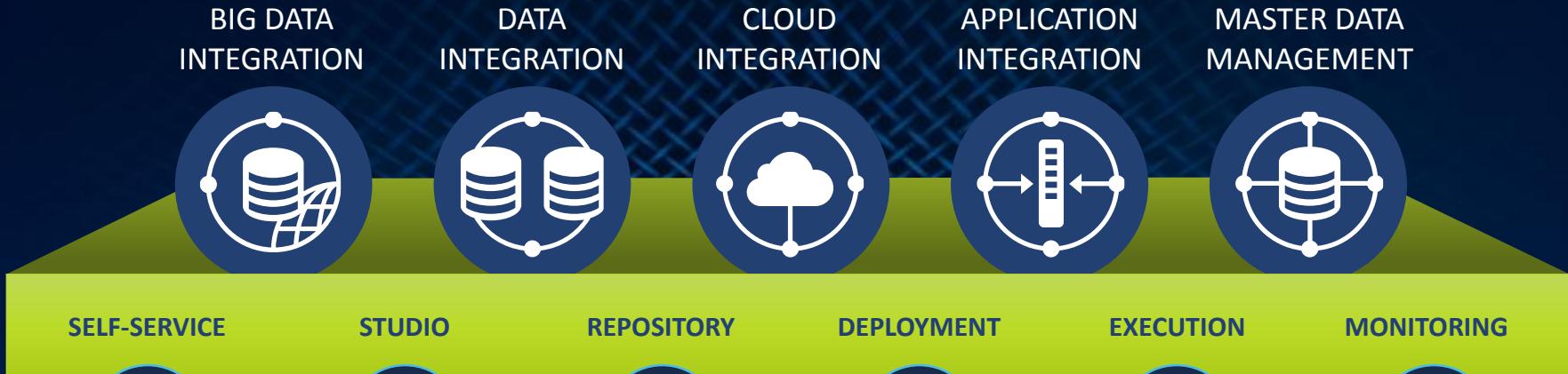




# Data Integration Basics

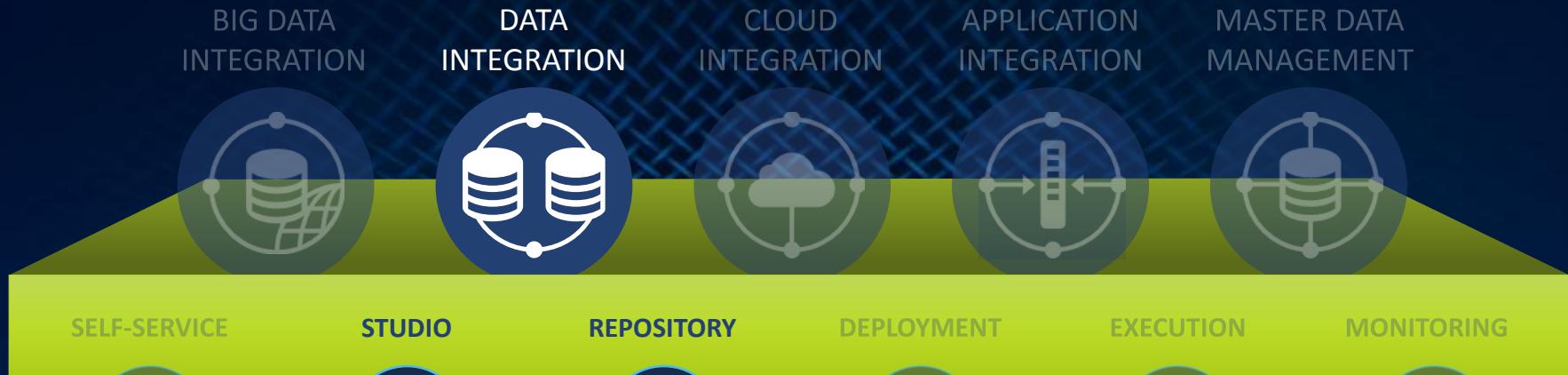
Version 6.3



-  1 Discovery and cleansing for business users
-  2 Comprehensive Eclipse-based user interface
-  3 Consolidated metadata and project information
-  4 Web-based deployment & scheduling
-  5 Same container for batch processing, message routing & services
-  6 Single web-based monitoring console



# This course focuses on:



1  
Discovery and  
cleansing for  
business users

2  
Comprehensive  
Eclipse-based  
user interface

3  
Consolidated  
metadata and  
project  
information

4  
Web-based  
deployment &  
scheduling

5  
Same container  
for batch processing,  
message routing &  
services

6  
Single web-  
based  
monitoring  
console

# Course Table of Contents

---



1. Getting Started
2. Working with Files
3. Joining Data Sources
4. Filtering Data
5. Using Context Variables
6. Error Handling
7. Generic Schemas
8. Working with Databases
9. Master Jobs
10. Working with Web Services
11. Running Jobs Standalone
12. Documenting a Job





# Introduction to Talend Data Integration

Version 6.3

# Objectives

---



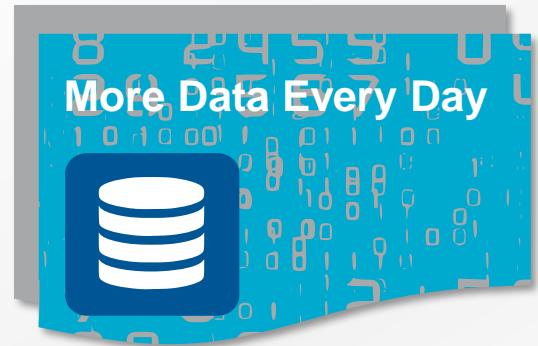
- Define Data Integration
- Explore typical project use cases
- Describe key benefits of Data Integration

# Why Data Integration?

---



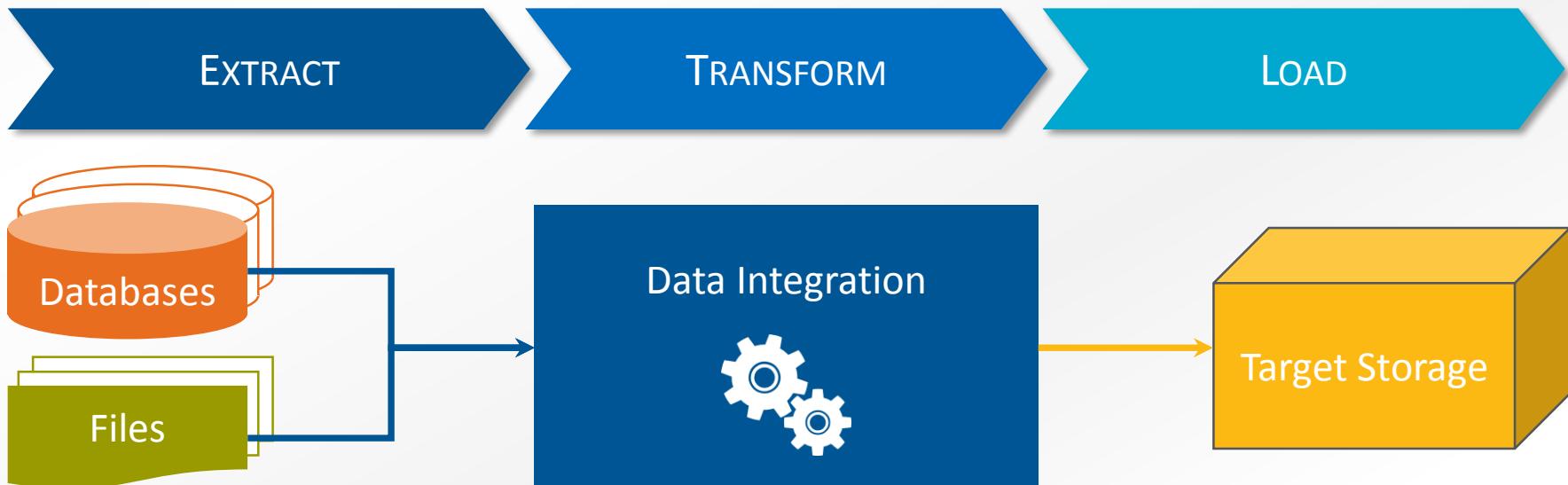
- Different formats
  - Different database structures and models
  - Different standards
  - Different systems
- 
- Removes mistakes and corrects data
  - Adjusts data from multiple sources to be used together
  - Structures and aggregates data to be used by BI tools



# What is Data Integration?



- Allows data coming from different systems to be merged
- Implementation is simplified by using an **ETL** tool



# Typical Use Cases

---



Data Migration



Data Warehousing



Data Consolidation



Data Synchronization



# Typical Use Cases



Data Migration



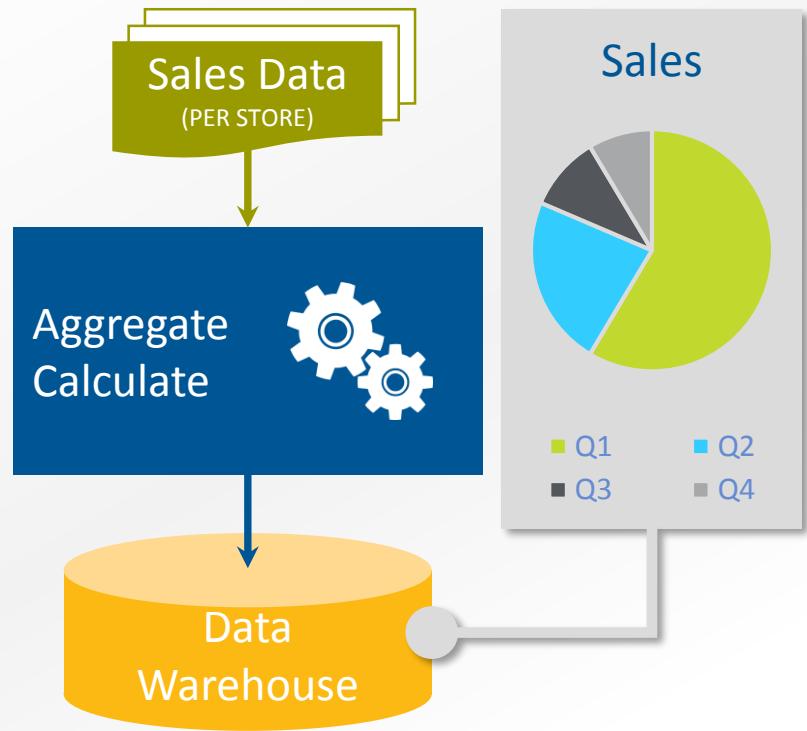
Data Warehousing



Data Consolidation



Data Synchronization



# Typical Use Cases



Data Migration



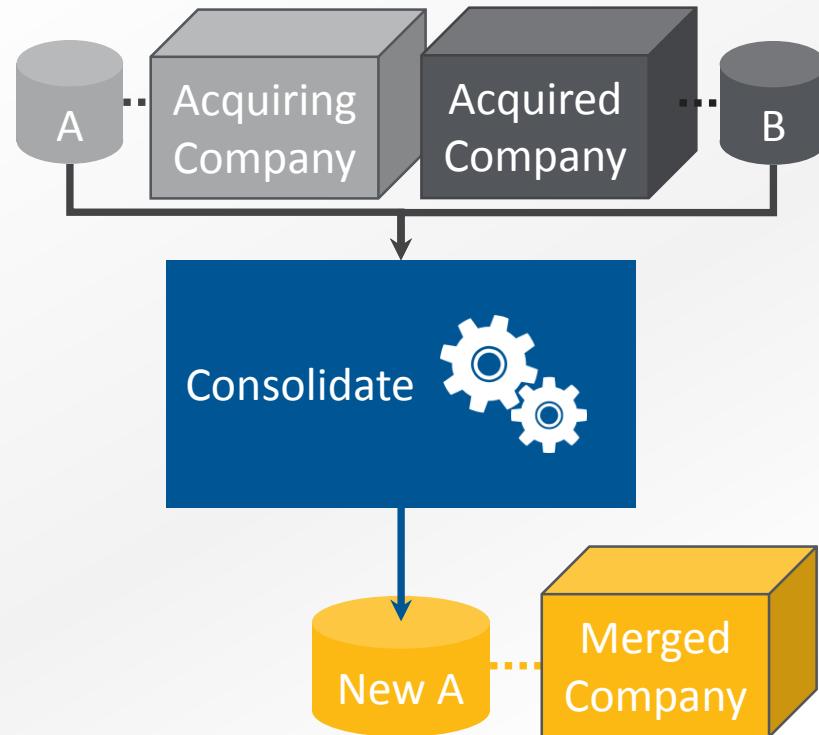
Data Warehousing



Data Consolidation



Data Synchronization



# Typical Use Cases



Data Migration



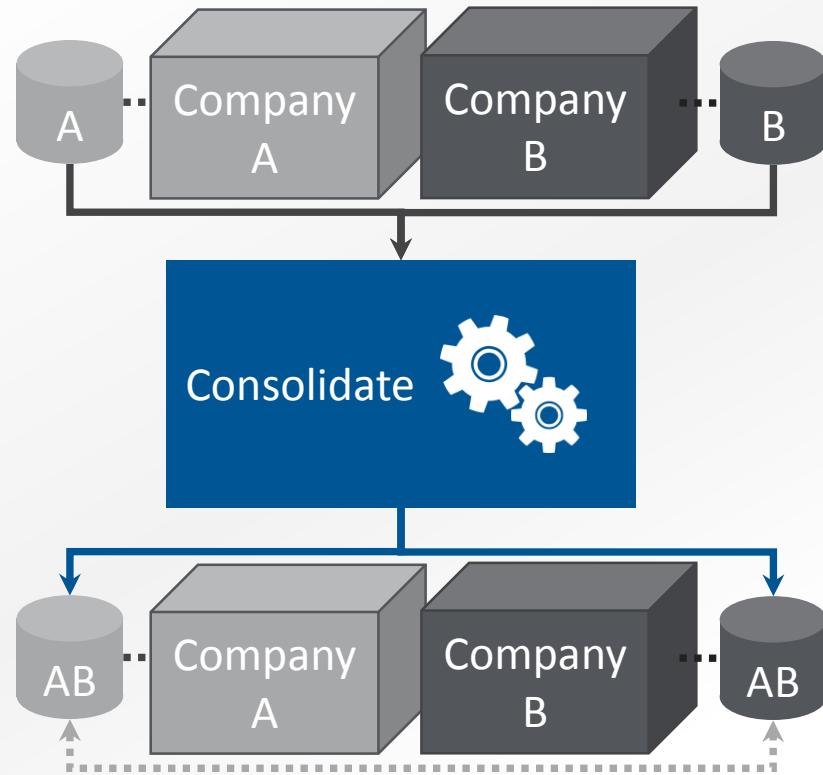
Data Warehousing



Data Consolidation



Data Synchronization

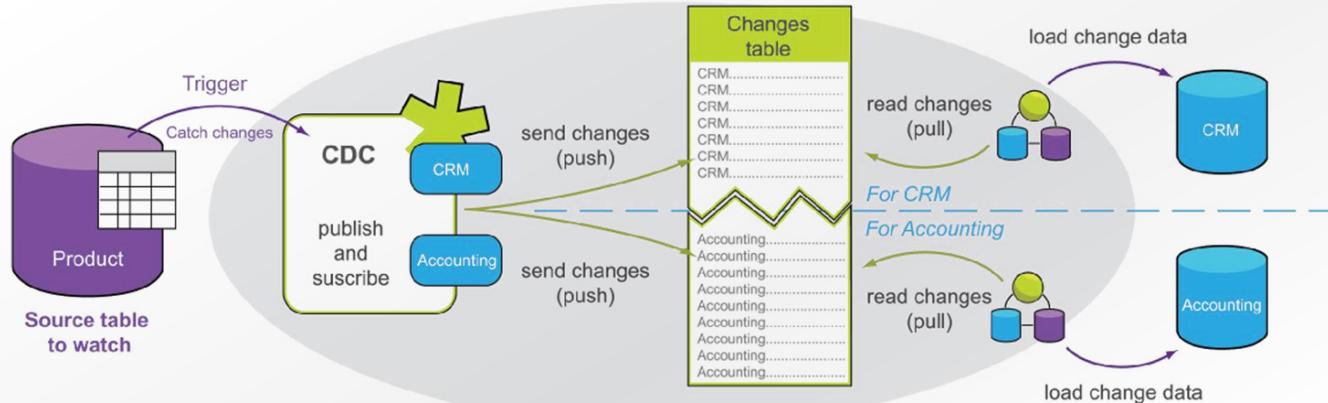


# Integration Using Delta Only



## Change Data Capture (CDC) Scenario

- The CDC is based on the database journal or triggers
- A **Changes Table** is used to track changes
- Employs a publish/subscribe mechanism



# Key benefits of Data Integration

---

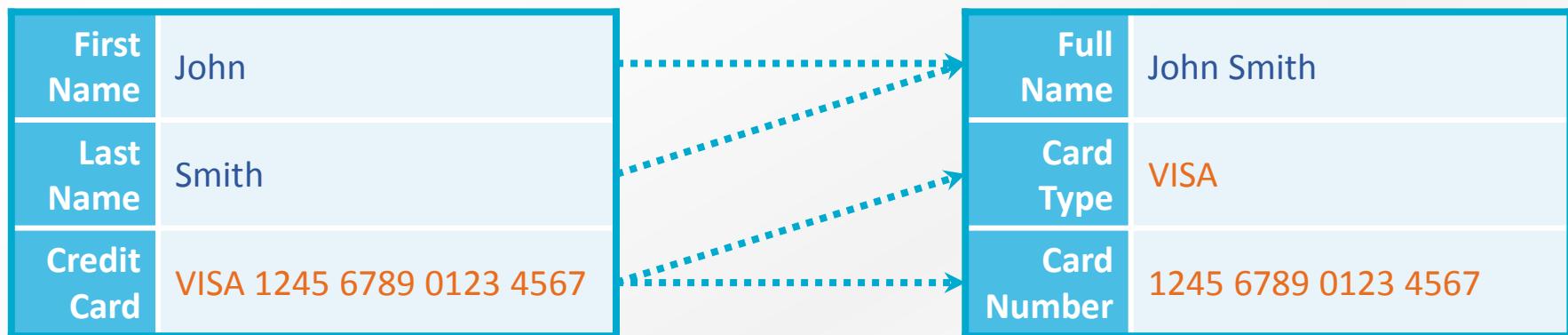


- Connectivity to source and target data systems
- Scalability and performance
- Transformation flexibility
- Data cleansing components
- Logging and exception handling
- Easy integration with web services

# Data Integration Power



- Splitting or merging fields
  - Hundreds of components to assist you



# Data Integration Power



- Data standardization
  - Dates
  - Units of measurement

Source Systems		
Product	Length	Data
Cable 1	5 m	25/01/2015
Product	Length	Data
Cable 2	38'6"	2015-16-02

Target System		
Product	Length	Data
Cable 1	5 m	25 Jan 2015
Cable 2	11.7 m	16 Feb 2015

# Data Integration Power



- Join data
  - Complete data with information from external sources



Target System

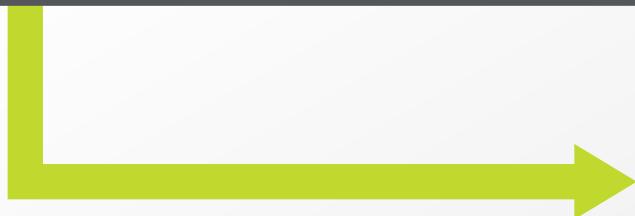
Product	Quantity	Zip Code	City	State	Customer
P167987	13	10004	New York	NY	C987674
P167987	16	10005	New York	NY	C987856

# Data Integration Power



- Aggregate data
  - Perform calculations
  - Group and order elements as needed

Product	Quantity	Zip Code	City	State	Customer
P167987	13	10004	New York	NY	C987674
P167987	16	10005	New York	NY	C987856



Product	Quantity	City
P167987	29	New York

# Data Integration Power



- Supports Data Deduplication in Data Quality and MDM projects
  - Several Data Quality components available



# Why Talend Data Integration?



## Respond Faster to Business Needs







# Getting Started

Talend Studio for Data Integration

# Objectives

---



- Start Talend Studio
- Create a new Job
- Find and add components to your Job
- Connect and configure components
- Get help with components
- Run a Job

# Talend Studio Main Window



## Repository

The screenshot shows the Talend Studio interface with the 'Repository' tab selected. On the left, the 'Business Models' tree view shows a 'Job Designs' node expanded, with a 'Standard' sub-node containing a 'Customers' folder and an 'UpperCase 0.1' job. Other nodes like 'Route Designs', 'Services', 'Contexts', 'Resources', 'Code', 'SQL Templates', 'Metadata', and 'Documentation' are also visible. The main workspace displays a 'Job Uppercase 0.1' job design with a 'Customer' component. A large callout box highlights the 'Customers' component with the text: 'The Repository organizes Jobs, metadata, context variables, and more.' Below the workspace, the 'Outline' and 'Code Viewer' panels are visible, showing code snippets for 'tFileInputDelimited\_1' and 'tMap\_1'. The right side of the interface features the 'Palette' with various component categories and a search bar, along with a sidebar listing 'Favorites' and other system components.

The Repository organizes Jobs, metadata, context variables, and more.

Business Models

- Job Designs
  - Standard
    - Customers
    - UpperCase 0.1
  - Big Data Batch
  - Big Data Streaming
  - Joblet Designs
- Route Designs
- Services
- Contexts
- Resources
- Code
- SQL Templates
- Metadata
- Documentation
- Recycle bin

Outline

Code Viewer

Customers File(tFileInputDelimited\_1)

Basic settings

Advanced settings

Dynamic settings

View

Documentation

Validation Rules

Property Type: Built-In

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

File name/Stream: C:/StudentFiles/DIBasics/Custs.csv

Row Separator: \n

Field Separator: ,

Header: 1

Footer: 0

Limit:

Integration Profiling MDM

Palette

Find component...

Favorites

Recently Used

Big Data

Business Intelligence

Business

Cloud

Custom Code

Data Quality

Databases

DotNET

ELT

ESB

File

Internet

Logs & Errors

Misc

Orchestration

Processing

System

Talend Cloud

Talend Data Preparation

Talend Data Stewardship

Talend MDM

Unstructured

XML

# Talend Studio Main Window



## Designer

The screenshot shows the Talend Studio Designer interface. On the left, the Navigator pane displays a project structure under LOCAL: DIBasics, including Business Models, Job Designs (Standard, Customers, Uppercase 0.1), Big Data Batch, Big Data Streaming, Joblet Designs, Route Designs, Services, Contexts, Resources, and Code. The 'Customers' folder under 'Job Designs' is selected. In the center, the 'Job Uppercase 0.1' window shows a graphical job flow: 'Customers File' → 'row1 (Main)' → 'tMap\_1' → 'CappedOut (Main)' → 'tFileOutputDelimited\_1'. Below this window, tabs for 'Designer', 'Code', and 'Jobscript' are visible. At the bottom, a detailed configuration panel for the 'Customers File(tFileInputDelimited\_1)' component is shown, with tabs for 'Basic settings', 'Advanced settings', 'Dynamic settings', 'View', 'Documentation', and 'Validation Rules'. The 'Basic settings' tab shows the file name as 'C:/StudentFiles/DIBasics/Custs.csv', row separator as '\n', field separator as ',', header as 1, footer as 0, and limit as 0. A note states: 'When the input source is a stream or a zip file, footer and random shouldn't be bigger than 0.' To the right of the Designer window, the Palette pane lists various component categories: Favorites, Recently Used, Big Data, Business Intelligence, Business, Cloud, Custom Code, Data Quality, Databases, DotNET, ELT, ESB, File, Internet, Logs & Errors, Misc, Orchestration, Processing, System, Talend Cloud, Talend Data Preparation, Talend Data Stewardship, Talend MDM, Unstructured, and XML. The top menu bar includes File, Edit, View, Window, Help, and a toolbar with various icons.

The Designer displays the Job in graphical mode.

# Talend Studio Main Window



## Palette

The screenshot shows the Talend Studio interface. On the left is the Repository browser, displaying a tree structure of Business Models, Job Designs, Route Designs, Services, Contexts, Resources, and various Talend components like Big Data Batch, Big Data Streaming, Joblet Designs, and Route Designs. Below the Repository is the Outline view, which lists the components used in the current job: tfileInputDelimited\_1 (Customers File), tfileOutputDelimited\_1, and tMap\_1. The central area is the Designer, where a job named "Job Uppercase 0.1" is being built. A flowchart shows data moving from a "Customers File" component through a "tMap\_1" component to an "tfileOutputDelimited\_1" component. The right side of the interface features the Palette, a catalog of Talend components organized into categories such as Favorites, Recently Used, Big Data, Business Intelligence, Business, Cloud, Custom Code, Data Quality, Databases, DotNET, ELT, ESB, Internet, Logs & Errors, Misc, Orchestration, Processing, System, Talend Cloud, Talend Data Preparation, Talend Data Stewardship, Talend MDM, Unstructured, and XML. A large callout box with a blue border and a white background is overlaid on the Designer area, containing the text: "You can drag and drop components from the Palette to the Designer." A blue circle highlights the "Favorites" section in the Palette.

You can drag and drop components from the **Palette** to the Designer.

# Talend Studio Main Window



## Component View

The screenshot shows the Talend Studio interface with the 'Component' view highlighted. A large blue callout box contains the text: 'The Component view displays configurable details about the selected components.' A white circle highlights the 'Component' tab in the top navigation bar. The central panel shows the configuration details for the 'Customers File(tFileInputDelimited\_1)' component.

The Component view displays configurable details about the selected components.

**Component View Details:**

- Property Type:** Built-In
- Description:** "When the input source is a stream or a zip file, footer and random shouldn't be bigger than 0."
- File name/Stream:** C:/StudentFiles/DIBasics/Custs.csv
- Row Separator:** \n
- Field Separator:** ,
- Header:** 1
- Footer:** 0
- Limit:** (empty)

# Talend Studio Main Window



## Run View

The screenshot shows the Talend Studio interface with the 'Run View' selected. A large callout box highlights the central workspace where a job named 'Job Uppercase' is displayed. Below the workspace, a 'Basic Run' panel is open, featuring buttons for 'Run', 'Kill', and 'Clear', along with a 'Line limit' input field set to 100 and a 'Wrap' checkbox. To the right of the workspace, a 'Default' context menu is open, listing various options like 'Name'. The left sidebar shows the 'Repository' with a tree view of 'Business Models', 'Job Designs', 'Route Designs', 'Services', etc., and a list of components below it. The top menu bar includes 'File', 'Edit', 'View', 'Window', and 'Help', along with links for 'Learn', 'Ask', 'Exchange', and social media integration. The right sidebar contains a 'Palette' with a search bar and a list of categories such as 'Favorites', 'Recently Used', and 'Integration'.

Use the Run view to execute Jobs and monitor output.

# Talend Studio Main Window



## Run View

A screenshot of the Talend Studio interface in 'Run' mode. The main area shows a 'Job UpperCase 0.1' diagram with various components like tFileInputDelimited and tMap. A large blue callout box highlights the top toolbar, which includes icons for Run, Kill, and Clear, along with dropdown menus for Debug Run, Advanced settings, Target Exec, and Memory Run. To the right is a palette with component search, favorites, and a list of perspectives: Integration, Profiling, MDM, Business Intelligence, Business, Cloud, Custom Code, Data Quality, Databases, DotNET, ELT, ESB, File, Internet, Logs &amp; Errors, Misc, Orchestration, Processing, System, Talend Cloud, Talend Data Preparation, Talend Data Stewardship, Talend MDM, Unstructured, and XML. The left sidebar shows a project tree under 'LOCAL: DIBasics' with nodes like Business Models, Job Designs, Route Designs, Services, Contexts, Resources, and Documentation. The bottom left shows an 'Outline' view with items like tfileInputDelimited\_1 and tMap\_1.

The tool bars provide:

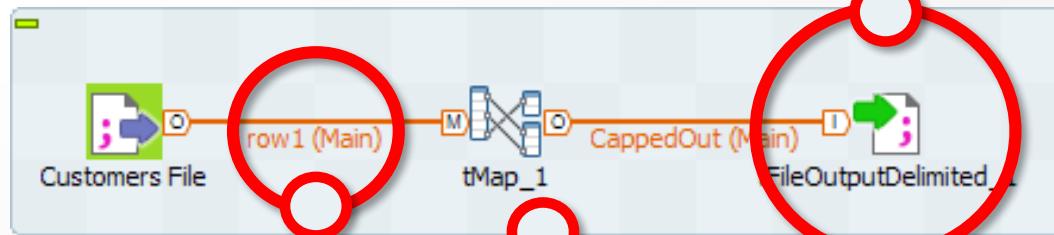
- Shortcuts (run a Job, create a Job, and more)
- Perspective switcher
- Menu items

# What Is a Talend Job?



A **component** is the executable part of a Job.

Talend provides hundreds of ready-to-use components to accomplish specific tasks.



Components are connected with **rows** (links that carry data) or **triggers** (links that transfer control).

A **Job** consists of one or more connected components that generates Java code.

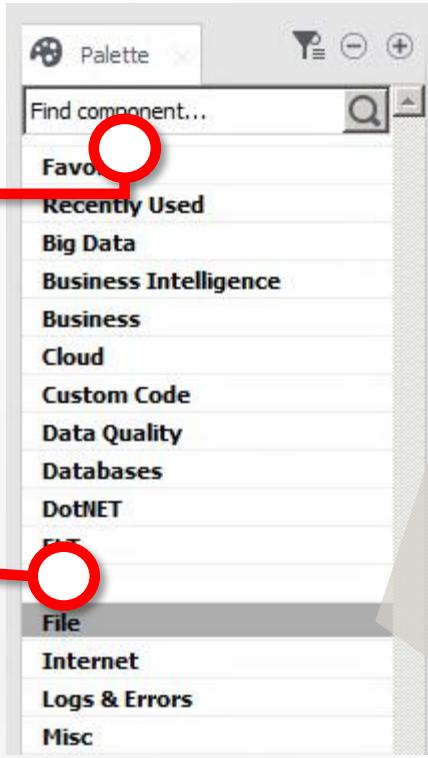
# Components



Components can be found in the **Palette** by name or keyword

Components are organized by family. For example:

- File
- Databases
- Processing
- Orchestration
- And many more

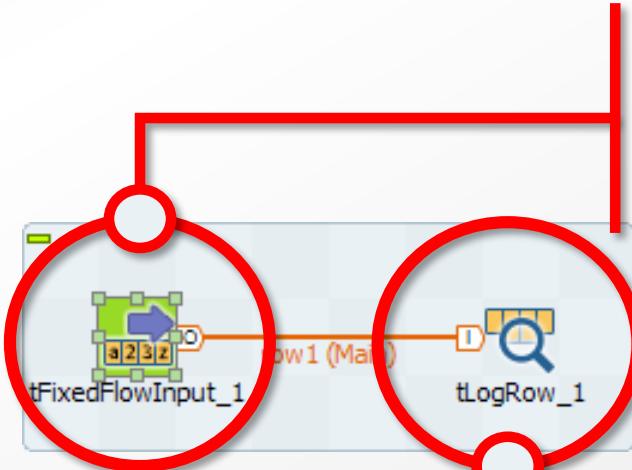


File
Hadoop
Input
tApacheLogInput
tFileInputARFF
tFileInputDelimited
tFileInputExcel
tFileInputFullRow
tFileInputJSON
tFileInputLDIF
tFileInputMail
tFileInputMSDelimited
tFileInputMSPositional
tFileInputMSXML
tFileInputPositional
tFileInputProperties
tFileInputRaw
tFileInputRegex
tFileInputXML
Management
NamedPipe
Output
Sqoop

# Build a Job



## Example Hello World Job



A **tFixedFlowInput** component is used to define a message to pass on to another component.

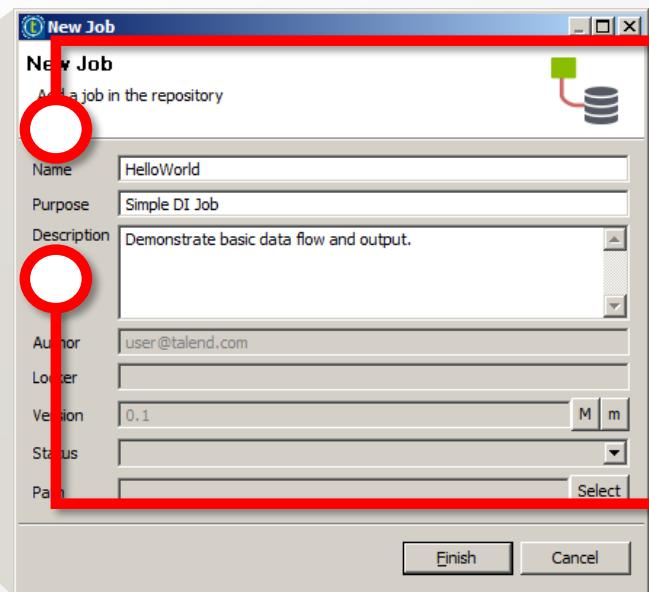
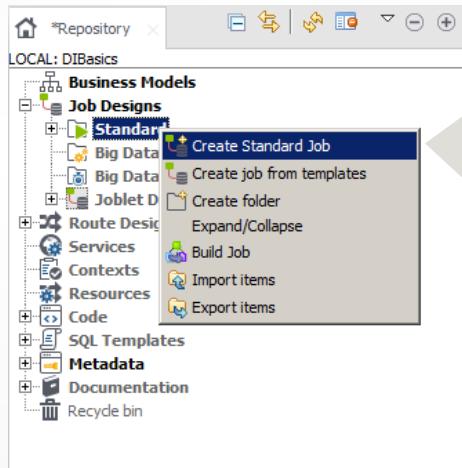
A **tLogRow** component is used to display messages in the Job execution console.

# Build a Job



## Create a Job

- Right-click Repository > Job Designs > Standard
- Select Create Standard Job



Name cannot contain spaces or special characters.

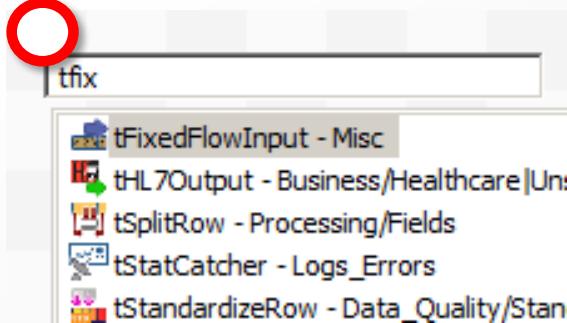
Fill out both Purpose and Description.

# Build a Job

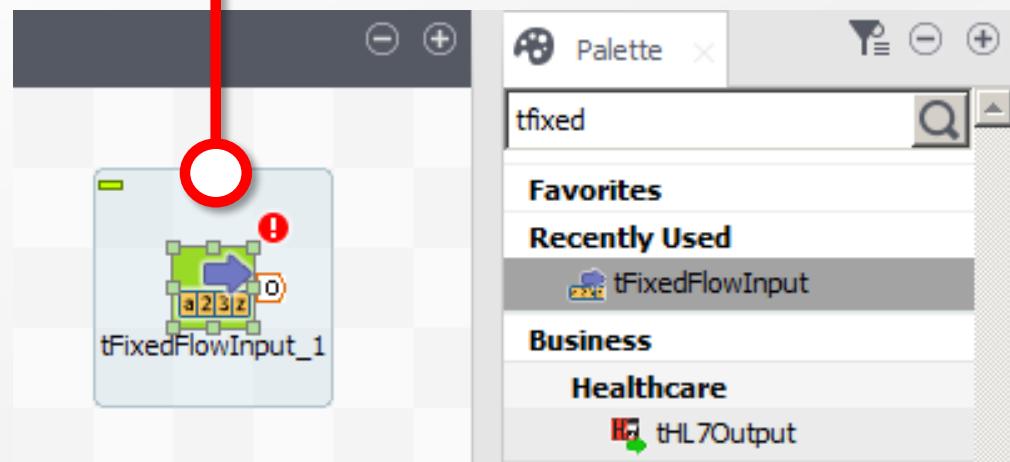


Options for adding components

Click and begin typing component name in **Designer**



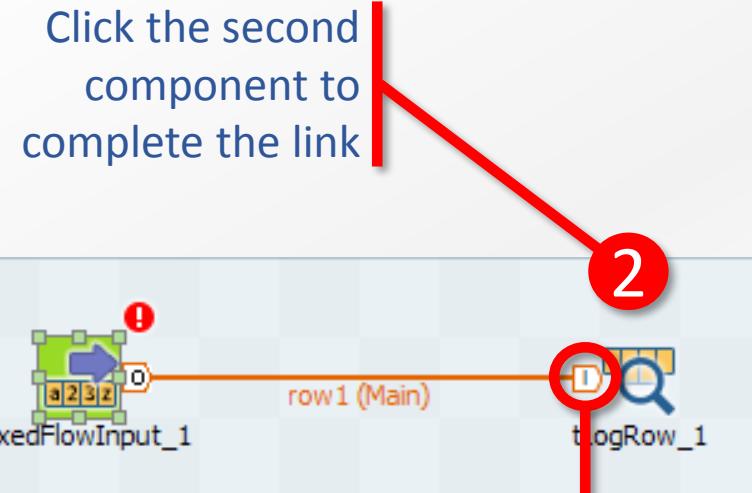
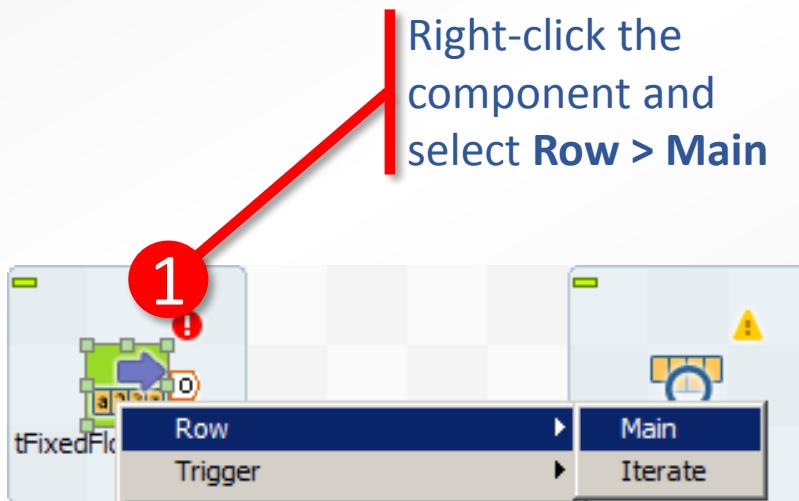
Drag and drop components from **Palette**



# Build a Job



## Connecting components



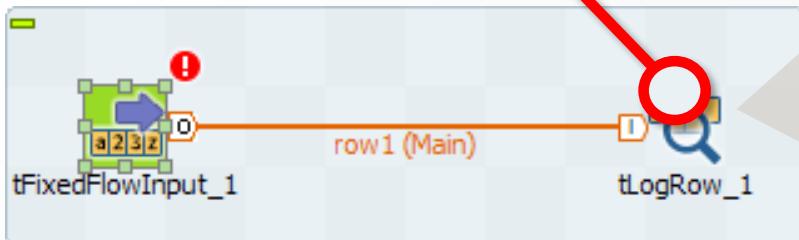
The arrow indicates the direction of the data flow in the link.

# Build a Job



## Online help

Select the component and press F1



Help x Contents Search Related Topics Bookmarks Index

Talend Components Reference Guide > Logs & Errors components

tLogRow

Chapter 16. Logs & Errors components

Prev Next

### tLogRow

Function	Displays data or results in the Run console.
Purpose	tLogRow is used to monitor data processed.

Depending on the *Talend* solution you are using, this component can be used in one, some or all of the following Job frameworks:

- Standard: see [tLogRow properties](#).

The component in this framework is generally available.

# Build a Job



## Rename a component

The screenshot shows the Talend Data Fabric interface with the following steps:

- Select the component you want to rename**: A red arrow points from the text to the 'tFixedFlowInput\_1' component in the main canvas.
- Select the View tab in the Component view**: A red arrow points from the text to the 'View' tab in the component's configuration panel.
- Specify new name in Label format box**: A red arrow points from the text to the 'Label format' input field in the configuration panel, which contains the value 'InputMessage'.

**Talend Data Fabric** interface details:

- Toolbar**: Includes standard icons for file operations, search, and navigation.
- Left Sidebar**: Shows project structure with 'HelloWorld 0.1' selected, and categories like LOCA, Route Designs, and Services.
- Right Sidebar**: Palettes for Integration, Profiling, and MDM, and a large 'fixed' palette containing various components.
- Bottom Panel**: Shows tabs for Designer, Code, and Jobscript, and a component configuration panel for 'InputMessage(tFixedFlowInput\_1)'.

# Build a Job



## Configure a component

Select the component you want to configure

Open the Component view

Configure component-specific settings

The screenshot shows the Talend Data Fabric interface with the following components visible:

- Job Hello World 0.1**: The main job definition window.
- InputMessage**: A component in the flow.
- tLogRow**: A component connected to the flow.
- Schema of InputMessage**: A dialog box showing the schema for the InputMessage component, with one column named "Message" of type String.
- InputMessage(tFixedFlowInput\_1)**: The component configuration dialog, showing basic settings like Schema (Built-In), Number of rows (1), and Mode (Use Single Table).

Red numbered callouts point to specific areas:

- Pointing to the **InputMessage** component in the job flow.
- Pointing to the **Schema of InputMessage** dialog.
- Pointing to the **InputMessage(tFixedFlowInput\_1)** configuration dialog.

# Run a Job



## Options for running a Job

Use the button in the quick access tool bar

Use the button in the Run view

Console output appears here

The screenshot shows the Talend Data Fabric 6.3.1 interface. On the left, there's a sidebar with options like 'HelloWorld 0.1', 'Big Data Batch', 'Big Data Streaming', 'Joblet Designs', 'Route Designs', and 'Services'. The main workspace displays a job design with two components: 'InputMessage' and 'tLogRow\_1' connected by a flow labeled 'row1 (Main)'. Below the workspace is a toolbar with various icons. A red circle highlights the 'Run' icon in the toolbar. Another red circle highlights the 'Run' button in the 'Basic Run' section of the 'Job HelloWorld' panel. A third red circle highlights the 'Run' button in the 'Execute' section of the same panel. The 'Code Viewer' tab is selected in the bottom-left corner. The 'Integration' tab is selected in the top-right palette. The 'Run' panel shows the console output:

```
[statistics] connecting to socket on port 3431  
[statistics] connected  
Hello world  
[statistics] disconnected  
Job HelloWorld ended at 11:07 16/02/2017. [exit code=0]
```

The 'Run' panel also includes settings for 'Line limit' (set to 100) and 'Wrap'.

# Lab Overview



Discover Talend Studio, build and run your first Job

- Start Talend Studio
- Build and run the following Job:



# Lesson Summary

---



- Talend Studio main window
  - Perspectives
- Views
  - **Designer**
  - **Palette**
  - **Repository**
  - **Run, Component**, and more
- Build a Job
  - Add and link components
  - Configure components
  - Run a Job
  - Use the online help system





# Working with Files

# Objectives

---

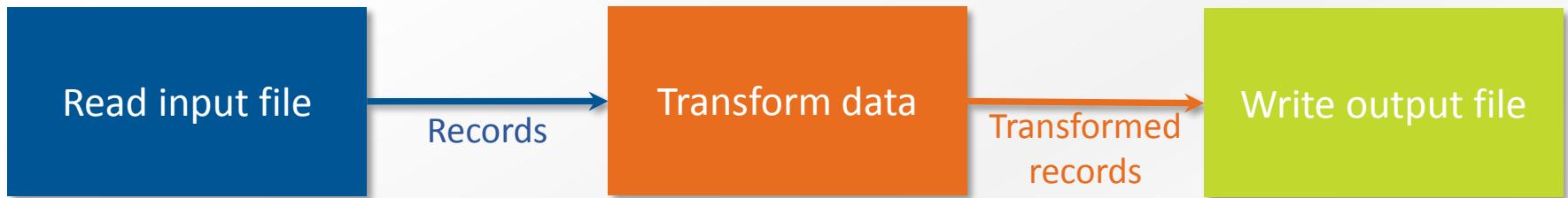


- Read and write delimited text files
- Read warnings and error messages
- Transform data using the **tMap** component
- Build simple expressions
- Explore data using the **Data Viewer**
- Duplicate a Job

# Use Case



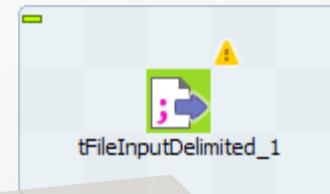
- Apply basic transformations to an input file
  - Transform a column to uppercase
  - Concatenate first name and last name into full name



# Read a Delimited File



- Use a **tFileInputDelimited** component
  - Configure file name
  - Check field separator
  - Specify number of header lines



Screenshot of the Talend interface showing the configuration of a tFileInputDelimited component.

The title bar shows the job name: "Job(UpperCase ...)" and the component type: "Component".

The component configuration window is open for "Customers File(tFileInputDelimited\_1)".

The left sidebar lists settings: Basic settings, Advanced settings, Dynamic settings, View, Documentation, and Validation Rules. The "Basic settings" tab is selected.

The main configuration area includes:

- Property Type:** Built-In
- Description:** "When the input source is a stream or a zip file, footer and random shouldn't be bigger than 0."
- File name/Stream:** "C:/StudentFiles/DIBasics/Custs.csv"
- Row Separator:** "\n"
- Field Separator:** ";" (highlighted with a red box)
- CSV options:**  CSV options (unchecked)
- Header:** 1 (highlighted with a red box)
- Footer:** 0
- Limit:** (empty field)

# File Delimited Schema



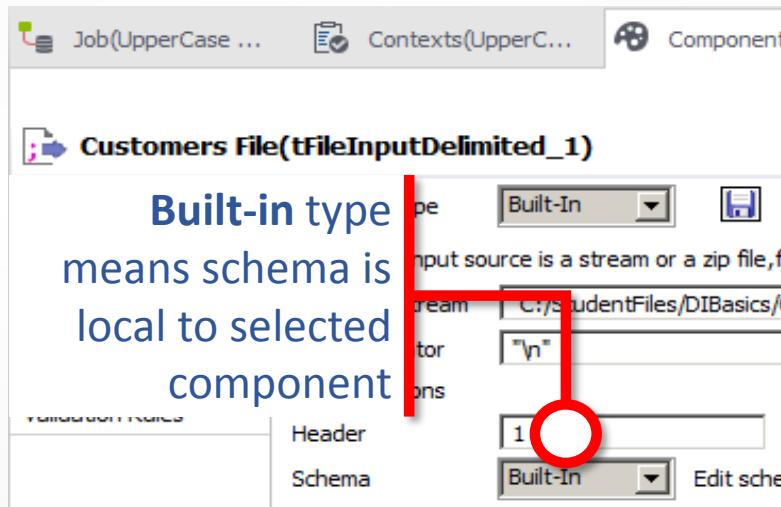
- The schema defines the structure of the file

Job(UpperCase ...    Contexts(UpperC...    Component

**Customers File(tFileInputDelimited\_1)**

Type: Built-In  
Input source is a stream or a zip file, f...  
Stream: C:/studentFiles/DIBasics/...  
Separator: "\n"  
Field Separator: ":"  
Header: 1  
Footer: 0  
Limit:   
Validation Rules  
Header  
Schema

Built-in type means schema is local to selected component

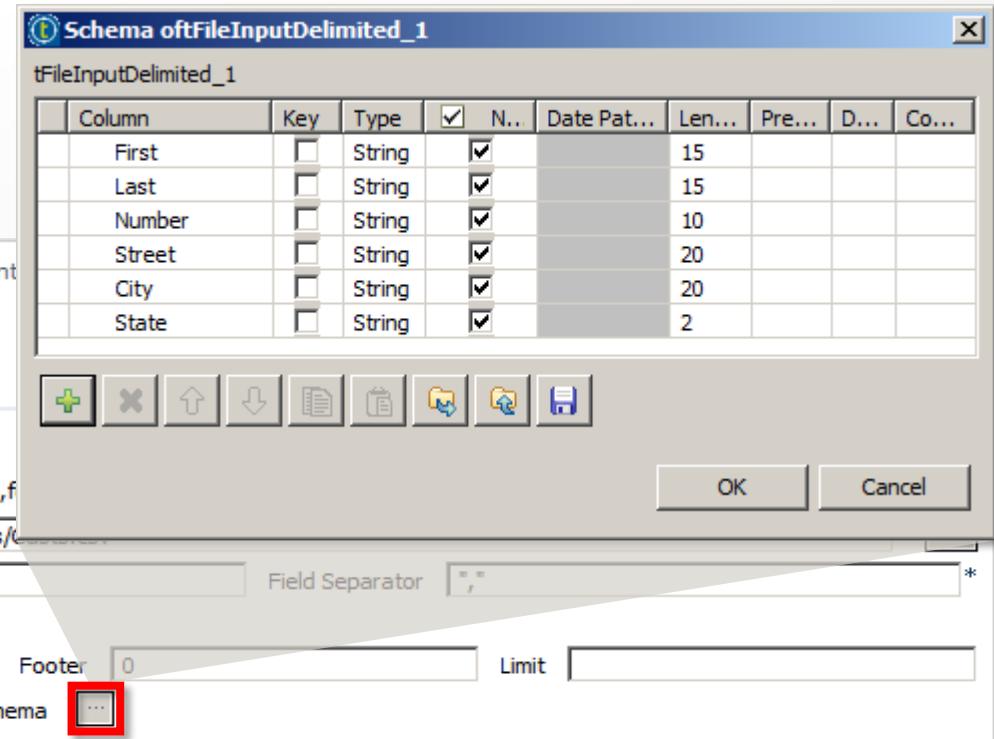


**Schema oftFileInputDelimited\_1**

tFileInputDelimited\_1

Column	Key	Type	N...	Date Pat...	Len...	Pre...	D...	Co...
First	<input type="checkbox"/>	String	<input checked="" type="checkbox"/>		15			
Last	<input type="checkbox"/>	String	<input checked="" type="checkbox"/>		15			
Number	<input type="checkbox"/>	String	<input checked="" type="checkbox"/>		10			
Street	<input type="checkbox"/>	String	<input checked="" type="checkbox"/>		20			
City	<input type="checkbox"/>	String	<input checked="" type="checkbox"/>		20			
State	<input type="checkbox"/>	String	<input checked="" type="checkbox"/>		2			

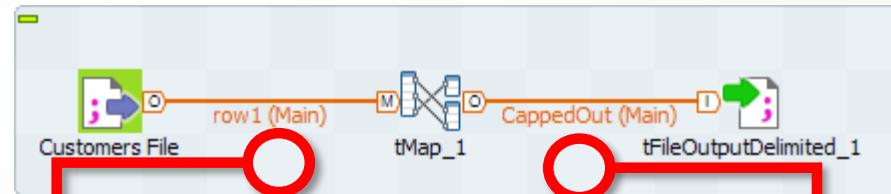
**Buttons:** +, X, Up, Down, File, Save, Cancel, OK



# Transforming Data



- Use a tMap component



**tMap\_1 Configuration (row1)**

Column
First
Last
Number
Street
City
State

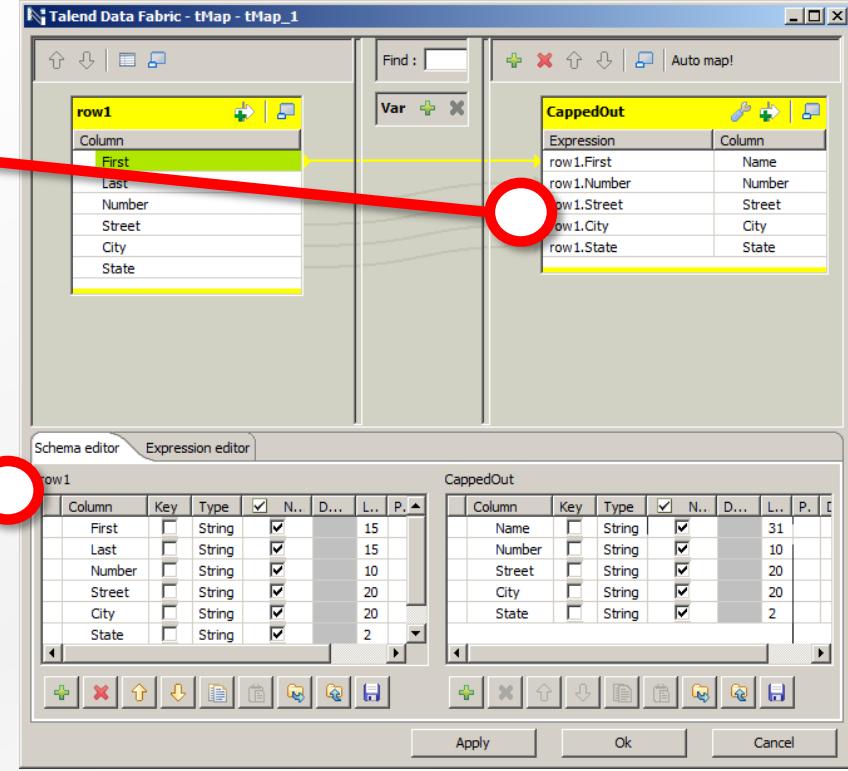
**tMap\_1 Configuration (CappedOut)**

Expression	Column
row1.First + " " + row1...	Name
row1.Number	Number
row1.Street	Street
row1.City	City
StringHandling.UPCASE...	State

# Transforming Data



Mapping editor allows you to drag and drop input columns to the output tables



Schema editor allows you to manage schemas of selected tables (add columns, change data types, reorder columns, and more)

# Transforming Data



- Use predefined functions to build expressions

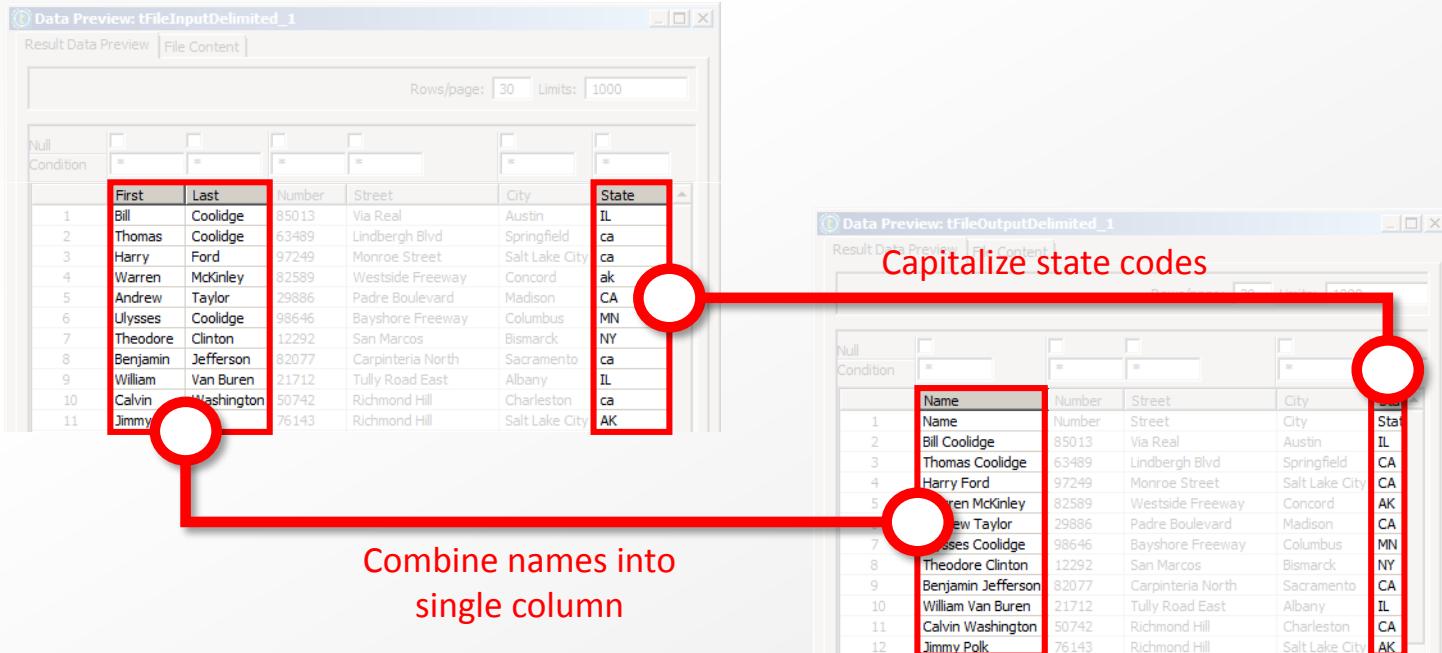
The screenshot shows the Expression Builder dialog box. The 'Expression' field contains the code 'StringHandling.UPCASE(row1.State)'. A red circle highlights the first part of the expression 'StringHandling.UPCASE('. Below the dialog, a portion of the 'CappedOut' table is visible, showing columns for 'row1.First', 'row1.Number', 'row1.Street', 'row1.City', and 'row1.State'. A red box highlights the '...' button in the 'Expression' column for 'row1.State'. A red callout box points from the 'row1.State' entry to the expression in the dialog, containing the text 'Convert row1.State to uppercase'.

- Use operators with input columns to build expressions

Expression	Column
row1.First + " " + row1.Last	Name
row1.Number	Number
row1.Street	Street
row1.City	City
StringHandling.UPCASE(row1.State)	State

Combine *row1.First* and *row1.Last* into one column

# Transforming Data



# Write to a Delimited File



- Use a **tFileOutputDelimited** component
  - Configure file name
  - Check field separator



Screenshot of the Talend Studio interface showing the configuration of the tFileOutputDelimited component.

The component is named **tFileOutputDelimited\_1**.

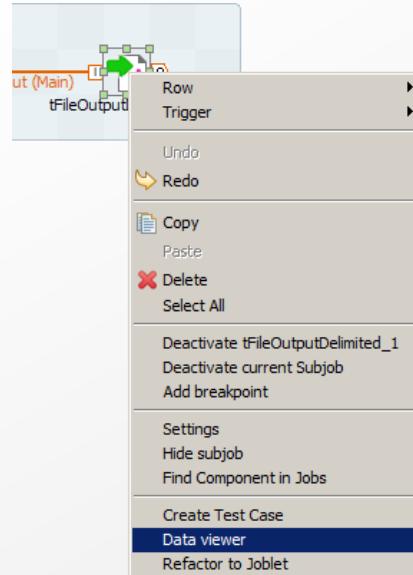
**Basic settings** tab selected:

- Property Type: Built-In
- Use Output Stream
- File Name:** "C:/StudentFiles/DIBasics/CappedOut.csv" (highlighted with a red box)
- Row Separator:** "\n"
- Field Separator:** ";" (highlighted with a red box)
- Append
- Include Header
- Compress as zip file
- Schema: Built-In
- 
-

# Data Viewer



- Quick access to data within Talend Studio
- Available for input and output components



Data Preview: tFileOutputDelimited\_1

Result Data Preview | File Content |

Rows/page: 30 Limits: 1000

Null Condition	*	*	*	*	*
Name	Number	Street	City	Stat	▲
1	Name	Number	Street	City	Stat
2	Bill Coolidge	85013	Via Real	Austin	IL
3	Thomas Coolidge	63489	Lindbergh Blvd	Springfield	CA
4	Harry Ford	97249	Monroe Street	Salt Lake City	CA
5	Warren McKinley	82589	Westside Freeway	Concord	AK
6	Andrew Taylor	29886	Padre Boulevard	Madison	CA
7	Ulysses Coolidge	98646	Bayshore Freeway	Columbus	MN
8	Theodore Clinton	12292	San Marcos	Bismarck	NY
9	Benjamin Jefferson	82077	Carpinteria North	Sacramento	CA
10	William Van Buren	21712	Tully Road East	Albany	IL
11	Calvin Washington	50742	Richmond Hill	Charleston	CA
12	Jimmy Polk	76143	Richmond Hill	Salt Lake City	AK
13	Calvin Adams	52386	Lake Tahoe Blvd.	Montgomery	NY
14	Ulysses Monroe	70511	Jones Road	Trenton	IL
15	Zachary Tyler	45040	Santa Rosa North	Carson City	AK
16	Ulysses Johnson	19989	Via Real	Juneau	AL
17	George Arthur	89874	Calle Real	Annapolis	AL
18	George Jefferson	67703	Fontaine Road	Pierre	IL
19	Herbert Grant	90635	North Vento Park Road	Columbus	AK

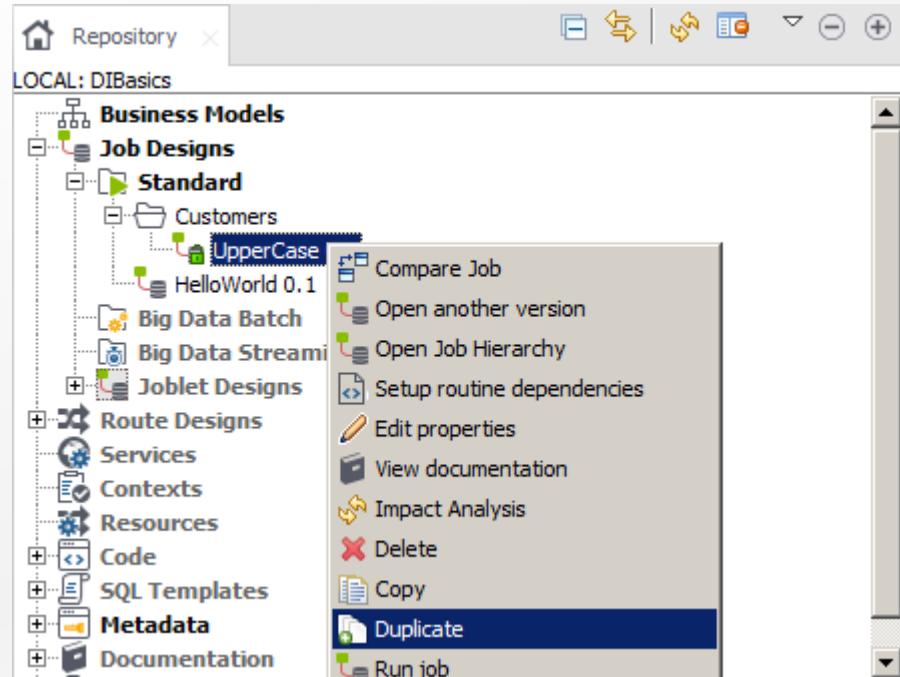
First previous next last 1 page of 4

Set parameters and continue Close

# Duplicate a Job



- Allows reuse of Jobs
- Right-click the Job and select Duplicate

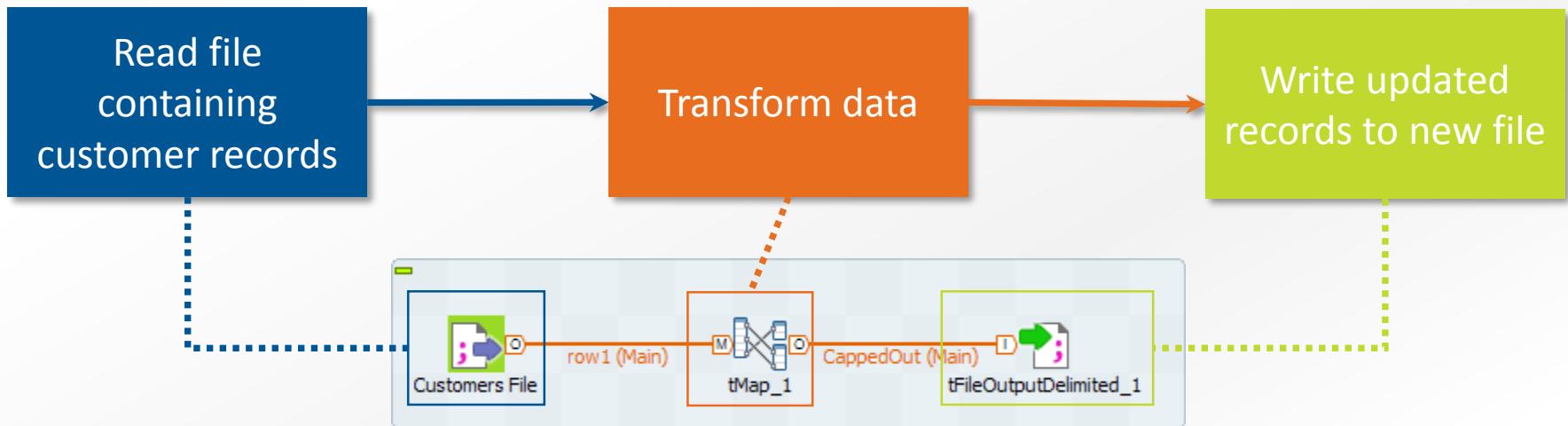


# Lab Overview



Read an input file and write transformed output to a different file

- Build and run the following Job:



# Lesson Summary

---



- **tFileInputDelimited**
  - Reads a delimited file
- **tFileOutputDelimited**
  - Writes a delimited file
- **tMap**
  - Transforms data using operators and predefined functions





# Joining Data Sources

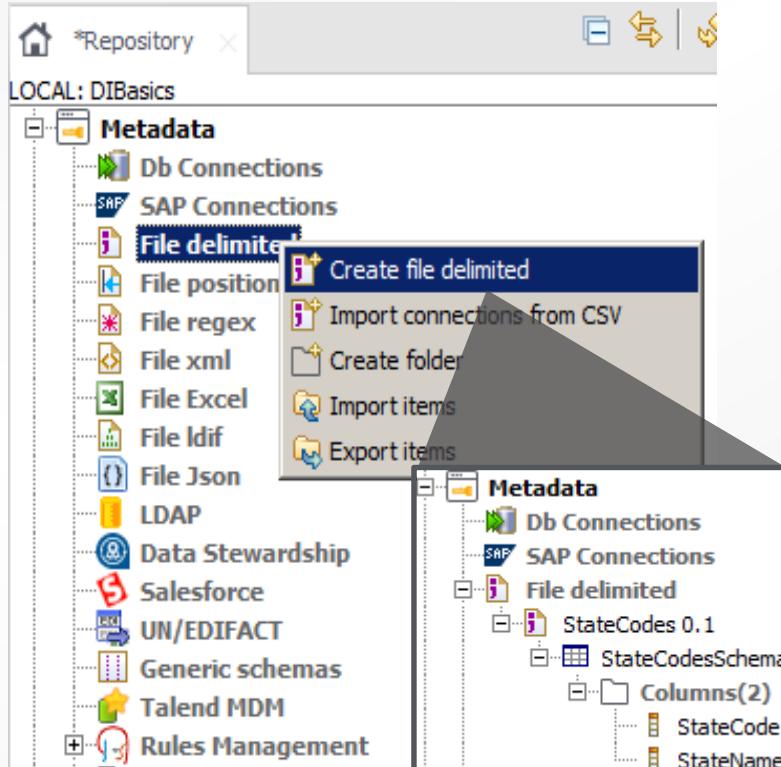
# Objectives

---



- Store metadata centrally for use in other components and Jobs
- Make use of metadata in your Job
- Join two data sources
- Troubleshoot join problems
- Log join rejects to the console

# Create Metadata



- **Metadata** stores configuration information that can be reused in multiple components
- **File Delimited** stores file information:
  - File Name
  - Separator and header information
  - Schema (fields, types, length)

# Use Metadata



## Schema/Property Types

### ● Built-In

- Schema is local to the specific component
- Details are defined locally

**Customers File(tFileInputDelimited\_1)**

**Basic settings**

Property Type: Built-In

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

Dynamic settings: File name/Stream: "C:/StudentFiles/DIBasics/Custs.csv"; Row Separator: "\n"; Field Separator: ";"

View: Documentation: Validation Rules:

CSV options: Header: 1; Footer: 0; Limit:

Schema: Built-In; Edit schema:

Skip empty rows:  Uncompress as zip file:  Die on error:

### ● Repository

- Metadata can be referenced by multiple components
- Details are predefined

**StateCodes(tFileInputDelimited\_2)**

**Basic settings**

Property Type: Repository: DELIM:StateCodes

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

Dynamic settings: File name/Stream: "C:/StudentFiles/DIBasics/States.txt"; Row Separator: "\n"; Field Separator: ";"

View: Documentation: Validation Rules:

CSV options: Header: 0; Footer: 0; Limit:

Schema: Repository: DELIM:StateCodes - StateCodesSchema \*; Edit schema:

Skip empty rows:  Uncompress as zip file:  Die on error:

# Joining Data Sources



- Use **tMap** to join and transform multiple inputs to multiple outputs
  - **tMap** is often part of a solution to various data challenges

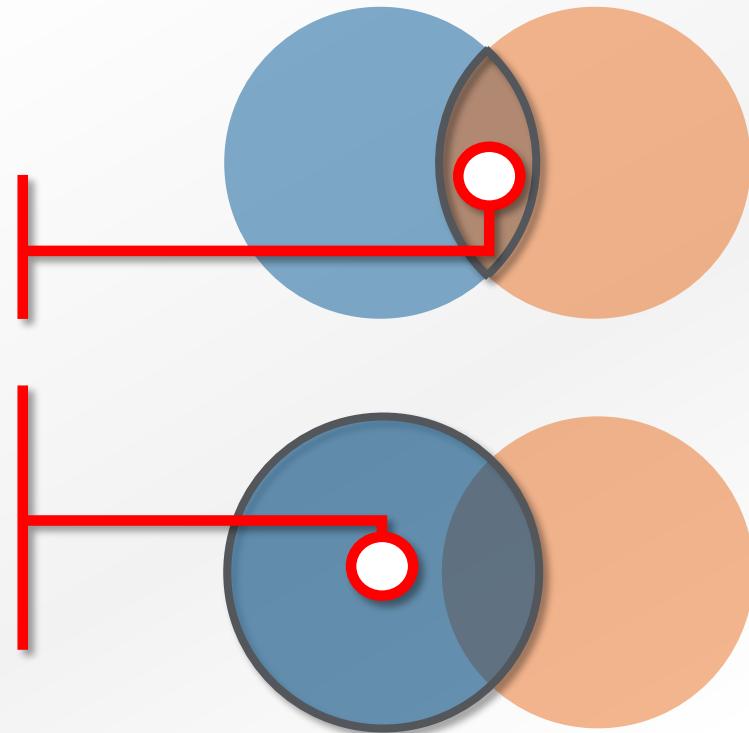


# Joining Data Sources



## Join Types

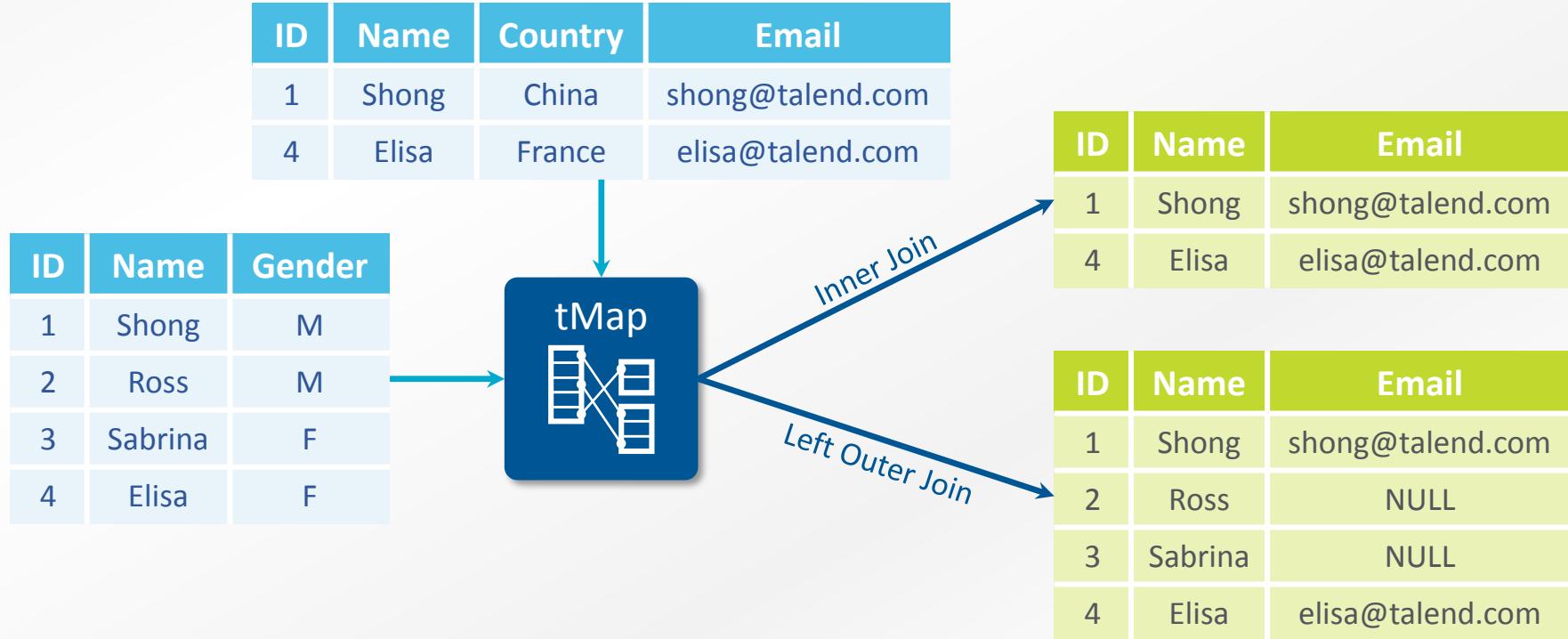
- A **join** combines rows from two or more tables
  - An **inner join** returns rows that only appear in both tables
  - A **left outer join** (or **left join**) returns all records from the left table with NULL column values if no match is found in the right table



# Joining Data Sources



## Comparing Join Types

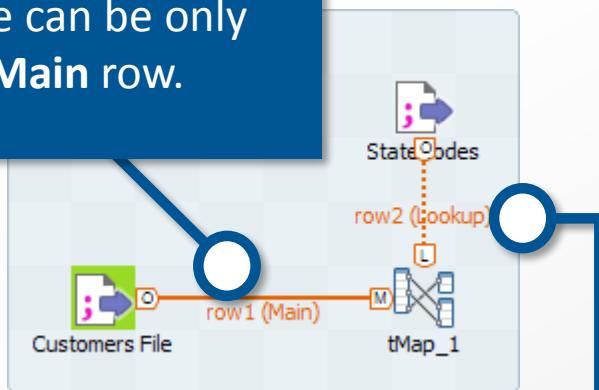


# Join Data Using tMap



- tMap accepts multiple input sources

There can be only one **Main** row.



The rest of the inputs are **lookup** rows.

**1** Define Multiple Input Sources

**2** Configure Join

**3** Define Multiple Output Sources

**4** Handle Rejects

# Join Data Using tMap



The screenshot shows the Talend tMap interface with two rows of data:

- row1:** Contains columns "City" and "State". A red box labeled "1" highlights the "Join data using drag and drop" section.
- row2:** Contains a table of properties:
  - Property: Lookup Model, Value: Load once
  - Property: Match Model, Value: Unique match
  - Property: Join Model, Value: Inner Join (highlighted by a red box labeled "2")
  - Property: Store temp data, Value: falseA red box labeled "3" highlights the "Set join model" section at the bottom.

**1** Define Multiple Input Sources

**2** Configure Join

**3** Define Multiple Output Sources

**4** Handle Rejects

# Join Data Using tMap



The screenshot shows the Talend tMap interface with two red annotations:

- 1** A red circle highlights the "Add output table" button, which is positioned above the "CappedOut" component table.
- 2** A red box highlights the "Schema type (Repository or Built-In)" dropdown menu, which is open over the "JoinFailures" component table.

**CappedOut Component Table:**

Expression	Column
row1.First + " " + row1.Last	Name
row1.Number	Number
row1.Street	Street
row1.City	City
StringHandling.UPCASE(row1.State)	State
row2.StateName	

**JoinFailures Component Table:**

Property	Value
Catch output reject	false
Catch lookup inner join reject	false
Schema Type	Built-In

**1** Define Multiple Input Sources

**2** Configure Join

**3** Define Multiple Output Sources

**4** Handle Rejects

# Join Data Using tMap



The screenshot shows the Talend Studio interface with the 'JoinFailures' configuration dialog open. This dialog contains a table with the following properties:

Property	Value
Catch output reject	false
Catch lookup inner join reject	false
Schema Type	Built-In

An 'Expression' field is present below the table. A red box highlights the ellipsis button ('...') located to the right of the Schema Type row. A secondary 'Options' dialog is displayed, showing a list with 'true' selected and 'false' as an option. The 'OK' and 'Cancel' buttons are at the bottom of this dialog.

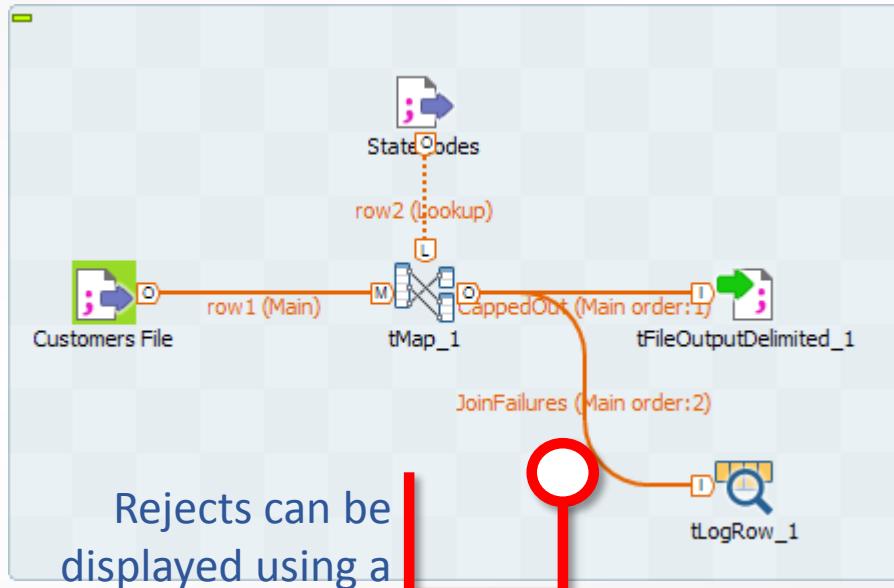
**1** Define Multiple Input Sources

**2** Configure Join

**3** Define Multiple Output Sources

**4** Handle Rejects

# Join Data Using tMap



1 Define Multiple Input Sources

2 Configure Join

3 Define Multiple Output Sources

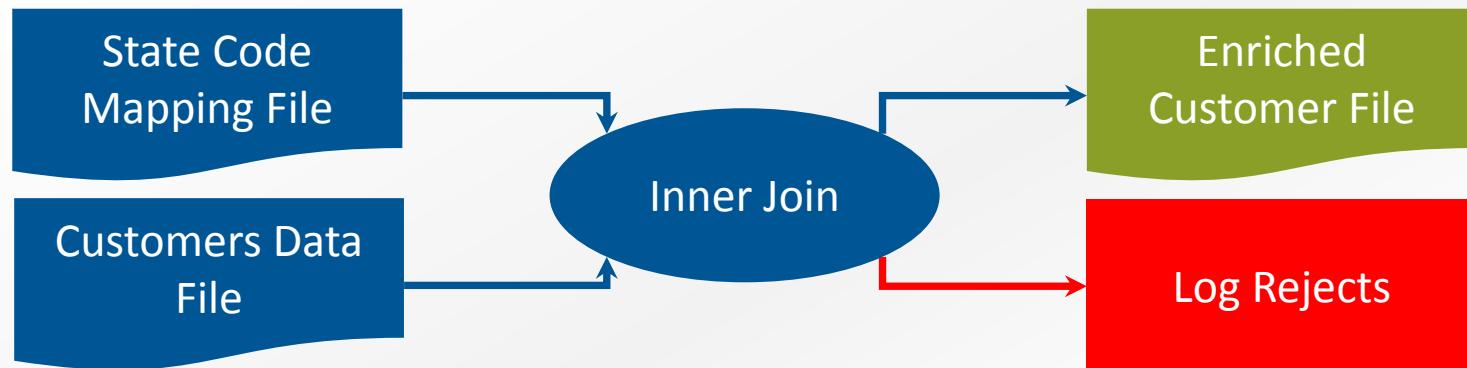
4 Handle Rejects

# Lab Overview



## Joining multiple data sources

- Enrich customer data with state names from an external file
  - Create file delimited repository metadata for states lookup file
  - Build and run the following Job:



# Lesson Summary

---



- Metadata
  - Stores configuration information that can be reused
  - Referenced by **Repository** schema type
- **tMap** includes extensive functionality, such as:
  - Multiple input flows (from multiple sources)
  - Ability to join data sources (inner or left outer join)
  - Complex transformations on the data
  - Catching of rejects
  - Output tables accept both Built-In and Repository schemas





# Filtering Data

# Objectives

---

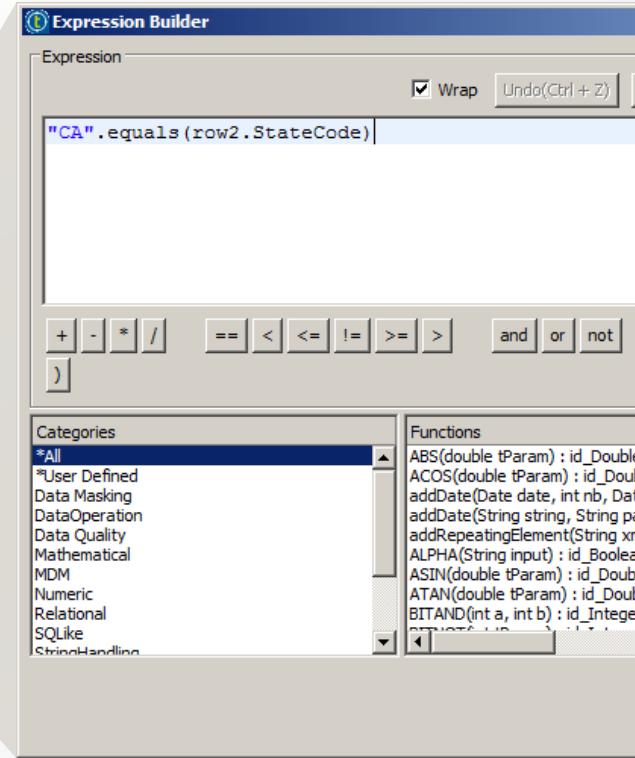
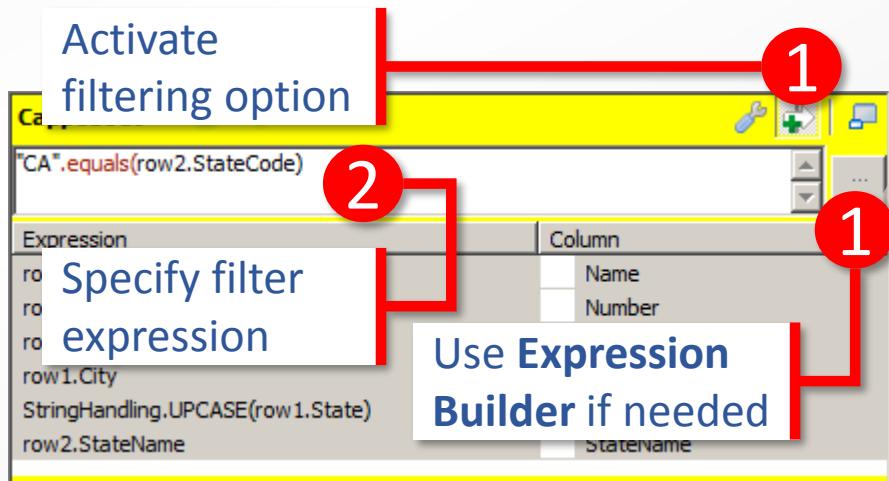


- Use the **tMap** component to filter data
- Execute Job sections conditionally
- Duplicate output flows

# Filtering Data



- The **tMap** component supports data filtering
    - Limits data based on configurable conditions
    - Filters can be applied on input or output table

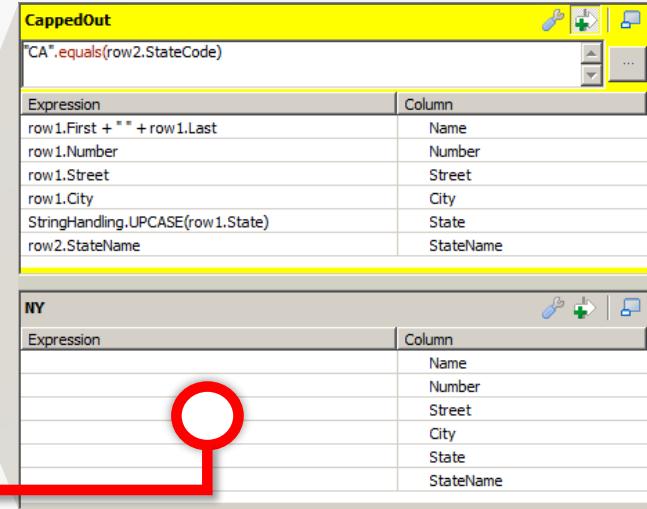
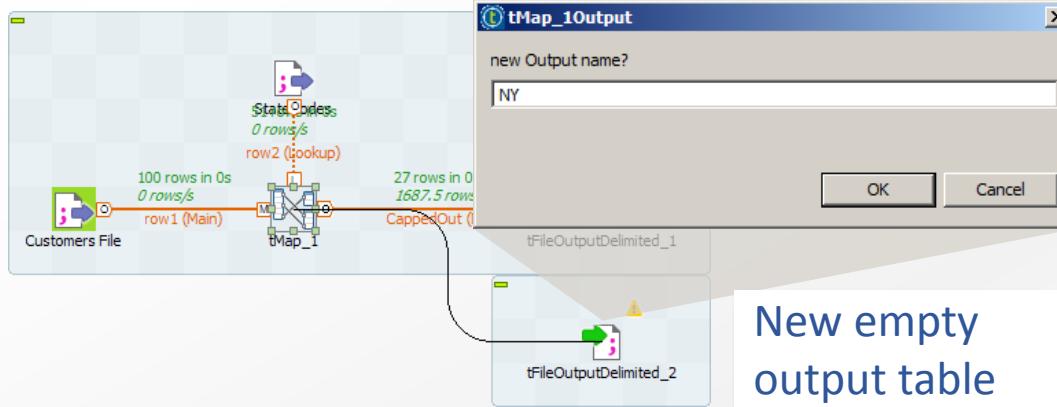




# Duplicate Output Flows



- Create second output component (copy and paste, for example)
- Connect **tMap** to new output component
  - Schema is propagated to the **tMap** component
  - Mapping configuration is empty by default



The screenshot shows two mapping configuration windows. The top window is for 'CappedOut' and the bottom one is for 'NY'. Both windows have columns for 'Expression' and 'Column'. The 'CappedOut' window contains the expression `"CA".equals(row2.StateCode)`. The 'NY' window contains the expression `row1.First + " " + row1.Last`.

Expression	Column
row1.First + " " + row1.Last	Name
row1.Number	Number
row1.Street	Street
row1.City	City
StringHandling.UPCASE(row1.State)	State
row2.StateName	StateName

Expression	Column
	Name
	Number
	Street
	City
	State
	StateName

New empty  
output table

# Duplicate Output Flows

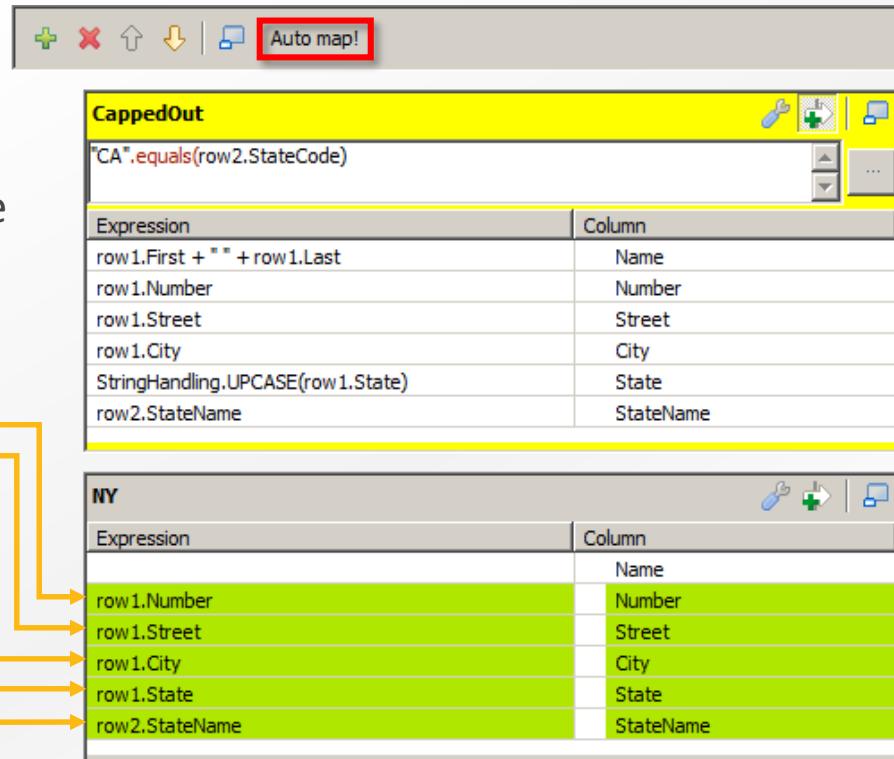


- Use Auto map function to duplicate output flows
  - Maps input columns with same name

row1	
Column	
First	
Last	
Number	
Street	
City	
State	

row2	
Expr. key	Column
row1.State.toUpperCase()	StateCode
	StateName



# Duplicate Output Flows



- Configure the new output table
  - Add filter
  - Apply transformation rules

Add filter expression

Apply transformation  
rules

CappedOut	
<code>"CA".equals(row2.StateCode)</code>	
Expression	Column
<code>row1.First + " " + row1.Last</code>	Name
row1.Number	Number
row1.Street	Street
row1.City	City
<code>StringHandling.UPCASE(row1.State)</code>	State
row2.StateName	StateName

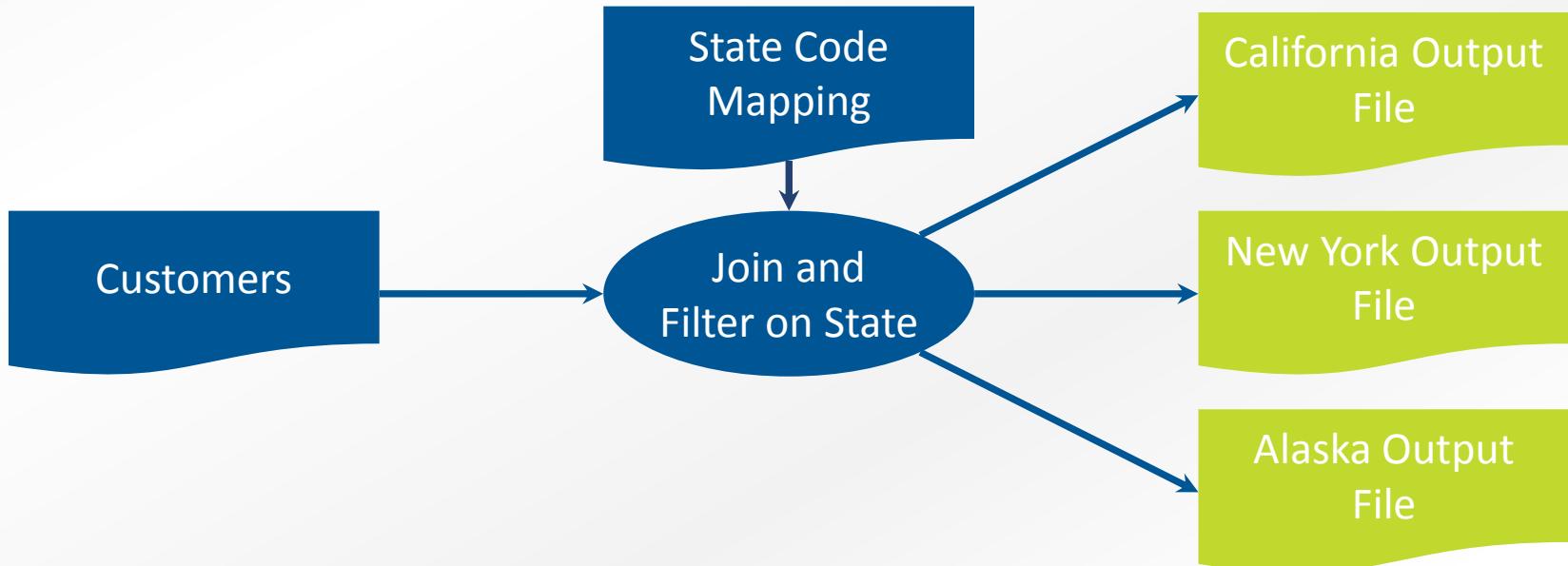
NY	
<code>"NY".equals(row2.StateCode)</code>	
Expression	Column
<code>row1.First + " " + row1.Last</code>	Name
row1.Number	Number
row1.Street	Street
row1.City	City
<code>StringHandling.UPCASE(row1.State)</code>	State
row2.StateName	StateName

# Lab Overview



Redirecting data to different output flows based on input criteria

- Generate three different output files by filtering input based on state



# Lesson Summary

---



- **tMap** allows:
  - Data filtering in input and output tables
  - Multiple input and output flows (from and to files, databases, and more)
  - Ability to join data sources (inner or left outer join)
  - Complex transformations on the data
  - Catching of rejects





# Using Context Variables

# Objectives

---



- Understand the need for context variables
- Create built-in and repository context variables
- Use context variables within your Jobs

# What is a Context Variable?



- A Job is always run in a specific **context**
- **Context variables** are:
  - Variables you can use within your Job
  - Take on a value depending on the context in which the Job is being run
  - Goal: specify the value to use by changing the context in which you run your Job

Context	Development	Test	Production
Variable			
InputDir	C:/Users/Joe/DevInput	C:/Test/Input	C:/Talend/Input
OutputDir	C:/Users/Joe/DevOutput	C:/Test/Output	C:/Talend/Output

# Define Context Variables



- Manage context variables in the **Contexts** view

The screenshot shows the 'Contexts' view in a software application. At the top, there are tabs: 'Job(Filter\_States...)', 'Contexts(Filter\_S...', 'Component', and 'Run (Job...)'. Below the tabs is a toolbar with icons for adding (+), deleting (-), moving up (^), moving down (^), and a refresh symbol.

A red callout points to the first icon with the number 1, labeled 'Add a new context variable'.

A red callout points to the second column of the table with the number 2, labeled 'Assign a name to the variable'.

	Name	Type	Comment
1	OutputDir	String	

At the bottom right of the table, it says 'Default context env'.

**1** Add Variables

**2** Add Contexts

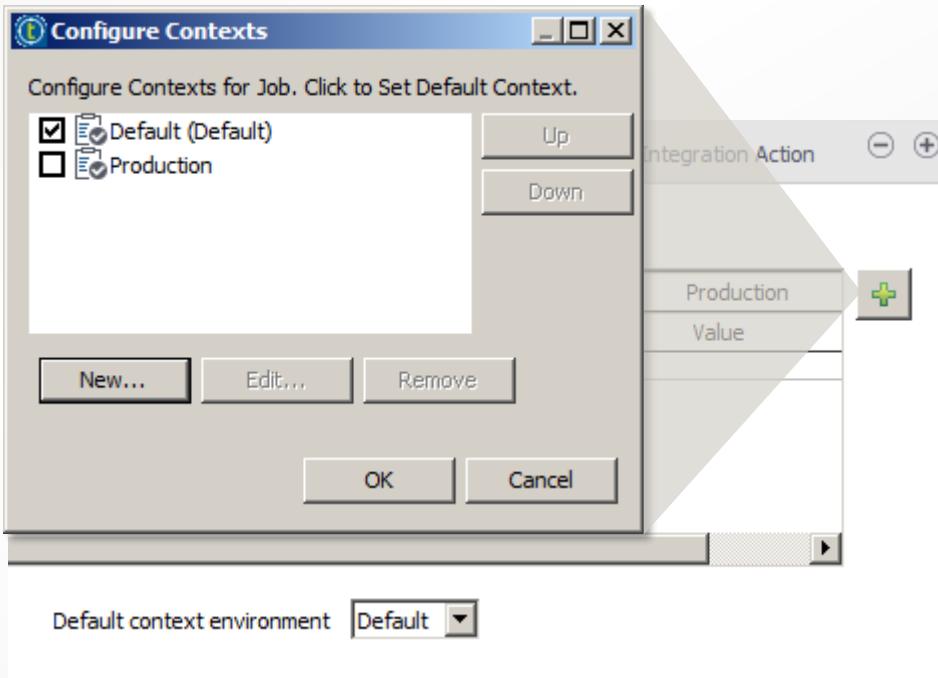
**3** Set Default Context Environment

**4** Set Predefined Values

# Define Context Variables



- Manage contexts in the **Contexts** view



- 1 Add Variables
- 2 Add Contexts
- 3 Set Default Context Environment
- 4 Set Predefined Values

# Define Context Variables



- Select default context in the **Contexts** view

Comment	Default	Production
	Value	Value

Default context environment: Default ▾

Default Production

1 Add Variables

2 Add Contexts

3 Set Default Context Environment

4 Set Predefined Values

# Define Context Variables



Define variable values for the contexts

Component Run (Job Filter\_S...) Test Cases Integration Action

Default		Production	
Value	Prompt	Value	
"Joe/DevOutput"	<input checked="" type="checkbox"/> OutputDir?	"C:/Talend/Output"	<input type="checkbox"/>

Default context environment Default

1 2 3 4

Prompt option enables update of values during execution

1 Add Variables

2 Add Contexts

3 Set Default Context Environment

4 Set Predefined Values

# Repository Context Variables



- Defined in **Repository -> Contexts**
  - Repository version of the context variables defined in the Job's **Contexts** view
  - Can be used across multiple Jobs

Can only be modified in the Repository

Import them from here or drag them from Repository to the Designer

	Name	Type	Comment	Value	OutputDir?	RepoOutputDir?
1	OutputDir	String		"C:/Users/Joe/DevOutput"	<input checked="" type="checkbox"/>	"C:/Ta
2	LocationContextVariables					
3	RepoOutputDir	String		"C:/Users/Joe/Devoutput"	<input type="checkbox"/>	RepoOutputDir?

Default context environment Default

Toolbar icons: +, -, up, down, list, save.

# Using Context Variables



Access the Context Variables from the Component of a Job

- All Context variables (built-in or repository) are accessed the same way
  - Example below illustrates an expression to access the *OutputDir* variable

The screenshot shows the configuration window for a **tFileOutputDelimited\_1** component. The top navigation bar includes tabs for Job, Contexts, Component (which is selected), Run, Test Cases, and Integration Action. On the left, a sidebar lists settings: Basic settings (selected), Advanced settings, Dynamic settings, View, Documentation, and Validation Rules. The main configuration area has the following fields:

- Property Type:** Built-In (highlighted with a red circle)
- Use Output Stream:**
- File Name:** context.OutputDir + "CappedOut.csv"
- Row Separator:** "\n"
- Field Separator:** ";"
- Append:**
- Include Header:**
- Compress as zip file:**
- Schema:** Built-In (highlighted with a red circle)
- Buttons:** Edit schema, Sync columns

A red callout box points to the **File Name** field with the text: "Enable file name to be controlled through the context variable *OutputDir*".

# Using Context Variables



## Set the Context when Running a Job

- Specify execution context in the Run View
  - The context, variables, and values are displayed
  - When the context is changed, values are updated – but they cannot be changed

The screenshot shows the Talend Run View interface. At the top, there's a toolbar with tabs: Job(Context\_Upper...), Contexts(Context...), Component, Run (Job Context...), Test Cases, Integration Action, and some icons. Below the toolbar, the title bar says "Job Context\_Upper". On the left, a sidebar has a "Basic Run" section with options: Debug Run, Advanced settings, Target Exec, and Memory Run. In the center, there are buttons for Run, Kill, and Clear. A text area says "Choose the context here". To the right, there's a dropdown menu set to "Production" and a table titled "Context variables (built-in and repository) and values". The table has columns "Name" and "Value", with two entries: OutputDir ("C:/Talend/Output") and RepoOutputDir ("C:/Talend/Output"). Red annotations with arrows point from the text "Choose the context here" to the dropdown menu, and from the text "Context variables (built-in and repository) and values" to the table.

Name	Value
OutputDir	"C:/Talend/Output"
RepoOutputDir	"C:/Talend/Output"

# Lab Overview

---



## Create Variables to Provide Different Values in Two Different Contexts

- Create build-in context variables
  - Use the context variable
  - Change context
- Create repository context variables
  - Add repository context variables to a Job
  - Use the repository context variable created
  - Update value of variable in the repository

# Lesson Summary

---



## Context Variable Types

- **Built-in context variables:**
  - Defined for each Job in the **Contexts** view
  - Can be updated in the **Contexts** view for the Job
- **Repository** context variables:
  - Defined in **Repository > Contexts**
  - Can be used in multiple Jobs
  - Can only be updated in the repository
- Execution Context can be set up in the **Run** view





# Error Handling

# Objectives

---

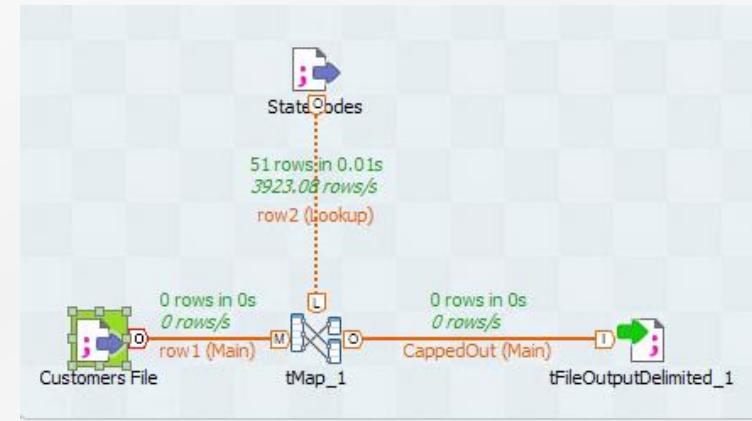
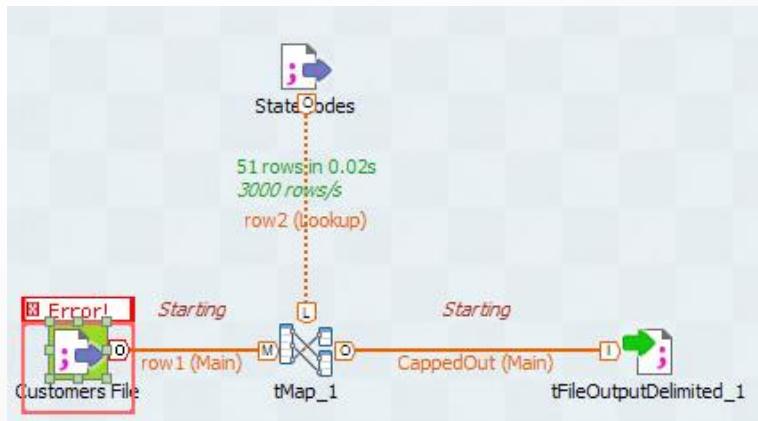


- Perform basic error handling in the Studio
  - Kill a Job when an error occurs
  - Implement a specific Job execution path on a component error
- Raise a warning under specific conditions
- Configure the verbosity level in the run console

# Error Handling



- In the event that a component encounters an error (for example, the input file is missing), execution options include:
  - Stop as soon as the error is encountered
  - Continue to completion despite the error



# Error Handling



## Die on Error option

- The **Die on error** option is available for many components
  - Some components always die on error, so the option is not available

Component

**Customers File(tFileInputDelimited\_1)**

**Basic settings**

Property Type: Built-In

File name/Stream: "C:/StudentFiles/nofile"

Row Separator: "\n"

Field Separator: ";"

CSV options

Header: 1

Footer: 0

Limit:

Schema: Built-In

Skip empty rows

Uncompress as zip file

Die on error

Validation Rules

Advanced settings

Dynamic settings

View

Documentation

Error!

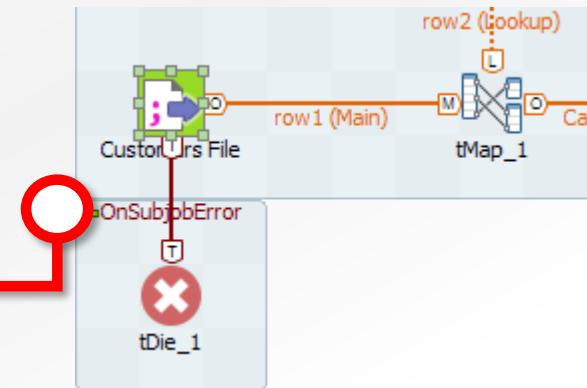
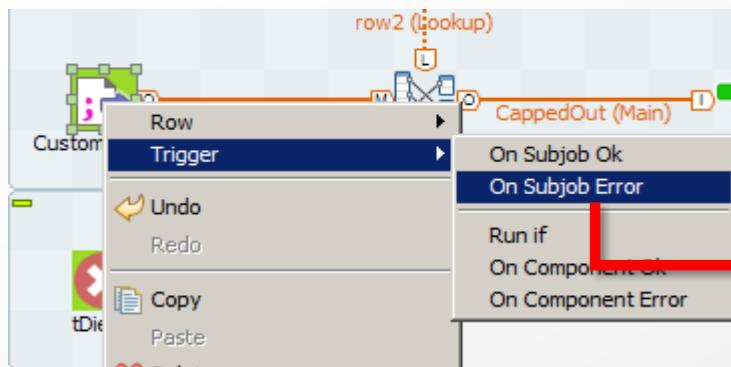
Customers File

# Error Handling



## Triggers

- Triggers transfer control from one component to another
- Allow the implementation of different execution paths based on:
  - The status of a component or subJobs
  - Other specific conditions



# Error Handling



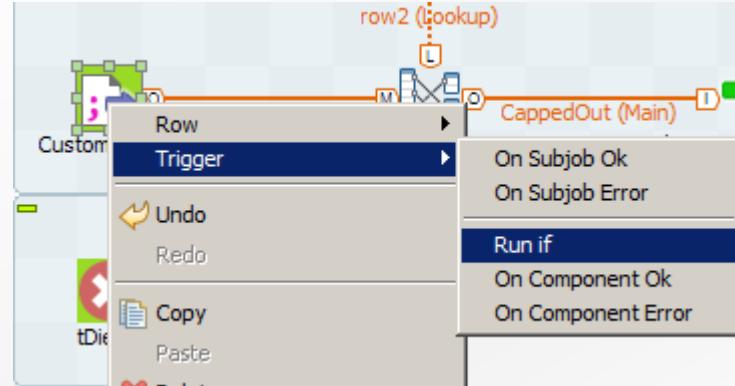
## Kill a Job Under Specific Conditions

- Requirement:

- Kill a Job under specific conditions  
(even if no technical error occurred)
- For example: input file has less than  
a certain number of lines

- Solution:

- Use **tDie** to kill the Job
- Connect with **Run if** trigger
- Configure the condition



# Raise a Warning

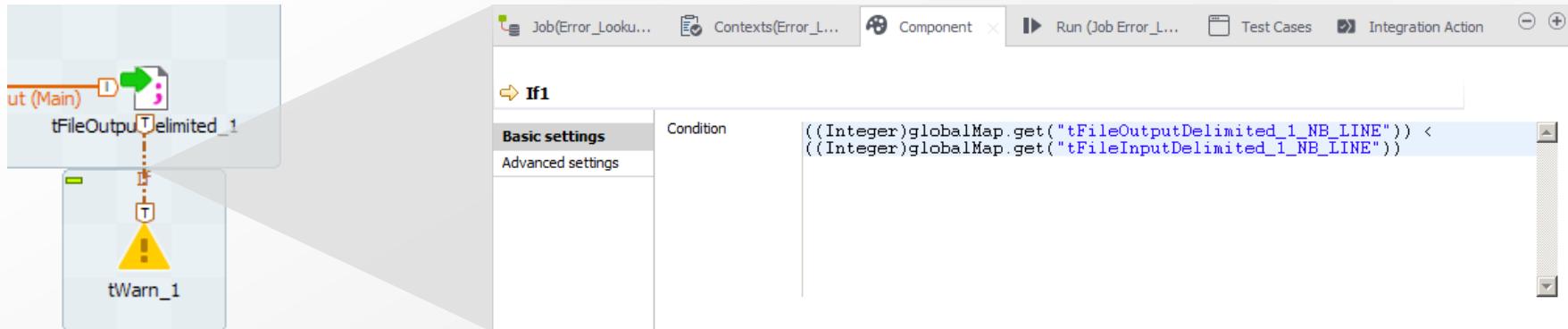


- Requirement:

- Raise a warning when a business rule is not met
- Example: the output file has fewer lines than the input

- Solution:

- Use **tWarn** to raise the warning
- Connect with **Run if** trigger
- Configure the **condition**



# Logging Level



- Configure the logging level with **log4jlevel**
  - Available under the **Advanced Settings** tab of the **Run View**
  - Specifies the types of log messages to be displayed

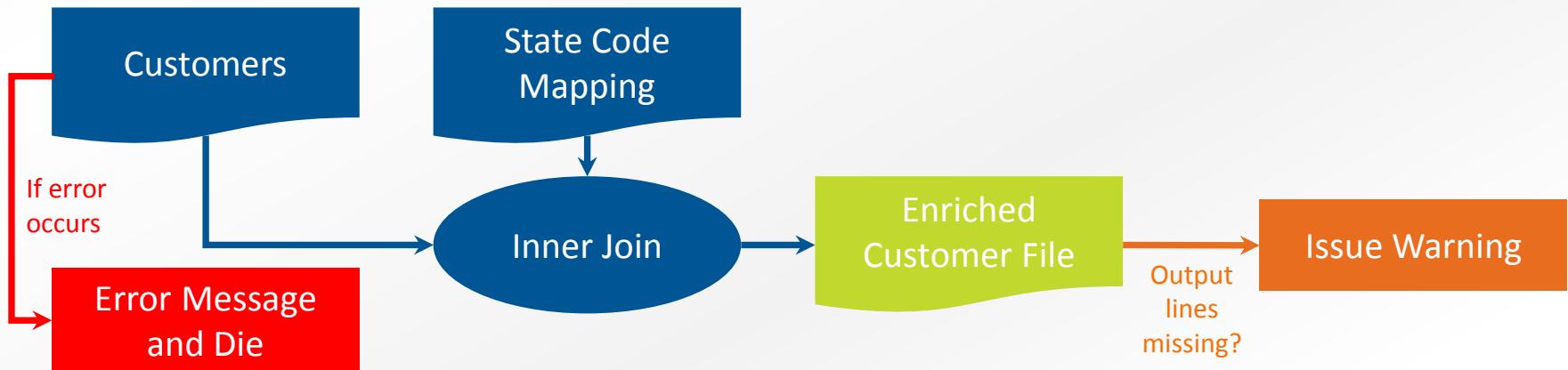
The screenshot shows the Talend Run View interface. The top navigation bar includes tabs for Job, Contexts, Component, Run, Test Cases, and Integration Action. The main area displays a job configuration for "Job Error\_LookupState". The left sidebar lists run types: Basic Run, Debug Run, Advanced settings (selected), Target Exec, and Memory Run. In the "Advanced settings" section, there are checkboxes for Statistics, Save Job before execution, Exec time, and Clear before run. A dropdown menu for "log4jLevel" is open, showing options: Trace, Debug, Info, Warn (selected), Error, Fatal, and Off. Below this, sections for JVM Setting, Job Run VM arguments, and Use specific JVM arguments are visible. On the right, the "Execution" panel shows the log output for running the job. A red box highlights a warning message: "[WARN ]: di\_b.error\_lookupstate\_0\_1.Error\_LookupState - tWarn\_1 - Message: Less than 100 rows written. Code: 42".

```
Starting job Error_LookupState at 02:46  
18/03/2016.  
[statistics] connecting to socket on port 3536  
[statistics] connected  
[WARN ]:  
di_b.error_lookupstate_0_1.Error_LookupState - tWarn_1 - Message: Less than 100 rows  
written. Code: 42  
[statistics] disconnected  
Job Error_LookupState ended at 02:46  
18/03/2016. [exit code=0]
```

# Lab Overview



- Introduce a technical error: input file does not exist
  - Kill the Job
  - Set the die message
- Create a business error condition: output file does not have enough lines
  - Raise a warning



# Lesson Summary

---



- Handling Errors
  - **Die On error** option kills a Job on component error
  - **tDie** component can be used to set the Die message or to kill a Job under certain conditions (in conjunction with a **Run if** trigger)
- Raise a Warning
  - **tWarn** component can be used
- Configure log level in the **Advanced settings** tab of the **Run** view





# Generic Schemas

# Objectives

---

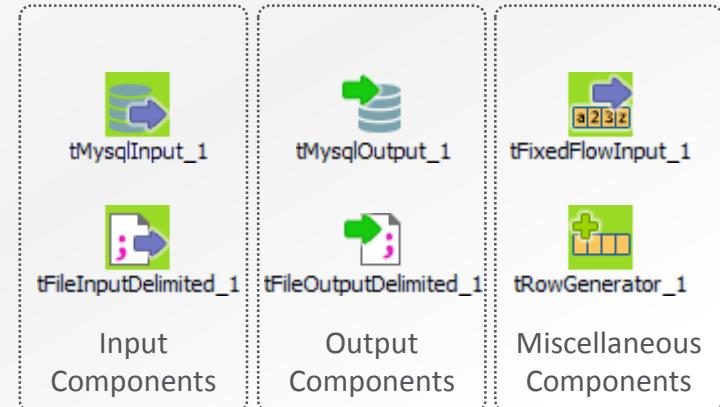
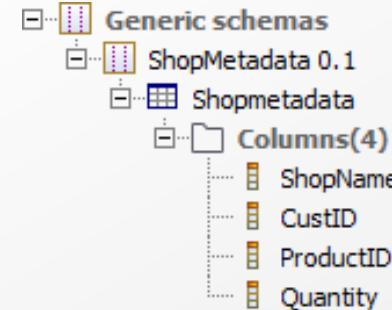


- Create a Generic Schema
- Generate data files based on random data
- Capture all information on your files in context variables
- Use subJobs

# Generic Schemas



- The schema metadata covered previously was bound to a specific file type
  - Delimited, XML, JSON, and more
  - Cannot be applied universally
- A **generic schema** is not linked to a resource type and can be used by any component
  - Some examples shown on right



# Create New Generic Schema



- Generic schemas can be found under **Repository > Metadata**



**Create new generic schema**

Create new generic schema of 2

Add a Schema on repository  
Define the Schema

Name: Shopmetadata

Comment:

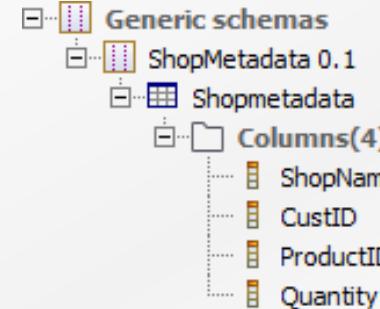
Select the database mapping type:

**Schema**

Description of the Schema

Column	Key	Type	N...	Date
ShopName	<input type="checkbox"/>	String	<input checked="" type="checkbox"/>	
CustID	<input type="checkbox"/>	String	<input checked="" type="checkbox"/>	
ProductID	<input type="checkbox"/>	String	<input checked="" type="checkbox"/>	
Quantity	<input type="checkbox"/>	String	<input checked="" type="checkbox"/>	

**Buttons:** +, -, Up, Down, Save, Cancel, Import, Export



The generic schema appears in the repository

# Generating Random Data



## Use the tRowGenerator Component

- The first step is to specify the schema:
  - Built-in
  - Repository

The screenshot shows the Talend component editor interface. The top navigation bar includes 'Job(gfhd 0.1)', 'Contexts(gfhd)', 'Component' (selected), 'Run (Job gfhd)', 'Test Cases', and other icons. Below the navigation is a toolbar with a green plus icon and the text 'tRowGenerator\_1'. The main area has a sidebar with tabs: 'Basic settings' (selected, highlighted in grey), 'Advanced settings', 'Dynamic settings', 'View', 'Documentation', and 'Validation Rules'. The main panel contains a 'Schema' section with a dropdown menu set to 'Repository' and a text input field showing 'GENERIC:ShopMetadata - Shopmetadata'. A 'RowGenerator Editor' button is also present.

- 1 Define Schema
- 2 Set Number of Rows
- 3 Select Functions
- 4 Set Environment Variable Values

# Generating Random Data



Talend Data Fabric - tRowGenerator - tRowGenerator\_1

Schema	Functions	Preview
Column	Type	Functions
ShopName	String	...
CustID	Integer	Numeric.random(int... min value=>0 ; ma...
ProductID	Integer	Numeric.random(int... min value=>0 ; ma...
Quantity	Integer	Numeric.random(nt... min value=>0 ; ma...

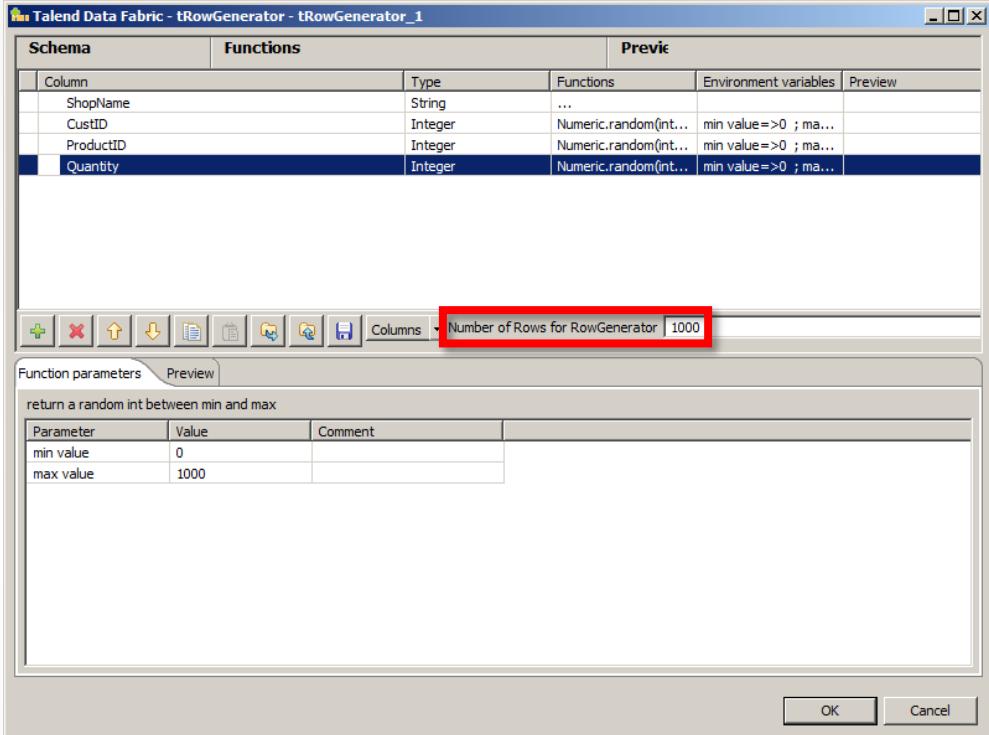
Number of Rows for RowGenerator: 1000

Function parameters Preview

return a random int between min and max

Parameter	Value	Comment
min value	0	
max value	1000	

OK Cancel



1

Define Schema

2

Set Number of Rows

3

Select Functions

4

Set Environment Variable Values

# Generating Random Data



- For each column in the schema, select the function that will be used to generate the data.

Schema	Functions	Previe
Column	Type	Functions
ShopName	String	...
CustID	Integer	Numeric.random(int... min value=>0 ; ma...
ProductID	Integer	Numeric.random(int... min value=>0 ; ma...
Quantity	Integer	ieric.random(int,int) ▾ min value=>0 ; ma...

Below the table is a list of available functions:

- DataOperation.XTD(String)
- Mathematical.BITAND(int,int)
- Mathematical.BITNOT(int)
- Mathematical.BITOR(int,int)
- Mathematical.BITXOR(int,int)
- Mathematical.INT(String)
- Mathematical.NUM(String)
- Mathematical.SCMP(String,String)
- Mathematical.SDIV(int,int)
- Numeric.random(int,int)
- Numeric.sequence(String,int,int)
- StringHandling.COUNT(String,String)
- StringHandling.INDEX(String,String)
- StringHandling.LEN(String)
- TalendDate.compareDate(Date,Date)
- TalendDate.compareDate(Date,Date,String)
- TalendDate.diffDateFloor(Date,Date,String)
- TalendDate.getPartOfDay(String,Date)

At the bottom left are icons for adding, removing, and navigating rows, along with buttons for "Columns" and "Number of Rows".

1 Define Schema

2 Set Number of Rows

3 Select Functions

4 Set Environment Variable Values

# Generating Random Data



- Depending on the selected Function, set the corresponding parameters.

Talend Data Fabric - tRowGenerator - tRowGenerator\_1

Schema	Functions		
Column	Type	Functions	Environment variables
ShopName	String	...	
CustID	Integer	Numeric.random(int... min value=>0 ; ma...	
ProductID	Integer	Numeric.random(int... min value=>0 ; ma...	
Quantity	Integer	Numeric.random(int... min value=>0 ; ma...	

Define range for selected column *Quantity*

Number of Rows for RowGenerator: 1000

Function parameters Preview

return a random int between min and max

Parameter	Value	Comment
min value	0	
max value	1000	

1 Define Schema

2 Set Number of Rows

3 Select Functions

4 Set Environment Variable Values



# Generating Random Data

## Preview the Data

- Verify the configuration using the **Preview** feature

Function parameters Preview

10 Number of Rows for Preview Preview

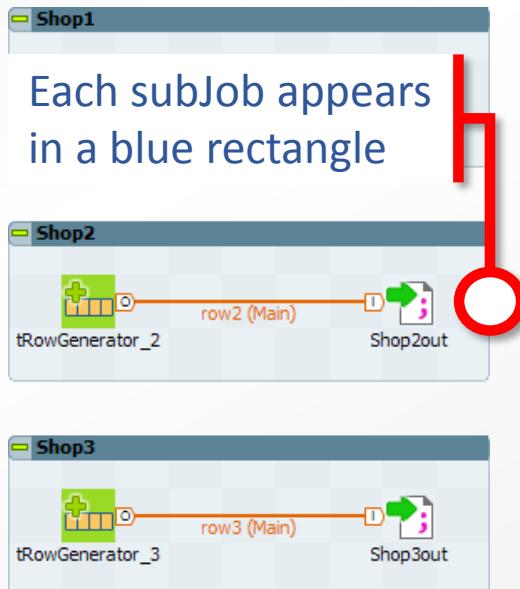
	ShopName	CustID	ProductID	Quantity
1	Shop1	11	31	596
2	Shop1	7	10	821
3	Shop1	89	40	158
4	Shop1	47	45	48
5	Shop1	76	35	629
6	Shop1	64	0	142
7	Shop1	79	19	730
8	Shop1	32	1	649
9	Shop1	47	45	875
10	Shop1	40	49	18

Check that generated values are as expected. For example, *Quantity* should be between 1 and 1000.

# What is a Subjob?



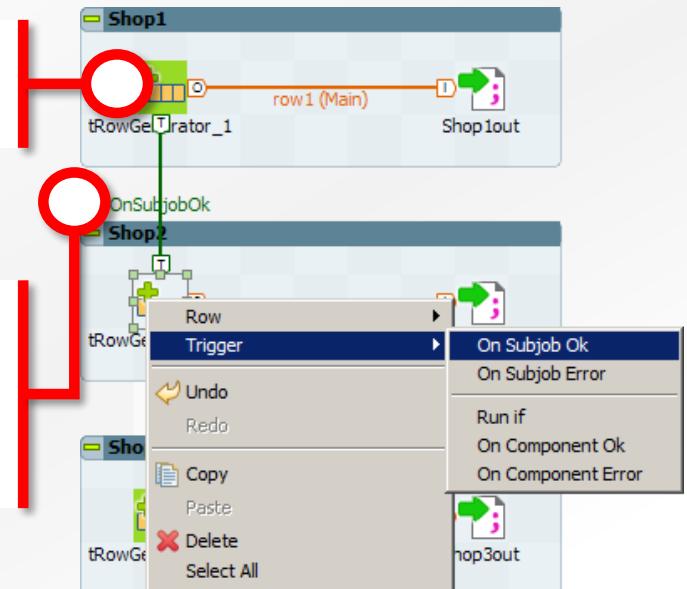
- A collection of connected components within a Job



Triggers are connected to first component in subJob

- Triggers are not flows (no data transferred)
- Used for sequencing subJob execution

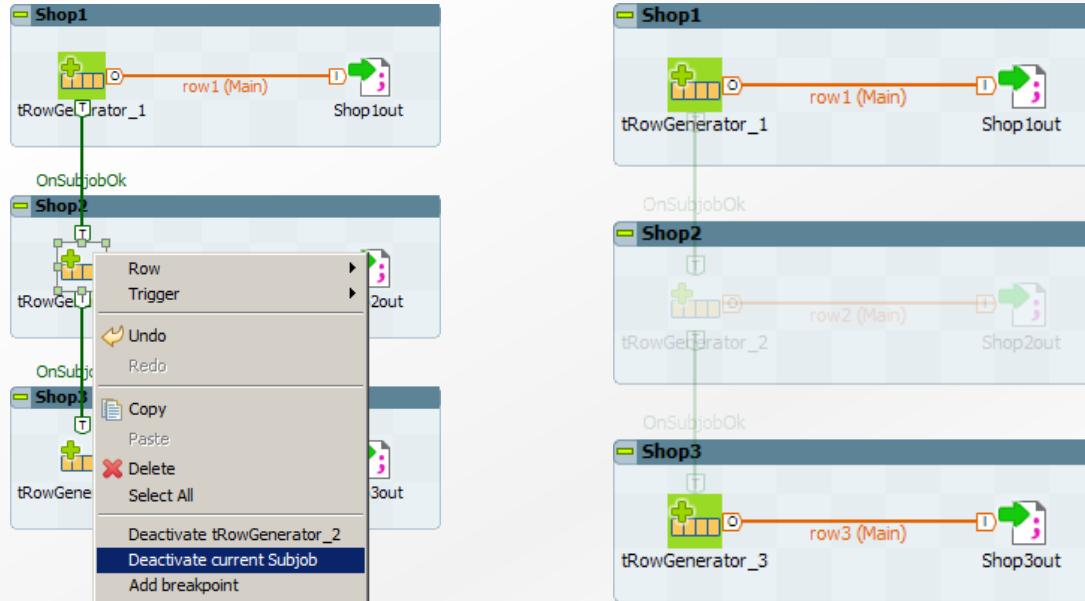
- SubJobs are connected with triggers



# Deactivating SubJobs



- Any subJob (or component) can be activated or deactivated.
  - Once deactivated, the subJob components appear grayed-out.

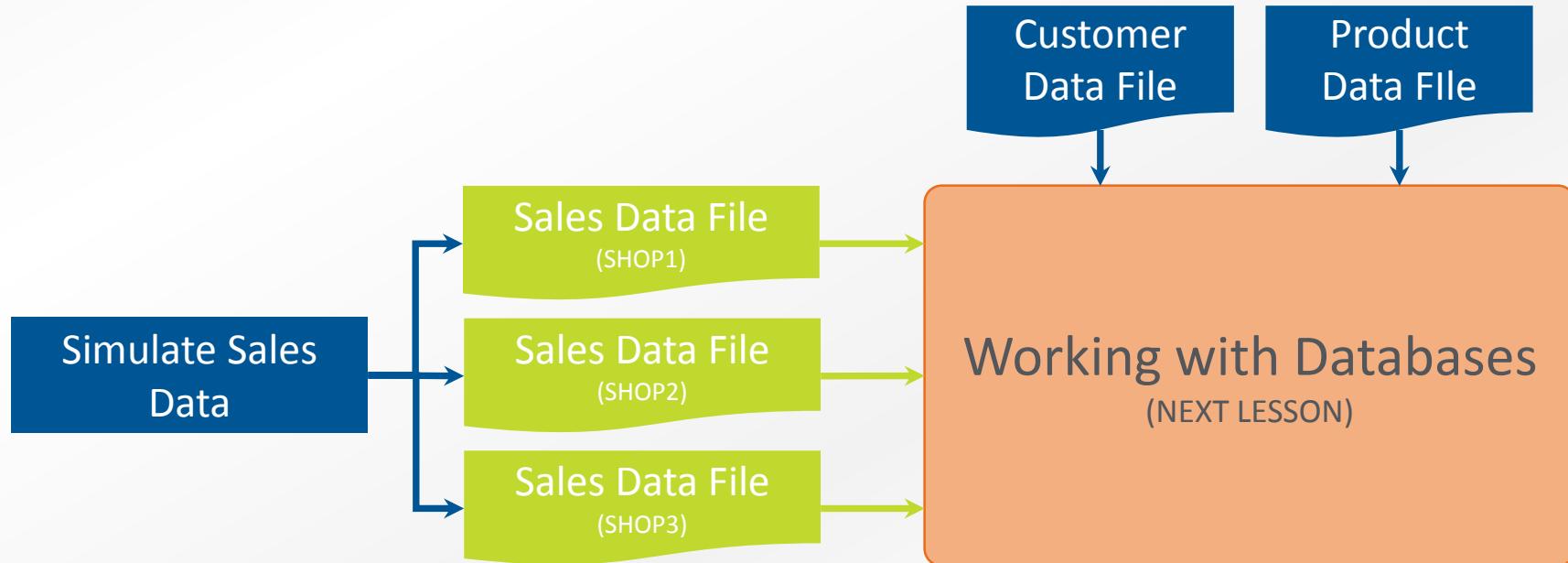


# Lab Overview



## Generic Schemas

- Prepare schemas, files, and metadata for the next lesson



# Lesson Summary

---



- **Generic schemas** are reusable schemas not linked to any type of resource
- **tRowGenerator** paired with generic schemas can be used to produce sample data for your JObs
  - Great for testing
  - Easy to test edge case values (for example, boundaries such as low or high limits)
- You can use subJobs with triggers to process data based on different flows
  - When things work properly (**On Subjob Ok**) or not (**On Subjob Error**)
  - **Run if** triggers can also be employed for arbitrary conditions





# Working With Databases

# Objectives

---



- Connect to a database from a Talend Job
- Use a component to create a database table
- Write to and read from a database table
- Filter unique data rows
- Aggregate data and perform calculations
- Write data to an XML file



# Database Connections

## Creating Metadata

- Database connection metadata is found under

**Repository > Metadata > Db connections**

The screenshot shows the SAP BusinessObjects interface with the following components:

- Left Sidebar:** A tree view under "Metadata" with nodes like "Db Connection", "SAP Connection", "File delimited", "File positional", "File regex", and "File xml".
- Central Area:** A context menu is open over the "Db Connection" node, listing options: "Create connection", "Import connections from CS", "Create folder", "Import items", and "Export items".
- Top Window:** A dialog titled "Database Connection" with the sub-tittle "New Database Connection on repository - Step 2/2". It contains a note: "You must press the Check Button to check the Database Setting".
- Bottom Window:** A "Check Connection" dialog showing the result: "'LocalMySQL' connection successful." with an "OK" button.
- Annotations:**
  - A red callout points to the "Create connection" option in the context menu with the text "Specify basic database connection information".
  - A red callout points to the "Check" button in the "Check Connection" dialog with the text "Check the connection".

# Database Connections



## Export as Context

- Save connection information as context variables

The screenshot shows two windows related to database connections:

**Left Window: Database Properties**

- SQL Syntax: SQL 92
- String Quote: `'`
- Buttons: Check, Export as context (highlighted), Revert Context, How to install a driver.
- Buttons at the bottom: < Back, Next >, Finish.

**Right Window: Create / Reuse a context group - Step 2 of 2**

Define the contexts, variables and values

	Name	Type	Comment	Default Value
1	LocalMySQL_AdditionalParam	String		noDatetimeStringSync=true
2	LocalMySQL_Port	String		3306
3	LocalMySQL_Login	String		root
4	LocalMySQL_Password	Password		*****
5	LocalMySQL_Database	String		training
6	LocalMySQL_Server	String		localhost

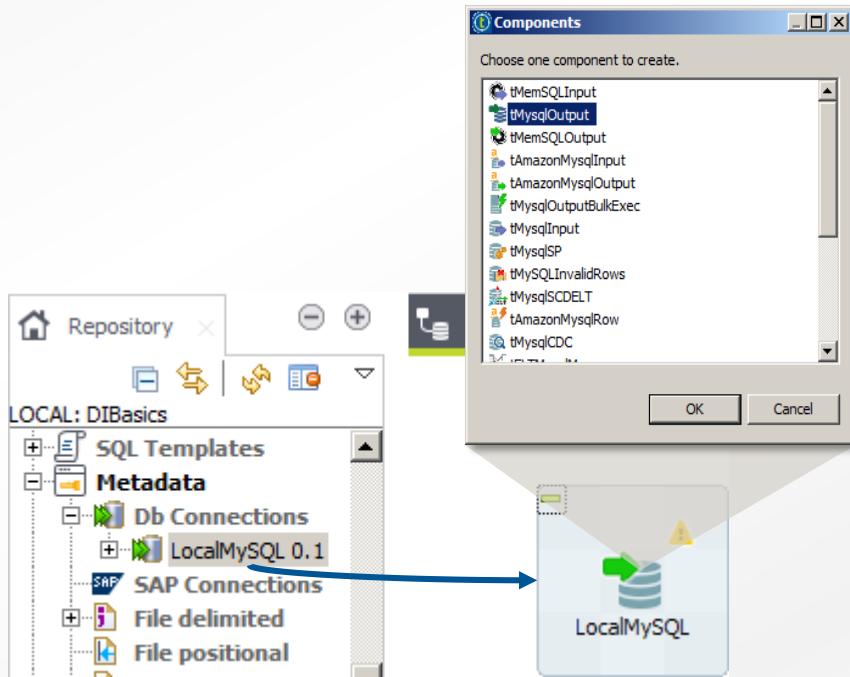
Buttons: +, -

Default context environment: Default

# Database Connections



## Using the Metadata in your Jobs



- Drag the database connection metadata to the **Designer**
  - Select a component from the list
  - Automatically configured with the connection information metadata

The screenshot shows the 'Job(test 0.1)' designer. A 'LocalMySQL(tMysqlOutput\_1)' component is selected. The 'Basic settings' tab is active, showing configuration for 'Property Type' (set to 'Repository'), 'DB Version' (set to 'Mysql 5'), and 'Host' (set to 'context.LocalMySQL\_Server'). The 'Database' field contains 'context.LocalMySQL\_Database'. Both the 'Property Type' dropdown and the 'Host' input field are highlighted with a red border.

# Writing to a Database



## Specifying Table and Schema

- To write to a MySQL database, use the **tMysqlOutput** component
  - Must also specify the **table name** and **schema**

The screenshot shows the Talend Studio interface with the following details:

- Job(test 0.1)**: The current job being edited.
- Contexts(test)**: The context used for this job.
- Component**: The active component tab.
- LocalMySQL(tMysqlOutput\_1)**: The specific component configuration window.
- Basic settings**:
  - Property Type: Repository (selected)
  - DB Version: Mysql 5
  - Use an existing connection
  - Host: context.LocalMySQL\_Server
  - Port: context.LocalMySQL\_Port
  - Database: context.LocalMySQL\_Database
  - Username: context.LocalMySQL\_Login
  - Password: context.LocalMySQL\_Password
  - Table: "CustomerMetadata" (highlighted with a red box)
  - Action on table: Default
  - Action on data: Insert
  - Schema: Repository (selected) / DELIM:RetailCustomers - CustomerMetadata
- Advanced settings**, **Dynamic settings**, **View**, **Documentation**, and **Validation Rules** sections are also present but not highlighted.

# Writing to a Database



## Specifying Table and Data Action

- Specify **Action on table** and **Action on data** as needed
  - Appropriate settings depend on Job requirements and database privileges

The screenshot shows the Talend Studio interface with a component named "LocalMySQL(tMysqlOutput\_1)". The "Basic settings" tab is selected. The "Property Type" dropdown is set to "Repository" and points to "DB (MYSQL):LocalMySQL". The "DB Version" dropdown is set to "Mysql 5". The "Action on table" dropdown is set to "Default" and has a red box around it. A context menu is open over this dropdown, listing options: Insert, Update, Insert or update, Update or insert, Delete, Replace, Insert or update on duplicate key or unique index, and Insert Ignore. The "Action on data" dropdown is also set to "Default" and has a red box around it, with the same context menu options visible above it. Other tabs like "Advanced settings" and "Dynamic settings" are visible but not selected.

# Writing to a Database



## Fixing the Schema

- The schema in a **tMysqlOutput** component requires Database column names and types

The screenshot shows two windows side-by-side. The left window is titled "Schema of LocalMySQL" and displays a table of database columns with various properties like Key, Type, and DB Type. The right window is also titled "Schema of LocalMySQL" and shows a more detailed view of the same or similar columns, with the "DB Type" column highlighted by a red box. Both windows have standard Windows-style toolbars at the bottom.

Column	Db Column	Key	T...	DB Type	✓ N...	Date ...	L...	Pr...	D...
id	id	<input checked="" type="checkbox"/>	In...	INT	<input checked="" type="checkbox"/>		2	0	
CustomerName	CustomerName	<input type="checkbox"/>	St...	VARCHAR	<input checked="" type="checkbox"/>		30	0	
CustomerAddress	CustomerAd...	<input type="checkbox"/>	St...	VARCHAR	<input checked="" type="checkbox"/>		35	0	
idState	idState	<input type="checkbox"/>	In...	INT	<input checked="" type="checkbox"/>		2	0	
id2	id2	<input type="checkbox"/>	In...	INT	<input checked="" type="checkbox"/>		2	0	
RegTime	RegTime	<input type="checkbox"/>	St...	VARCHAR	<input checked="" type="checkbox"/>		19	0	
RegisterTime	RegisterTime	<input type="checkbox"/>	D...	DATETIME	<input checked="" type="checkbox"/>	"yyyy...	23	0	
Sum1	Sum1	<input type="checkbox"/>	F...	FLOAT	<input checked="" type="checkbox"/>		11	0	
Sum2	Sum2	<input type="checkbox"/>	F...	FLOAT	<input checked="" type="checkbox"/>		11	5	
Sum3	Sum3	<input type="checkbox"/>	F...	FLOAT	<input checked="" type="checkbox"/>		11	0	
Sum4	Sum4	<input type="checkbox"/>	F...	FLOAT	<input checked="" type="checkbox"/>		11	5	
Sum5	Sum5	<input type="checkbox"/>	F...	FLOAT	<input checked="" type="checkbox"/>		11	0	
Sum6	Sum6	<input type="checkbox"/>	F...	FLOAT	<input checked="" type="checkbox"/>		11	0	

# Reading from a Database



## Specifying the Query

- To read from a MySQL database, use the **tMysqlInput** component
  - Must also specify **table name** and an **SQL query** to read data from the database

The screenshot shows the Talend Studio interface with the following details:

- Job(RetailSales 0.1)**: The current job name.
- Contexts(RetailSales)**: The current context name.
- Component**: The selected component type.
- Run (Job RetailSales)**: Run button.
- Test Cases**: Test cases tab.
- Integration Action**: Integration action tab.
- ProductDataIn(tMysqlInput\_2)**: The specific component instance being configured.
- Basic settings**: Configuration tab.
- Property Type**: Built-In.
- DB Version**: Mysql 5.
- Use an existing connection**: Unchecked.
- Host**: context.LocalMySQL\_Server.
- Port**: context.LocalMySQL\_Port.
- Database**: context.LocalMySQL\_Database.
- Username**: context.LocalMySQL\_Log.
- Password**: context.LocalMySQL\_Password.
- Schema**: Repository.
- Table Name**: "RetailProducts".
- Query Type**: Built-In.
- Query**:  
"SELECT  
`RetailProducts`.`id`,  
`RetailProducts`.`Product`  
FROM `RetailProducts`"

A large red box highlights the **Table Name** field and the **Query** field. A red bracket on the right side of the screen points to the text "Guess Query features is helpful in configuring the query".

# Processing Data



- Other components that are commonly used to help process data include:



## tSortRow

- Sort input data based on one or more columns
- Specify sort type and order



## tUniqRow

- Filters out duplicate entries
- Can also dump duplicates to a separate output flow if desired



## tAggregateRow

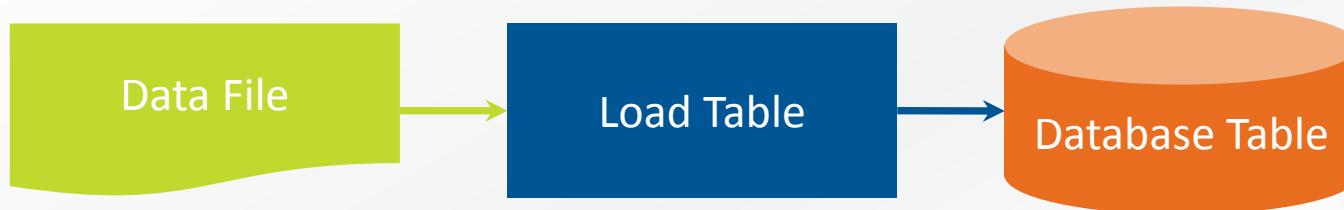
- Aggregates input data based on one or more columns
- Sample operations include min, max, avg, sum, and more

# Lab Overview



## Working with Databases

- First, you will build Jobs to create three tables:
  - Customers
  - Products
  - Sales tables (based on the files generated in the previous lesson)

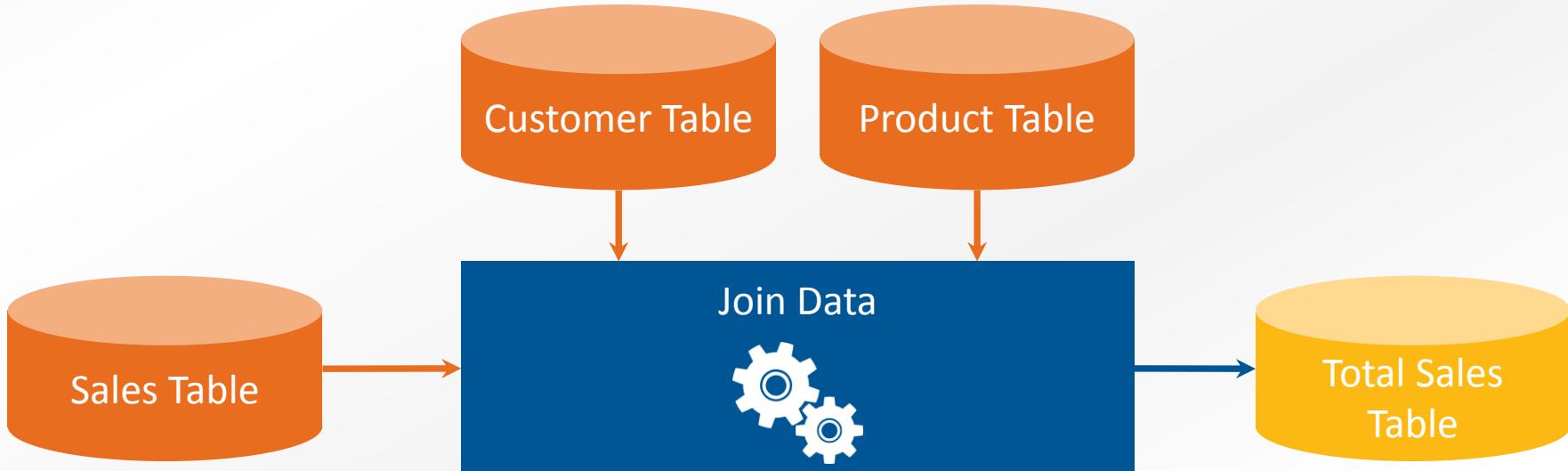


# Lab Overview



## Working with Databases

- Then, you will join the Sales table to the Customer and Product Tables to generate a Total Sales Table



# Lesson Summary

---



- Create database connection information as reusable metadata
- Reading from and writing to databases use the following components:
  - **tCreateTable**
  - **tMysqlInput**
  - **tMysqlOutput**
- Processing data often makes use of the following components:
  - **tSortRow**
  - **tAggregateRow**
  - **tUniqRow**





# Master Jobs

# Objectives

---

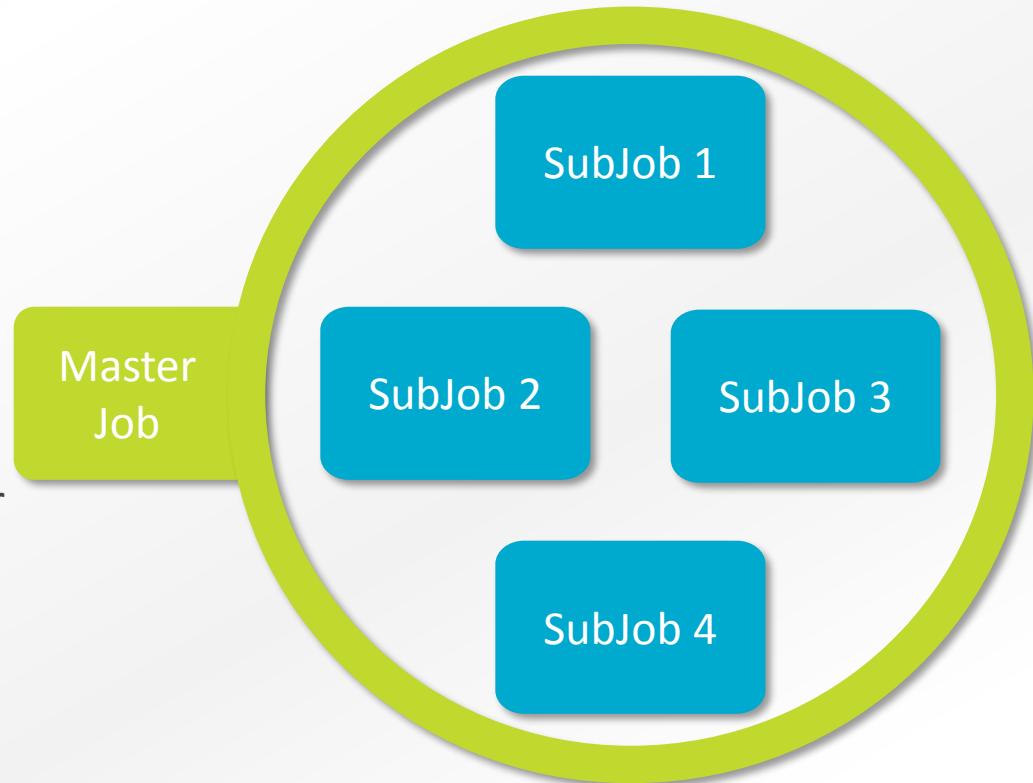


- Create a Master Job
- Override context variables
- Export a Job and its dependencies

# Master Jobs



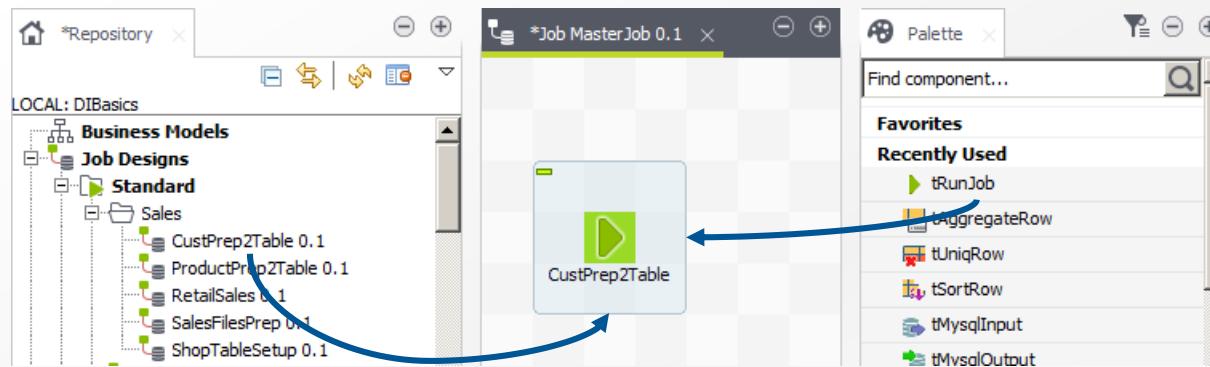
- A **master Job** orchestrates child Jobs
  - **When to use?** Useful for complex Jobs that need to run in sequence.
  - **Why would you use them?** Easy to use orchestration layer within the Talend Studio.
  - **How to do it?** Primarily through **tRunJob** component.



# Master Jobs



- **tRunJob** components are placed like any other component
  - They can also be placed by dragging and dropping Jobs from the repository



# Master Jobs



## Configuration Settings

- Specify important settings, such as the version of the Job to run and what Context to run it in

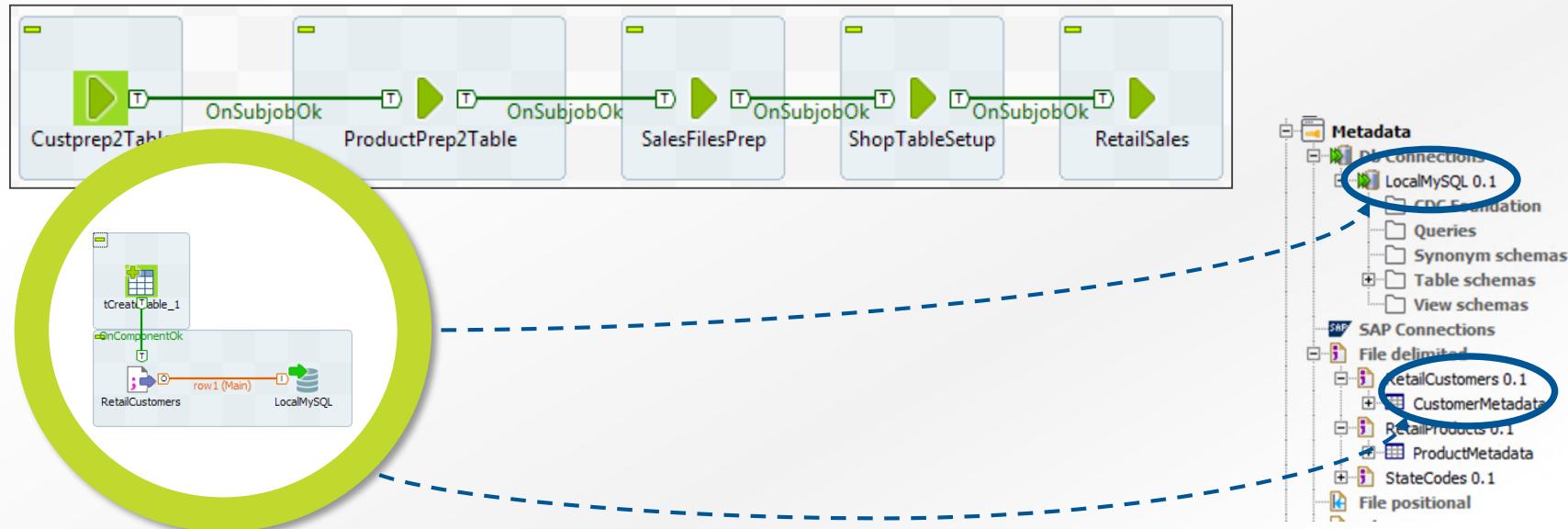
The screenshot shows the Talend Studio interface with the following details:

- Job(MasterJob 0.1) Contexts(MasterJob) Component Run (Job MasterJob) Test Cases Integration Action** are visible in the top navigation bar.
- The job name is **CustPrep2Table(tRunJob\_1)**.
- The **Basic settings** tab is selected.
- The schema is set to **Built-In**.
- The **Version** dropdown is set to **Latest**.
- The **Context** dropdown is highlighted with a red box and a white circle, indicating it can be overridden.
- The **Default** dropdown is also highlighted with a red box and a white circle.
- A tooltip text **Can also override the value of any context variable** is overlaid on the Context dropdown.
- The **Parameters** table shows one entry: **RetailCustomers\_File** with a value of **"C:/Users/Joe/TestFile.csv"**.

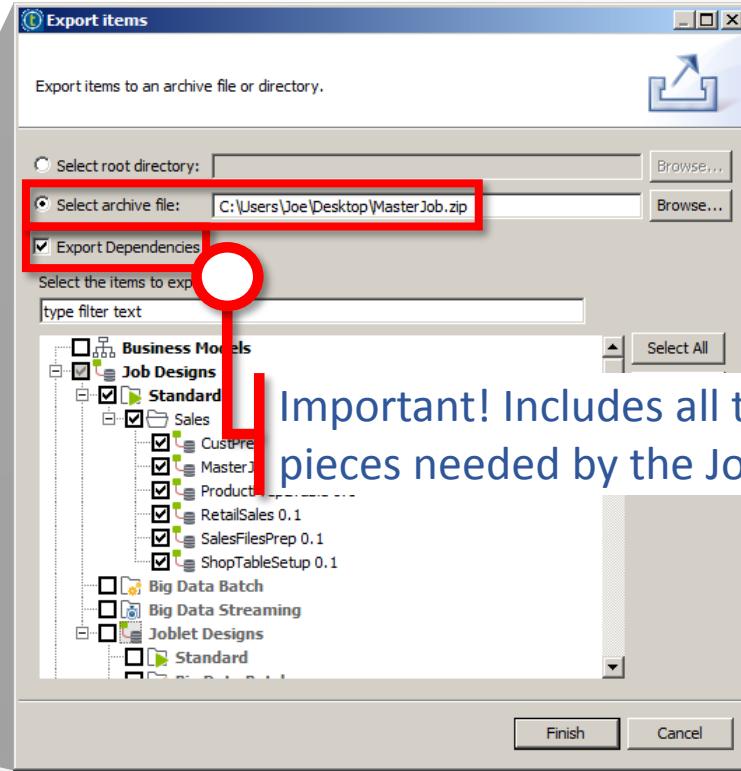
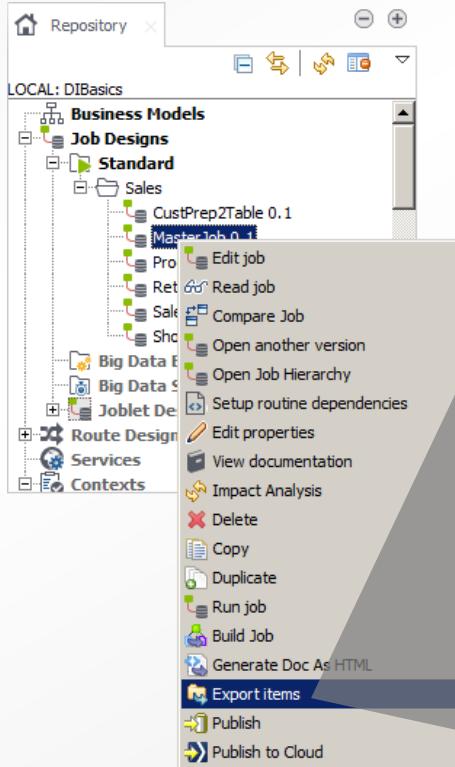
# Exporting Jobs



- Suppose you need to share a master Job with colleagues.
  - It must be packaged with all dependencies so it works in another Studio instance:



# Exporting Jobs



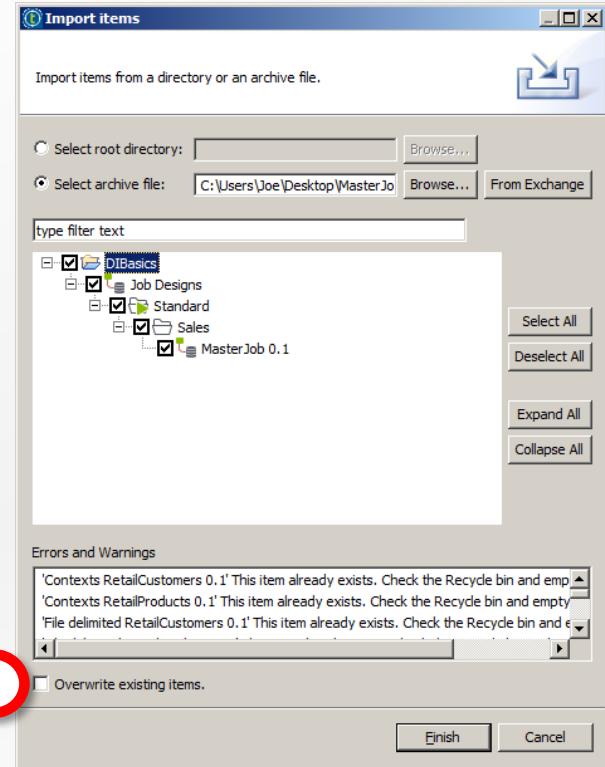
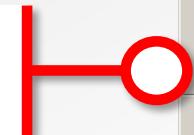
Important! Includes all the extra pieces needed by the Job.

# Importing Jobs



- Select items prior to import
- Select **Overwrite existing items** as needed to clear errors and warnings
  - Typically these arise when some items already exist in the repository
- **Finish** is not actionable until errors are resolved

Use as needed if Job being imported contains items already in repository.

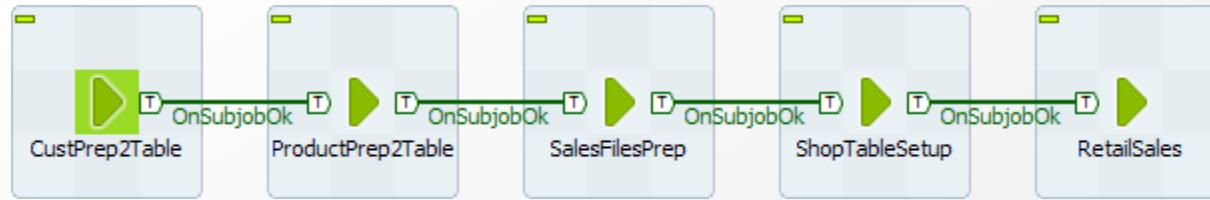


# Lab Overview



## Master Jobs

- You will create a Master Job to control the execution of five other data integration Jobs.



- Then, you will export the Job and all its dependencies.

# Lesson Summary

---



- **tRunJob** is a key component in orchestrating Jobs
- You use triggers to connect subJobs based on a condition; typically:
  - **On Subjob Ok**
  - **On Subjob Error**
- Use **Export items** to share your work with others
  - They can then import your work





# Web Services

# Objectives

---

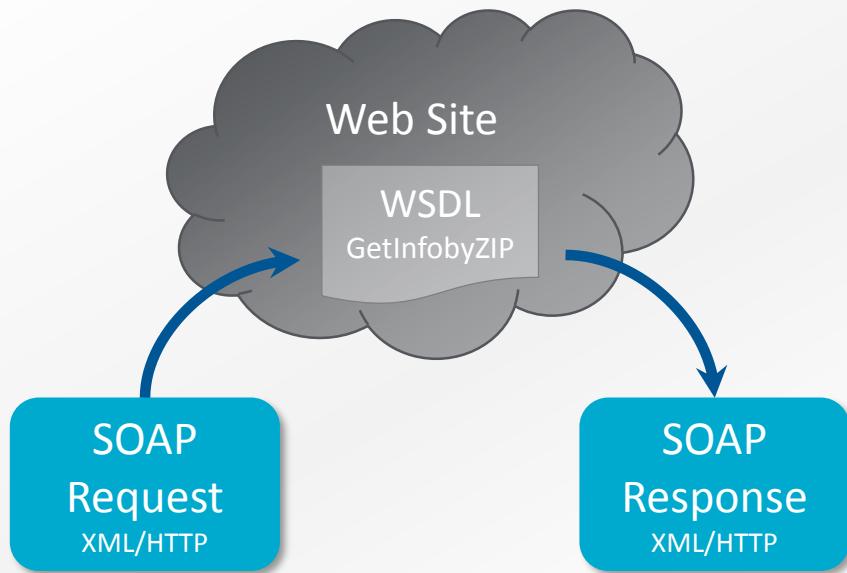


- Use a Talend component to access a Web Service method
- Extract specific elements from a Web Service reply
- Store Web Service WSDL schemas for use in a **tXMLMap** component

# Web Services



- Web Services are applications that communicate over a network
- Web Services offer a standard way of interoperation for different platforms and frameworks
- Web Services are made available through a network endpoint
  - They use messages to exchange requests and responses



# Web Service Description Language



- Web Service Description Language (WSDL) is a contract between the Web Service provider and consumer



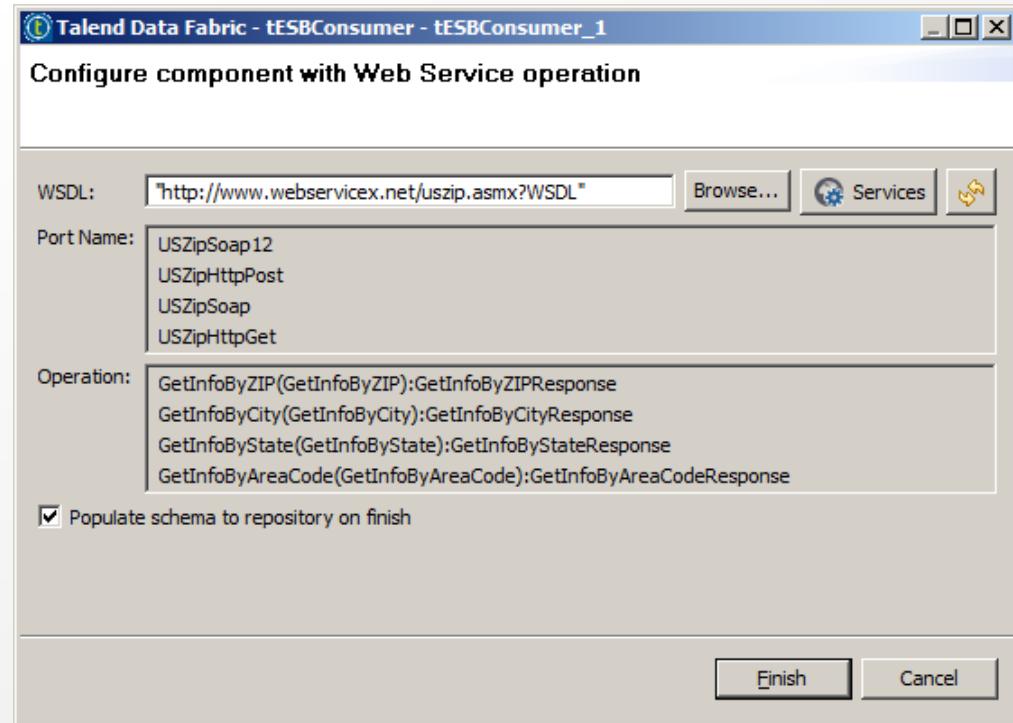
- WSDL defines what operations are supported by the provider
  - WSDL is critical to SOAP-based Web Services
  - In a Talend Job, operations on a Web Service can be invoked with a **tESBConsumer** component

# The tESBConsumer Component



## Example Component Configuration for the Request

- Configure the consumer to issue a SOAP request to get city information by ZIP code
  - Specify WSDL URL
  - Select Port Name and Operation



# The tESBConsumer Component



Example Component Configuration for the Processing the Response Schema

- The schema of the Web Service response can be complex
  - tESBConsumer allows you to retrieve the schema and save it

The screenshot illustrates the configuration of the tESBConsumer component and its resulting schema.

**Talend Data Fabric - tESBConsumer - tESBConsumer\_1**

**Configure component with Web Service operation**

**WSDL:** `http://www.webservicex.net/uszip.asmx?WSDL`

**Port Name:** USZipSoap12, USZipHttpPost, USZipSoap, USZipHttpGet

**Operation:** GetInfoByZIP(GetInfoByZIP):GetInfoByZIPResponse, GetInfoByCity(GetInfoByCity):GetInfoByCityResponse, GetInfoByState(GetInfoByState):GetInfoByStateResponse, GetInfoByAreaCode(GetInfoByAreaCode):GetInfoByAreaCodeResponse

**Populate schema to repository on finish** (checkbox checked)

**Finish** | **Cancel**

**Metadata**

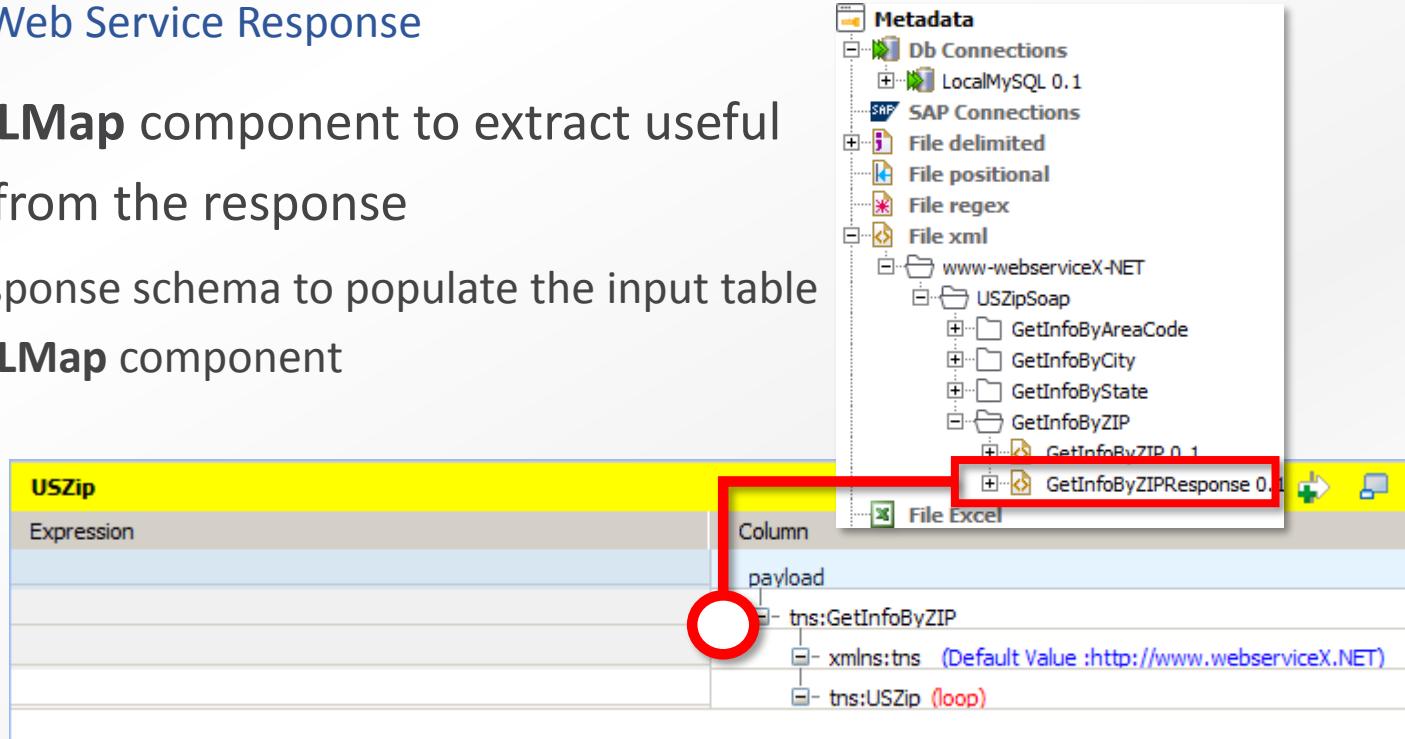
- Db Connections
- SAP Connections
- File delimited
- File positional
- File regex
- File xml
- www-webserviceX-NET
  - USZipSoap
    - GetInfoByAreaCode
    - GetInfoByCity
    - GetInfoByState
    - GetInfoByZIP
      - GetInfoByZIP 0.1
        - metadata
          - Columns(1)
        - USZip
      - GetInfoByZIPResponse 0.1

# The tXMLMap Component



Process the XML Web Service Response

- Use the **tXMLMap** component to extract useful information from the response
  - Use the response schema to populate the input table of the **tXMLMap** component

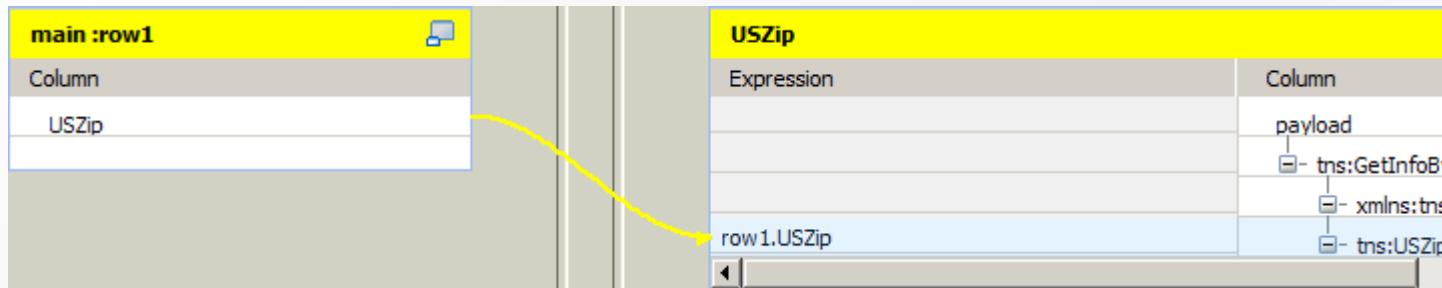


# The tXMLMap Component



Process the XML Web Service Response

- The **tXMLMap** editor is similar to the **tMap** editor when processing the response data flow
  - Provides similar capabilities for routing and transforming data from one or more data sources to one or more outputs

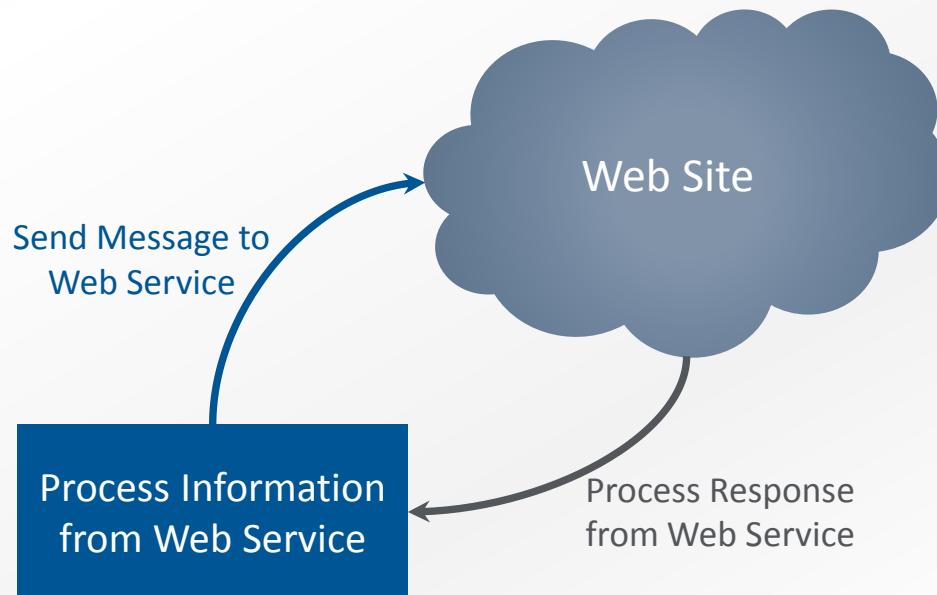


# Lab Overview



## Working with Web Services

- You will create a data integration Job that accesses a public SOAP based Web Service



# Lesson Summary

---



- In addition to reading and writing to and from files, databases, and other sources, Talend also supports accessing Web Services
  - **tESBConsumer** is a key component for accessing Web Service providers
  - **tXMLMap** can be used to transform the XML response in the data flow





# Standalone Jobs

# Objectives

---



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

# Building a Job



## ● Exporting a Job

- Bundles a Job and its dependencies into a shareable format
- Exported Job is only useable within Talend Studio
- Useful for collaboration, back-ups, or for training others



Talend Studio



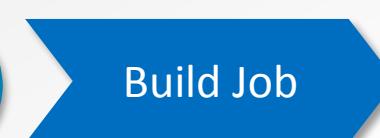
Talend Studio

## ● Building a Job

- Bundles a Job and its dependencies into a standalone format
- Job can be run anywhere (outside of Studio)
- Delivers flexible deployment options



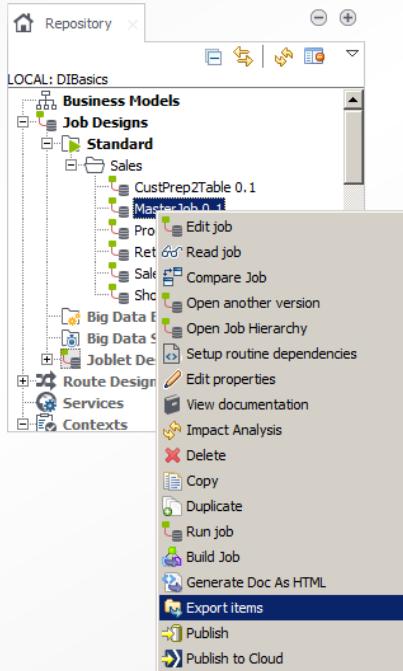
Talend Studio



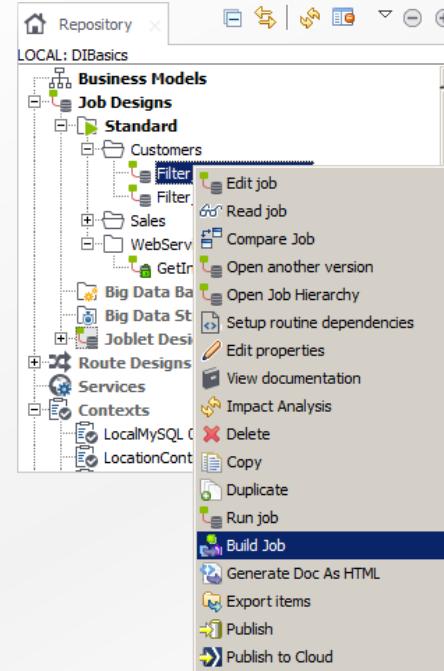
# Building a Job



## Exporting a Job



## Building a Job

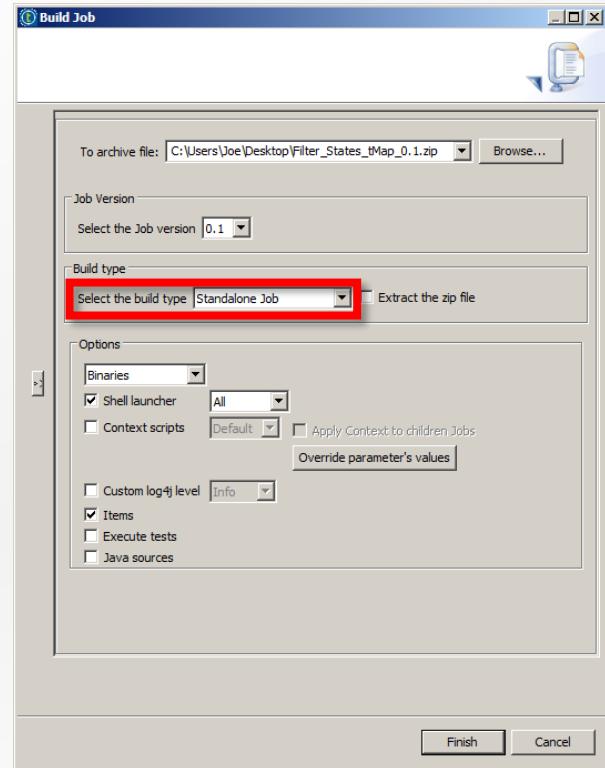


# Building a Job



## Build Type

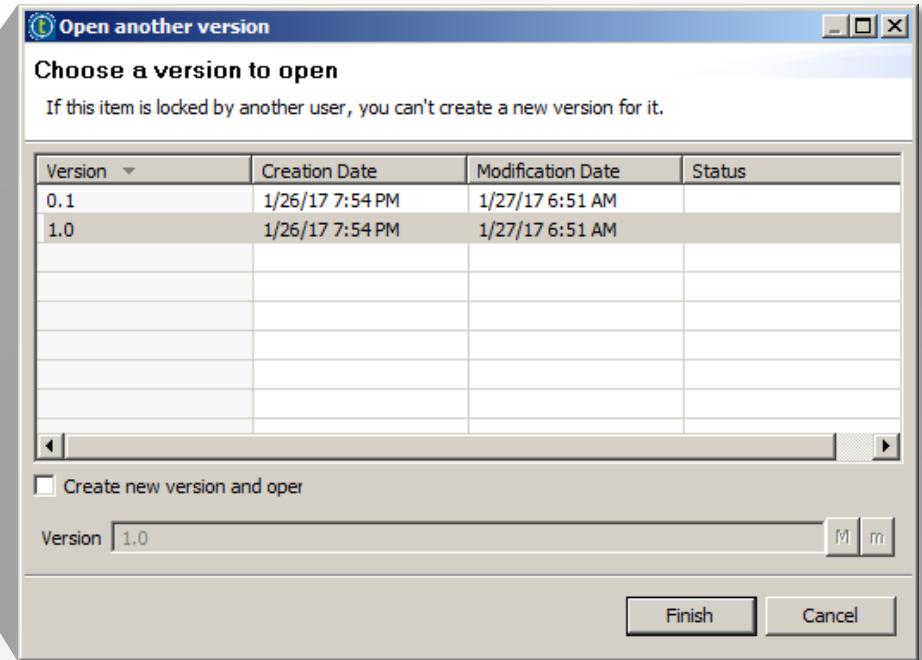
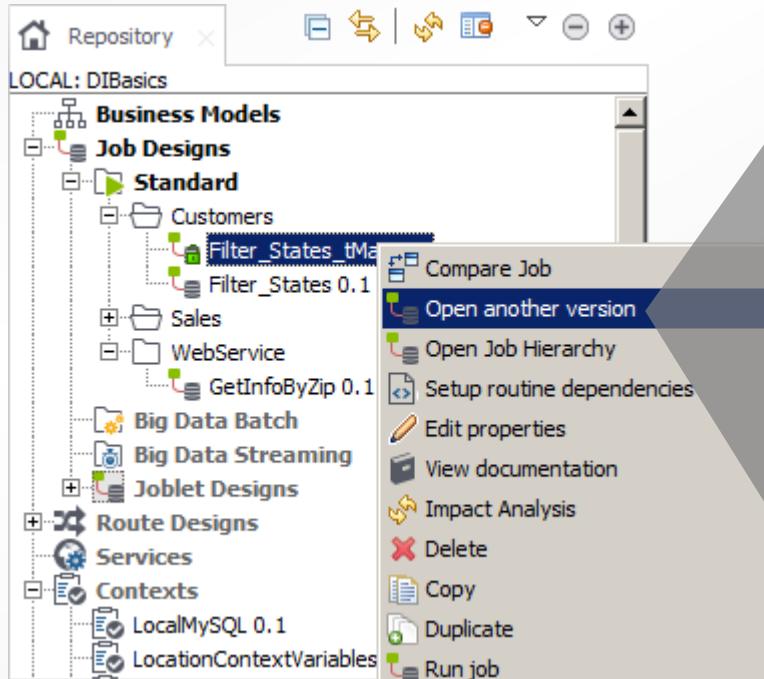
- The **Standalone Job** build type builds an archive with the following that contains:
  - **Java Archive (JAR)** files
  - Package of files needed for deployment
  - **.bat** files: Microsoft batch files for launching the Job on a Windows platform
  - **.sh** files: Unix shell scripts for launching the Job on a Linux-like platform



# Job Versions



- Studio offers basic version control (**major.minor** numbering scheme)

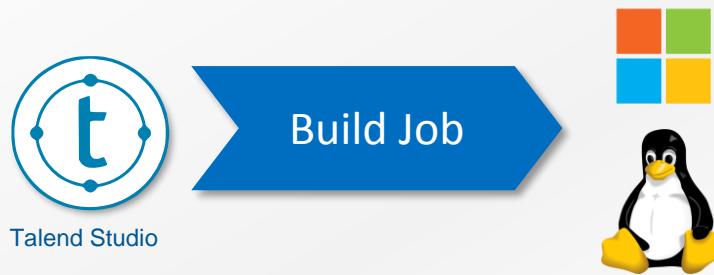


# Lab Overview



## Standalone Jobs

- Build a Job
  - Run the Job outside of the Talend Studio and validate results
- Use Studio's basic version control to create and build alternate versions of the Job



# Lesson Summary

---



- Building a Job is different than exporting one
  - Built Jobs can be run outside of the Studio, whereas exported Jobs can only be imported into an instance of Talend Studio
- Basic version control is easy to implement:
  - Create new versions
  - Open different version
  - Build Job of various versions





# Best Practices and Documentation

# Objectives

---



- Name a Job and fill out its properties using best practices
- Provide additional information for Job components
- Generate and view HTML documentation for your Job

# Scenario

---



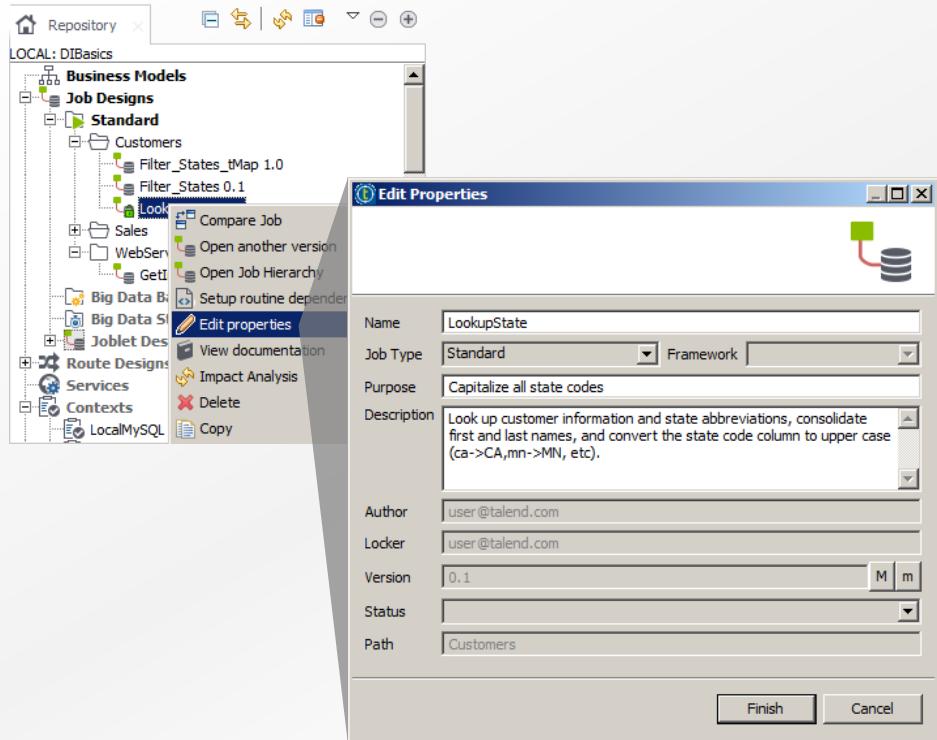
- There are a lot of inconsistencies and no documentation on Jobs developed by your team.
- Your new manager wants to implement several best practices for the development team. Most are documentation-related, from Job and component naming conventions to generating HTML documentation for others departments to view.

# Best Practices



## Naming and Describing Your Jobs

- Specify important information when creating new Jobs
  - **Name:** adopt and follow a consistent naming convention
  - **Purpose:** brief description
  - **Description:** detailed description that appears in repository hover text
- Can also add or modify at any time after creation

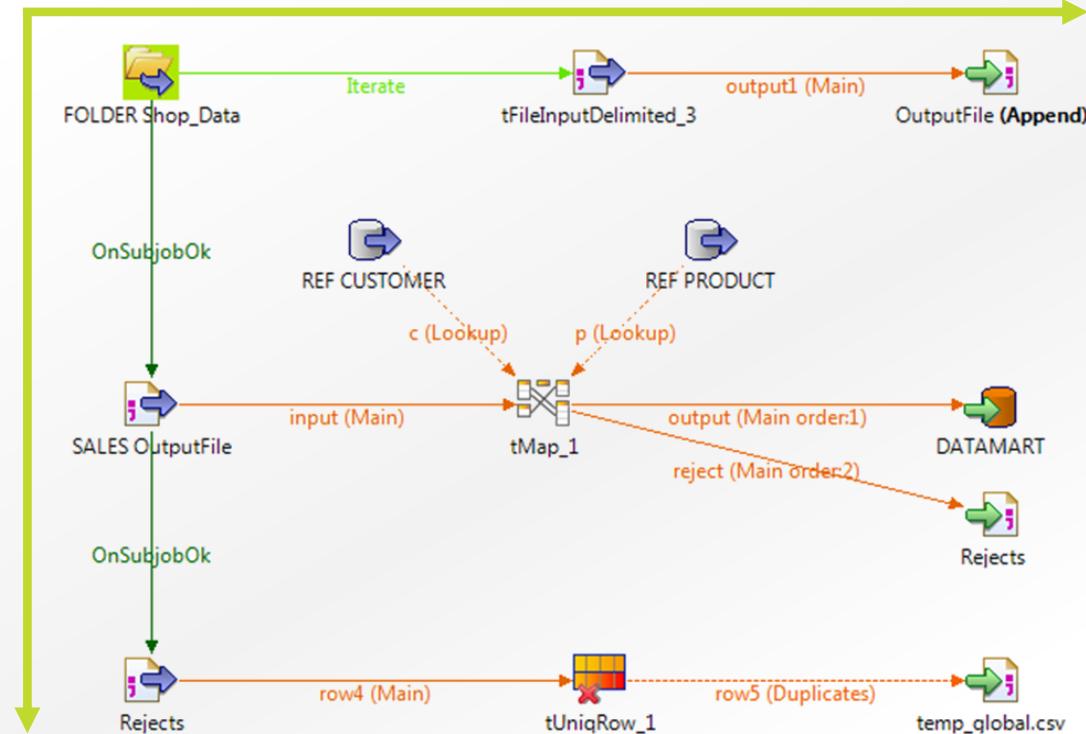


# Best Practices



## Job Layout

- Work from the top-left corner.
  - Place and connect components from left to right.
  - Work also from top to bottom.

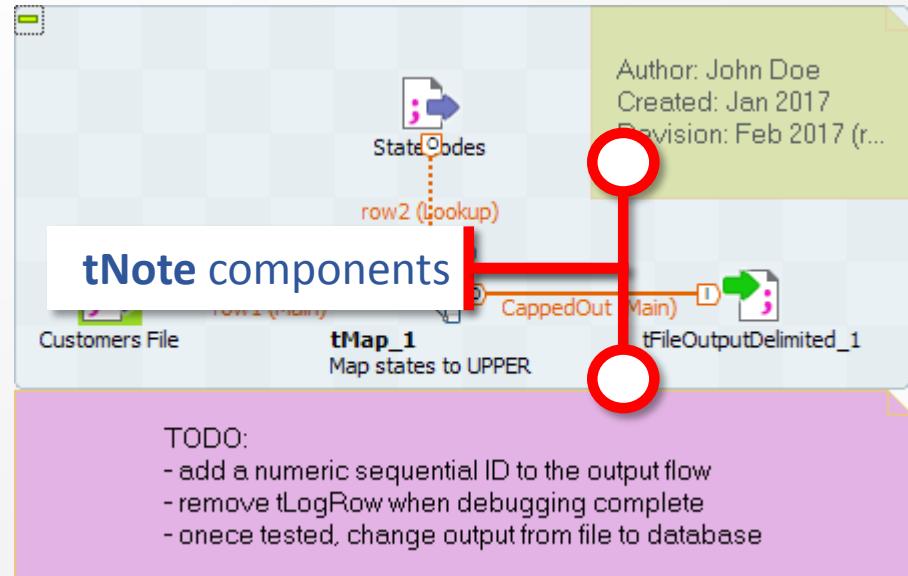


# Best Practices



## Annotating Jobs with tNote

- Place notes in the design workspace
- Explore basic format options
  - Text layout
  - Font, size, bold, italics
  - Opacity
  - Colors (text, border, fill)
  - Vertical/Horizontal alignment



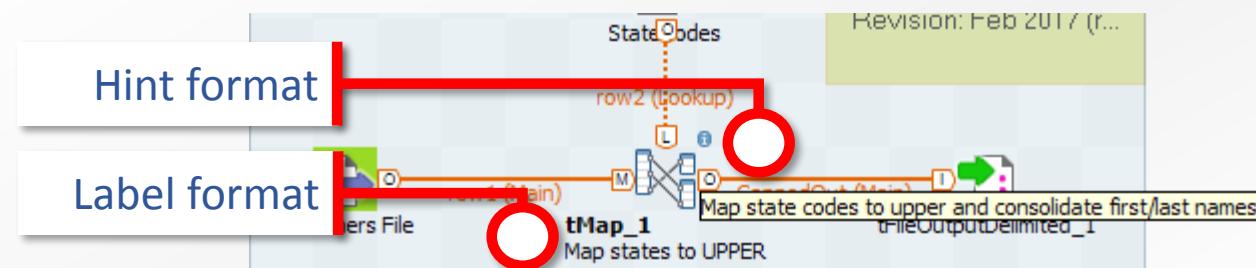
# Best Practices



## Labelling Components

- Customizing the component label improves readability in the Designer

Field	Description	Default Value
Label format	Text label shown in <b>Designer</b> (component name)	<code>_UNIQUE_NAME_</code>
Hint format	Hidden tooltip. Visible when hovering over component and <b>Show Information</b> option is selected.	<code>&lt;b&gt;_UNIQUE_NAME_&lt;/b&gt;&lt;br&gt;_COMMENT_</code>



# Documentation



## View Documentation in the Studio During Development

- Open a tab with HTML documentation for review
  - When development is done, generate a permanent version of the documentation

The screenshot shows the Talend Studio interface. On the left, the Repository view displays a tree structure of projects and designs. A context menu is open over a job named 'LookupState 0.1'. The menu items include 'Compare Job', 'Open another version', 'Open Job Hierarchy', 'Setup routine dependencies', 'Edit properties', 'View documentation' (which is highlighted in blue), 'Impact Analysis', 'Delete', and 'Copy'. To the right, a browser window titled 'Job LookupState 0.1' shows the generated documentation. The title is 'talend Jobs Generated by Talend Data Fabric'. Below the title is a table with project details:

Project Name	AUTHOR	GENERATION DATE	Talend Data Fabric VERSION
DIBasics	user@talend.com	Feb 19, 2017 6:43:40 AM	6.3.1.20161216_1026

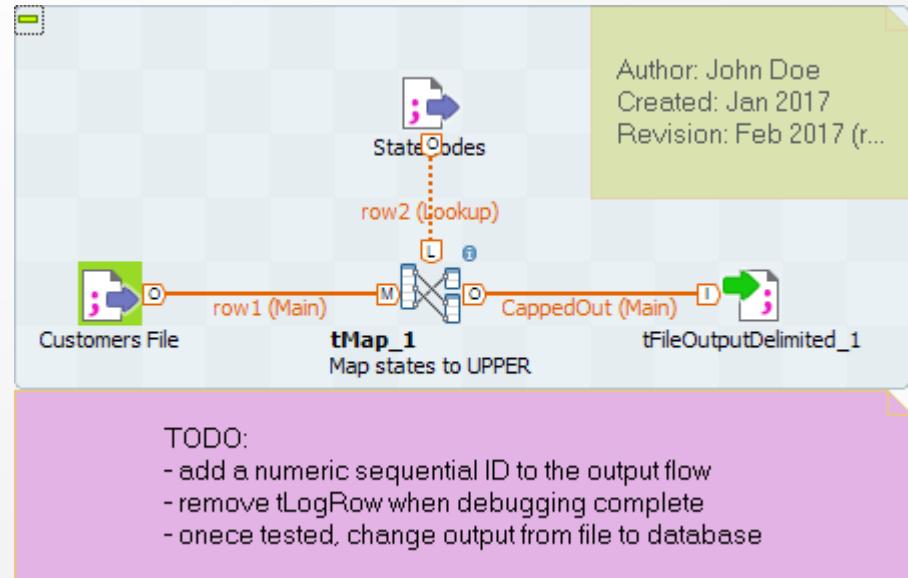
The documentation page also features a 'Summary' section and a list of links: Project Description, Description, Preview Picture, Settings, Context List, Component List, and Components Description.

# Lab Overview



## Best Practices and Documentation

- Apply several best practices while developing a Job
- View HTML Job documentation in the Studio
- Generate and view documentation outside the Studio



# Lesson Summary

---



There are several best practices to utilize during the development of a Job that help with the overall Job documentation:

- Planning
  - Business Model
- Development
  - Naming conventions, labels and hints
- Documentation
  - Generate and View HTML

