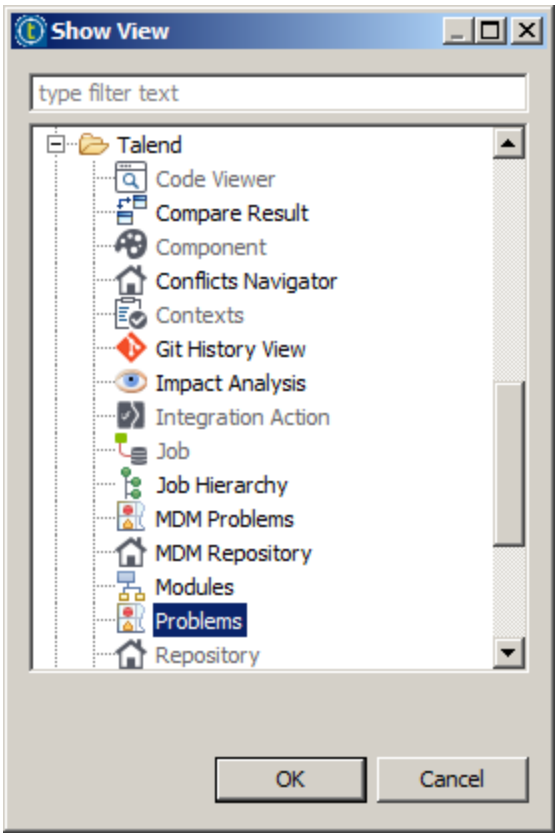
Warnings

1. DISPLAY THE PROBLEMS VIEW

   Select **Window > Show View** from the main menu bar.



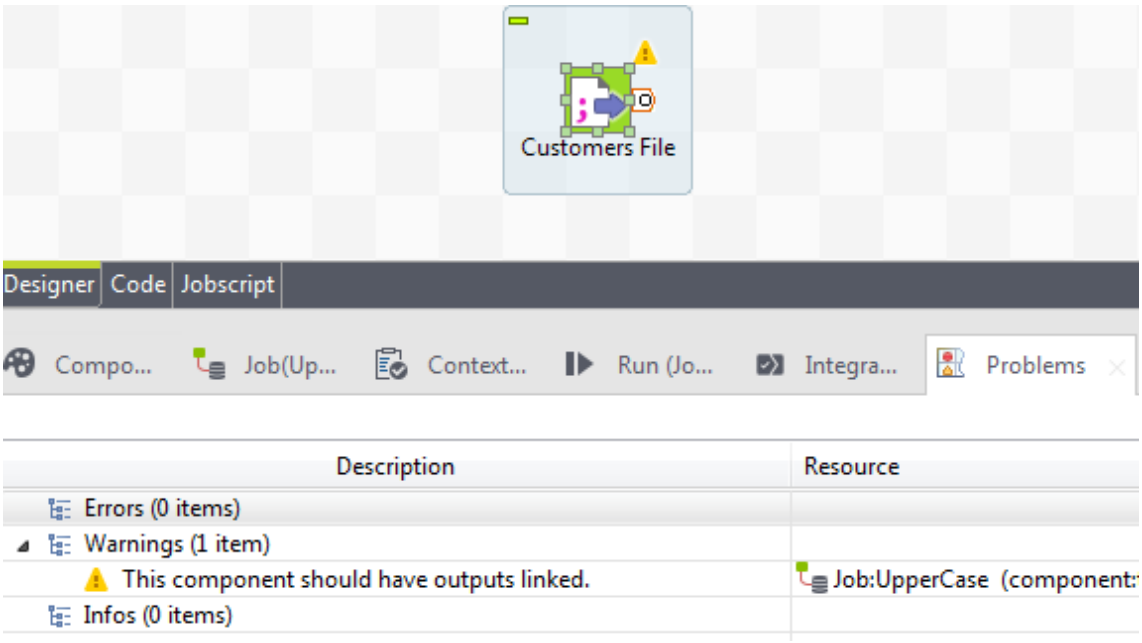
   The **Show View** window appears. This allows you to choose additional views to display in the current perspective. Select **Talend > Problems**, then click **OK**.



   The **Problems** view now appears in the lower portion of the main window.

2. EXPLORE WARNINGS

   In the **Problems** view, expand **Warnings**.



In this case, the warning tells you that your input component needs to be linked to some kind of output. Note that one of the warnings from earlier is no longer displayed because that issue was resolved by configuring a schema.

Next

The next step is to send the data from the input file to a component that performs the capitalization.

Transforming Data

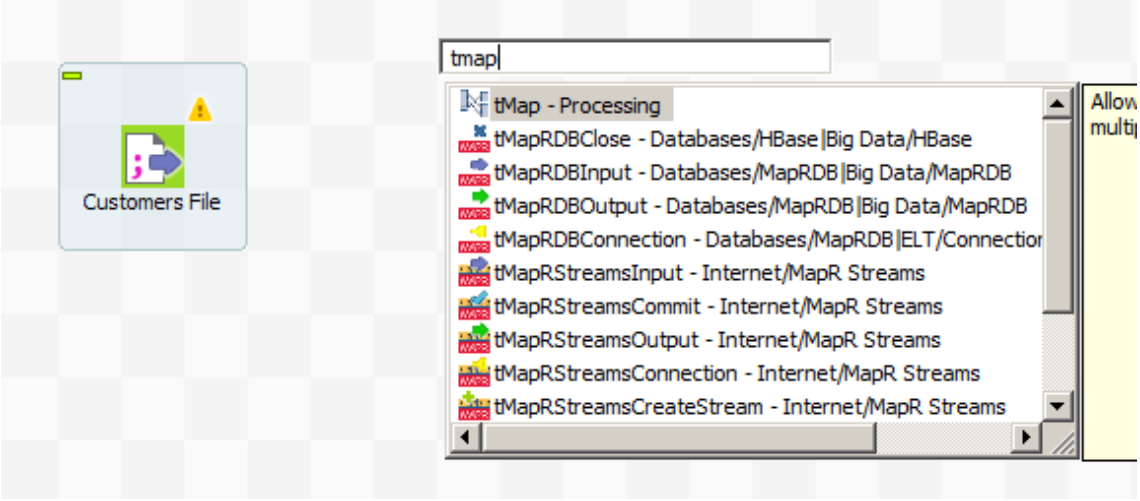
Overview

Your **UpperCase** Job contains a component to provide the input data. Now you need to add a component to perform the data transformation: converting any lower-case state abbreviation to upper case.

Add a Transformation Component

1. FIND THE tMap COMPONENT

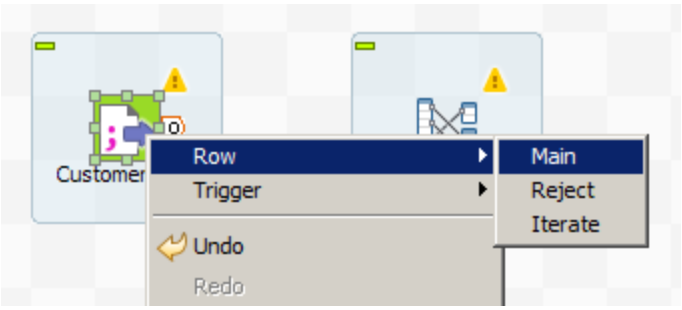
   Click anywhere in the **Designer** to the right of **CustomersFile**. Begin typing the name of the component you wish to add, in this case **tMap**, and a list of matching components will appear. Search the list for the component.



2. PLACE AND CONNECT THE COMPONENT

   Double-click the **tMap** component in the list to plant the component at the selected spot in the **Designer**.

   Now, right-click **Customers File** and select **Row > Main**, then click **tMap_1**. This creates a connection between the two components.



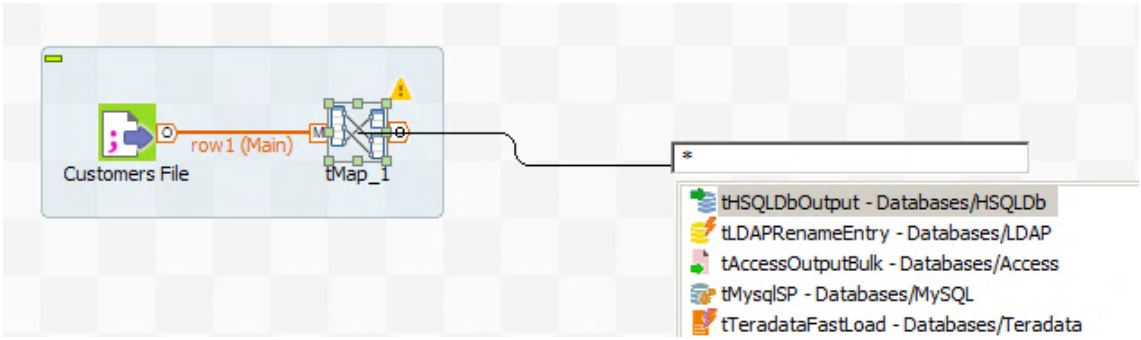
   The **tMap** component maps data from input to output, with the capability to perform a variety of transformations on the data. You will find yourself using it as the centerpiece of almost all of your Talend Data Integration Jobs.

   Note the warning displayed for the **tMap** component. The **tMap** component transforms data, so it needs an output destination.

Add an Output Component

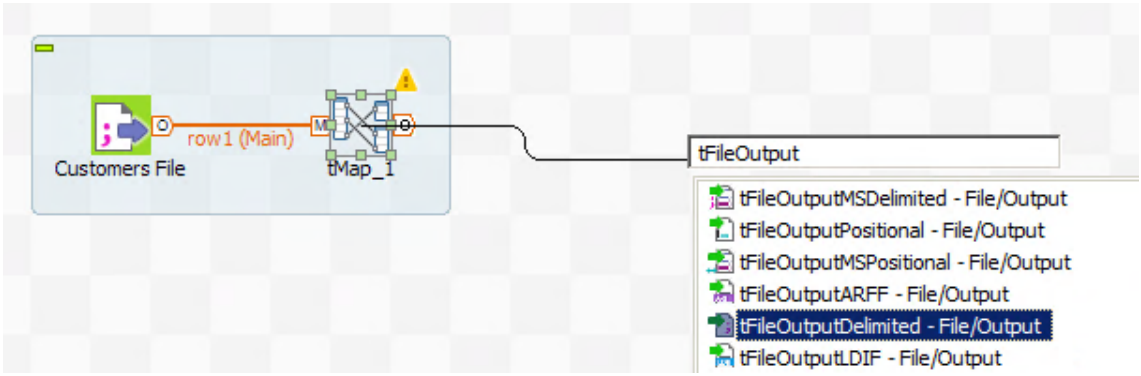
1. CHOOSE AN OUTPUT COMPONENT

   Right-click **tMap_1**. Holding the right-click, move the mouse outside the component.

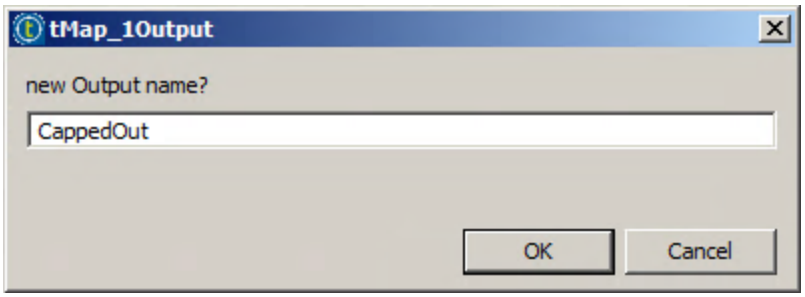


   A list of components are then suggested. Start typing the name of the desired output component, in this case *tFileOutputDelimited*, into the search box. As its name implies, a **tFileOutputDelimited** component writes rows to a file in delimited format.

   Double-click the component to add it to your Job.



   A window appears asking you to supply a name for the output connection. Enter *CappedOut* (no spaces allowed) into the text box, then click **OK**.

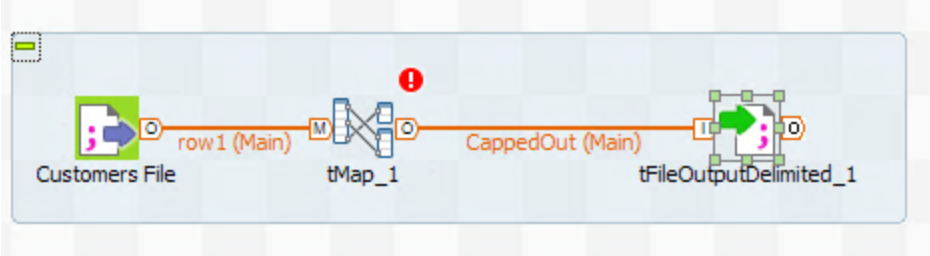


2. INVESTIGATE THE ERROR

   The new output connection is a **Main** row, just like the input connection.

   Hover over the **tMap_1** error or check the **Problems** view to see what the error is. There is an issue with the output

schema, which will get resolved next.

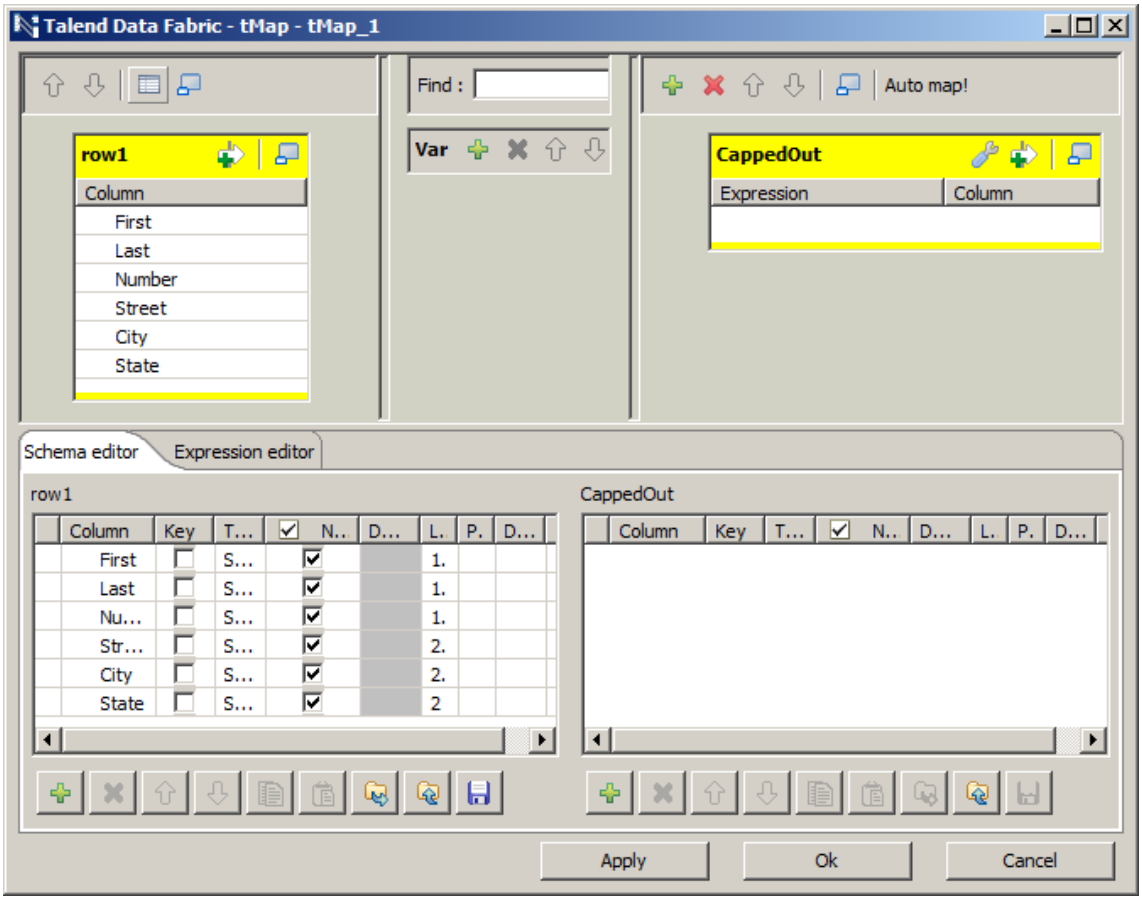


Map Rows

In order to define the output rows and the data transformations to perform, you need to define the output schema for the **tMap** component.

1. OPEN THE MAPPING EDITOR

   Double-click **tMap_1** to open the mapping editor.

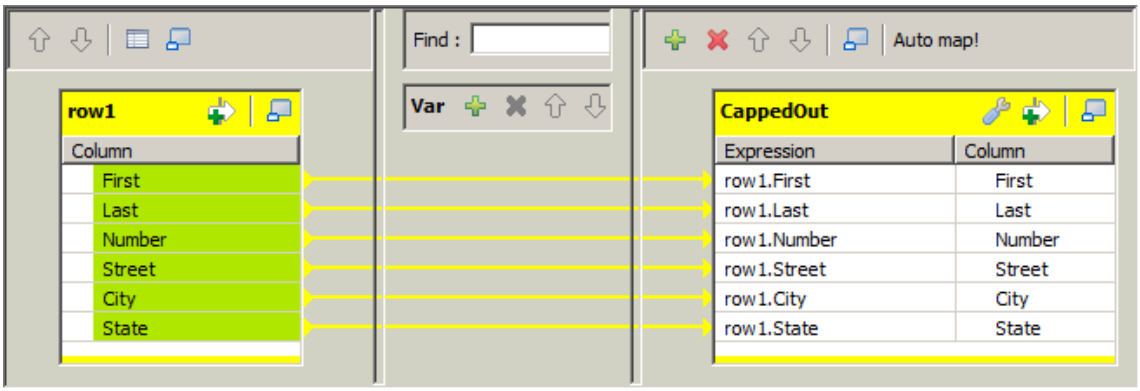


   The **tMap** component provides so much functionality, so this window has many elements. Notice the table on the left, labeled **row1**, and the table on the right labeled **CappedOut**. Those tables represent the schema of the input and output connections, with names that match the rows. The schema for **row1** is copied from the input component and should look familiar. The schema for the new row **CappedOut** is not defined yet.

   The goal of this Job is to pass all the data from **row1** to **CappedOut**, while capitalizing the values in the *State* column.

2. COPY ALL INPUT ROWS

In **row1**, click on the top row and then shift-click on the bottom row, to select all the rows. Then, drag the selected rows from **row1** onto **CappedOut**.

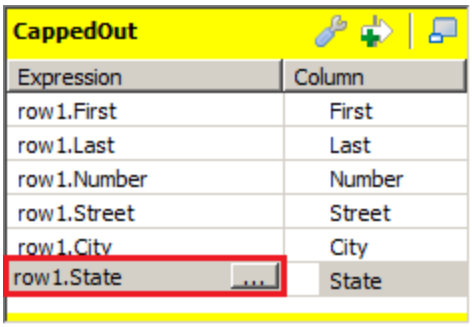


Notice the arrows indicating the mapping of columns from one schema to the other. In the **CappedOut** table, notice the **Expression**, which defines the data content, as well as the column names. At this point, the data and column names are an exact match to those in **row1** (the expression **row1.<name>** specifies the data in the name column of **row1**).

Build an Expression

1. SELECT THE OUTPUT YOU WISH TO MODIFY

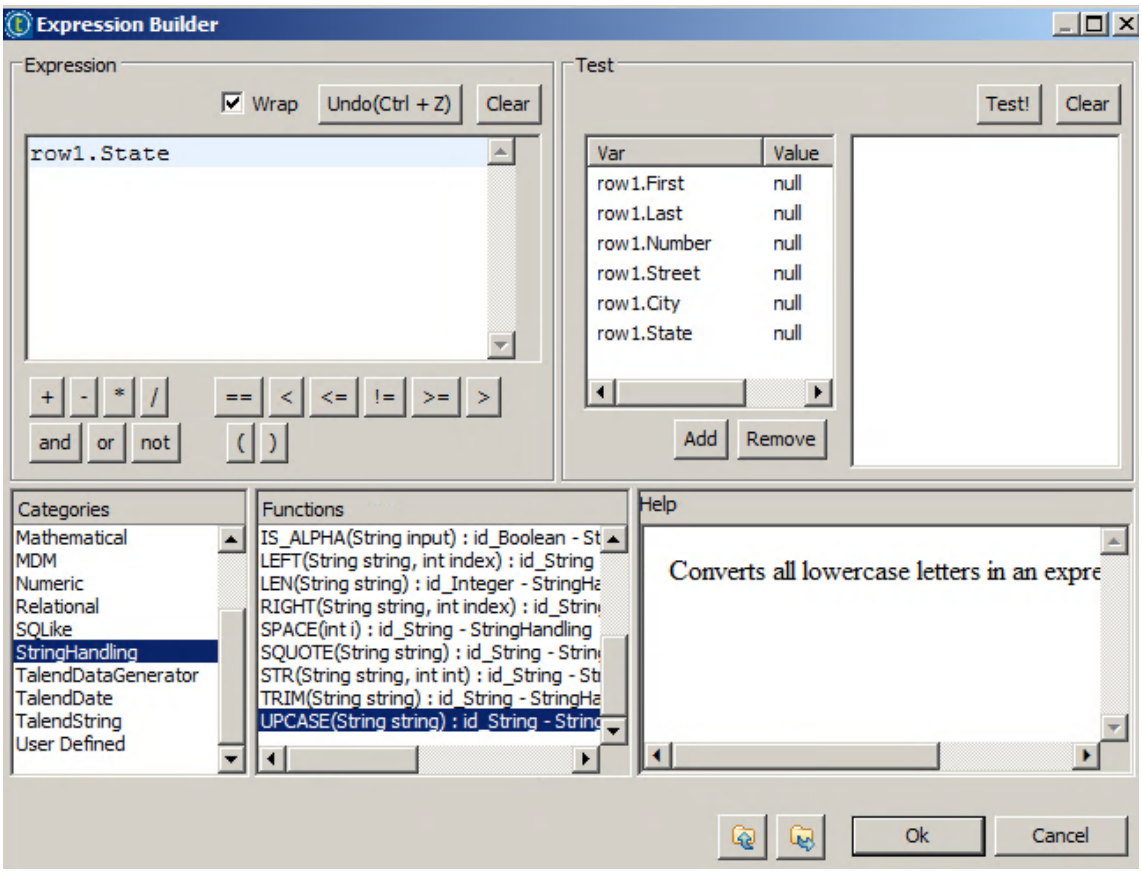
   The goal is to convert the **State** column data to uppercase, so click **row1.State** in the **CappedOut** output table, and then click the button marked with an ellipsis **[...]** that appears.



   The **Expression Builder** opens. This window allows you to construct expressions in Java syntax. You can enter the expression as text yourself, or you can take advantage of the tool to simplify the building of such expressions.

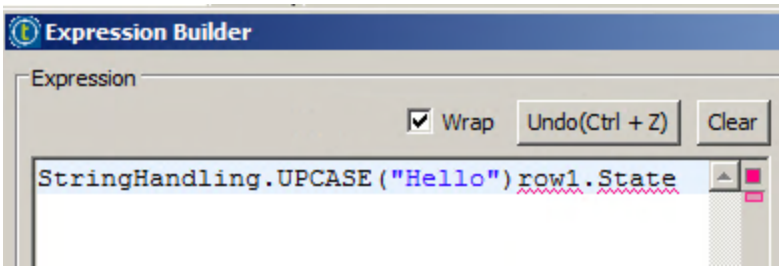
2. BUILD AN EXPRESSION TO CONVERT DATA TO UPPERCASE

   Select **Categories > StringHandling**, then scroll down in the **Functions** list until you find **UPCASE**. Select it. Notice the text that appears in the **Help** area.



3. INSERT THE FUNCTION INVOCATION INTO THE EXPRESSION

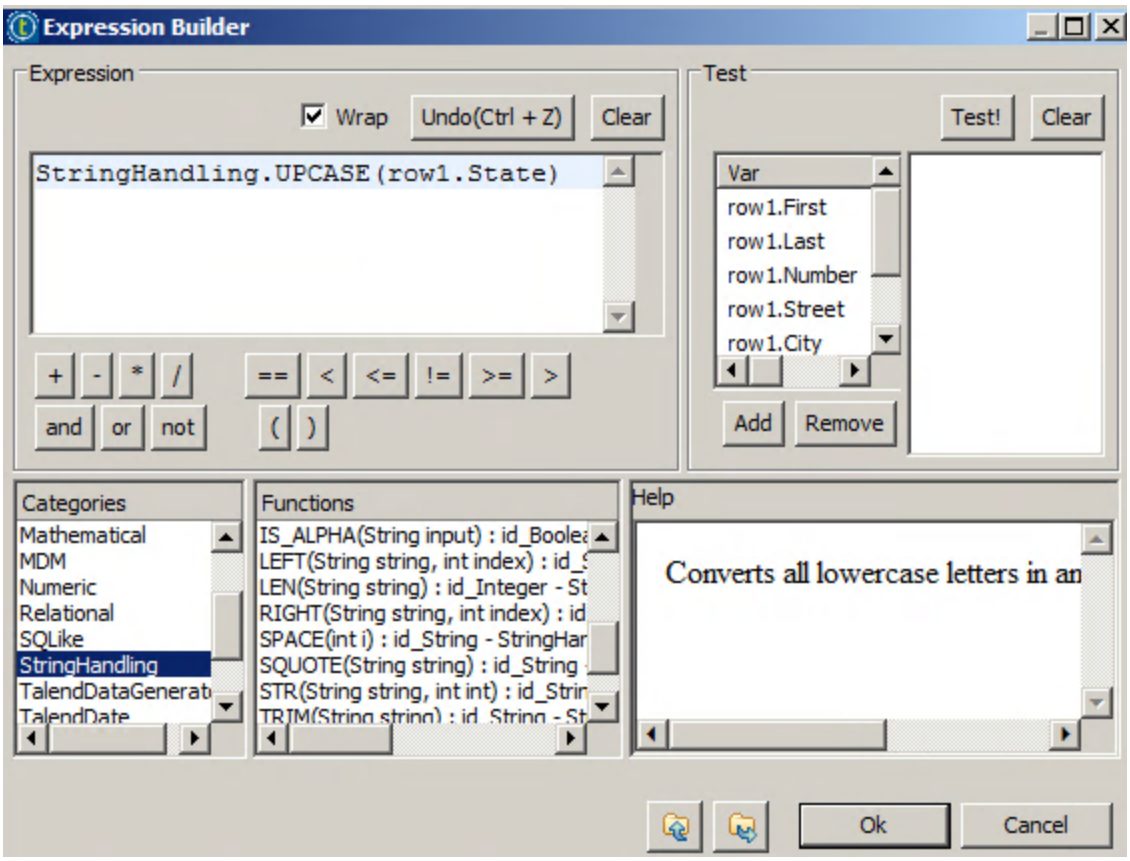
   Double-click **UPCASE** to insert the function into the expression at the insertion point.



   Notice the format of the expression that is inserted. As a default example, the function contains the literal string "Hello". You want to reconfigure this to convert the contents of the **State** column.

4. COMPLETE THE EXPRESSION

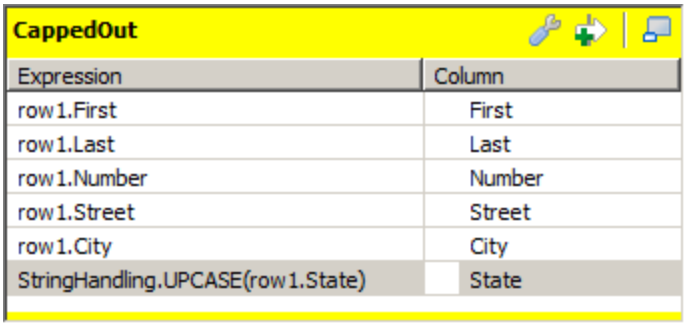
   Correct the expression by moving *row1.State* in between the parentheses so that it replaces "Hello". Once your expression is entered as shown below, click **Ok**.



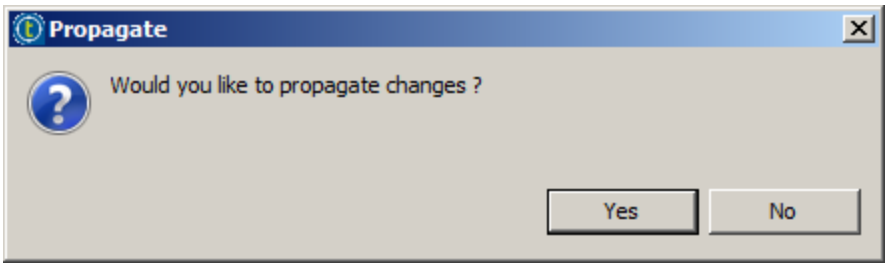
WARNING:
This expression operates on a variable rather than a string literal, so make sure you delete the enclosing quotation marks in this case.

5. APPLY THE CHANGES

The new expression appears in the **CappedOut** table.



Now the output column **State** will contain the string from the column of the same name in the **row1** table, converted to upper case. You are finished with your **tMap** component configuration, so click **Ok**. Since you have made changes to the schema of the output connection, you are prompted to propagate the changes (in other words, reflect those changes in the schema of the component on the other end of the connection). Click **Yes**.

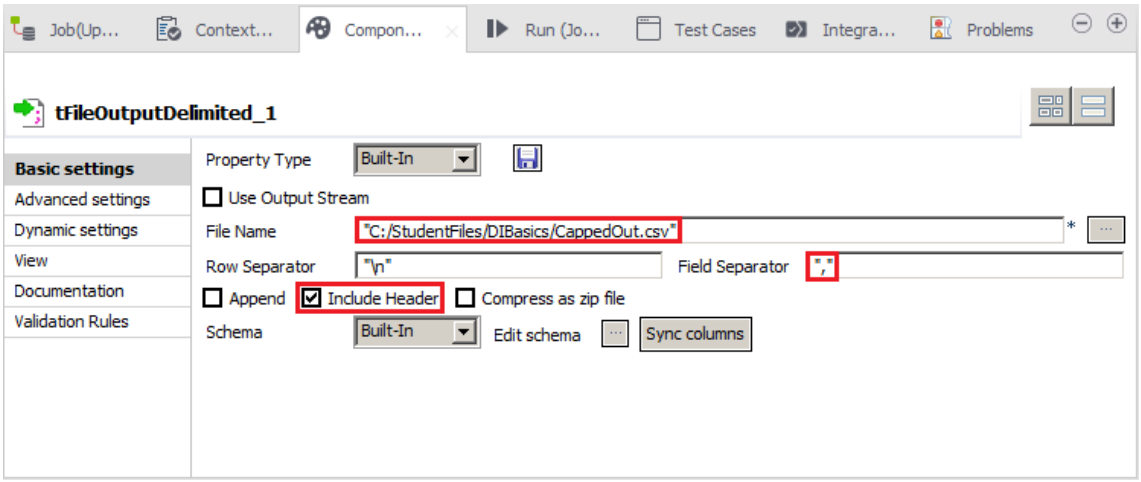


Configure Output

1. CONFIGURE THE OUTPUT COMPONENT

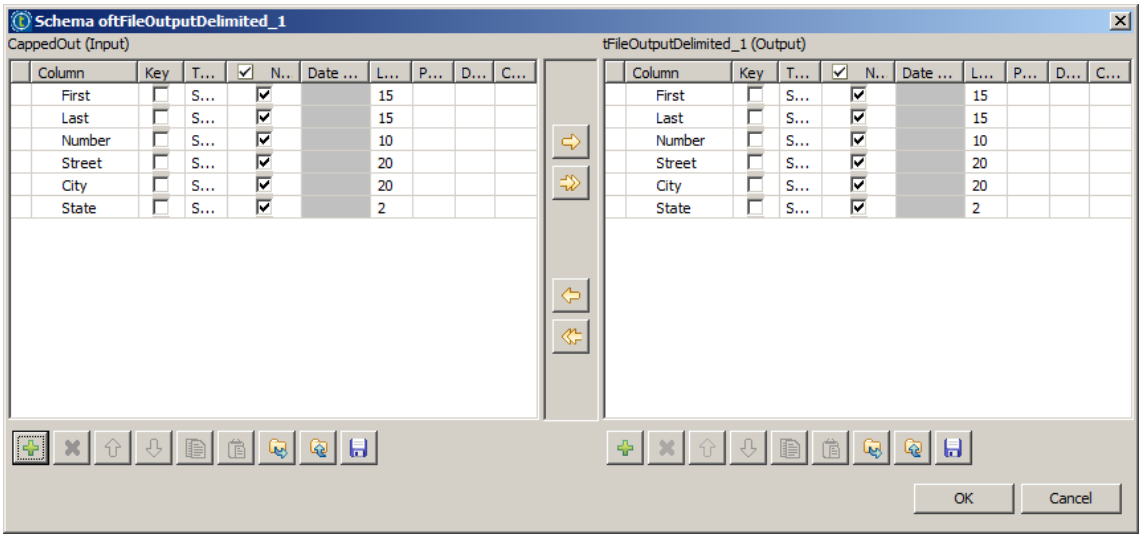
   Double-click **tFileOutputDelimited_1** to open the **Component** view.

   Change the name of the output file to *C:/StudentFiles/DIBasics/CappedOut.csv*. Change the **Field Separator** to a comma, and select the **Include Header** check box to include the column names as the first row of the output file.



2. CHECK THE SCHEMA

   Click the **Edit schema [...]**(ellipsis) button in the **Component** view.



Note that the schema of the component matches the schema you specified in the **tMap** component, which matches the schema of the input connection (**CappedOut**).

There is no need to make any changes here, so click **OK**.

Next

Your Job is complete and you are now ready to run it.

Running a Job

Overview

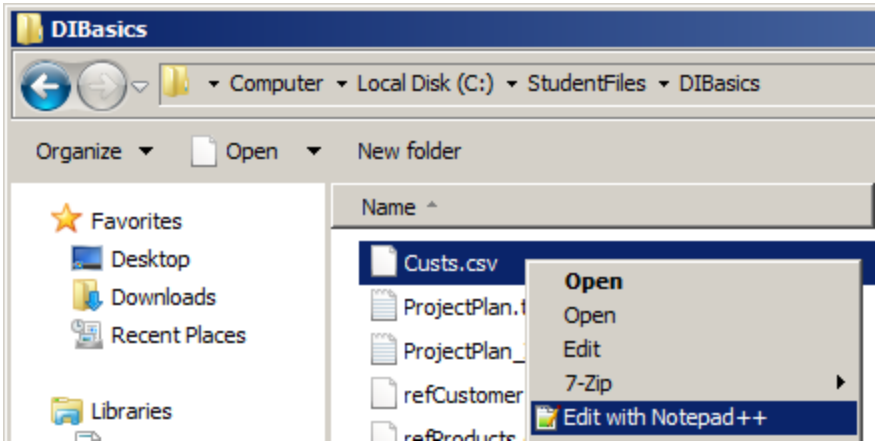
> The construction of your Job is complete, and now you are ready to run it and see the results.

Examine Input File

Before running your Job, examine the input file.

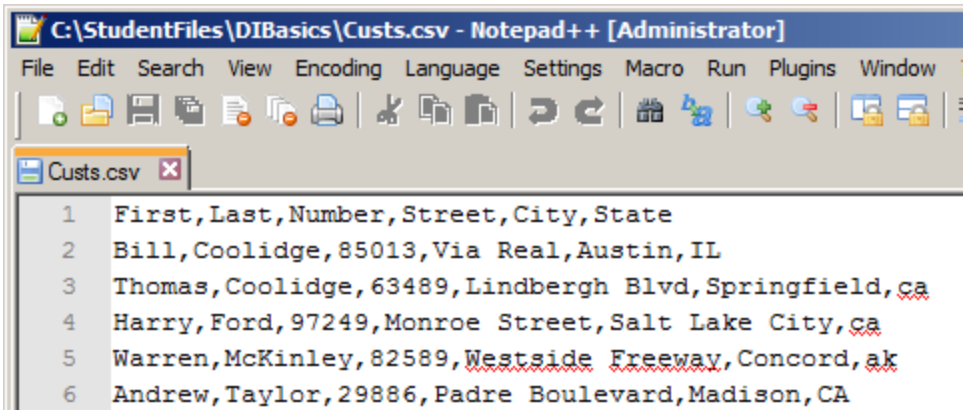
1. OPEN THE FILE IN A TEXT EDITOR

   Use the file browser to locate the file *C:\StudentFiles\DIBasics\Custs.csv*. Right-click on the file and select **Edit with Notepad++**.



2. EXAMINE FILE CONTENTS

   Examine the contents of the file. Notice that several states are in lowercase. Your Job is designed to change all states to uppercase.



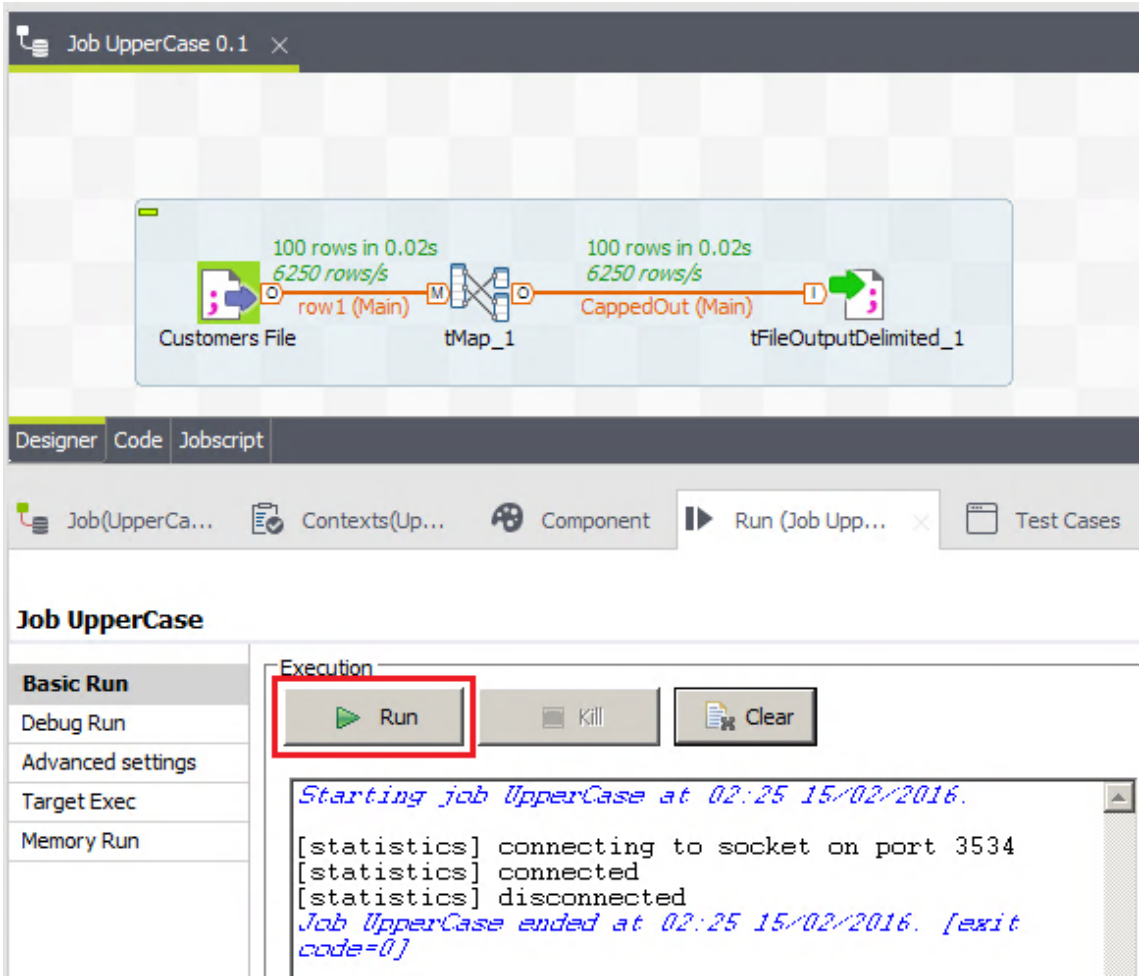
   When done, close the editor.

Run the Job

1. RUN THE JOB

   Click the **Run** tab under the design space to expose the **Run** view. Click the **Run** button.

Note that statistics about the Job execution display in the design workspace.



This shows that 100 rows passed through each of the connections.

2. CHECK THE OUTPUT FILE

Locate and open the output file *C:\StudentFiles\DIBasics\CappedOut.csv*. Notice that all state abbreviations are now in uppercase.

```
First,Last,Number,Street,City,State
Bill,Coolidge,85013,Via Real,Austin,IL
Thomas,Coolidge,63489,Lindbergh Blvd,Springfield,CA
Harry,Ford,97249,Monroe Street,Salt Lake City,CA
Warren,McKinley,82589,Westside Freeway,Concord,AK
Andrew,Taylor,29886,Padre Boulevard,Madison,CA
Ulysses,Coolidge,98646,Bayshore Freeway,Columbus,MN
Theodore,Clinton,12292,San Marcos,Bismarck,NY
Benjamin,Jefferson,82077,Carpinteria North,Sacramento,CA
William,Van Buren,21712,Tully Road East,Albany,IL
Calvin,Washington,50742,Richmond Hill,Charleston,CA
Jimmy,Polk,76143,Richmond Hill,Salt Lake City,AK
```

Next

As mentioned earlier, the **tMap** component provides a great deal of transformation functionality, so your next step is to explore some different kinds of transformations.

Combining Columns

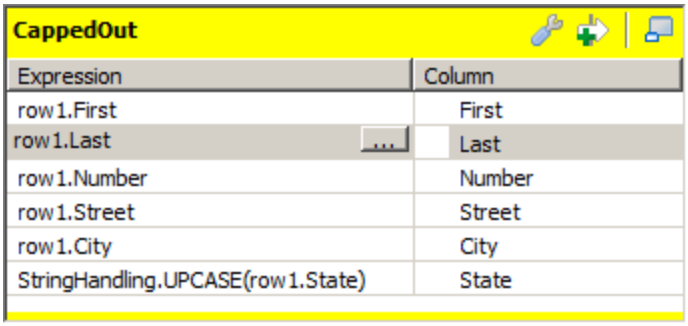
Overview

The **tMap** component provides the capability to perform a variety of data mapping and transformation functions. Imagine that your destination data store requires the first and last names to be combined in a single column. Follow the steps below to achieve that.

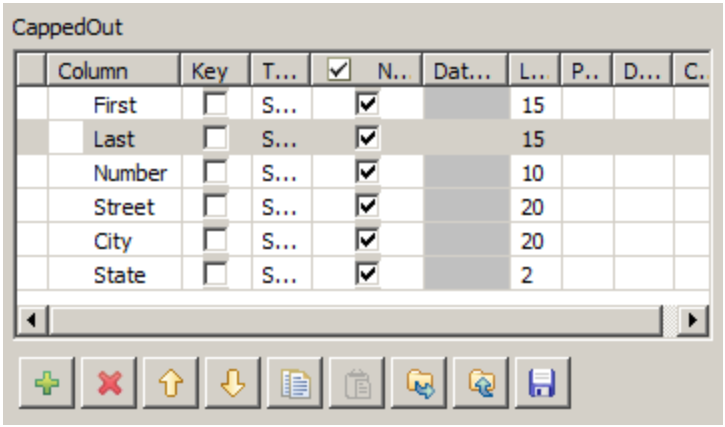
Edit the Schema

1. OPEN THE MAPPING EDITOR

   Double-click the **tMap** component in the **Designer**. Click the **row1.Last** expression in the *CappedOut* table.



   At the bottom of the **Schema editor** window, notice that the same column is selected.

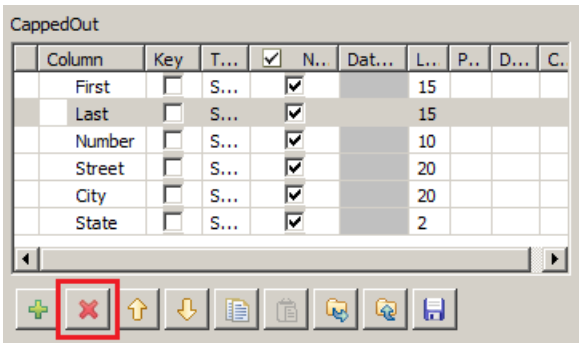


   You are going to combine the data from two columns into a single column, so this column is no longer necessary.

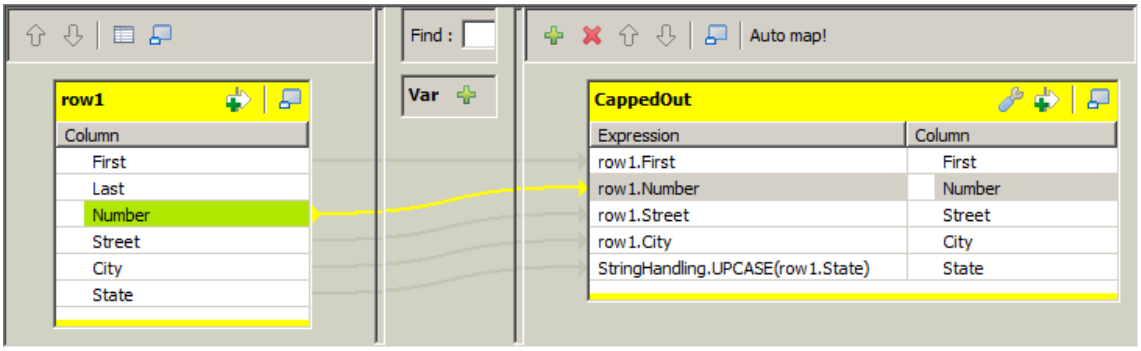
2. DELETE THE COLUMN

   Click the **Remove selected items** button (), to delete the **Last** column from the output schema.

   WARNING:
   Make sure to click **Remove selected items** in the schema editor of *CappedOut* (lower right part of the map editor) and not the **Remove table** in the upper right part of the map editor. If you click **Remove table**, you will delete your mapping and will have to rebuild it from the beginning.



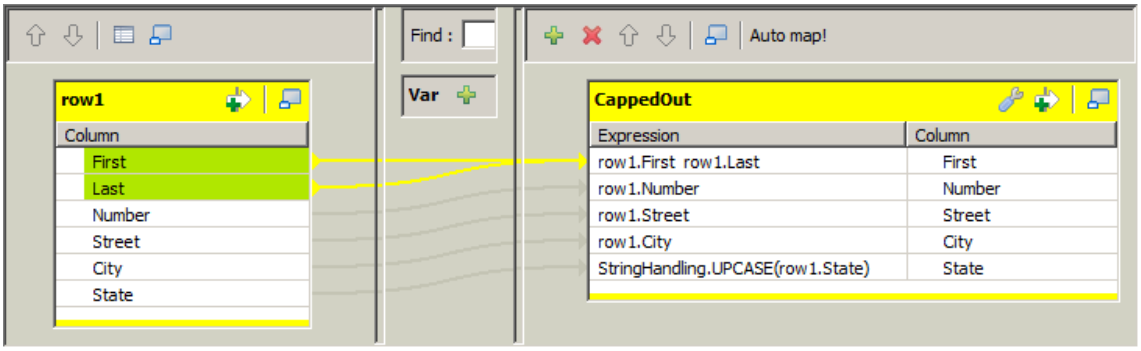
The column disappears from the *CappedOut* table at the top of the window as well. Notice that there is no longer a mapping present for the *Last* column in the *row1* table.



Combine Columns

1. MAP FIRST AND LAST NAMES TO ONE OUTPUT COLUMN

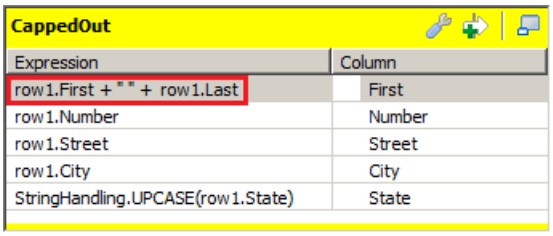
   Drag the column **Last** from the **row1** table on the left and drop it onto **First** in the **CappedOut** table.



   Notice that the expression now contains references to two columns from the **row1** table.

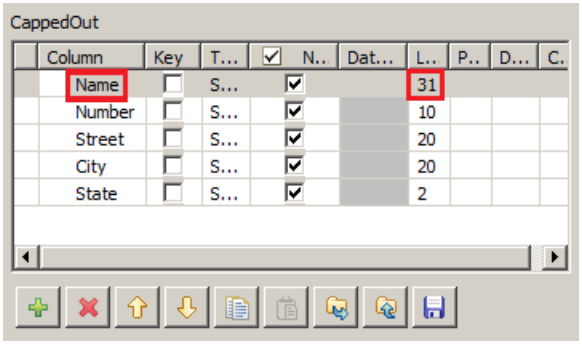
2. CORRECT THE EXPRESSION

   Insert a plus sign (+) between the two references. This is required to concatenate two strings together. Additionally, insert a space character to separate the two names in the output. In summary, the expression should read `row1.First + " " + row1.Last` when done.



3. UPATE COLUMN NAME AND LENGTH

   In the **Schema editor** at the bottom of the window, click *First* in the *CappedOut* table and then change it to *Name*. Also, update the length of the that field from 15 to 31.



   Now the schema column name in the **CappedOut** table is **Name** rather than **First**, allowing for a length of 31 characters to support the concatenation of the first and last names.

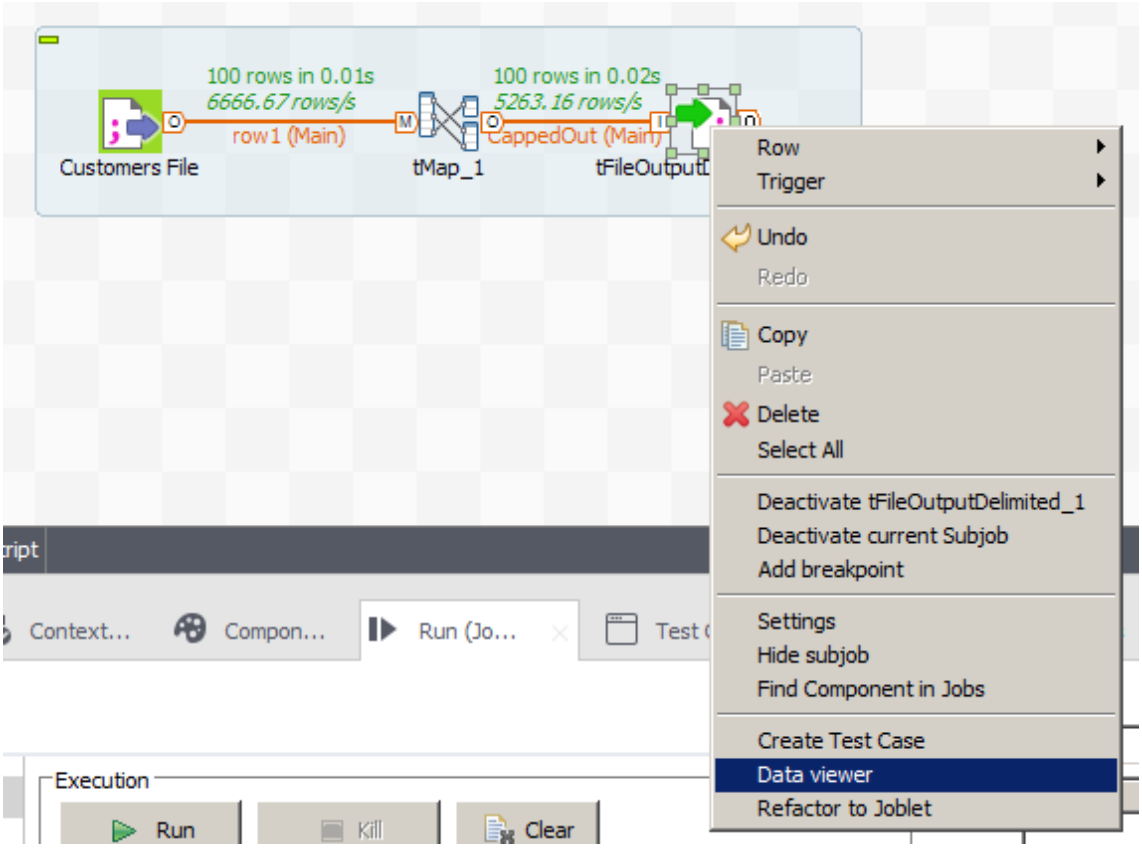
4. SAVE CHANGES

   Click **Ok** to save your changes, and then propagate the changes when prompted.

Run

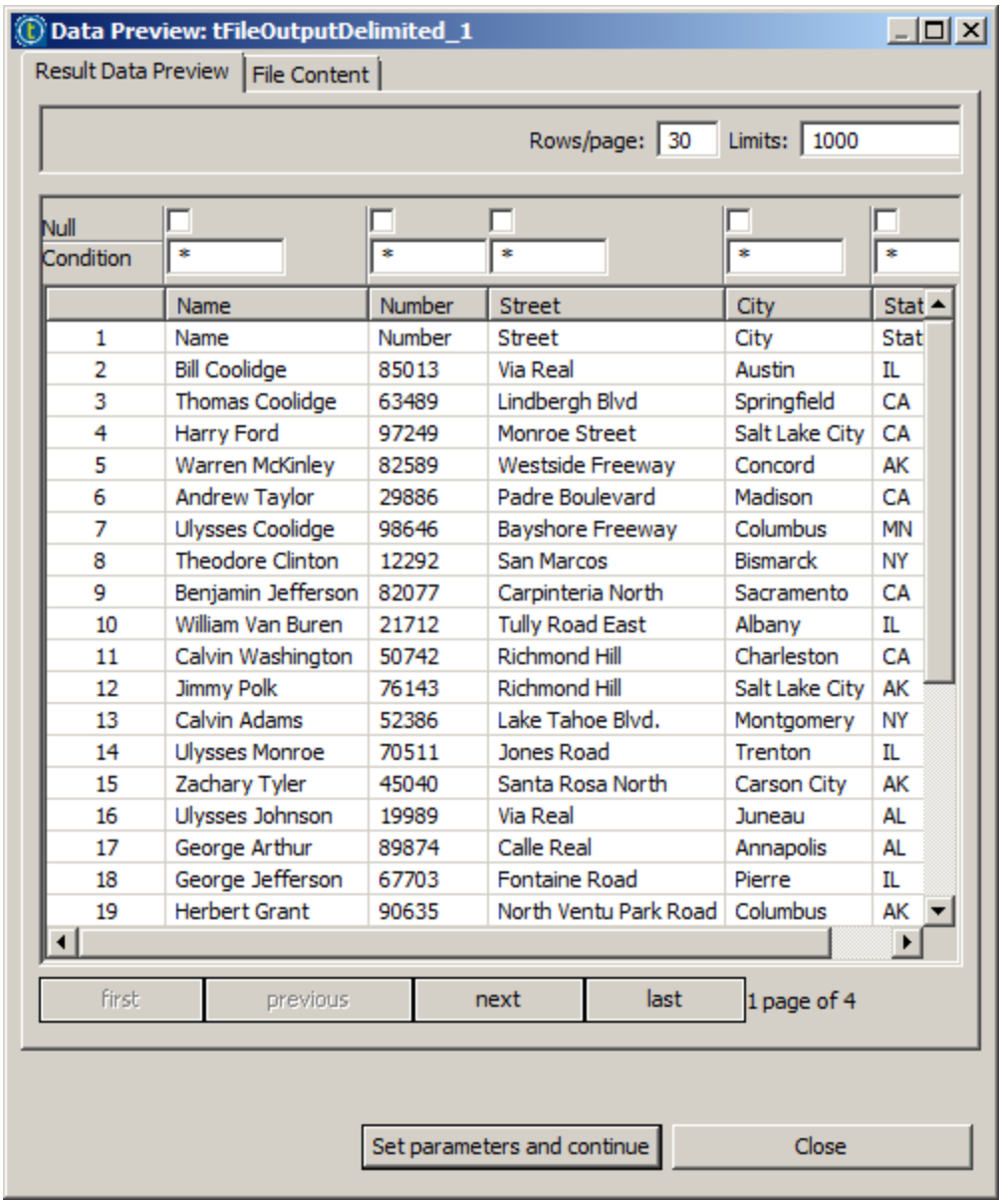
1. VIEW THE OUTPUT

   Run the Job, then right-click the **tFileOutputDelimited** component and select **Data viewer**.



2. EXPLORE THE OUTPUT

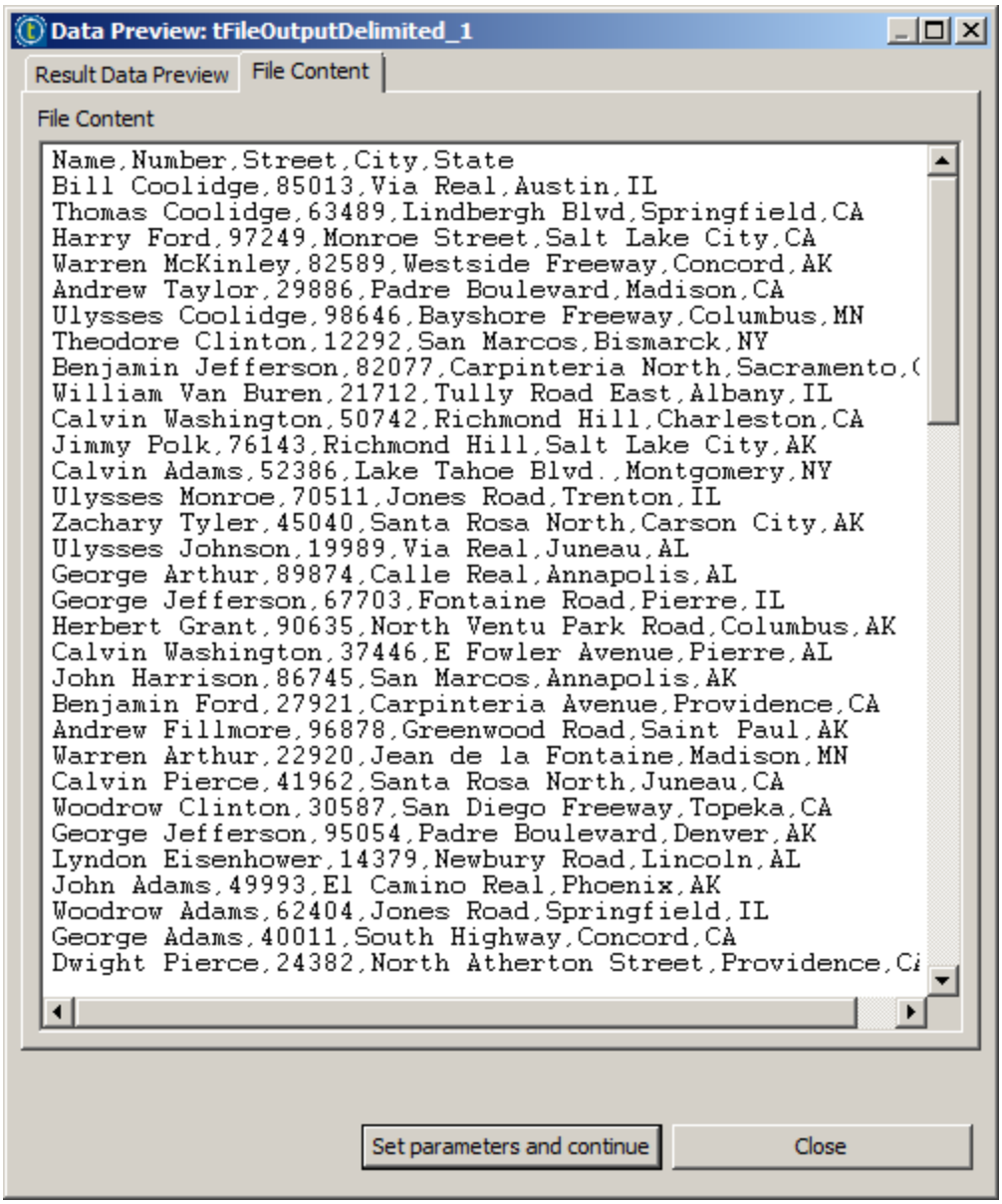
   Explore the content of the output file in the **Data Preview** window.



Notice that the first column is now a combined name (as specified in the Expression Builder).

3. VIEW RAW FILE CONTENT

   Click on the **File Content** tab to see the raw CSV format of the file. Click **Close** when done.



Next

You have now finished this Job. Next you will duplicate the Job, modify it, and execute the new Job.

Duplicating a Job

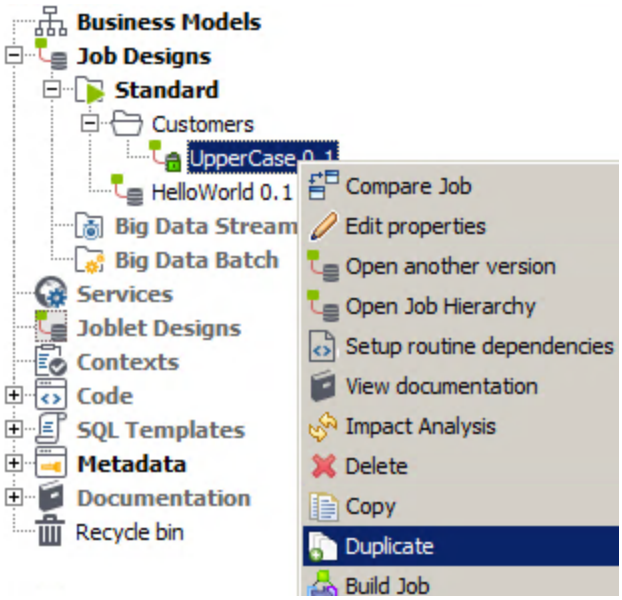
Overview

In the following exercises you will build an extension of the previous Job. Rather than starting from scratch with a new Job, or modifying the existing Job so that it no longer performs its intended function, you can duplicate it as the basis for the following exercises.

Duplicate

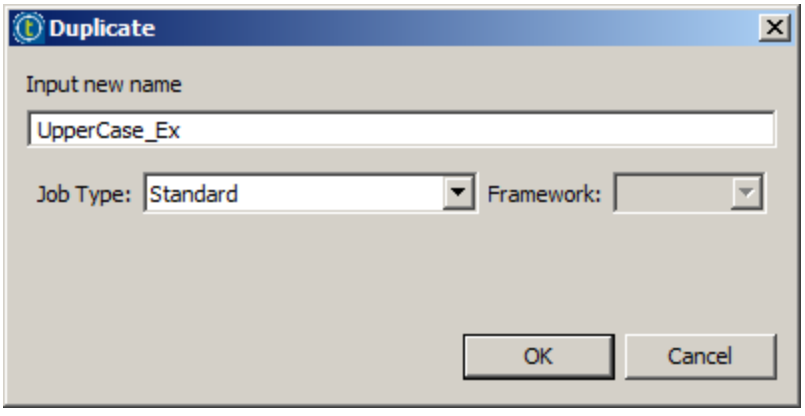
1. DUPLICATE A JOB

   Right-click **Repository > Job Designs > Standard > Customers > Uppercase** and select **Duplicate**.



2. NAME THE JOB

   Enter *UpperCase_Ex* and click **OK**.



TIP:
When you create a new Job, they automatically open in the **Designer**, but when you duplicate an existing Job like you

did here, they do not open automatically. Open the Job by double-clicking on it in the **Repository**.

Next

You have now finished the lesson. Complete the exercises to reinforce your understanding of the topics covered.

Challenges

Overview

Complete these exercises to further explore the use of **tMap** in transforming data. See Solutions for possible solutions to these exercises.

Combine Columns

Modify the **UpperCase_Ex** Job to combine the number and street values into a single column named **Address**.

TIP:

Refer back to the the Combining Columns lesson to see how you performed this action before.

Add a Column

Modify the Job again, adding a new *integer* column named *id* to the output that contains an automatically generated index value.

TIP:

After you add the column from the mapping editor, open the **Expression Builder** and examine the functions in the **Numeric** category for one that generates a sequential number.

Add a Zip Code Column

Modify the **tMap** configuration so that the output contains a new column named *Zip* containing a randomly generated zip code.

TIP:
For the purpose of this challenge, you can assume a zip code is any 5-digit number whose first digit is greater than zero; that is, ranging from 10000 through to 99999.

Next

You have now finished the lesson. It's time to Wrap-up.

Solutions

Overview

These are possible solutions to the challenges. Note that your solutions may differ and still be valid.

Combine Columns

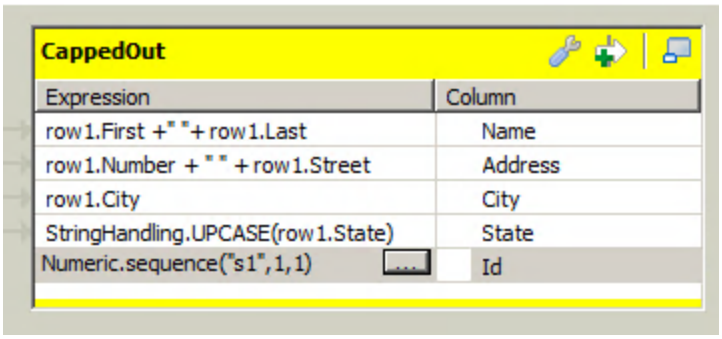
Configure the **CappedOut** table in the **tMap** component, deleting the **Number** column, and replacing the **Street** column with a column named **Address** that contains the following:

row1.Number + " " + row1.Street

Add a Column

Add a column of type **Int** with the name *Id* to the **CappedOut** table containing the following expression:

Numeric.sequence("s1",1,1).



Add a Zip Code Column

Click the **CappedOut** table and then add a column in the **Schema editor** of type **int** named **Zip**. Open the **Expression Builder** for the new column expression and then create an expression like the following:

Numeric.random(10000,99999)

Run

Run your Job and then open **CappedOut.csv** to check the result.

Next

You have now completed this lesson. It's now time to Wrap-Up

Wrap-Up

In this lesson you learned how to read a delimited file and how to define the related schema. You also learned how to apply simple transformations on the input data and write them into an output file.

Next step

Congratulations! You have successfully completed this lesson. To save your progress, click **Check your status with this unit** below. To go to the next lesson, on the next screen, click **Completed. Let's continue >**.

LESSON 3

Joining Data Sources

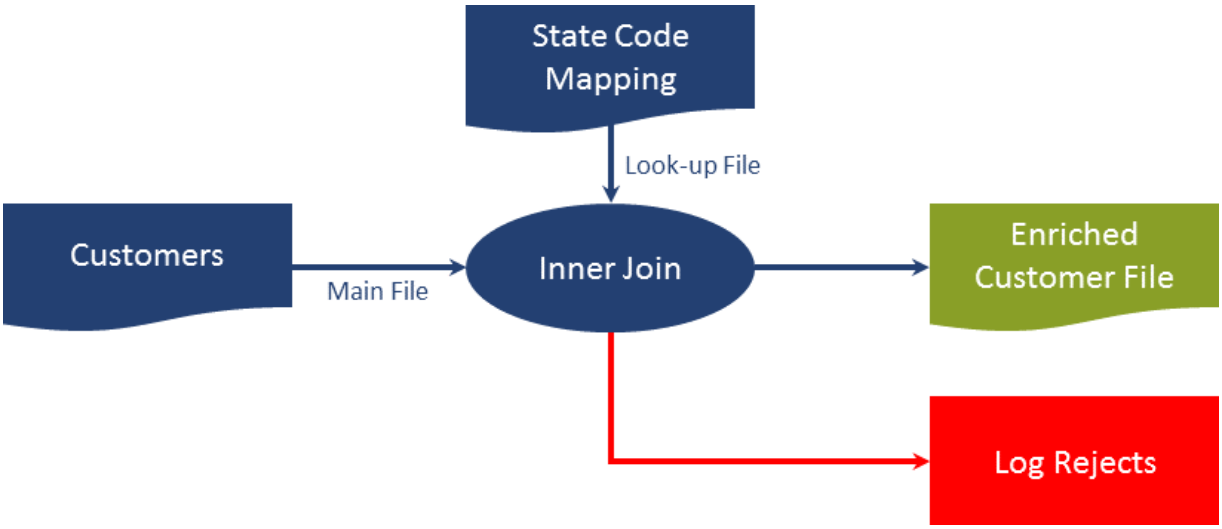
This chapter discusses:

Joining Data Sources

Lesson Overview

This lesson provides you with practice joining data from multiple sources. Most enterprises have data in multiple locations and need to combine that data, either to store it in a unified format or to process it consistently. The example in this lesson builds on the previous example that capitalized US state codes. In this lesson, you will incorporate a look-up table containing a list of US state codes and names, so that the output contains both the abbreviation and the full state name.

A common column (the state code) is used for the join between the two input sources.



Objectives

After completing this lesson, you will be able to:

- Use metadata
- Store metadata centrally for use in other components and Jobs
- Join two data sources
- Troubleshoot a join by examining rejects
- Log rejected data rows to the console

Next Step

The first step will be to create metadata for the states file.

Creating Metadata

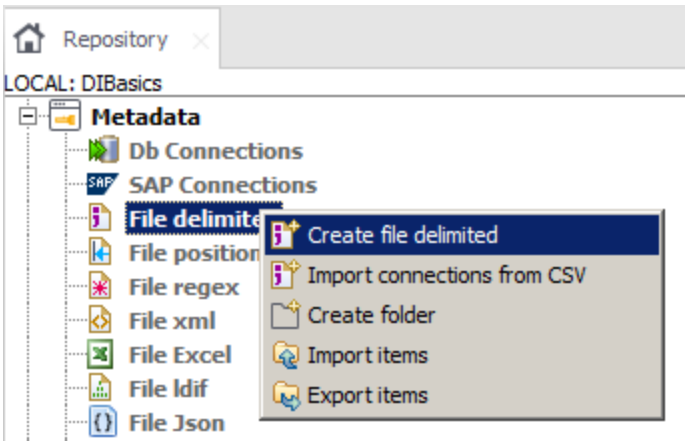
Overview

You previously configured a file input component with information about the source file and its schema. That information was local to that particular component. With Talend Studio, you can store such configuration information as **Metadata** so that you can reuse it for multiple components, whether in the same or different Jobs. You are creating information about a delimited file containing US state codes and names so that you can use the same information for multiple components in multiple Jobs.

Create Metadata

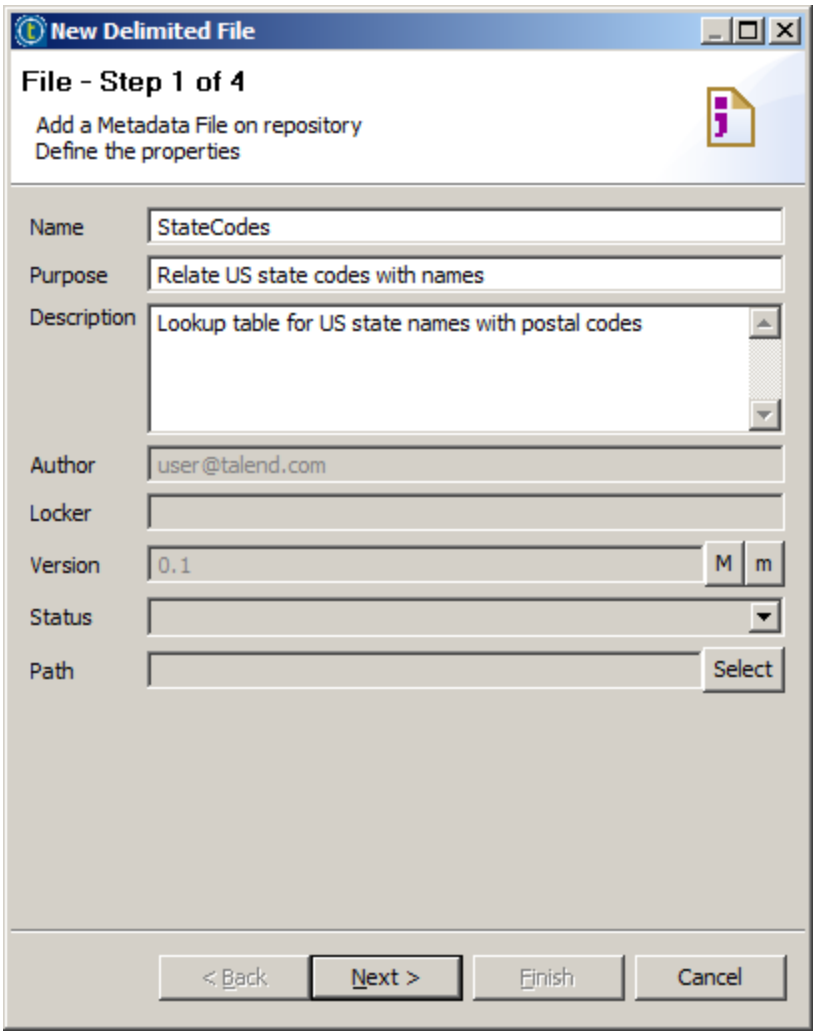
1. CREATE DELIMITED FILE METADATA

   Right-click **Repository > Metadata > File delimited** and select **Create file delimited**.



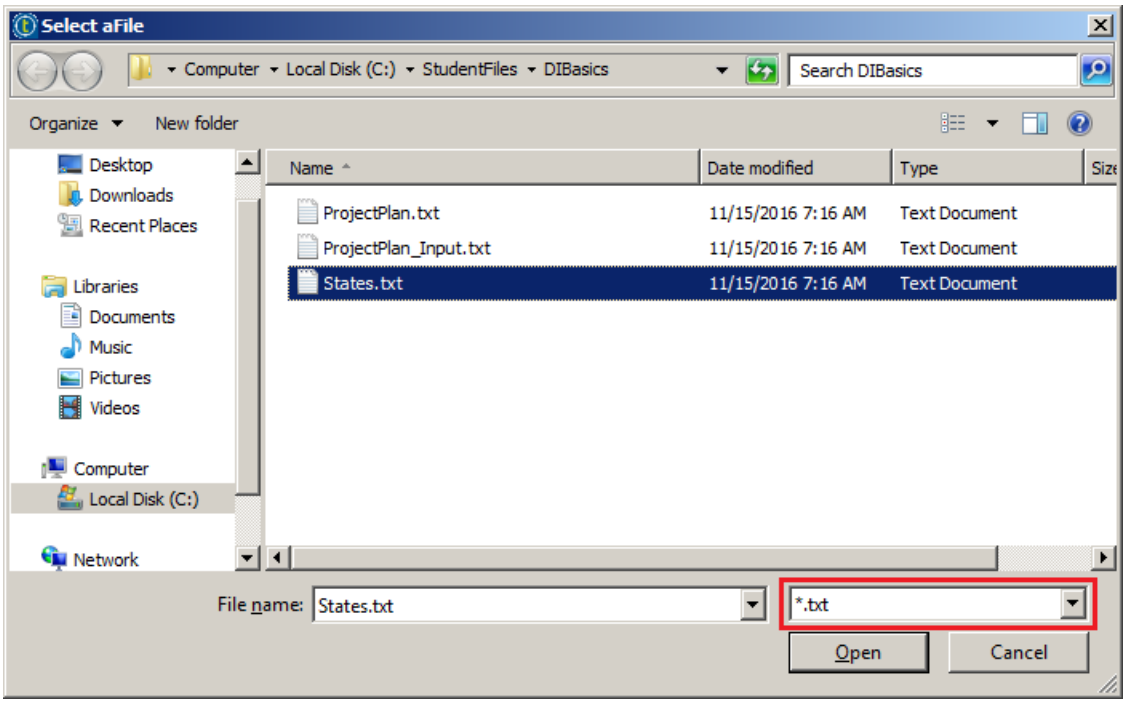
2. ENTER DETAILS

   Enter *StateCodes* in the **Name** box, and appropriate descriptions into the **Purpose** and **Description** boxes, then click **Next**.

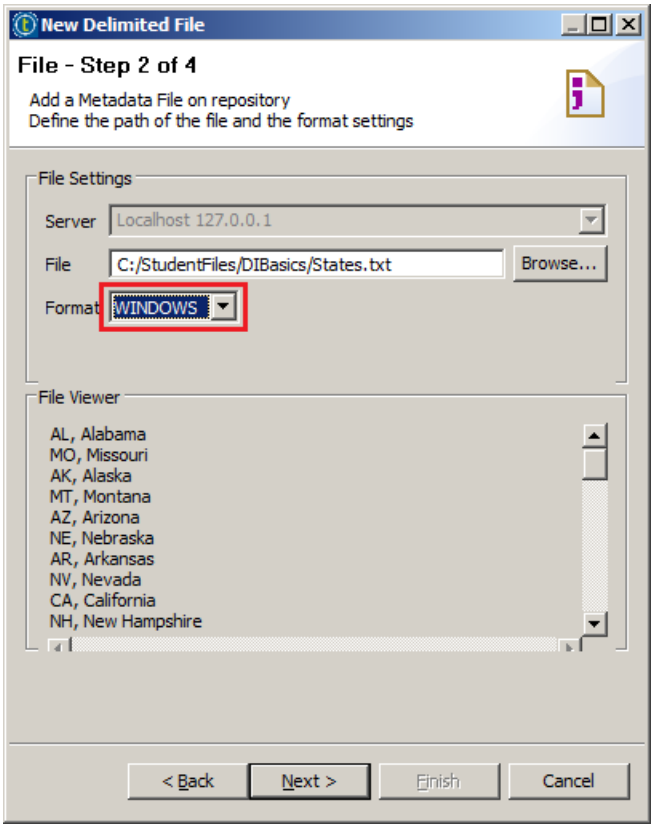


3. SPECIFY FILE LOCATION AND FORMAT

   Click **Browse** to search for the required text file. In the **Select a File** dialog, select ***.txt** in the file type list, and choose the file *States.txt* in the folder *C:\StudentFiles\DIBasics*, then click **Open**.



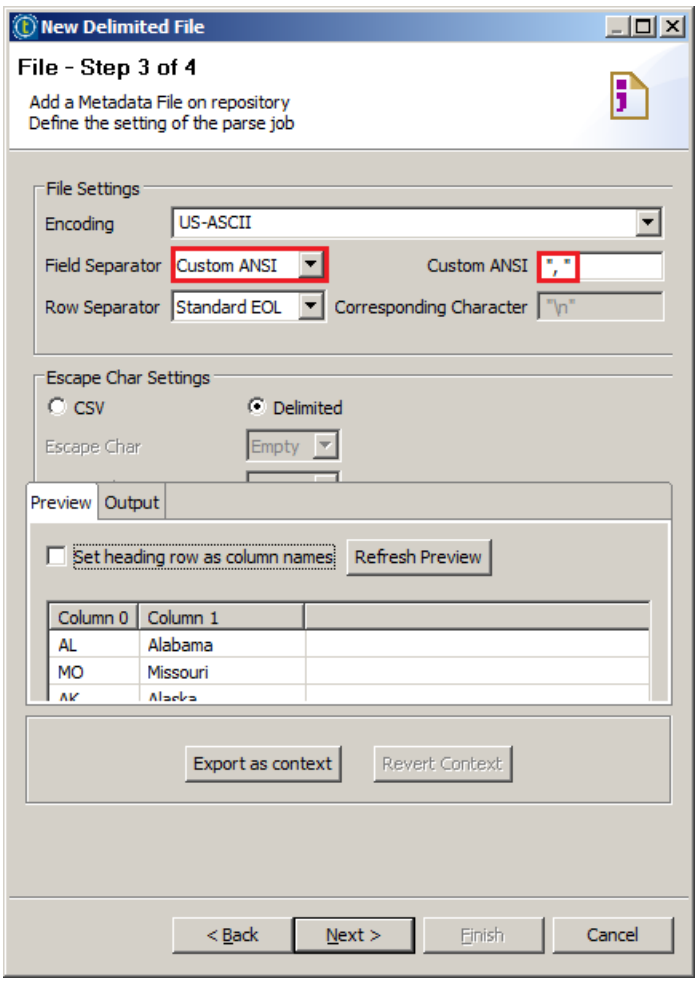
Set the **Format** to **WINDOWS**, then click **Next**.



4. CONFIGURE THE FORMAT

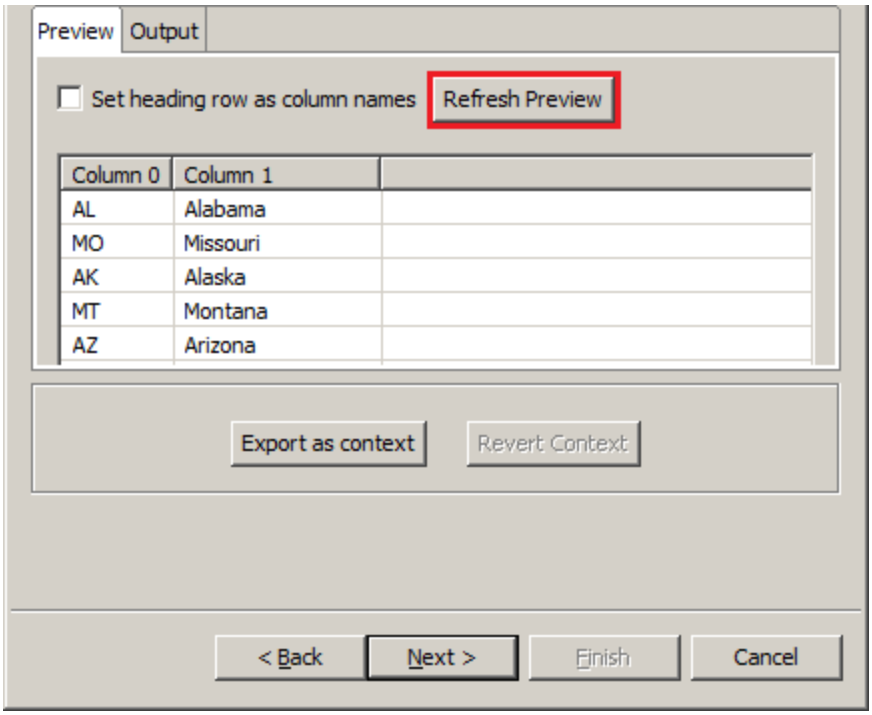
Specify the **Field Separator**. For a typical CSV (comma-separated value) file, *Comma* would be an appropriate setting, however if you look closely at the format of the *States.txt* file, you will see that the fields are separated by a comma and a space, not just a comma. To deal with this issue, specify *Custom ANSI* for **Field Separator** and enter *", "* (comma followed by a space, enclosed in quotation marks) in the **Custom ANSI** box.

Leave the other settings as they are.



5. TEST THE SETTINGS

In the bottom area of the page, click **Refresh Preview**. Now that you have specified the correct delimiter, Talend Studio recognizes that the file contains two columns, and the values for both fields are extracted properly. Click **Next**.



6. CONFIGURE THE SCHEMA

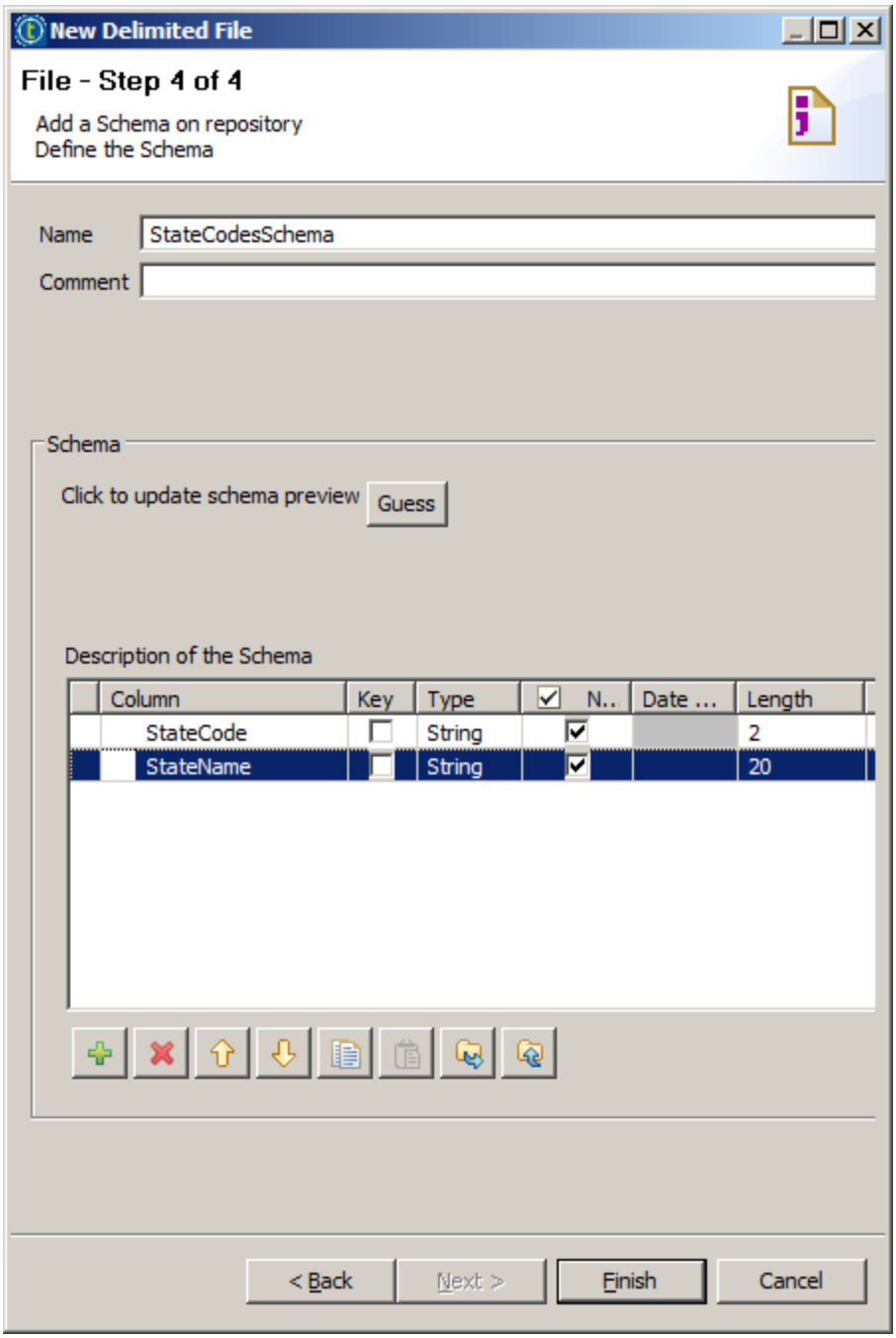
   The final step is where you specify the schema, or structure, of the file.

   Enter *StateCodesSchema* into the **Name** box.

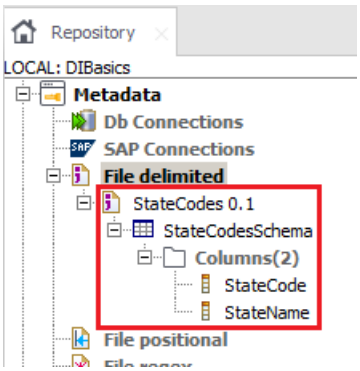
   Then, replace **Column0** with *StateCode* and set its **Length** to *2*.

   Next, replace **Column1** with *StateName* and set its **Length** to *20*.

   When done, the schema should look like this. Click **Finish**.



The new metadata appears in the **Repository**.



Next

Now you are ready to use this metadata to create a component that will read state codes from a file in order to join them to customers information.

Creating a Join

Overview

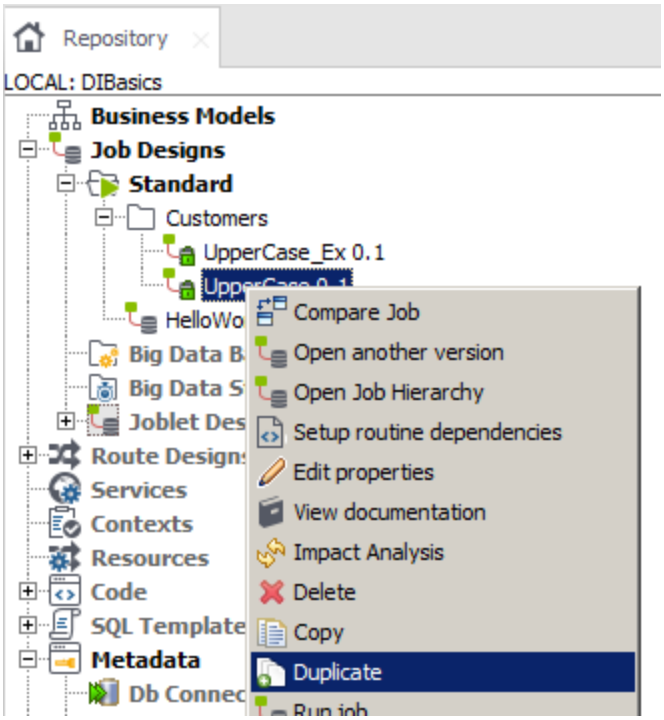
In this section, you will first duplicate the Job created in the previous lesson.

Then you will use the file containing state codes and names as a lookup table to include the full state name in the output.

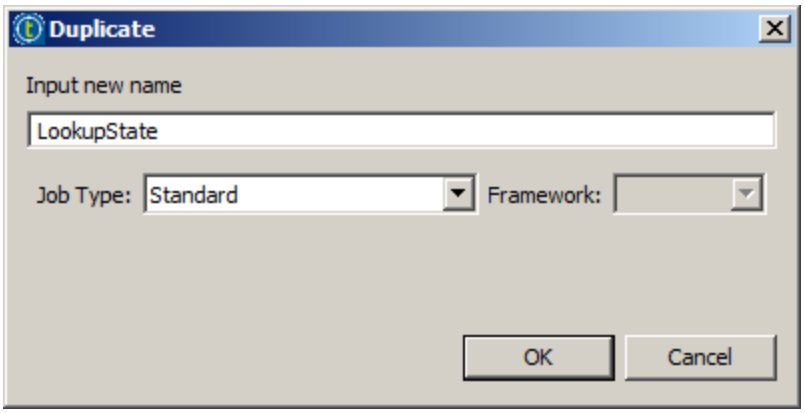
Duplicate the Job

1. DUPLICATE THE JOB

   Right-click **Repository > Job Designs > Standard > Customers > UpperCase** and select **Duplicate**.

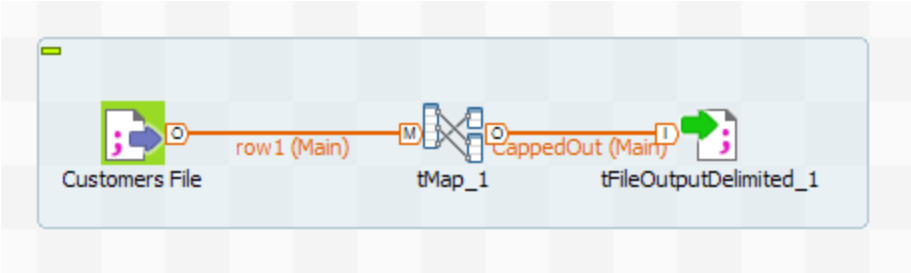


   Enter *LookupState* and click **OK**.



2. OPEN THE JOB

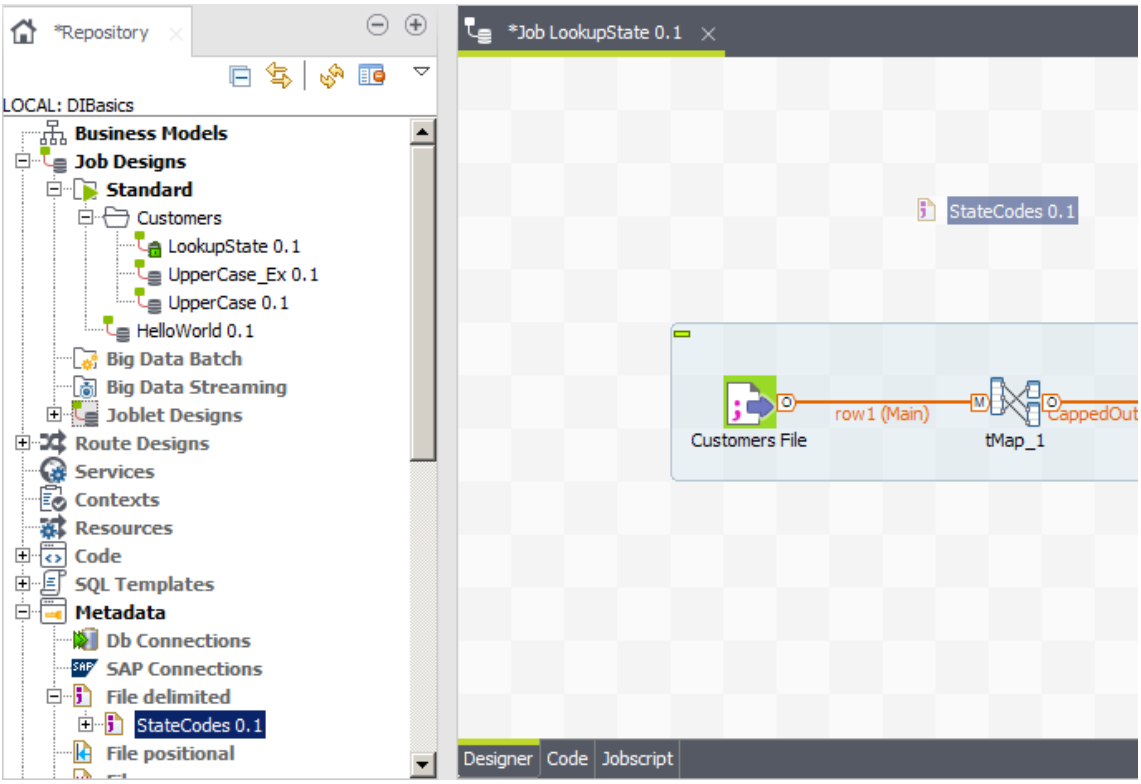
Open the job by double-clicking the new **LookupState** in the **Repository**. Take a moment to examine the components to remind yourself of the function of the current Job.



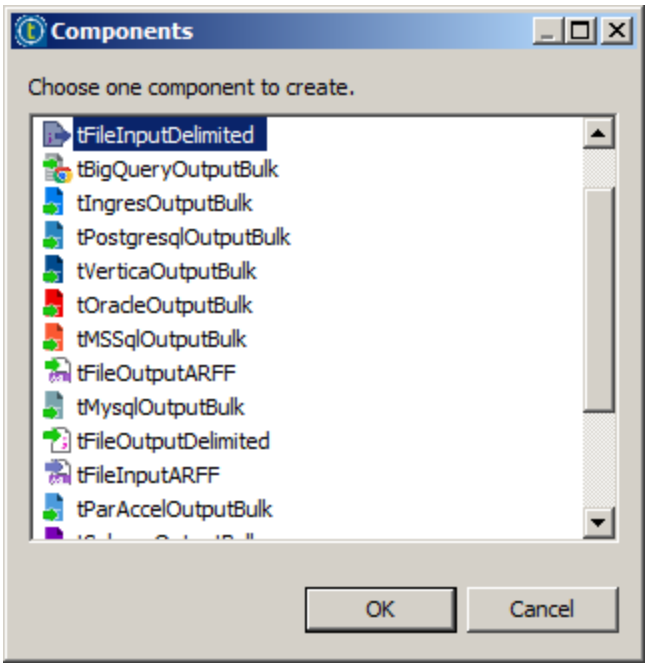
Add Second Source

1. APPLY THE METADATA AS A SECOND INPUT

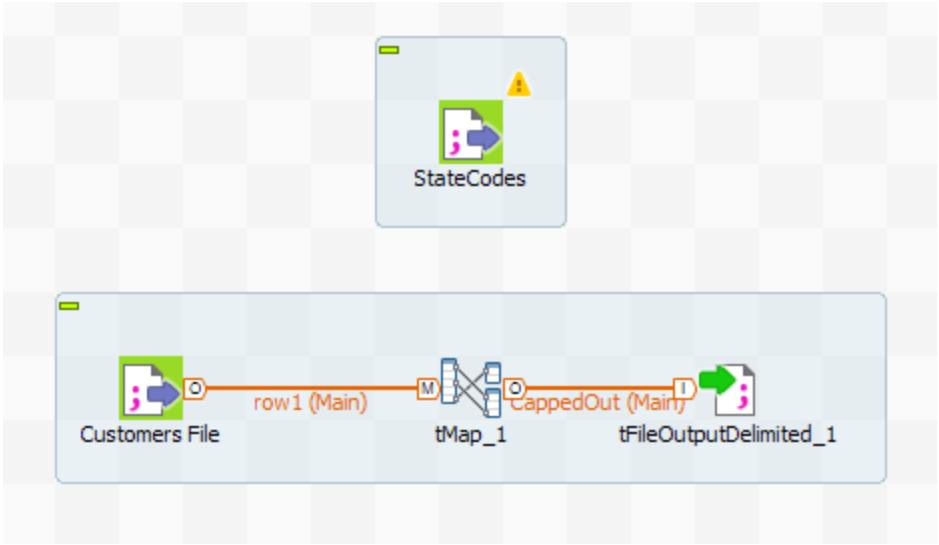
Drag the new **StateCodes** metadata item created earlier onto the **Designer** workspace, above the **tMap** component. If you don't have room, you can click the component group and simply drag them lower in the **Designer**.



The **Components** dialog allows you to choose which component you want to use with the metadata. Choose **tFileInputDelimited**, then click **OK**.



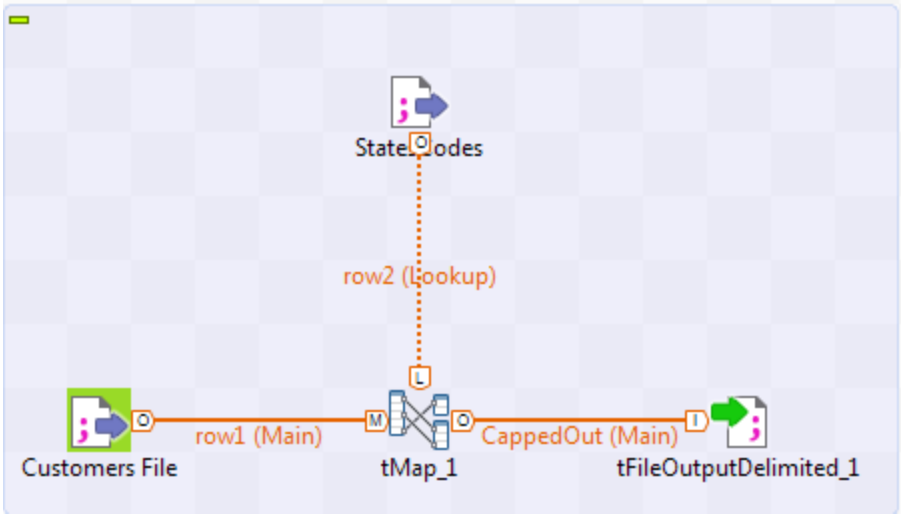
The Job should now resemble the following figure.



Notice that the component label is the same as the name of the metadata item. When you store configuration information as metadata in the **Repository**, you can use that metadata as a starting point, as you did here, rather than starting from scratch.

2. CONNECT THE NEW COMPONENT

   Right-click **StateCodes**, then select **Row > Main**, then click the **tMap** component to create a row.

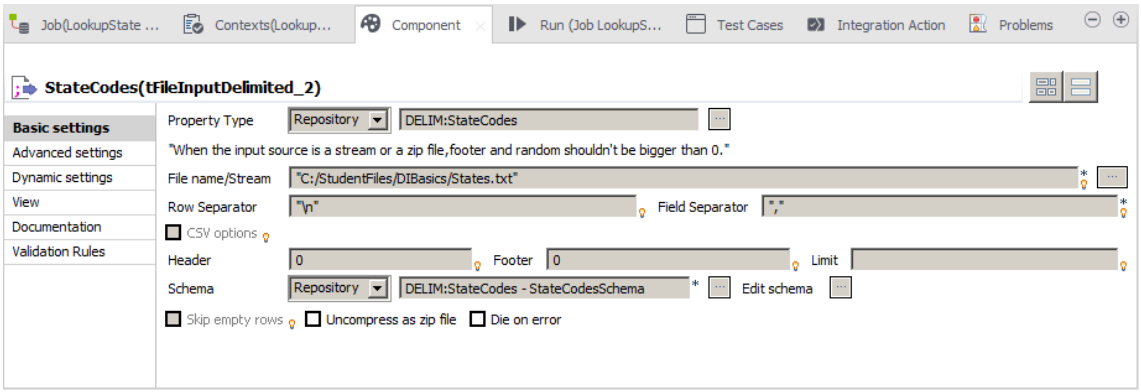


Notice that the row is a **Lookup**, not a **Main**. You can provide multiple input sources for a **tMap** component, but only one can be the **Main** row with the rest being **Lookups**.

3. VIEW COMPONENT CONFIGURATION

   Double-click the **StateCodes** component to open the **Component** view.

   Notice that the **Property Type** and **Schema** value is **Repository**, not **Built-In**.



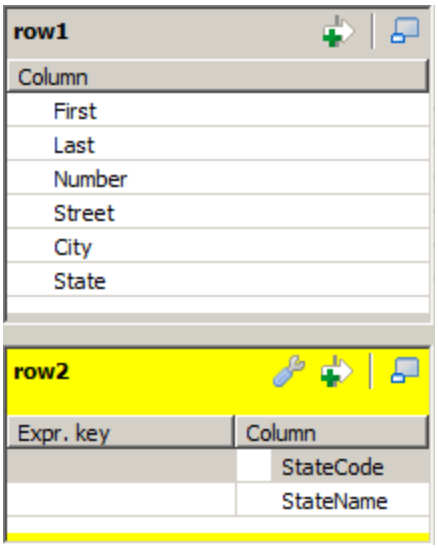
4. COMPARE BUILT-IN AND REPOSITORY CONFIGURATION

   Double-click the **Customers File** component to see that this component uses **Built-In** configuration information.
   So, a Built-in property type is specific to a single component, while a Repository property type is stored as Metadata and can be used by multiple components in multiple Jobs.

Configure tMap

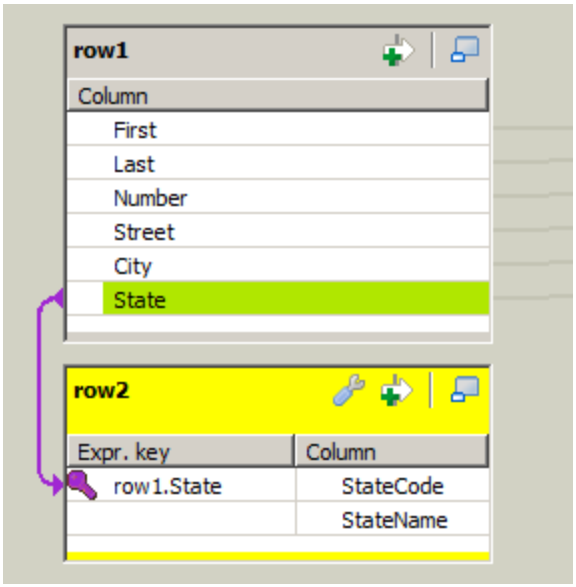
1. OPEN THE MAPPING EDITOR

   Double-click the **tMap** component. In the mapping editor, notice that there is now a second input table on the left, called **row2**, corresponding to the input row connecting **StateCodes** to the **tMap** component.



2. LINK THE ELEMENTS TO COMPARE

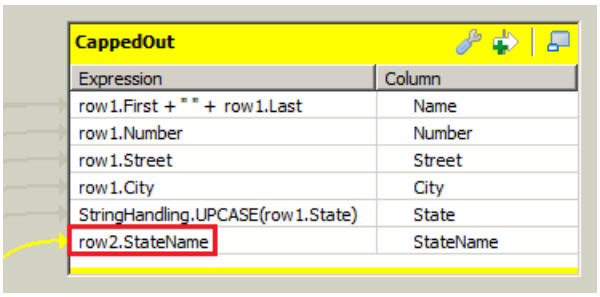
   Drag *State* from the *row1* table to the *Expr.key* field for *StateCode* in the *row2* table.



   This creates a Join, so that the value of the *State* column from *row1* will be compared to the value of the *StateCode* column in *row2*.

3. LINK TO THE OUTPUT TABLE

   Drag *StateName* from the *row2* table to the bottom empty row of the *CappedOut* table on the right.



This adds the value of the *StateName* column to the output.

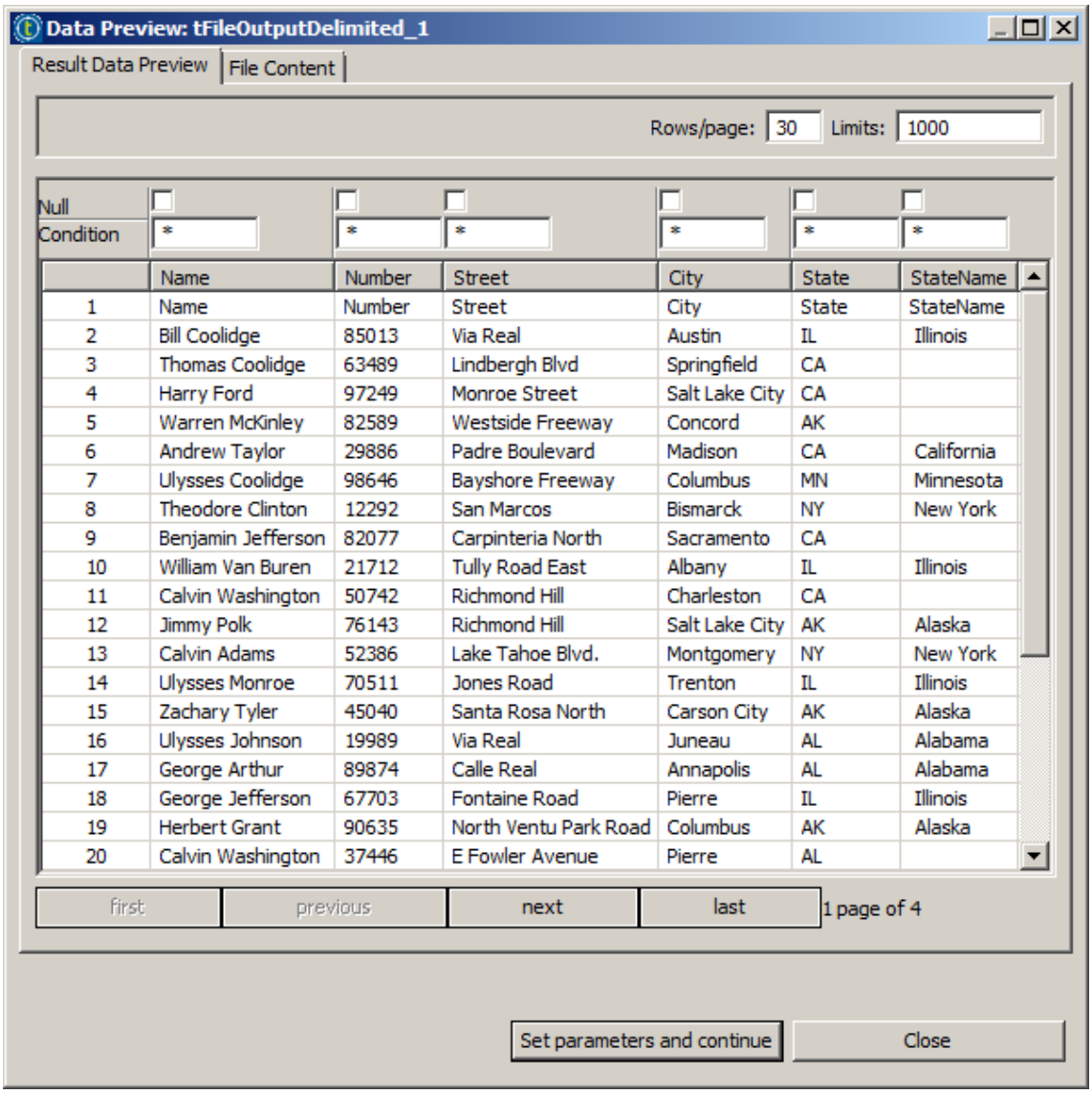
4. SAVE CHANGES

   Click **Ok**, then click **Yes** when prompted to propagate the changes.

Run Job

1. RUN THE JOB

   Run the job, then examine the output file. Right-click the **tFileOutputDelimited** component and select **Data viewer**.



Notice that not all rows include a value in the *StateName* column. This means that the Job is not doing exactly what you intended, so you need to investigate further.

Next

The next step is to determine why the Job is failing to perform as expected so that you can correct it.

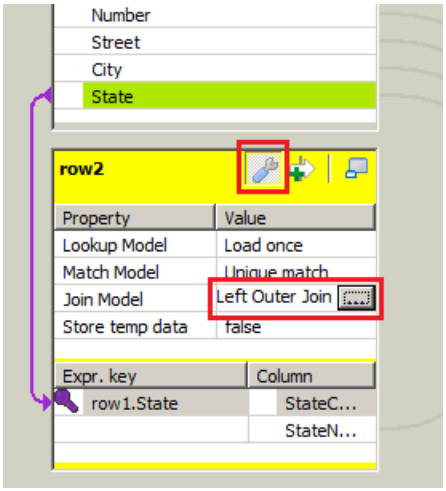
Capturing Rejects

Overview

Configure Join Model in tMap

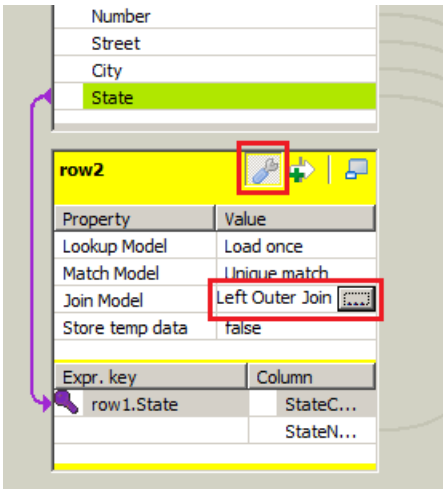
1. VIEW TABLE SETTINGS

   Double-click the **tMap** component. In the *row2* table, click the **tMap settings** button ( ). This is where you can configure additional parameters about the table. Notice that the **Join Model** says **Left Outer Join**. A left outer join includes all rows from the primary table even if there is not a row in the look-up table with a matching value in the join column, which explains why some rows did not include the state name. But now you need to know why some look-up rows did not match, and in order to do that, you need to use an inner join.



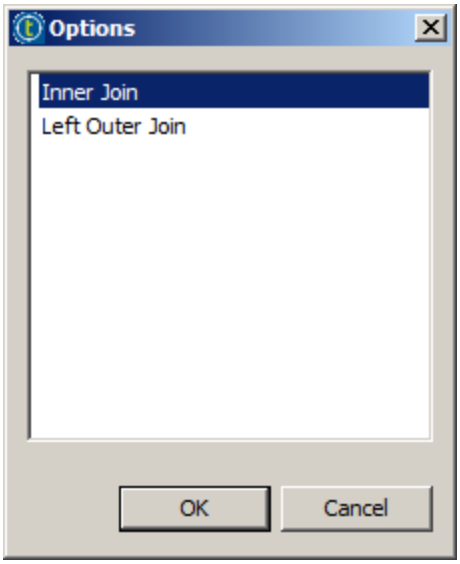
2. USE AN INNER JOIN

   Click **Left Outer Join** in the *Value* column, next to **Join Model**, and then click the button marked with an ellipsis **[...]** to change the value.



   The **Options** window opens. Choose **Inner Join** and then click **OK**. By changing the join to an inner join, rows that don't

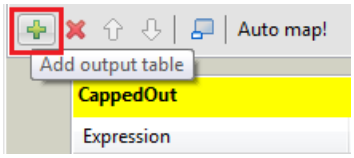
match will be excluded from the output, and you can capture the rejects for troubleshooting purposes.



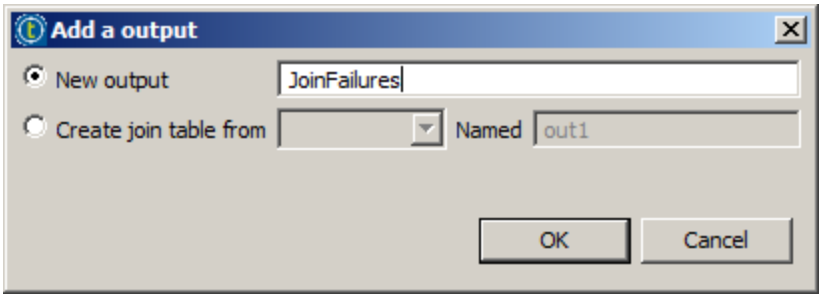
Add a Rejects Table

1. ADD A NEW TABLE TO CATCH REJECTS

   Still in the **tMap** component, click the **Add output table** icon (+) above the **CappedOut** table.



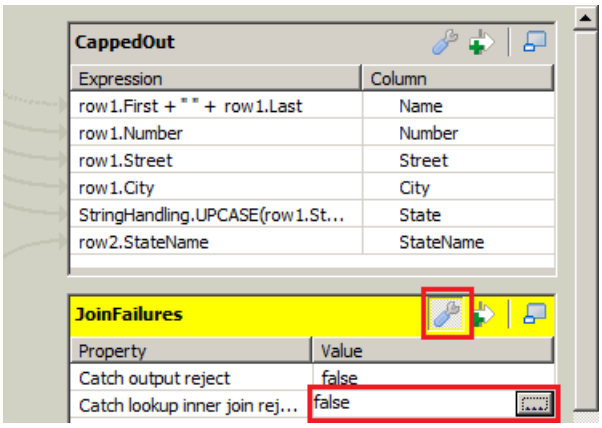
   You are creating a new table to capture rows that fail the inner join so that you can determine the problem. Enter *JoinFailures* into the text box to name the new output, and then click **OK**.



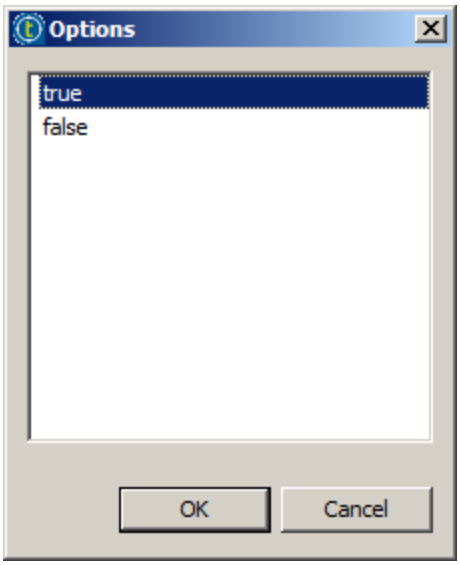
   The new table appears below *CappedOut*.

2. CONFIGURE SETTINGS TO CATCH INNER JOIN REJECTS

   Click the **tMap settings** button ( ) in the *JoinFailures* table. Click **false** in the *Value* column, next to **Catch lookup inner join reject**, and then click the button marked with an ellipsis **[...]** to change the value.

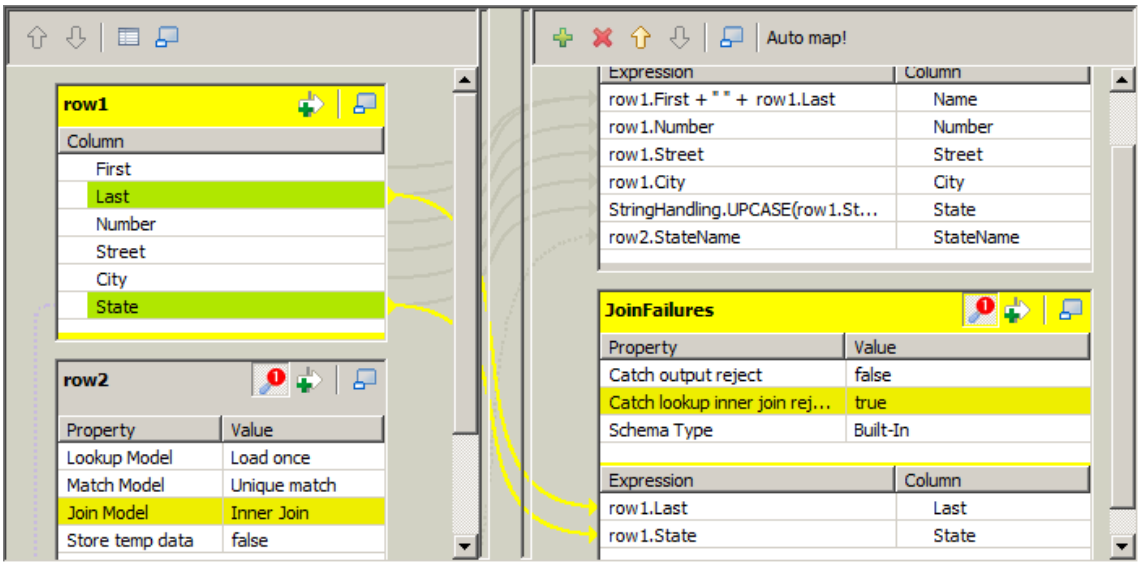


In the **Options** window, select **true**, then click **OK**.



3. DEFINE COLUMNS OF THE OUTPUT TABLE

   Drag *Last* and *State* from the *row1* table to the **JoinFailures** table.



Now the *JoinFailures* output will hold the last name and state for rows that fail the inner join.

4. SAVE CHANGES

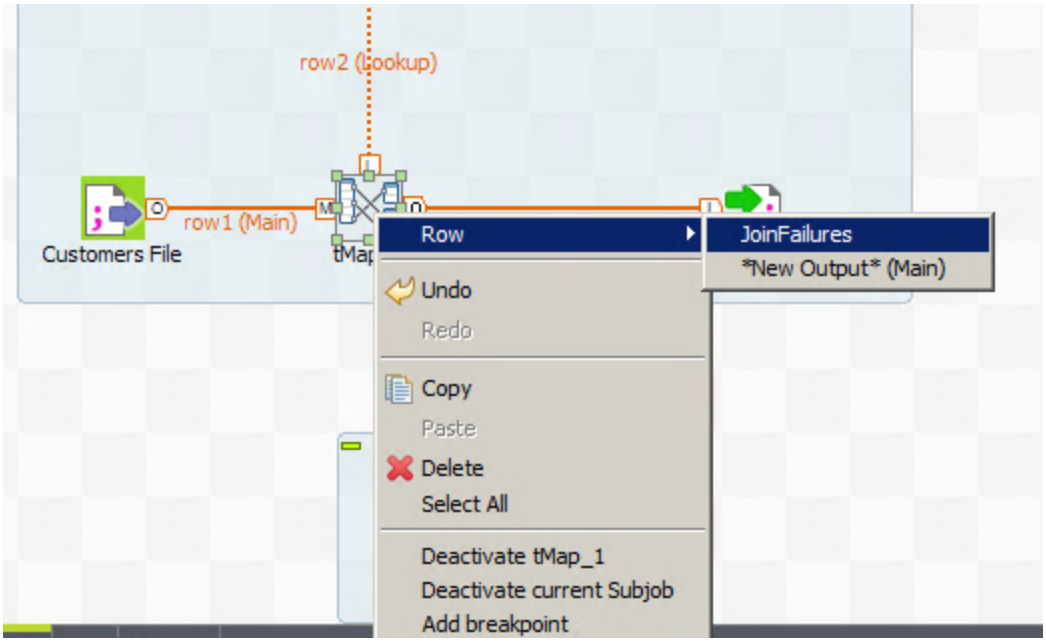
   Click **Ok** to apply the changes to the **tMap** component.

Log Failures

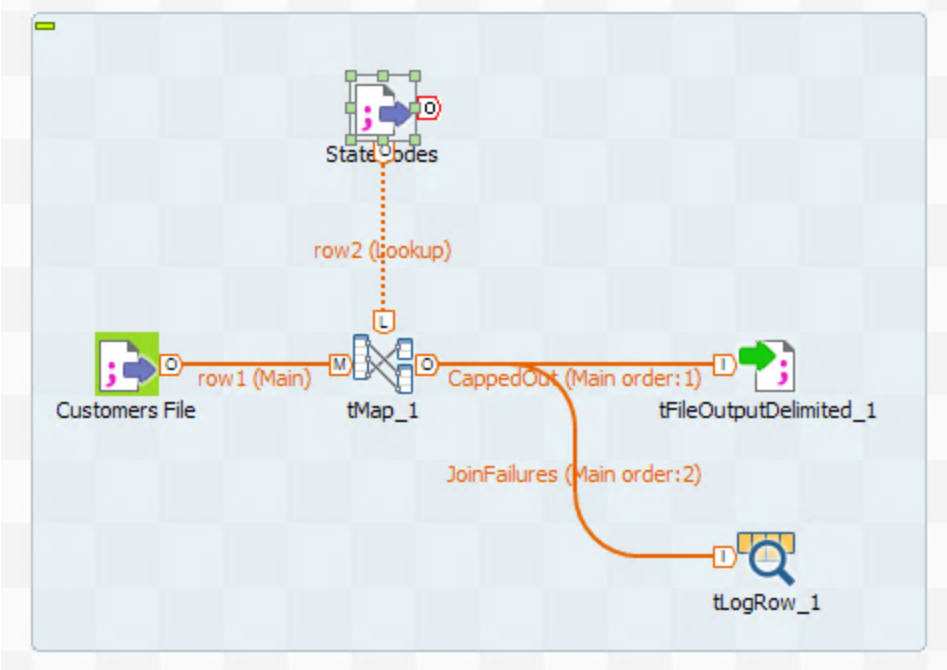
1. ADD AN OUTPUT COMPONENT TO LOG THE FAILURES

   Add a **tLogRow** component just below the **tMap** component (recall that this component writes rows to the console in the **Run** view).

   Connect the new component by right-clicking the **tMap** component, selecting **Row > JoinFailures** , then clicking on the **tLogRow**.



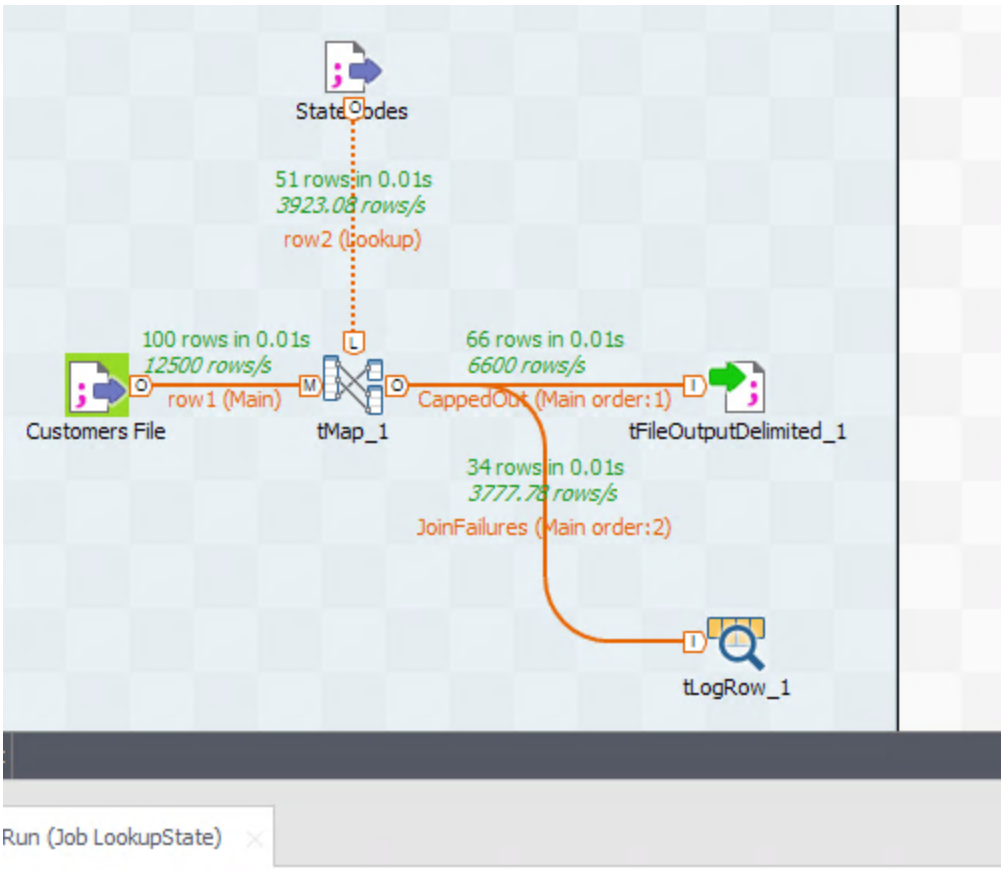
Notice that the *JoinFailures* row corresponds to the output table you added to the **tMap** component:.

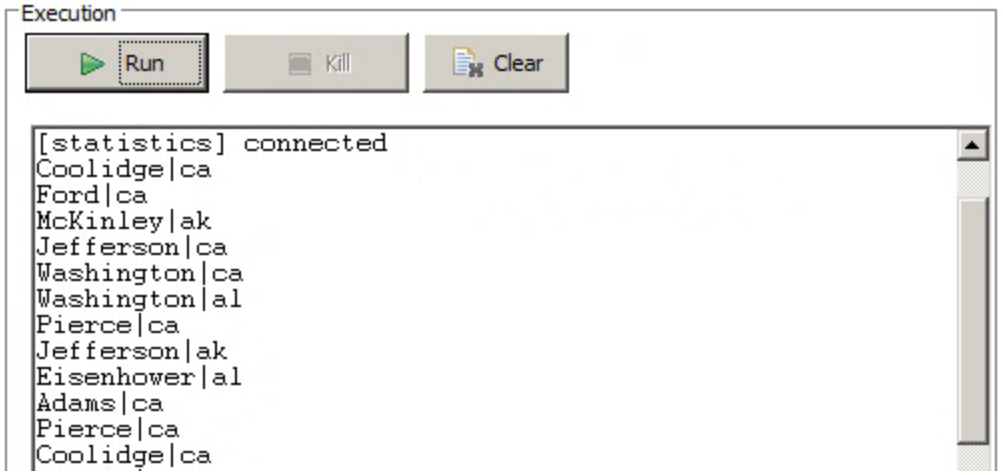


Run the Job

1. RUN THE JOB

   Run the Job and examine the results.





Notice from the output a couple of important points:

- The summary in the **Designer** tells you that 100 rows were processed, of which 34 were rejected
- From the console output, it is evident that the state codes in the rejected rows are all lowercase. They will not match the uppercase codes from the look-up file.

Next

Now that you know the problem with the join, you can make a correction so that the look-up works as expected.

Correcting a Lookup

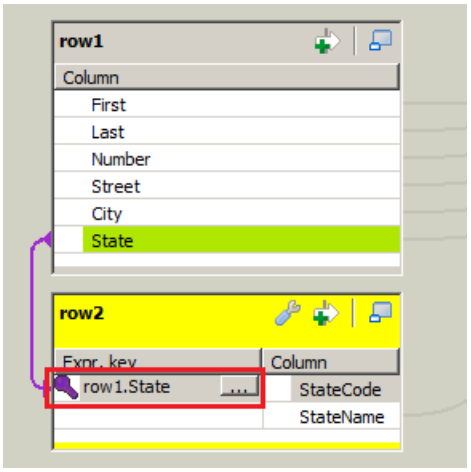
Overview

Now that you know the lookup is failing because of lowercase string values, you can correct the configuration. You need to convert the state codes to uppercase before comparison against entries in the lookup table.

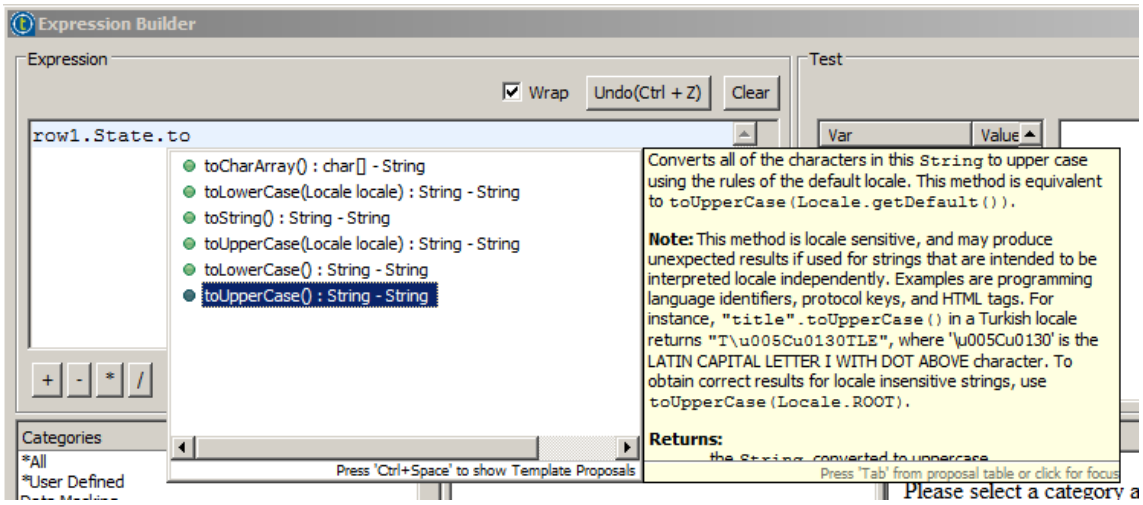
Configure tMap

1. BUILD AN EXPRESSION TO CONVERT TO UPPERCASE

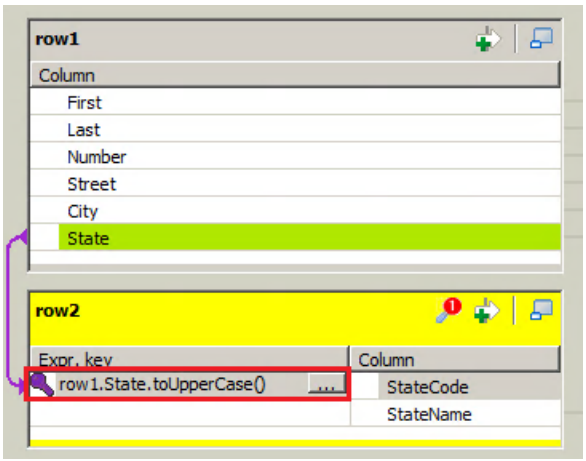
   Double-click the **tMap** component. Remember that *row2* is the lookup table for the join. In the *row2* table on the left, click **row1.State** and then click the button marked with an ellipsis **[...]** to open the **Expression Builder**.



   After the last character of *row1.State* begin typing *.to* and a list will appear displaying matching options. Double-click **toUppercase()** in order to select it. Click **Ok** when done.



   The value of the *State* column is now converted to uppercase before being compared against the lookup table, so the join should work properly.



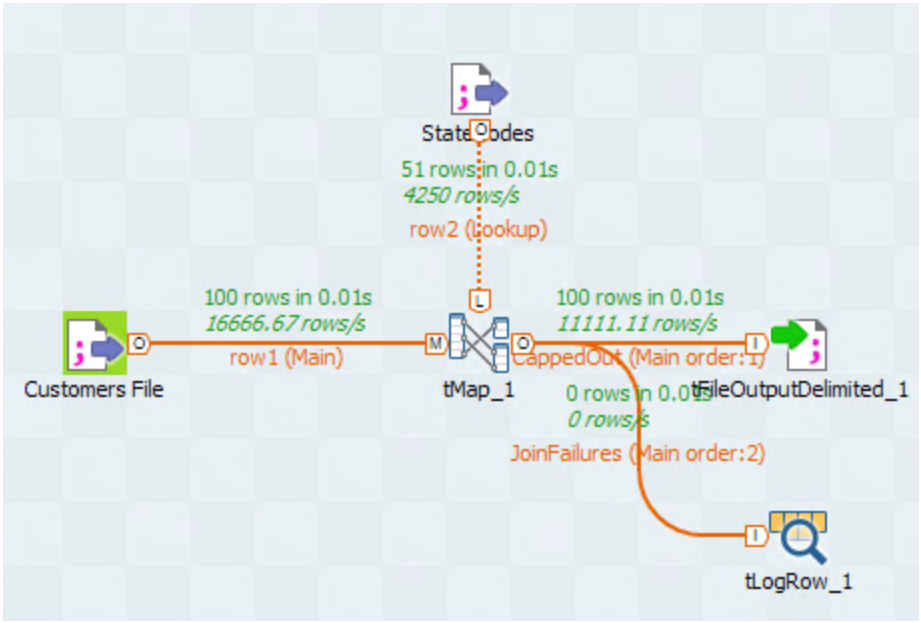
2. SAVE CHANGES

   Click **Ok** to apply the changes to the **tMap** component.

Run the Job

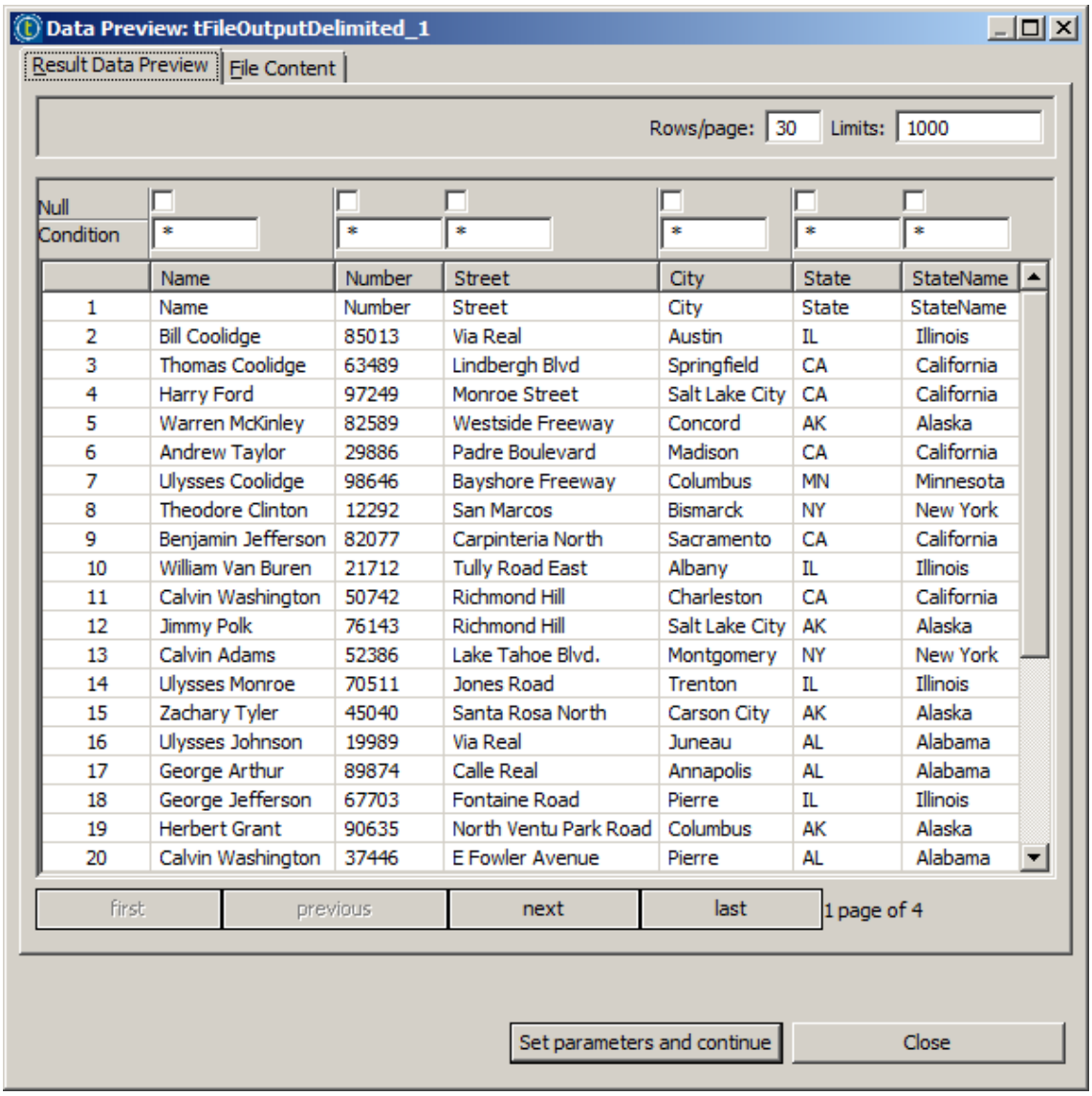
1. RUN THE JOB

   Run the Job. Notice that the **tLogRow** component did not process any rows, so no rejects were found. All the input rows flow to the output file.



2. USE THE DATA PREVIEWER TO CHECK THE OUTPUT

   Examine the output file content using the **Data Preview** option to make sure that all rows include the state name.

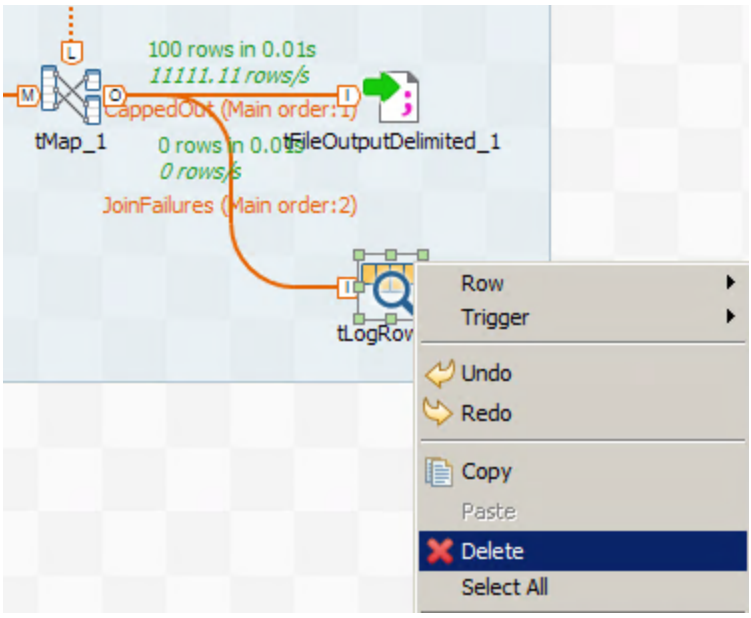


At this point, there is no longer a need to log the rejects, so you can clean up the Job by removing the **tLogRow** component.

Clean up the Job

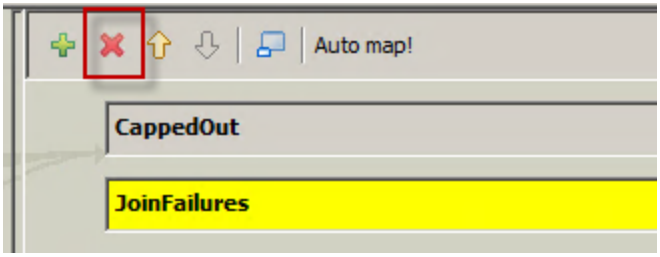
1. REMOVE UNNECESSARY COMPONENT

   Right-click the **tLogRow** component and select **Delete** to remove the component.

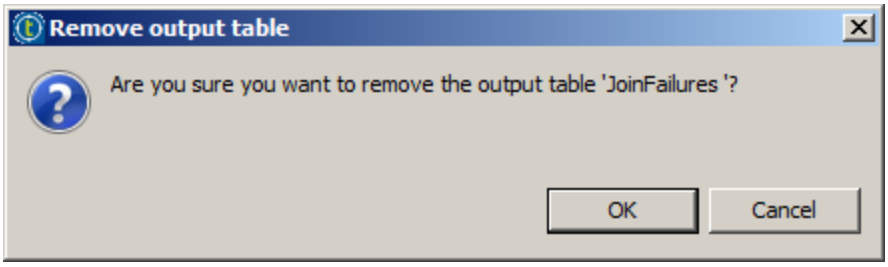


2. REMOVE UNNECESSARY TABLE

   Double-click the **tMap** component to open the mapping editor. Click the *JoinFailures* table on the right and then click the **Remove selected output table** icon ( ).



   Answer **OK** when asked if you are sure you want to remove the **JoinFailures** table.



   Finally, click **Ok** to close the mapping editor.

3. SAVE THE JOB

   Save the changes you have just made by pressing **Ctrl+S**, or by clicking the Save button ( ) in the main tool bar, at the top-left corner of the main window.

Next

You have completed this Job. It is time to Wrap-Up.

Wrap-Up

In this lesson, you extended the Job from the previous lesson to explore joining two data sources through a **tMap** component. You looked at how to troubleshoot data issues by capturing join failures, and practiced writing rows to the console. You also stored component configuration information as metadata in the Repository so that it could be used later by multiple components and other Jobs.

Next step

Congratulations! You have successfully completed this lesson. To save your progress, click **Check your status with this unit** below. To go to the next lesson, on the next screen, click **Completed. Let's continue >**.

LESSON 4

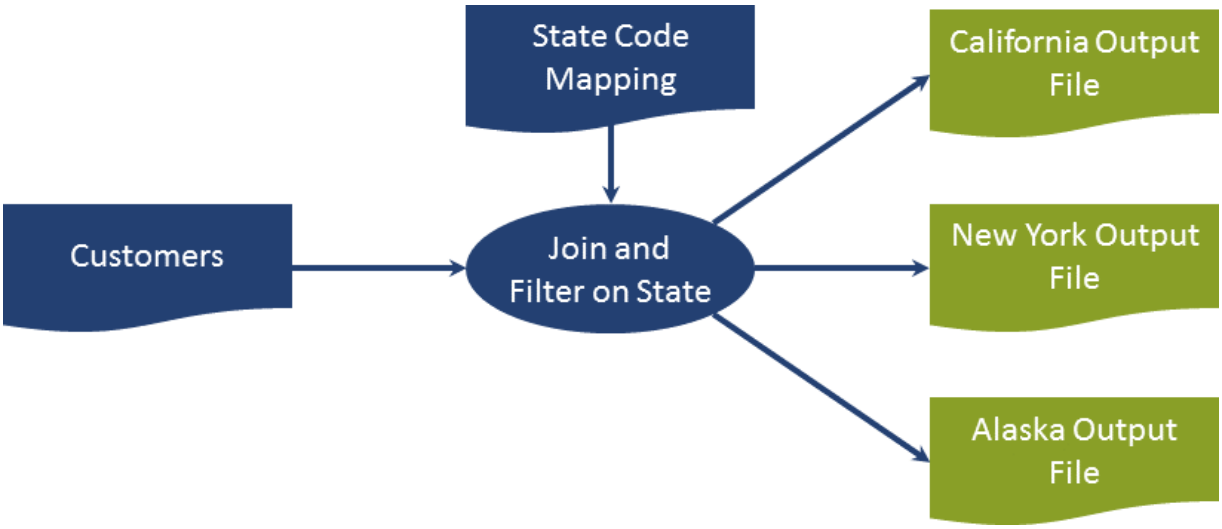
Filtering Data

This chapter discusses:

Filtering Data

Lesson Overview

A common task in data integration projects is to filter data rows based on content for separate processing, storage, or reporting. In this lesson, you will build a Job that extends the previous Job to join the customer data with state data, then separates the results into different output flows based on the value of the column containing a state code.



Objectives

After completing this lesson, you will be able to:

- Use the tMap component to filter data
- Execute Job sections conditionally
- Duplicate output flows

Next Step

The first step is to add a filter on the data so that customers from only one state are written to each output file.

Filtering Output Data

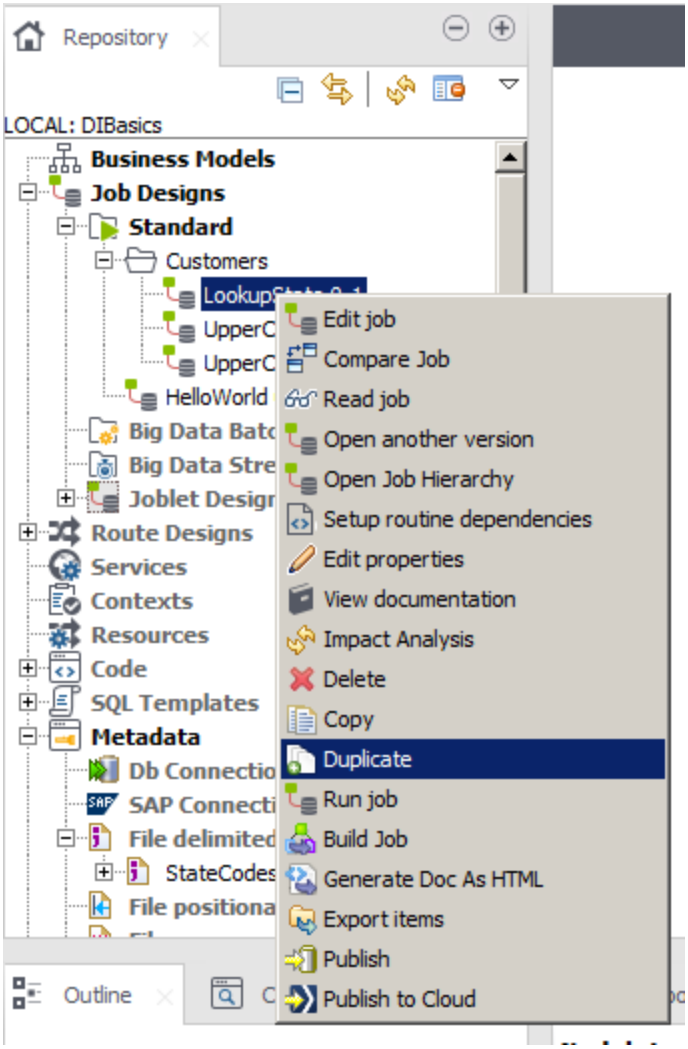
Overview

Your Job needs to filter the output based on the state code so that the resulting output files are restricted to specific states.

Duplicate Job

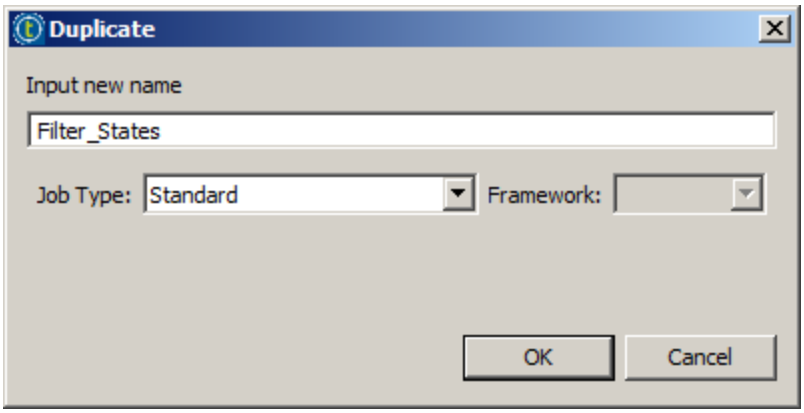
1. DUPLICATE A JOB

   Right-click **Repository > Job Designs > Standard > Customers > LookupState** and select **Duplicate**.



2. NAME THE JOB

   Enter *Filter_States*, then click **OK**.



3. OPEN THE JOB

   Double-click the new Job to open it in the **Designer**.

Add a Filter

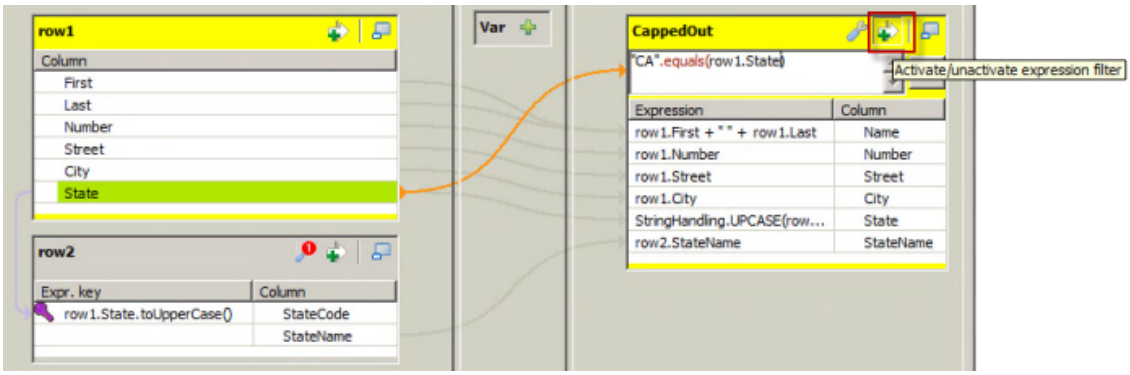
1. ADD A FILTER EXPRESSION

   Double-click the **tMap** component.

   Click the **Activate/unactivate expression filter** icon on top right of the *CappedOut* table ( ) and enter `"CA".equals(row1.State)` for the filter definition.

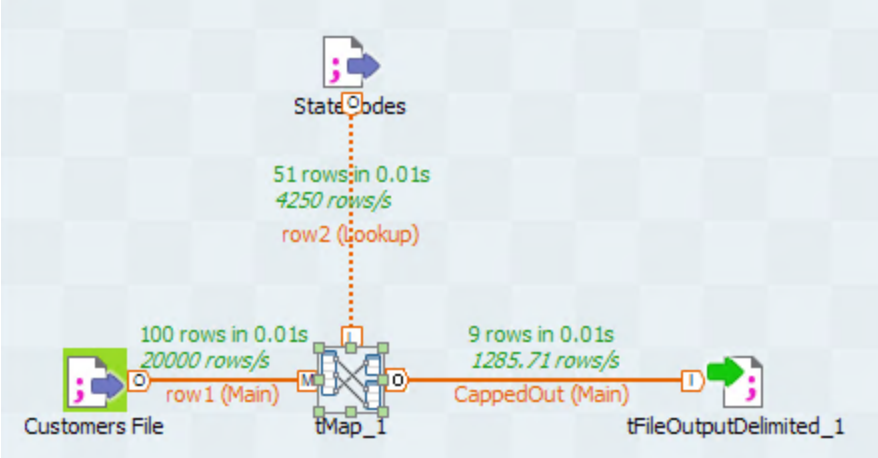
   Notice that an orange arrow is added for you once the filter expression is complete. This illustrates the mapping from the input table to the output table.

   Click **Ok** when done.



2. RUN THE JOB

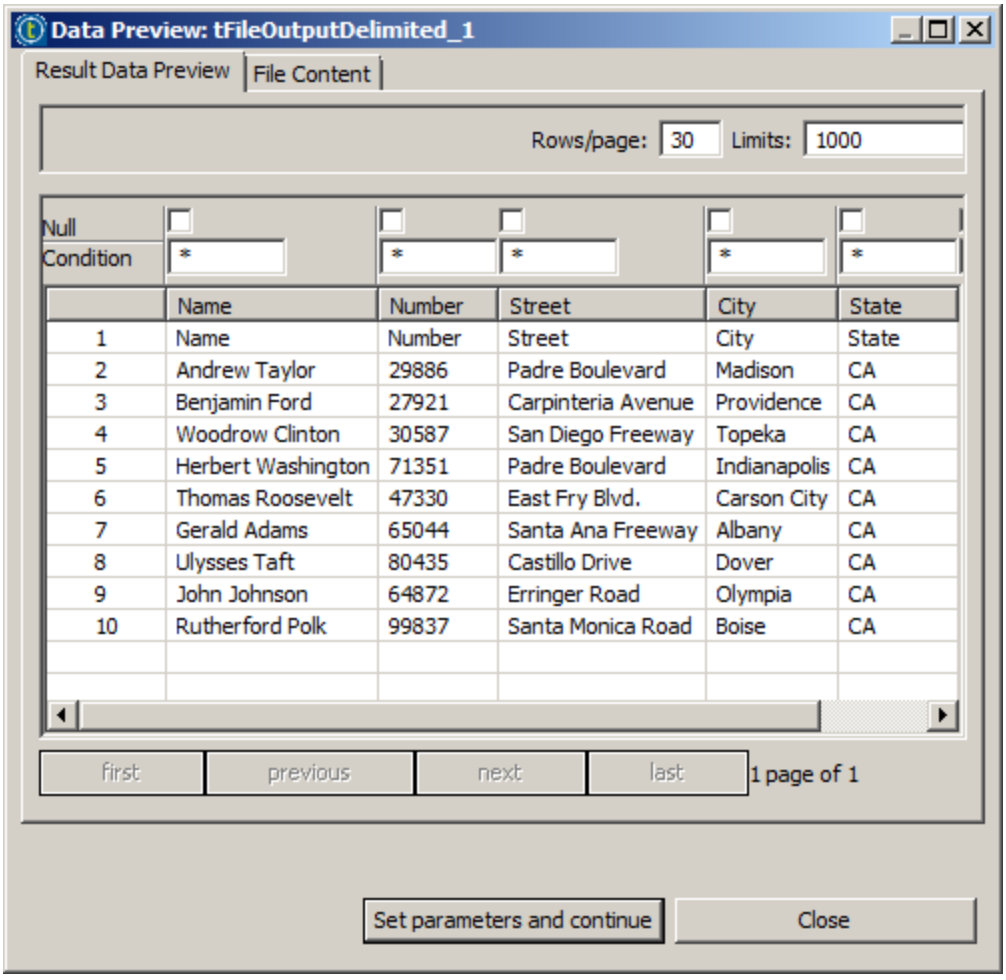
   Run the Job.



Notice that only 9 rows are now written to the output file.

3. EXAMINE THE OUTPUT

Right-click the output component **tFileOutputDelimited** and select **Data viewer**.



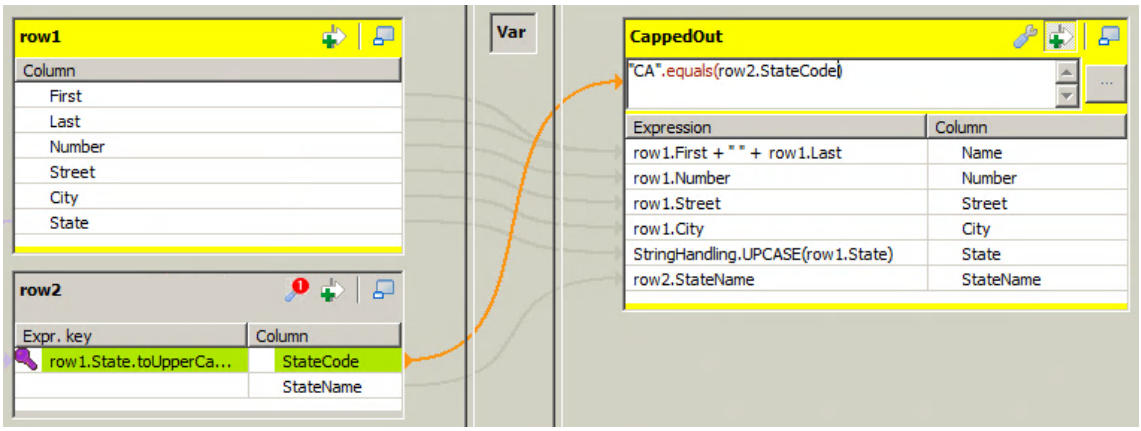
Notice the filtered data contains only records having uppercase letters for the *State*. This filter expression hasn't included the lowercase instances.

Click **Close**.

4. CORRECT THE FILTER EXPRESSION

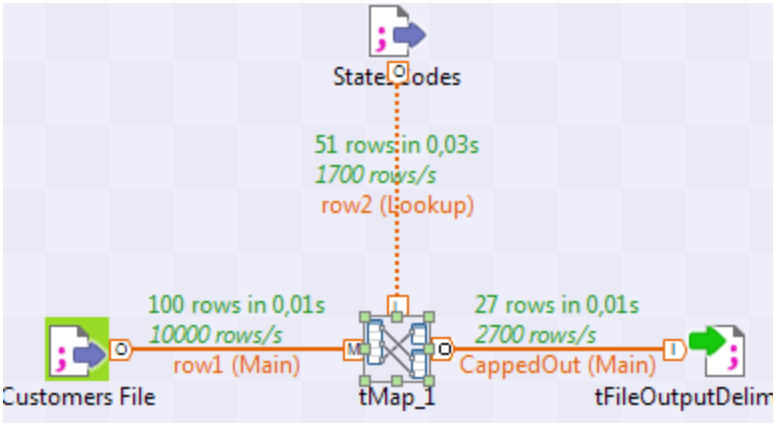
   Double-click **tMap** again and update the filter expression to `"CA".equals(row2.StateCode)`, which contains the state code in uppercase. Notice that once again the orange arrow is placed for you and identifies the data flow for the changed expression.

   Click **Ok**.



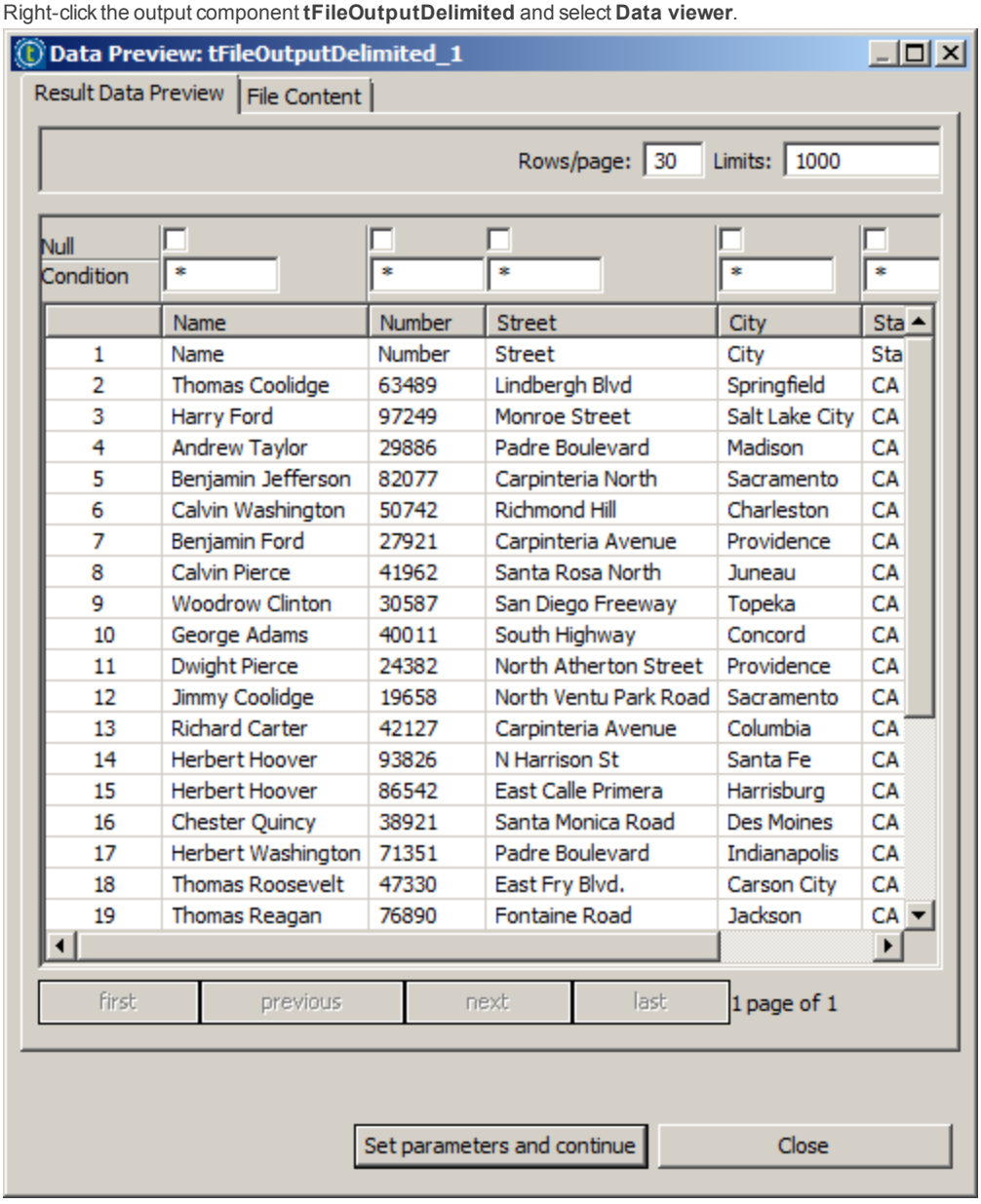
5. RUN THE JOB

   Run the Job again and you will see that the output now contains 27 rows.



6. CHECK THE OUTPUT

Right-click the output component **tFileOutputDelimited** and select **Data viewer**.



Next

Now that you have filtered data for one state code, you can learn how to create multiple outputs, filtering different states in the **tMap** component.

Using tMap for Multiple Filters

Overview

You will now learn how to filter on several states in one **tMap** component, following best practices. You will create three output files each containing a different state.

Duplicate Job

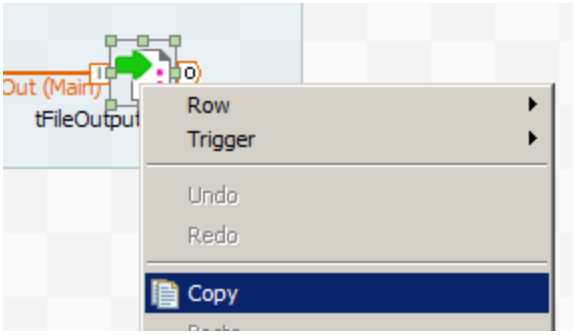
1. DUPLICATE THE JOB

   Duplicate the **Filter_States** Job and assign to it the name *Filter_States_tMap*. Open the new Job.

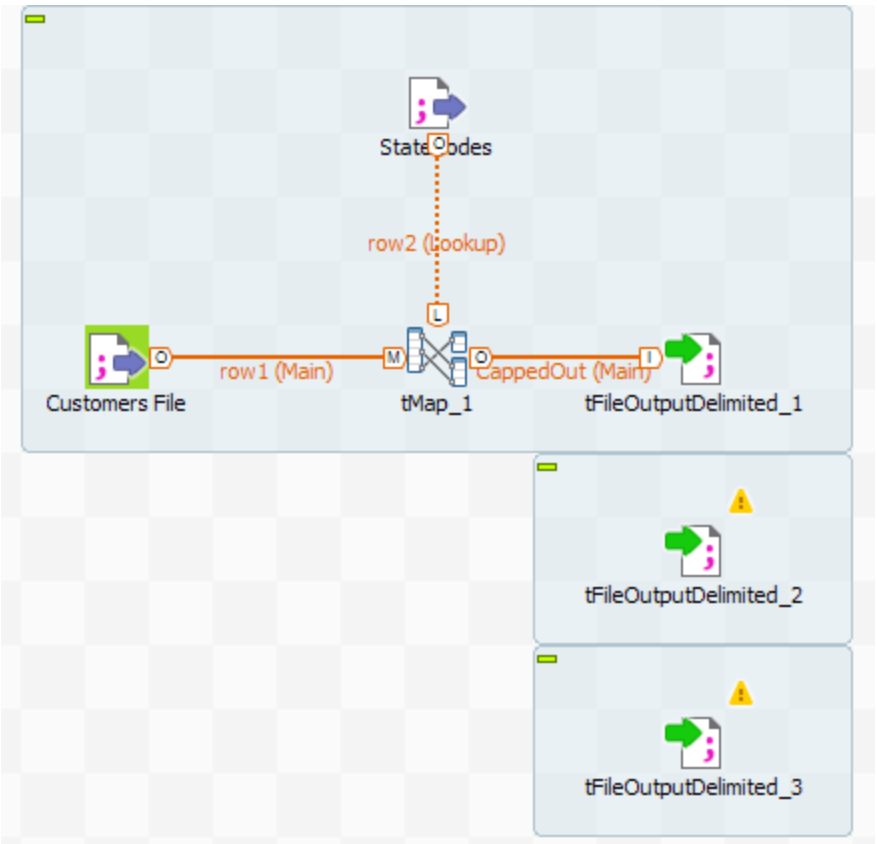
Edit Output

1. COPY THE OUTPUT COMPONENT

   Make a copy of the existing output component. Right-click on **tFileOutputDelimited** and select **Copy**.

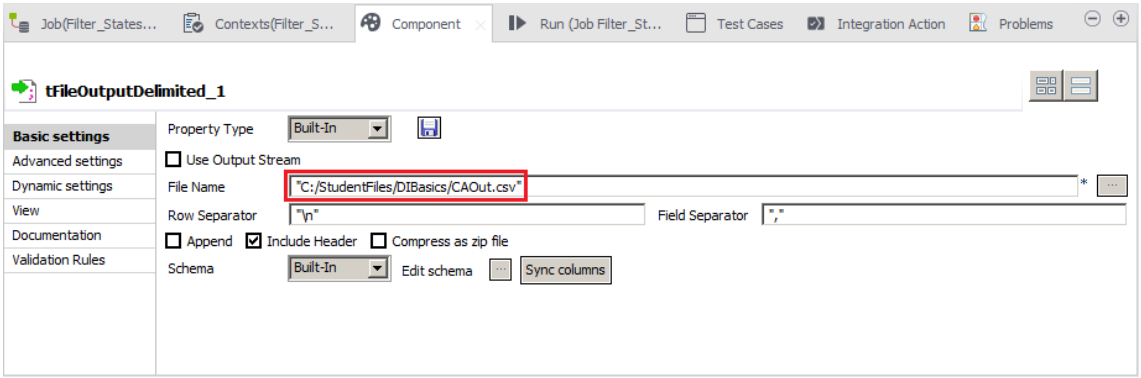


   Right-click below the output component in the canvas and click **Paste** to place the new **tFileOutputDelimited** component below the existing one. Repeat this step to create a third instance. The result should look like this.



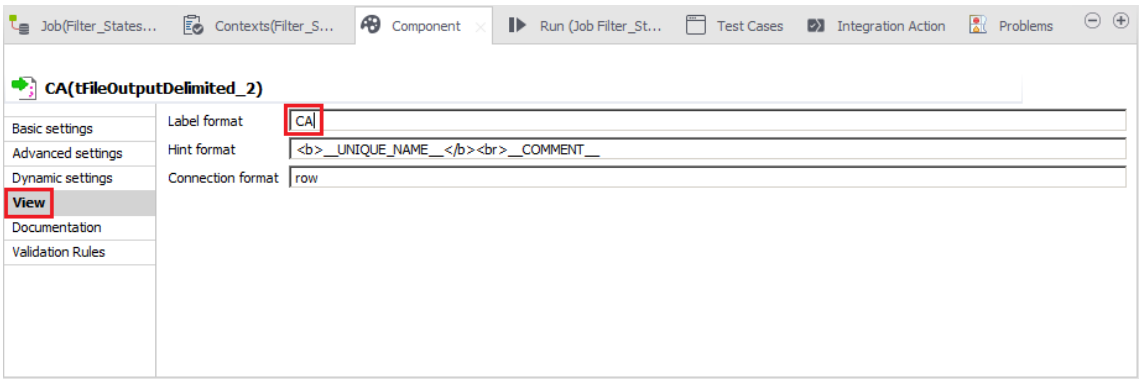
2. CHANGE THE OUTPUT FILE PATH FOR THE FIRST OUTPUT COMPONENT

   Recall from the previous section that the **tMap** component is configured to filter through entries whose *State* value is set to California. Now you will edit the settings for the first output component and change the output file name to reflect this. Double-click **tFileOutputDelimited_1** and change the **File Name** to *"C:/StudentFiles/DIBasics/CAOut.csv"*.



3. LABEL THE COMPONENT

   Click the **View** tab. Enter *CA* into the **Label format** box.



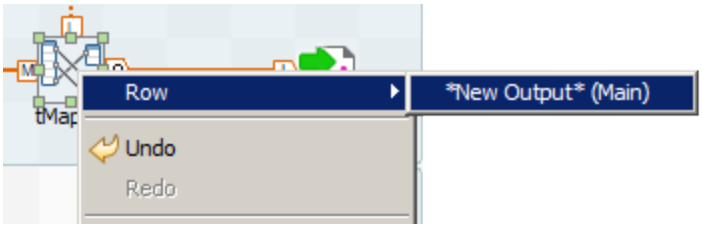
4. RECONFIGURE THE REMAINING OUTPUT COMPONENTS

   Repeat the process for the second output component, **tFileOutputDelimited_2**. Change the output file name to *NYOut.csv* and the label to *NY*.

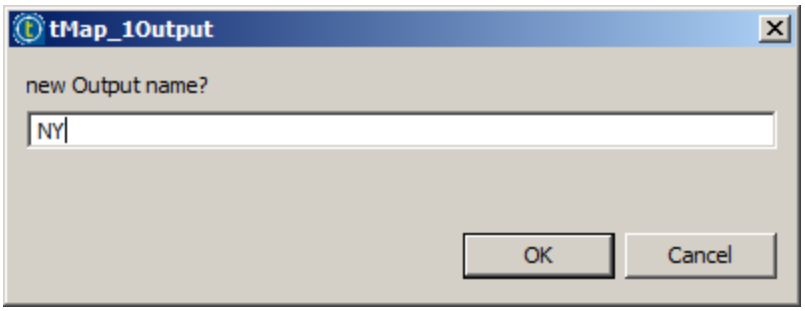
   Repeat again for the last output component, **tFileOutputDelimited_3**. Change the output file name to *AKOut.csv* and the label format to *AK*.

5. ADD A NEW OUTPUT ROW FROM THE tMap COMPONENT

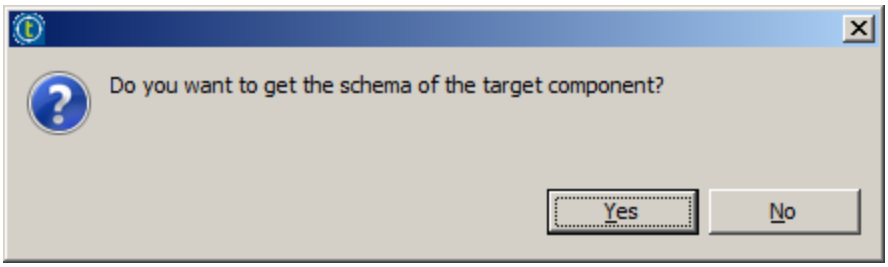
   Right-click the **tMap** component and select **Row > *New Output* (Main)**. Link the new output to the **NY** component by clicking on it.



   Enter an output name of *NY* and then click **OK**.

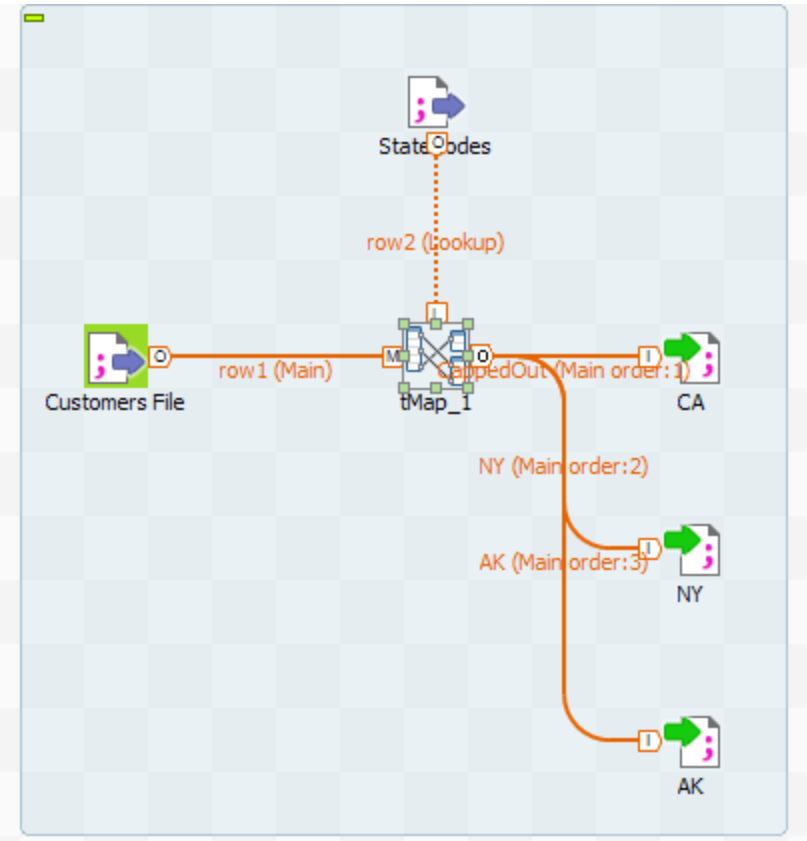


   Click **Yes** when prompted to get the schema of the target component.



6. CONNECT THE LAST OUTPUT COMPONENT

Repeat the previous step for the last output component **AK**. The Job should now look like this.

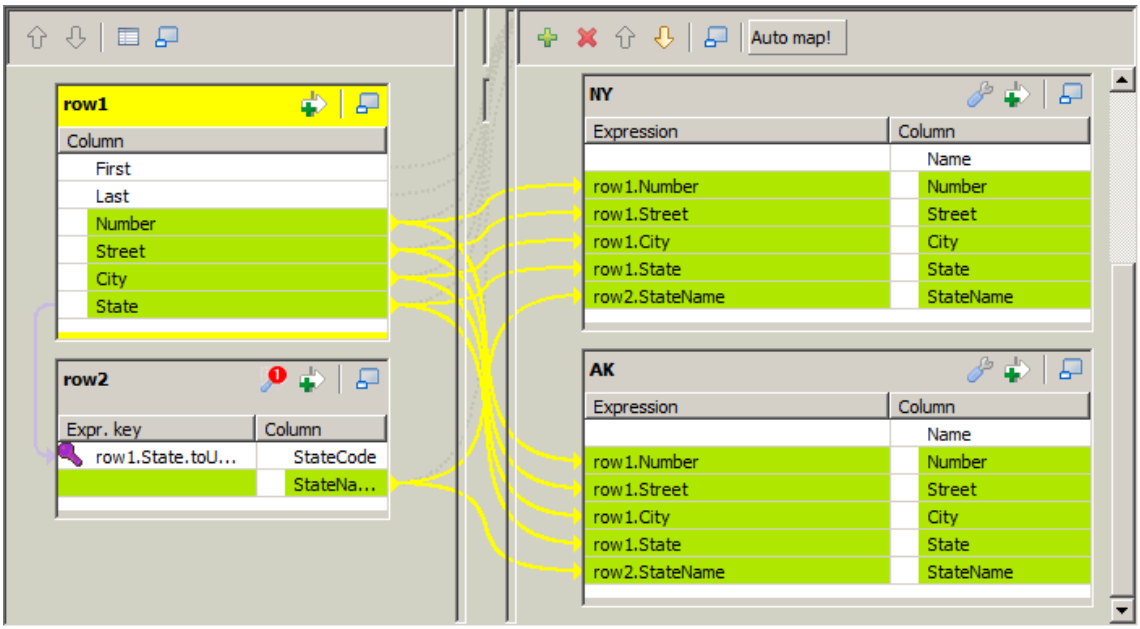


Edit tMap

1. MAP THE INPUT TO THE NEW OUTPUT

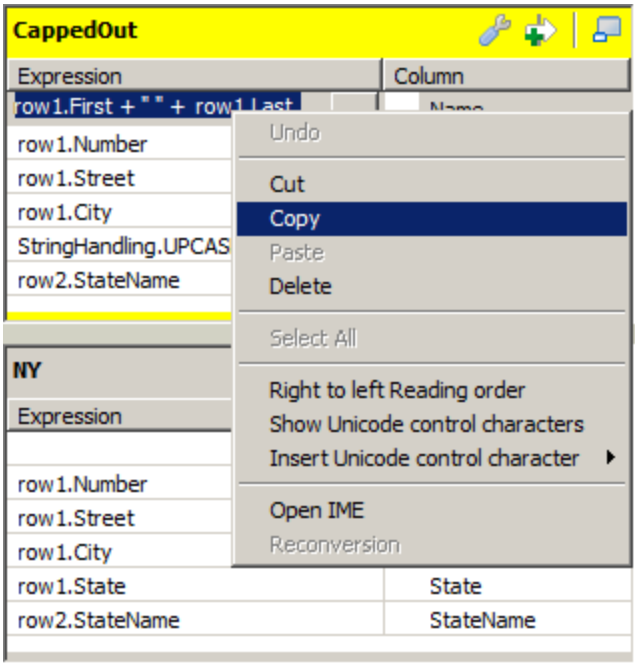
   Double-click the **tMap** component to open the mapping editor.

   Click the **Auto map!** button on the top right and the system will map the elements as follows.

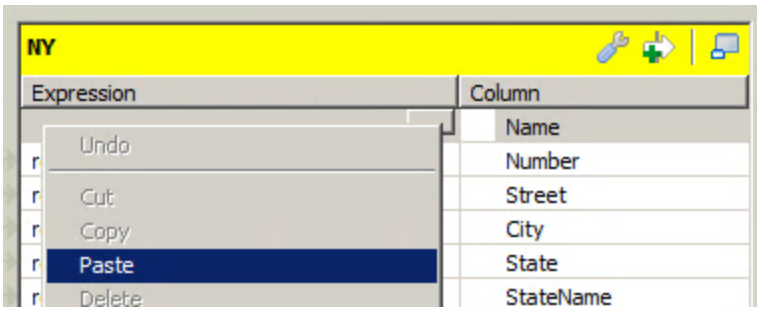


2. COPY AN EXPRESSION

You can copy and paste expressions. Select the expression for the *Name* column in the *CappedOut* schema, right-click, then select **Copy**.



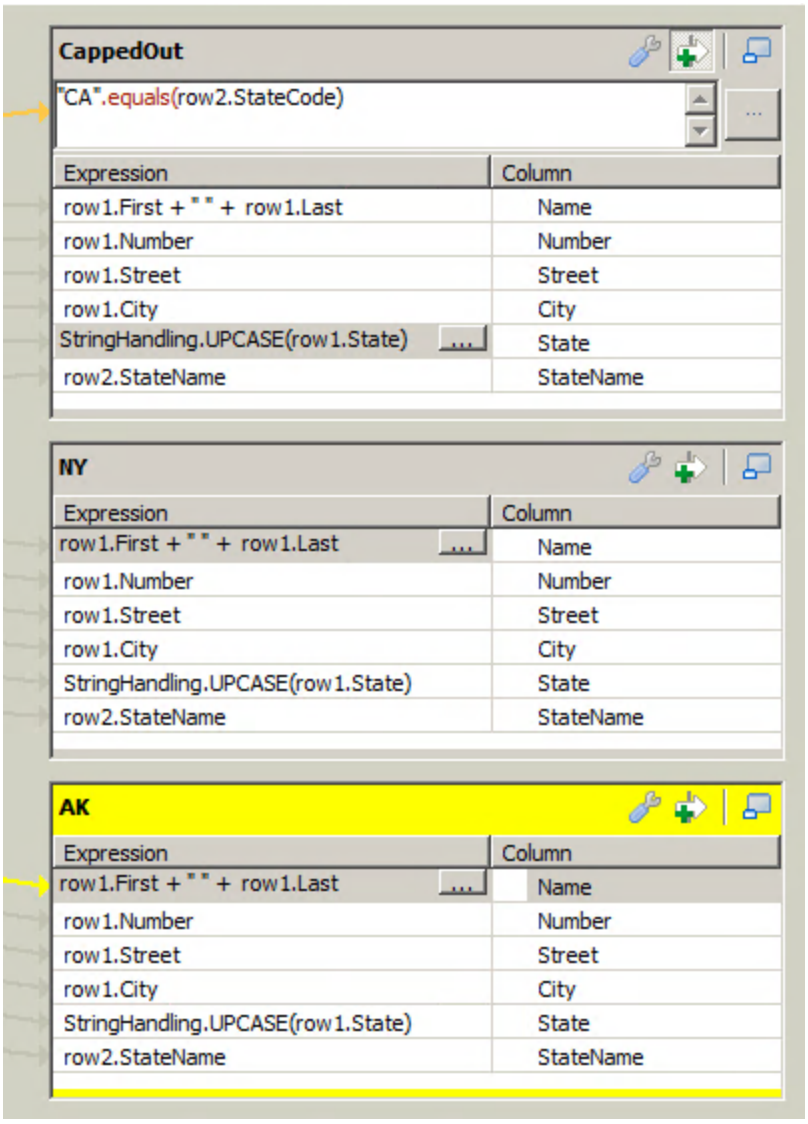
Select the expression for the *Name* column in the *NY* table, right-click, then select **Paste**.



Repeat this step for the *AK* schema.

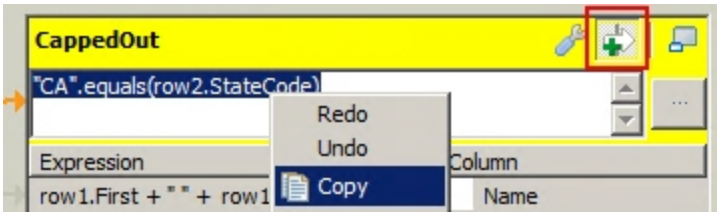
3. REPEAT FOR THE STATE EXPRESSION

   Now apply the same technique to copy the expression from the *State* column of the *CappedOut* table over to the *NY* and *AK* tables. When done, your output tables should resemble the following figure.

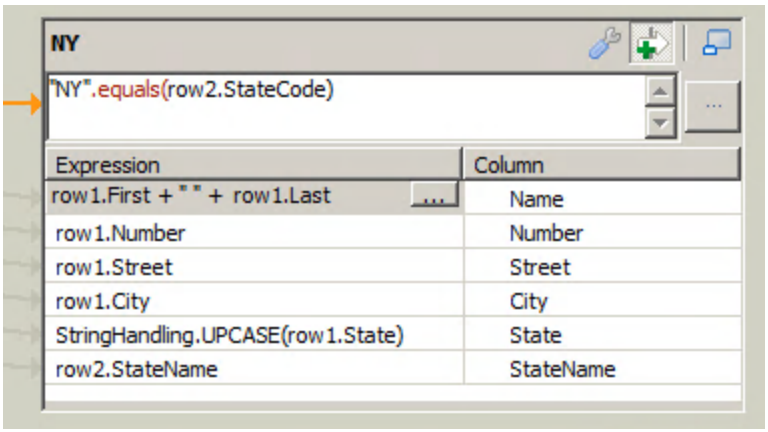


4. COPY THE FILTER EXPRESSION

If it is not already opened, click the **Activate / unactivate the expression filter** icon on the top right of the output schema **CappedOut** and copy the expression `"CA".equals(row2.StateCode)`.

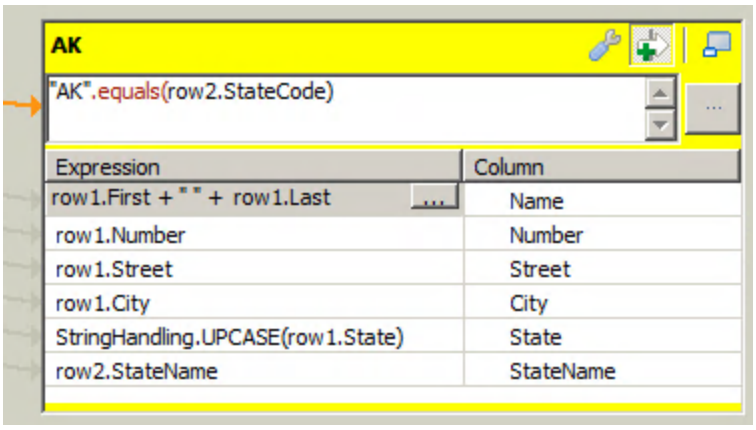


Click the **Activate / unactivate the expression filter** icon on the top right of the output schema *NY* and paste the expression to it, changing *"CA"* to *"NY"*, as shown below.



5. REPEAT FOR THE AK TABLE

   Make a similar change to the *AK* table.



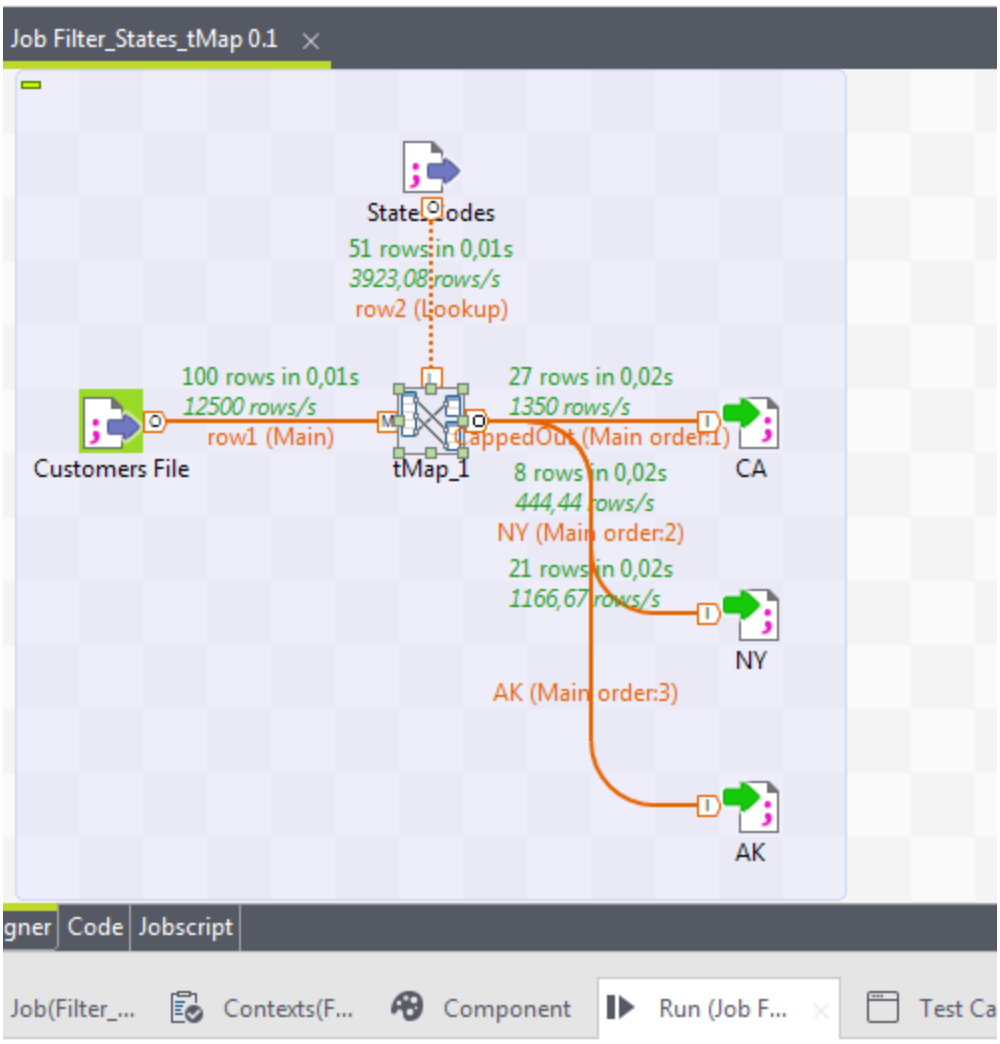
6. SAVE CHANGES

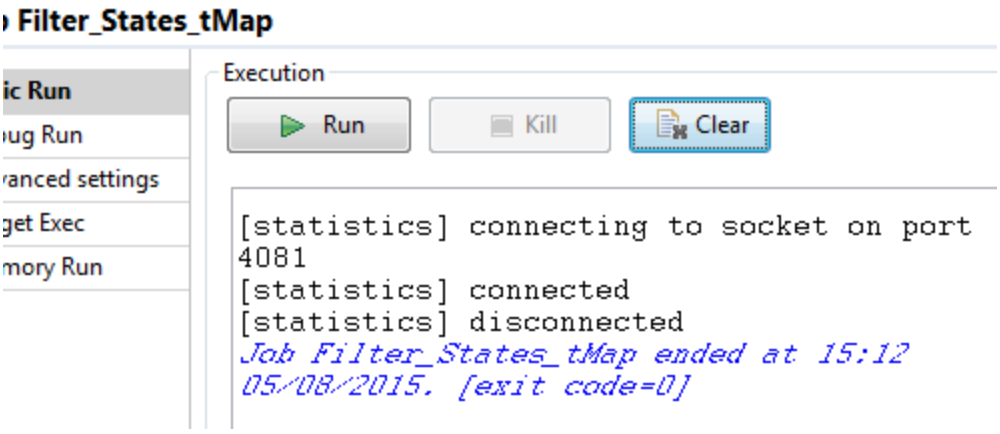
   Click **OK** to save your mapping changes.

Run Job

1. RUN THE JOB

   Run the Job and notice how the results are written to different output files based on the filters.





2. CHECK THE OUTPUT FILES

   Navigate to the *C:\StudentFiles\DIBasics* folder and examine the three output files. The *NYOut.csv* output is displayed here as an example.

TIP:
Recall that to view a text file, you can simply right-click it, then select **Edit with Notepad++**.

AKOut.csv | CAOut.csv | NYOut.csv

```
1  Name,Number,Street,City,State,StateName
2  Theodore Clinton,12292,San Marcos,Bismarck,NY, New York
3  Calvin Adams,52386,Lake Tahoe Blvd.,Montgomery,NY, New York
4  Martin Johnson,32782,Padre Boulevard,Springfield,NY, New York
5  Abraham Tyler,89260,Carpinteria North,Annapolis,NY, New York
6  Herbert Coolidge,65573,Cabrillo Highway,Salt Lake City,NY, New York
7  Gerald Eisenhower,12903,Jones Road,Albany,NY, New York
8  Abraham Wilson,83959,Hutchinson Rd,Nashville,NY, New York
9  Rutherford Harding,46743,Steele Lane,Boston,NY, New York
10
```

Next

You have now completed this lesson. It's now time to Wrap-Up.

Wrap-Up

In this lesson, you extended the previous Job to generate several output files depending on the state code. You learned how to filter data based on the value of a column, and you created several output tables using the **tMap** component. You used the **Automap!** functionality of the **tMap** component to map the new output tables. You copied and pasted filter expressions and changed the filter value. You defined output files for the new output tables, and created multiple filters wrote to the different output files.

Next step

Congratulations! You have successfully completed this lesson. To save your progress, click **Check your status with this unit** below. To go to the next lesson, on the next screen, click **Completed. Let's continue >**.

LESSON 5

Using Context Variables

This chapter discusses:

Using Context Variables

Lesson Overview

Usually, when you are developing a Talend Data Integration project, you are not working in the final production environment. Because of this, some of the configuration parameters may be different in your development or testing environment than in production. A common example of this is the location of files within the file system.

In this lesson, you will create variables to supply different values for different environments.

Objectives

After completing this lesson, you will be able to:

- Create Built-in context variables, specific for a Job
- Run a Job using Built-in context variables
- Create Repository context variables, available for all Jobs
- Run a Job using Repository context variables

Next Step

The first step is to create a variable for a directory location.

Understanding and Using Context Variables

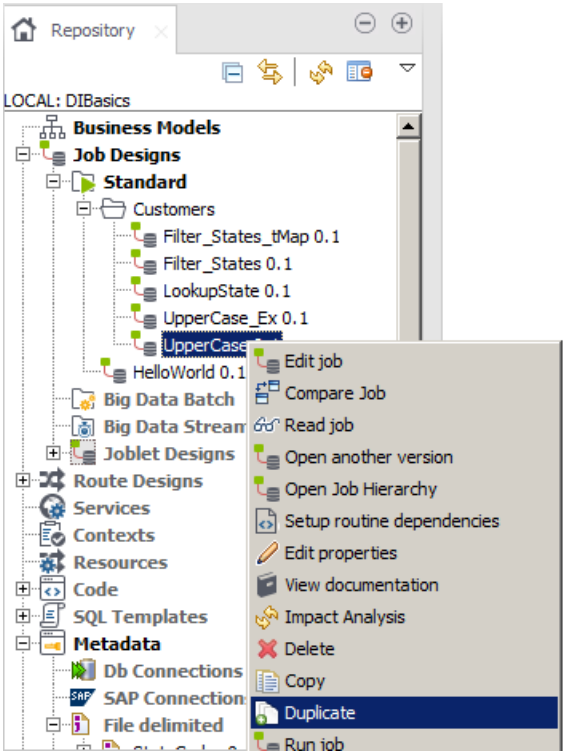
Overview

With Talend Studio, you can define contexts and context variables that allow you to change the value of different configuration parameters by running a Job in a specific context.

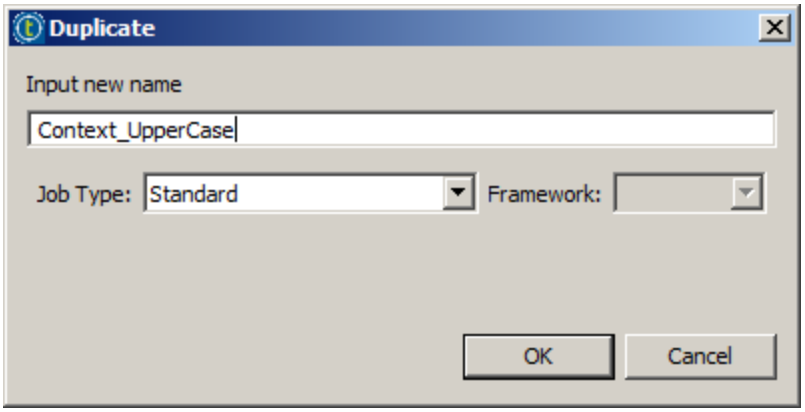
Create a Variable

1. DUPLICATE A JOB

   Right-click **Repository > Job Designs > Standard > Customers > UpperCase** and select **Duplicate**.



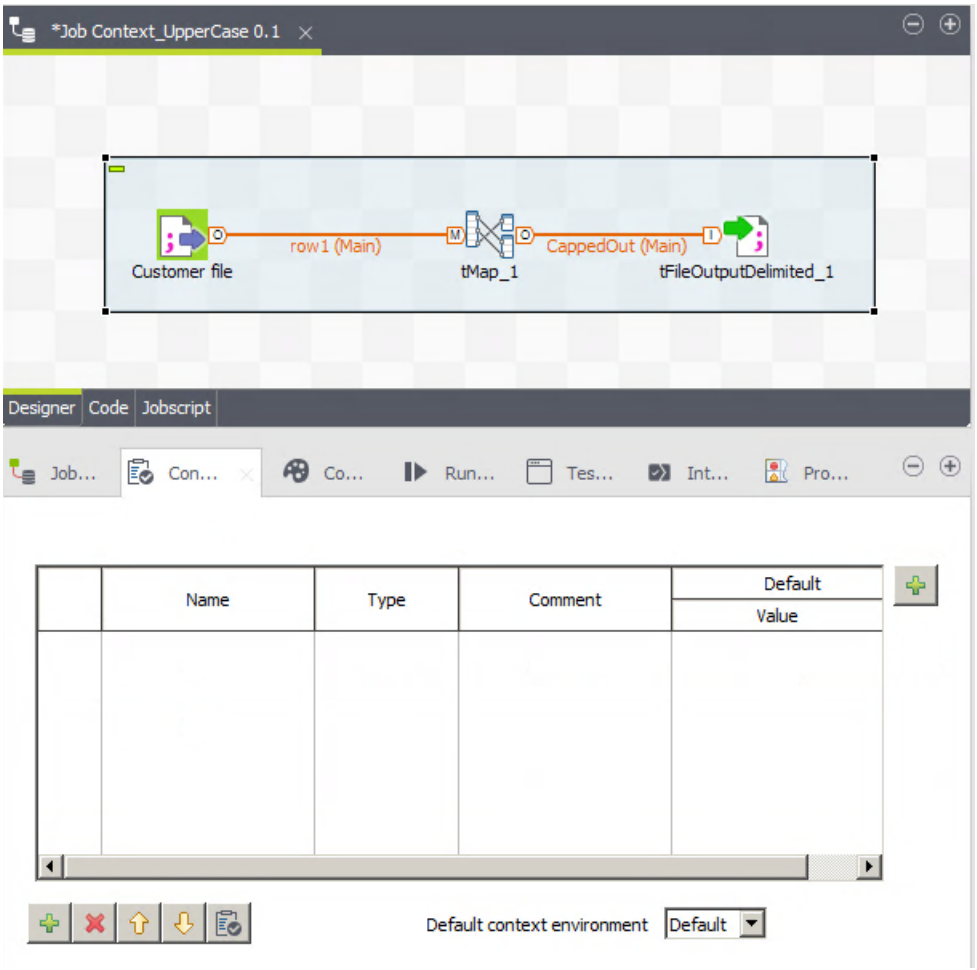
   Enter *Context_UpperCase* and click **OK**.



Open up the new Job by double-clicking it.

2. OPEN THE CONTEXTS VIEW FOR THE JOB

With the newly created Job open, click the **Contexts** tab.



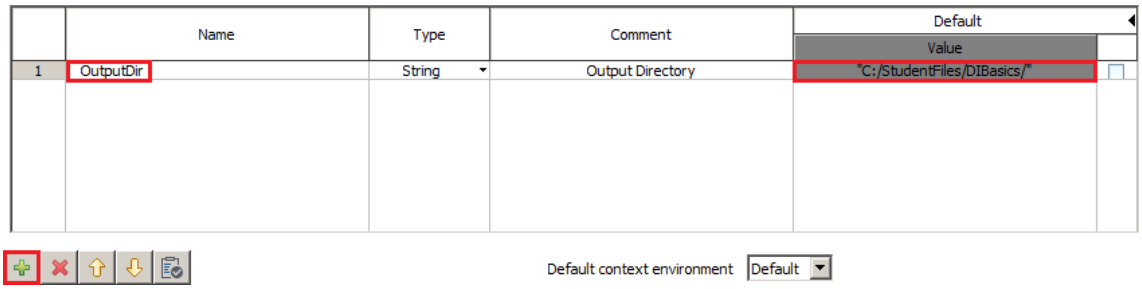
This is where you create context variables and specify values for those variables. When you run a Job, you run it within a specific context. If you add additional contexts and variables, you can specify the value to use for those variables by changing the context in which you run a Job.

3. CREATE A CONTEXT VARIABLE

Defining a variable will allow you to specify the location of the output directory. Click the **Add** button (at the lower left corner of the table) to add a new context variable. Populate the columns as follows:

- Set the **Name** of the variable to *OutputDir*
- Specify a **Comment** that describes the variable
- Set the **Default Value** to *"C:/StudentFiles/DIBasics/"*

WARNING:
When entering the **Default Value**, be sure to include the opening and closing quotation marks, as well as the trailing slash.

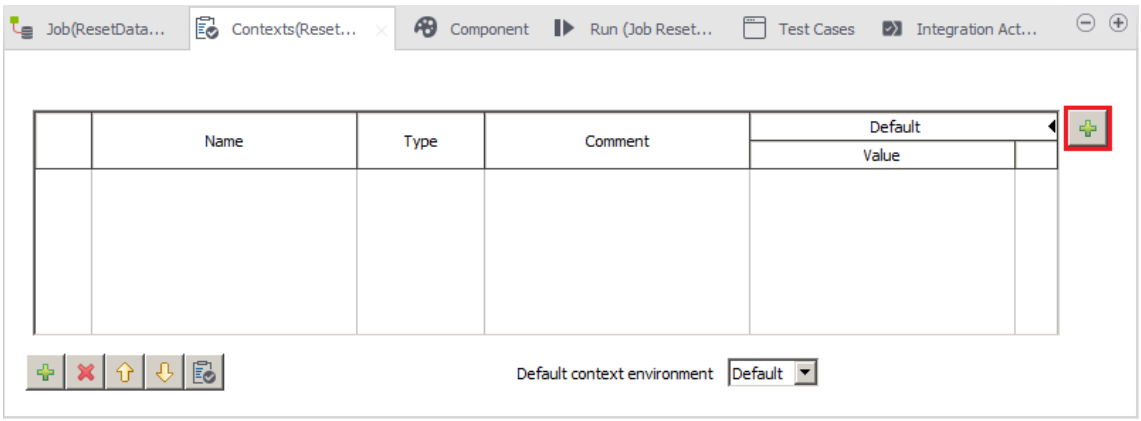


You are creating a context variable that will allow you to easily override the location where output files should be stored. In order for a context variable to be useful though, you need to define contexts.

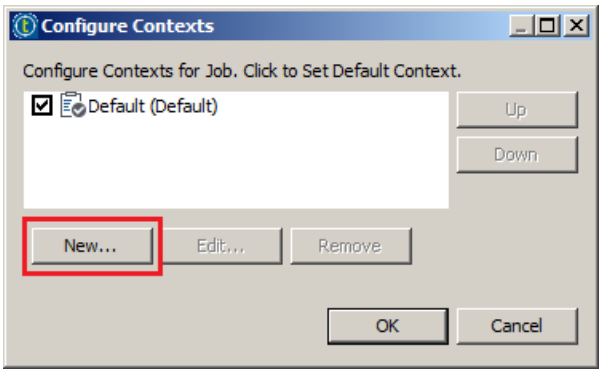
Create a new Context

1. CREATE A NEW CONTEXT

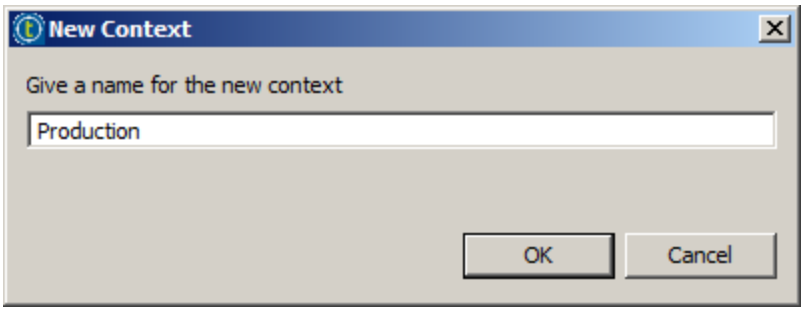
   Click the **Configure Contexts** button (at the top right corner of the **Contexts** View) to manage contexts.



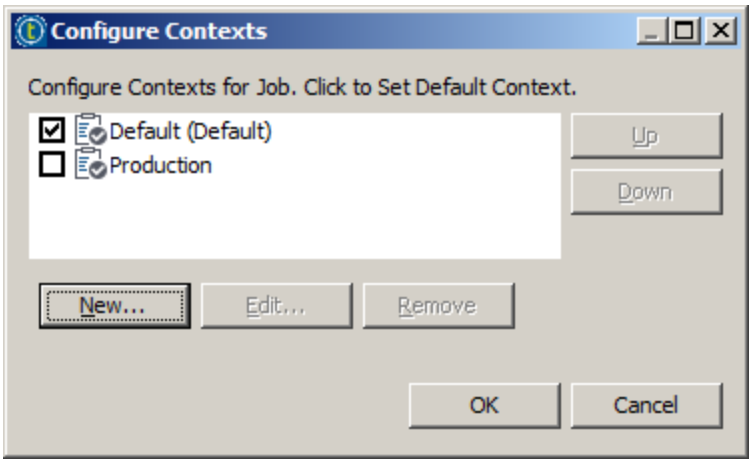
   In the **Configure Contexts** window that appears, click **New**.



Enter *Production* and click **OK**.



The new context appears in the list. When running a Job, Talend Studio uses the context you select here (marked with a check) as the default context. While building out your projects, you might need to create an additional context for testing. Usually you will create one context for each environment. Click **OK**.



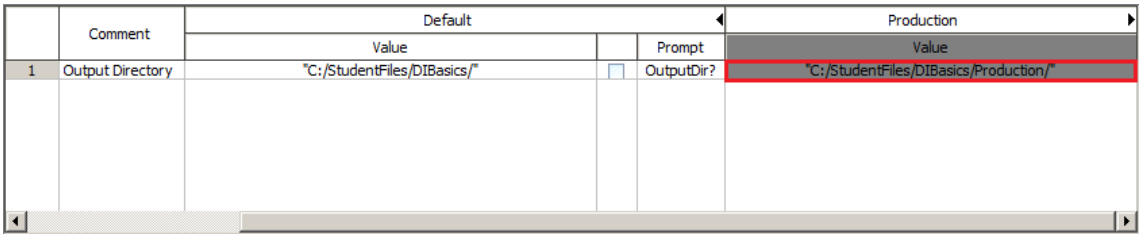
The new context appears as a new column in the **Contexts** view, with variable values copied from the default context.

2. CHANGE VARIABLE VALUE FOR THE NEW CONTEXT

   Click the value under **Production** and change it to *"C:/StudentFiles/DIBasics/Production/"*.

   WARNING:

   Again, when defining the path values, be sure to include the opening and closing quotation marks, as well as the trailing slash. Omitting these is a common mistake.



	Comment	Default		Production
		Value	Prompt	Value
1	Output Directory	"C:/StudentFiles/DIBasics/"	OutputDir?	"C:/StudentFiles/DIBasics/Production/"

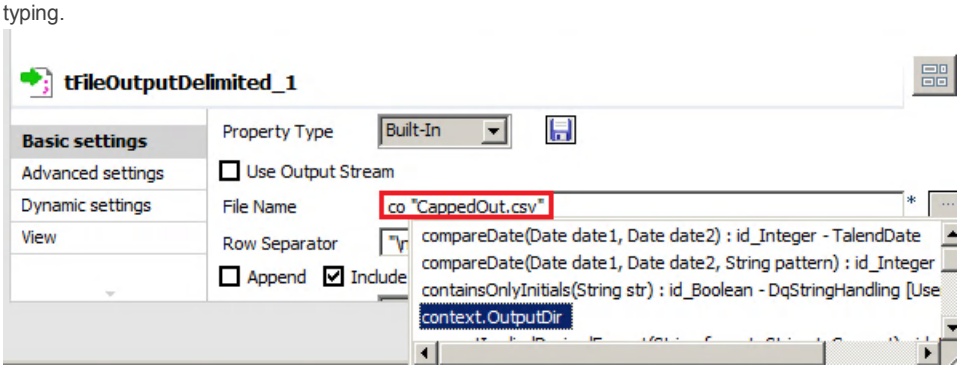
Now you have a context variable called *OutputDir* that has a different value depending on the context.

Use the Context Variable

1. CONFIGURE THE OUTPUT COMPONENT TO USE THE CONTEXT VARIABLE

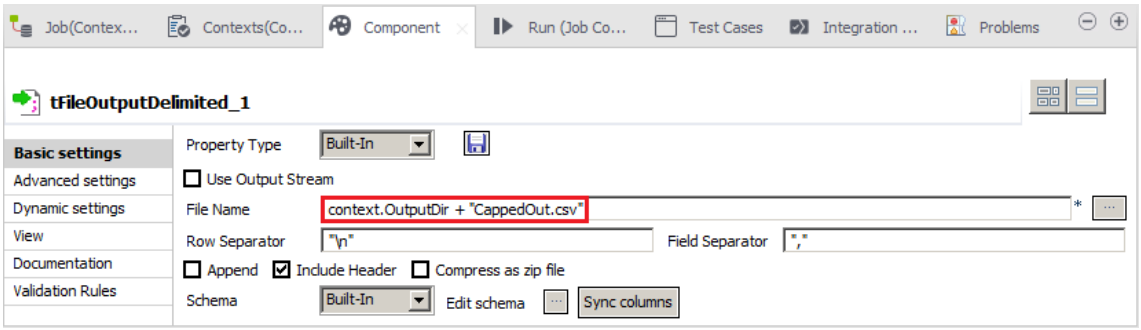
   Double-click the **tFileOutputDelimited_1** component to open the **Component** view.

Currently, the **File Name** stores a hard-coded value which never changes. Delete the path component, keeping only the file name *"CappedOut.csv"*. In front of the file name, start typing *context.OutputDir*. Press **Ctrl + Space** after a few characters to take advantage of the auto-completion feature. A list of suggestions appear that is specific to the box in which you are typing.



Double-click the context variable *context.OutputDir* to insert the context variable reference. This is actually code for accessing the value of the context variable you defined earlier.

Insert a plus sign to turn this into a valid expression. The final **File Name** should read `context.OutputDir + "CappedOut.csv"`.



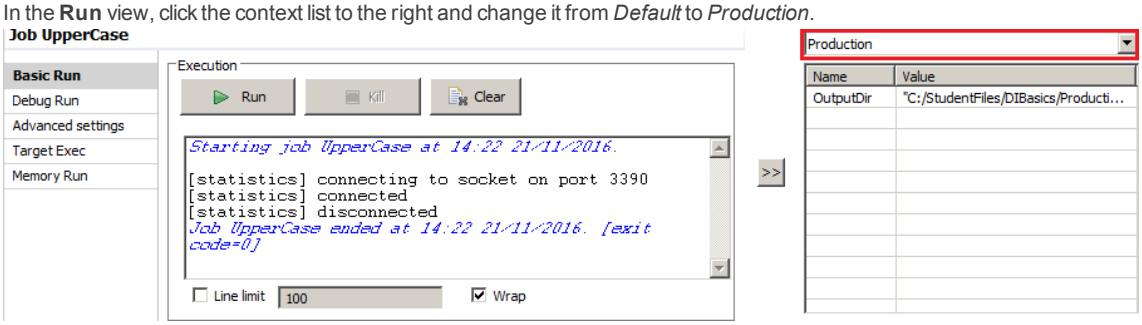
2. RUN THE JOB

   Run the Job and examine the output file. The output file *CappedOut.csv* should have been overwritten under the *C:/StudentFiles/DIBasics* folder, evidenced by a refreshed time stamp.

Change Contexts

1. CHANGE THE CONTEXT IN THE RUN VIEW

   In the **Run** view, click the context list to the right and change it from *Default* to *Production*.



   Notice that the value of the *OutputDir* variable changes.

2. RUN THE JOB AND CHECK THE RESULTS

   Run the Job again and then examine the output file that appears in the new folder *C:/StudentFiles/DIBasics/Production*.

By changing the context in which you ran the Job, the configuration parameter changed because of the different value of the context variable, affecting the output location.

NOTE:
If you forgot to include the final slash in the definition of the *Production* context Value, then a new file *ProductionCappedOut.csv* gets created in the *DIBasics* folder.

Next

Now you will create a Repository context variable so that you can use it in any Job.

Using Repository Context Variables

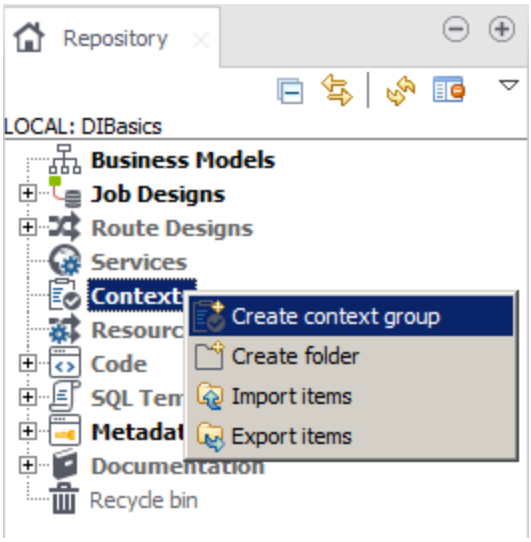
Overview

In the previous exercise, you created a built-in context variable that was only available to a single Job. In this exercise, you will create a Repository context variable that can be reused by any Job in the project.

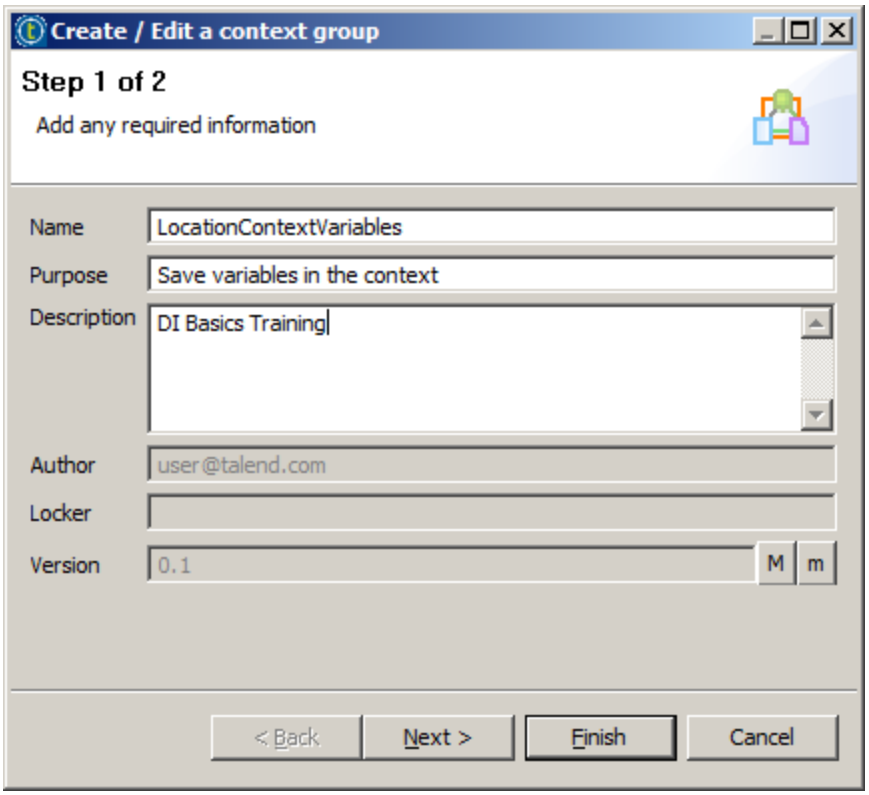
Create Repository Context Variable

1. CREATE A CONTEXT GROUP IN THE REPOSITORY

   Context groups enable you to organize your context variables. Create one by right-clicking on **Repository > Contexts** and selecting **Create context group**.

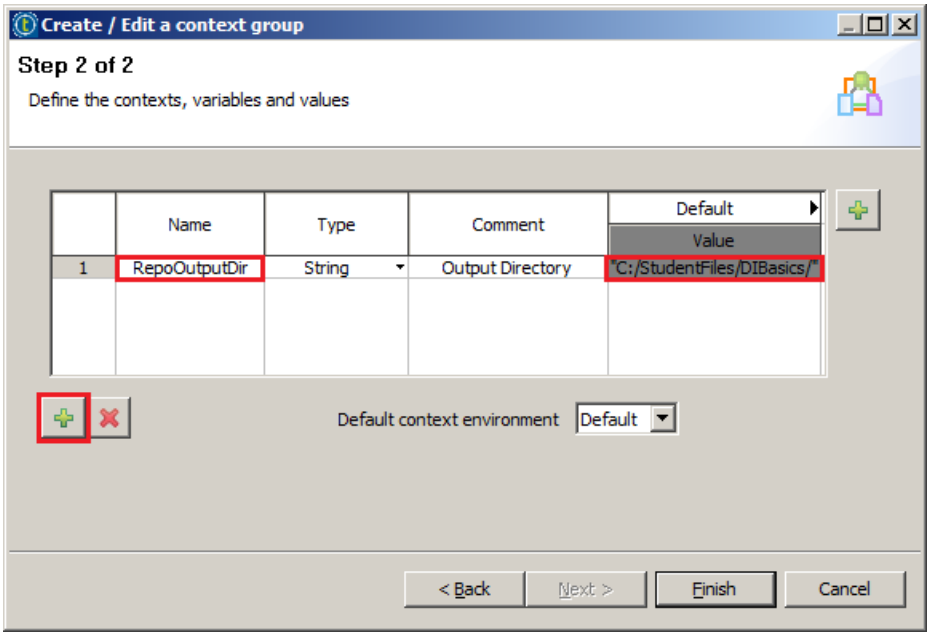


   Enter *LocationContextVariables* for the **Name** and then enter appropriate descriptive information into the **Purpose** and **Description** boxes, then click **Next**.



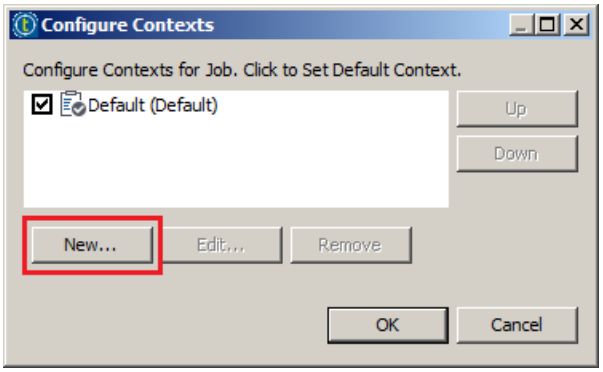
2. ADD A NEW CONTEXT VARIABLE

Click the **Add** button (at the bottom left of the table) to add a new variable. Name it *RepoOutputDir* and set the default value to *"C:/StudentFiles/DIBasics/"*.

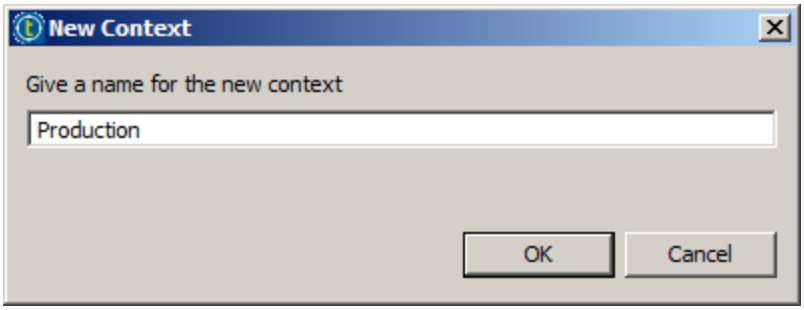


3. ADD A NEW CONTEXT

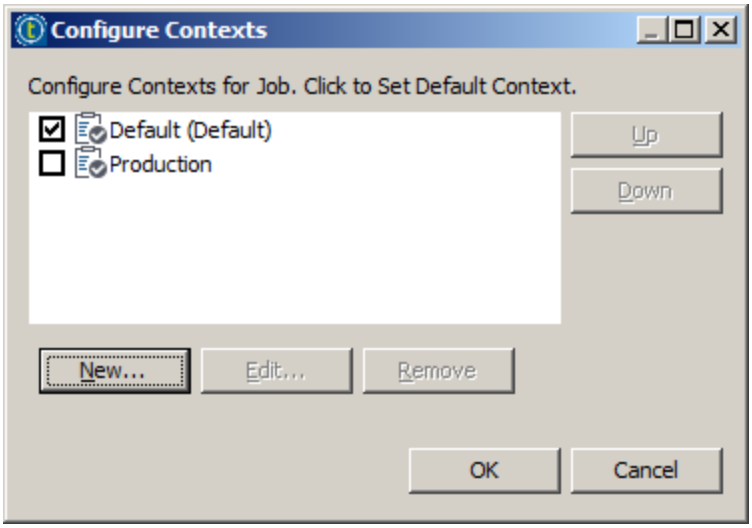
Click the **Configure Contexts** button (at the top right of the table). In the **Configure Contexts** window that appears, click **New**.



Name the context *Production* and then click **OK**.

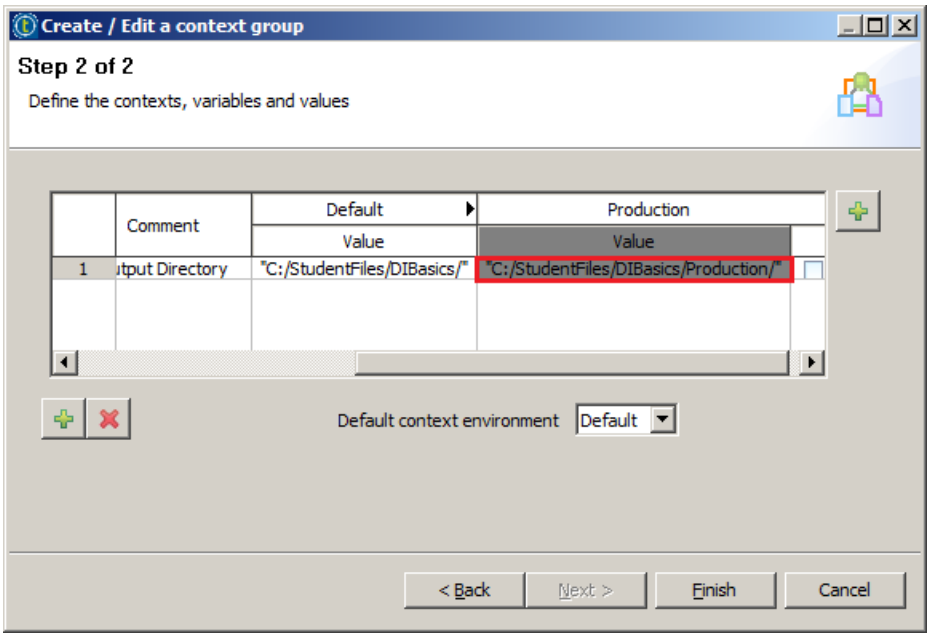


Back in the **Configure Contexts** window, click **OK**.

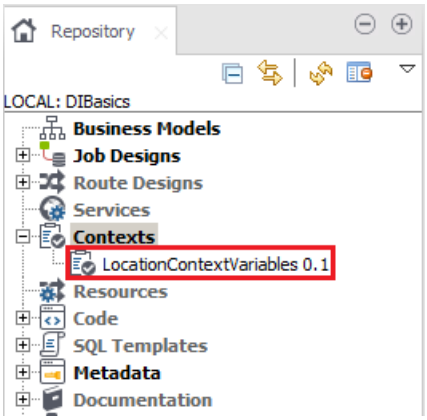


4. CONFIGURE THE NEW CONTEXT

   Back in the **Create/Edit a context group** window, set the value for *RepoOutputDir* in the *Production* context to *"C:/StudentFiles/DIBasics/Production/"*. When done, click **Finish**.



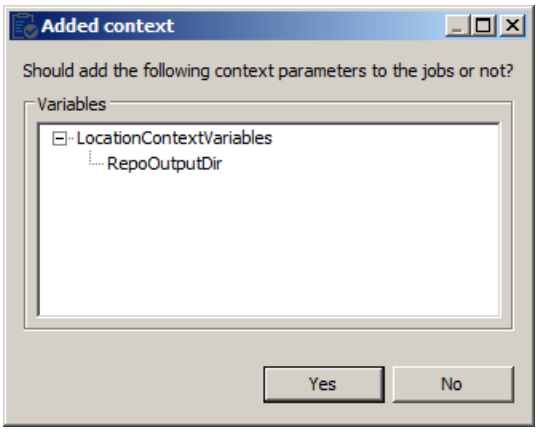
The new group appears in the **Repository**, containing a single variable.



Add Repository Context Variables to a Job

1. ADD A CONTEXT GROUP TO A JOB

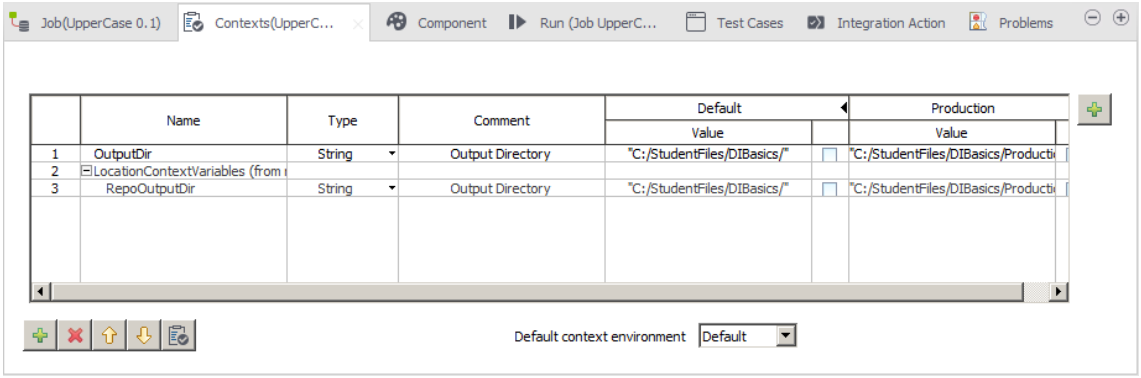
   In the **Repository**, click **Contexts > LocationContextVariables**, drag it to the **Designer**, and drop it. In the **Added context** window that appears, click **Yes**.



NOTE:

You can also achieve the same result by clicking the **Contexts** button ( ) below the **Variables** table in the **Context** view.

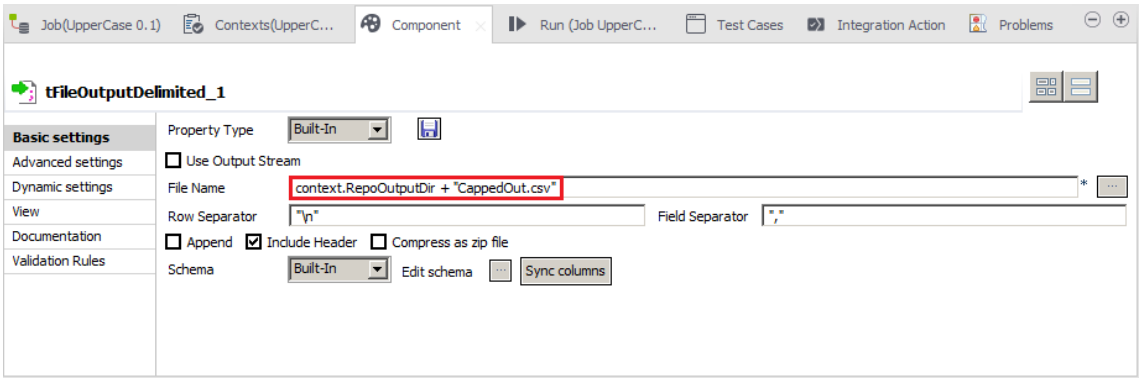
The group appears in the **Contexts** view. The values are the ones you set in the **Repository**. You cannot change them from here, although you can edit them from the **Repository**.



Use Repository Context Variables

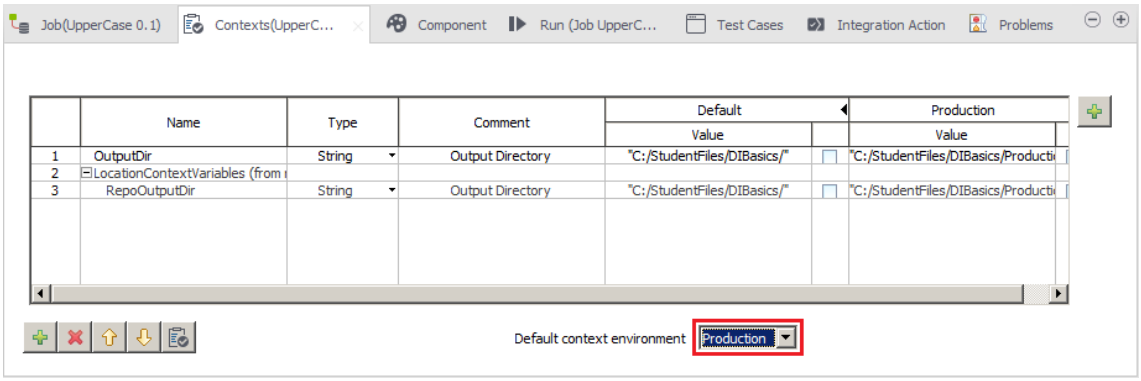
1. CHANGE THE OUTPUT PATH TO USE THE NEW CONTEXT VARIABLE

   Double-click the **tFileOutputDelimited_1** component to open the **Component** view. Update the expression in the **File Name** box to `context.RepoOutputDir + "CappedOut.csv"`.



2. SET THE DEFAULT CONTEXT

   Click the **Contexts** tab again and select *Production* for the **Default context environment**.



3. RUN THE JOB

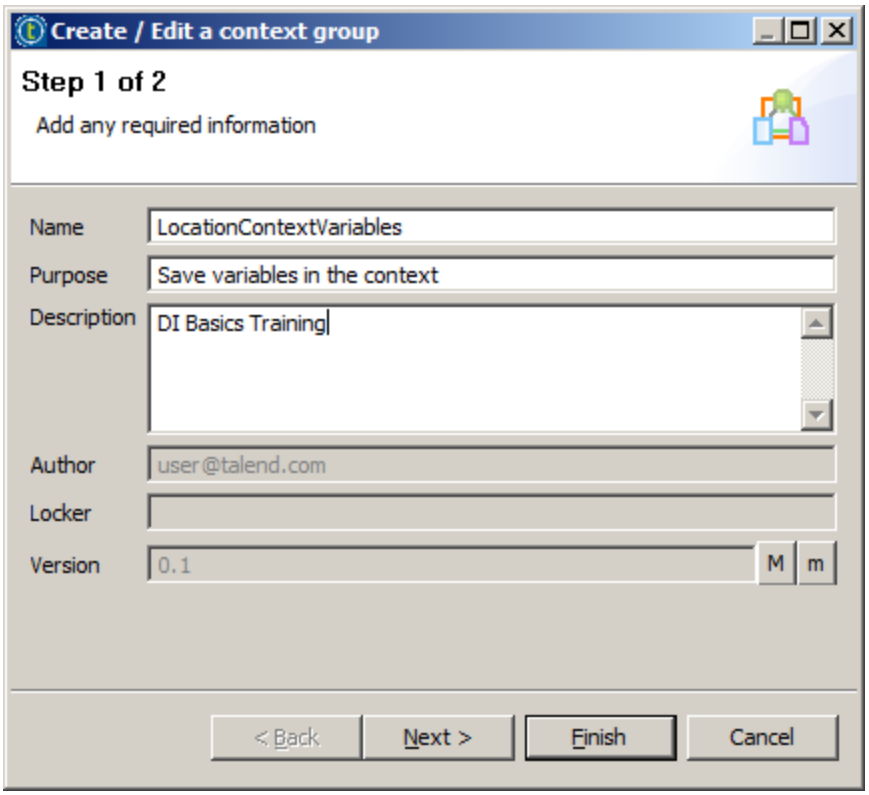
   Run the Job. The file created during this run of the Job appears in the *C:/StudentFiles/DIBasics/Production* folder because of the default context variable, which was set to *Production*.

Modify Repository Context Variables

1. OPEN THE REPOSITORY CONTEXT GROUP

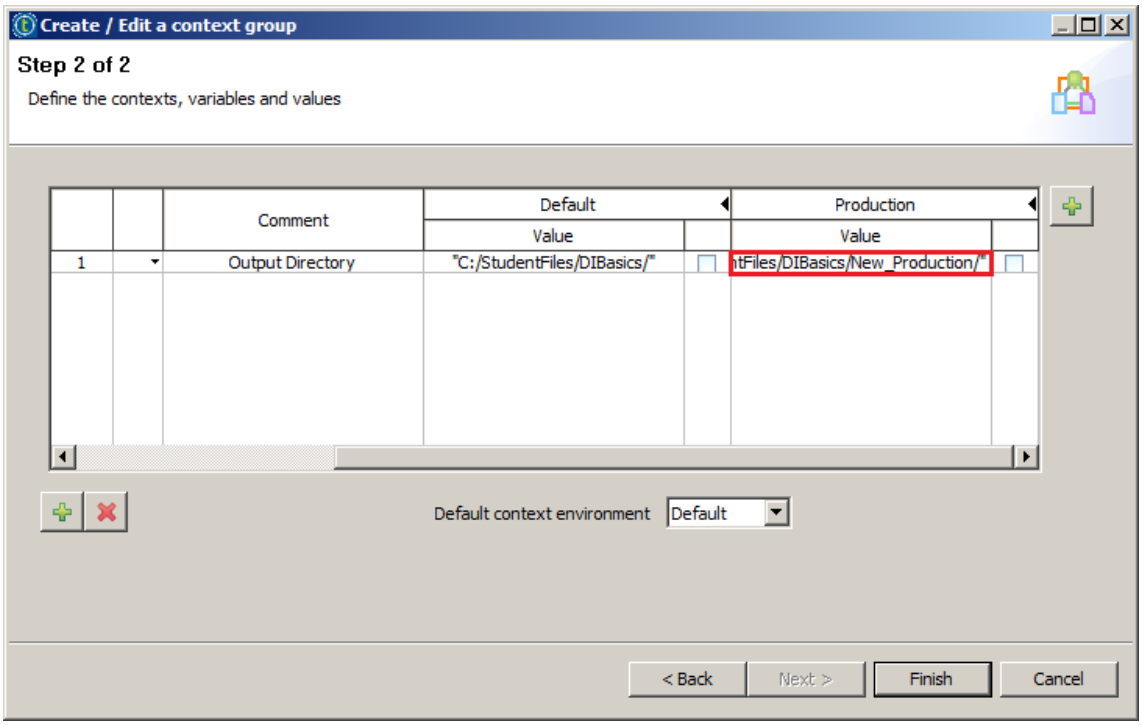
   Double-click **Repository > Contexts > LocationContextVariables** to open the group for edit.

   In the **Create/Edit a context group** window that opens, click **Next**.

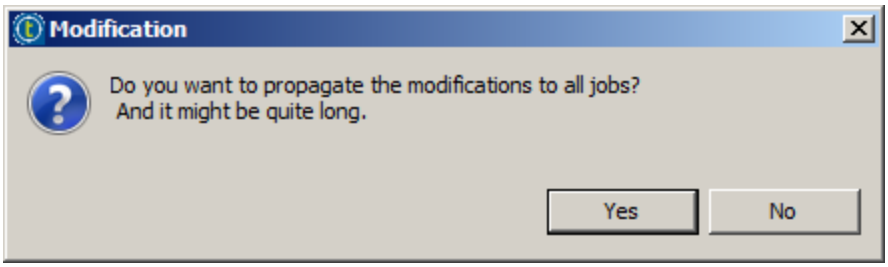


2. MODIFY THE VARIABLE VALUE

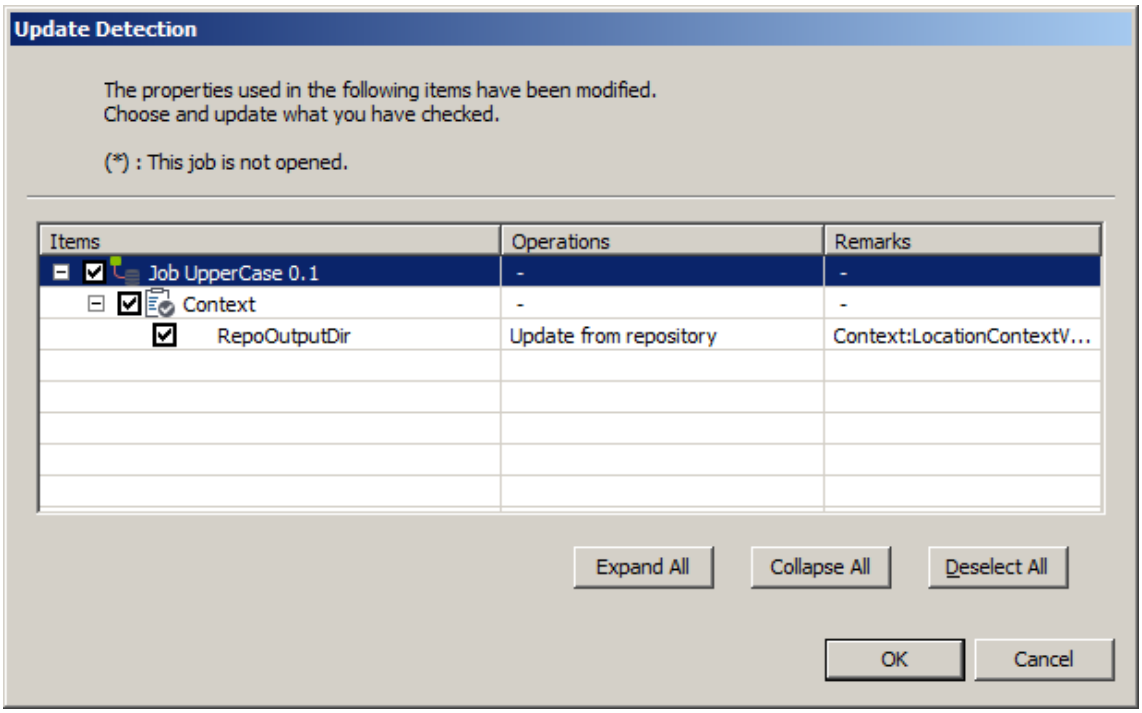
   Change the value under **Production** to *"C:/StudentFiles/DIBasics/New_ Production/"* and click **Finish**.



When prompted to propagate the change to all Jobs that use the Repository variable, click **Yes**.



The **Update Detection** window appears. This lists the items in a particular Job that use this context variable. If you had multiple Jobs using the variable, you would see all of them listed so that you could choose which one to change. Leave all items selected and click **OK**.



3. RUN THE JOB

   Run the Job using the **Production** context again. Note that *CappedOut.csv* file is now created in the *New_Production* folder, as you would expect from changing the value of the context variable.

Next

You have now completed this lesson. Work through the exercises to reinforce what you learned.

Challenges

Overview

Complete the following exercise to further explore context variables. See Solutions for possible solutions to this exercise.

New Context and Variable

Create a new Repository context variable in the existing context group for the input directory of the customer data file. Modify the component to use that variable. Run the Job and check the results.

Next

You have now finished this lesson. It's time to Wrap-Up.

Solutions

Overview

These are possible solutions to the challenges. Note that your solutions may differ and still be valid.

New Context and Variable

1. Right-click **LocationContextVariables** in the **Repository** and then click **Edit context group**.
2. Click **Next**.
3. Click **Add**.
4. Change the name of the new variable to *RepoInputDir*.
5. Click **Values as Table** tab.
6. Add values for the two contexts. Make sure the default value is *"C:/StudentFiles/DIBasics/"*.
7. Click **Finish**.
8. In the **Contexts** view of the Job, click the **Contexts** button and then add the new variable (click **Select All** followed by **OK**).
9. In the **Component** view for **CustomersFile**, change the value of the **File Name** box to `context. RepoInputDir + "Custs.csv"`
10. Run the Job again and check that the result is the same as before.

Next

You have now completed the exercises. It's now time to Wrap-up.

Wrap-Up

In this lesson, you created contexts and context variables so that you could change the value of specific parameters based on the context in which you run a Job. Typically this is set up for the various environments needed for your company processes, such as development, test and production.

You should be able to differentiate between built-in context variables (available only to a specific Job), and Repository context variables (available for all Jobs in the project).

Next step

Congratulations! You have successfully completed this lesson. To save your progress, click **Check your status with this unit** below. To go to the next lesson, on the next screen, click **Completed. Let's continue >**.

LESSON 6

Error Handling

This chapter discusses:

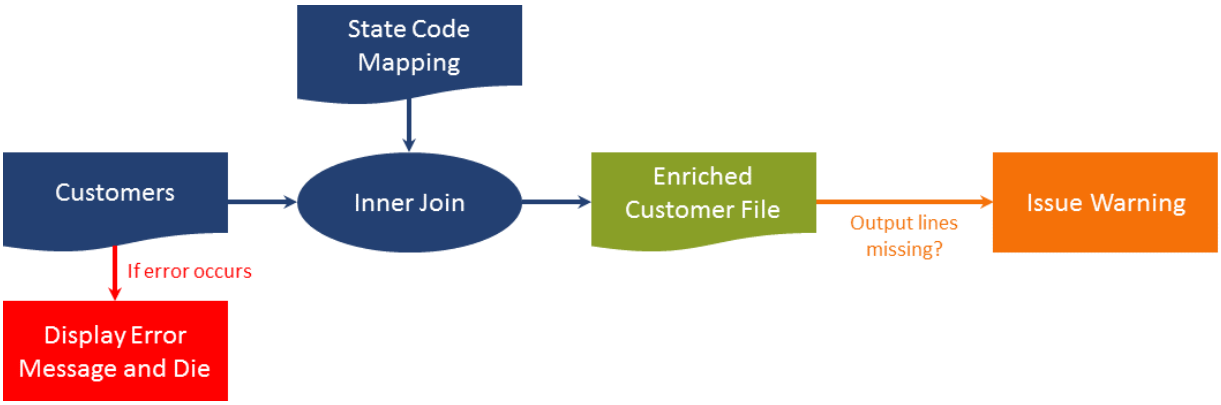
Error Handling

Lesson Overview

In a development or test environment, you can handle errors simply by observing the console, but when your project is deployed to production, you want it to deal with errors automatically.

In this lesson, you will learn how to handle technical errors but also business errors. You will also learn how to define error and warning messages.

Here is an overview of the Job you will create :



In case of technical error on the input component, the Job will die immediately in order to prevent the Job from finishing.

In case of business error on the output component, a warning will be raised to notify you ofthe anomaly.

Objectives

After completing this lesson, you will be able to:

- Kill a Job on component error
- Implement a specific Job execution path on component error
- Raise a warning under specific conditions
- Configure the log level in the execution console

Next Step

The first step is to kill a Job that generates an error.

Detecting and Handling Basic Errors

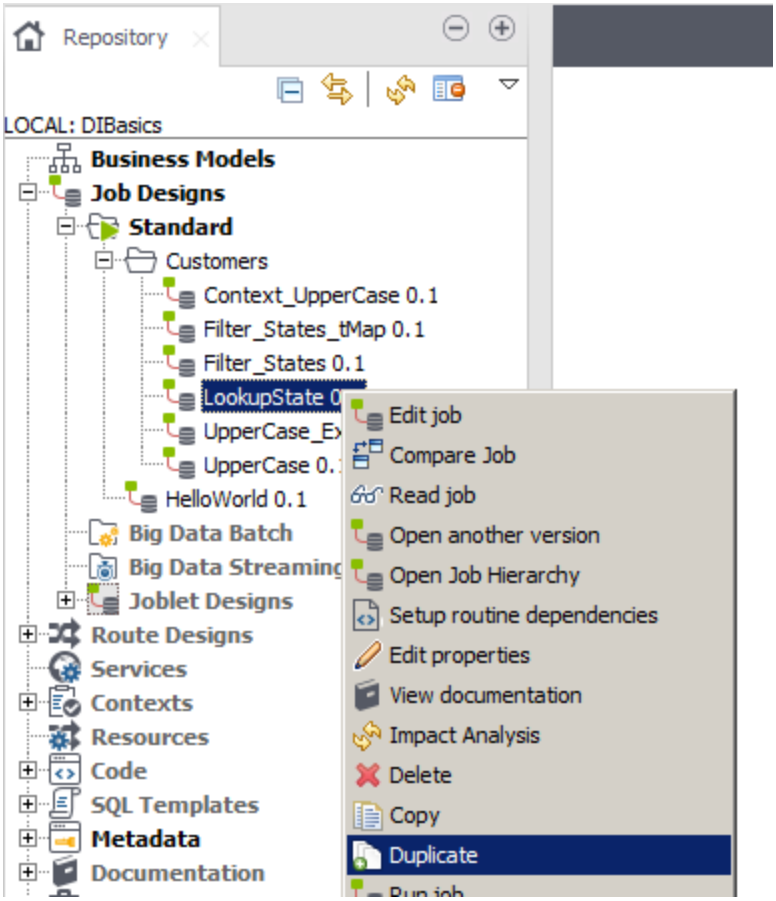
Overview

In this exercise you will create an error, configure the component to raise an exception, then update the level of logs displayed in your execution console.

Duplicate Job

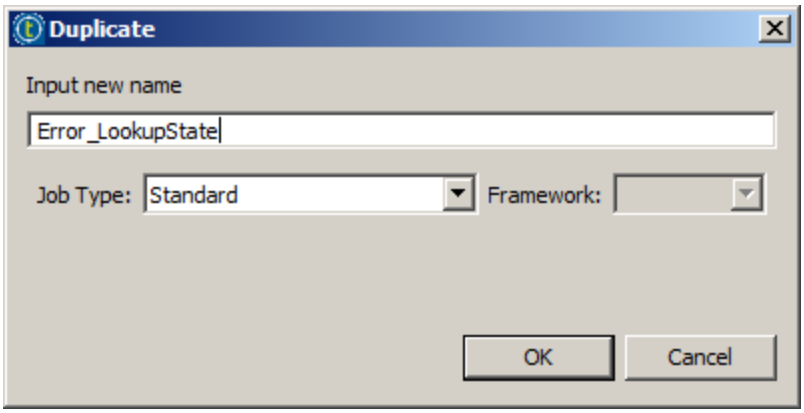
1. DUPLICATE A JOB

   Right-click **Repository > Job Designs > Standard > Customers > LookupState** and select **Duplicate**.



2. NAME THE JOB

   Enter *Error_LookupState* and click **OK**.



3. OPEN THE JOB

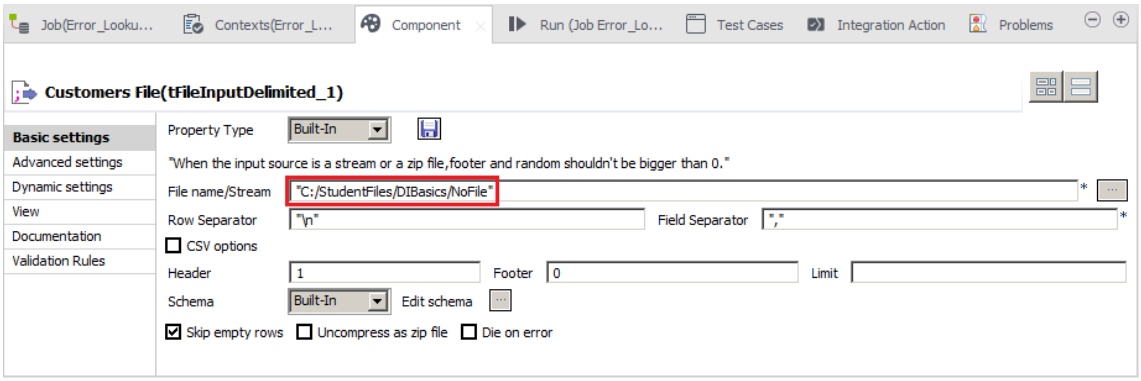
   Double-click the Job **Error_LookupState** to open in it the **Designer**.

Create an error

1. CONFIGURE AN ERRONEOUS PATH ON THE INPUT COMPONENT

   Double-click the **Customers File** component to open the **Component** view. In the **File Name/Stream** field, change the value to to *"C:/StudentFiles/DIBasics/NoFile"*.

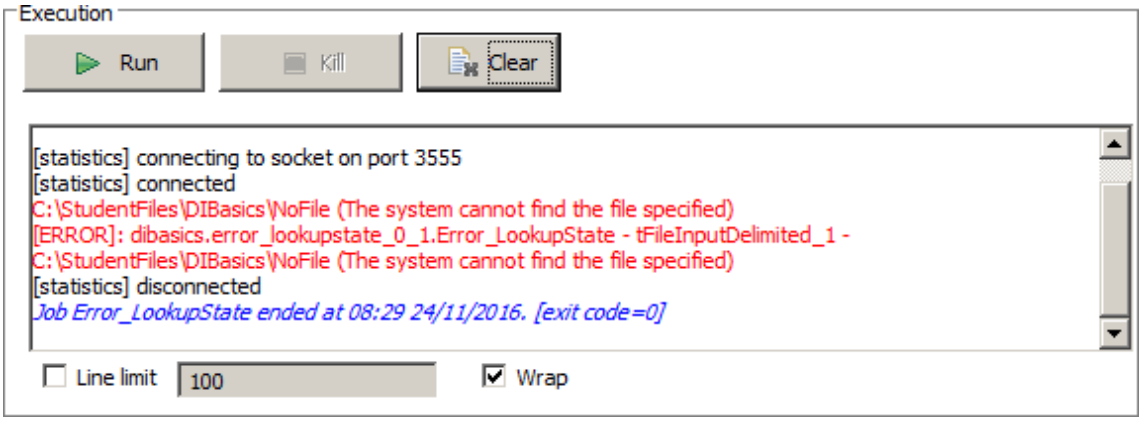
   WARNING:
   Be sure to include the enclosing quotation marks.



   This is a nonexistent file, so the component should not succeed.

2. RUN THE JOB

   Run the Job and then examine the console output.



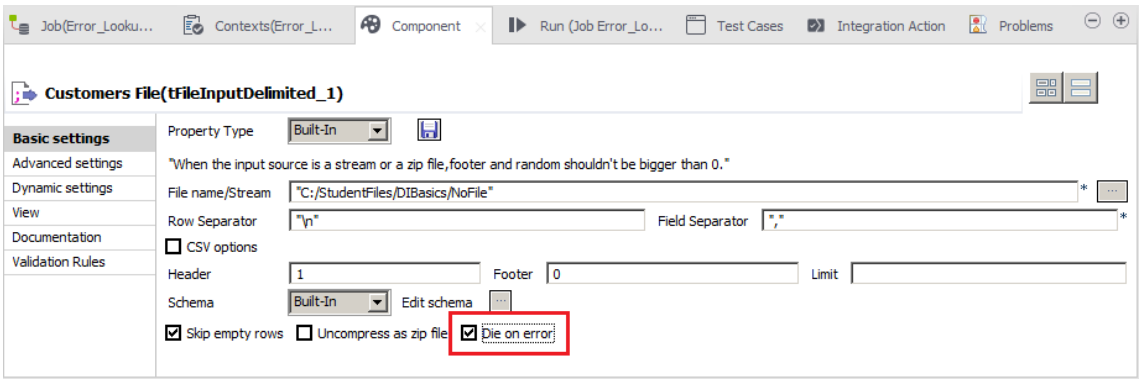
Notice that the problem with the nonexistent input file did not stop the Job from running. An error is reported on the console but the Job completes and overwrites the *CappedOut.csv* file with 0 rows. Clearly, this is not the desired outcome.

In the following steps, you will update the Job to raise an exception if the input file does not exist. This will prevent the Job from completing when the **Customers File** component encounters an error, rather than silently failing.
You will also configure the Job to display a specific message in this case.

Kill the Job on Error

1. CONFIGURE THE INPUT COMPONENT TO DIE ON ERROR

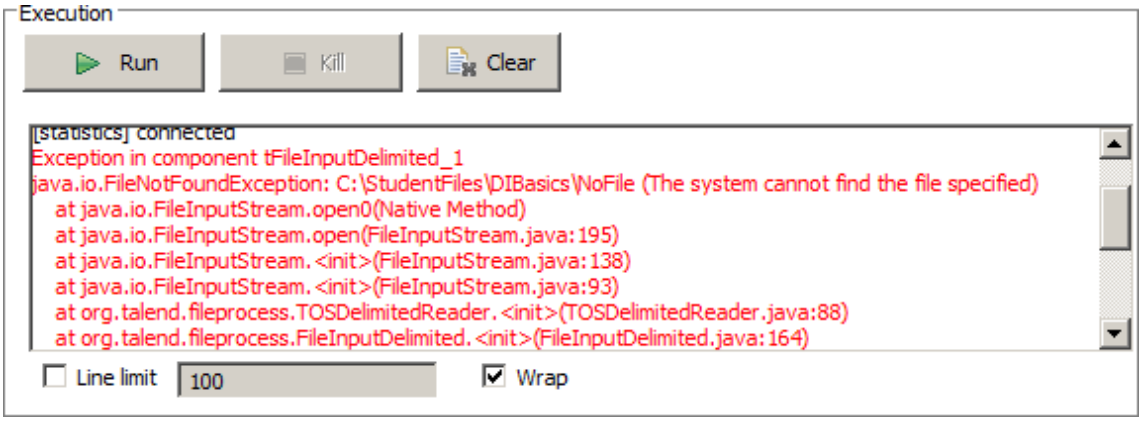
   Double-click the **Customers File** component to open the **Component** view. Select the **Die on error** check box.



   This option stops the Job if the component encounters an error.

2. COMPARE EXECUTION RESULTS

   Run the Job and then examine the result.



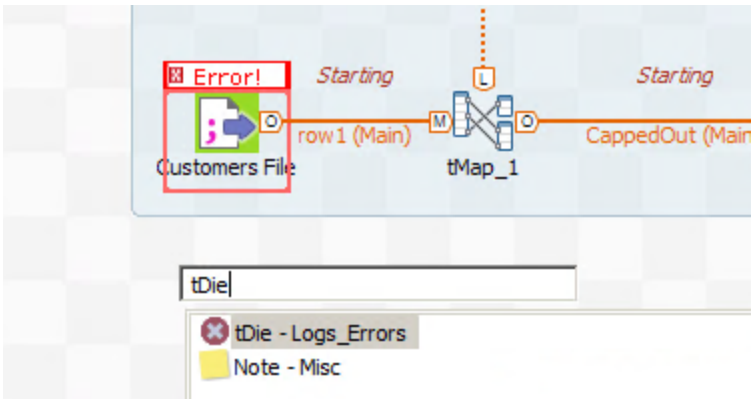
This time, an exception is raised and the Job does not complete. This is the expected behavior when the input file does not exist.

You will now explore the usage of a **tDie** component to get a customized message in case of error.

3. ADD A COMPONENT TO DELIVER A MESSAGE ON ERROR

   Add a **tDie** component to your Job under the **Customers File** component.

   TIP:
   Recall that to add a component, click on the **Designer** and begin typing the name of the desired component.



   The **tDie** component is used to send a message when it is triggered.

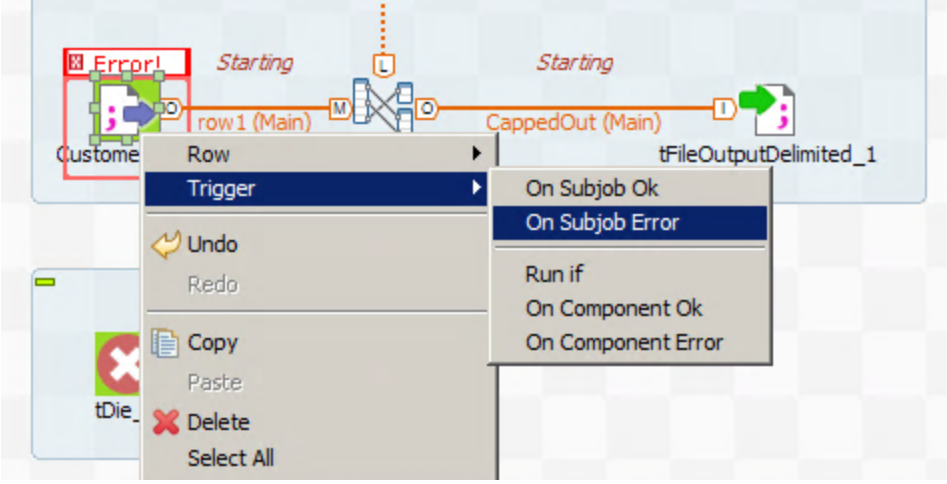
4. CREATE A TRIGGER LINK TO THE NEW COMPONENT

   To invoke the **tDie** component, you must set up a trigger.

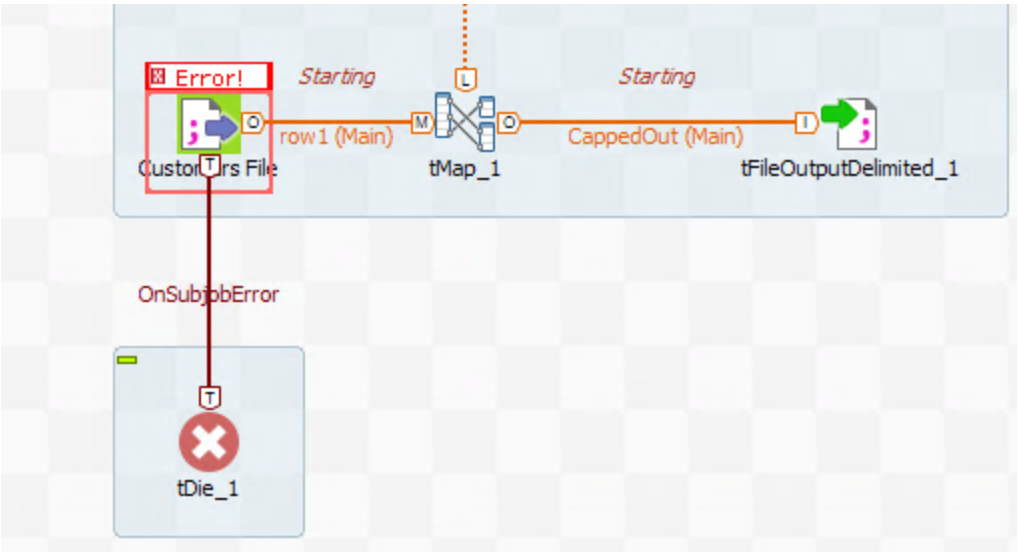
   NOTE:
   A trigger is a connection between two components that transfers control from one component to another when a particular event occurs. Unlike the row-type connections you have seen until now, triggers do not have any data flow associated with them.

   Since input components drive the data flow in a subJob, they are a natural trigger source to flag errors in a subJob and, as such, is considered best practice.

   Right-click the **Customers File** component and select **Trigger > On Subjob Error**.



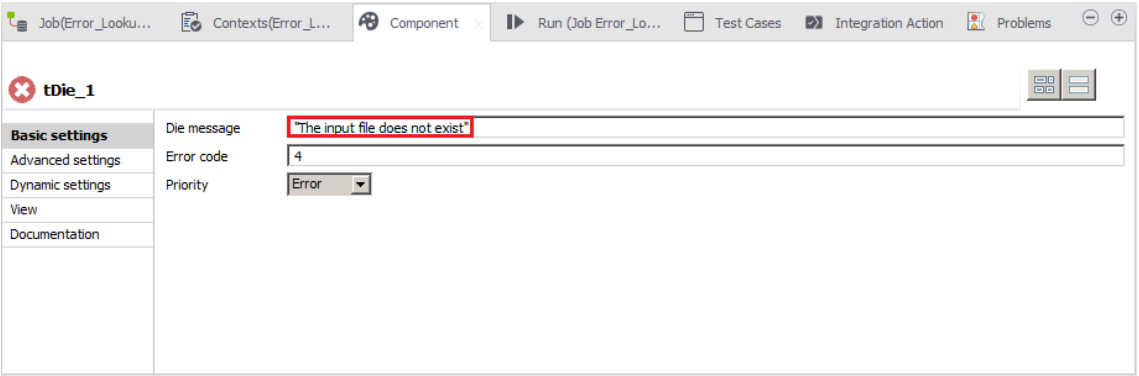
Then select the **tDie** component to create the link.



5. SPECIFY THE ERROR MESSAGE

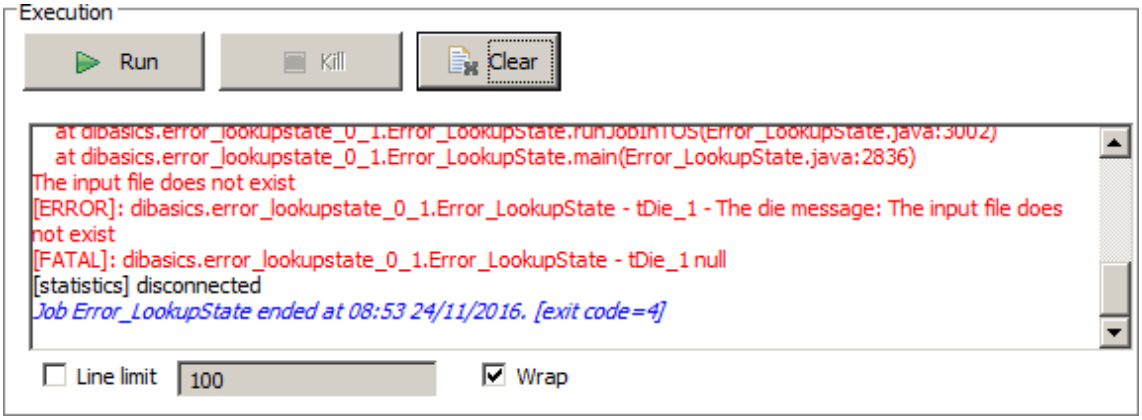
   Double-click the **tDie** component to open the **Component** view.

   Set the **Die Message** to *"The input file does not exist"*.



6. RUN THE JOB

   Run the Job and examine the result.



   Notice the **Die Message** appears in the Execution console. The **exit code=4** argument corresponds to the error code in the **tDie** component, which can be configured if desired.

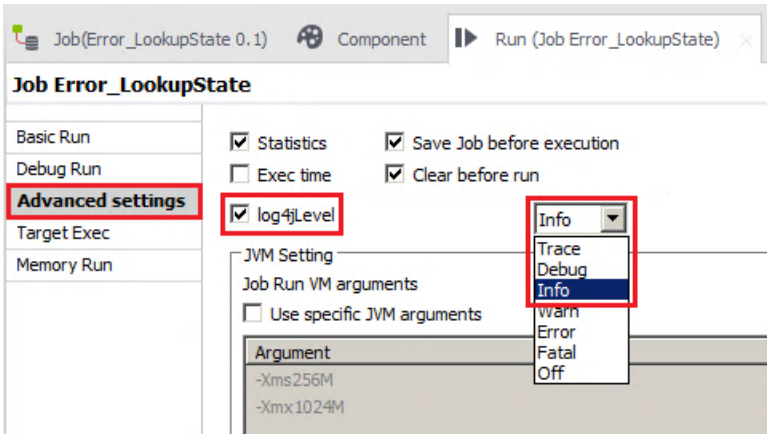
In the following steps, you will learn how to update the log level in order to control the messages that appear in the execution console.

Configure the log level in the execution console

1. SELECT THE LOG LEVEL

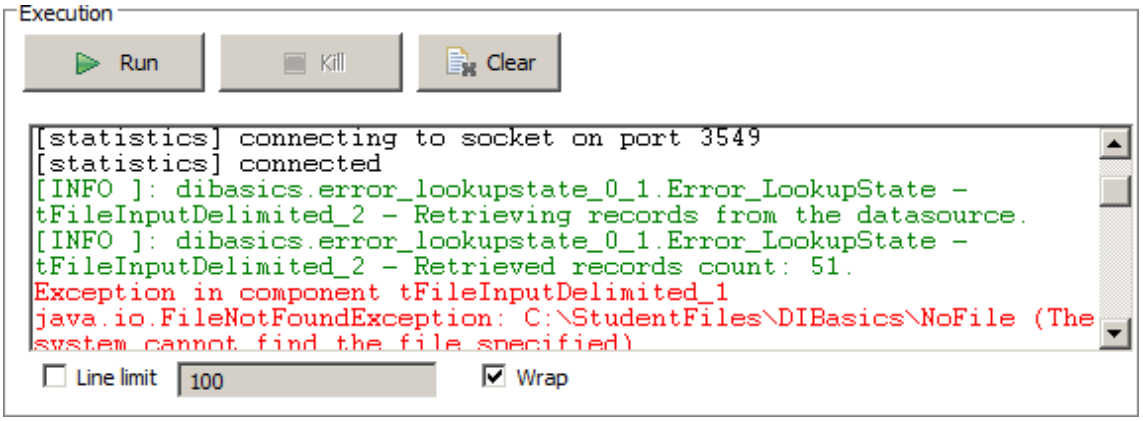
   In the **Run** view, select the **Advanced Settings** tab.

   Select the **log4jLevel** check box and select *Info* from the list of values.



2. RUN THE JOB

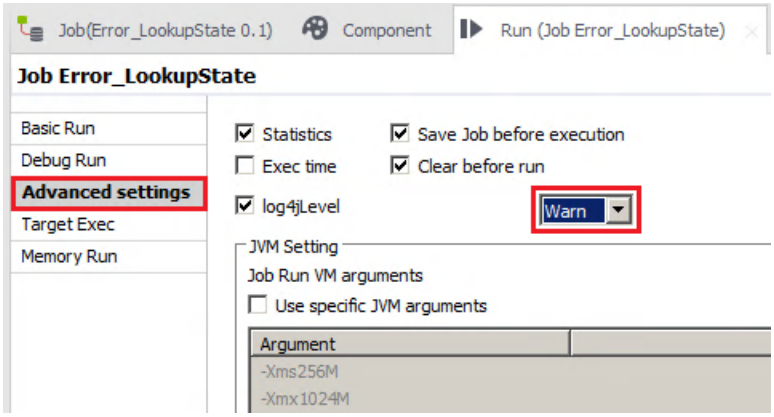
   Go back to the **Basic Run** tab and run the job again.



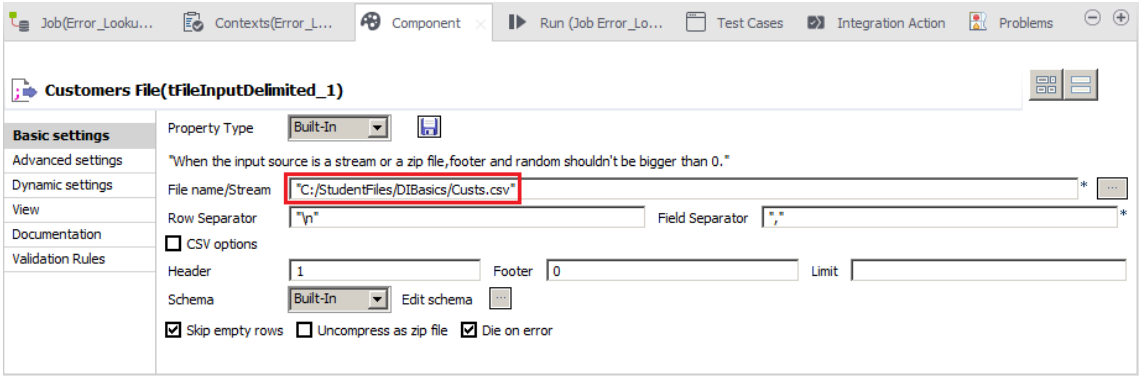
Notice that after this configuration change, info messages are also displayed in the execution console.

3. CLEANUP

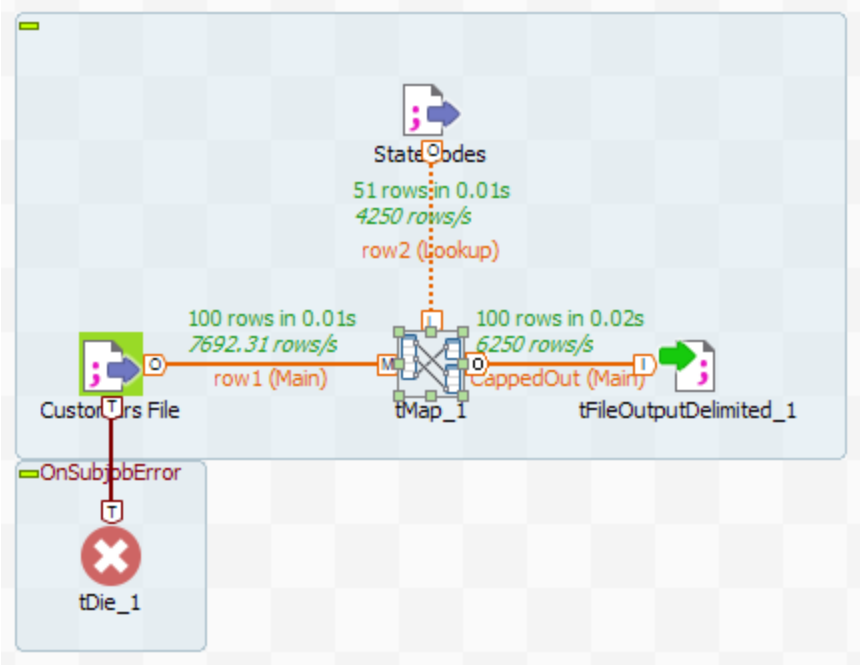
Before advancing to the next exercise, click the **Advanced Settings** tab again and set the logging level to **Warn**, so that only warnings and errors will be displayed in the running console.



Fix the problem with the **Customers File** component you introduced earlier by changing the **File Name** back to *"C:/StudentFiles/DIBasics/Custs.csv"*.



Run the Job to verify that it performs again as expected.



As before, 100 rows should be processed.

Next

In this exercise, you induced a run-time error. You configured the response to this error condition by raising an exception and displaying a specific error message. In the next section, you will raise a warning.

Raising a Warning

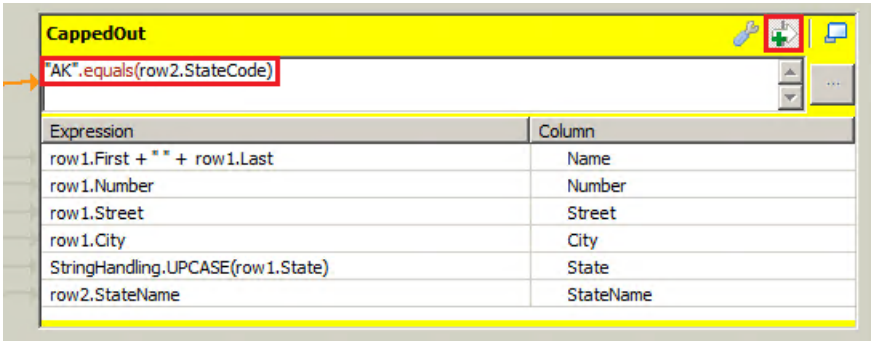
Overview

In this exercise you will raise a warning every time the output file contains fewer rows then the input file. This is a condition that often signifies problems with the data flow.

Limit the Output

1. INTRODUCE A FILTER TO LIMIT OUTPUT ROWS

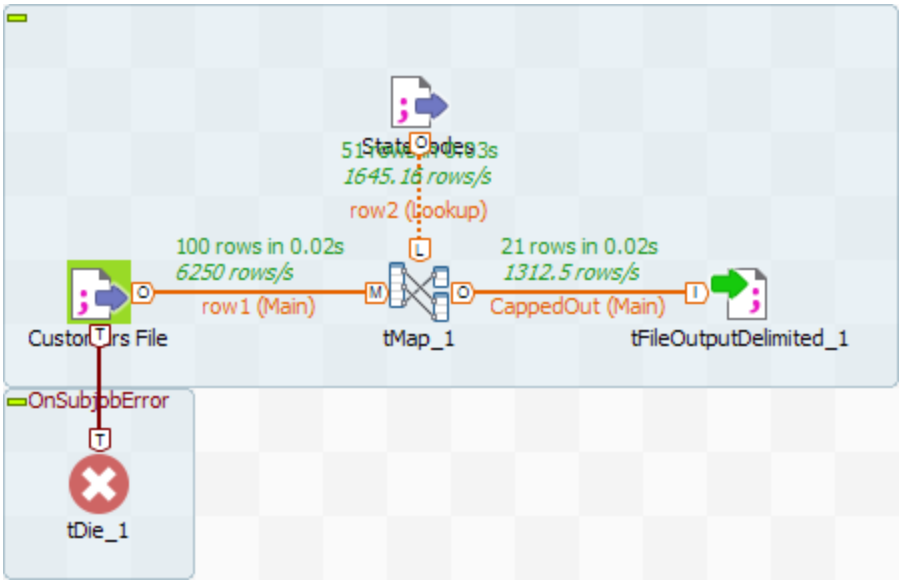
   Double-click the **tMap** component and click the **Activate filter expression** button ( ) on the **CappedOut** table. Enter `"AK".equals(row2.StateCode)` into the box and then click **Ok**.



   This expression limits the output to rows having the *StateCode* value set to **AK** (Alaska) . Some, but not all, of the rows contain that value.

2. RUN THE JOB

   Run the Job and then examine the results.



   The output from the **tMap** component is now only 21 rows, which is significantly lower than the input flow of 100 rows.

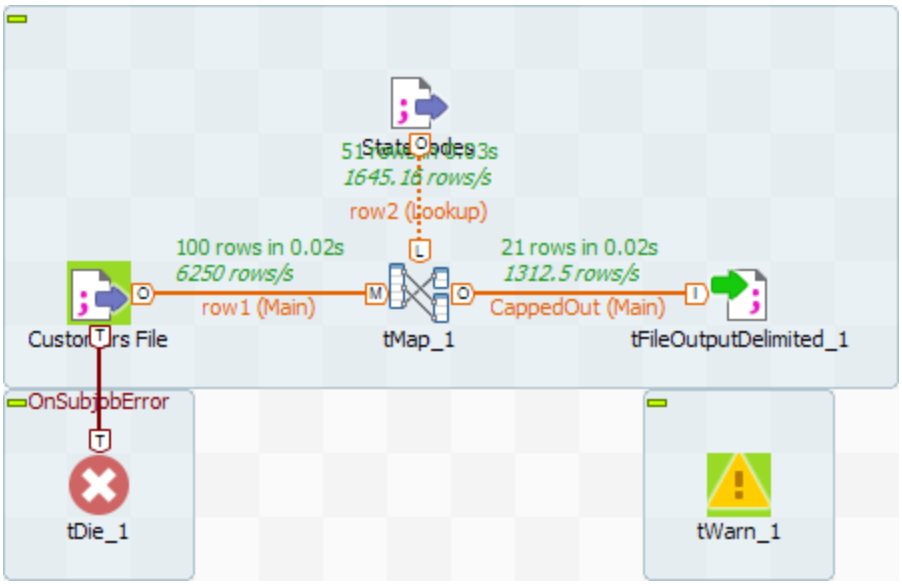
   In some cases, a condition like this, where the output flow is fewer than the input flow, might indicate some sort of problem with the data flow. Knowing how to display a warning under these conditions can therefore be helpful.

In the following section, you will configure the Job to display a warning message if the output contains less rows than the input.

Raise a warning

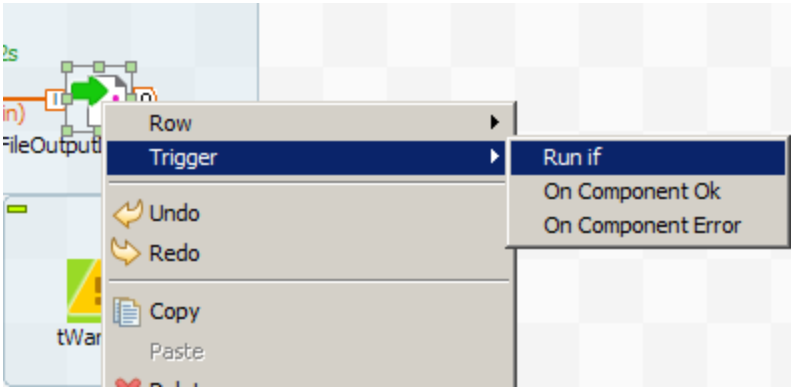
1. ADD A WARNING COMPONENT

   Add a **tWarn** component just after the **tFileOutputDelimited** component.



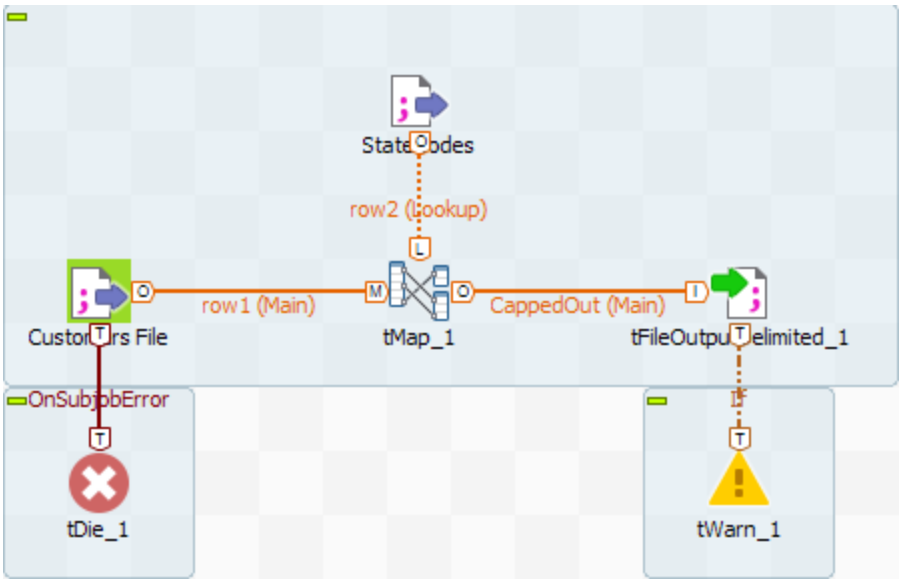
2. ADD A TRIGGER FROM THE OUTPUT COMPONENT

   Right-click the **tFileOutputDelimited** component and select **Trigger > Run if**, then click the **tWarn** component to create the link.



   With this kind of a connection, the **tWarn** component executes only if a particular condition is met.

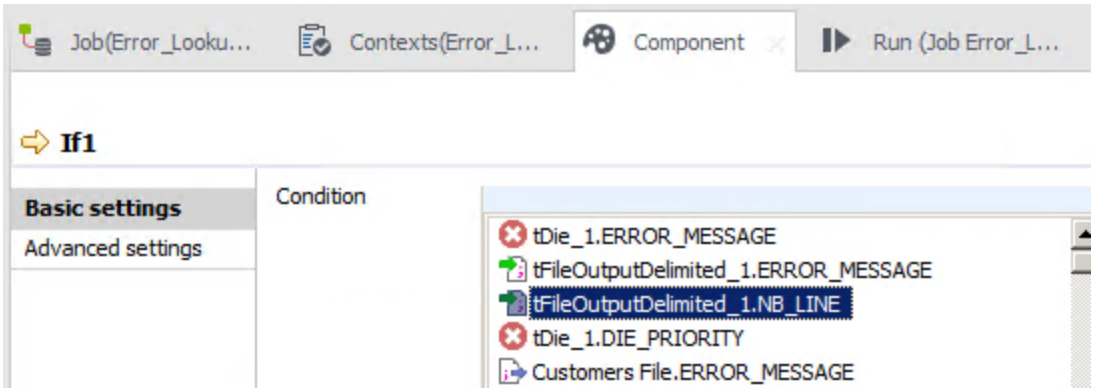
   After the connection, the Job should resemble the following figure.



3. SPECIFY THE CONDITION FOR THE TRIGGER

   Select the **If** connector between **tFileOutputDelimited_1** and **tWarn_1** and then open the **Component** view. In the **Condition** box, press **Ctrl+Space** to invoke an auto-complete list. Scroll down the list until you find **tFileOutputDelimited_1.NB_LINE** and then click it. Feel free to read the explanation that appears. It explains that this item represents the number of lines written by the specified component.

   Double-click to insert the variable into the **Condition** box.



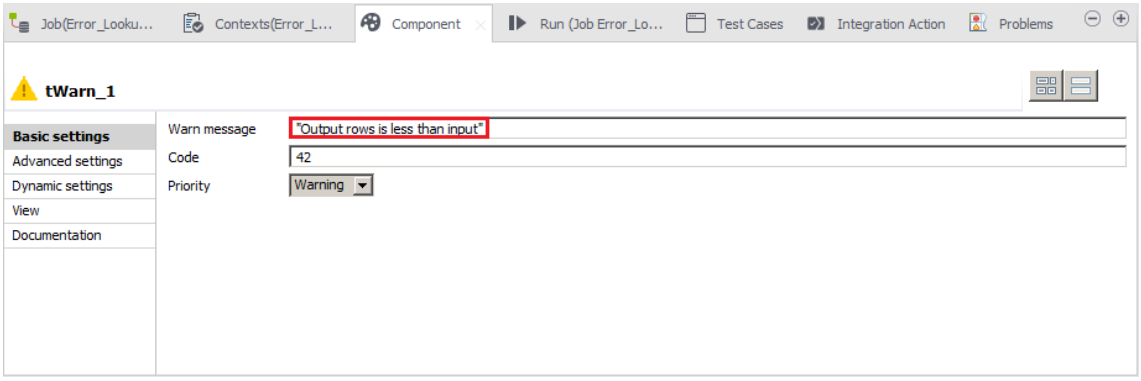
   Append a less than (<) character to the string in the **Condition** box, then press **Ctrl+Space** again. Scroll down the list until you find *Customers File.NB_LINE* and double-click to insert the variable. Once complete, the expression should look like this:

```
Condition    ((Integer)globalMap.get("tFileOutputDelimited_1_NB_LINE")) <
             ((Integer)globalMap.get("tFileInputDelimited_1_NB_LINE"))
```

   Now, the **tWarn** component will execute only if the number of rows written to the output file is less than the number of rows passed in from the input file.

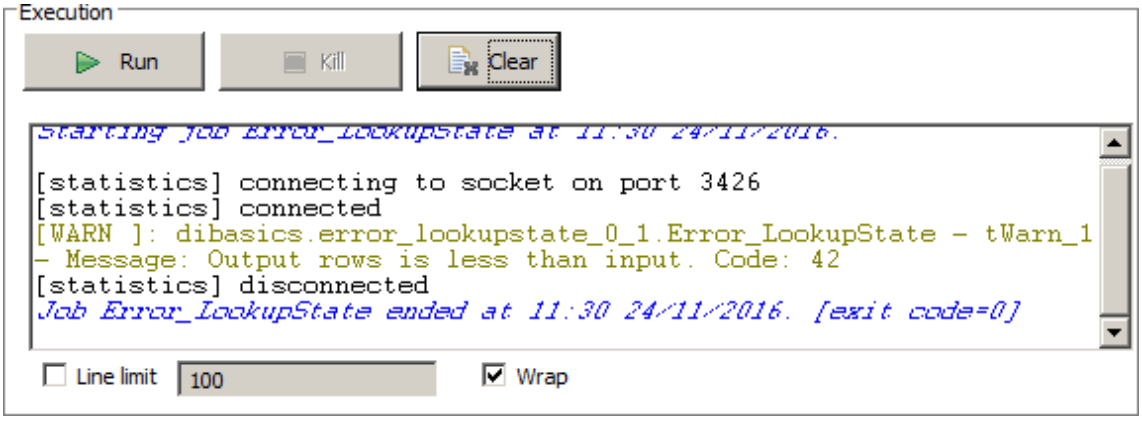
4. ADD A WARNING MESSAGE

   Double-click the **tWarn** component to open the **Component** view. Change the **Warn message** text to *"Output rows is less than input"*.



5. RUN THE JOB

   Run the Job again and verify that the warning message is displayed in the execution console.



Next

You have now completed this lesson. It's now time to Wrap-up.

Wrap-Up

In this lesson, you configured a component to stop the Job's execution by using the **Die on error** option. Then, you used the **tDie** component to display a specific error message in the execution Console.
You also modified the logging level, which allows you to choose the level of information displayed in the Console.

Finally, you used the **tWarn** component to raise a warning message under specific conditions and the **Run If** trigger to limit the execution of components.

Next step

Congratulations! You have successfully completed this lesson. To save your progress, click **Check your status with this unit** below. To go to the next lesson, on the next screen, click **Completed. Let's continue >**.

LESSON 7

Generic Schemas

This chapter discusses:

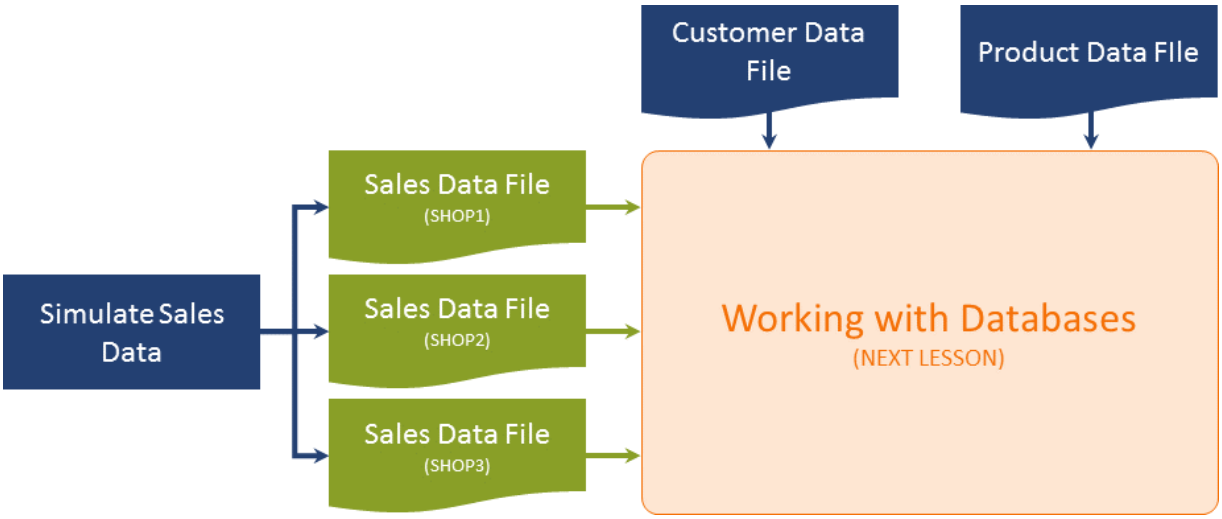
Working with Generic Schemas

Lesson Overview

You will now start working on a new use case with the following scenario: you are taking raw sales data from different retail stores, joining that data with existing customer and product information, and loading the combined data into a combined sales data table.

In this instance, the sales data does not yet exist, so you will create a Job to generate the data. Once the stores are online and processes are in place, the same resulting Job would work against actual store data, not artificially generated data.

Here is a high-level diagram showing the data flow for the use case:



In this lesson, you prepare the necessary schemas, files, and metadata to have them ready in the following lesson while working with databases. Here are the steps you will follow in this lesson:

- Define a generic schema for the Sales data files
- Generate three sales data files (one for each shop)
- Define the metadata for customers file (file is provided)
- Define the metadata for products file (file is provided)

Objectives

After completing this lesson, you will be able to:

- Create a generic schema
- Generate files containing random data
- Capture all information on your files in context variables
- Use subJobs

Next Step

The first step will be to define a generic schema for sales data and generate three sales data files following that schema.

Setting Up Sales Data Files

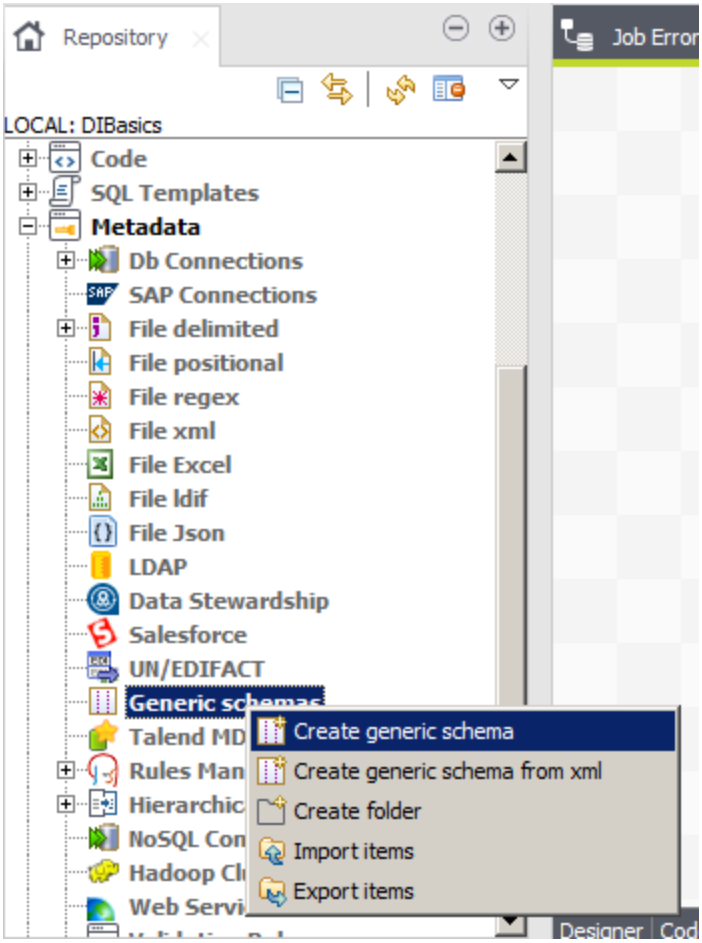
Objective

In this lesson you will create a generic schema in the Repository. This schema will describe sales data information. The generic schema will then be used to generate sales data files.

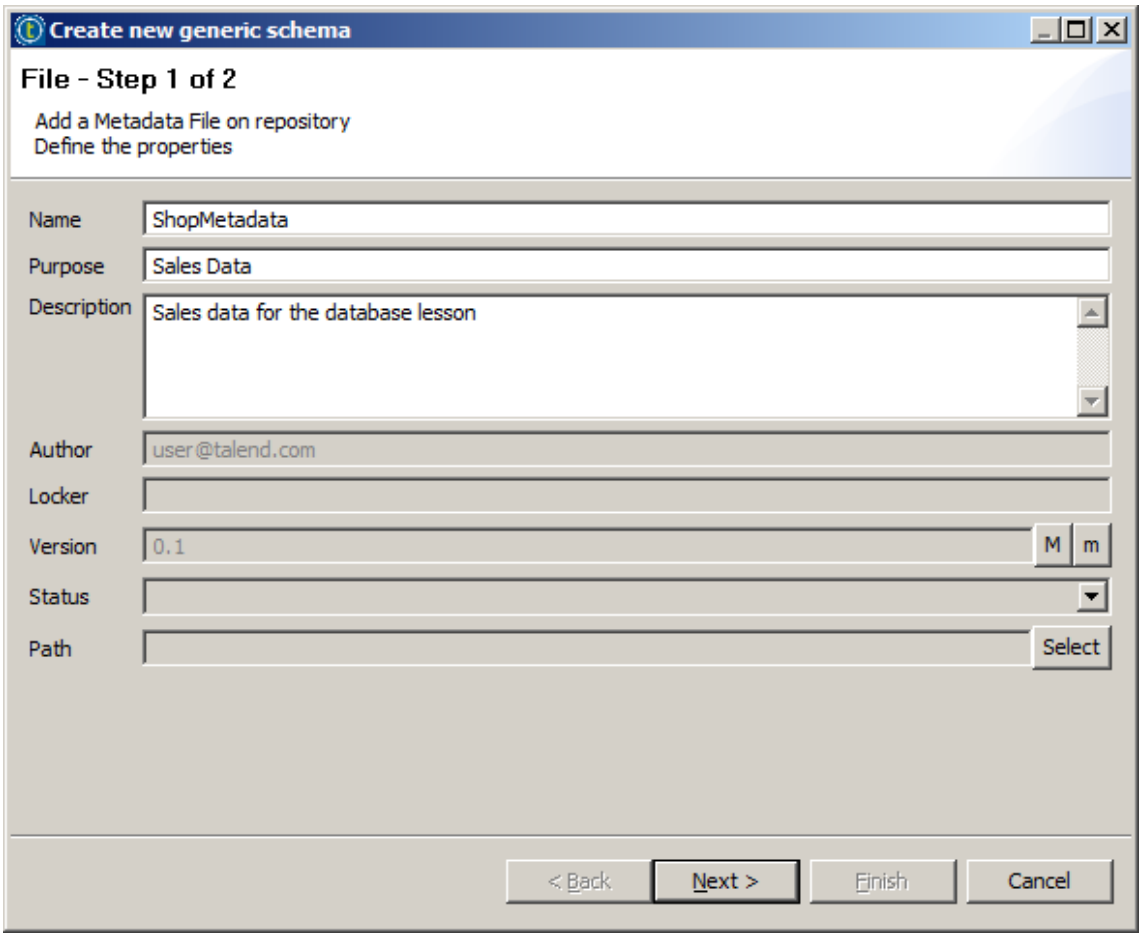
Create the generic schema

1. CREATE A GENERIC SCHEMA

   Right-click **Repository > Metadata > Generic schemas** and select **Create generic schema**.



   Enter *ShopMetadata* for the **Name**. Populate **Purpose** and **Description** and click **Next**.



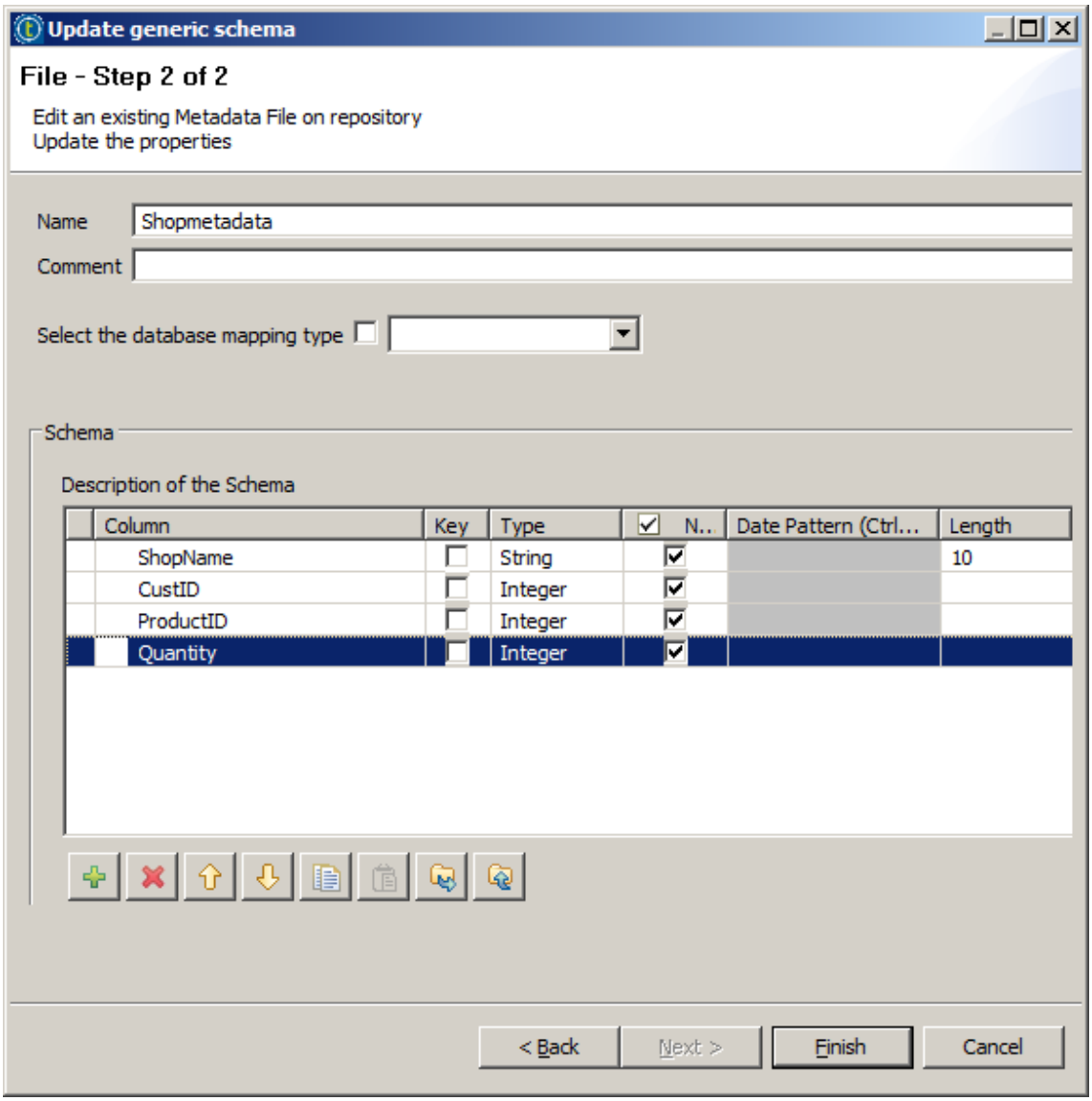
2. CONFIGURE THE COLUMNS

   Set the **Name** to *Shopmetadata*.

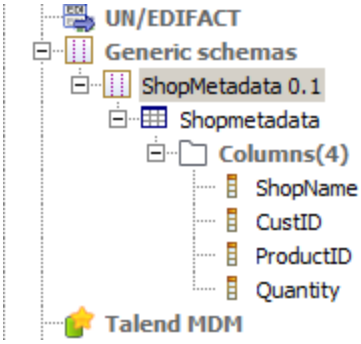
   Click the **Add** button (below the **Description of the Schema**) four times to add four columns to the schema. Populate the schema as follows:

   - *ShopName* of type **String** with a **Length** of *10*
   - *CustID* of type **Integer**
   - *ProductID* of type **Integer**
   - *Quantity* of type **Integer**

   When done, your schema should resemble the figure below. Click **Finish**.



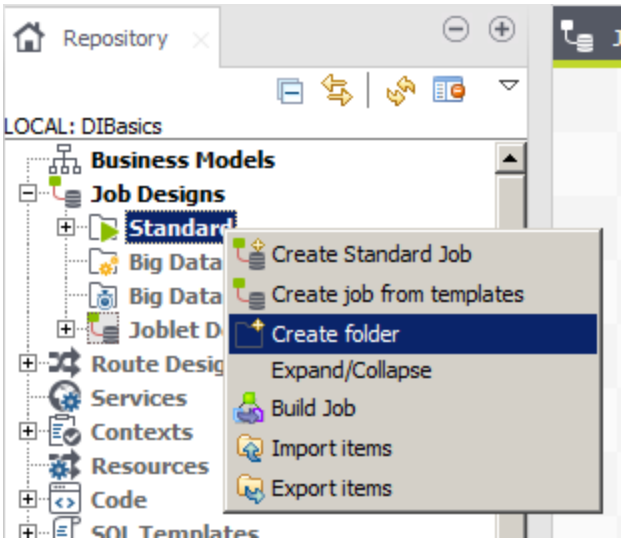
The schema then appears in the **Repository**.



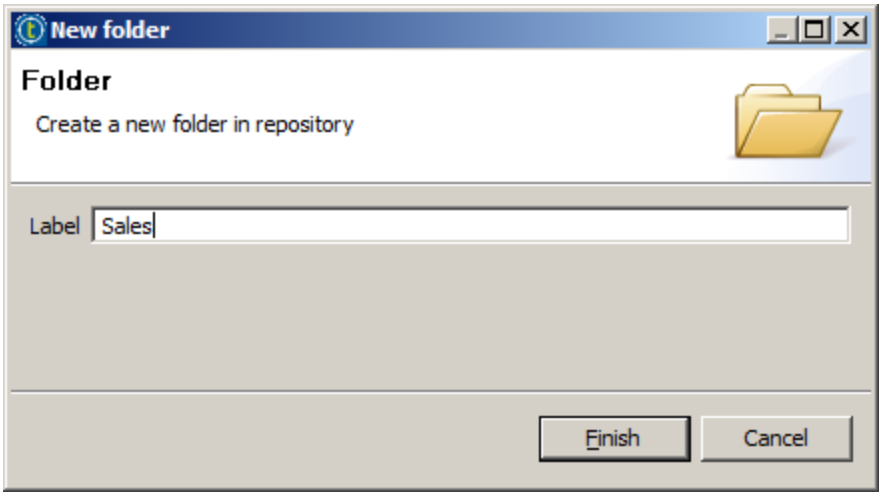
Generate sales data files

1. CREATE A NEW JOBS FOLDER

   Right-click on **Repository > Job Designs > Standard** and select **Create folder**.

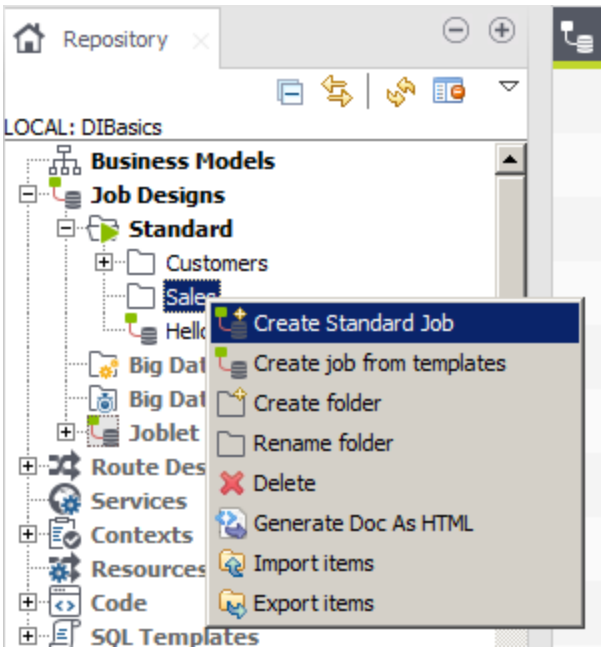


   Set the **Label** to *Sales* and click **Finish**.

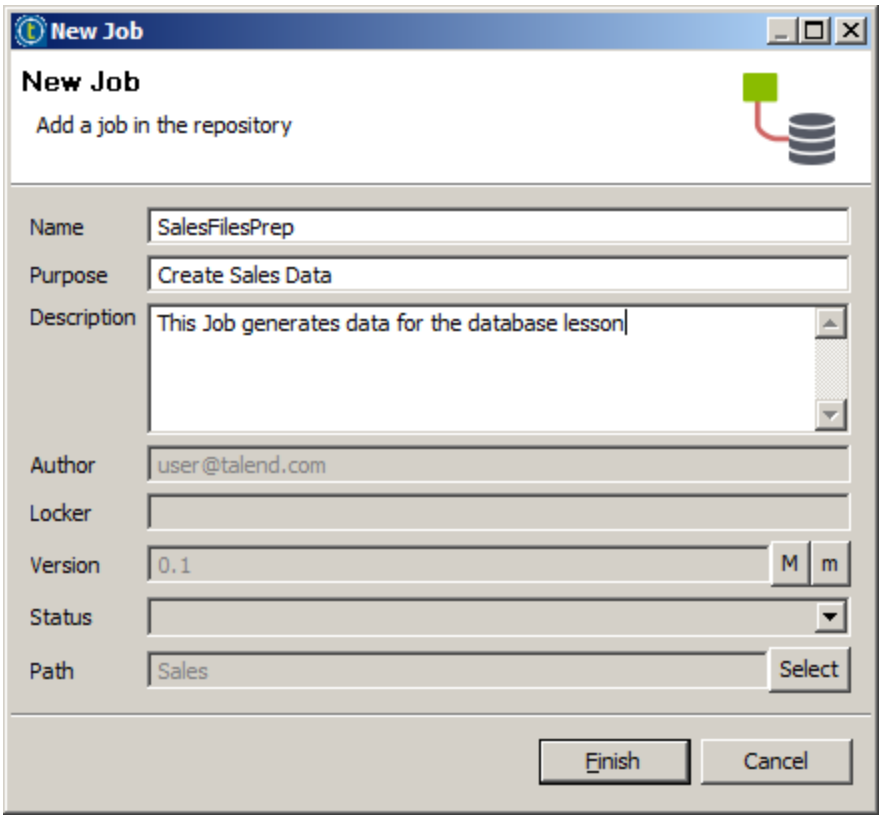


2. CREATE A JOB IN THE FOLDER

   Right-click on the **Sales** folder and select **Create Standard Job**.



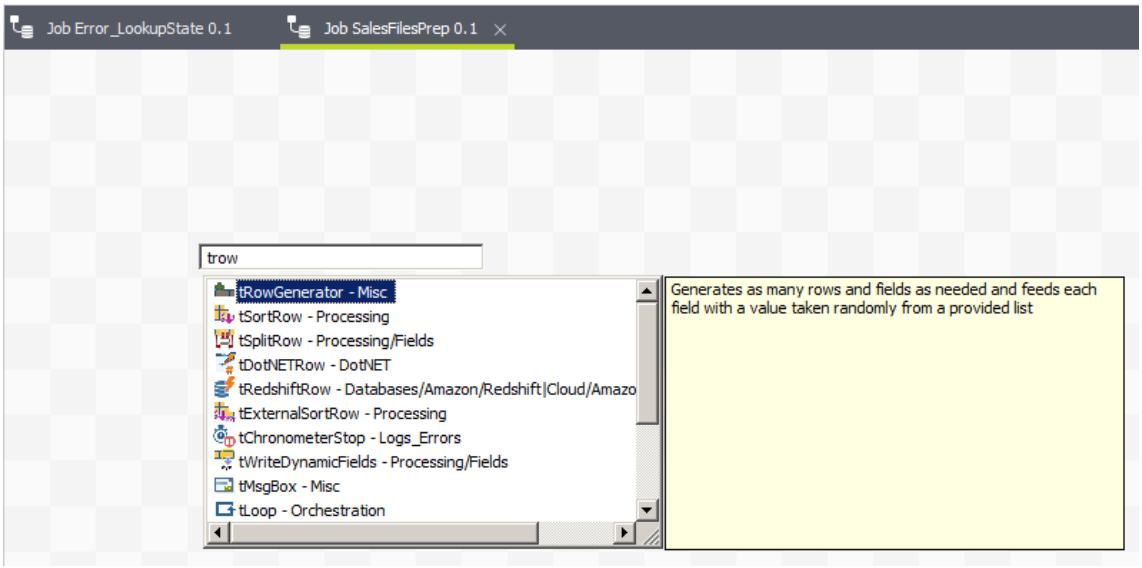
Enter *SalesFilesPrep* for the **Name**. Populate **Purpose** and **Description** and click **Finish**.



The newly created Job opens in the **Designer**.

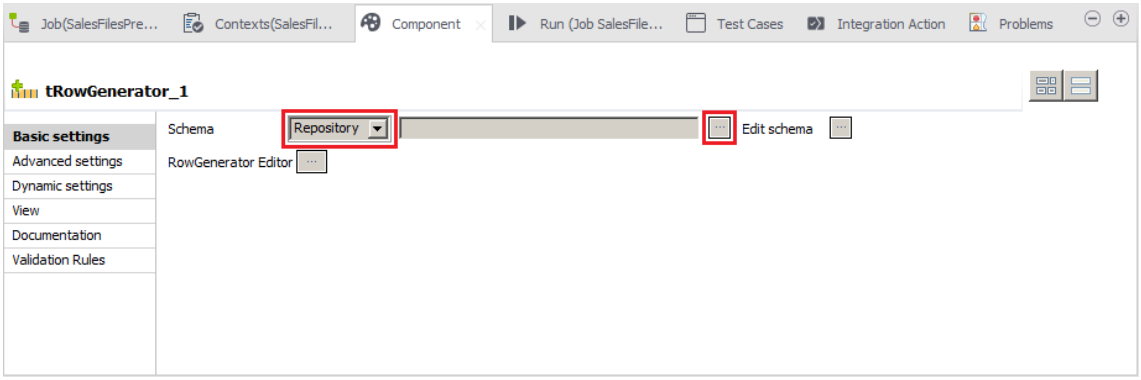
3. ADD A COMPONENT TO GENERATE RANDOM DATA

Add a **tRowGenerator** component to the **Designer**.

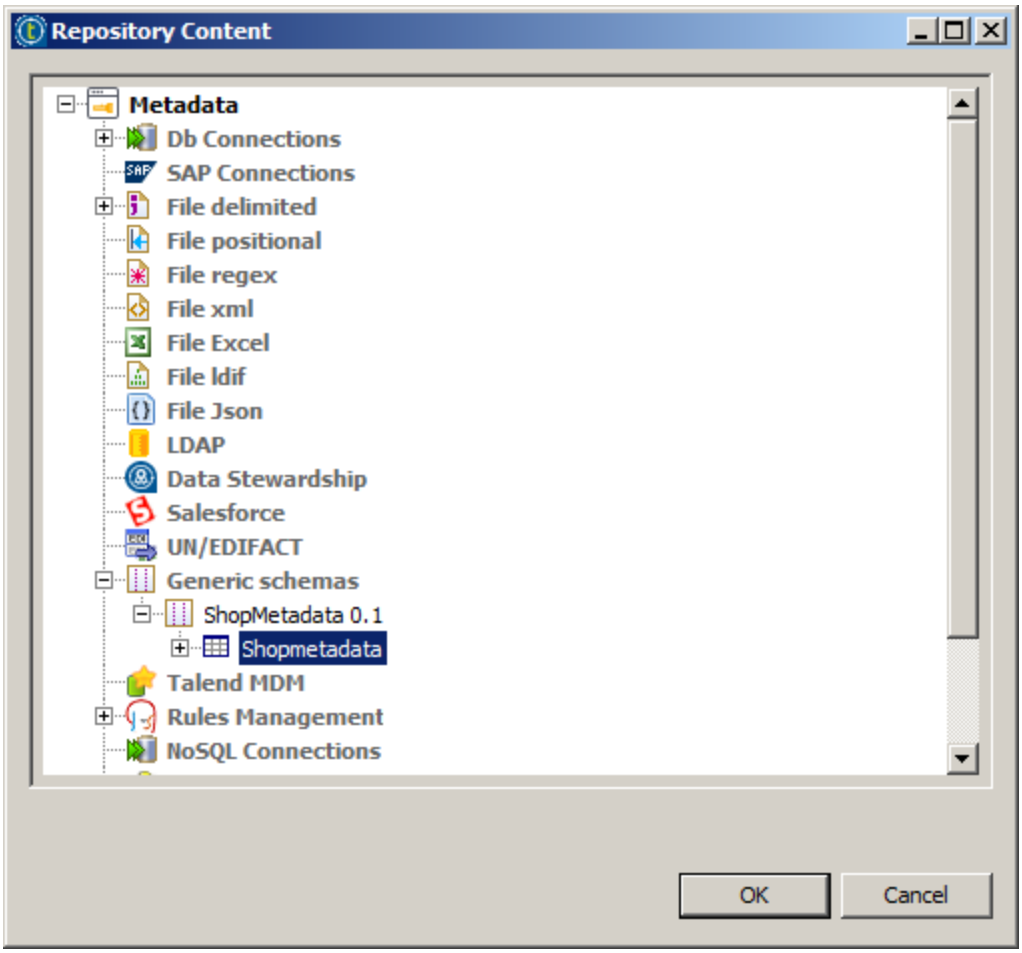


4. CONFIGURE THE COMPONENT SCHEMA

Click the **Component** tab to open the **Component** view and specify *Repository* for the **Schema**. Then, click the **[...]** button to the left of the **Edit schema** label.

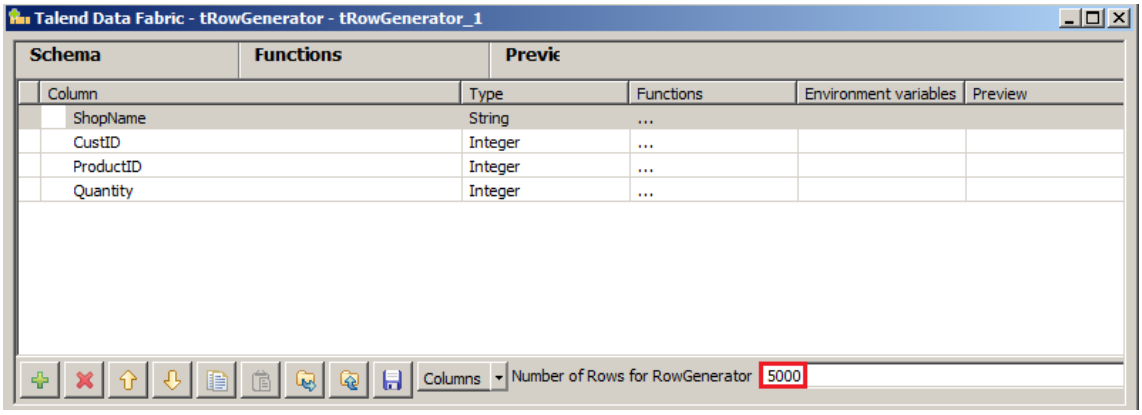


In the **Repository Content** window that opens, select **Metadata > Generic schemas > ShopMetadata 0.1 > Shopmetadata** and click **OK**.



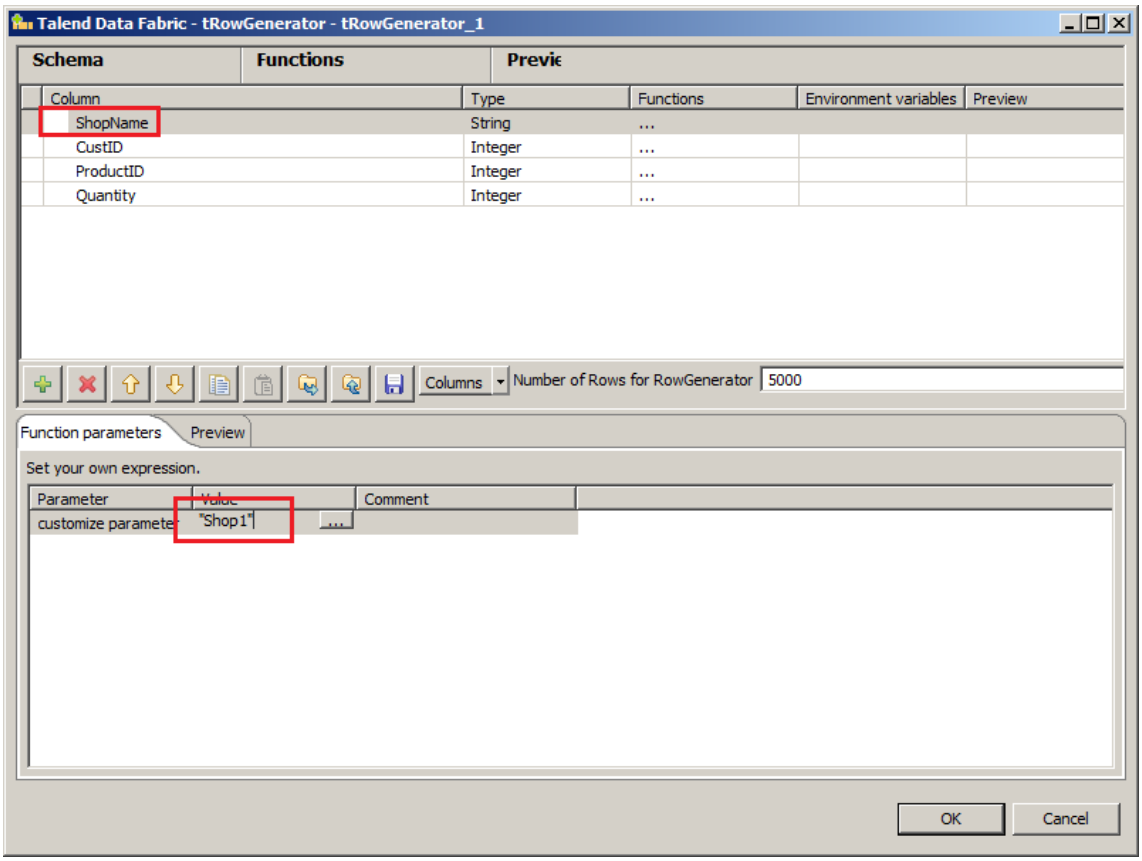
5. CONFIGURE THE AMOUNT OF DATA TO GENERATE

   Double-click the **tRowGenerator_1** component. The schema should look familiar. Enter *5000* for **Number of Rows for RowGenerator**. This will cause the component to generate 5000 entries.

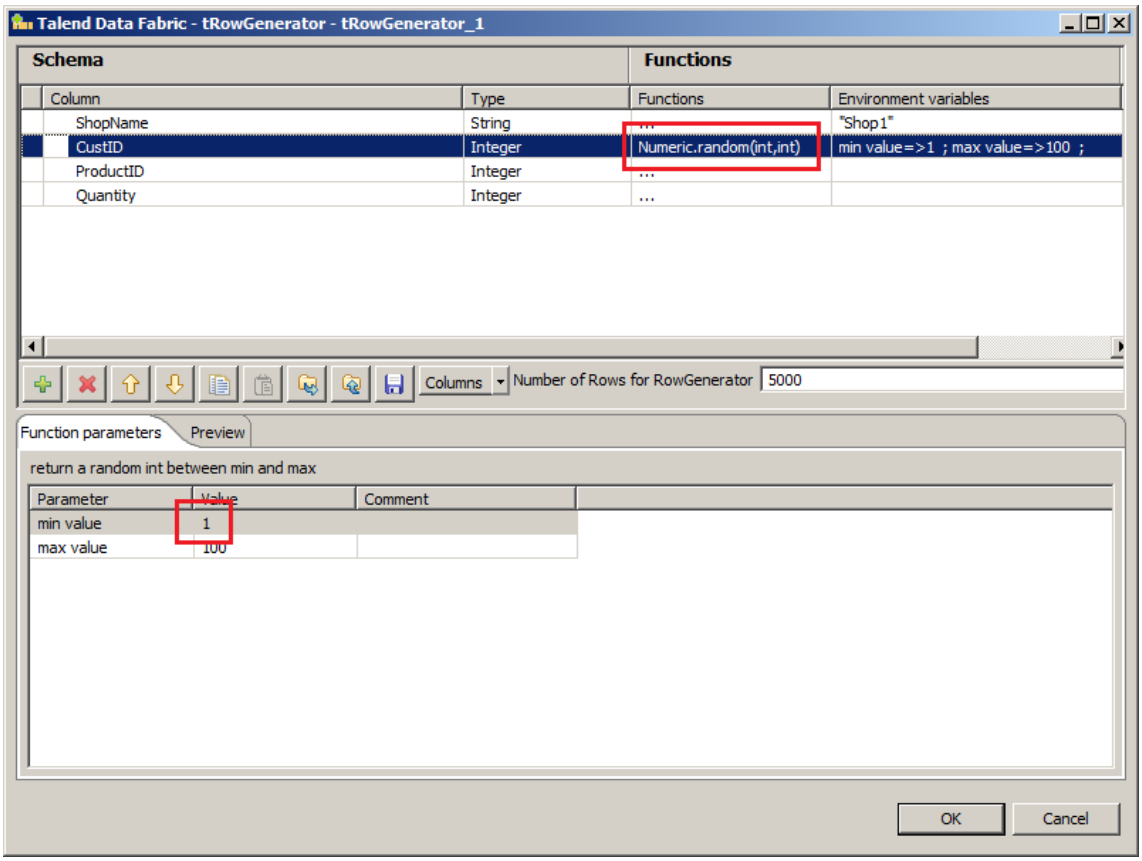


6. CONFIGURE THE GENERATED VALUES

   Select the **ShopName** column in the upper table, then enter *"Shop1"* for the **Value** in the table below, thereby setting *Shop-name* to a hard-coded value for every row generated by this component.

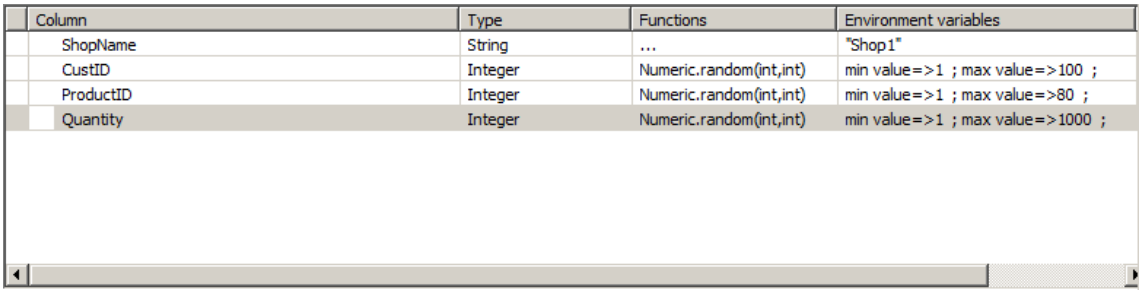


Now, select the **CustID** column in the upper table. Then, in the **Functions** column, select *Numeric.random(int,int)*.In the **Function parameters** table below, enter a value of *1* for **min value** and leave *100* for **max value**. This will generate a random number between 1 and 100 as the value for **CustID** in every row.



Similarly, configure the **ProductID** to generate a random number from 1 to 80 , and then **Quantity** to generate a random number from 1 to 1000.

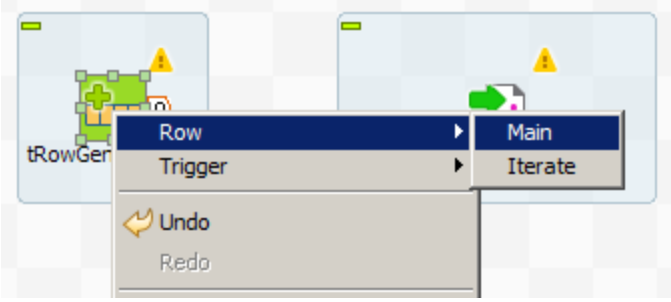
When finished, your settings should resemble the figure below. Click **OK**.



7. ADD A COMPONENT TO WRITE GENERATED DATA TO A FILE

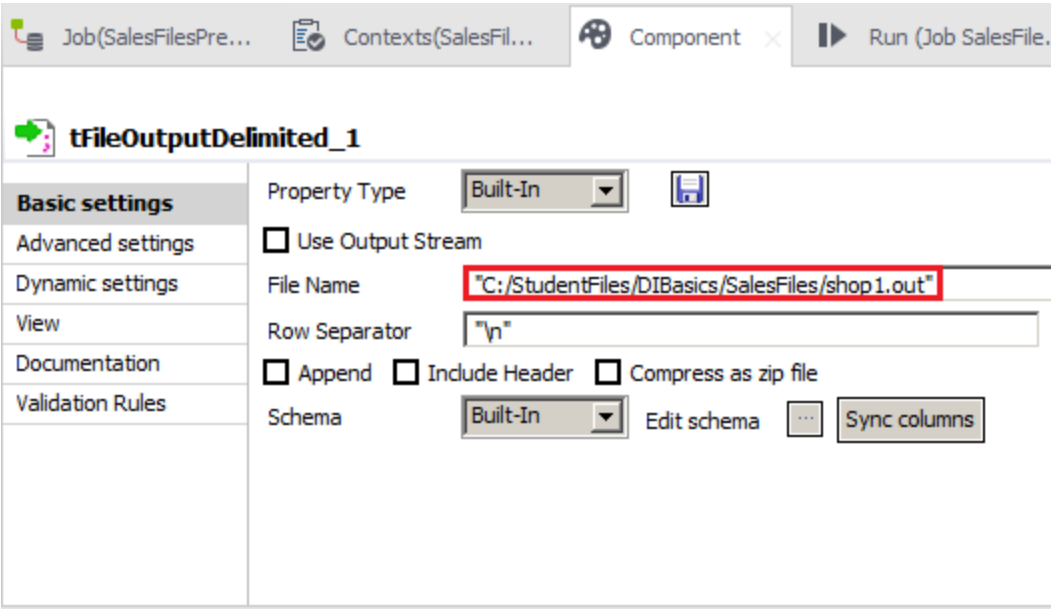
   Add a **tFileOutputDelimited** to the right of **tRowGenerator_1**.

   Connect the components by right-clicking **tRowGenerator_1**and selecting **Row > Main**, then clicking on **tFileOutputDelimited_1**.



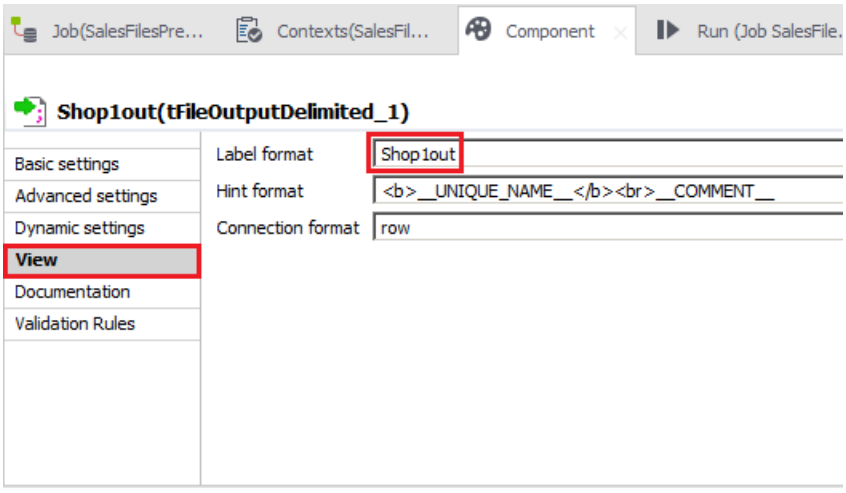
8. CONFIGURE THE OUTPUT FILE

Double-click **tFileOutputDelimited_1** to open the **Component** view. Set the **File Name** to *"C:/StudentFiles/DIBasics/SalesFiles/shop1.out"*. Leave the **Field Separator** as is.



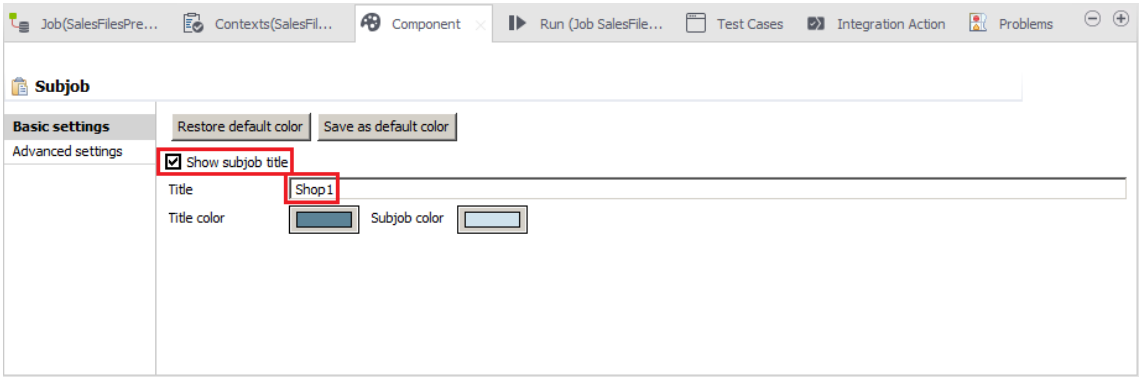
9. CHANGE THE COMPONENT NAME

In the **Component** view, click the **View** tab and enter *Shop1out* in the **Label format** field.

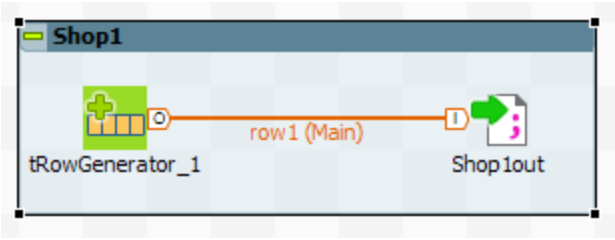


10. CONFIGURE THE SUBJOB

Click anywhere in the subJob frame (denoted by the blue region surrounding the linked components) in the **Designer**. Then, click the **Component** tab to open the subJob settings. Select the **Show subjob title** check box in the **Basic settings** section. In the **Title** box, enter *Shop1*.



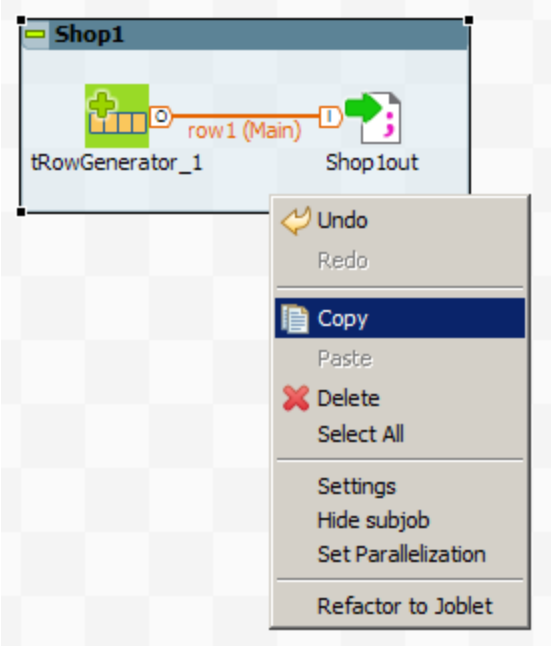
Notice how the subJob's appearance is transformed in the **Designer**.



Create additional subJobs

1. COPY THE SUBJOB

Copy the Subjob to the clipboard by right-clicking anywhere within the Subjob frame of **Shop1**, and then selecting **Copy**.

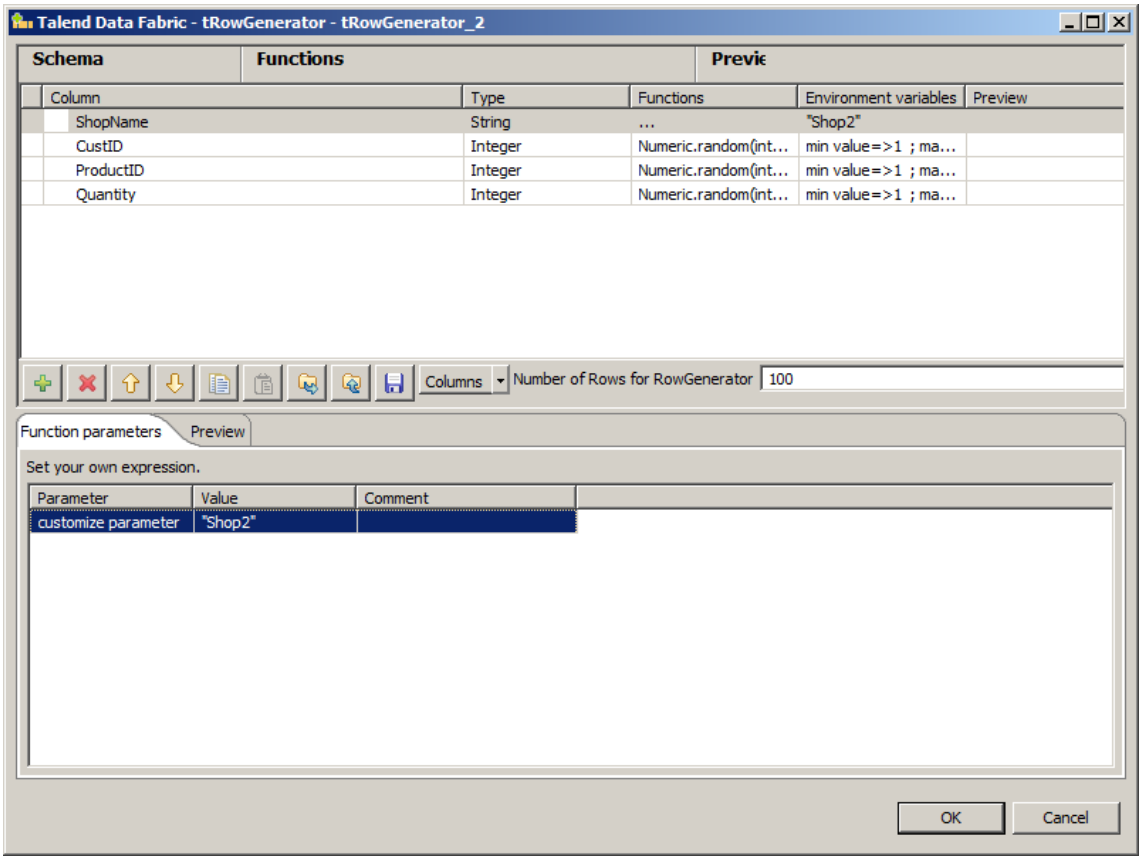


Right-click in the workspace below **Shop1** and select **Paste** to insert a copy of the subJob.

2. CUSTOMIZE THE NEW SUBJOB

   Click the frame of the newly inserted subJob. In the **Component** view, change the **Title** to *Shop2*.

   Double-click **tRowGenerator_2**. Change the value for **ShopName** to *"Shop2"*. The settings should now look like this.



   Double-click the **Shop1out** component of **Shop2**. In the **Component** view, change the **File Name** to *"C:/StudentFiles/DIBasics/SalesFiles/shop2.out"*.

   Still in the **Component** view, click **View** and change the **Label format** to *Shop2out*.

3. CREATE A THIRD SUBJOB

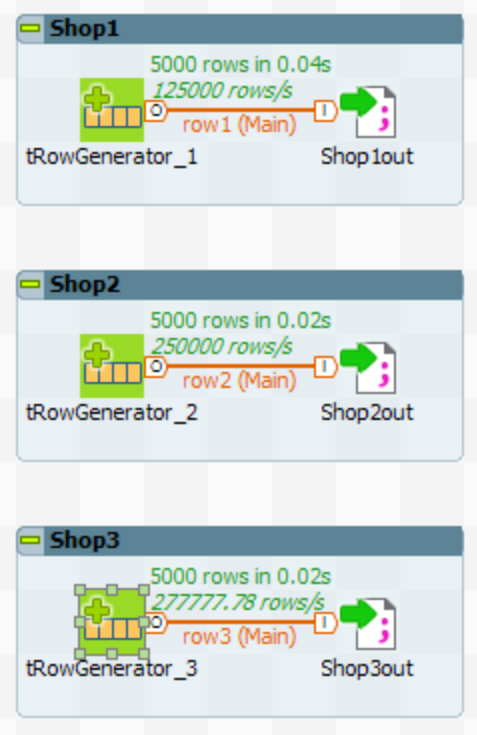
   Create another copy of the subJob and customize it for *Shop3*. Adjust all settings accordingly.

Run the Job

Now you are going to run the Job and check that you have generated the random data for the three shops.

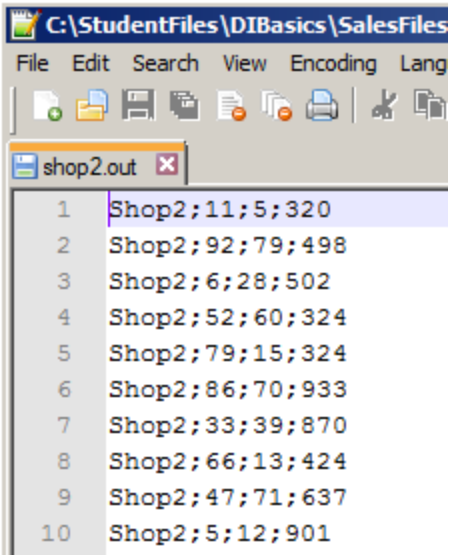
1. RUN THE JOB

   Run the Job.



Verify that the output folder *C:/StudentFiles/DIBasics/SalesFiles* was created along with three data files within it. Open each one and make sure the first column reflects the correct shop name, and the next three entries show random numbers within the correct ranges delimited by a ";". There should be 5000 rows in each file.

As an example, the file *shop2.out* should look similar to the following figure, although the numbers will differ due to the random nature of the generated data.



Next

You have now finished generating the sales data files. Next you will create the metadata for a customer file.

Creating Customer Metadata

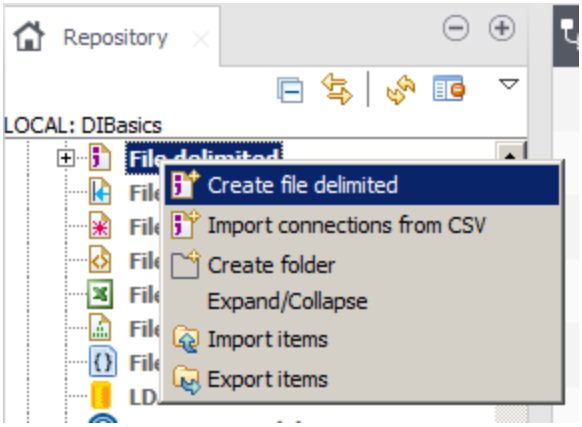
Overview

Your next step is to create metadata relating to the customer file so that you can use it to configure a component. You will also export metadata information as context variables. This provides the best way to manage your variables.

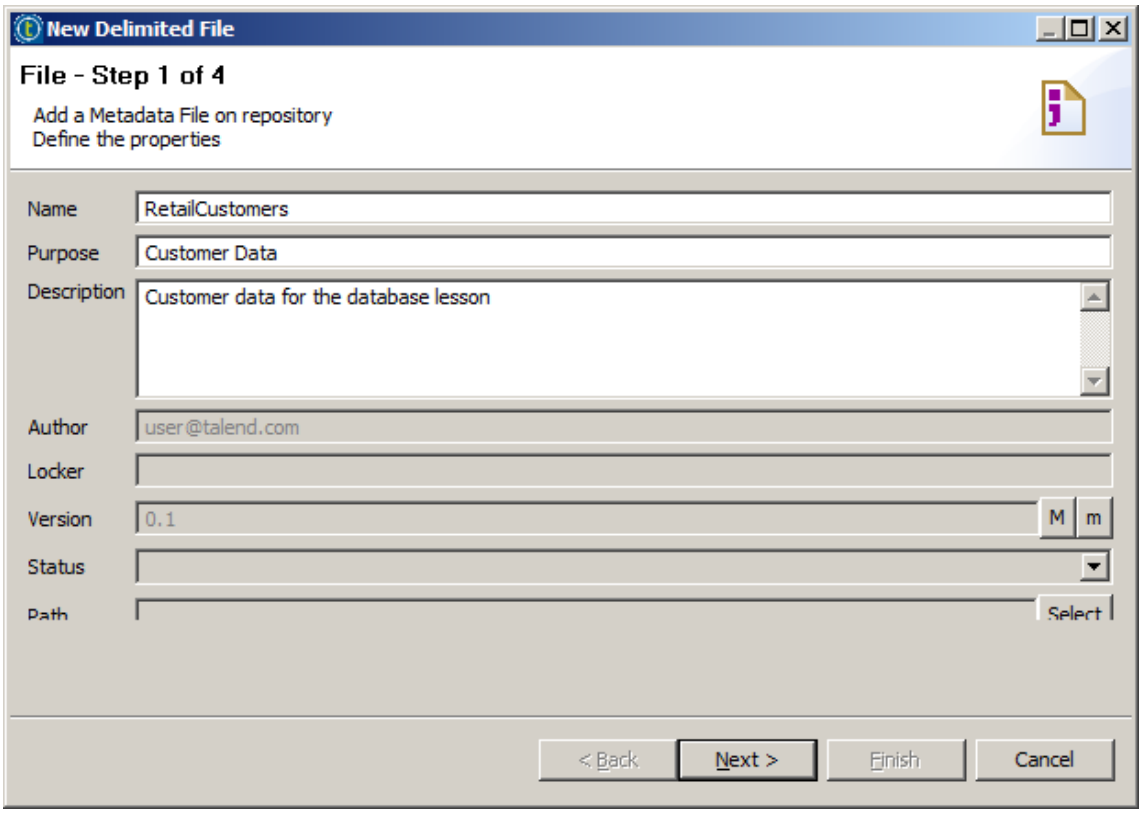
Create file metadata

1. CREATE A FILE DELIMITED METADATA IN THE REPOSITORY

   Right-click **Repository > Metadata > File delimited** and select **Create file delimited**.



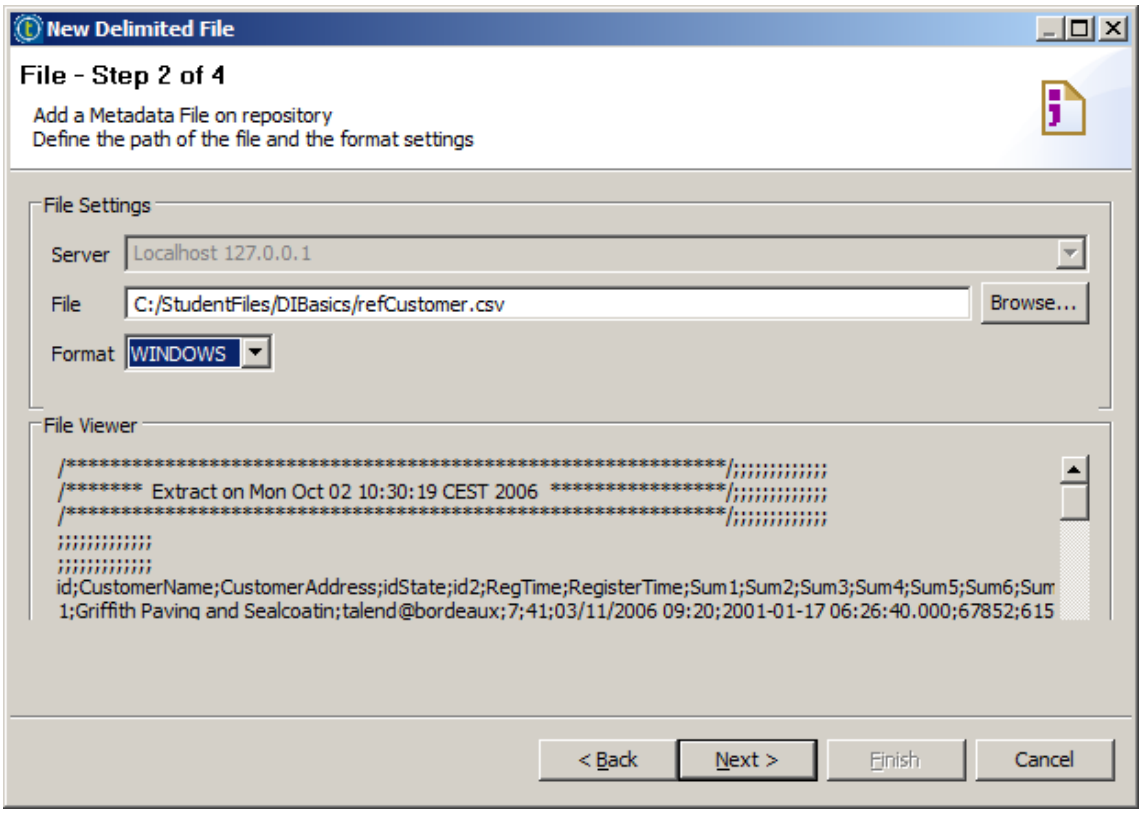
   Name the metadata *RetailCustomers*, provide a purpose and description, and click **Next**.



2. SET FILE PATH AND FORMAT

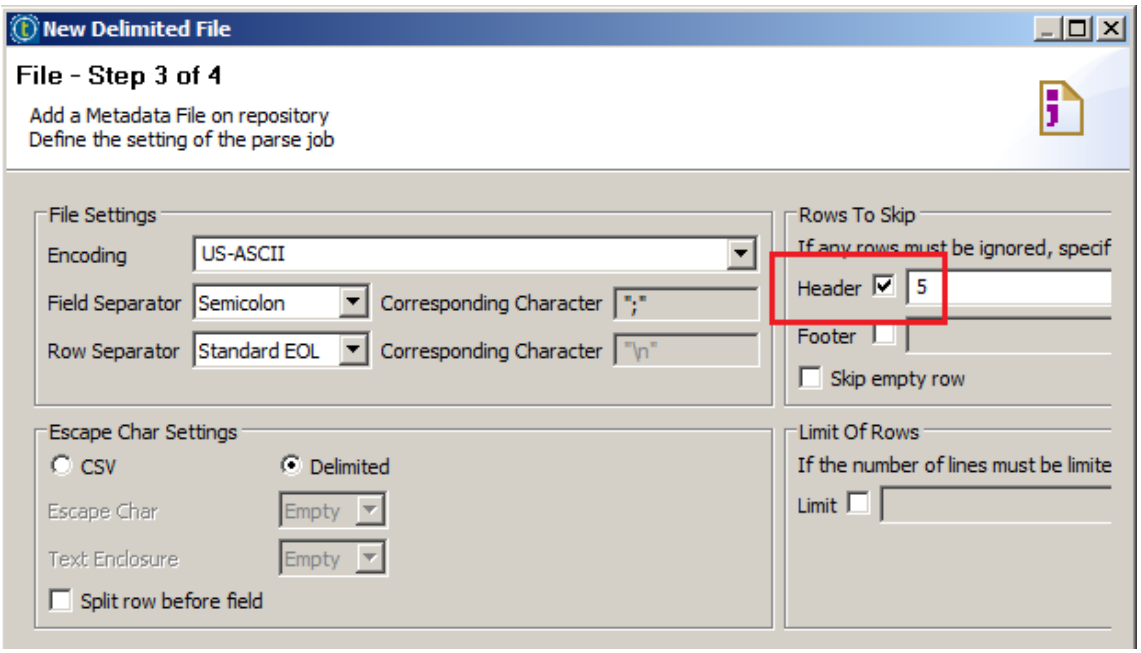
   Set **File** to *C:/StudentFiles/DIBasics/refCustomer.csv*, and change the **Format** to *WINDOWS*. Click **Next**.

   NOTE:
   Since this file path is not processed by the generated code, enclosing quotation marks are not necessary.



3. CONFIGURE THE FILE PARSER

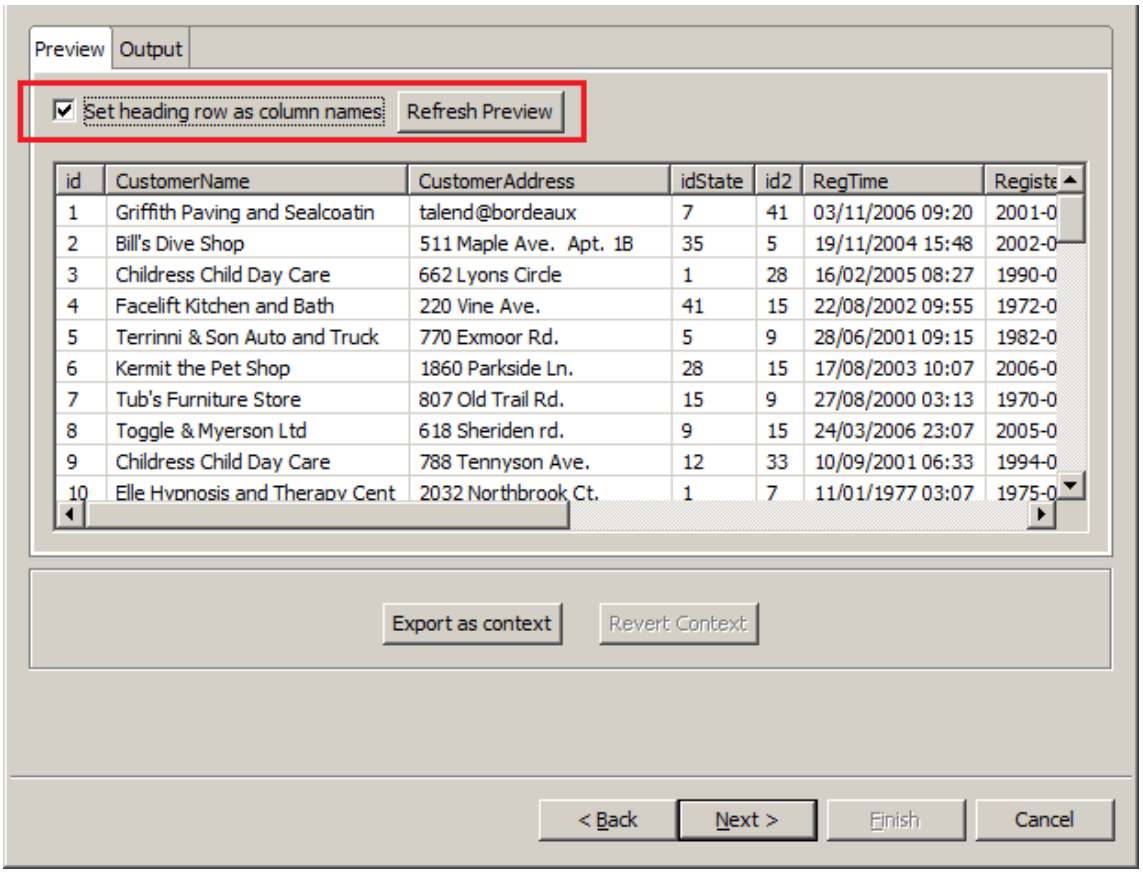
In the **Rows To Skip** section, set **Header** to *5*. This will cause the first five lines of the file, containing extraneous text, to be ignored.



Under **Preview**, select **Set heading row as column names** to indicate that the first row of meaningful data in the file actually specifies the column names.

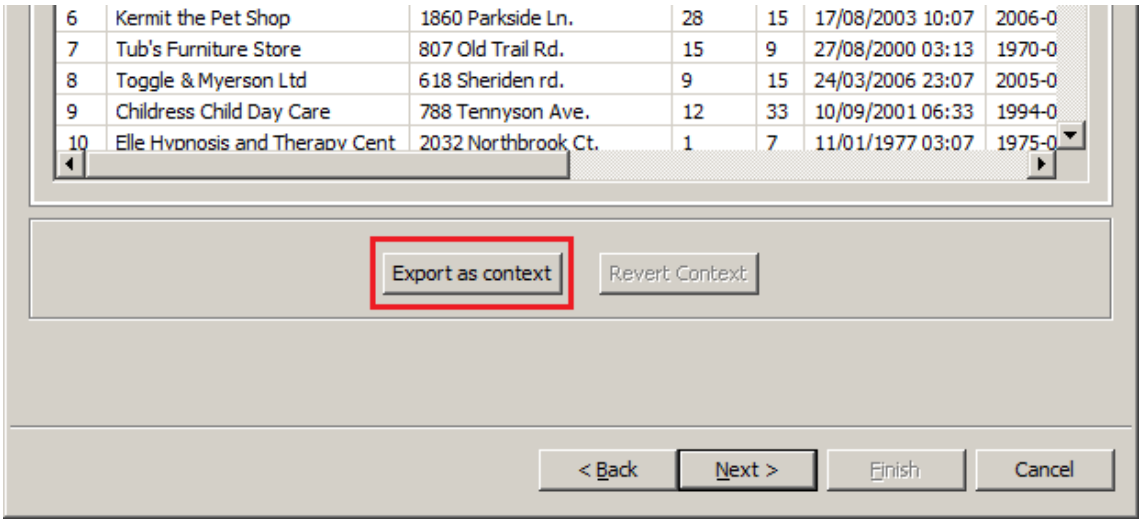
NOTE:
Enabling **Set heading row as column names** increases the **Header** value above from 5 to 6. Leave this updated value as is.

Click **Refresh Preview**. The **Preview** area displays the contents of the customer data file in a format that is easy to read.

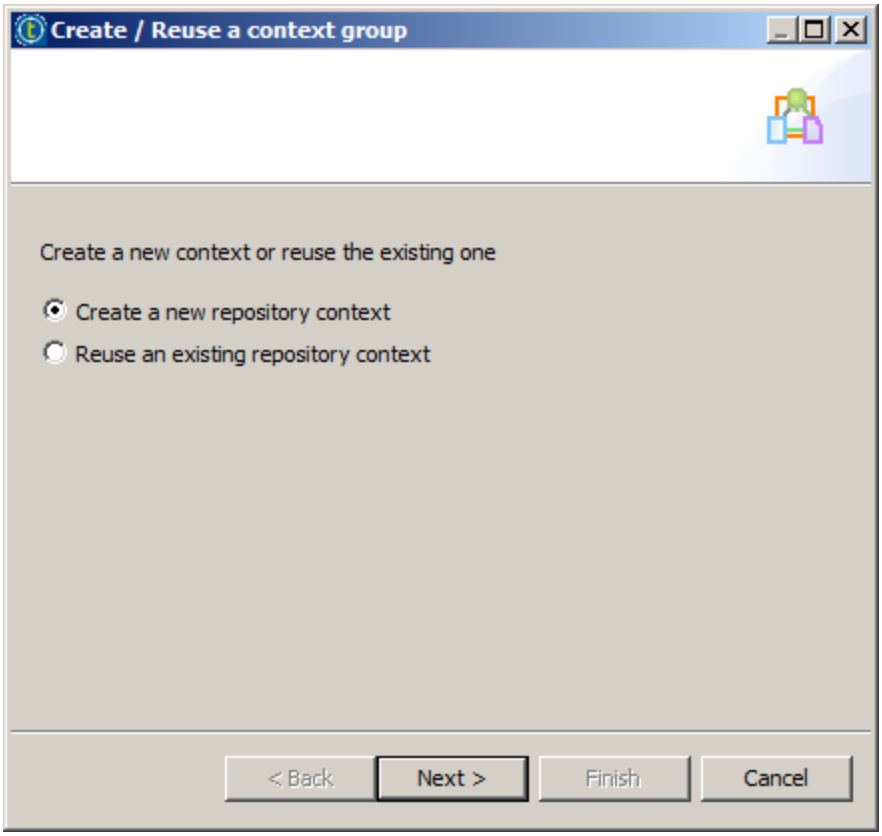


4. EXPORT A CONTEXT

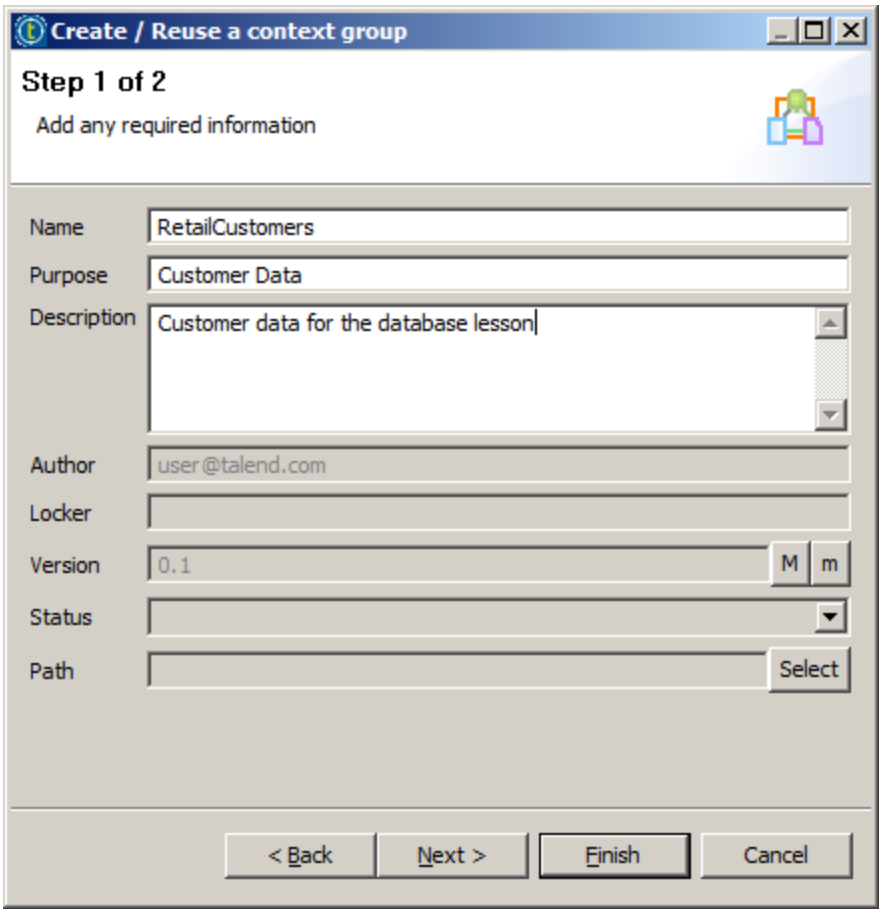
Before finalizing the metadata information, export this configuration as a collection of context variables by clicking the **Export as context** button.



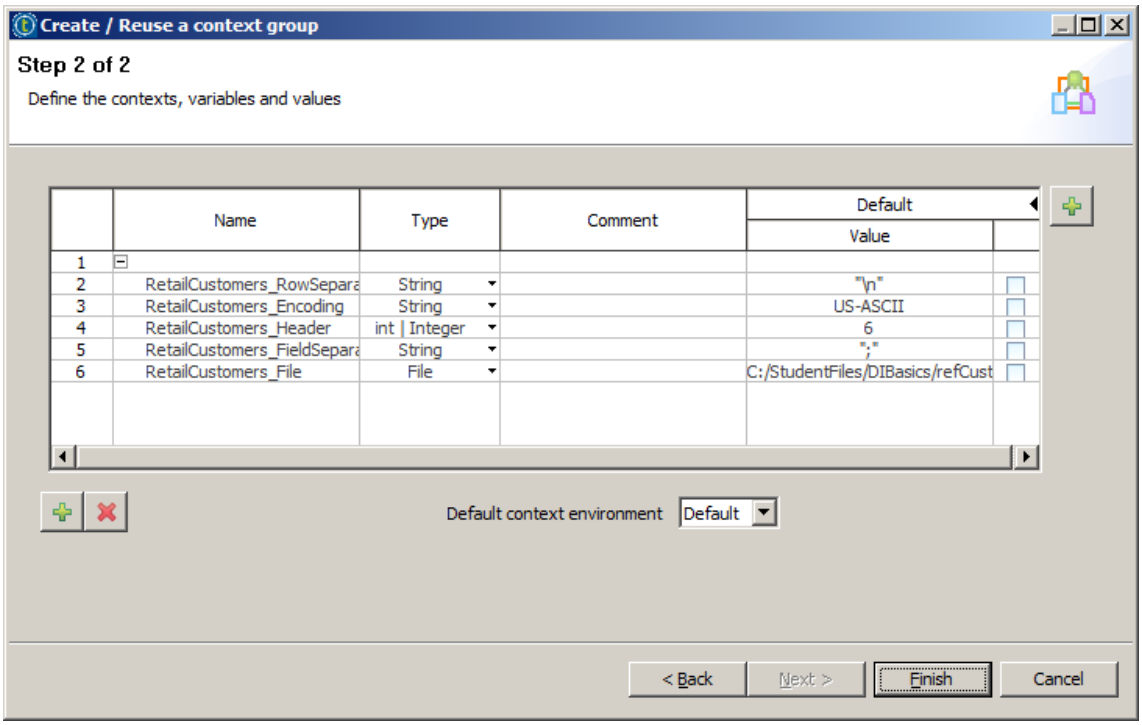
In the **Create/Reuse a context group** window, select **Create a new repository context**, and click **Next**.



Leave *RetailCustomers* for the **Name**, populate the **Purpose** and the **Description** fields, and click **Next**.



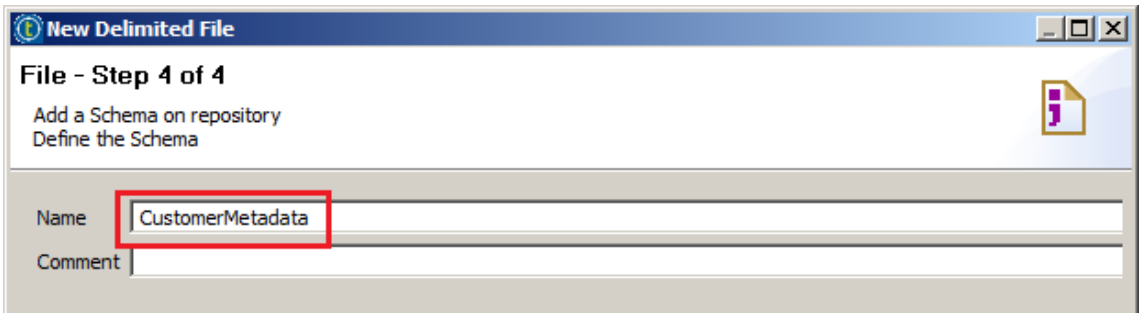
Examine the context variables that will be created, then click **Finish**.



Do not close the **New Delimited File** window yet, but do click **Next** to advance to the final step.

5. CONFIGURE THE SCHEMA

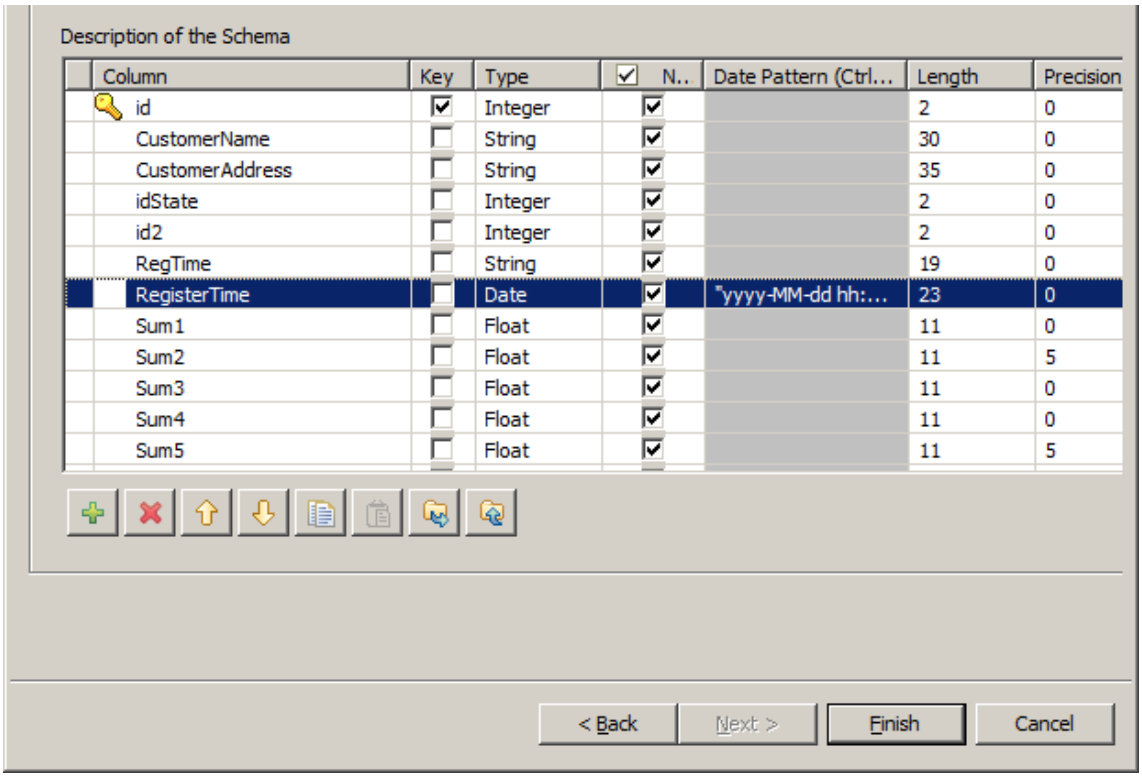
Change the name of the schema to *CustomerMetadata*.



Talend Studio derives a schema from the data, but some corrections must be made to the guessed values. Make the following changes to the **Description of the schema**:

- specify the column *id* as the key
- change the length of *CustomerAddress* to *35*
- configure all the columns *Sum1* through *Sum7* as type *Float* with a length of *11*
- configure the *RegTime* column as type *String* with a length of *19*
- configure the column *RegisterTime* as type *Date*. For the format, enter a custom value of *"yyyy-MM-dd hh:mm:ss.SSS"*. Be sure to include the enclosing quotation marks.

When done, the schema should look like the figure below. Click **Finish** when done.



Next

Now that you have defined the metadata for the Customers reference file you can start Creating the metadata for the products file.

Creating Product Metadata

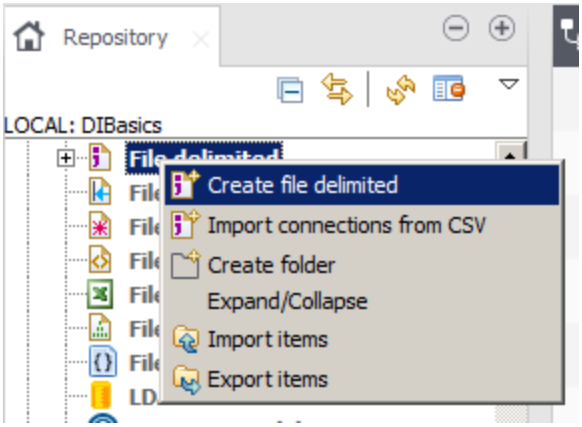
Overview

Your next step is to create metadata relating to the the products file so that you can use it to configure a component. As before, you will again export the metadata information as context variables.

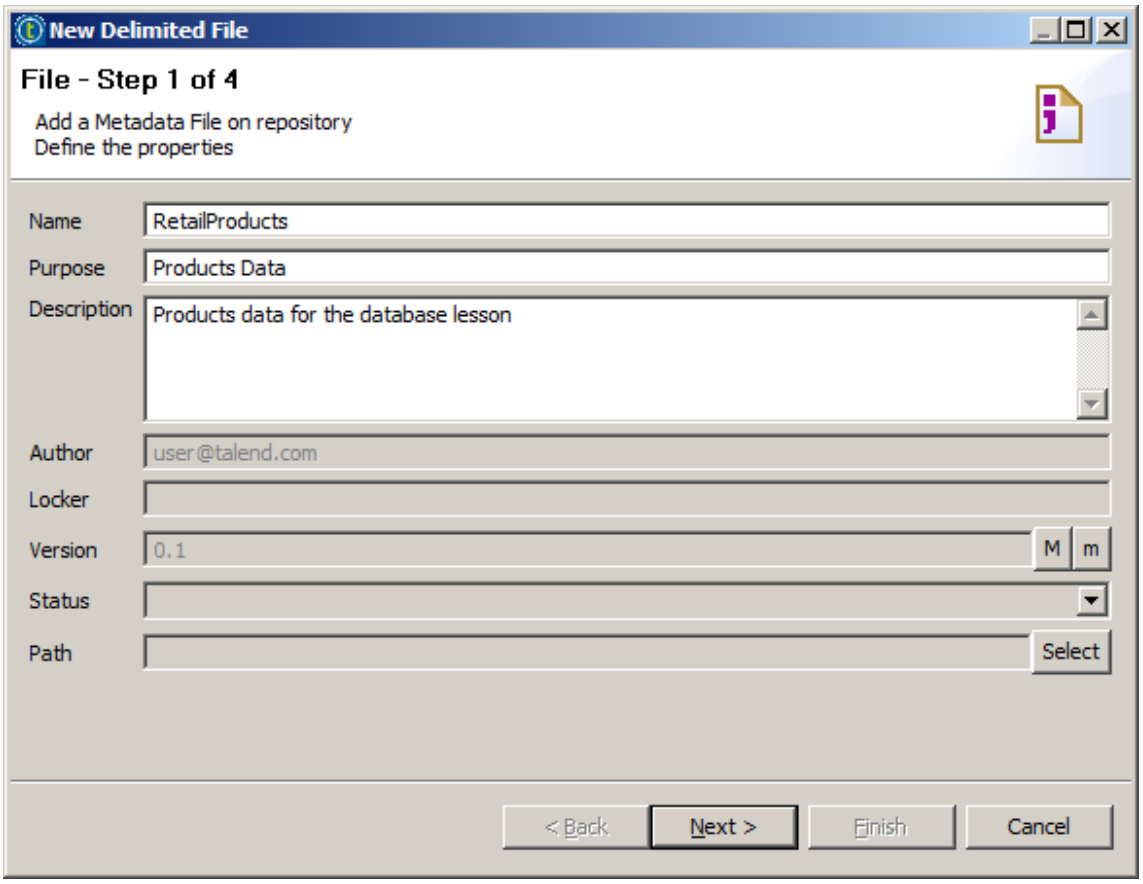
Create File Metadata

1. CREATE A FILE DELIMITED METADATA IN THE REPOSITORY

   Right-click **Repository > Metadata > File delimited** and select **Create file delimited**.

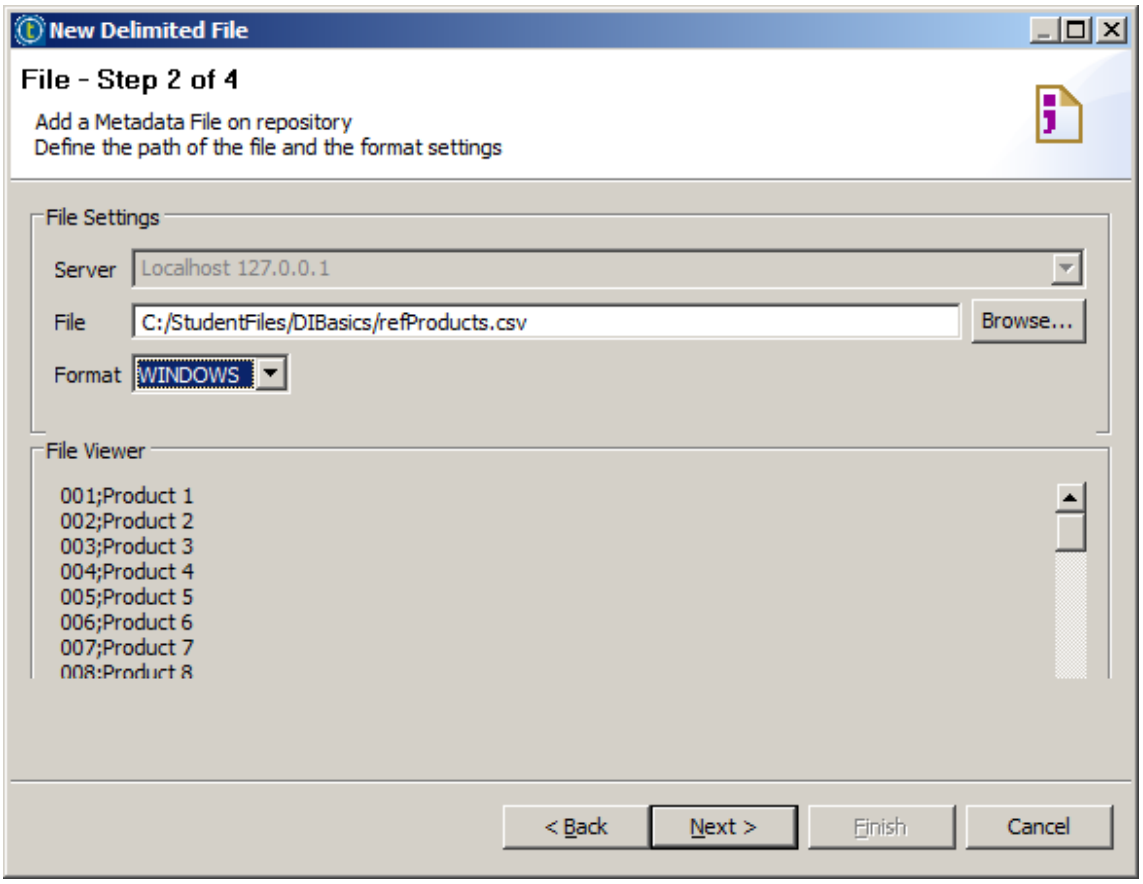


   Name the metadata *RetailProducts*, provide a purpose and description, and click **Next**.



2. SET FILE PATH AND FORMAT

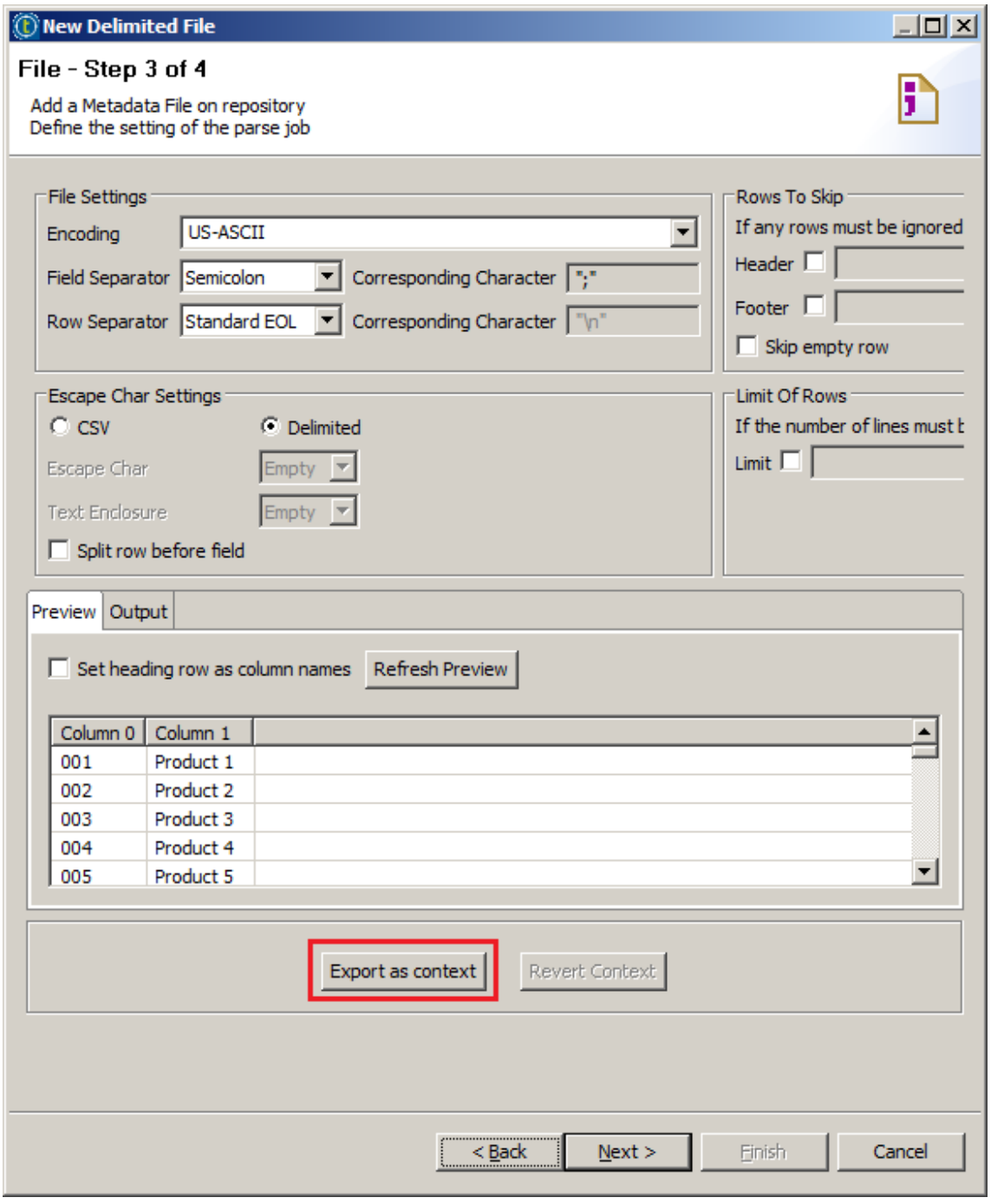
   Set **File** to *C:/StudentFiles/DIBasics/refProducts.csv*, and change the **Format** to *WINDOWS*. Click **Next**.



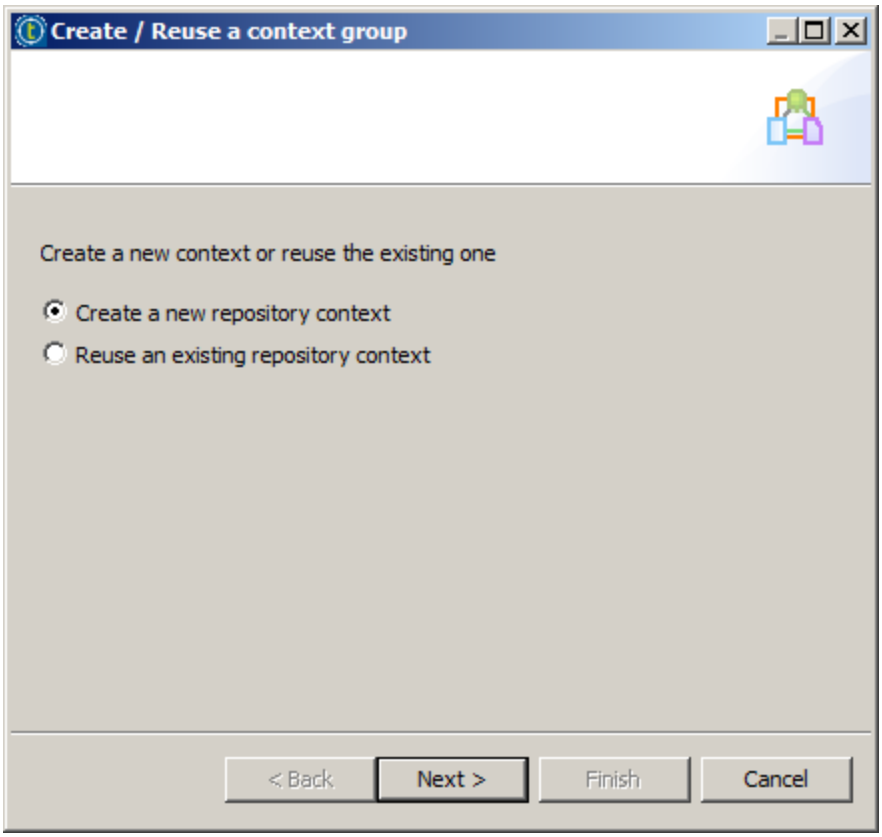
3. EXPORT A CONTEXT

   There are no header rows in this file, so no additional configuration is needed for the parsing.

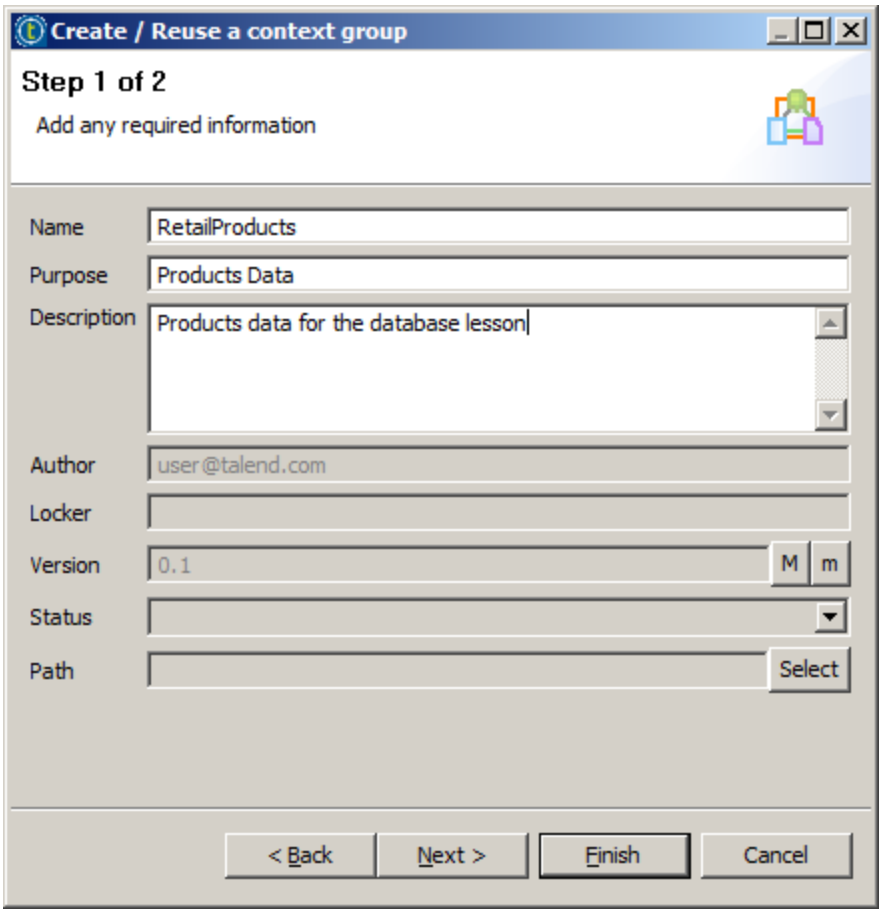
   Click **Export as context** to export all the information as context variables.



In the **Create/Reuse a context group** window, select **Create a new repository context**, and click **Next**.

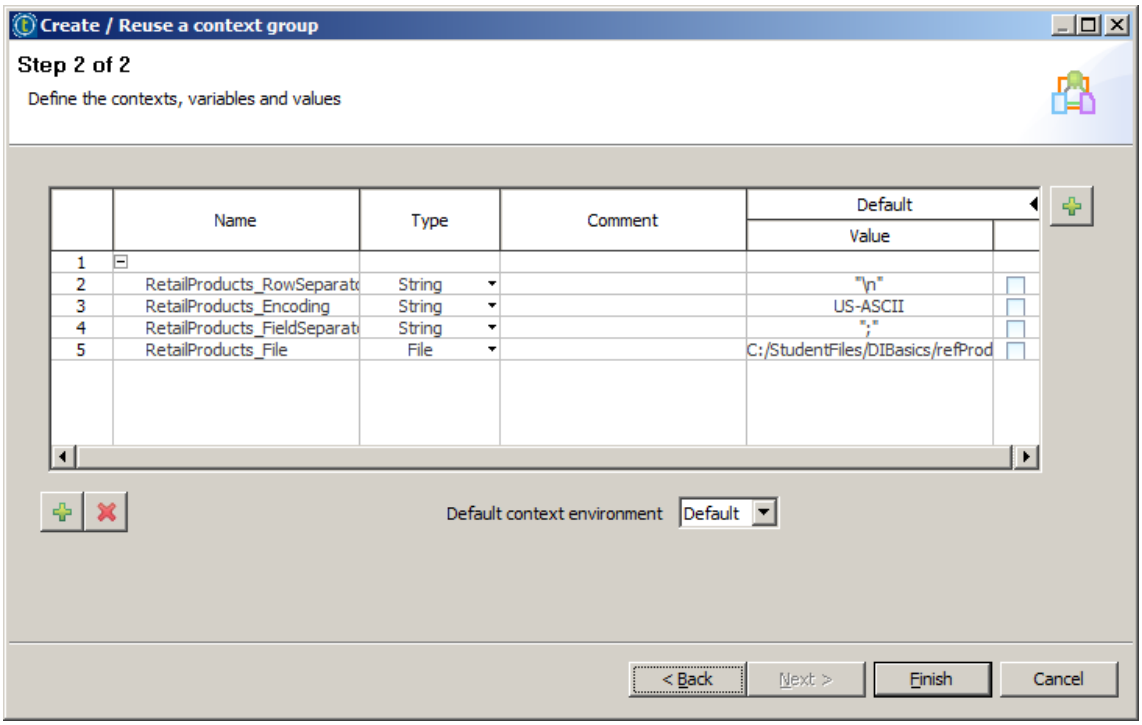


Leave *RetailProducts* for the **Name**, populate the **Purpose** and the **Description** fields, and click **Next**.



4. VERIFY THE CONTEXT VARIABLES

   Examine the context variables that will be created, then click **Finish**.



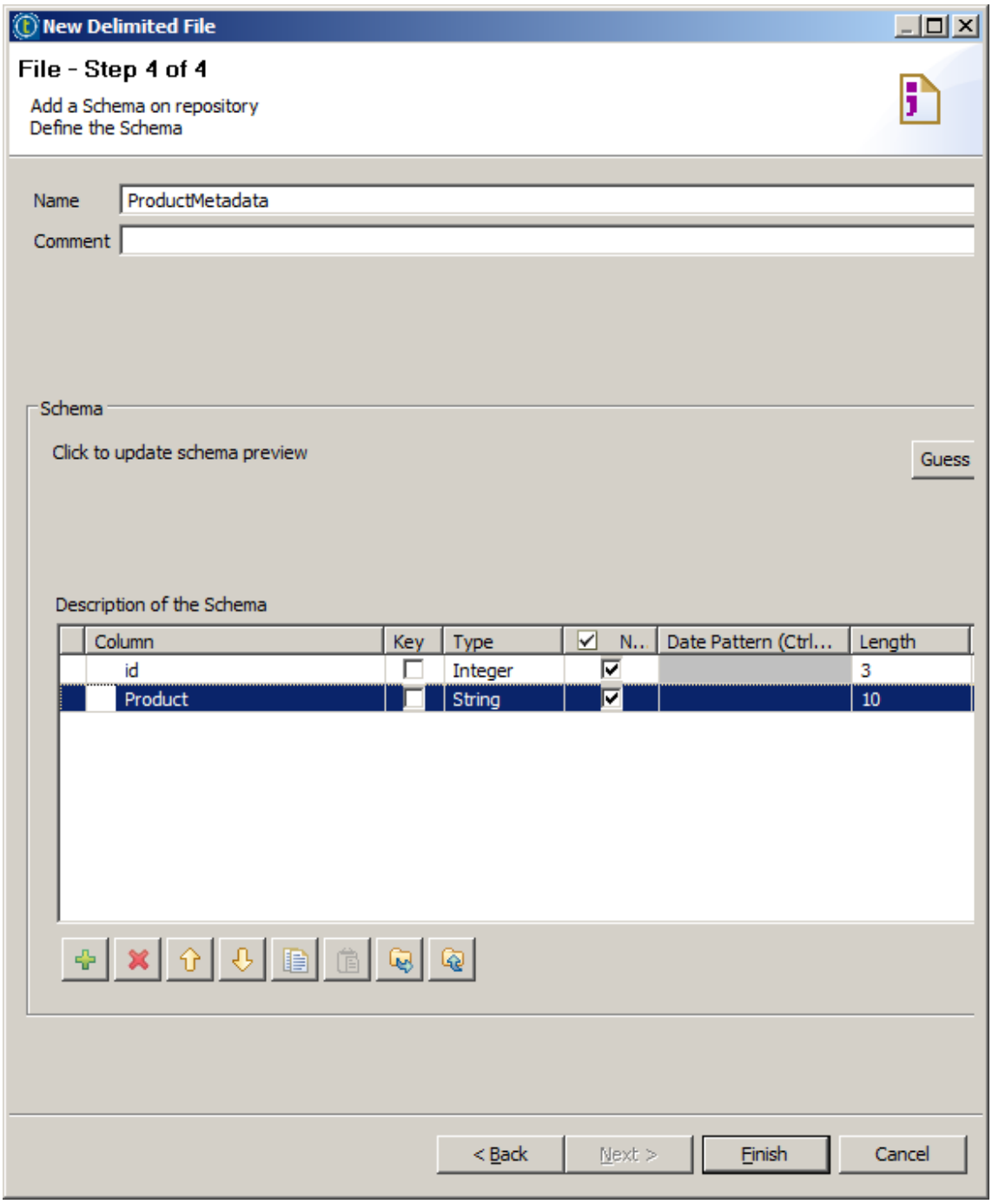
Do not close the **New Delimited File** window yet, but do click **Next** to advance to the final step.

5. CONFIGURE THE SCHEMA

Change the name of the schema to *ProductMetadata*. Make the following changes to the **Description of the schema**:

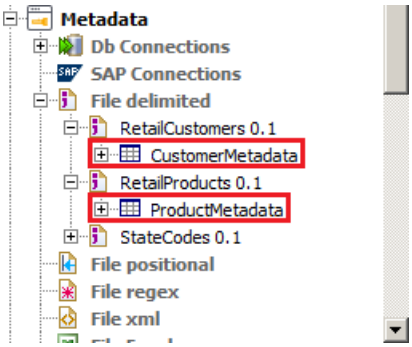
- specify *id* as the name of the first column, and set its type to *Integer* with a length of *3*
- specify *Product* as the name of the second column, leaving its type as *String* with a length of *10*

When done, the schema should look like this. Click **Finish** when done.



6. CHECK THE METADATA

   Verify that the metadata for both Customer and Products appear in the **Repository**.



Next

You have now completed this lesson. It's time to Wrap-up.

Wrap-Up

In this lesson, you created a generic schema to define the structure of sales files.

Then you used the **tRowGenerator** to create data files for three shops. You used the copy and paste function for creating new subJobs that functioned similarly to an existing subJob.

You combined 3 subJobs in one Job to create three different output files.

You also defined the metadata for customers and products using delimited files.

All of this has been done in preparation for the next lesson, where you will begin to progress beyond files to working with databases.

Next step

Congratulations! You have successfully completed this lesson. To save your progress, click **Check your status with this unit** below. To go to the next lesson, on the next screen, click **Completed. Let's continue >**.

LESSON 8

Working with Databases

This chapter discusses:

Working with Databases

Lesson Overview

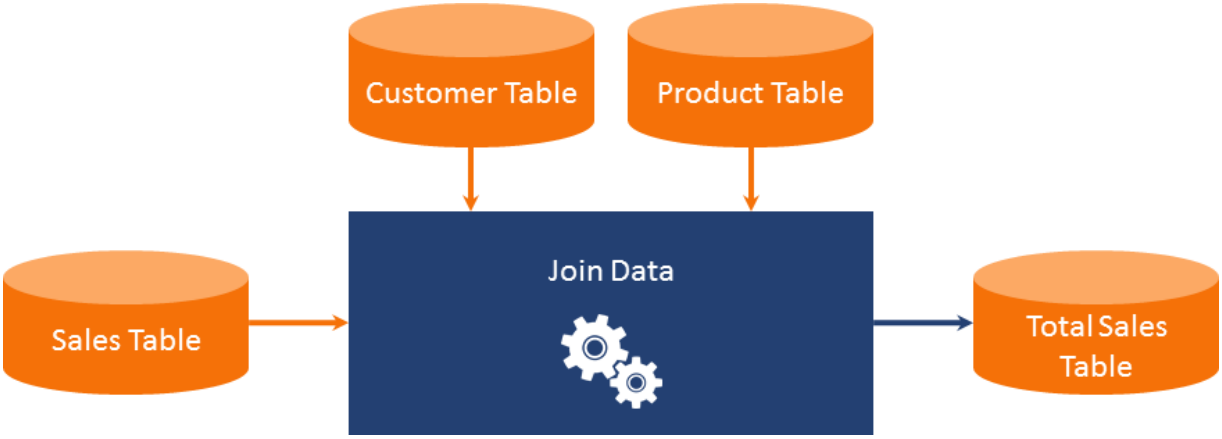
In this lesson, you explore the features of Talend Data Integration that allow you to interact with databases.

Recall that the scenario is that you are taking raw sales data from different retail stores, joining that data with existing customer and product information, and loading the combined data into the combined sales data table.

As part of the setup for this scenario, you will first create a database table for customer data, load that data from a text file into the database, and then repeat the process for product data:

You will then take data files representing sales information from three different stores, and then load that information into a database table.

Finally, you will join the sales table with the product and customer tables to create a combined master sales table.



As further exploration, you will generate total sales numbers by customer and by product.

Objectives

After completing this lesson, you will be able to:

- Connect to a database from a Talend Job
- Use a component to create a database table
- Write to and read from a database table from within a Talend Job
- Filter unique data rows
- Perform aggregate calculations on rows
- Write data to an XML file from a Talend Job

Next Step

The first step is to create database metadata

Creating Database Metadata

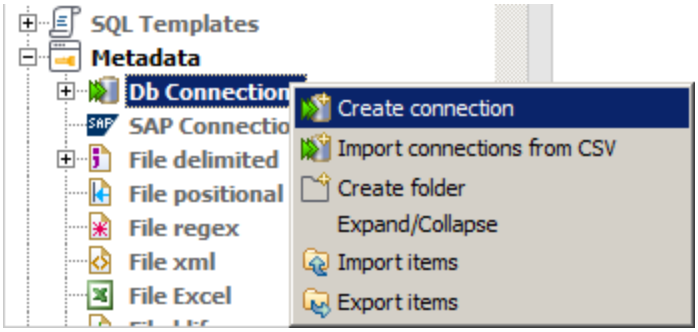
Overview

Now you are ready to create metadata regarding the connection to your local database server and the specific database. Creating metadata with this information in the Repository makes it easier to reference across different Jobs, and saves you from having to enter or update the information repeatedly for every Job that uses it.

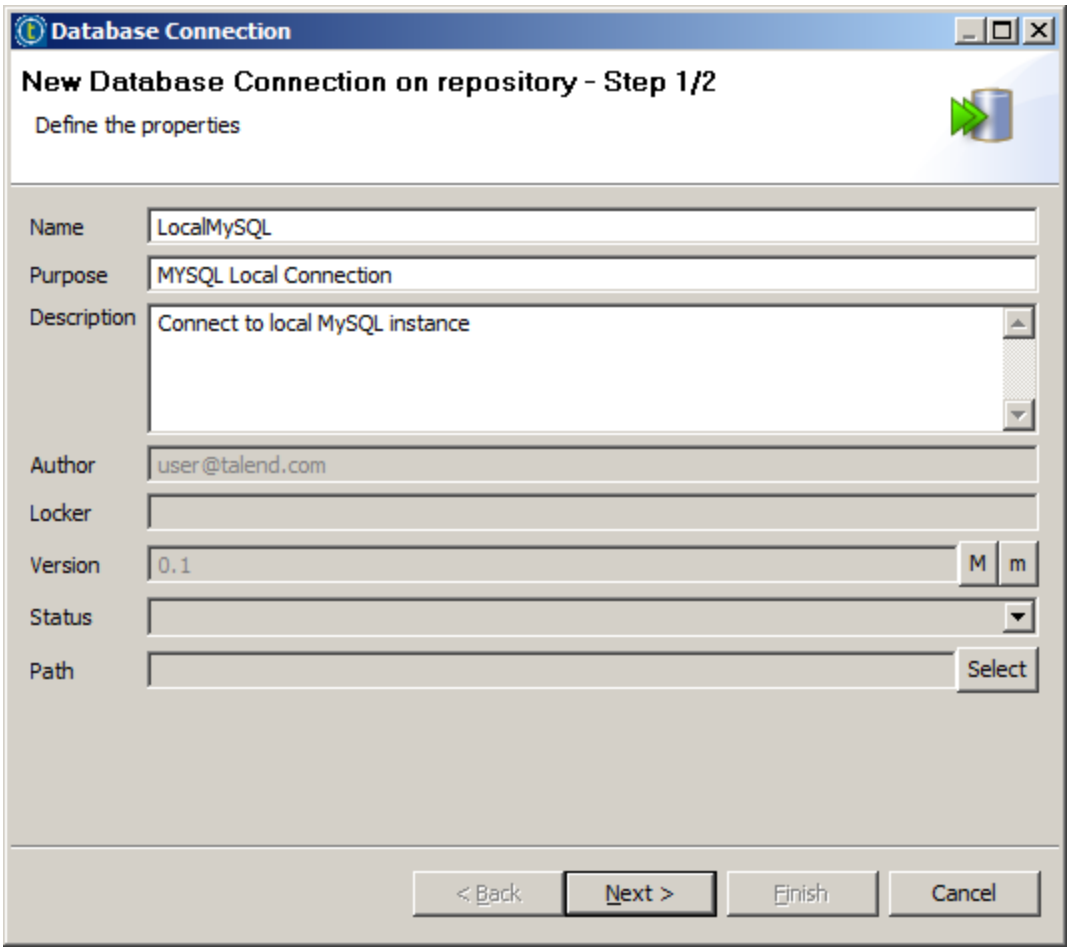
Create Database Connection Metadata

1. CREATE A DATABASE CONNECTION METADATA IN THE REPOSITORY

   Right-click **Repository > Metadata > Db Connections** and select **Create connection**.



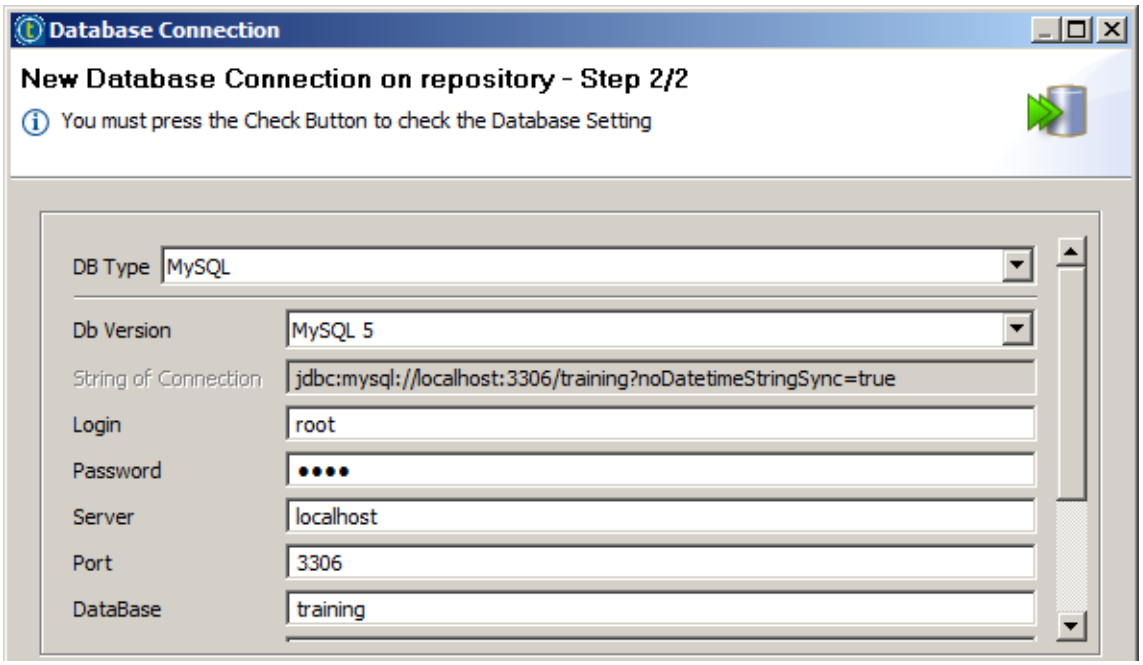
   Enter *LocalMySQL* for the the **Name**, and provide a **Purpose** and a **Description**. Click **Next**.



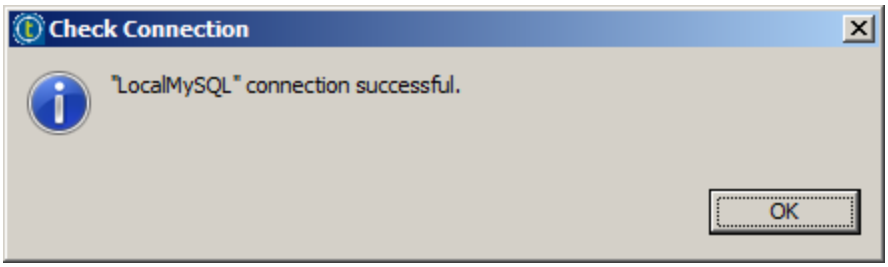
Specify *MySQL* for **DB Type**. Then configure the connection as follows :

- Specify *root* for both **Login** and **Password**
- Specify *localhost* for **Server**
- Specify *training* for **DataBase**

NOTE:
Please note that this configuration information is specific to the training environment, where a MySQL database is already set up for you. In your own environment, you would provide the connection information appropriate for your configuration.

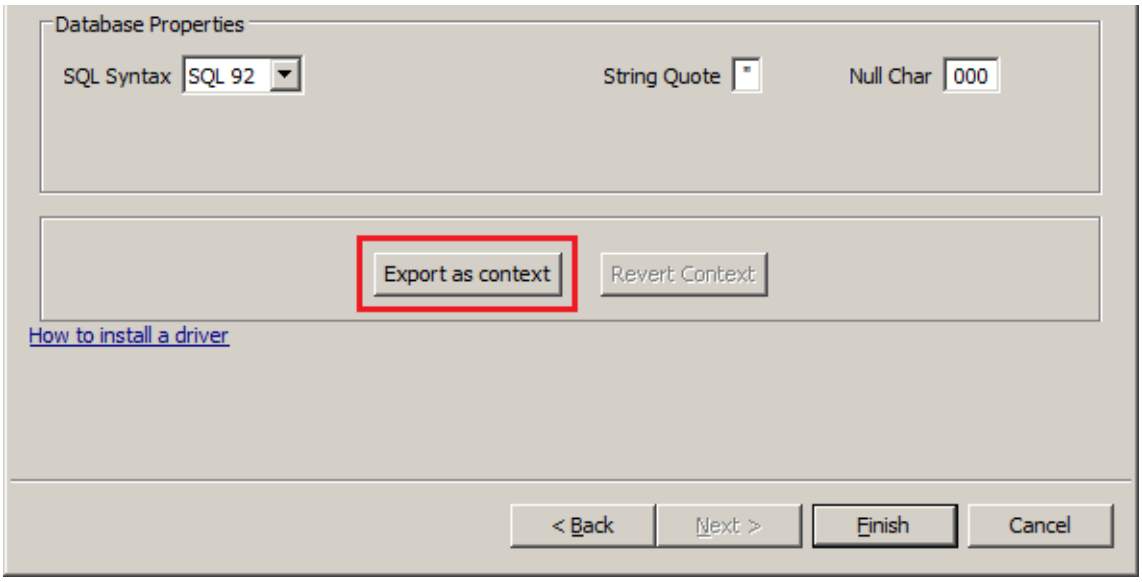


Now, click the **Check** button. Assuming all information was entered correctly, the check will succeed. If the connection is not successful, review and correct the connection information and try again.

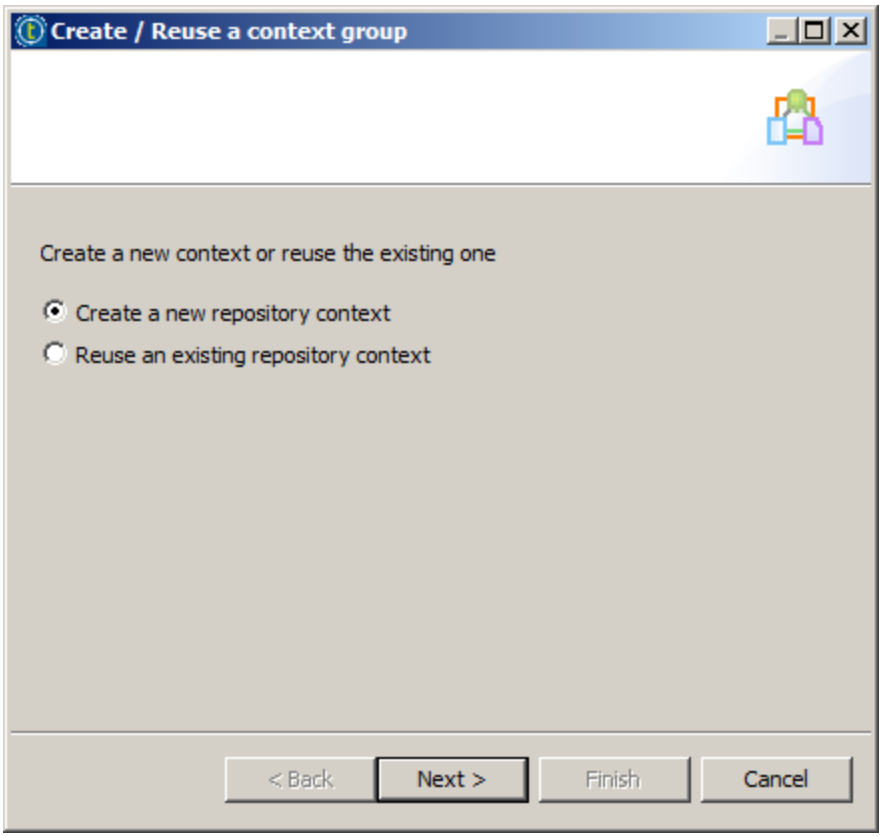


2. EXPORT A CONTEXT

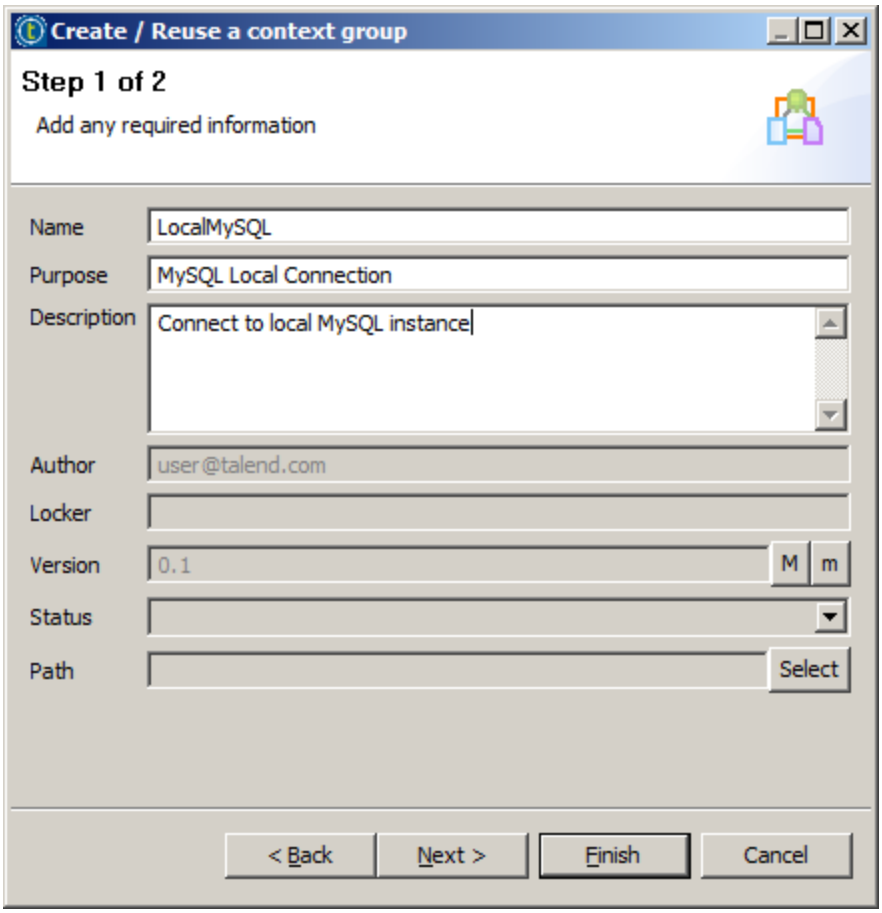
   Export this configuration as a collection of context variables by clicking the **Export as context** button.



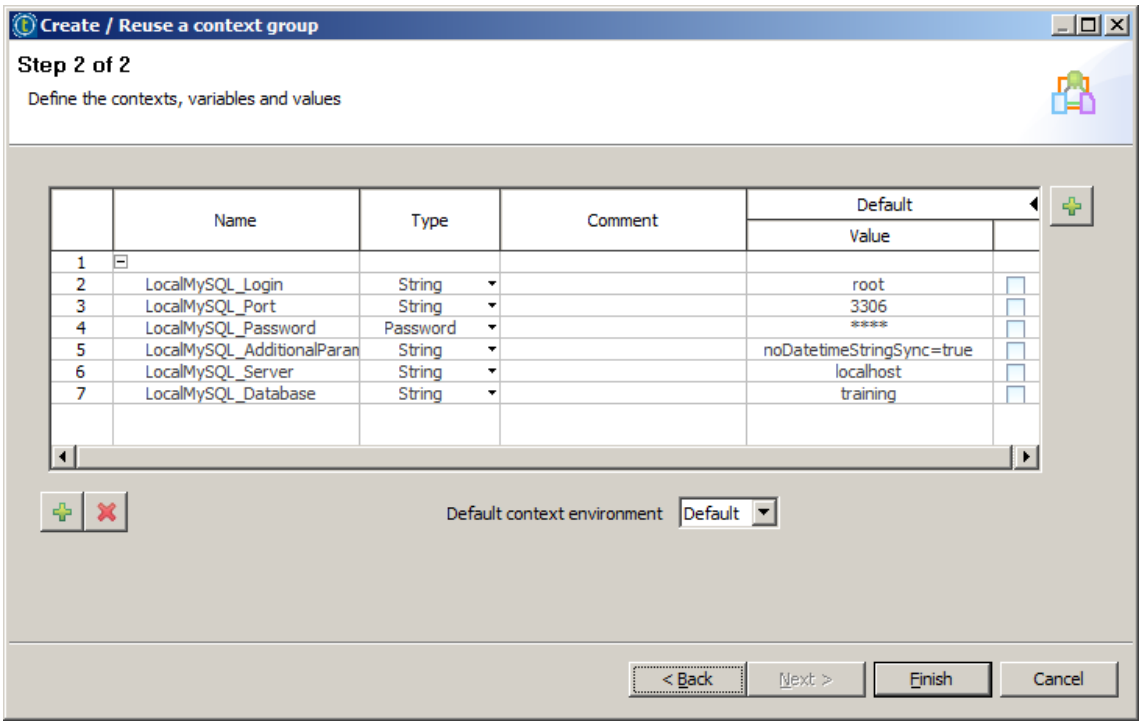
In the **Create/Reuse a context group** window, select **Create a new repository context**, and click **Next**.



Leave *LocalMySQL* for the **Name**, populate the **Purpose** and the **Description** fields, and click **Next**.

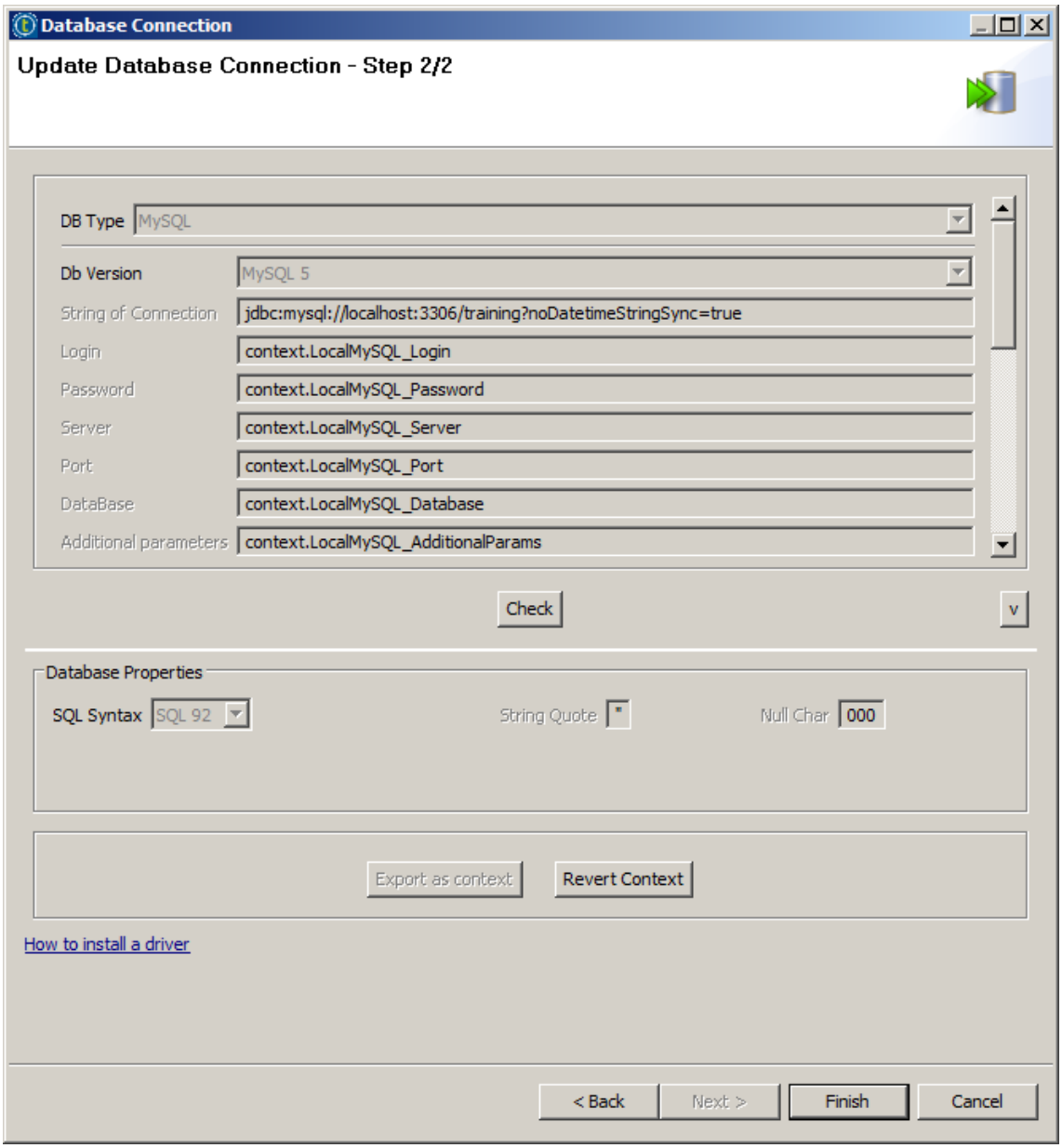


Examine the context variables that will be created, then click **Finish**.



3. FINALIZE THE DATABASE CONNECTION METADATA

   Going back to the **Database Connection** window, notice that the values you typed in each box are replaced with context variables. Click **Finish**.



Next

Now that connection information for your local MySQL database is set up, you can create the first database table.

Creating a Customer Table

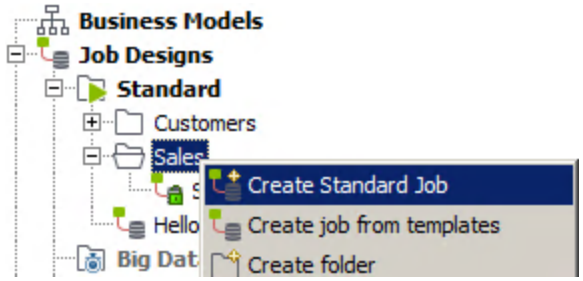
Overview

Your next step is to connect to an existing database, create a table for customer data, and then load data from a text file into that database table.

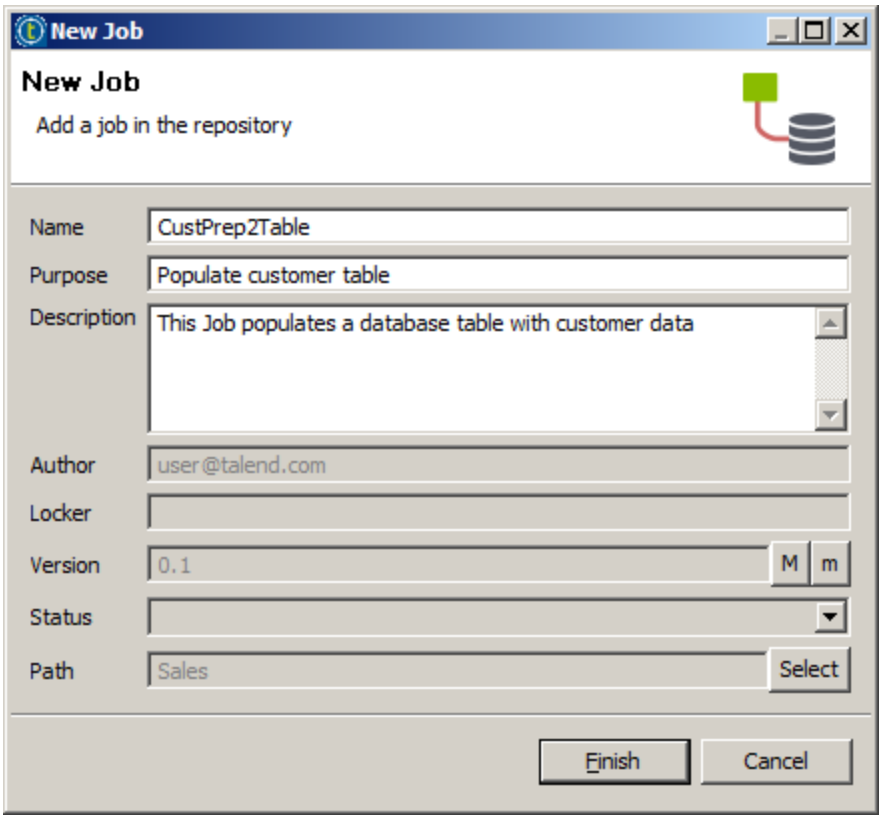
Create a New Job That Creates a Table

1. CREATE A JOB

   Right-click **Repository > Job Designs > Standard > Sales** and select **Create Standard Job**.

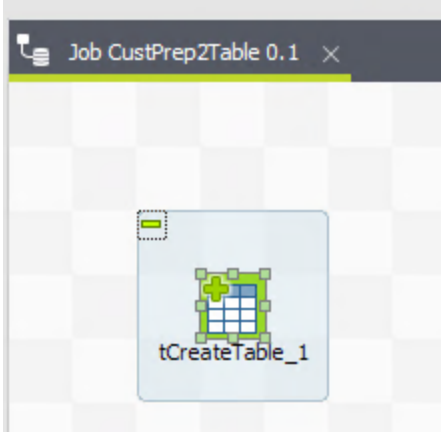


   Name the Job *CustPrep2Table*. Provide a **Purpose** and **Description** and click **Finish**.



2. ADD A COMPONENT TO CREATE A DATABASE TABLE

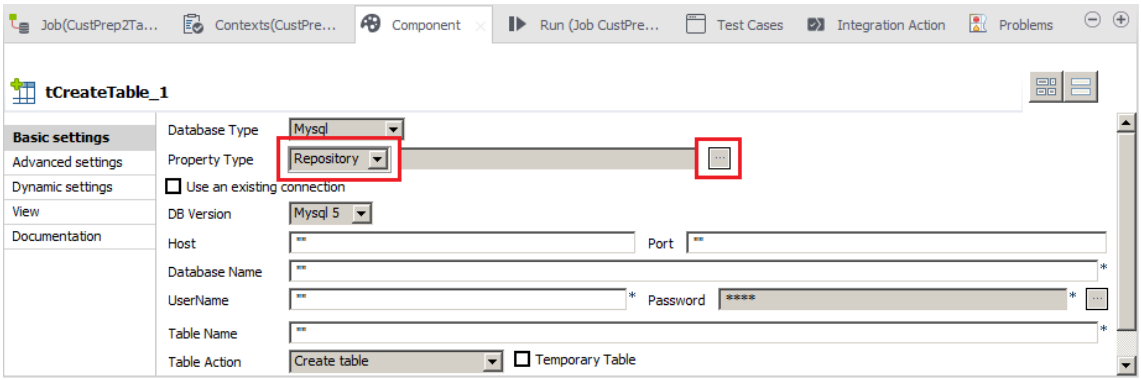
   Add a **tCreateTable** component. As the name implies, this component creates a table in a database.



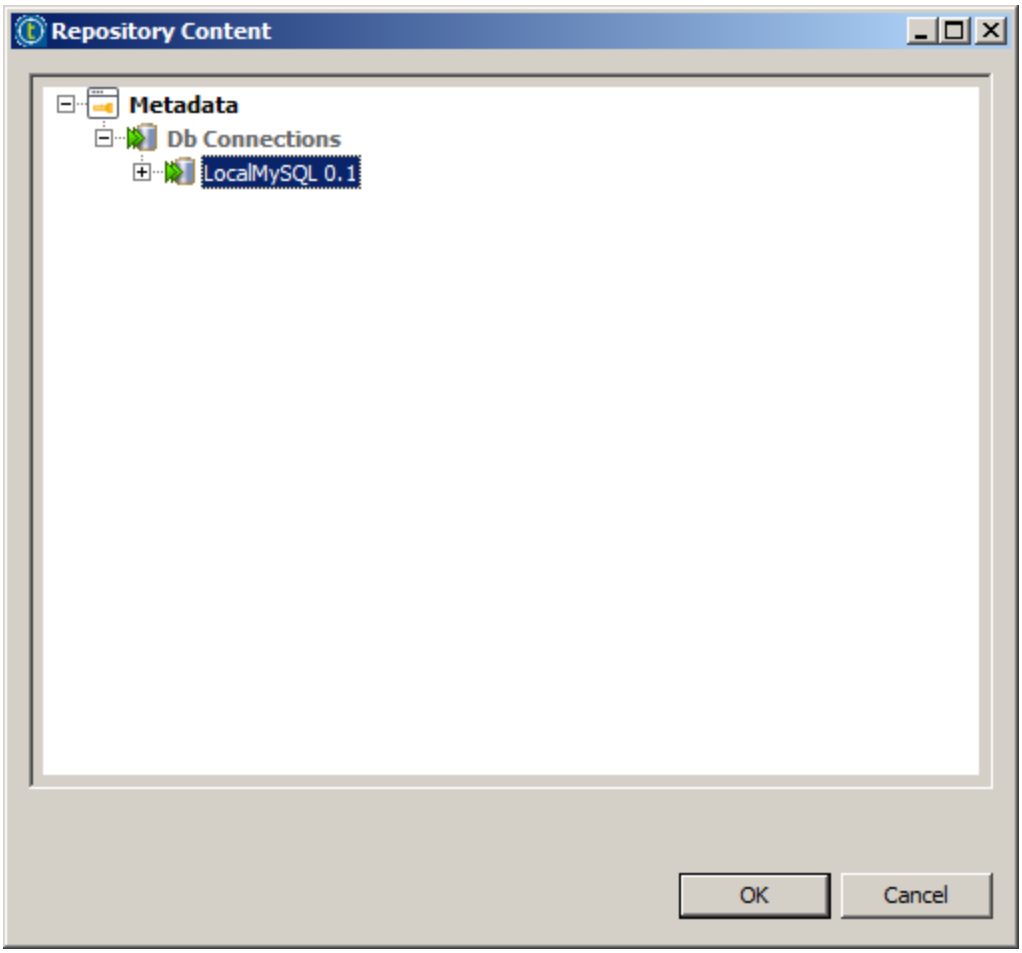
3. SPECIFY THE DATABASE CONNECTION

   Double-click the **tCreateTable_1** component to open the **Component** view.

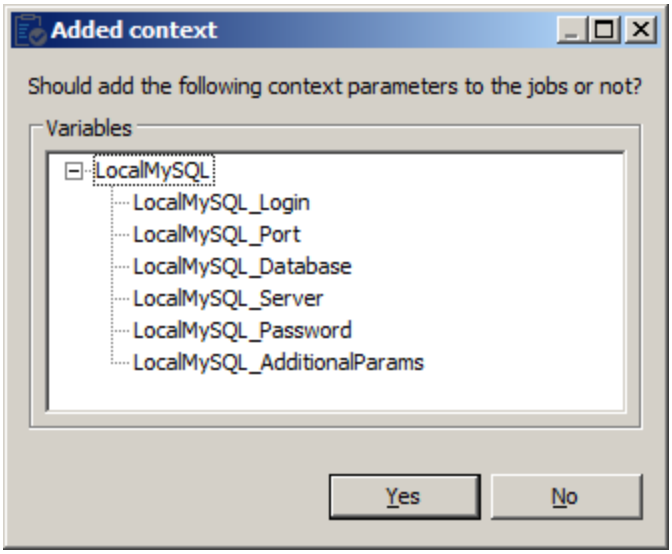
   Specify *Repository* for the **Property Type** list to use the stored database connection information, then click the button marked with an ellipsis **[...]** next to the field **Property Type**.



   In the **Repository Content** window that appears, select **Metadata > Db Connections > LocalMySQL** and click **OK**.

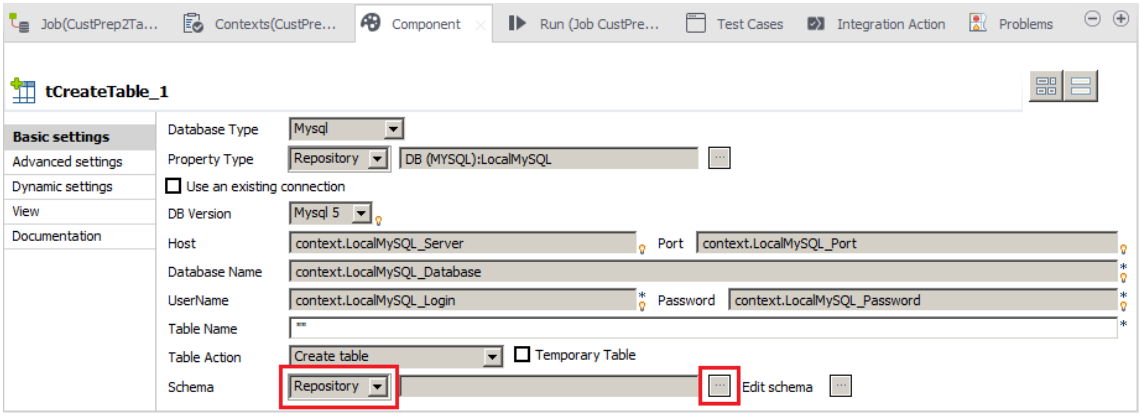


When the **Added context** window appears, click **Yes** to add the context parameters to the Job.

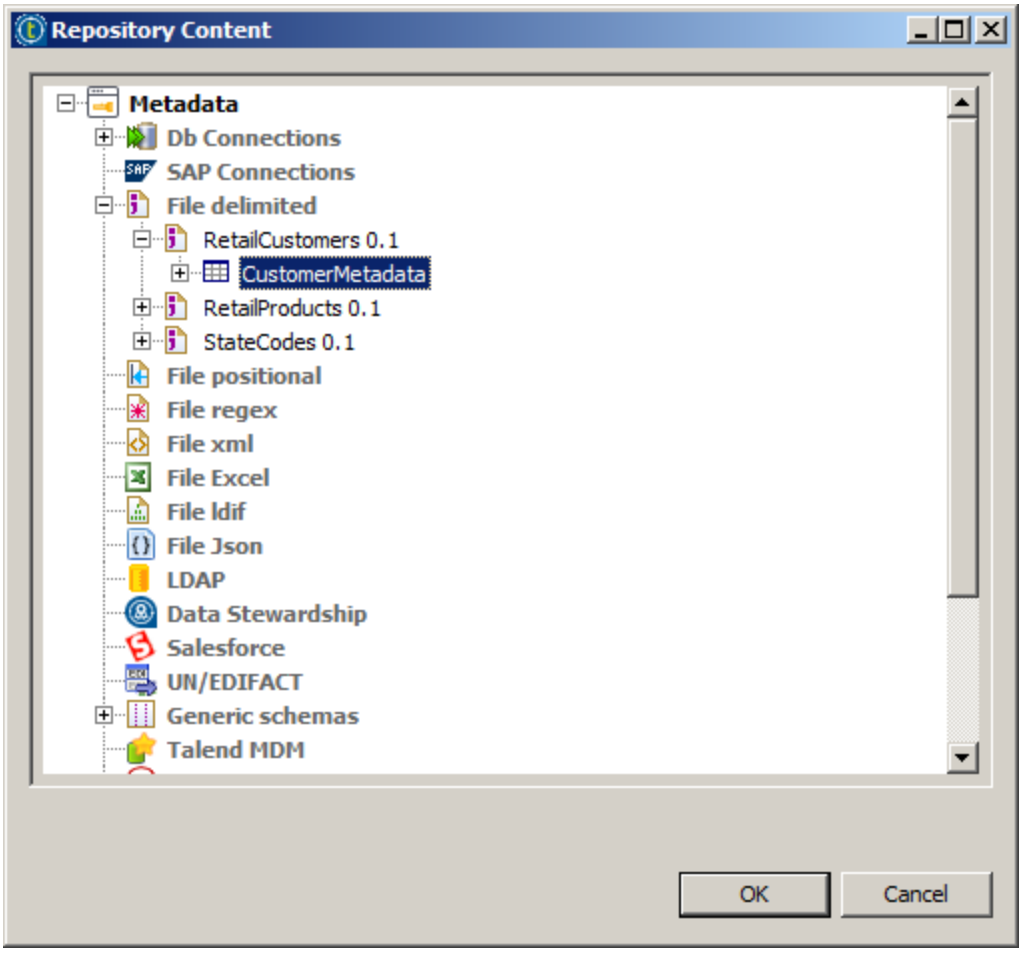


4. SPECIFY THE SCHEMA

Specify *Repository* for the **Schema** and click the button marked with an ellipsis (to the left of **Edit schema**) to choose the schema.



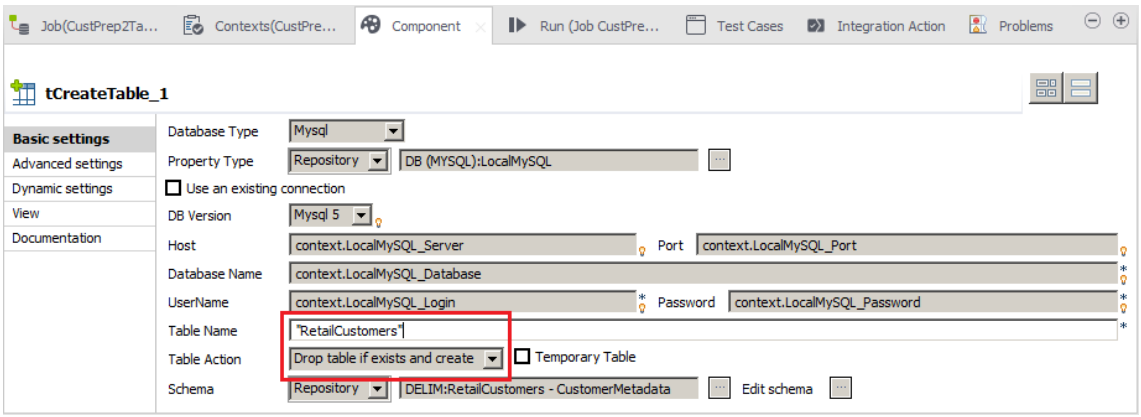
In the **Repository Content** window that appears, select **Metadata > File delimited > RetailCustomers > CustomerMetadata** and click **OK**.



Recall that this is the schema you defined earlier.

5. SPECIFY THE TABLE NAME AND ACTION

   Specify *Drop table if exists and create* for **Table Action**. Enter *"RetailCustomers"* for the **Table Name**.



Now this component is configured to connect to your local MySQL server and create a table named *RetailCustomers* with the stored *RetailCustomers* schema in the *training* database (the database that was created for you in this training environment).

The **Table Action** parameter is set to **Create table** by default, but that action can only happen once. By changing the action to **Drop table if exists and create**, you can run this Job repeatedly if necessary without causing an error; if the table already exists it will be deleted and a new one will be created.

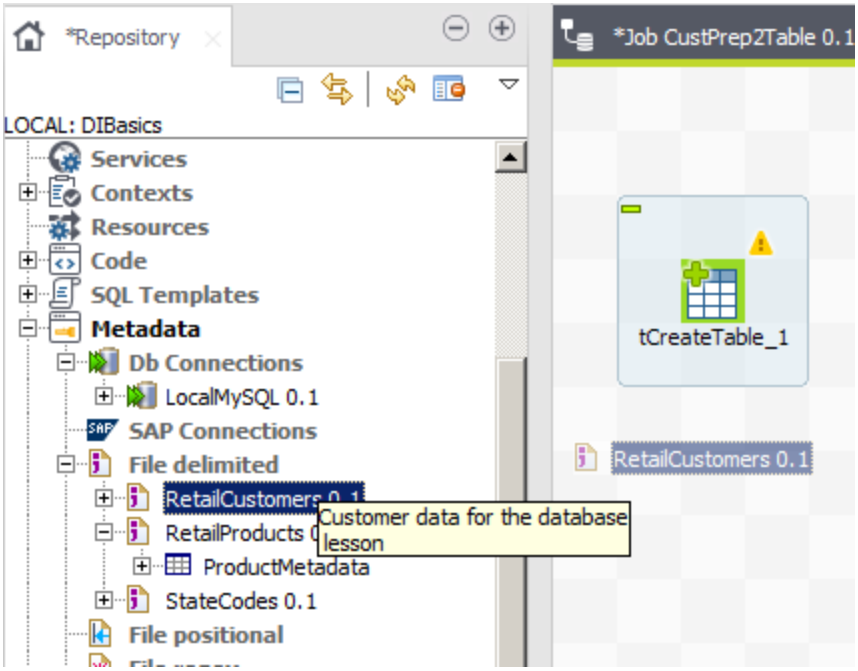
Populate the Customer Table

You are now going to transfer the data from the *C:/StudentFiles/DIBasics/refCustomer.csv* file into your database table. You have already created metadata for this file and for the database connection.

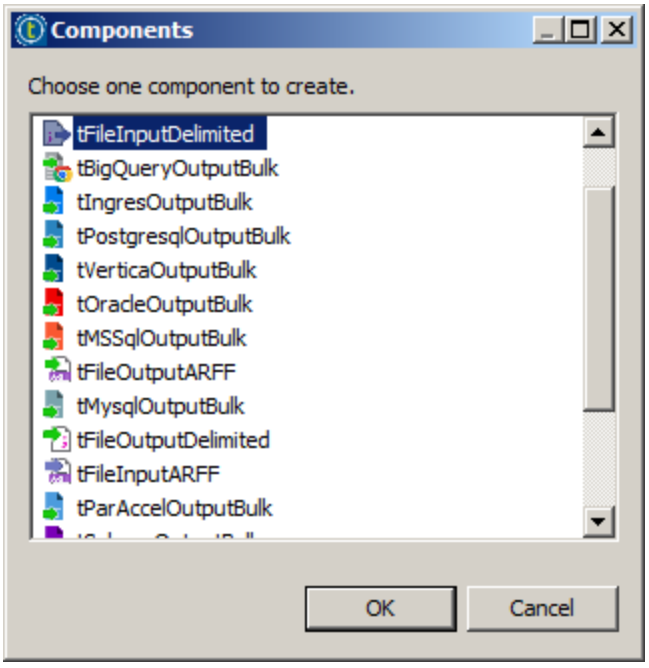
- You have a component, **tCreateTable**, that creates your database table.
- When the table is created, it will trigger an input component to read the information from your CSV file.
- Now you will choose a component that writes this input to your database table.

1. ADD A NEW COMPONENT BASED ON INPUT FILE METADATA

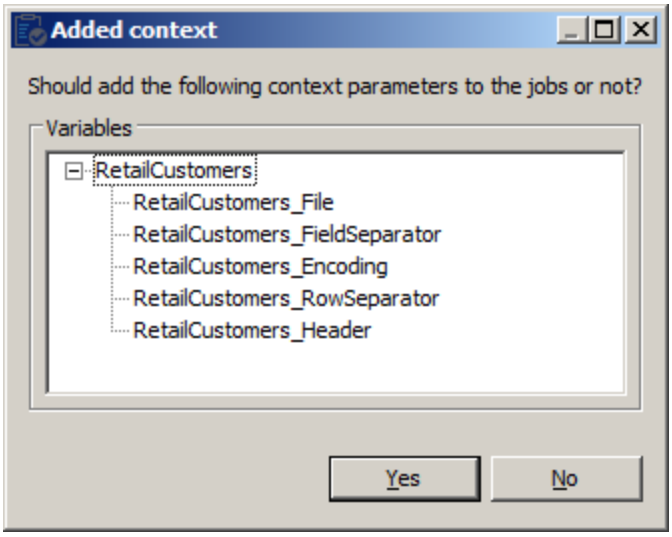
   Drag the **RetailCustomers** metadata onto the **Designer**, under the **tCreateTable_1** component.



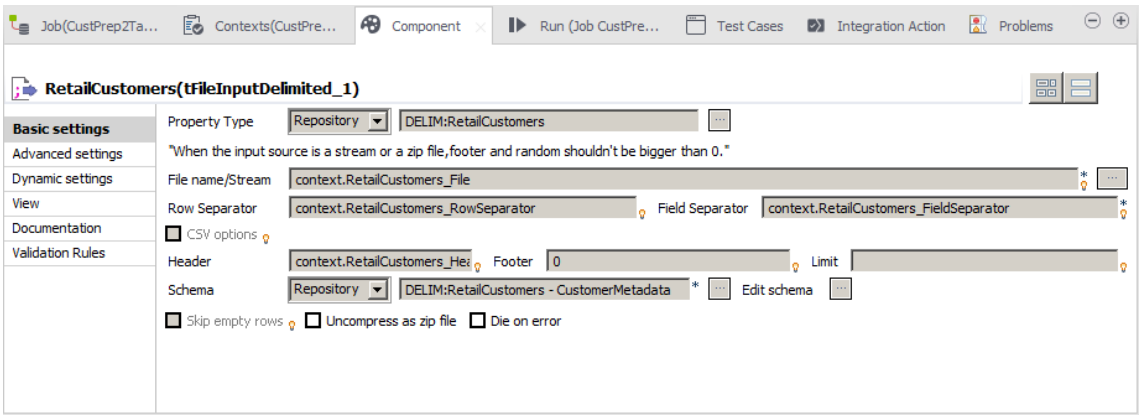
Choose **tFileInputDelimited** as the component type and click **OK**.



This file input component reads the customer data file. As you exported your information as context variables, you will be prompted to add context parameters to the Job. Click **Yes**.

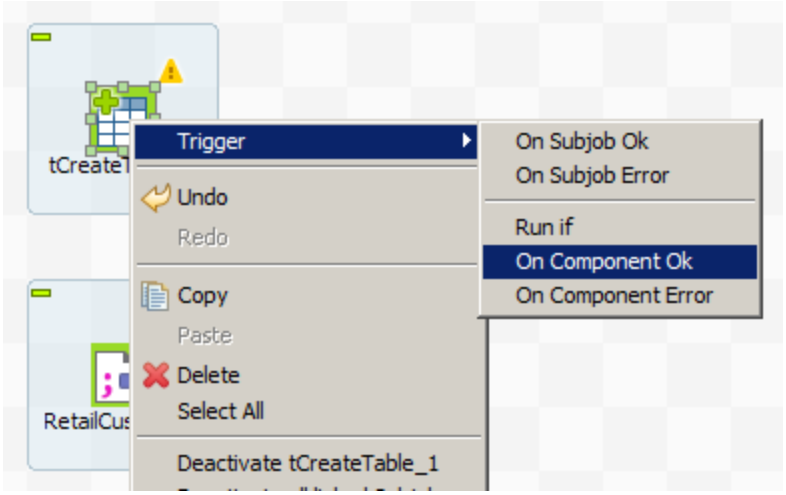


Check the component configuration by double-clicking the new component **RetailCustomers** to open the **Component** view. The information displayed is exactly as you entered it in the repository metadata and saved as context variables.



2. TRIGGER THE INPUT COMPONENT

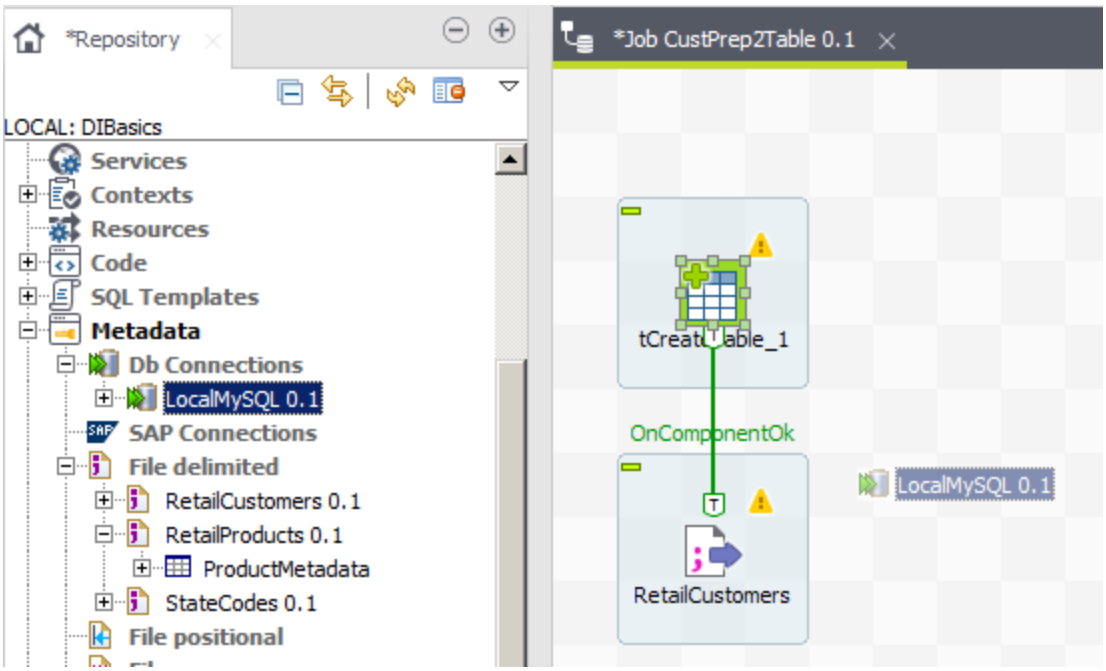
Connect the **tCreateTable** component to the **RetailCustomers** component with an **OnComponentOK** trigger by right-clicking on **tCreateTable_1** and selecting **Trigger > On Component Ok**, then clicking **RetailCustomers**.



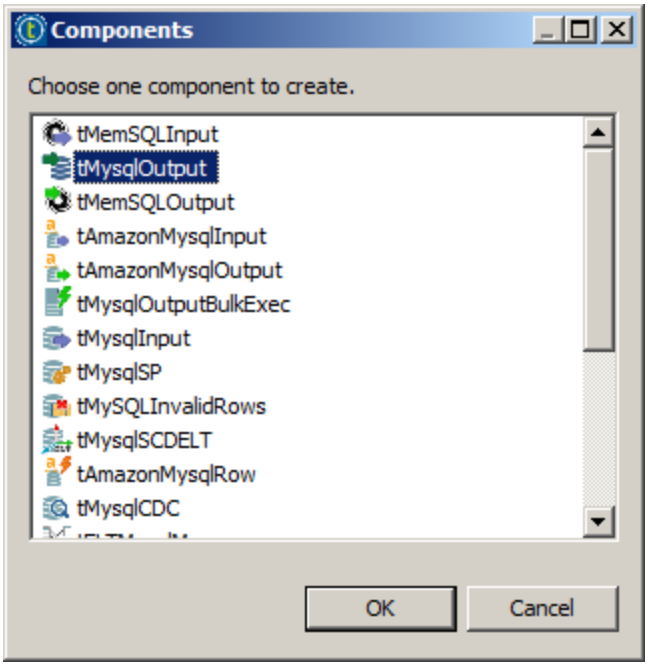
This trigger specifies that the connected subJob only executes when the component runs without error. In other words, the Job will not populate the table unless the table is successfully created first.

3. ADD A COMPONENT TO WRITE TO THE DATABASE

Drag the **LocalMySQL** connection from the **Metadata** onto the **Designer** to the right of **RetailCustomers**.

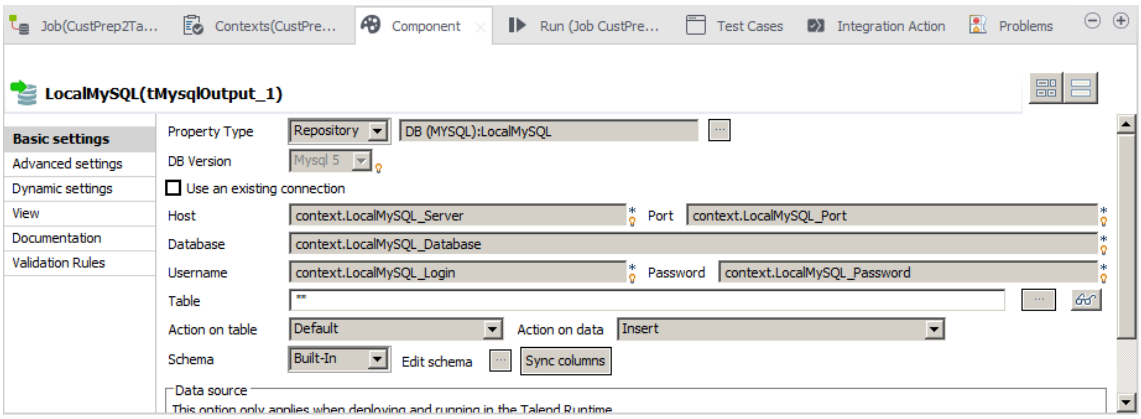


Select **tMysqlOutput** in the **Components** window and then click **OK**.



4. CONFIGURE THE COMPONENT

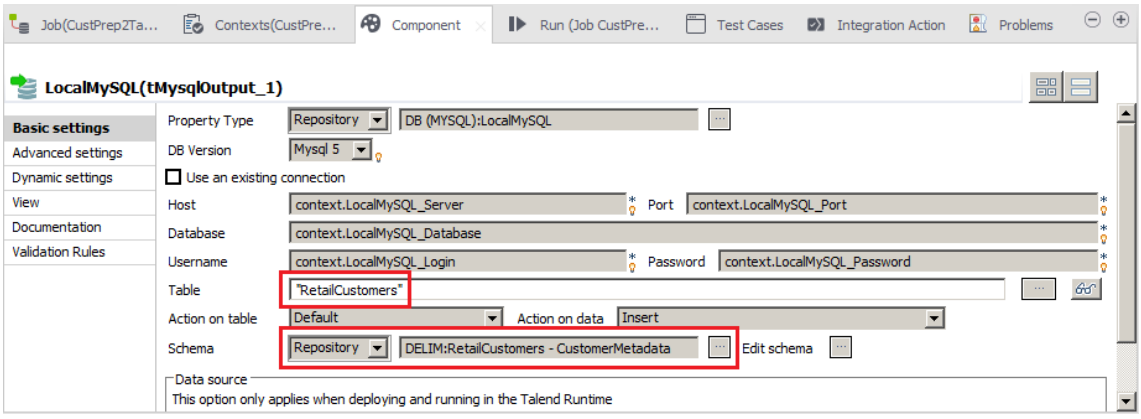
   Double-click the **LocalMySQL** component to open the **Component** view. Notice that the connection information is already set based on the metadata.



5. SPECIFY THE SCHEMA AND TABLE NAME

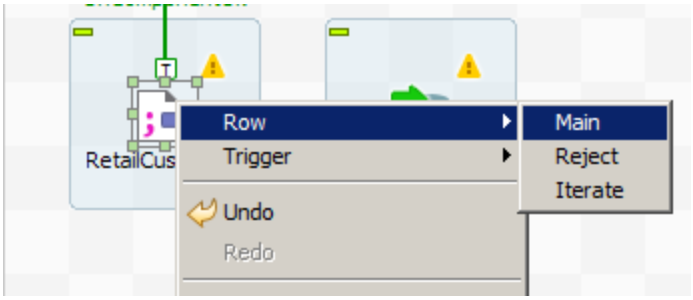
   As you did with the **tCreateTable_1** component, specify *Repository* for the schema and use the button marked with an ellipsis to choose the schema *RetailCustomers - CustomerMetadata*.

   Set the table name to *"RetailCustomers"*. This is the name of the table the **tCreateTable** component creates, so this component writes data to that new table.



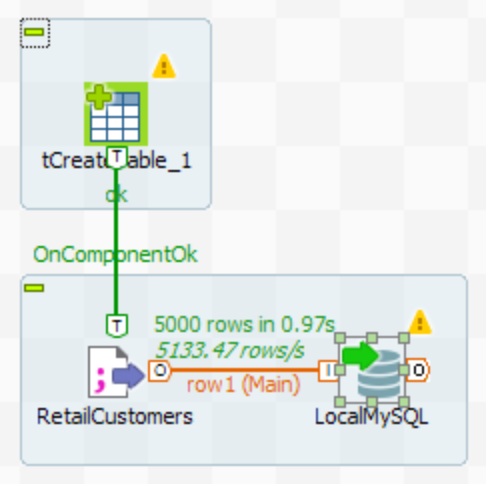
6. LINK THE INPUT COMPONENT TO THE DATABASE CONNECTION

   Right-click **RetailCustomers** and select **Row > Main**, then click the **LocalMySQL** component to connect them.



7. RUN THE JOB

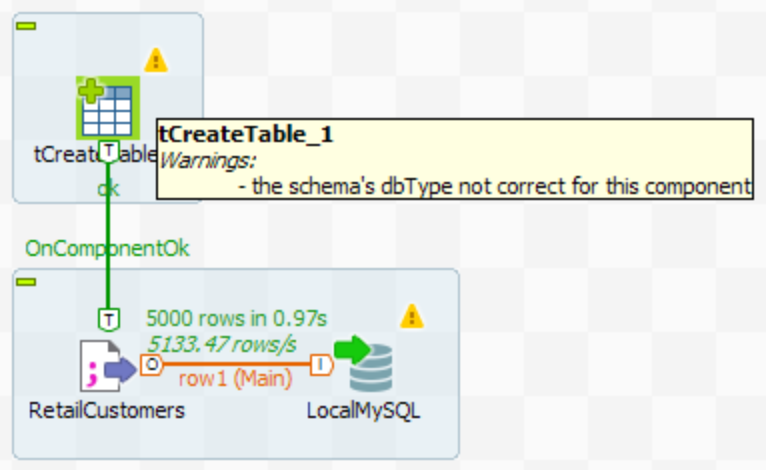
   Run the Job and examine the results.



Recall that the customer data file contains 5000 rows, which corresponds to the number of rows being moved to the database.

8. EXAMINE THE WARNINGS

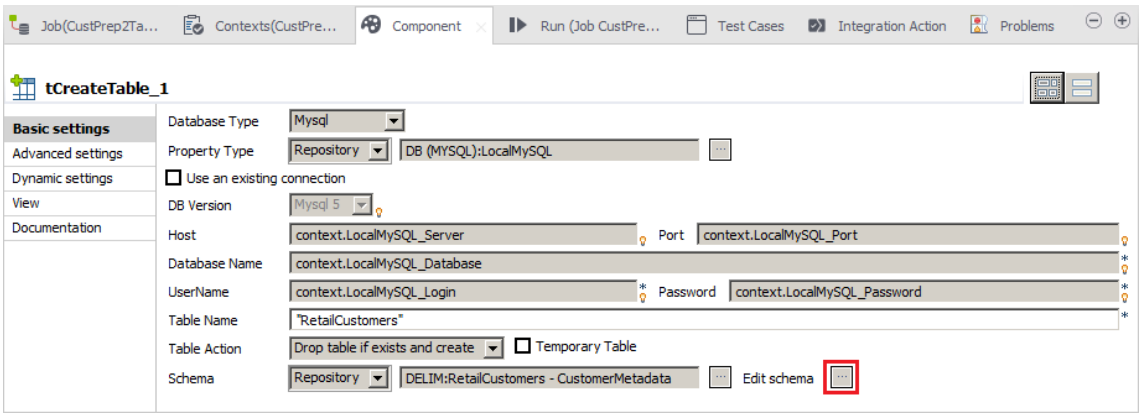
   Notice that a warning is being displayed for **tCreateTable_1** and **LocalMySQL**. Hover over the exclamation mark to read the warnings.



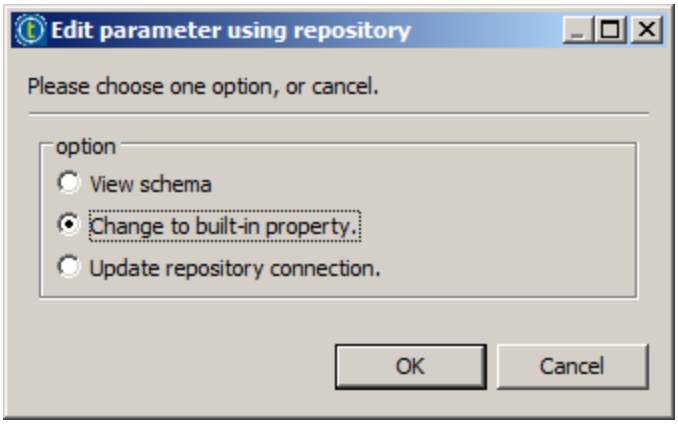
You will resolve this warning next.

9. FIX THE SCHEMA

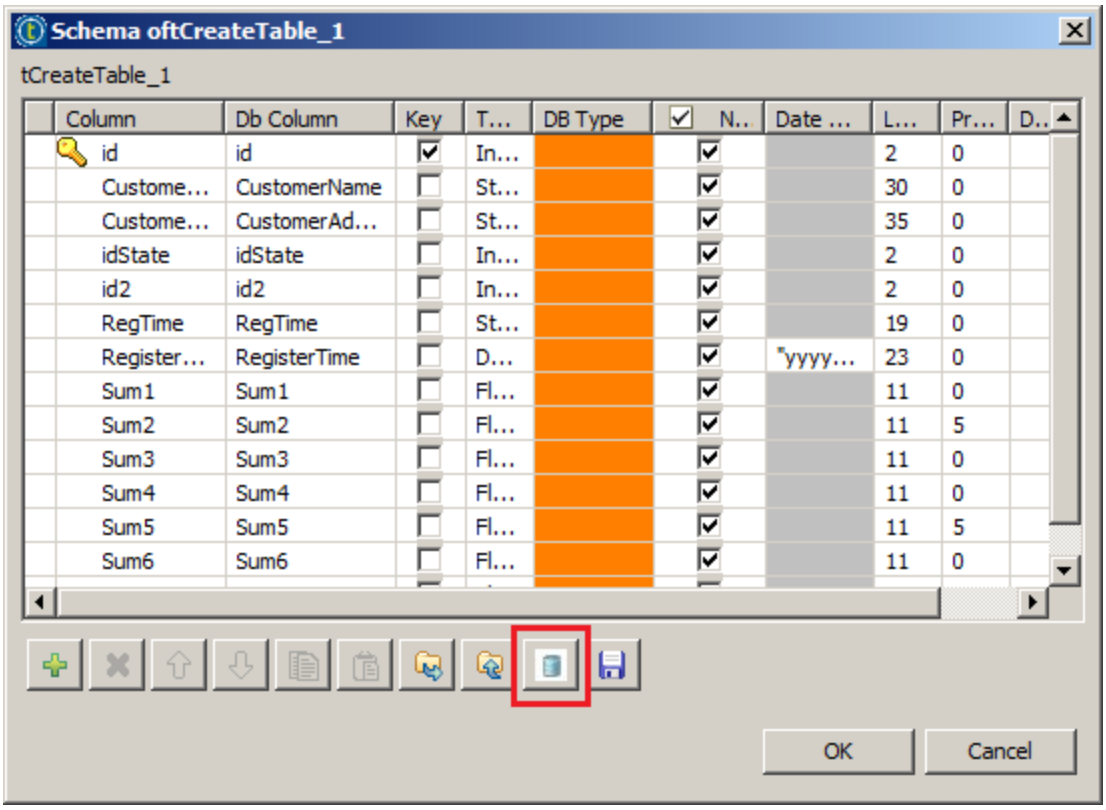
   To resolve the warning, click the **Edit Schema** button in the **Component** view for the **tCreateTable_1**component.



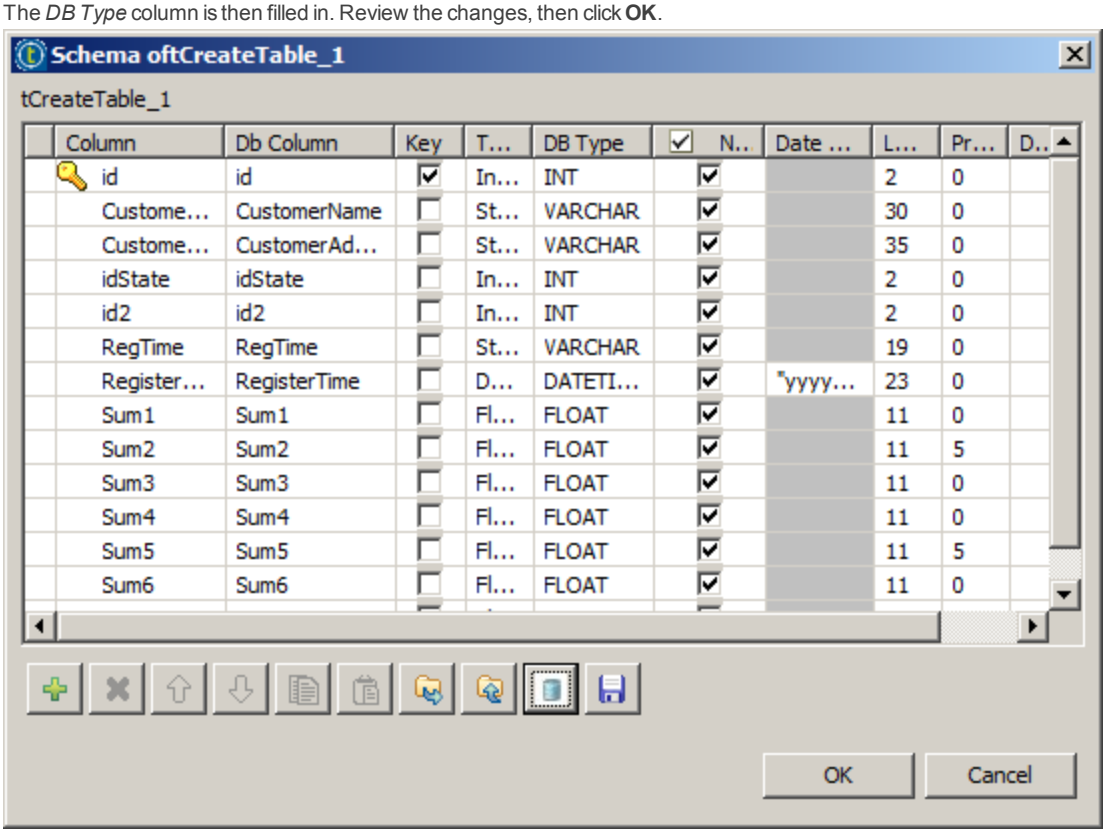
To avoid modifying the schema defined in the repository, select **Change to built-in property**. This way, you will be operating on a local copy of the schema rather than the shared version stored in the repository. Then, click **OK**.



The schema of the **tCreateTable_1** is displayed. Notice that the *DB Type* column is empty, and highlighted in orange. Click the **Reset DB Types** button ( ) below.


Schema ofCreateTable_1
tCreateTable_1
Column
Db Column
Key
T...
DB Type
N..
Date ...
L...
Pr...
D..
id
id
In...
2
0
Custome...
CustomerName
St...
30
0
Custome...
CustomerAd...
St...
35
0
idState
idState
In...
2
0
id2
id2
In...
2
0
RegTime
RegTime
St...
19
0
Register...
RegisterTime
D...
"yyyy...
23
0
Sum1
Sum1
Fl...
11
0
Sum2
Sum2
Fl...
11
5
Sum3
Sum3
Fl...
11
0
Sum4
Sum4
Fl...
11
0
Sum5
Sum5
Fl...
11
5
Sum6
Sum6
Fl...
11
0
OK
Cancel

The *DB Type* column is then filled in. Review the changes, then click **OK**.



Notice that the warnings have now disappeared for **tCreateTable_1**.

10. FIX THE REMAINING WARNING

    The warning for **LocalMySQL** is also related to the fact that the *DB Type* is not defined in the schema. Remove that warning by repeating the previous step you performed to remove the warning from the **tCreateTable_1** component.

Verify Results

You are going to check that the database table has been created and populated using a tool called MySQL Workbench.

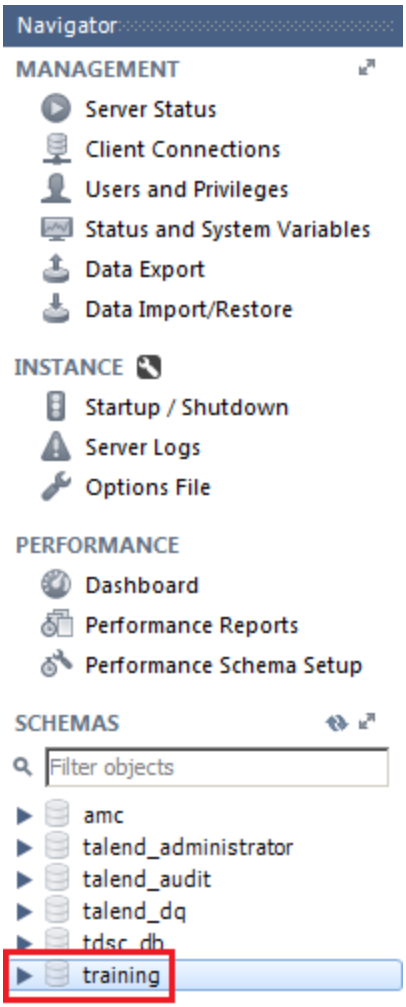
1. OPEN MYSQL WORKBENCH

   From the Windows **Start** menu, select **All Programs > MySQL > MySQL Workbench 6.3 CE** .
   The welcome window appears briefly, and then the **MySQL Workbench** window appears. Double-click **Local instance MySQL56** to open the connection.

   TIP:
   If prompted for authentication, enter *root* for both **User** and **Password**.

2. SELECT THE DATABASE

   Double-click **training** at the bottom of the **Navigator** on the left to specify the database to which you are connecting.

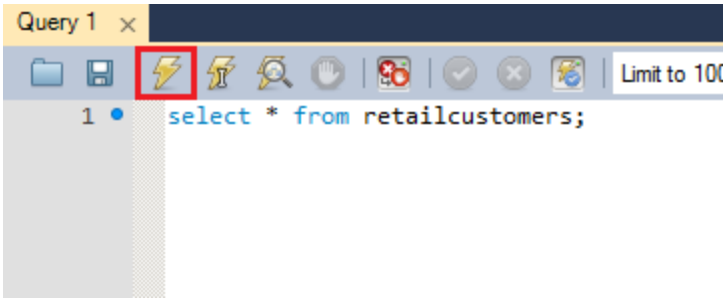


3. QUERY THE DATABASE

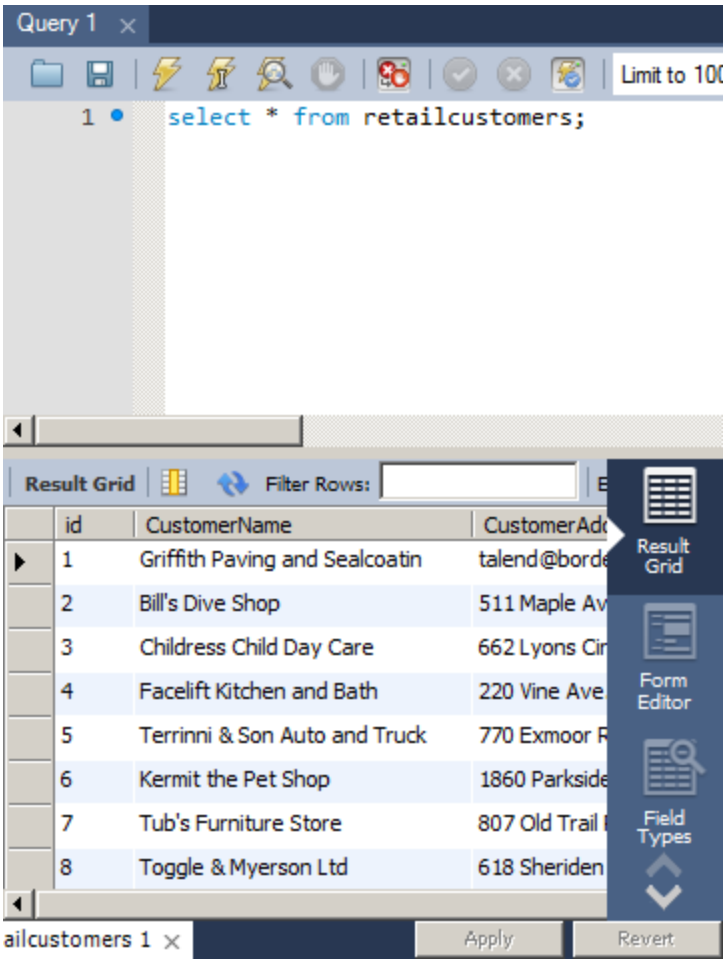
Enter the following command into the **Query 1** window. Be sure to include the trailing semi-colon:

```
select * from retailcustomers;
```

Then click the **Execute** button ( ) to execute the query.



Review the contents of the **Result Grid** that appears below.



This shows the rows from the table, demonstrating the data that your Job loaded. If your query does not return any results, double-check your work and make any necessary corrections.

As you will need **MySQL Workbench** for the next exercise, please leave it running.

Next

Now that you have the customer database table populated, you can populate the product database table.

Creating a Product Table

Overview

The next step is to create and populate the product table. This procedure is very similar to the **CustPrep2Table** Job so you will be duplicating it and making a few changes.

When you are creating a Job that is very similar to an existing Job, you can save time by duplicating it rather than starting from scratch.

Duplicate Job

1. DUPLICATE THE JOB

   Ensuring that any changes to the **CustPrep2Table** Job are saved, then duplicate it. Name the new Job *ProductPrep2Table*.

   NOTE:

   To save a Job, press **Ctrl+S** or click the Save button ( ) in the main tool bar at the top-left corner of the main window.

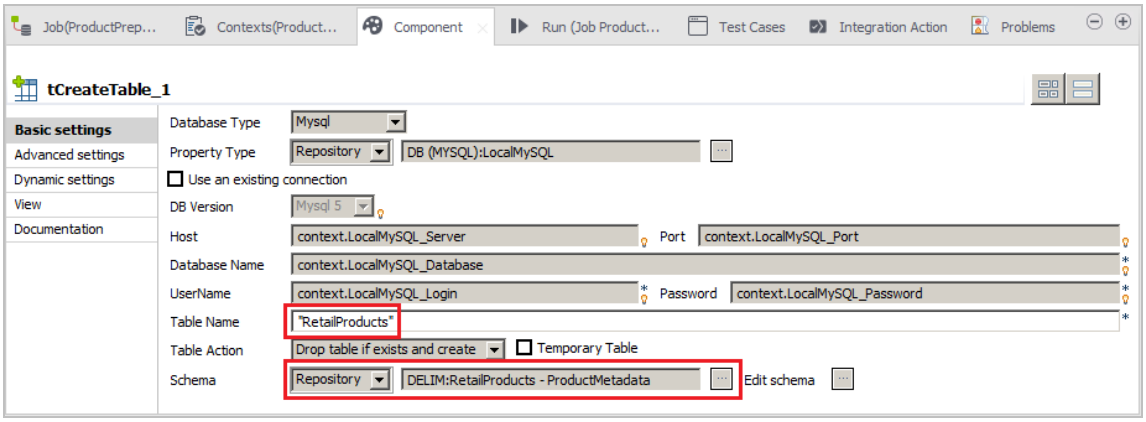
2. OPEN THE JOB

   Open the new Job. Close any other open Jobs to avoid confusion.

Create and Populate the Product Table

1. CHANGE THE SCHEMA FOR THE tCreateTable COMPONENT

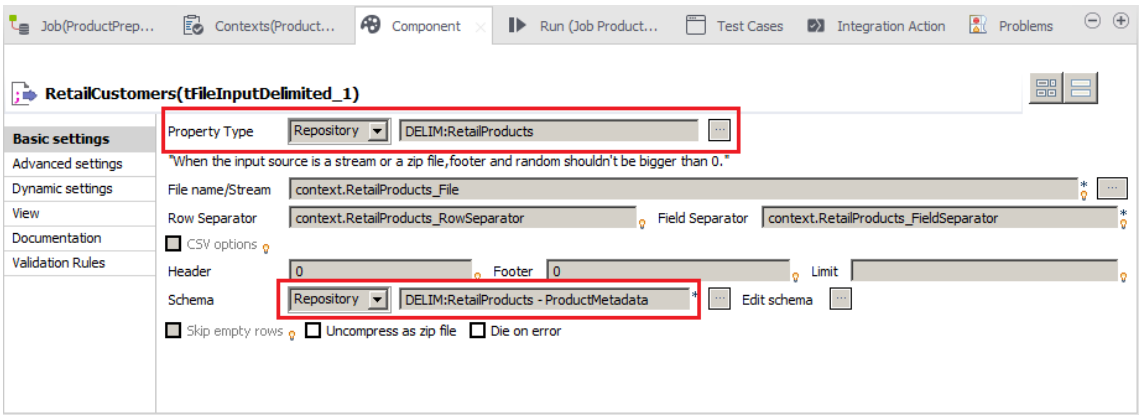
   Double-click the **tCreateTable_1** component. In the **Component** view, change the **Schema** type to *Repository* and select the *RetailProducts - ProductMetadata* schema.
   Then change the **Table Name** to *"RetailProducts"*.



2. CHANGE THE INPUT METADATA

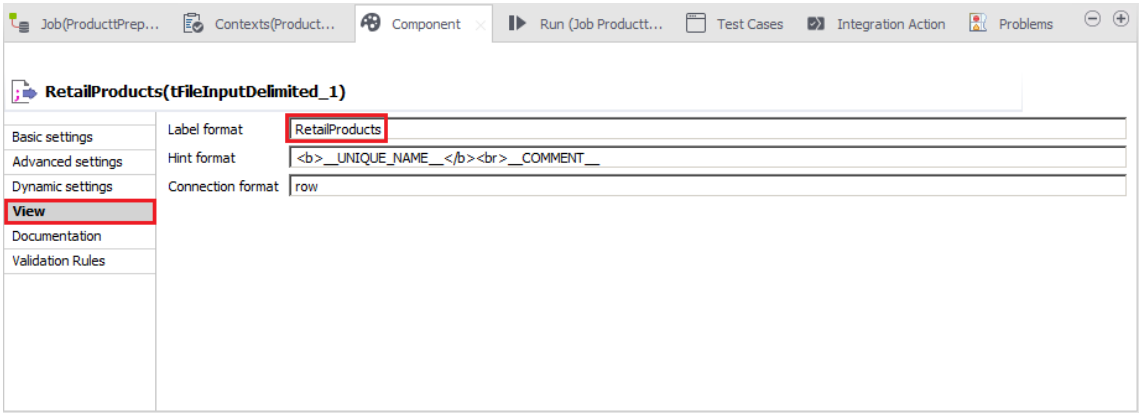
   Double-click the **RetailCustomers** component to open the **Component** view.

   Use the **[...]** button to change the **Property Type** value to *RetailProducts*. Notice that doing so also adjusts the value for **Schema**, which is desirable.



3. CHANGE THE COMPONENT NAME

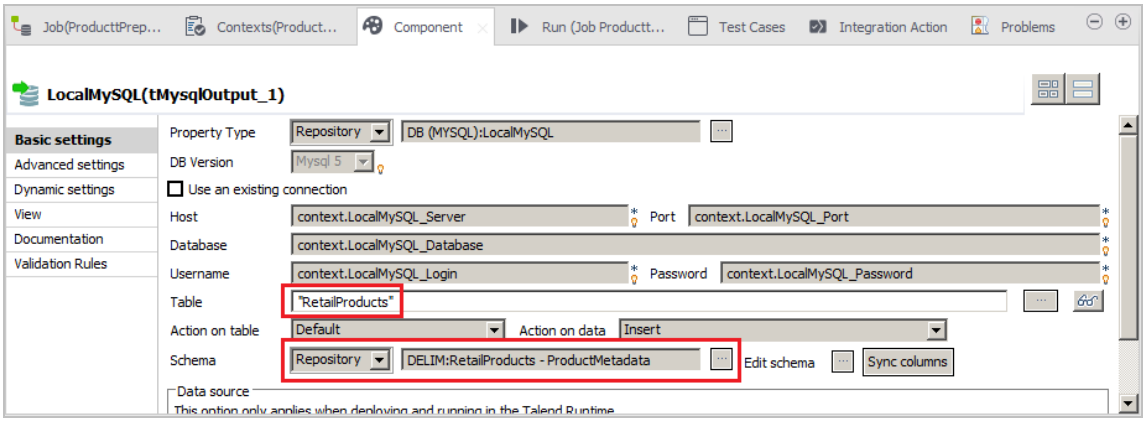
   Go to the **View** tab and change the name of the component to *RetailProducts*.



4. CONFIGURE THE OUTPUT COMPONENT

   Double-click the **LocalMySQL** component to open the **Component** view. Notice that the schema is now set to *Built-in*, because you propagated the schema changes from the **RetailProducts** component.

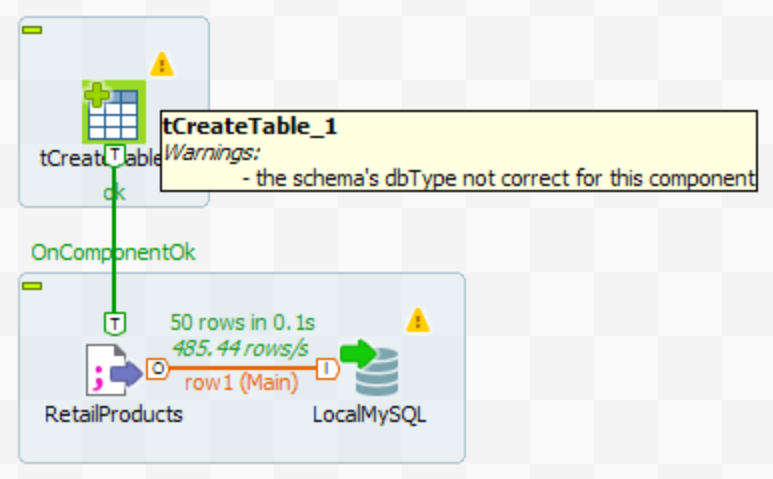
   Change the schema to *Repository*, specify the *RetailProducts - ProductMetadata* schema, and then change the table name to *"RetailProducts"*.



   Now this component will write to a table named *RetailProducts* with the schema taken from the Repository.

5. RUN THE JOB AND CHECK WARNINGS

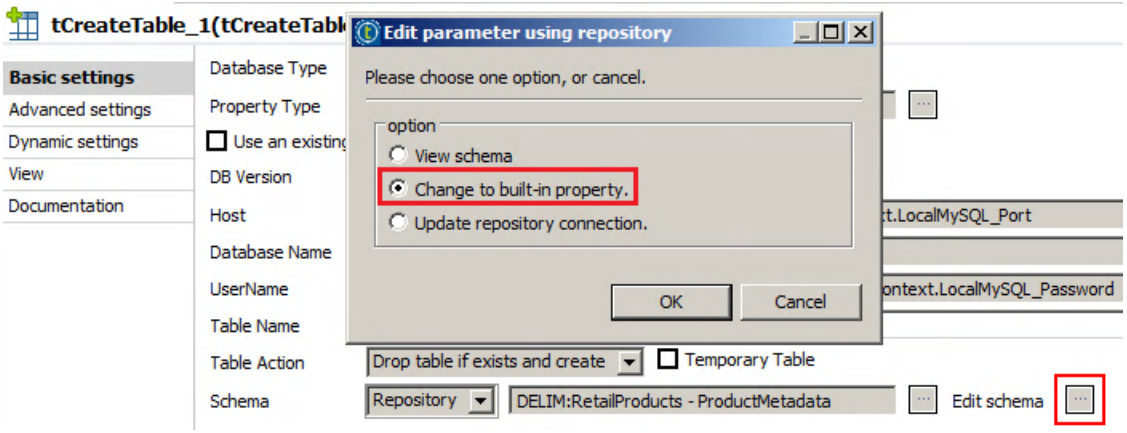
Run the Job. The product file contains 50 rows, but notice that a warning appears on **tCreateTable_1** and **LocalMySQL**. Hover over the exclamation mark to read the warnings.



Once again, the database type is not set for the columns and it needs to be. You will resolve this next.

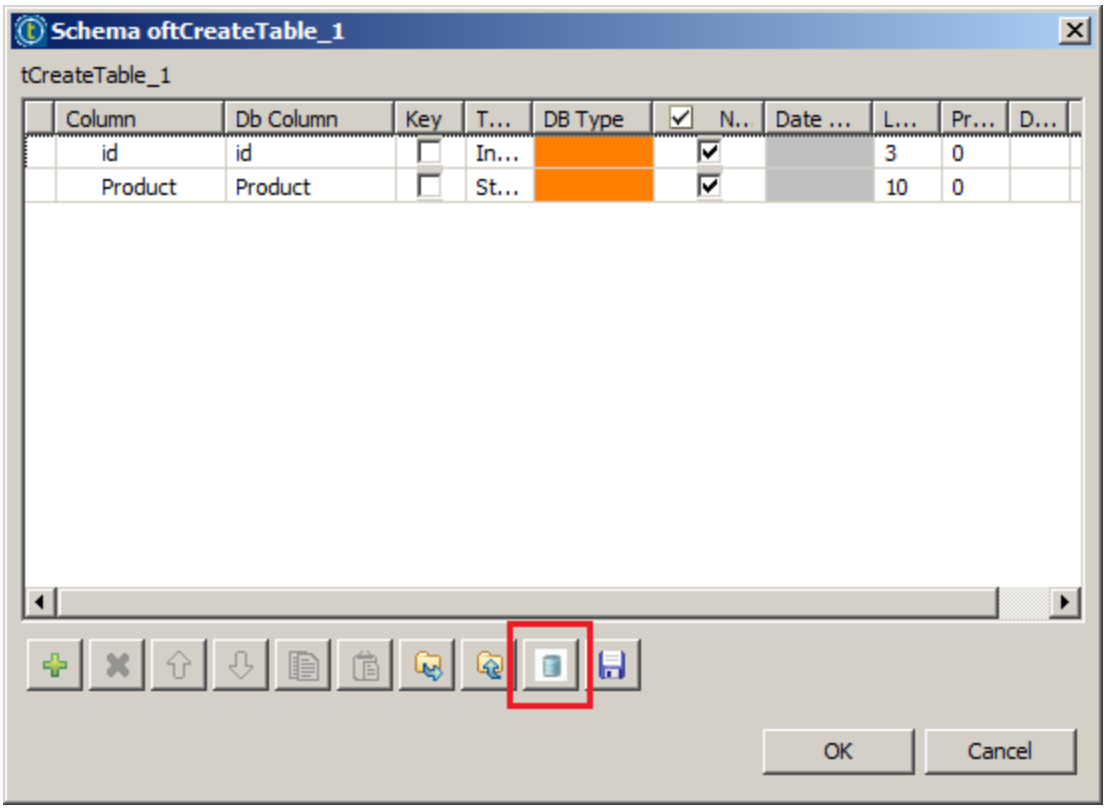
6. FIX THE SCHEMA

   To resolve the warning, click the **Edit Schema** button in the **Component** view for the **tCreateTable_1**component. Select **Change to built-in property** and then click **OK**.

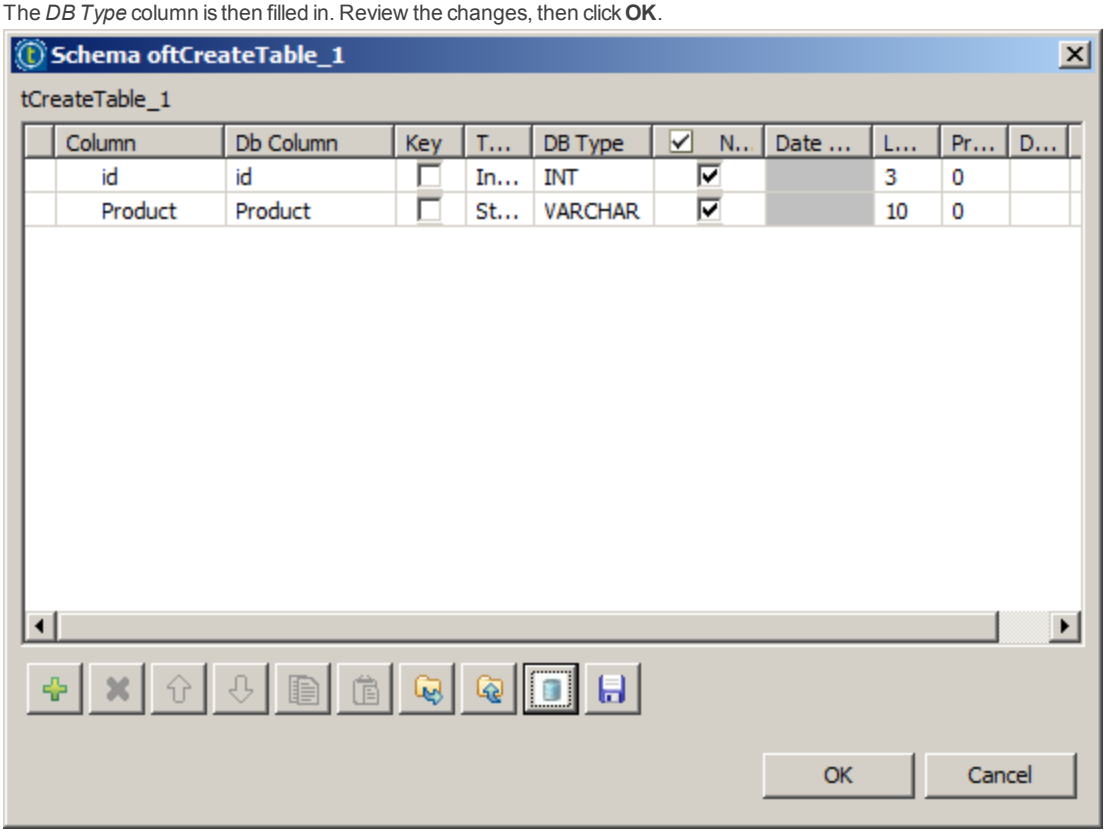


7. RESET DATABASE TYPES

The schema of the **tCreateTable_1** is displayed. Notice that the *DB Type* column is empty, and highlighted in orange. Click the **Reset DB Types** button ( ) below.



The *DB Type* column is then filled in. Review the changes, then click **OK**.



Notice that the warnings have now disappeared for **tCreateTable_1**.

8. REMOVE THE REMAINING WARNING

   The **LocalMySQL** warning is also stems from a similar root cause. In order to remove the warning on this component, repeat the previous step you just performed to remove the warning from the **tCreateTable_1** component.

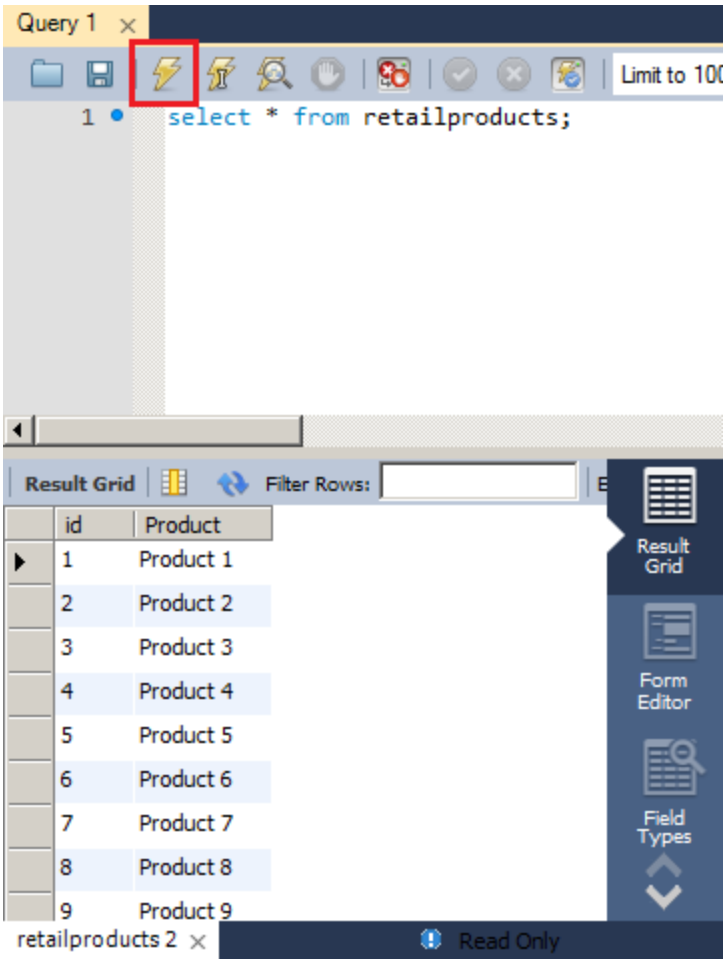
Verify Results

1. CHECK THAT THE TABLE IS POPULATED

   In **MySQL Workbench**, replace the existing text in the **Query** window with the following command:

   ```
   select * from retailproducts;
   ```

   Then, click the **Execute** button ( ) to verify the result.



If the query does not return results, double-check your work and make any necessary corrections.

Next

You now have database tables containing both product and customer data. Now you can set up the shop tables.

Setting Up a Sales Table

Overview

In this training scenario, you are collecting periodic sales information from individual shops to be collated and stored in a master table. First, the temporary sales data you created earlier needs to be combined into a single staging table.

Create the Table

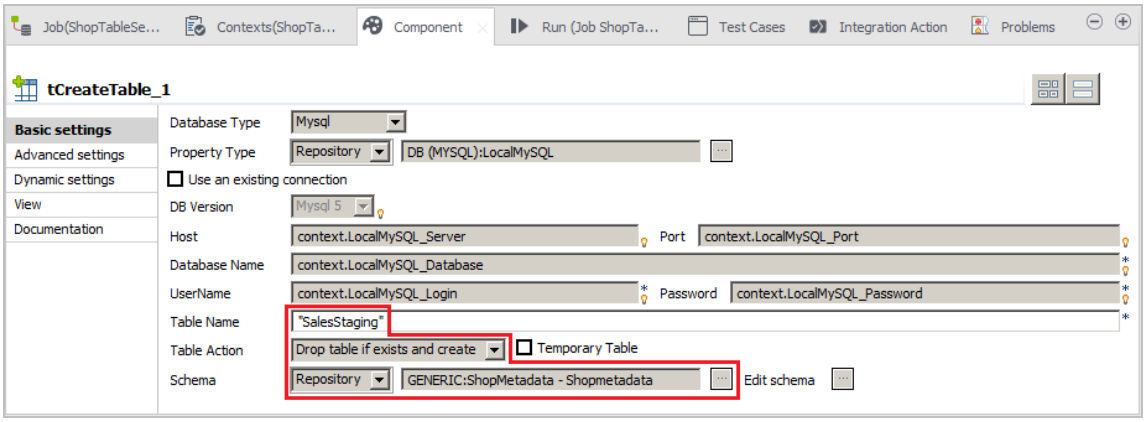
1. DUPLICATE A SIMILAR JOB

   Ensure that any changes to the **ProductPrep2Table** Job are saved, then duplicate it. Name the new Job *ShopTableSetup*. Open the new Job and then close all other Jobs to avoid confusion.

2. SPECIFY THE SCHEMA, NAME, AND TABLE ACTION

   Double-click the **tCreateTable_1** component to open the **Component** view. Set the **Schema** type to *Repository*, and select the generic schema *ShopMetadata*.

   Set the **Table Name** to *"SalesStaging"*, and specify *Drop table if exists and create* for the **Table Action**.



   Now this component creates a database table named **SalesStaging** using the schema specified by *ShopMetadata*.

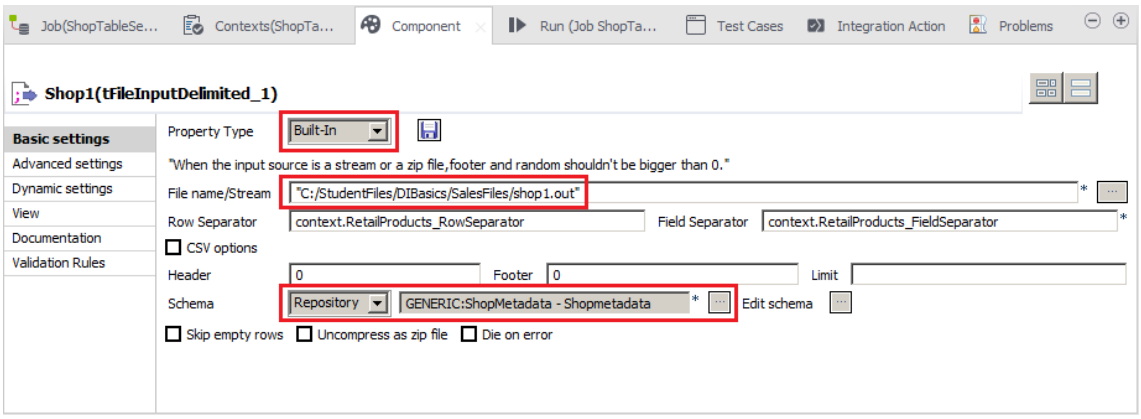
Modify the Components

Next, you will reconfigure the subjob to load the data for *Shop1* into the database.

1. CHANGE THE NAME AND SCHEMA OF THE INPUT COMPONENT

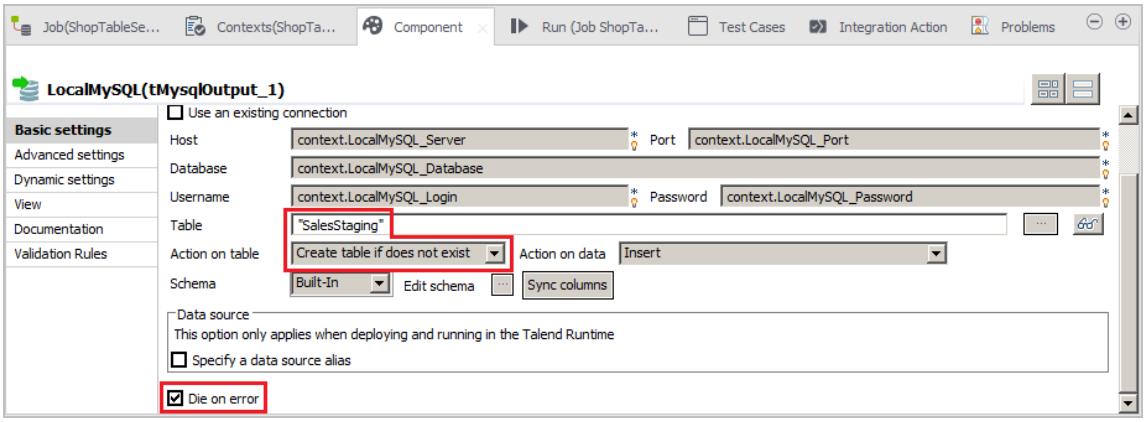
   Change the name of the **RetailProducts** component to *Shop1*, then configure it to use the *ShopMetadata* schema. Click **Yes** when prompted to propagate the changes.

   Set **Property Type** to *Built-In* and then update the **File name** to *"C:/StudentFiles/DIBasics/SalesFiles/shop1.out"*.



2. CHANGE THE TABLE NAME IN THE OUTPUT COMPONENT

   Double-click the component **LocalMySQL** and change the **Table** name to *"SalesStaging"*. Select the **Die on error** check box and then specify *Create Table if it does not exist* for **Action on table**.



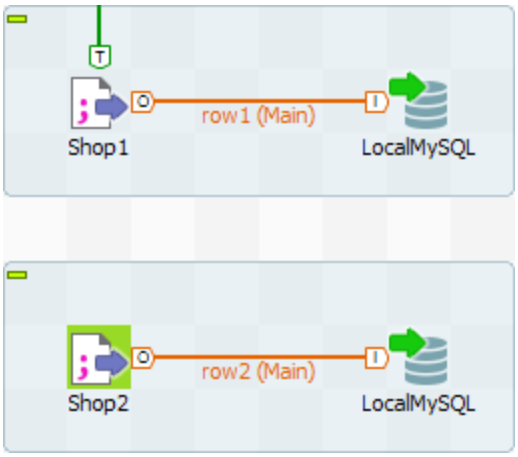
   Notice that because the schema was propagated from the **Shop1** component earlier, there is no need to change the schema here.

Duplicate the SubJob

You now need to configure subJobs to handle data for the other two shops.

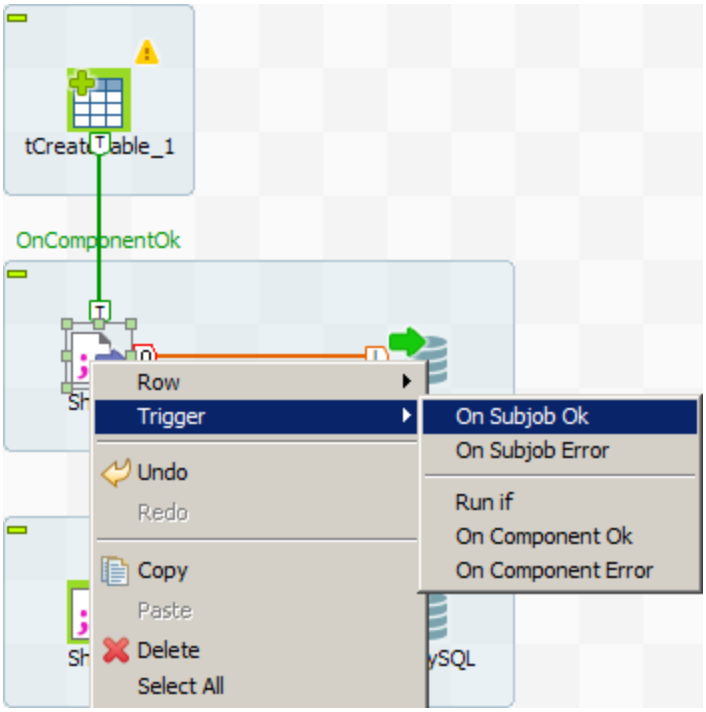
1. DUPLICATE THE SUBJOB

   In the **Designer**, copy and paste the subJob.



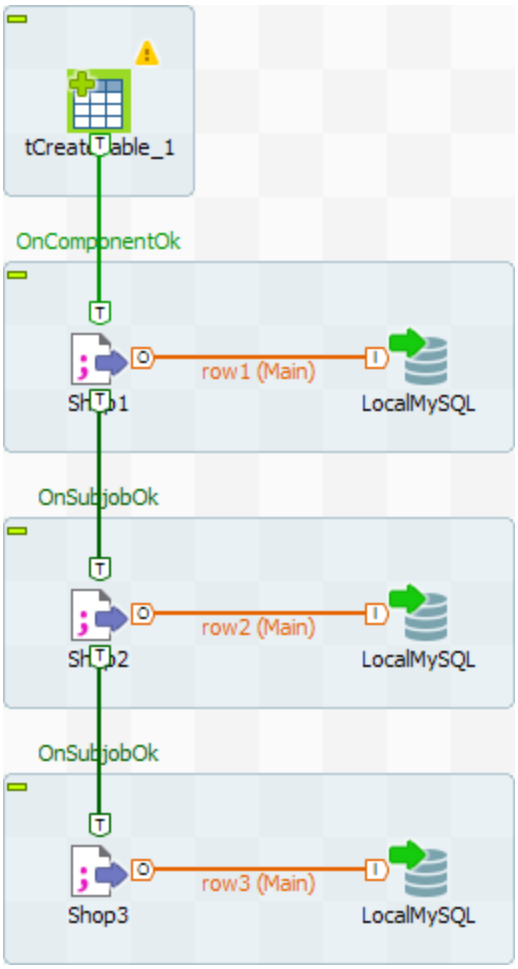
2. RECONFIGURE THE INPUT COMPONENT AND CONNECT THE SUBJOBS

   Change the name of the **Shop1** component in the second subJob to *Shop2* and configure it to read the *shop2.out* file. Then connect the two subjobs with an **On Subjob Ok** trigger.



3. CREATE AND CONFIGURE A THIRD SUBJOB

   Create another copy of the subJob, this time for *Shop3*, and configure it as you did for **Shop2**. Now all three sales files will be loaded into a single database table.



4. RUN THE JOB

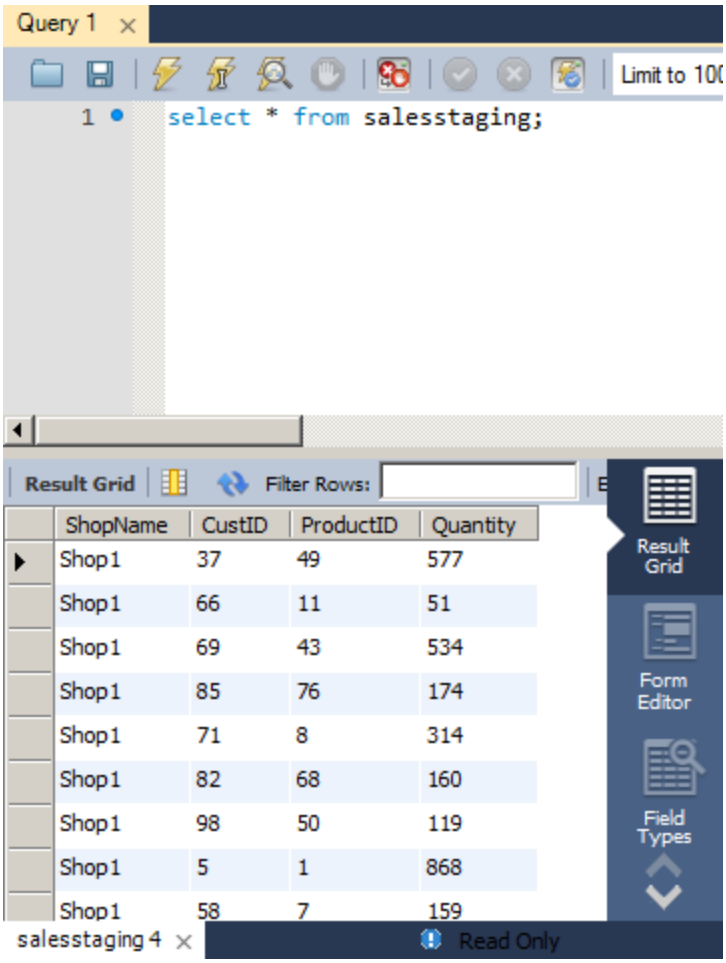
   Save and run the completed Job.

Verify Results

1. QUERY THE DATABASE

   Use **MySQL Workbench** to run the following query.

   ```
   select * from salesstaging;
   ```

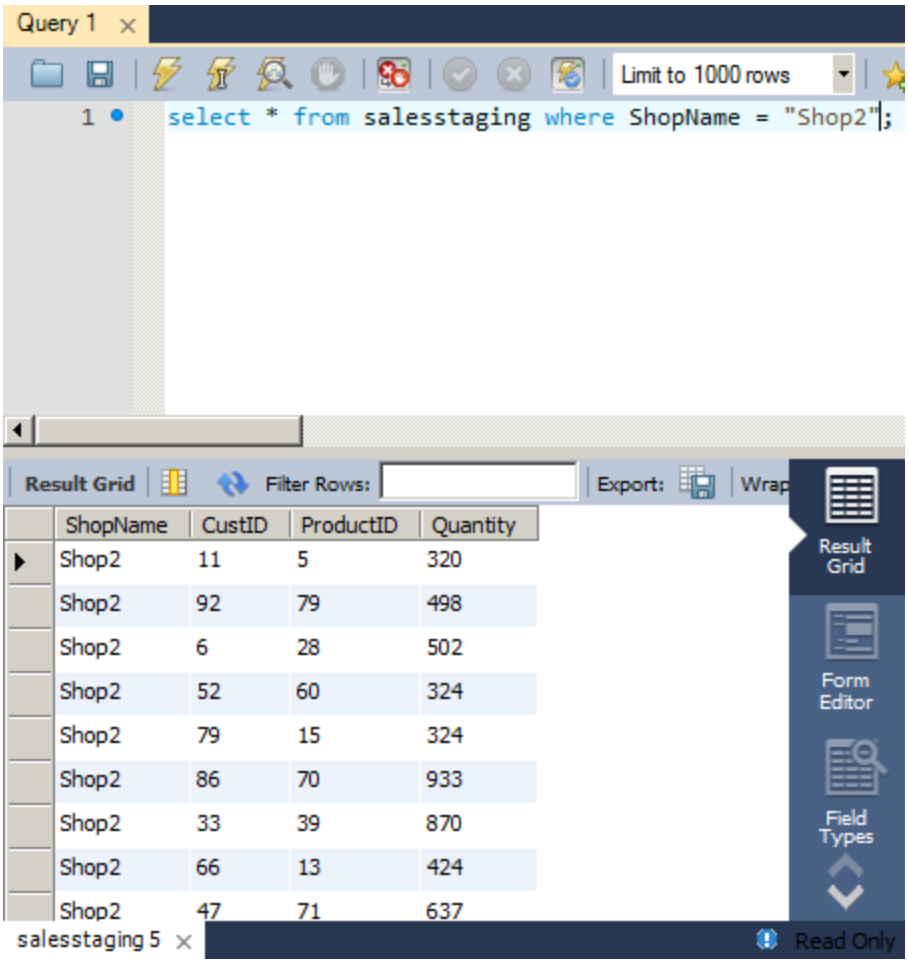


If the query does not return results, double-check your work and make any necessary corrections.
Notice that the results only show rows for *Shop1*. By default, the query only shows the first 1000 rows. Since they are sorted by the shop name, only *Shop1* rows are displayed.

2. FILTER THE QUERY

Now execute the following query to verify that the table holds rows for *Shop2*.

```
select * from salesstaging where ShopName = "Shop2";
```



3. CHECK FOR SHOP3

   Use a similar query to verify that the table holds data for *Shop3*. If any of the data is missing, double-check your work and make necessary corrections.

Next

You are finally ready to start building the main Job to join the data from the various sources.

Joining Data

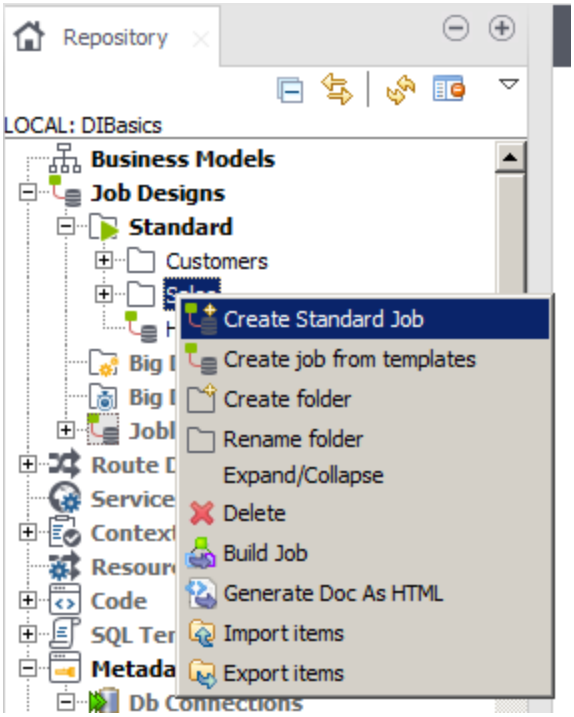
Overview

With all of the preparation done, you can now create the main Job to join the data and store it in a master table.

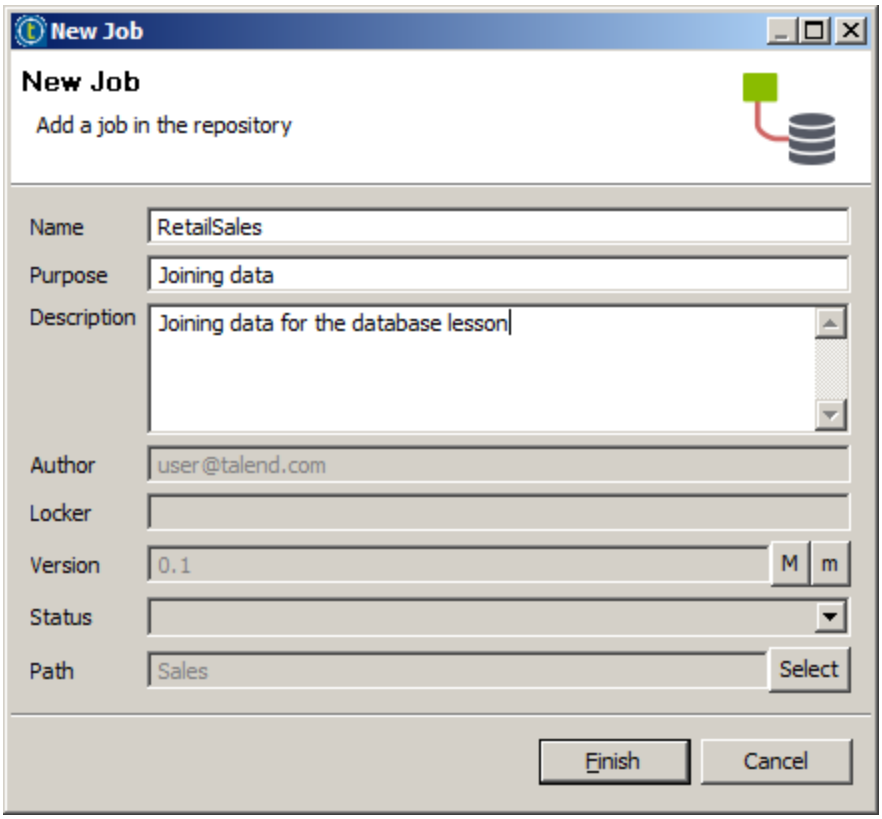
Read the Sales Data

1. CREATE A NEW JOB

   Create a new Job in the **Sales** folder.



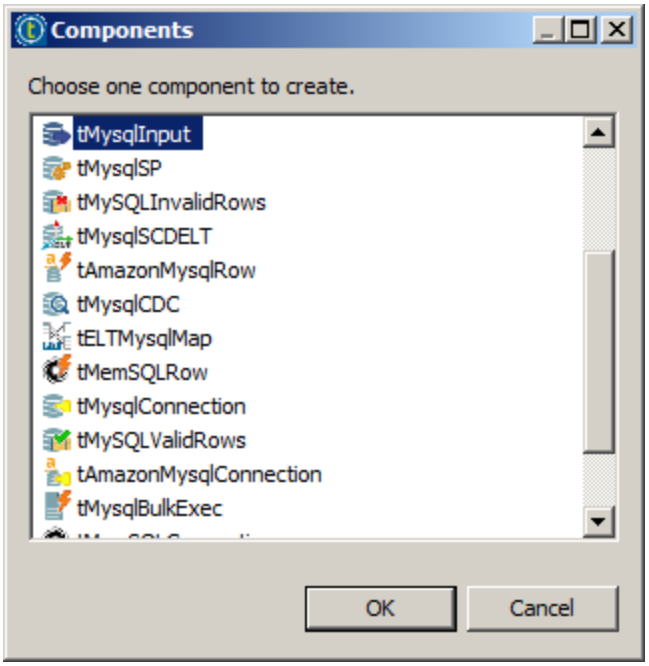
   Enter *RetailSales* for the **Name** and enter text of your choice for the **Purpose** and the **Description**.



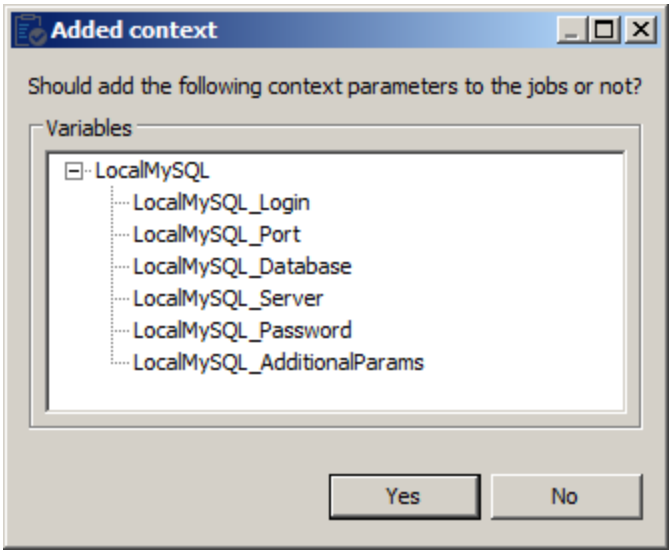
2. ADD A COMPONENT TO READ A DATABASE TABLE

   Drag the **LocalMySQL** database connection metadata from the **Repository** onto the **Designer**.

   When prompted, choose the **tMysqlInput** component and click **OK**.

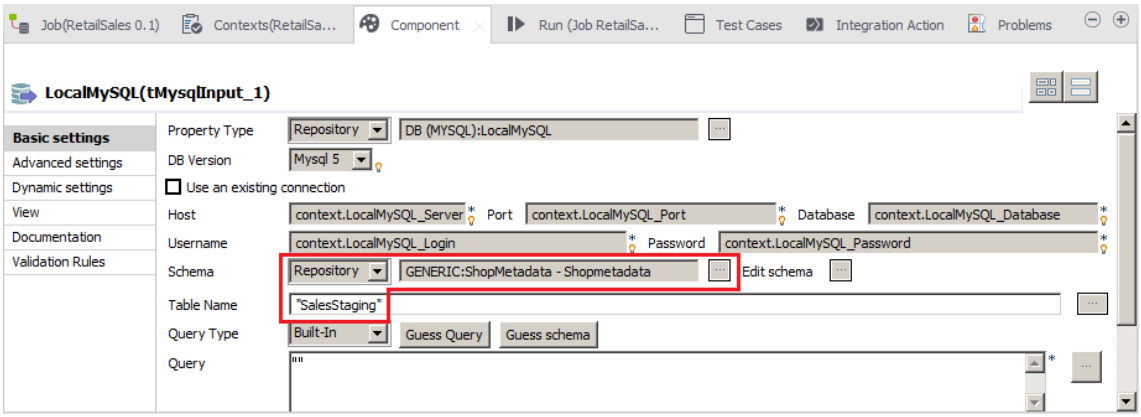


Click **Yes** to add corresponding context variables to your Job.

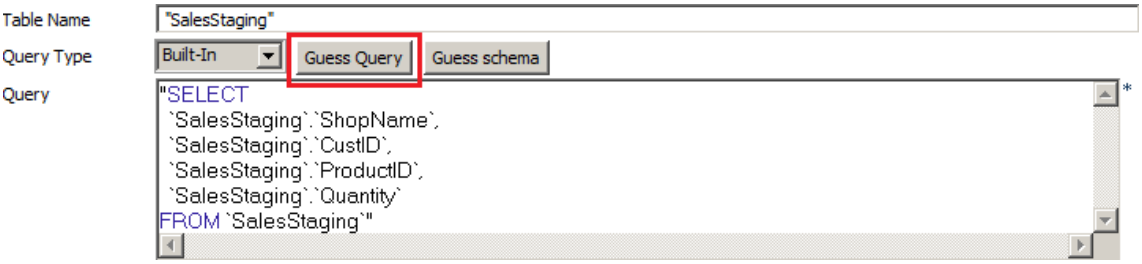


3. CONFIGURE THE COMPONENT

Configure the component to use the *ShopMetadata* repository schema and then set the **Table Name** to *"SalesStaging"*.



Click the **Guess Query** button and verify that the query looks like the figure below.



You can see that the default query retrieves all columns from the table, which is exactly what you want in this exercise.

4. LABEL THE COMPONENT

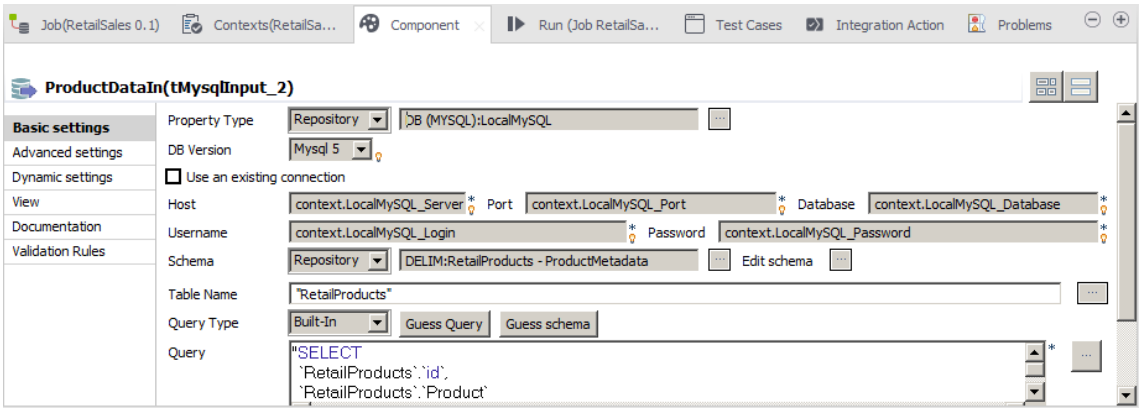
Click the **View** section from the **Component** tab and enter *SalesDataIn* for the **Label format** of the component.

Read Customer and Product Data

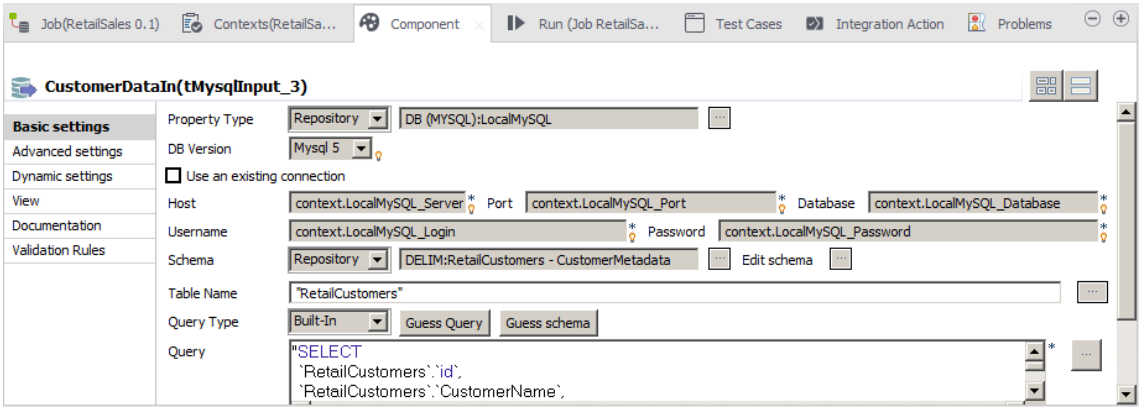
1. ADD INPUT COMPONENTS FOR THE PRODUCT AND CUSTOMER DATA

   Using a similar approach as in the previous section, add two more **tMysqlInput** components, one to read from the *RetailProducts* table and one from *RetailCustomers*.

   Be sure to use the correct schema and table name for each. The configuration for **ProductDataIn** should look like this.



   Your configuration for **CustomerDataIn** should look like this.

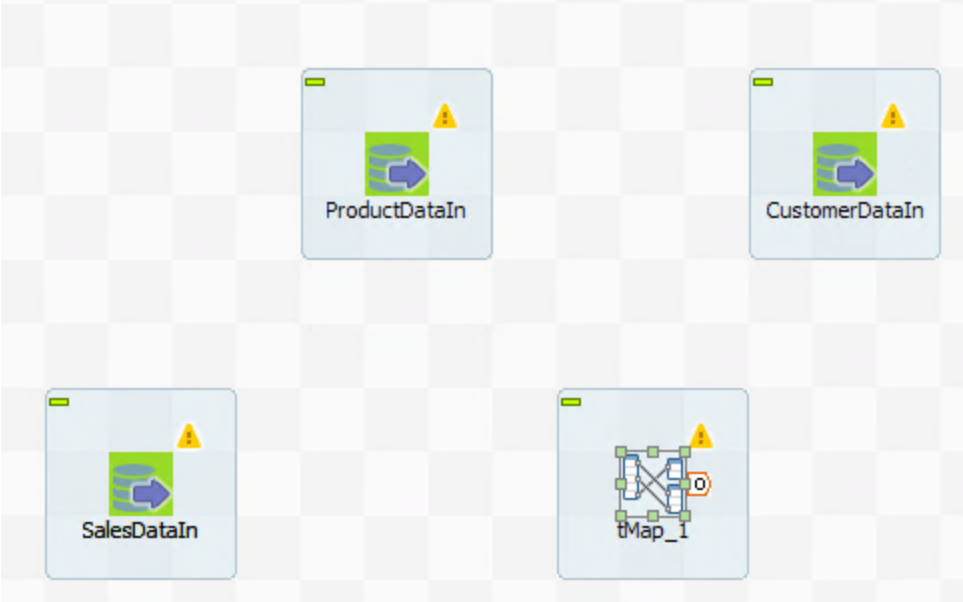


Join the Data

You will now use a **tMap** component to join the data.

1. ADD A tMap COMPONENT

   Add a **tMap** component to the Job.



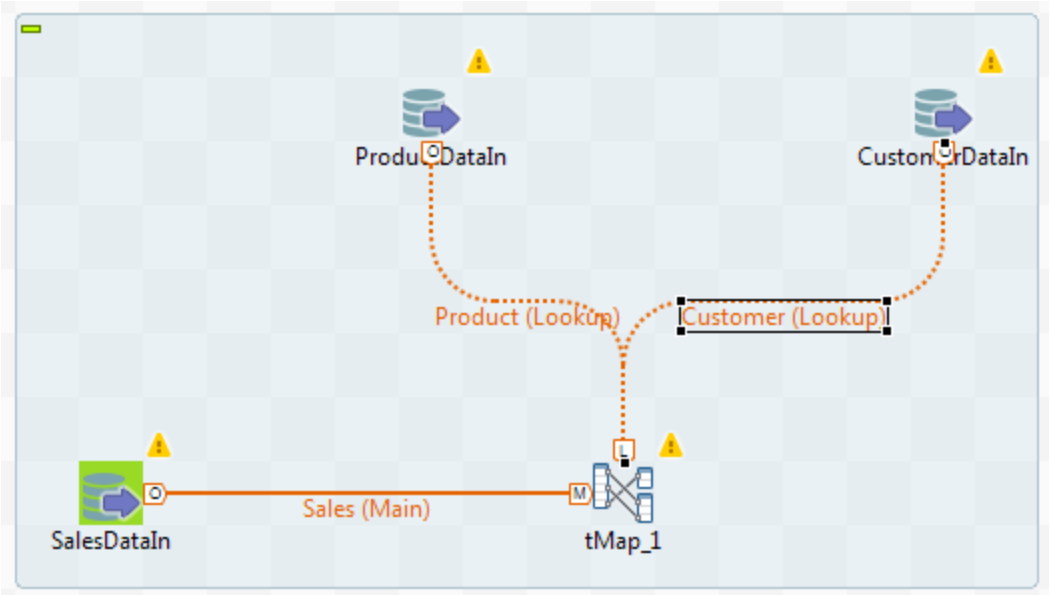
Notice that the input should be in the left side of **tMap** and look-ups on top of **tMap**.

2. LINK THE INPUT TO THE tMap COMPONENT

   Right-click **SalesDataIn** then select **Row > Main** and connect it to the **tMap** component. Change the name of this connection to *Sales* by clicking on the name twice, slowly, and then entering the new name.

3. LINK THE REMAINING COMPONENTS

   Similarly, connect **ProductDataIn** and **CustomerDataIn** components to the **tMap** component and name the connections *Product* and *Customer*, respectively. When done, your Job should resemble the figure below.

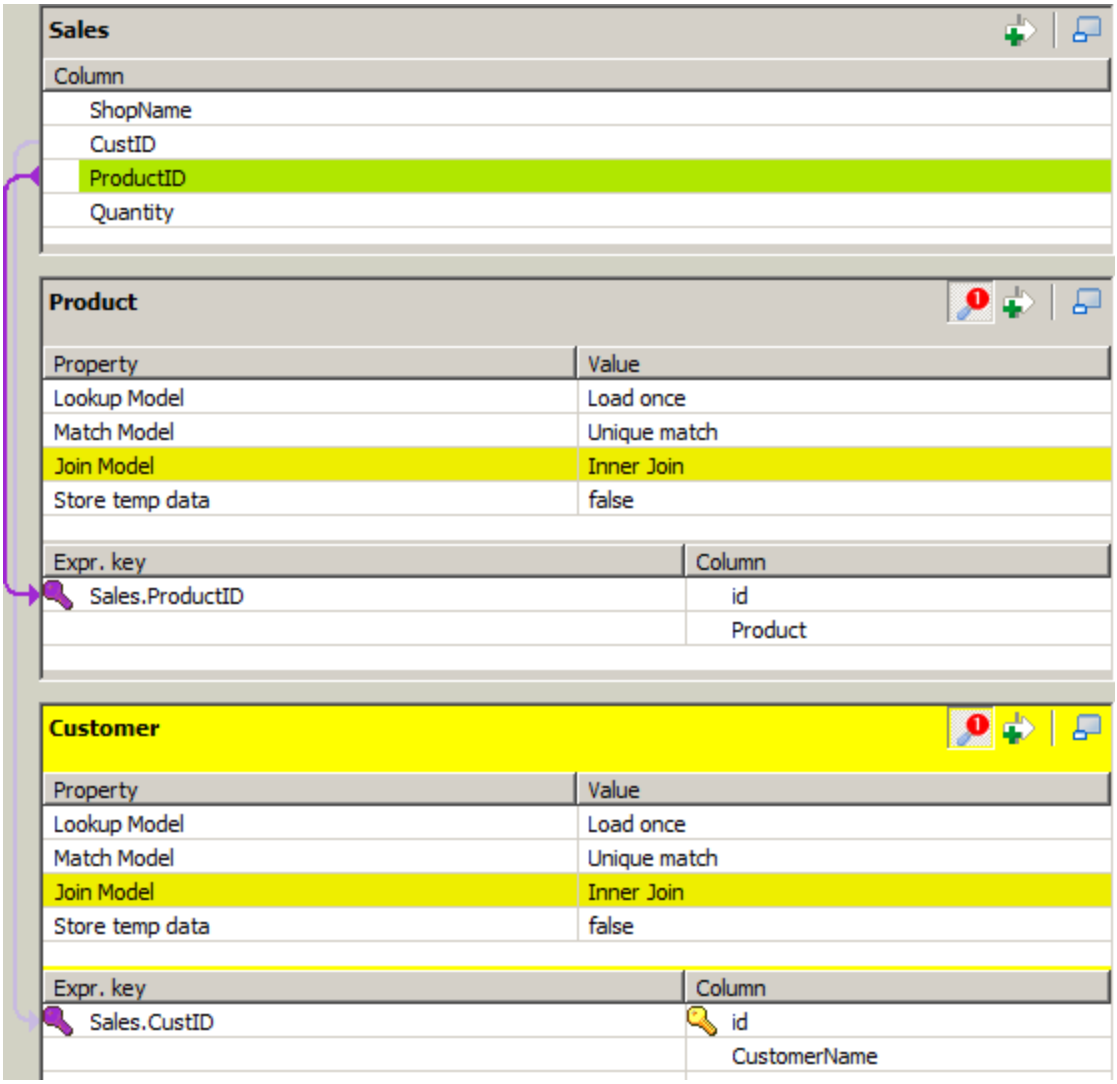


4. JOIN THE SALES INPUT TO THE CUSTOMER AND PRODUCT LOOK-UPS

   Double-click the **tMap** component.

Join the *CustID* and *ProductID* columns from the *Sales* table to the *id* columns of the *Customer* and *Product* tables, respectively, by dragging and dropping.

Set the **Join Model** for both joins to *Inner Join* using the **tMap settings** icon ( ).



5. ADD AN OUTPUT TABLE

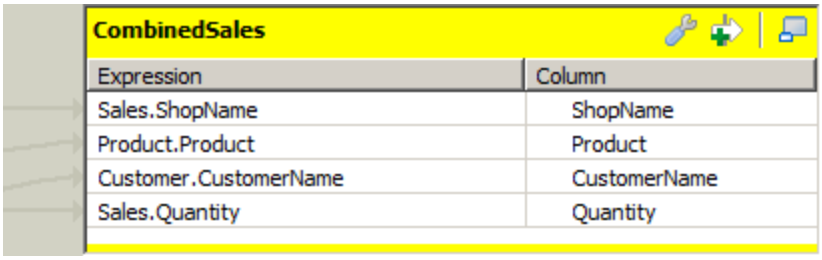
Add an output table named *CombinedSales* using the **Add output table** icon ( ) and map the following input columns into this new output table, by dragging and dropping:

- *ShopName* and *Quantity* from the *Sales* table
- *CustomerName* from the *Customer* table
- *Product* from the *Product* table

When done, click **Ok** to save the map configuration.

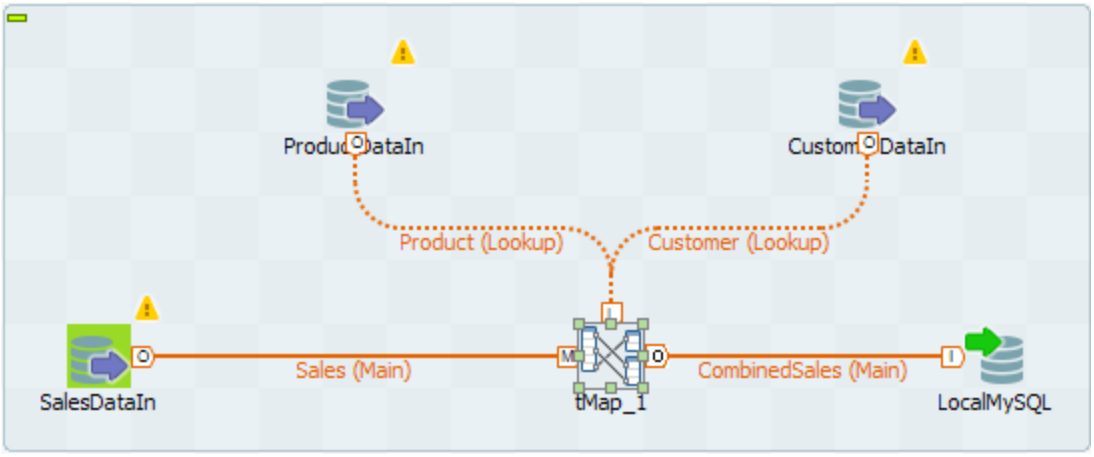
TIP:
As this is not a production system, the ordering of the columns is not particularly important. However, if you would like

to rearrange the columns, select the table and use the **Move up** ( ) and **Move down** ( ) buttons at the bottom of the mapping editor window.



6. ADD AN OUTPUT COMPONENT

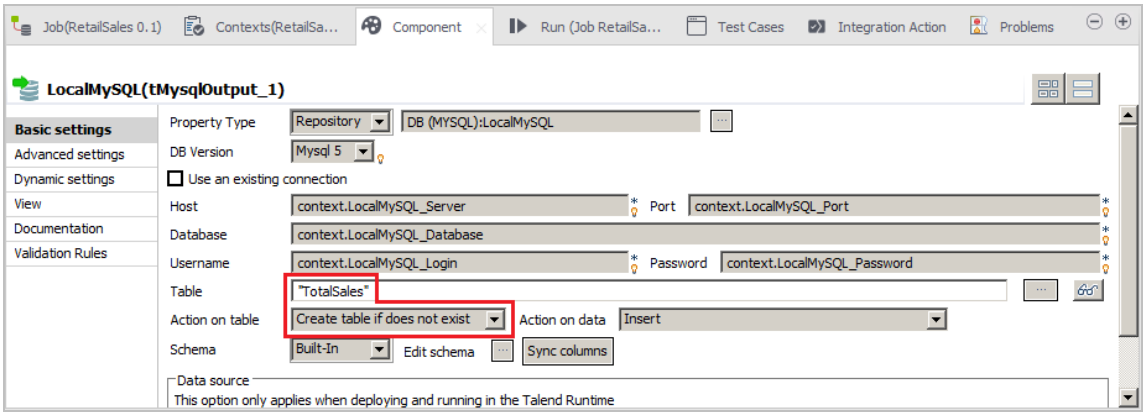
   Add a **tMysqlOutput** component using the **Db connection** metadata and then connect the **tMap** component to it with a **CombinedSales** row.



7. CONFIGURE THE OUTPUT COMPONENT

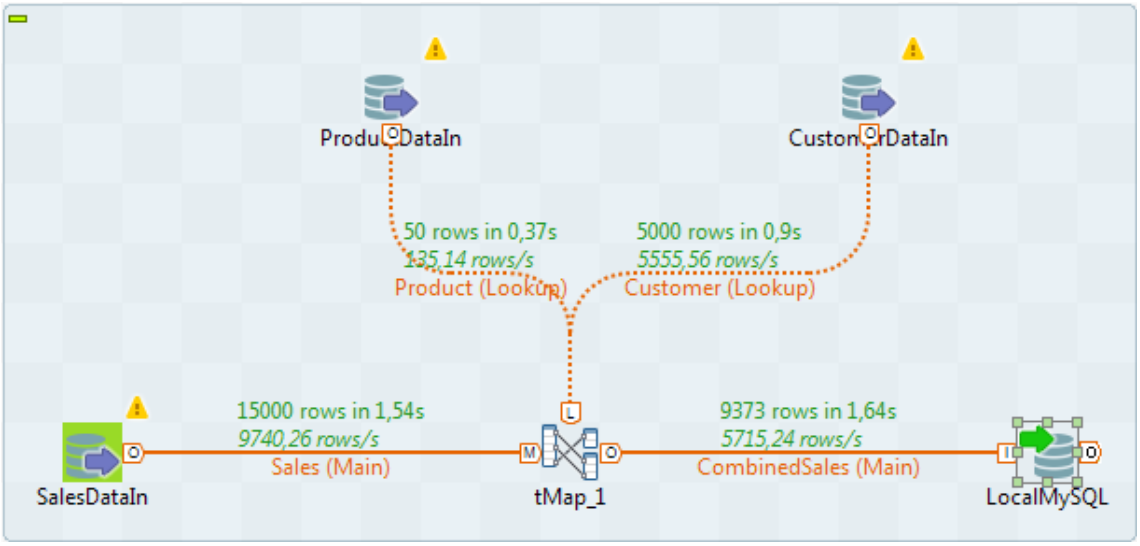
   Double-click the **LocalMySQL** component to open the **Component** view.

   Change the **Action on table** parameter to *Create table if does not exist*, and name the table *"TotalSales"*.



8. RUN THE JOB

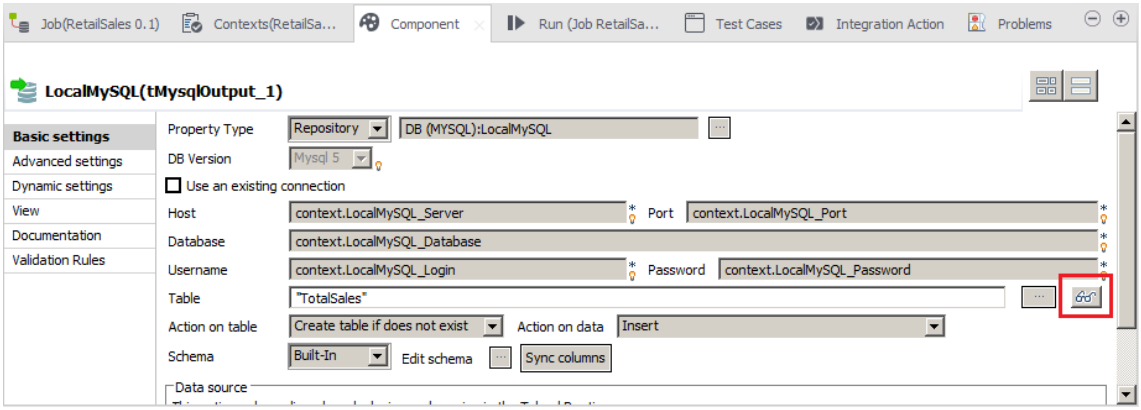
   Run the Job and then verify the resulting statistics.



15,000 rows originate from the sales data table, but notice that considerably fewer end up in the final table. Some of the rows are being rejected by the inner joins. Note also that the exact number of rows processed in the **CombinedSales** flow will vary due to the random nature of the input data.

9. INSPECT THE OUTPUT TABLE

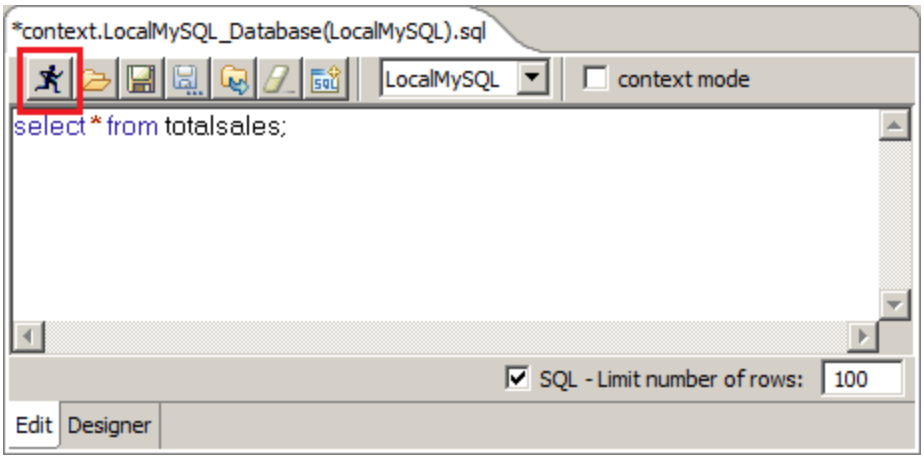
Double-click on the **LocalMySQL** component to open the **Component** view. Click on the **SQL Builder** button ( ).



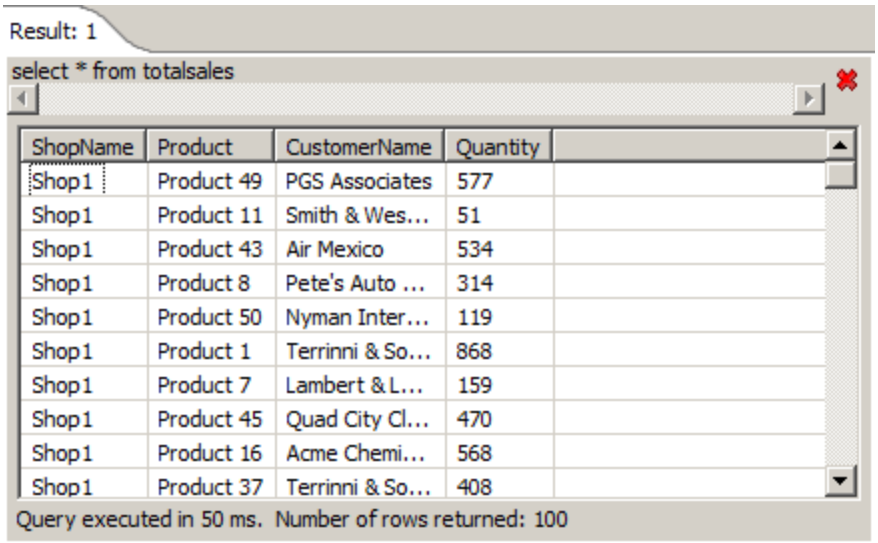
10. QUERY THE OUTPUT TABLE

Enter the following query into the **SQL Builder** window: `select * from totalsales;`

Run the query using the **Execute SQL** button ( ).



The data appears at the bottom left of the window.

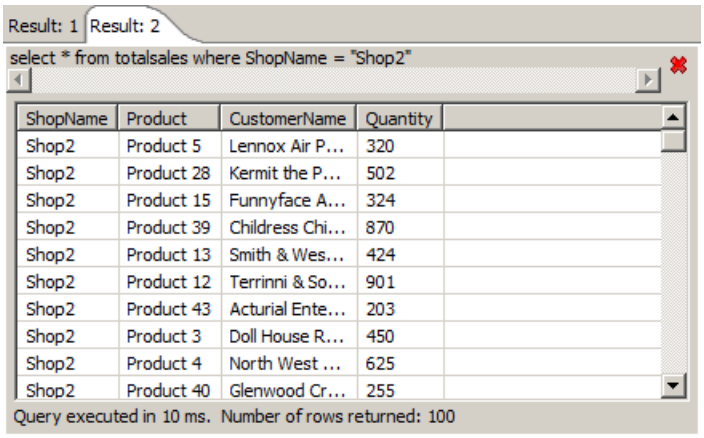


NOTE:
In this instance, you are using the **SQL Builder** functionality built into Talend Studio to check the database contents, but you could also use the **SQL Workbench** as you did in previous exercises.

11. UPDATE THE QUERY

Now update the query as follows: `select * from totalsales where ShopName="Shop2";`

Run the query to check that data related to Shop2 has also been inserted into the *TotalSales* table.



Next

The next step is to determine why some rows are being rejected, and to perform some calculations on sales totals.

Finalizing the Job

Overview

You have a couple of steps left to finalize the Job before running it. Those include determining why some rows are rejected, and performing basic calculations on the sales data figures.

Capture Rejects

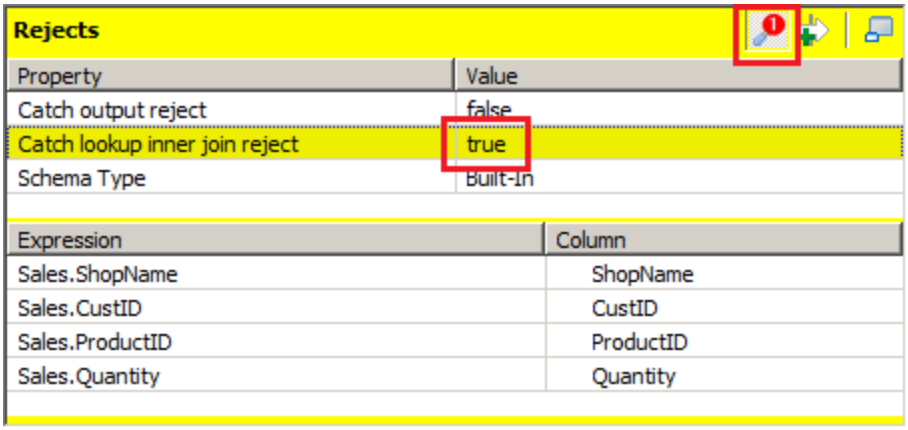
1. ADD A NEW OUTPUT TABLE TO CAPTURE REJECTS

   Continuing with the **RetailSales** Job to determine why some rows are being rejected, double-click the **tMap** component to configure it.

   Click the **Add output table** icon () to create a new output table named *Rejects*.

2. CONFIGURE THE NEW OUTPUT TABLE

   Click the **tMap settings** icon () to capture inner join rejects by setting **Value** to *true* for **Catch lookup inner join reject**. Then map all of the columns from the **Sales** input to this table and save the **tMap** configuration.



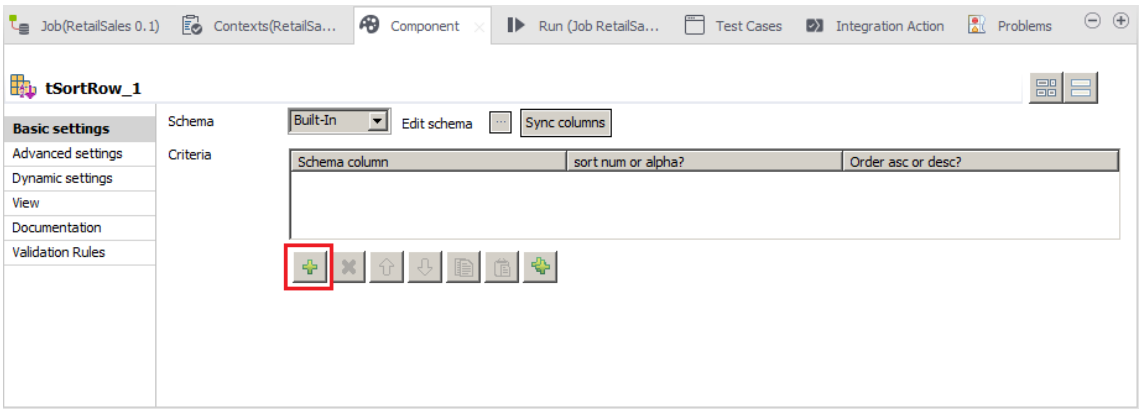
3. ADD A COMPONENT TO SORT THE REJECTS

   Add a **tSortRow** component to the Job. As the name implies, a **tSortRow** component sorts data rows based on column values.

   Connect it by right-clicking on the **tMap** component and selecting **Row > Rejects**.

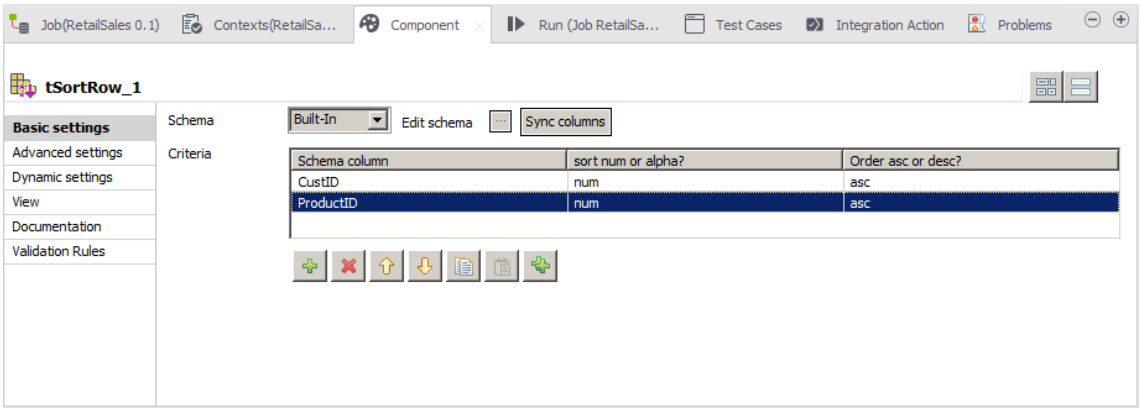
4. SPECIFY THE SORTING

   Double-click the **tSortRow_1** component to open the **Component** view and click the **Add** button ( at the bottom left) twice to add two columns to the table.



5. CHOOSE THE COLUMNS

   Since *CustID* and *ProductID* form the look-ups, it makes sense to sort and examine these elements. Select *CustID* for the first **Schema column** and *ProductID* for the second entry.

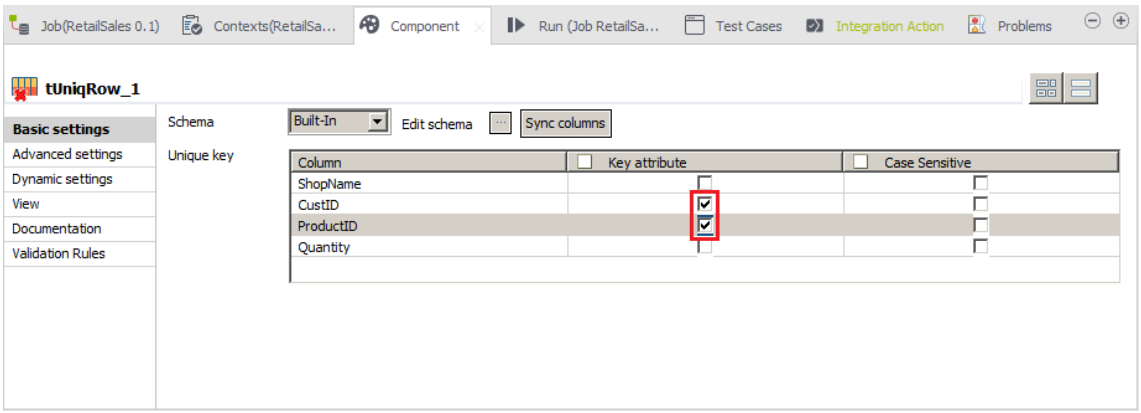


   With this configuration, the data sent to this component is sorted by the combination of column values of *CustID* and *ProductID*.

6. ADD A COMPONENT TO ELIMINATE DUPLICATE ROWS

   Add a **tUniqRow** component, connecting it to the **tSortRow** component with a **Main** row.

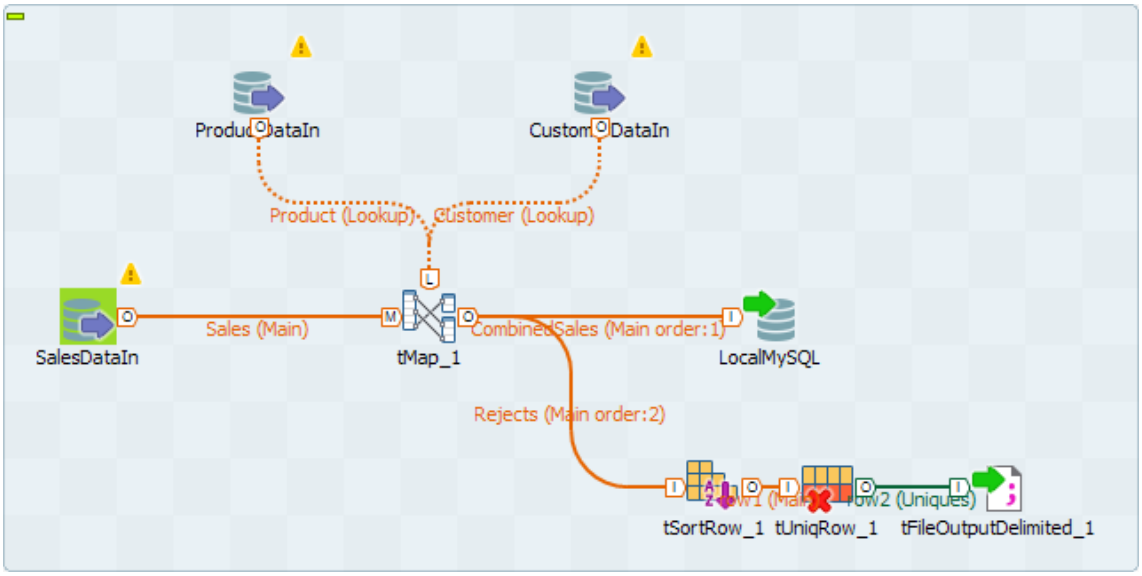
   In the **Component** view for **tUniqRow_1**, select the **Key attribute** check box for both **CustID** and **ProductID**.



Note that the rows entering this component are already sorted by customer ID and product ID by the **tSortRow** component, so the end result will be a single row for every unique combination of customer and product that is rejected by the inner join.

7. ADD AN OUTPUT COMPONENT

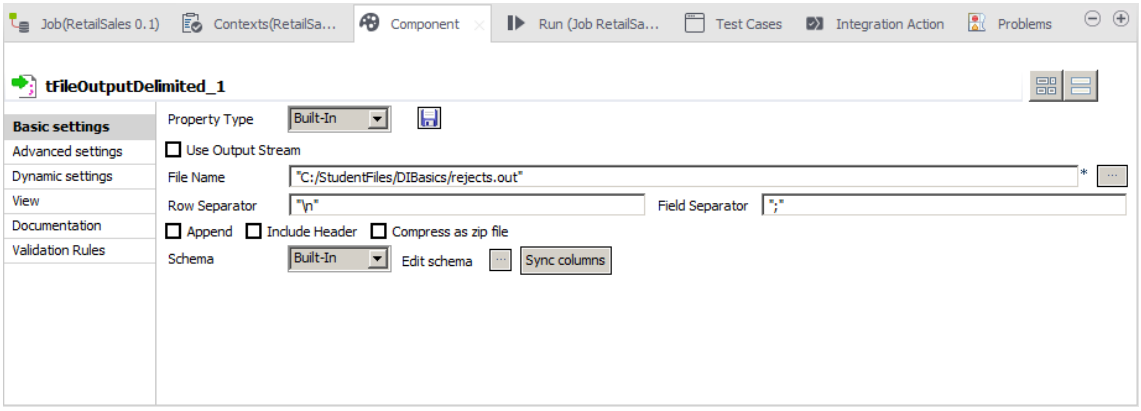
   Add a **tFileOutputDelimited** component to write out the sorted rejects. Connect it to the **tUniqRow** component with a **Uniques** row. The completed Job should look like this.



NOTE:
You can also capture the duplicate rows from a **tUniqRow** component, which can be useful in other situations.

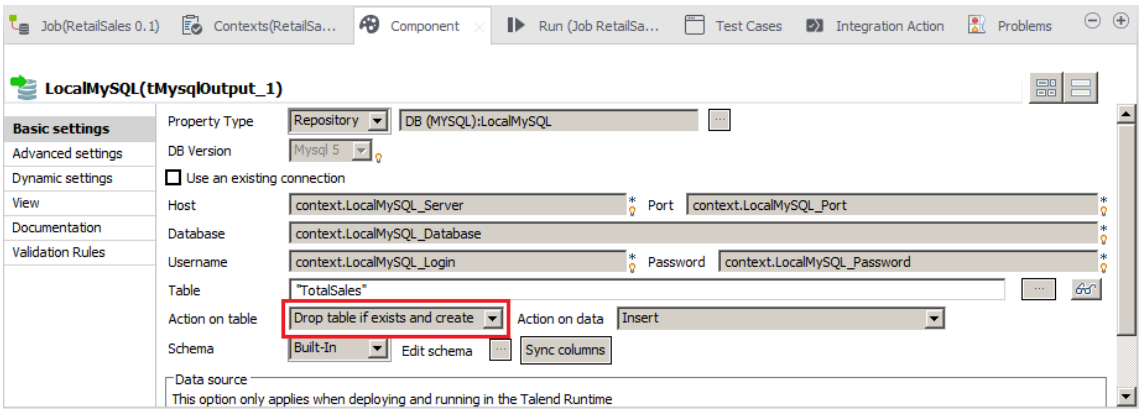
8. CONFIGURE THE OUTPUT COMPONENT

   Configure **tFileOutputDelimited_1** to store the results under *C:/StudentFiles/DIBasics/rejects.out*.



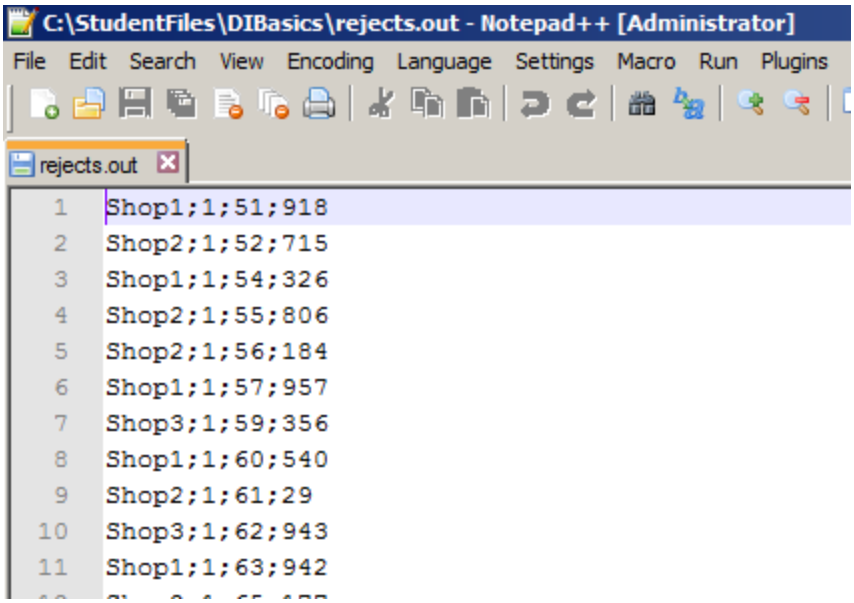
9. CHANGE THE ACTION ON THE DATABASE OUTPUT COMPONENT

   Open the **Component** view for the **LocalMySQL** component and change the **Action on Table** parameter to *Drop table if exists and create*. This prevents duplicate records, since you have already loaded the database table once before.



10. RUN THE JOB

    Run the Job and then examine the results in the *C:/StudentFiles/DIBasics/rejects.out* file.



    Notice that the values of the *ProductID* column in the output are all greater than 50. Recall that the product table only defines 50 products, but you configured the row generator to create product ID values up to 80. This explains the rejects. In a real project, you would go back and resolve this discrepancy, but for the purposes of this lesson, you can ignore the rejects.

Perform Calculations

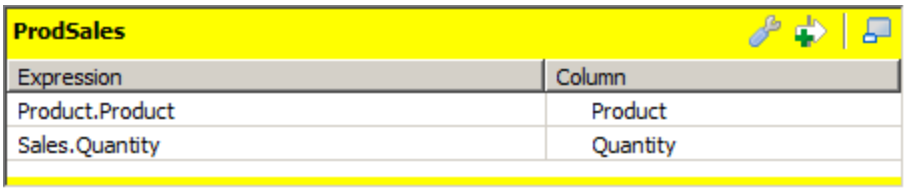
In this exercise, you will calculate the total quantity of each product ordered.

1. ADD ANOTHER OUTPUT TABLE

   Double-click the **tMap** component and add an output table named *ProdSales*.

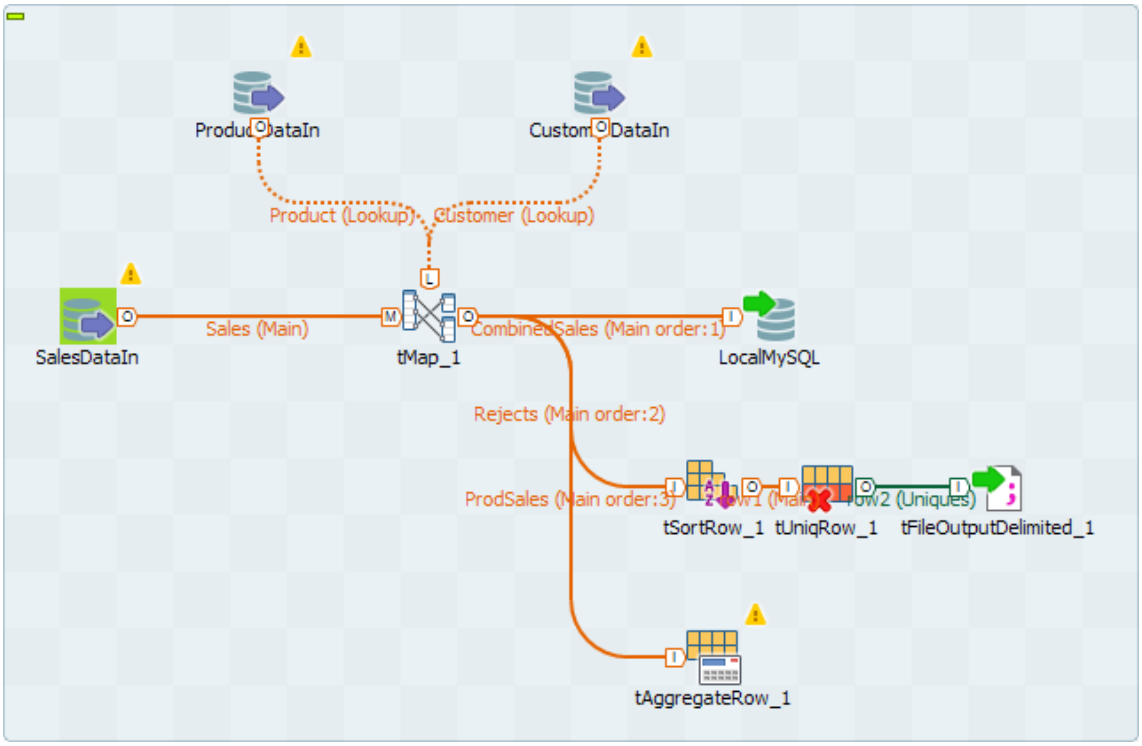
2. MAP PRODUCT NAME AND QUANTITY TO THE NEW OUTPUT TABLE

   Map the product name (*Product* column from the *Product* table) and the quantity ordered (*Quantity* from the *Sales* table) and save the **tMap** configuration.



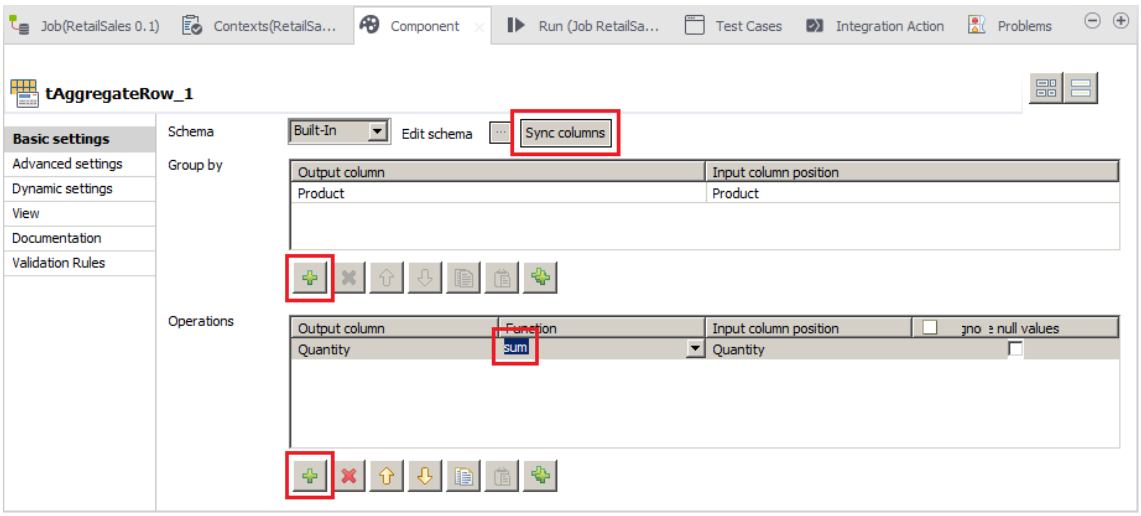
3. ADD A COMPONENT TO CALCULATE TOTAL SALES

Add a **tAggregateRow** component, connecting it to the **tMap** component with the **ProdSales** row. When done, your Job should resemble the following figure.



4. CONFIGURE THE NEW COMPONENT

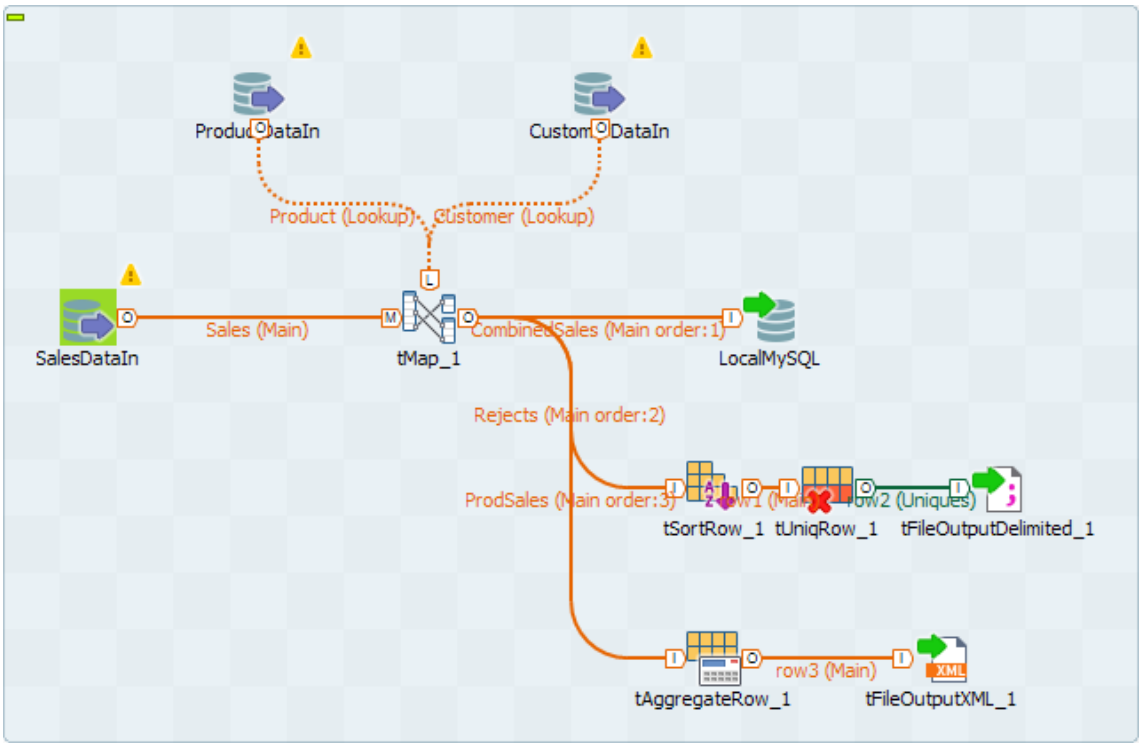
In the **Component** view for the **tAggregateRow** component, click **Sync columns** to synchronize the component schema with the row. Then use the **Add** buttons (+) to add *Product* to the **Group by** table and to add *Quantity* to the **Operations** table. Finish by changing the **Function** to *sum*.



This component now aggregates a sum of the values in the *Quantity* column for each value of the *Product* column.

5. ADD AN OUTPUT COMPONENT FOR THE AGGREGATED DATA

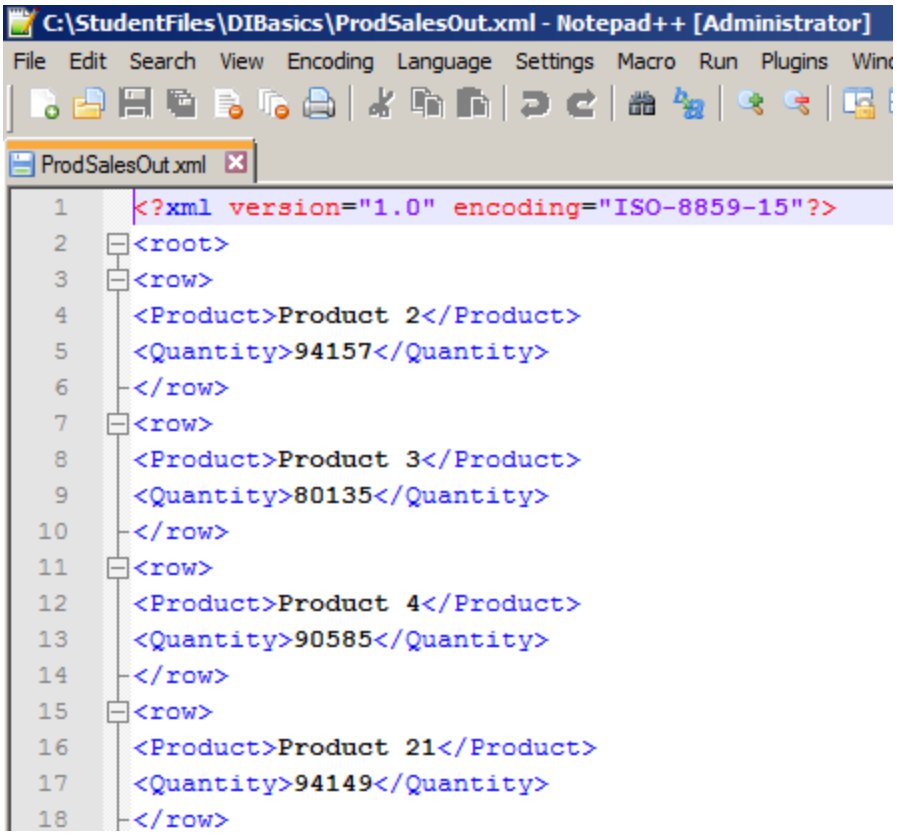
   Add a **tFileOutputXML** component to the design workspace and connect the **tAggregateRow** component to it with a **Main** row. Configure it to write to *C:/StudentFiles/DIBasics/ProdSalesOut.xml*. When done, your Job should resemble the following figure.



This output component creates an output file in XML format. While XML is not particularly user-friendly for humans to read, it can be readily used by a variety of other software applications for further analysis.

6. RUN THE JOB

   Run the Job and then examine the file *C:/StudentFiles/DIBasics/ProdSalesOut.xml* for results.



Next

Now, it's time for you to reinforce your knowledge with exercises.

Challenges

Overview

Complete these exercises to further explore the topics covered here. See Solutions for possible solutions to these exercises.

Total by Customer

Duplicate the **RetailSales** Job and modify it so that it also calculates the total sales quantity by customer in addition to the total by product and stores the results in a new file called *CustSalesOut.xml*.

Clean Up Temporary Files

Add a subJob to your new Job that first creates an archive of the directory containing the three sales files (*C:/StudentFiles/DIBasics/SalesFiles*) and then deletes the *SalesFiles* directory. Be sure that the subJob only runs if the main Job completes successfully, and only delete the sales files if the archive is created successfully.

TIP:
You may find the components **tFileArchive** and **tFileDelete** useful.

Next

You have now finished the lesson. It's time to Wrap-up.

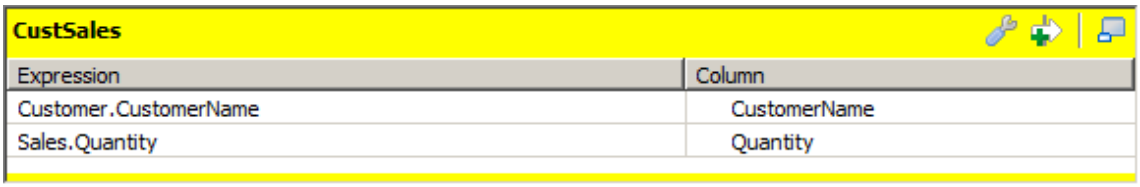
Solutions

Overview

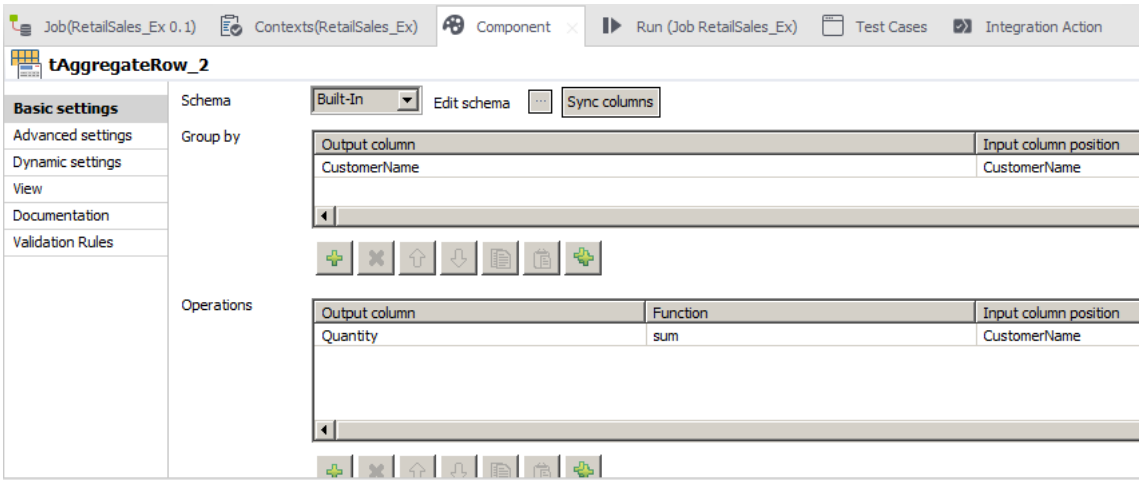
These are possible solutions to the challenges proposed. Note that your solutions may differ and still be valid.

Total by Customer

1. Add another output table to the **tMap** component that includes the customer name and the sales quantity (here named *CustSales*):



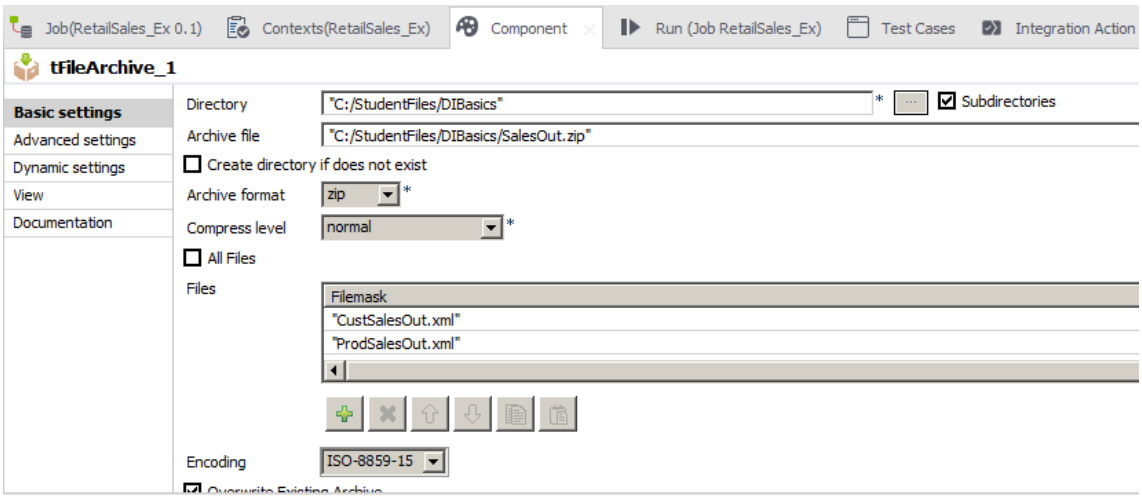
2. Add another **tAggregateRow** component, connecting it to the **tMap** component with a *CustSales* row.
3. Sync the columns and then configure the component to group by the customer name and sum the values in the **Quantity** column.



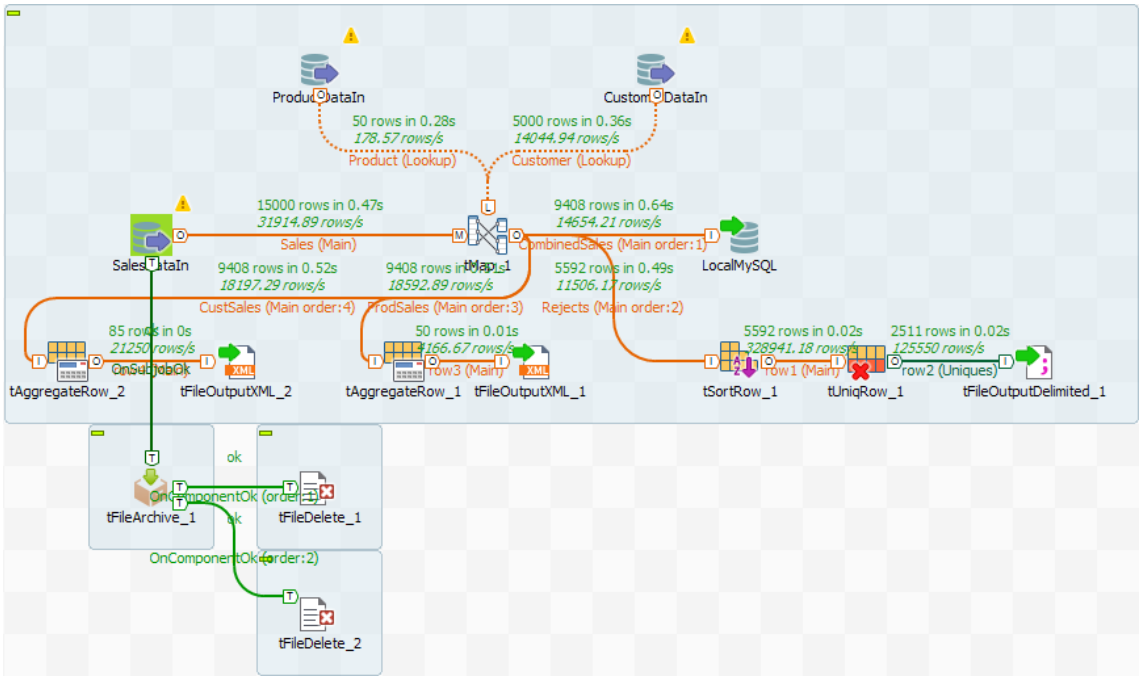
4. Connect this component to a new **tFileOutputXML** component that outputs to *C:/StudentFiles/DIBasics/CustSalesOut.xml*.

Clean Up Temp Files

1. Add a **tFileArchive** component to the Job, connecting it to the **SalesDataIn** component in the main Job with an **OnSubjobOk** trigger.
2. Configure the component to archive the two output files to a *C:/StudentFiles/DIBasics/SalesOut.zip*:



3. Add two **tFileDelete** components next to the **tFileArchive** component, connecting each to the **tFileArchive** with an **OnComponentOk** trigger. Configure one to delete *CustSalesOut.xml* and the other *ProdSalesOut.xml*. When done, your Job should look like this:



Next

You have now finished the lesson. It's time to Wrap-up.

Wrap-Up

In this lesson, you worked with a variety of database access components, including **tCreateTable**, **tMysqlInput**, and **tMysqlOutput**. You learned how to store database connection information as Repository metadata and context variables.

Along the way, you worked with **MySQL Workbench** to retrieve database information, the **tSortRow** and **tUniqRow** components to single out unique combinations of column values, and used **tAggregateRow** to calculate aggregate values for data.

Next step

Congratulations! You have successfully completed this lesson. To save your progress, click **Check your status with this unit** below. To go to the next lesson, on the next screen, click **Completed. Let's continue >**.

LESSON 9

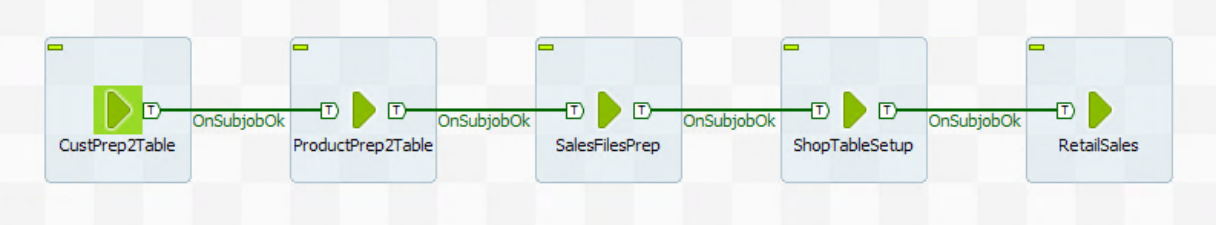
Creating Master Jobs

This chapter discusses:

Creating Master Jobs

Lesson Overview

In this lab you will create a master Job which includes most of the previous Jobs you have created:



Once the master Job is validated, you will export it with all dependencies so that you can provide a package to someone else who might need to run your Job.

Objectives

After completing this lesson, you will be able to:

- Create a master Job
- Override context variables for Subjobs
- Export a Job and its dependencies
- Use components to create an archive and delete files

Next Step

The first step will be to create a master Job which runs the different Jobs.

Controlling Job Execution Using a Master Job

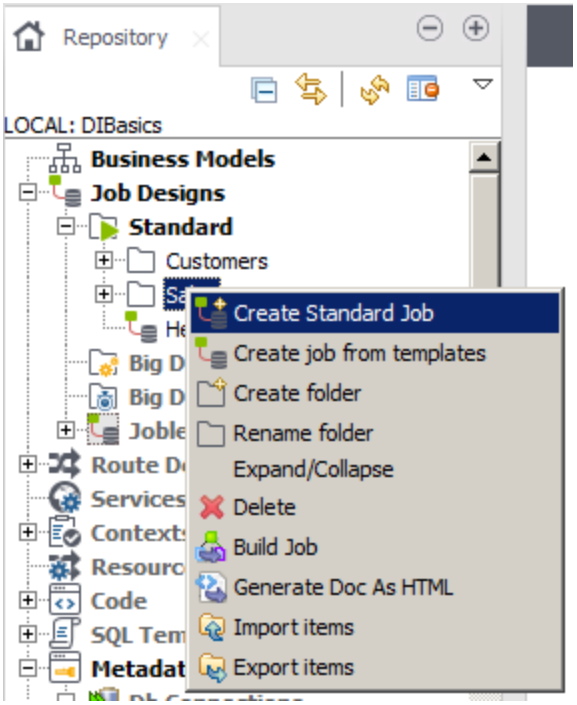
Overview

In this exercise you will build a master Job, based on the smaller Jobs you built previously, and export it with its dependencies. These dependencies include context variables and metadata.

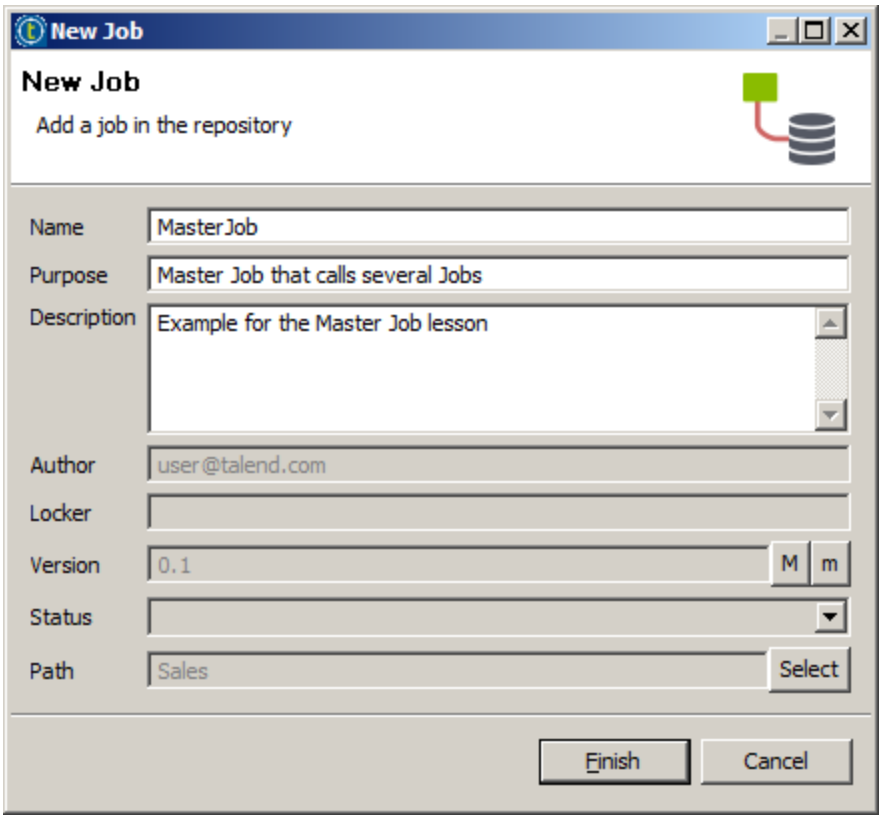
Create a Master Job

1. CREATE A STANDARD JOB

   Right-click **Repository > Job Designs > Standard > Sales** and select **Create Standard Job**.

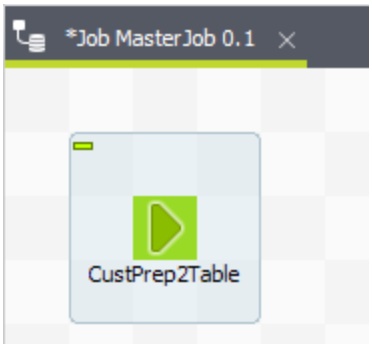


   Name the new Job *MasterJob*. Enter a **Purpose** and **Description**, then click **Finish**.



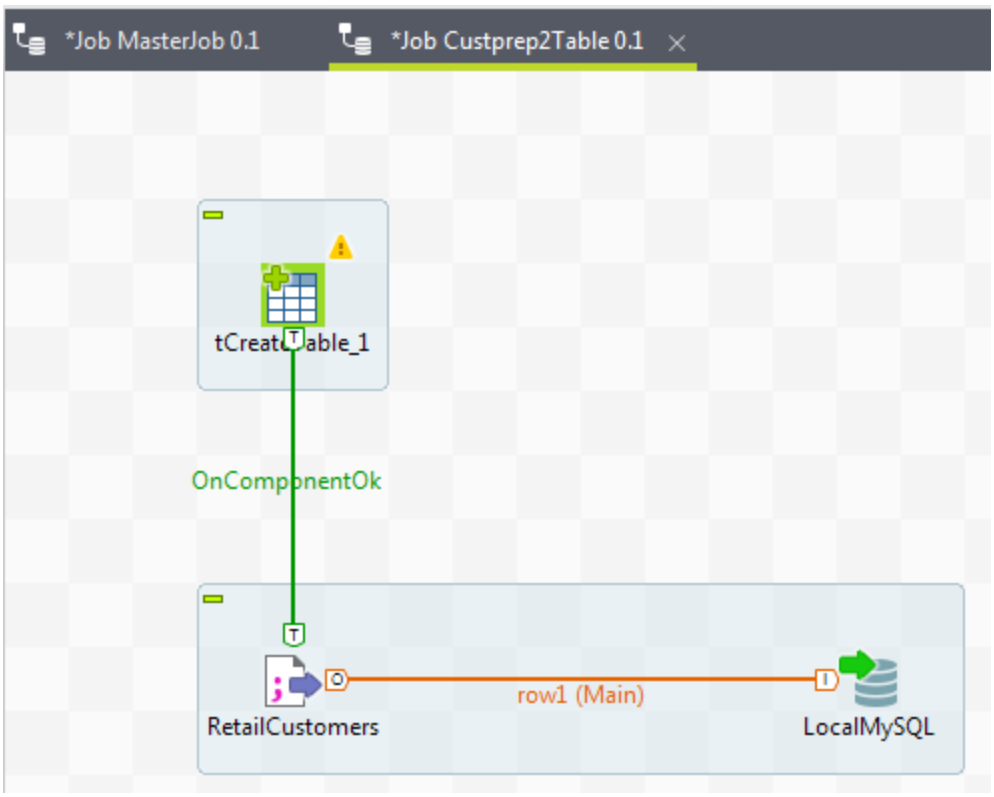
2. ADD A JOB

   From the **Repository**, drag the **CustPrep2Table** Job to the **Designer**. This will add a **tRunJob** component configured to run your **CustPrep** Job.



3. EXPLORE THE tRunJob COMPONENT

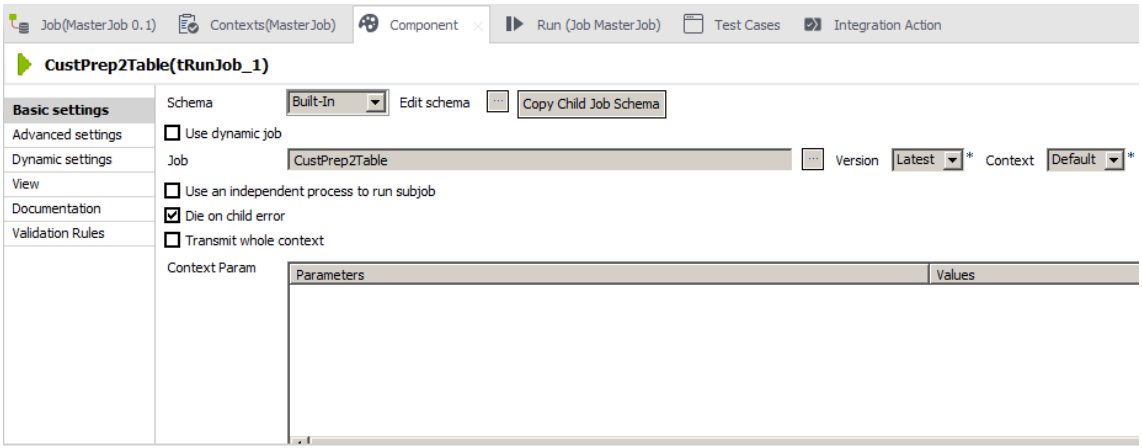
   Double-click the **CustPrep2Table** component. Typically, double-clicking a component opens the **Component** view. But, for **tRunJob** components, a double-click will open the referenced Job as a separate tab in the **Designer**.



4. EXPLORE THE CONFIGURATION

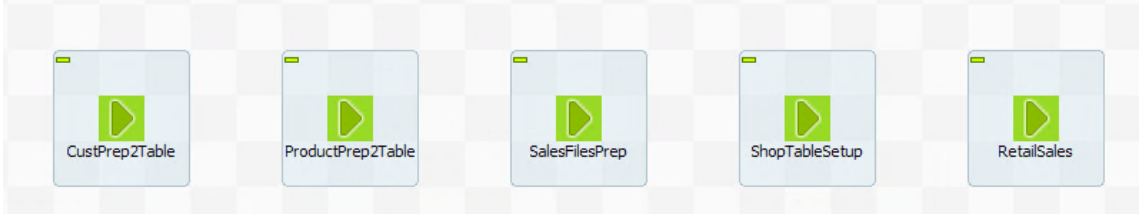
   Switch back to the **MasterJob** tab. Click the **CustPrep2Table** component to select it, then click the **Component** tab to see the configuration.

   Here, there are options to control the version of the Job to run, as well as the context in which to run the Job. Individual context parameters are also configurable.



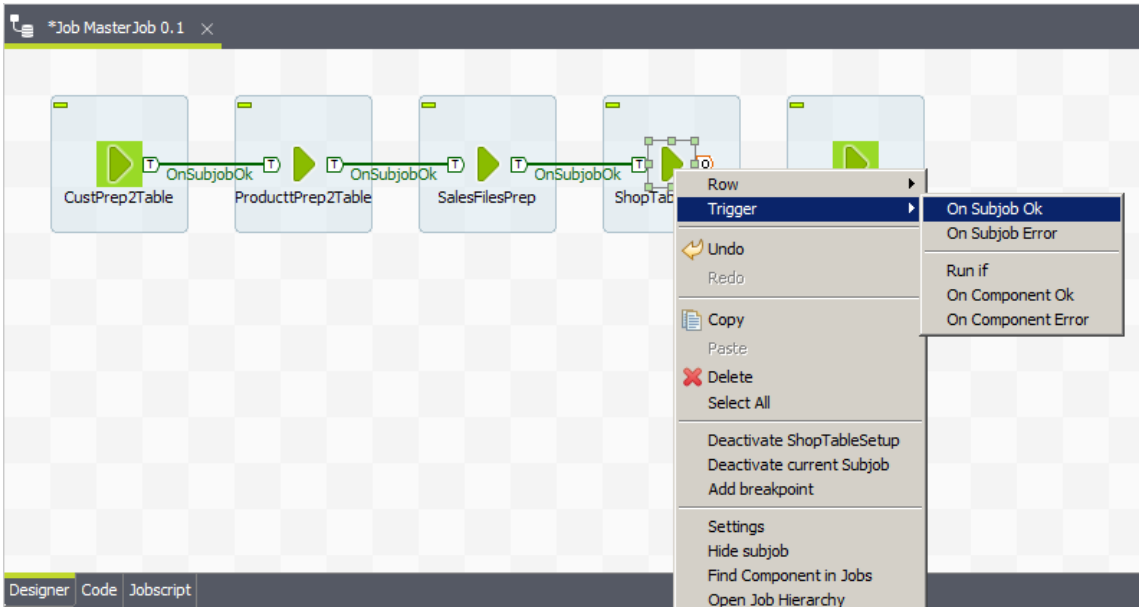
5. ADD MORE JOBS

   Using the same procedure, add the **ProductPrep2Table** , **SalesFilesPrep** , **ShopTableSetup** , and **RetailSales** Jobs.



6. CONNECT THE JOBS

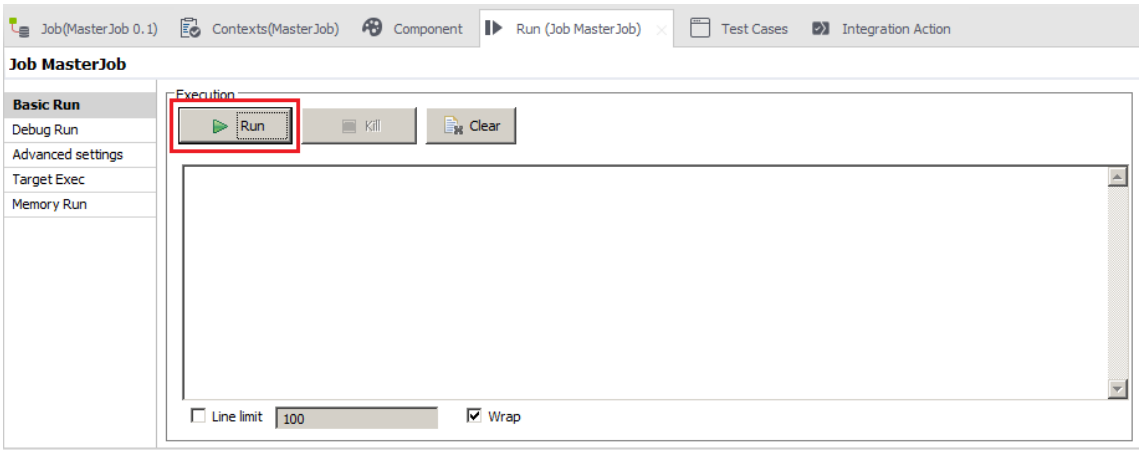
Connect the Jobs with an **OnSubjobOK** trigger.



NOTE:
When sequencing Jobs using a collection of **tRunJob** components, it is considered best practice to link them using the **On Subjob Ok** trigger (rather than the **On Component Ok** trigger).

7. RUN THE MASTER JOB AND CHECK RESULTS

Run the m aster Job.



Since you tested every Job independently, your master Job should execute without any errors. Review the results in the *C:/StudentFiles/DIBasics* folder, being sure to examine the time stamps on the generated files.

Overriding Context variables

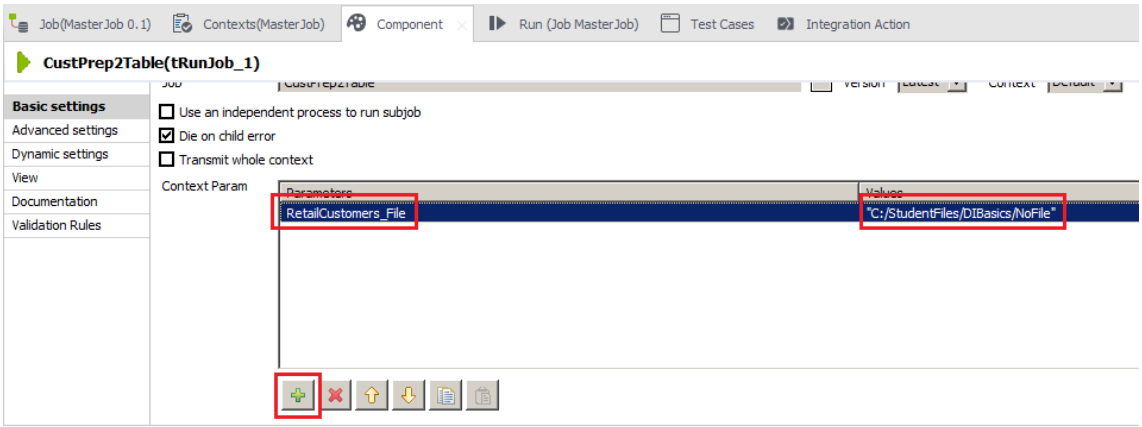
To run the master Job, the Studio used the default context variables values. Those values can be changed in each **tRunJob** instance.
Recall that the file names to be read by the different Jobs were saved as context variables. In this exercise, you will take advantage of this fact to change the file names read by the **CustPrep2Table** and **ProductPrep2Table** subJobs.

1. CONFIGURE THE CustPrep2Table COMPONENT

   Click the **CustPrep2Table** component and then click the **Component** tab.

   Below the **Context Param** table, click the **Add** button (). Then, in the **Parameters** column, select *RetailCustomers_File* from the list.

   In the **Values** column, enter a value of *"C:/StudentFiles/DIBasics/NoFile"*.



2. CONFIGURE THE ProductPrep2Table COMPONENT

   In a manner similar to the previous step, set the value of the *RetailProducts_File* parameter in the **ProductPrep2Table** Job. This time, use a value of *"C:/StudentFiles/DIBasics/NoFile2.csv"*.

3. RUN THE JOB

   Run the Job and check the messages in the **Run** view. This time, you will see error messages saying that *NoFile* and *NoFile2.csv* cannot be found.

4. FIX THE ERRORS

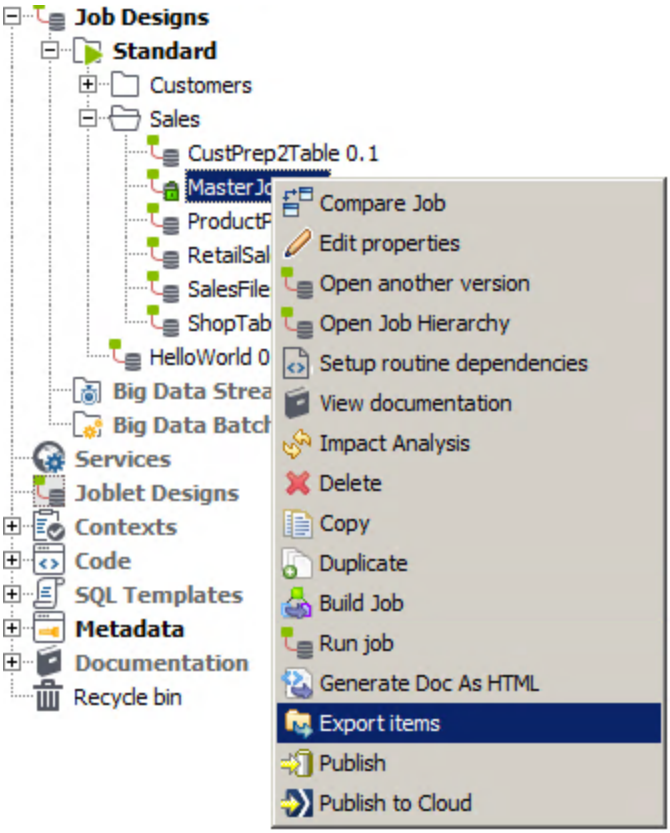
   In order to export a functional Job, delete the context variables you just created. Run the Job to verify that it works properly again, and finally, save the Job.

Exporting a Master Job

Now that the master Job has been successfully tested the next step is to export it so that you can send it to someone else for future use.

1. EXPORT THE JOB

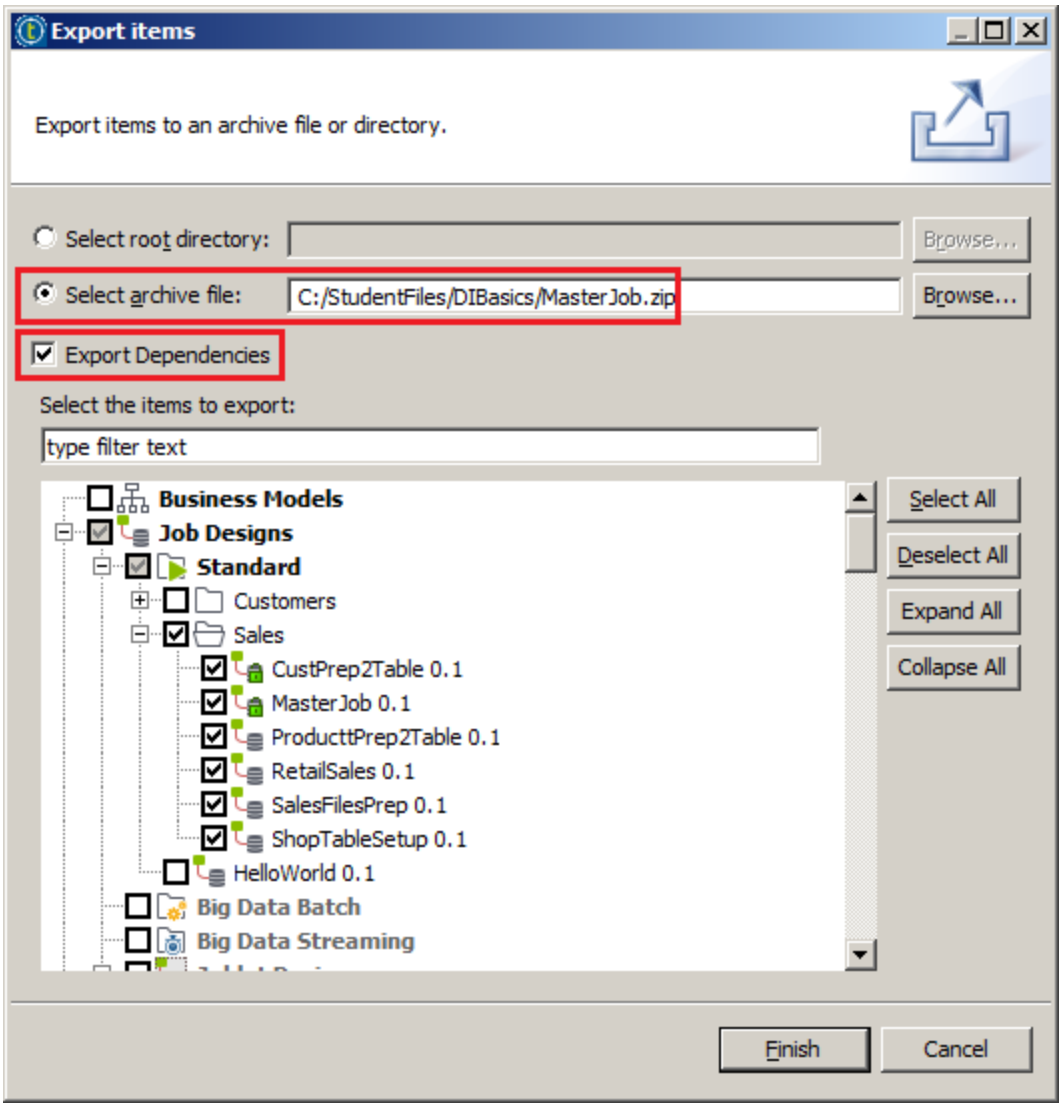
   Right-click **Repository > Job Designs > Standard > Sales > MasterJob** and select **Export Items**.



The **Export Items** window enables you to configure what you want to export and how you want to export it. For this example, you will export your Job as an archive file with all dependencies, so that everything needed to run *MasterJob* is included in the archive file.

Click **Select archive file** and enter *C:/StudentFiles/DIBasics/MasterJob.zip*. Select **Export Dependencies**. Notice that enabling the **Export Dependencies option** automatically selects additional components to export. Disable and enable the options a few times to see the effects this option has, but ensure it is selected before proceeding.

Finally, click **Finish**.



2. CHECK THE ARCHIVE FILE

   Navigate to the *C:/StudentFiles/DIBasics* folder and check that the archive file *MasterJob.zip* has been created there. Feel free to explore the contents.

Next

You have now finished the lesson. It is time to Wrap-up.

Wrap-Up

In this lesson, you learned how to build a master Job: you put several Jobs in a single master Job using **tRunJob** components. You also exported this master Job in a zip file, including all its dependencies such as context variables and metadata.

Next step

Congratulations! You have successfully completed this lesson. To save your progress, click **Check your status with this unit** below. To go to the next lesson, on the next screen, click **Completed. Let's continue >**.

Intentionally blank

LESSON 10

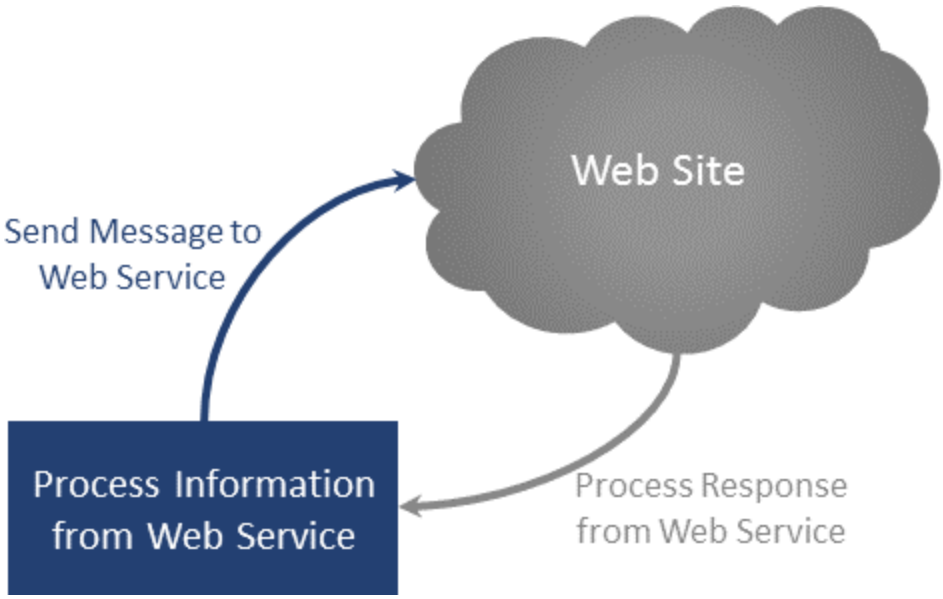
Working with Web Services

This chapter discusses:

Working with Web Services

Lesson Overview

You have used Talend Studio to read from and write to both files and database tables. In this lesson you will access a Web Service. Specifically, you will create a Job that uses a Web Service method to retrieve information for a location identified by a US Zip code:



Objectives

After completing this lesson, you will be able to:

- Use a Talend component to access a Web Service method
- Extract specific elements from a Web Service reply
- Store Web Service WSDL Schemas for use in **tXMLMap** component

Next Step

The first step is to create a new Job.

Accessing a Web Service

Overview

Talend Studio allows you to interact with a Web Service as part of a project. In this example, you retrieve information from a free Web Service that can be used for testing Web Service calls.

WARNING:
We cannot guarantee the full-time availability of this open access Web Service. Moreover, it may not have any results for certain Zip codes.

Retrieve Information from a Web Service

1. CREATE A NEW JOB

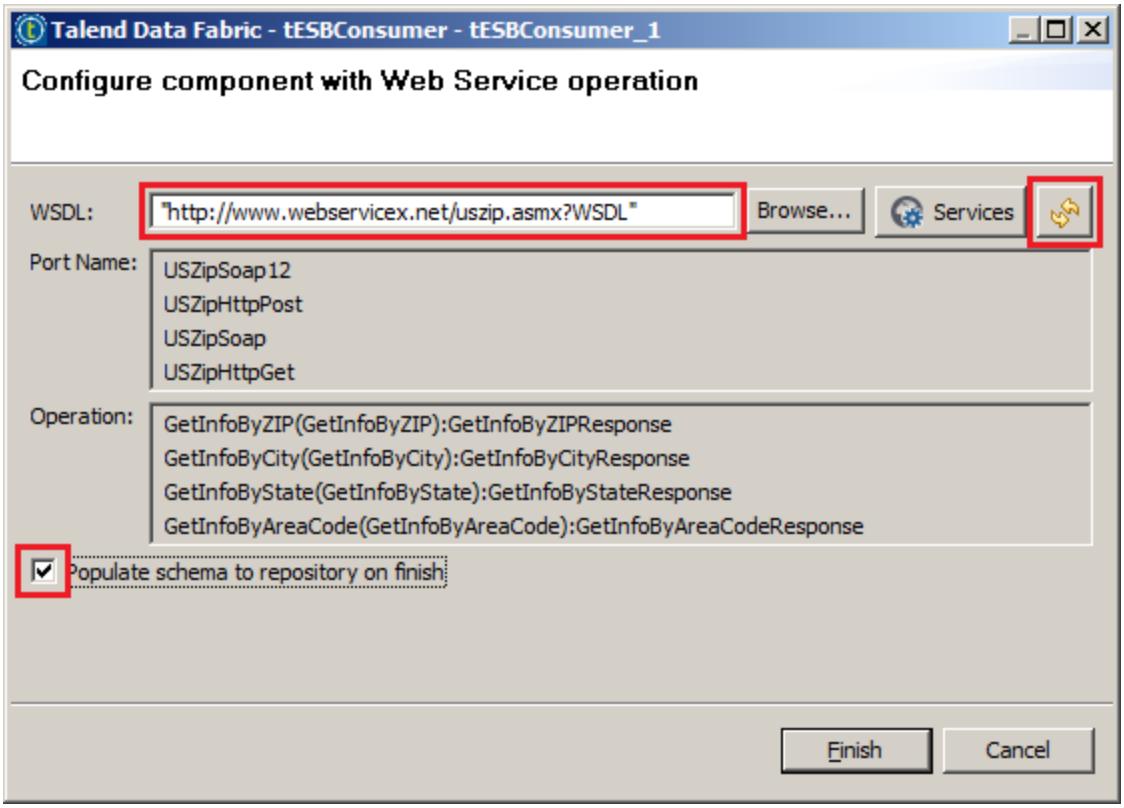
   Under **Repository > Job Designs > Standard**, create a new folder called *WebService*, and inside that folder, create a new Job called *GetInfoByZip*.

2. ADD AND CONFIGURE A COMPONENT TO INVOKE A WEB SERVICE

   Add a new **tESBConsumer** component and double-click it to open the configuration window.

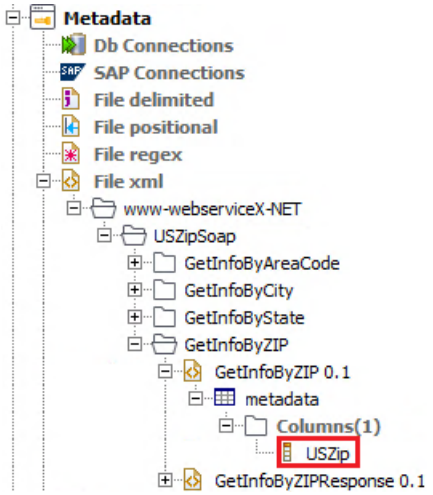
   Update the **WSDL** field with the following URL: *"http://www.webservicex.net/uszip.asmx?WSDL"*. Then click the refresh button ( ) to populate the **Port Name** and **Operation** sections.

   Select the **Populate schema to repository on finish** check box and click **Finish**.



3. CHECK THE SCHEMA

Open **Repository > Metadata > File xml > www-webserviceX-NET**. Continue opening the sub-folders until you see the request **metadata** of the **GetInfoByZIP** SOAP operation. The request should contain only the column **USZip**.

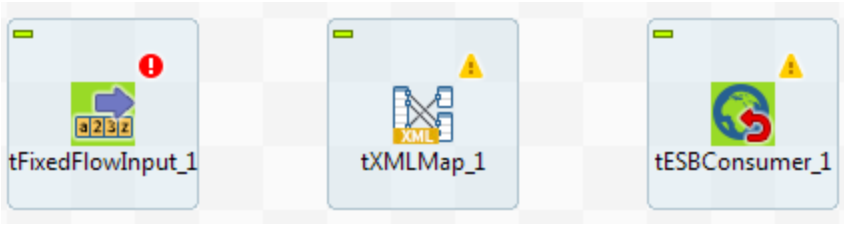


4. ADD A COMPONENT TO INPUT A ZIP CODE

   Enter a new component **tFixedFlowInput** as the input of your Job. Place it to the left of the **tESBConsumer** component.

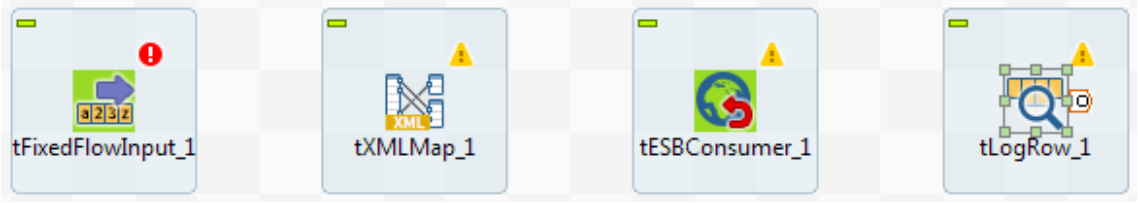
5. ADD A MAPPING COMPONENT

   Place a **tXMLMap** component in between the input component and the **tESBConsumer** component.



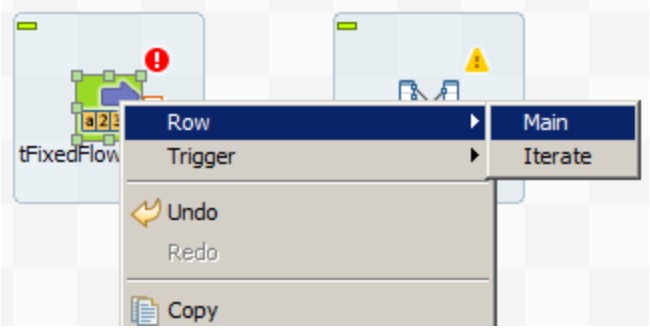
6. ADD AN OUTPUT COMPONENT

   Add a **tLogRow** component to the right of **tESBConsumer_1**. This will write output to the console.



7. CONNECT THE COMPONENTS

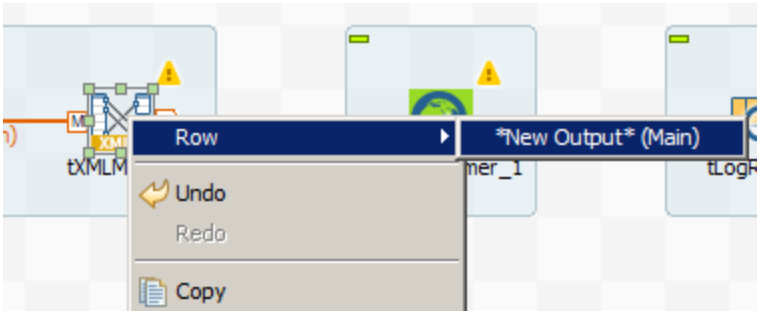
   Right-click **tFixedFlowInput** and connect it to **tXMLMap_1** with a **Main** connection.



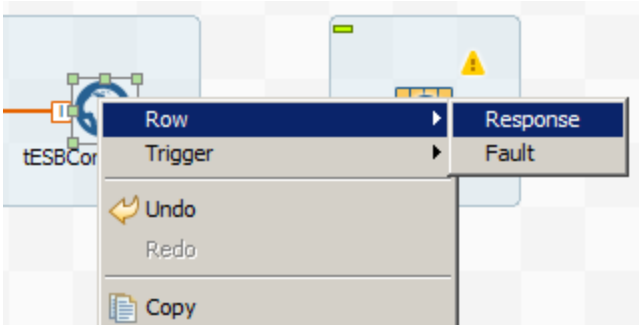
Right-click **tXMLMap** and connect it to **tESTConsumer_1** with a ***New Output* (Main)**connection.

NOTE:

- When prompted, name the connection *USZip*.
- When prompted to get the schema of the target component, click **Yes**.

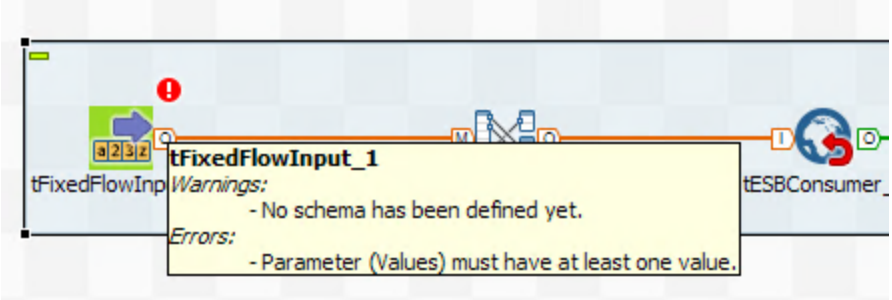


Right-click **tESBConsumer_1** and connect it to **tLogRow_1** with a **Response** connection.

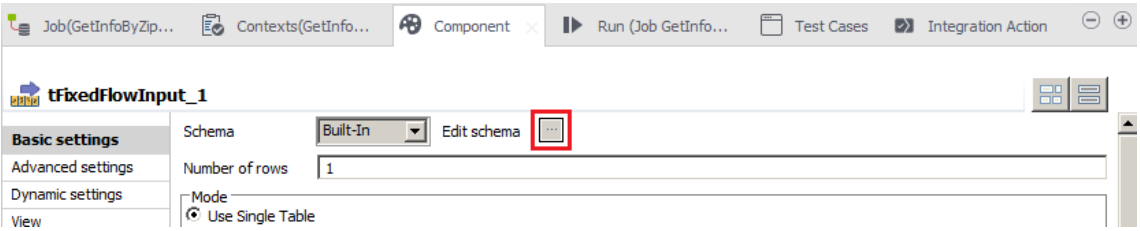


8. CORRECT ERRORS

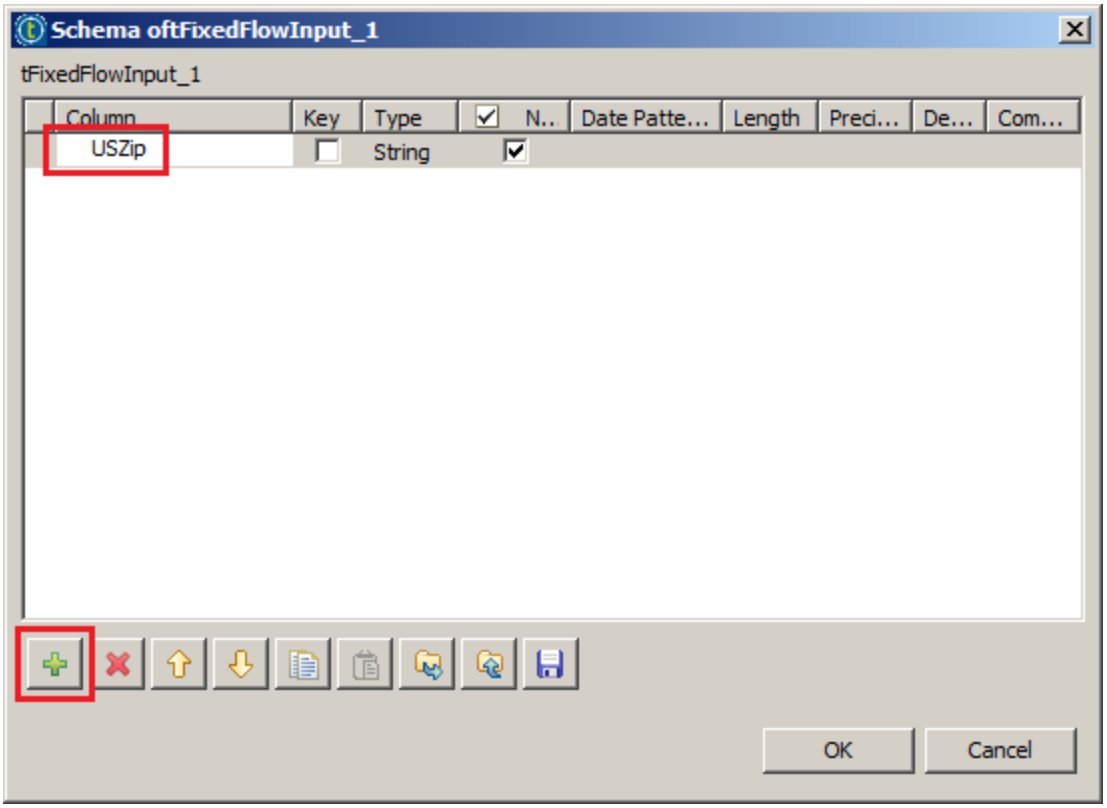
   Notice that the input component shows an error indicating that no schema has been defined yet.



Double-click the **tFixedFlowInput** component to open the **Component** view, then click the **[...]** button next to **Edit schema**.

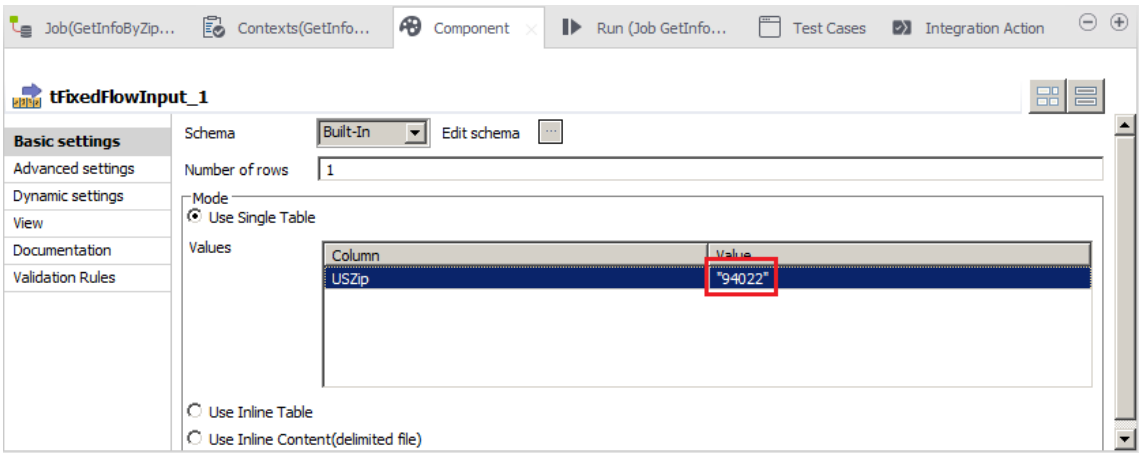


In the **Schema of tFixedFlowInput_1** window, click the **Add** button (+). Name the new column *USZip*, then click **OK**.



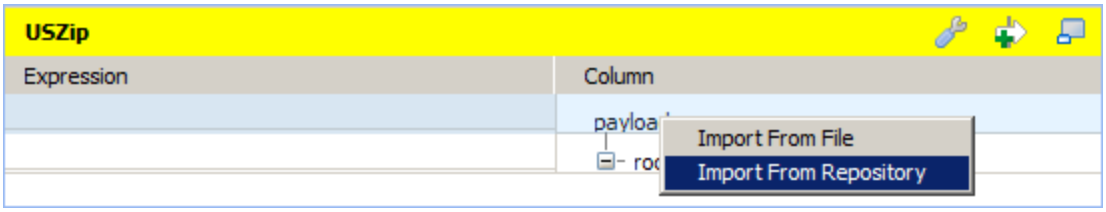
9. ENTER A ZIP CODE

   Still in the **Component** view of **tFixedFlowInput_1**, enter *"94022"* into the **Value** column.

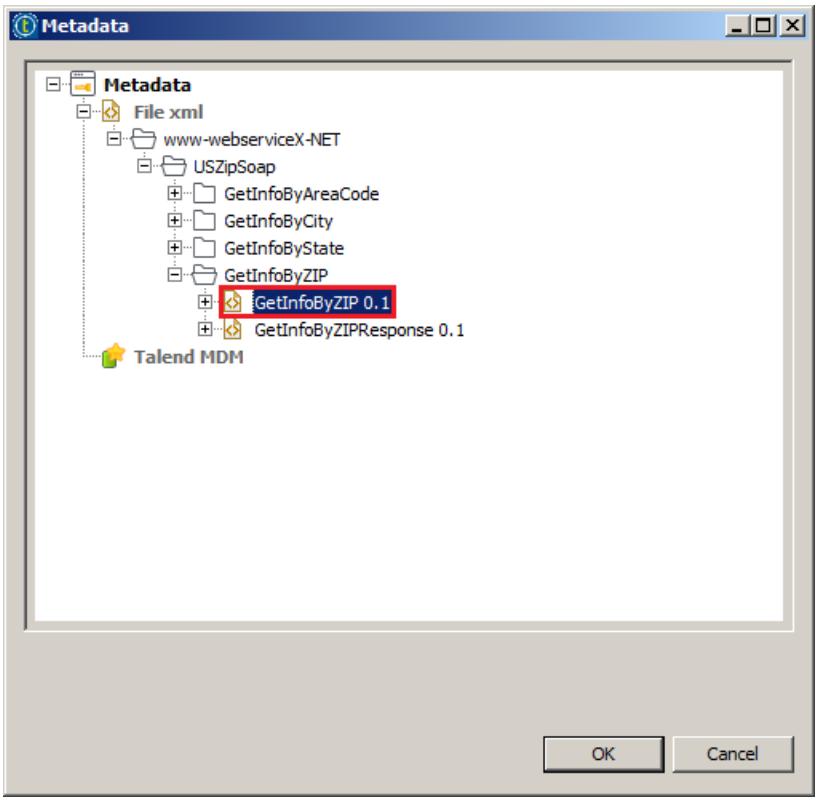


10. CONFIGURE THE tXMLMap OUTPUT TABLE

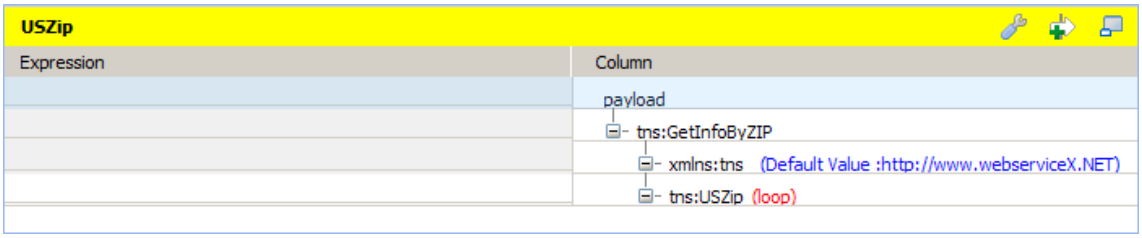
Double-click the **tXMLMap** component to display the mapping editor. Then, in the output table, right-click on **payload** and select **Import From Repository**.



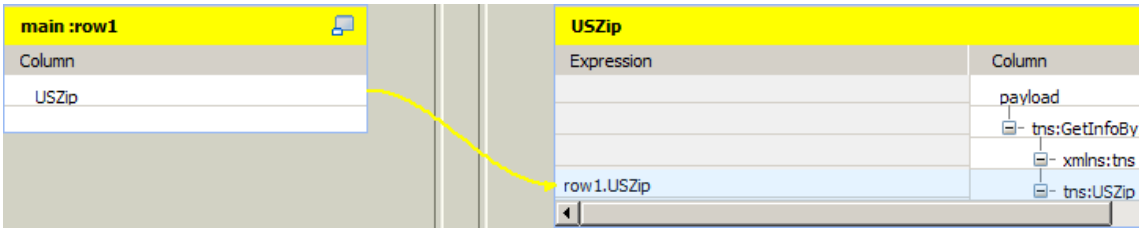
Search for the previously saved metadata under the **Metadata** folder until you find **GetInfoByZIP 0.1**. Select it and then click **OK**.



You can now see the request metadata was added to the payload in your **tXMLMap** component.

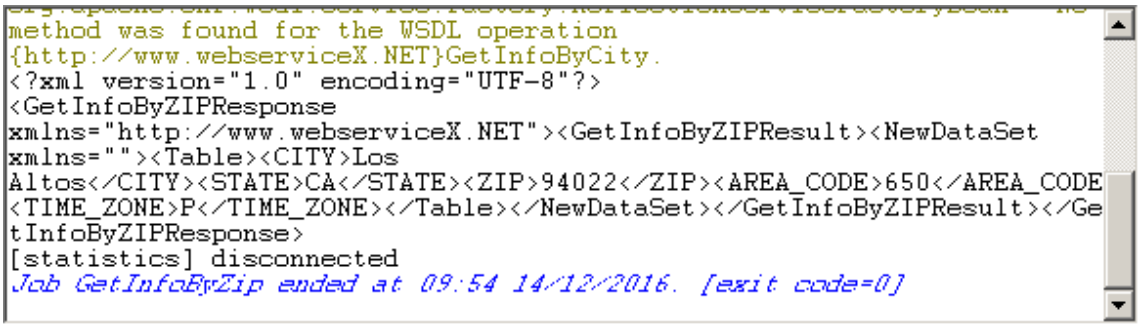


Now, map the *USZip* element by dragging it from the *main:row1* schema to the **Expression** cell beside **tns:USZip**. Click **OK** to save the changes.



11. RUN THE JOB

    Run the Job and examine the results.



Notice that the message received relates to the **GetInfoByZIPResponse** metadata. The data returned by the web service is a collection of geographical details for Los Altos CA (which is the city with a zip code of 94022).

Next

You have now finished this lesson. It's time to Wrap-Up.

Wrap-Up

In this lesson, you used the **tESBConsumer** component to call a Web Service. You saw how to map the input parameters to a request schema and how to check the resulting response schema.

Next step

Congratulations! You have successfully completed this lesson. To save your progress, click **Check your status with this unit** below. To go to the next lesson, on the next screen, click **Completed. Let's continue >**.

Intentionally blank

LESSON 11

Running a Standalone Job

This chapter discusses:

Running Jobs Standalone

Lesson Overview

So far, you have run your Jobs using Talend Studio. In a production environment, you will want to be able to run your Jobs independently of the Studio. For example, you may wish to run them on a different Windows or Linux server.

In this lesson, you will take an existing Job and set it up so that it can run independently.

Objectives

After completing this lesson, you will be able to:

- Build and export a Job
- Run an exported Job outside of the Talend Studio
- Create a new version of an existing Job

Next Step

The first step is to build a Job.

Building a Job

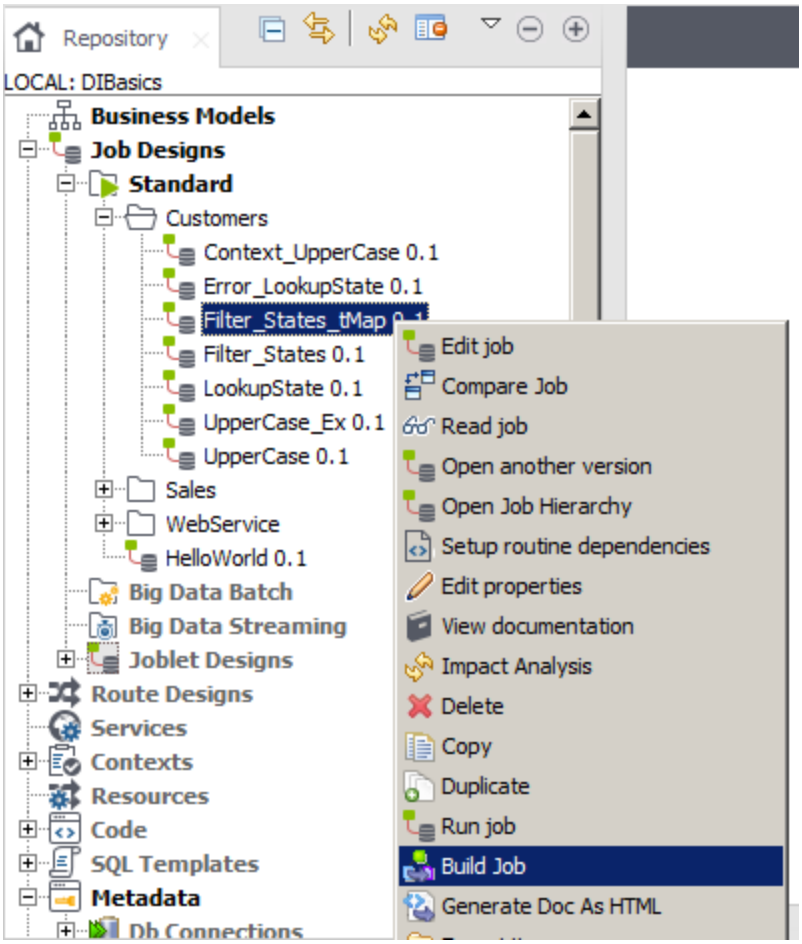
Overview

In an earlier lesson, you exported Jobs. While these Jobs can be shared with others, they still only run within the context of the Talend Studio. In many situations, you will need to run a Talend Job independently of the Talend Studio. In order to do that, you need to build and export the Job.

Build Job

1. BUILD A JOB

   Right-click **Repository > Job Designs > Standard > Customers > Filter_States_tMap** and select **Build Job**.

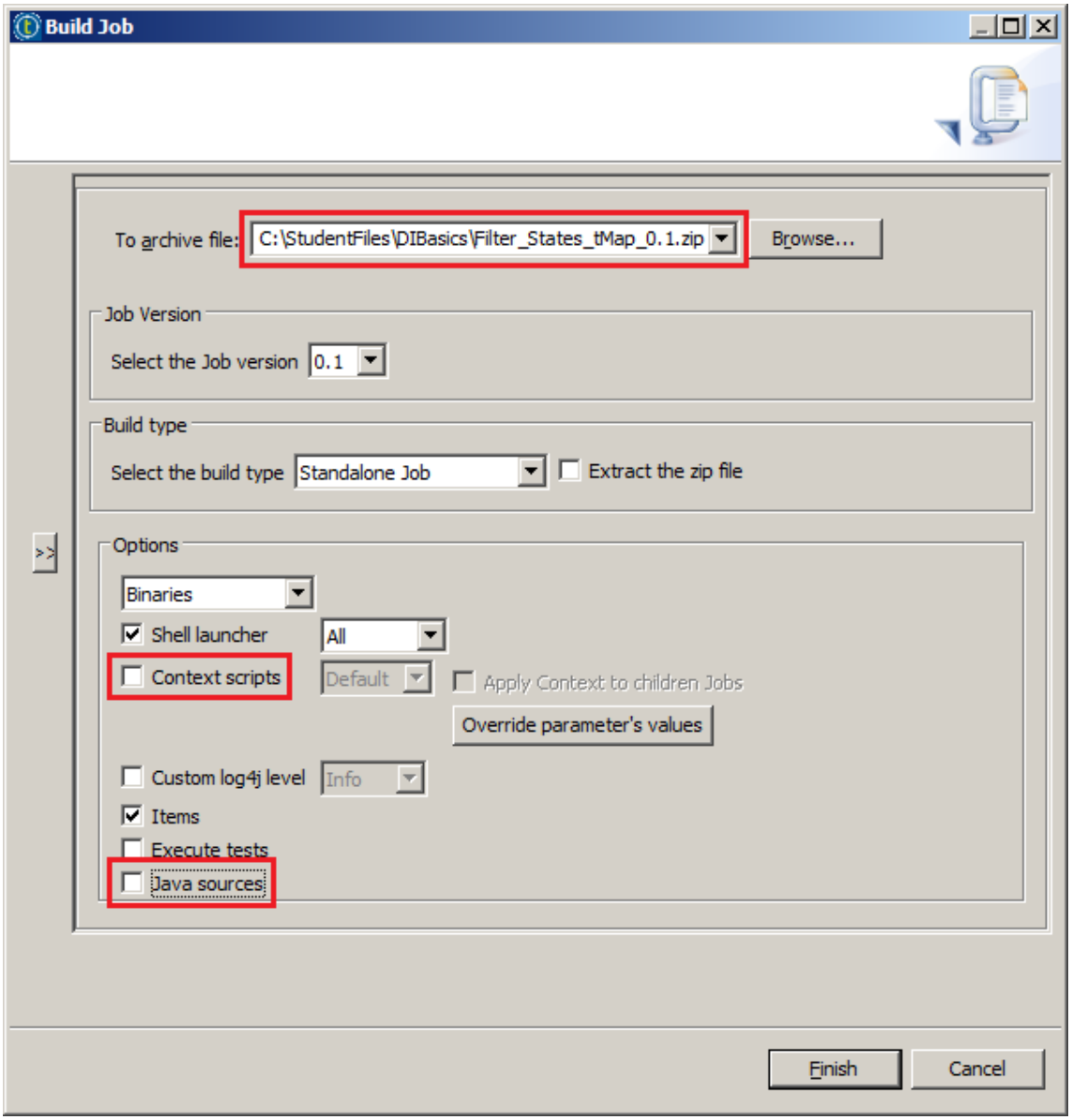


   The **Build Job** window opens, providing you with several options for exporting the Job in a variety of formats.

   Use the **Browse** button to navigate to and specify a path of *C:/StudentFiles/DIBasics/Filter_States_tMap_0.1.zip*. Alternatively, you can manually enter the path.

   Next, clear the **Context scripts** and **Java sources** check boxes. The Context scripts option is useful if you created context variables in your Job that you want written to a configuration file. The **Java sources** option is useful if you need the Java source files to rebuild outside of the Talend Studio. For the purpose of this exercise, neither of these options is needed.

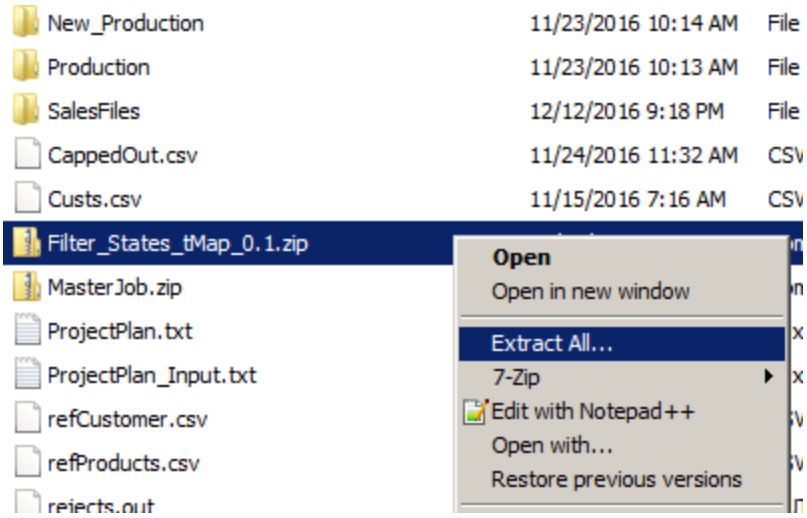
Finally, click **Finish**.



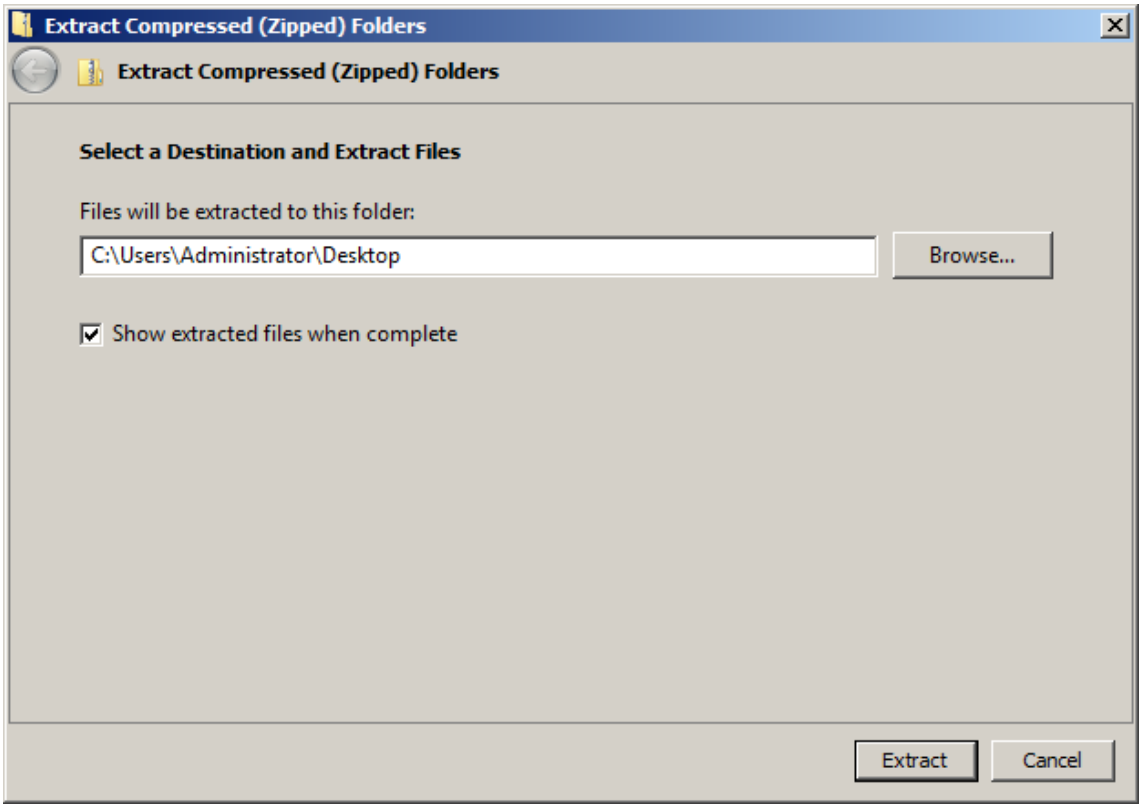
Extract and Run

1. EXTRACT THE ARCHIVE

   Locate the file you just created, *Filter_States_tMap_0.1.zip*. Right-click on it and select **Extract All** to extract the content.



Specify the location of your choice. The location shown below shows the user's *Desktop* folder.



2. EXECUTE THE SCRIPT THAT RUNS THE JOB

   Locate the script that runs the job. In this instance, this is located at *Filter_States_tMap/FilterStates_tMap_run.bat* folder, relative to the location at which the archive was extracted.

   Double-click **FilterStates_tMap_run.bat** to execute it.

   A command window opens briefly as the Job executes, but because the Job has no visible results, it is not easy to see what happened. However, recall that this job generates three output files: *AKOut.csv*, *CAOut.csv*, and *NYOut.csv*. Check that

these files have been created in the *C:/StudentFiles/DIBasics* folder, and have a date and time associated with them which correspond to the time you ran **FilterStates_tMap_run.bat**.

Next

Next you will modify the Job to explore Job versions.

Modifying a Job

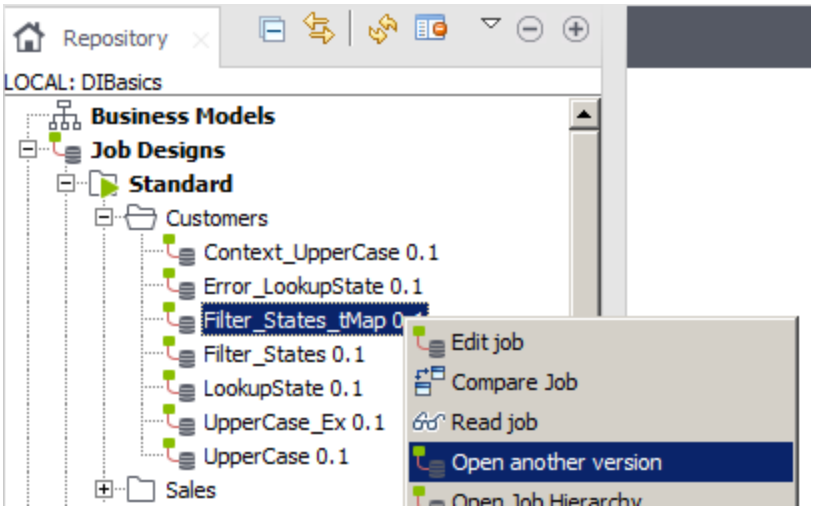
Overview

When you are working on a large project, you may want to save the current version of a Job before making changes, in case you need to revert.

Create a New Version

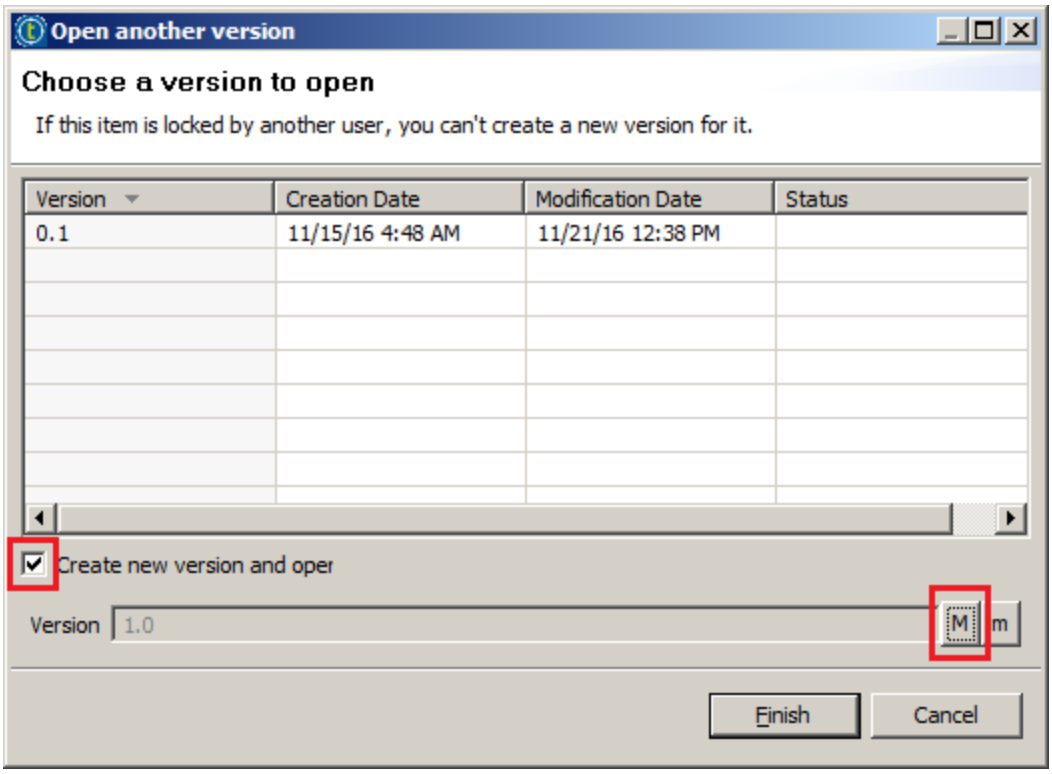
1. OPEN A NEW VERSION OF A JOB

   Right-click **Repository > Job Designs > Standard > Customers > Filter_States_tMap** and select **Open another version**.



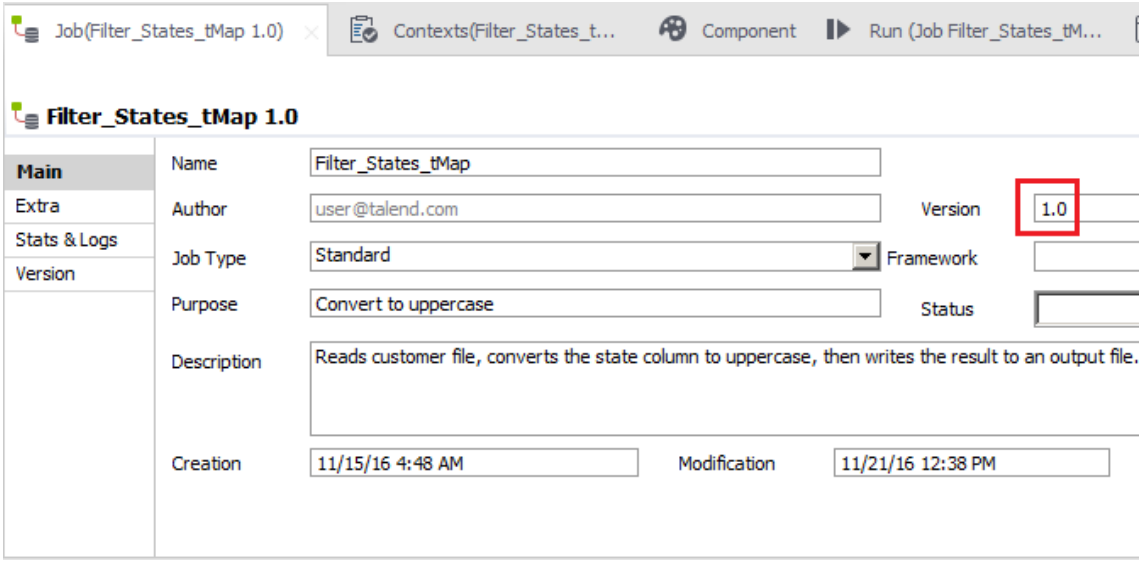
   The **Open another version** window is displayed, which allows you to open other versions or create new versions of a Job. Select the **Create new version and open** check box and click the button marked with a capital **M**. Notice that **Version** changes from *0.1* to *1.0*. These two buttons are used to make adjustments to the major and minor version number.

   Click **Finish**.

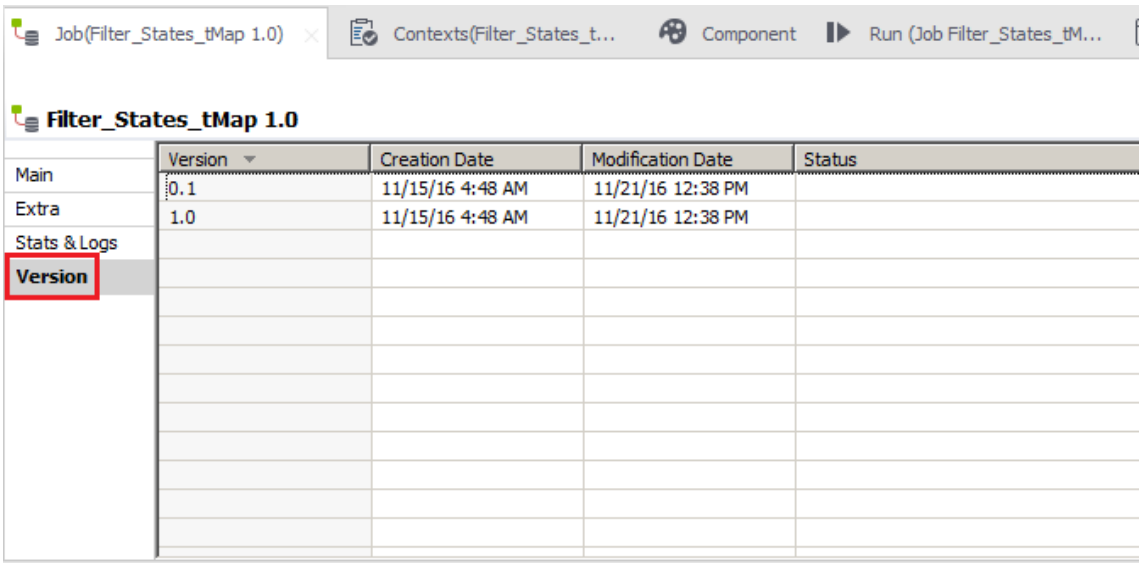


2. OBSERVE THE NEW JOB INFORMATION

In the **Job** view you can now see the new version number.



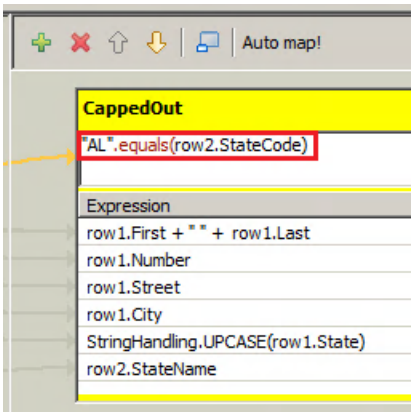
Click on the **Version** tab and notice the revision history of the Job that is kept here.



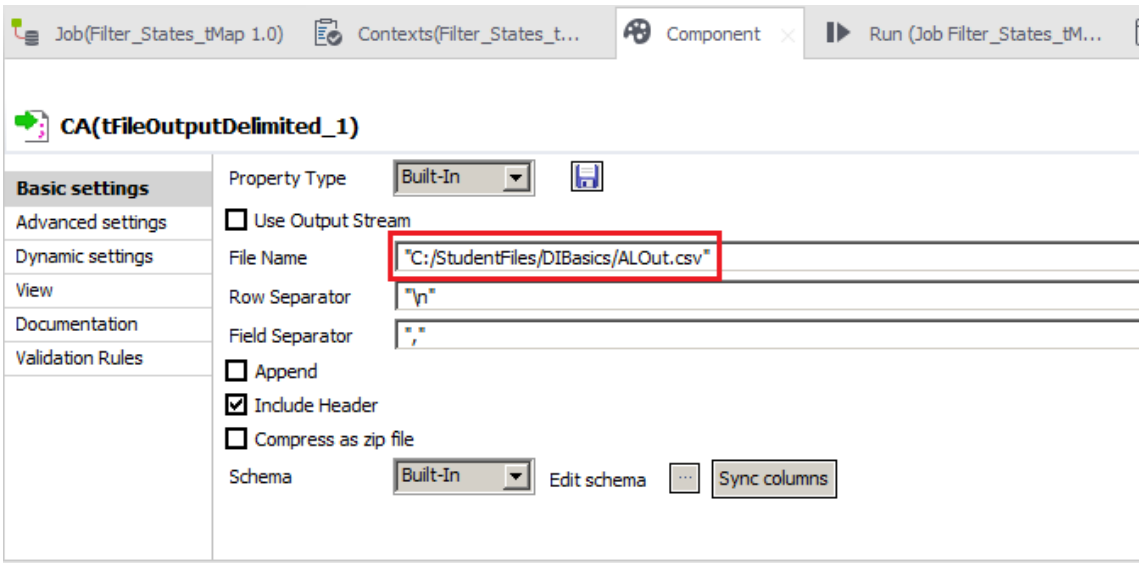
Modify the new version of the Job

1. MODIFY THE JOB

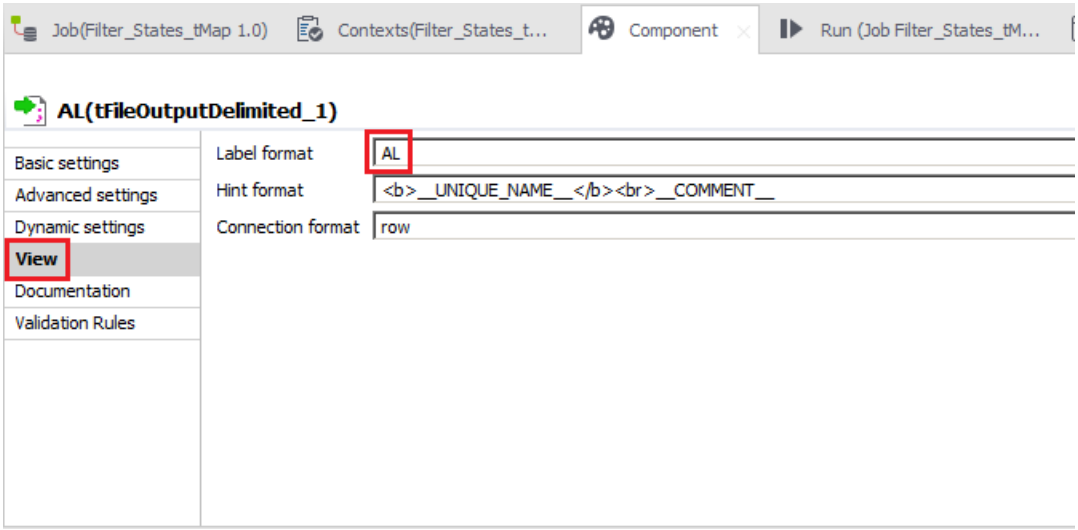
   Double-click the **tMap_1** component and change the filter criteria for the **CappedOut** table to `"AL".equals(row2.StateCode)` and click **Ok**.



   Match the change to the filter with a corresponding change to the output filename. Double-click the output component **CA** to open the **Component** view and change the file name to *"C:/StudentFiles/DIBasics/ALOut.csv"*.



For consistency, change the component name by clicking the **View** tab and updating the **Label Format** to *AL*.



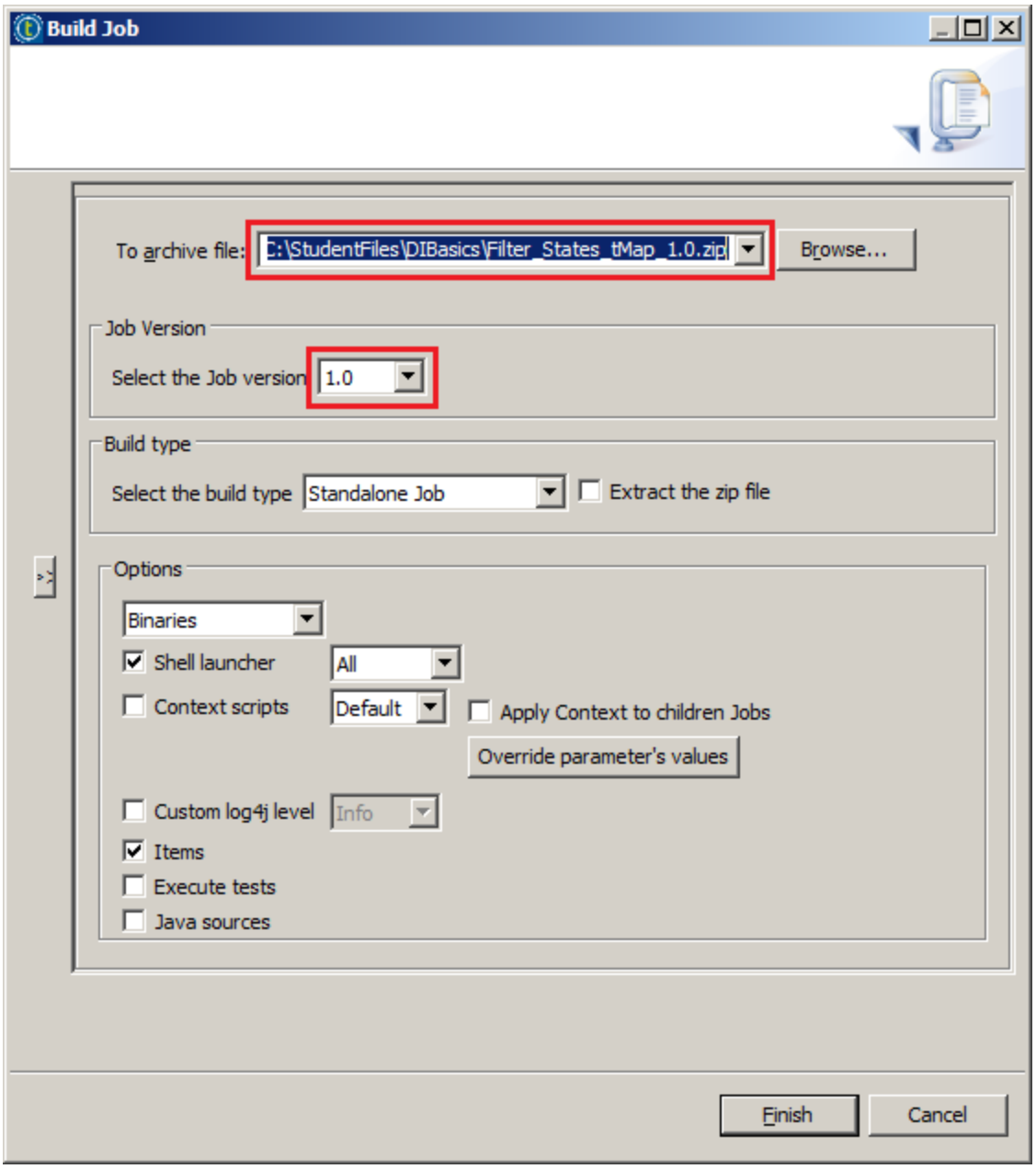
2. SAVE THE JOB

   Save the modified Job but do not run it yet.

Build and Run

1. BUILD THE NEW VERSION OF THE JOB

   Following the same procedure as before, build the Job again. This time, make sure both the version and file name both reflect version *1.0*.



2. RUN THE BUILD JOB AND VERIFY THE RESULTS

   Extract the contents of the archive, locate and execute the script that runs the Job, and then examine the new output file, *ALOut.csv*.

   The presence of the new file alone is evidence that the Job executed successfully.

Next

You have now finished this lesson. Complete the exercises to reinforce your understanding of the topics covered.

Challenges

Overview

Complete these exercises to further explore the topics covered here.

Display Output

Change the other two filter criteria in the **tMap** component to *MN* and *IL* and also change the output names and files.

Build and run

Build the Job once again, extract the contents, and execute the script.

Next

You have now finished this lesson. It's time to Wrap-Up.

Solutions

Overview

These are possible solutions to the challenges. Note that your solutions may differ and still be valid.

Display Output

Follow the same steps that you followed to modify the version 1.0 of the Job and then check the new output files.

Build and run

Build and then Run the Job by using the executable file and verify the output files.

Next

You have now finished this lesson. It's time to Wrap-Up.

Wrap-Up

In this lesson, you built a Job so that you could run it independently of Talend Studio.
You also learned how to create multiple versions of the same Job.

Next step

Congratulations! You have successfully completed this lesson. To save your progress, click **Check your status with this unit** below. To go to the next lesson, on the next screen, click **Completed. Let's continue >**.

Intentionally blank

LESSON 12

Documenting a Job

This chapter discusses:

Best Practices Documenting a Job

Lesson Overview

At the highest level, there are typically three phases of project development when using the Talend Studio for Data Integration:

- Planning
- Development
- Documentation

At each phase, best practices should be utilized. The focus of this lesson is using best practices during the development phase that impact the resulting documentation.

The best practices addressed in this lab mostly have to do with naming and other descriptive tips while working in the design workspace.

It is also possible to generate Job-related documentation. The documentation contains the settings for all components, so it can be generated when needed (so it is automatically maintained). The documentation is extensible and accepts personalized comments and notes.

Disclaimer: Best practices discussed in this lesson are for individual developers using the Talend Studio to develop DI Jobs. It does not take into account enterprise-scale deployments with multiple developers who collaborate on large projects. That level of complexity is beyond scope and does suggest additional best practices. For detailed information and suggestions for such conventions please see the Talend Knowledge Base article titled *Best Practice: Conventions for Object Naming*.

Objectives

After completing this lesson, you will be able to:

- Name a Job and fill out its properties with best practice naming conventions
- Provide additional information for Job components
- View HTML documentation on your Job within the Studio
- Generate and then view HTML documentation external to the Studio

Next Step

First step is to document a Job from within the Talend Studio.

Using Best Practices While Documenting a Job

Overview

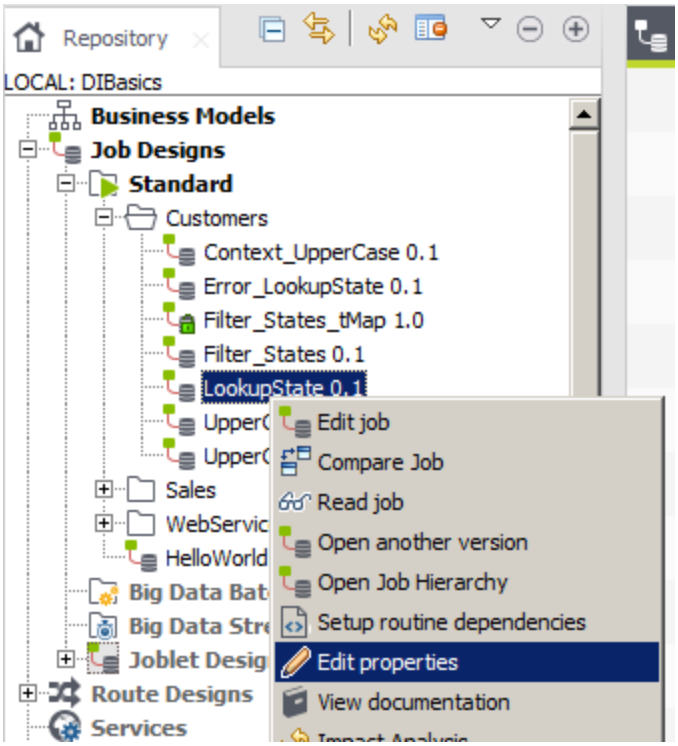
These lab exercises will cover important best practices related to the development phase as it pertains to documentation. Specifically, several guidelines with respect to Job properties, component names, and labels, as well as using notes in the design workspace are introduced. Once that is completed, you will progress to the documentation phase and generate and view HTML documentation for a Job you built earlier. As with any project, good documentation can be critical for other people to understand a Talend Job.

This section uses the **LookupState** Job as an example.

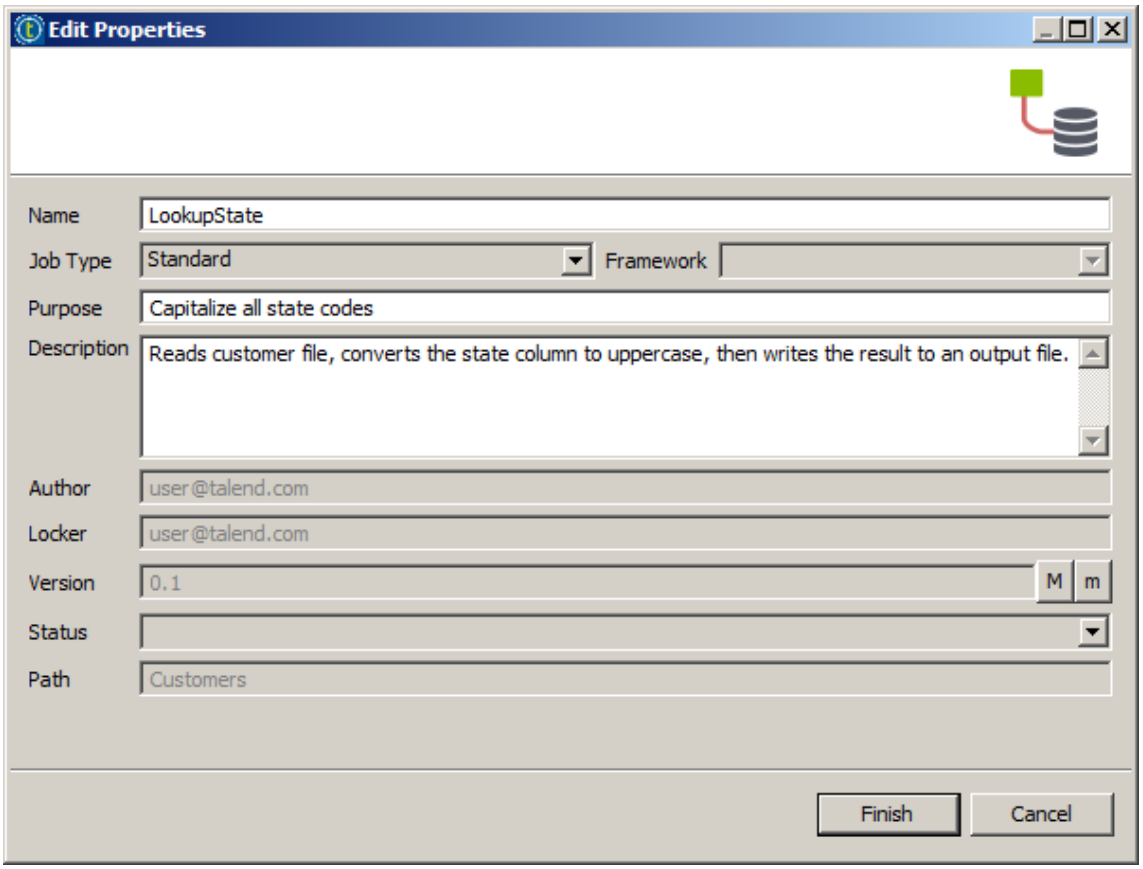
Improve Key Job Properties

1. PROVIDE CLEAR DESCRIPTIONS FOR JOBS

   Right-click **Repository > Job Designs > Standard > Customers > LookupState** and select **Edit Properties**.



   The **Edit Properties** window opens, which displays the information you supplied when you created the Job. Provide more concise text for **Purpose**, such as *Capitalize all state codes*.

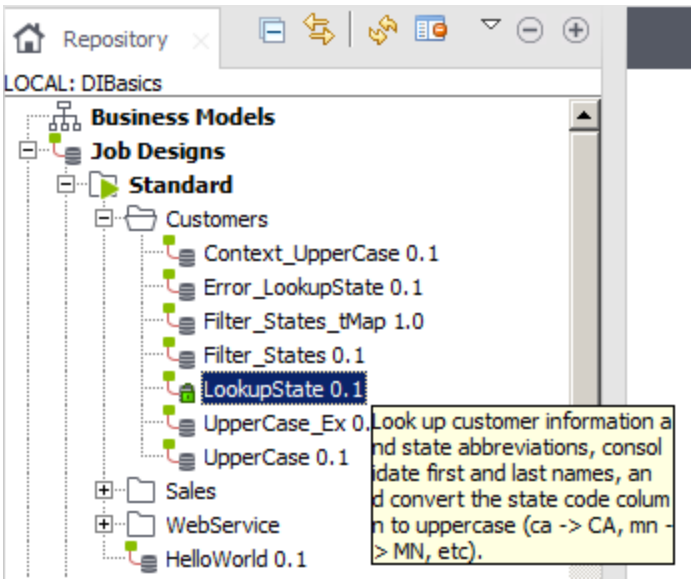


Similarly, enter something more helpful for the **Description** field, for example: *Look up customer information and state abbreviations, consolidate first and last names, and convert the state code column to upper case (ca -> CA, mn -> MN, etc)*.

Click **Finish** to save the changes to the Job properties.

2. VIEW THE JOB DESCRIPTION IN THE REPOSITORY

   Hover over the Job in the **Repository** and note the **Description** text is displayed.



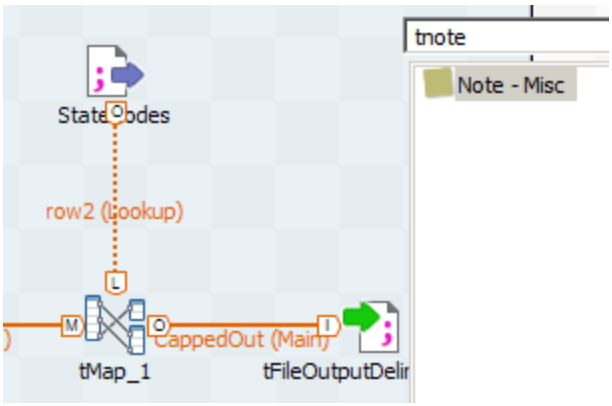
It is very handy to see a bit more detail than just the Job name, without having to open the Job.

Add a Note

1. ADD A NOTE TO THE JOB

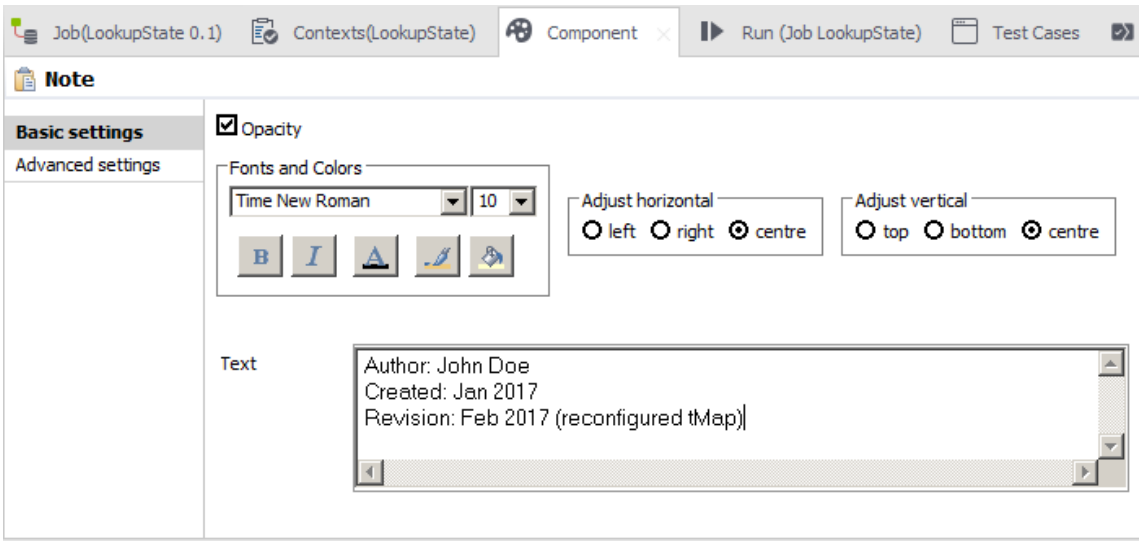
   Open the **LookupStates** Job if you have not already done so.

   Add a **tNote** component to the Job.



2. FORMAT THE NOTE

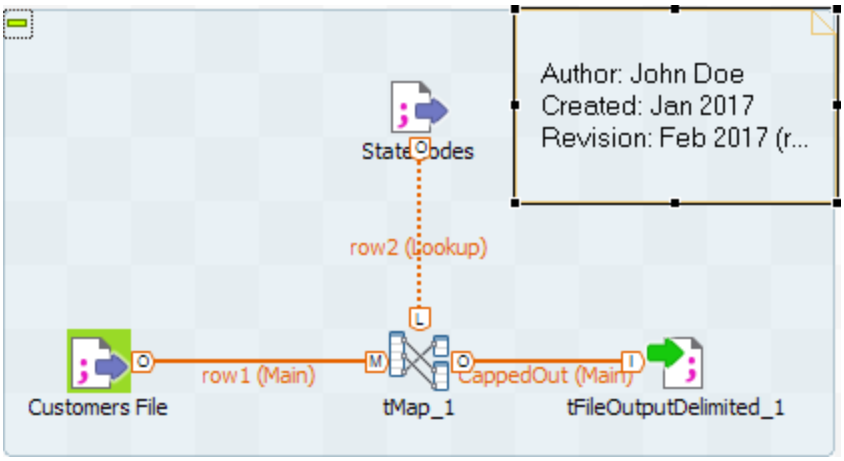
   Double-click the **tNote** component and in the **Component** view, enter some text into the **Text** field. Consider a high-level convention that includes basic information and formatting.



NOTE:
The text does not automatically wrap, so you may need to enter the text on multiple lines. There are several formatting options. If time permits, feel free to experiment with a few options for a couple of minutes.

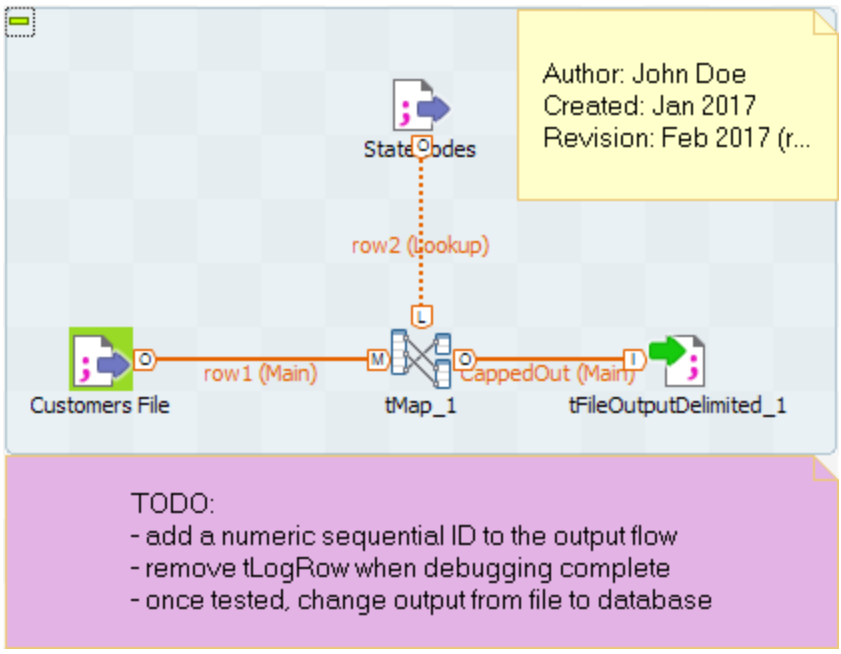
3. RESIZE AND POSITION THE NOTE

Position the **tNote** near the top-right of the Job components. Change the size and position as needed to accommodate the text you entered.



4. ADD A "TO DO" LIST

Place another **tNote** below the components to document outstanding items remaining to be completed. Incorporating a different fill color helps to differentiate the component.



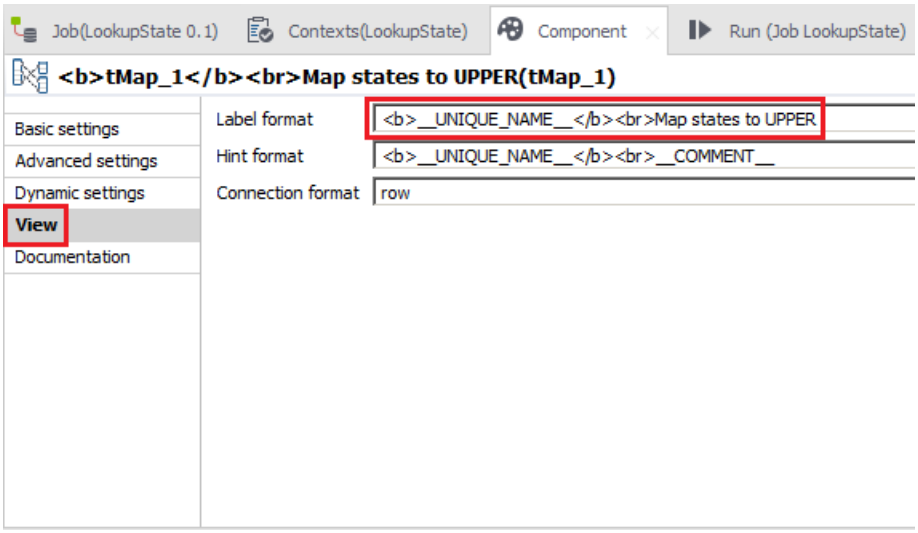
Document a Component

So far, you have placed components in the design workspace, left their default names alone, and provided little further information. This section will introduce you to several best practices surrounding component names and other helpful descriptive information.

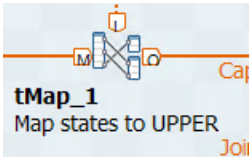
1. PROVIDE A MORE DESCRIPTIVE LABEL FOR A COMPONENT

   Open the **Component** view for **tMap_1**. Click the **View** tab and modify the default **Label Format** to something more descriptive, such as *__UNIQUE_NAME__
Map states to UPPER*.

   NOTE:
   __UNIQUE_NAME__ is a macro generated automatically by Talend Studio, and it indicates that the component will be assigned a unique name when it is created.



Although this text does not look good in the **Component** view, turn your attention to the **Designer**. You can see the component icon with its name in bold, and a brief reminder what it does. All of this information can be helpful, especially for new developers or others looking at your Job.

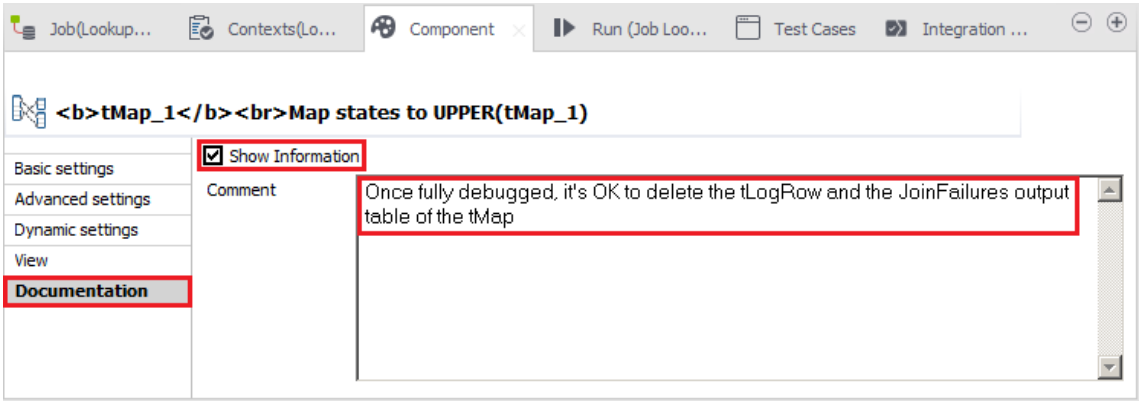


2. ADD A HINT TO A COMPONENT

   Change the **Hint format** to something more helpful such as: *Map state codes to upper and consolidate first/last names*.

   NOTE:
   The component does not reflect the change yet. The next step will enable the **Hint format** field.

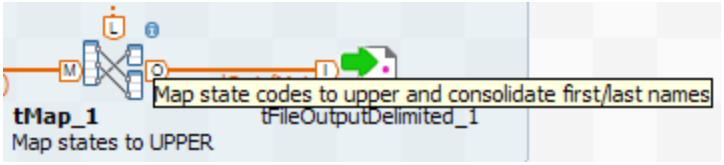
   Click the **Documentation** tab. Select the **Show Information** check box, and enter additional information into the **Comment** box.



   NOTE:
   This information will be seen when you generate and view the HTML documentation in the next section.

3. OBSERVE THE INFORMATION ICON NEXT TO THE COMPONENT

   Hover over the information icon (ⓘ) by the **tMap** component and notice that the hint text is displayed.



4. SIMILARLY DOCUMENT OTHER COMPONENTS

   If time permits, repeat these steps to document another component, such as an input or output component.

5. SAVE THE JOB

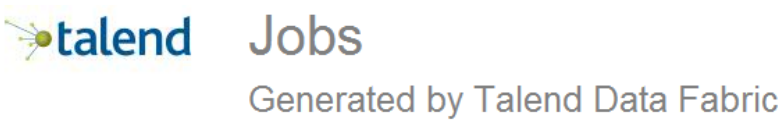
   When done, save the Job.

View and Generate HTML Documentation

Before generating the documentation and viewing it from the file system, you will view it directly from within the Studio. This is a nice feature for quickly checking what the HTML looks like during the development cycle.

1. VIEW HTML DOCUMENTATION FROM STUDIO

   Right-click **Repository > Job Designs > Standard > Customers > LookupState** and select **View documentation**. A new tab with an HTML documentation preview opens.

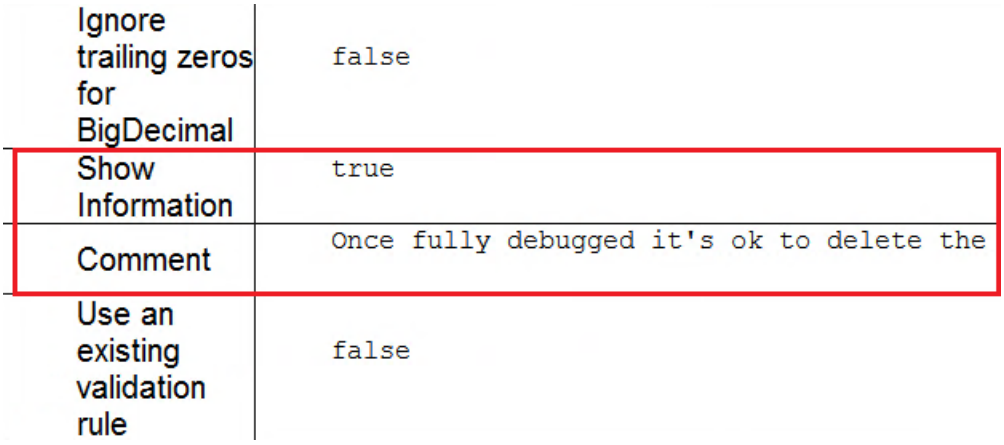


Project Name	DIBasics	GENERATION DATE	Dec 20, 2016 1:00:22 PM
AUTHOR	user@talend.com	Talend Data Fabric VERSION	6.3.0.20161026_1012-EP

Summary

2. EXAMINE THE COMPONENT LIST

   Click the **Component List** hyperlink, then select **tMap_1** from the list of components in your Job. Notice the detailed component parameters as you scroll down. Towards the bottom of the parameters you should see the **Comment** and its associated value, which matches the comment you added earlier in this exercise.

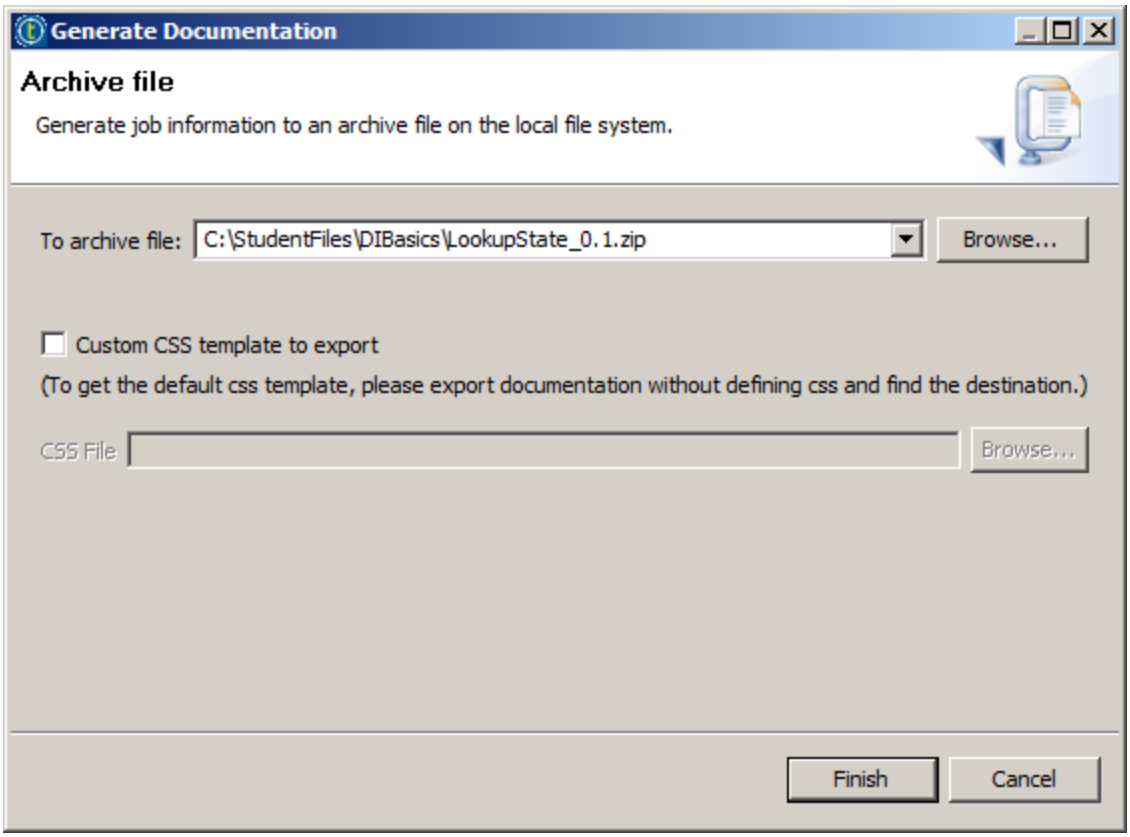


Ignore trailing zeros for BigDecimal	false
Show Information	true
Comment	Once fully debugged it's ok to delete the
Use an existing validation rule	false

   Now that viewing documentation from within the Studio looks correct, you can generate a more permanent version of the documentation.

3. GENERATE HTML OUTPUT FOR THE JOB DOCUMENTATION

   Right-click **Repository > Job Designs > Standard > Customers > LookupState** and select **Generate Doc As HTML**. Specify a directory of *C:\StudentFiles\DIBasics* (keep the file name as the default) and click **Finish**.



4. EXTRACT THE ARCHIVE AND EXAMINE THE CONTENTS

   Extract the archive, then locate and open *LookupState_0.1.html*. You will see that all of your comments are included, as well as component configuration settings, and a variety of other data.

   If needed, you could post or share the documentation with others.

Next

You have completed this lesson and now it is time to Wrap-Up.

Wrap-Up

In this lesson you learned several best practices for three typical phases of a project:

- Planning: this phase includes building a graphical business model and associating external documentation to it.
- Development: this phase includes several Job properties, such as the Name, Purpose, and Description. Further, you learned more about component labels, and tips on additional Design Workspace and Documentation information.
- Documentation: this last phase shows how easy it is to view HTML documentation within the Studio, and how to generate and view a more permanent (and sharable) set of HTML documentation from the local file system.

Next step

Congratulations! You have successfully completed this lesson. To save your progress, click **Check your status with this unit** below. To go to the next lesson, on the next screen, click **Completed. Let's continue >**.

APPENDIX

Additional Information

NOTE:
If the links below yield no search results, please make sure the language filter is set to English.

Talend Documentation:

- Talend Help Center

Lesson 01 - Getting Started

- Talend Studio Getting Started Guide
  - Getting Started Guide
- Component Reference Guide
  - tFixedFlowInput
  - tLogRow

Lesson 02 - Working with Files

- Talend Studio User Guide
  - Getting started with a basic Job
- Component Reference Guide
  - tFileInputDelimited
  - tMap
  - tFileOutputDelimited

Lesson 03 - Joining Data Sources

- Talend Studio User Guide
  - Centralizing File Delimited Metadata
  - Mapping data flows

Lesson 04 - Filtering Data

- Talend Studio User Guide
  - Mapping the Output setting

Lesson 05 - Using Context Variables

- Talend Studio User Guide
  - Using contexts and variables

Lesson 06 - Error Handling

- Talend Studio User Guide
  - Handling Job execution
- Component Reference Guide
  - tDie
  - tWarn

Lesson 07 - Generic Schemas

- Talend Studio User Guide
  - Setting up a generic schema
  - Managing Jobs
- Component Reference Guide
  - tRowGenerator

Lesson 08 - Working with Databases

- Talend Studio User Guide
  - Setting up a database connection
- Component Reference Guide
  - Database components

Lesson 09 - Creating Master Job

- Component Reference Guide
  - tRunJob

Lesson 10 - Working with Web Services

- Component Reference Guide
  - tESBConsumer
  - tXMLMap

Lesson 11 - Running Jobs Standalone

- Talend Studio User Guide
  - Importing/exporting items and building Jobs
  - Managing Job versions

Lesson 12- Best Practices and Documentation

- Talend Studio User Guide
  - How to generate HTML documentation

- Talend Knowledge Base article
  - Best Practice: Conventions for Object Naming

Intentionally blank