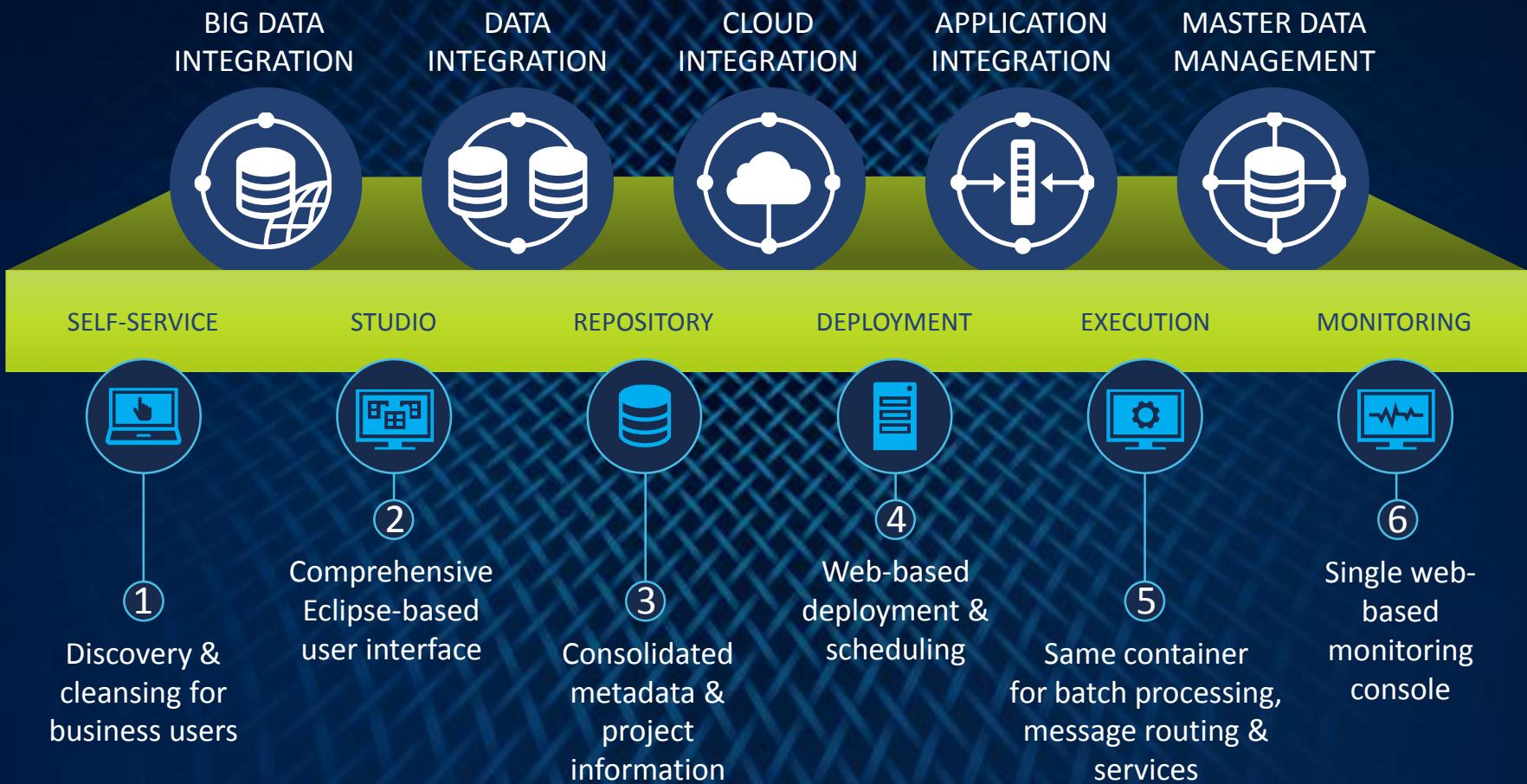




Data Integration Basics

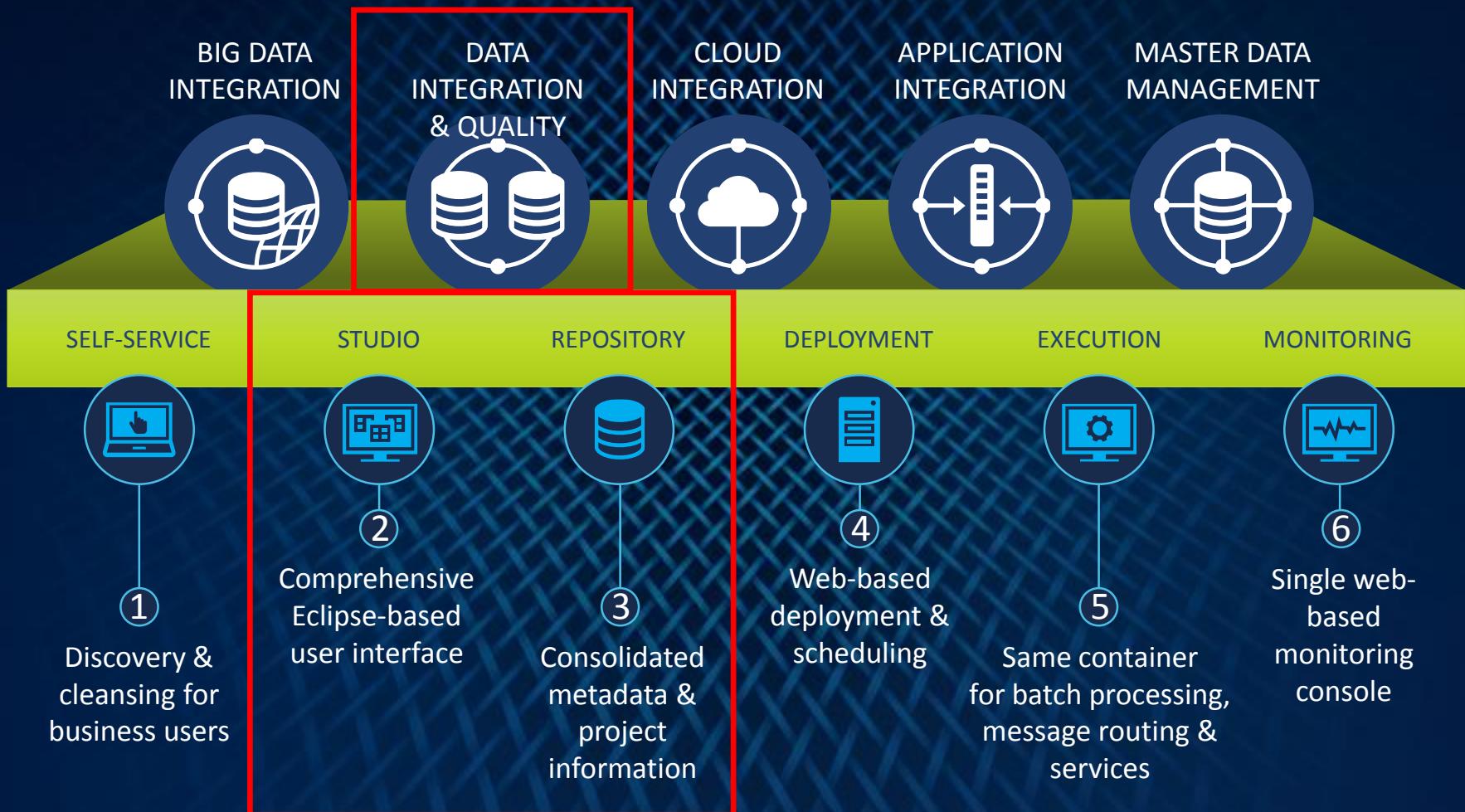
Version 6.2

talend | Data Fabric





In this course, we focus on





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. Web Services
11. Running Jobs Standalone
12. Best Practices and Documentation





Introduction to Talend Data Integration

Version 6.2



Objectives

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



Why Data Integration?

More data every day

- Different formats
- Different database structures and models
- Different standards
- Different systems

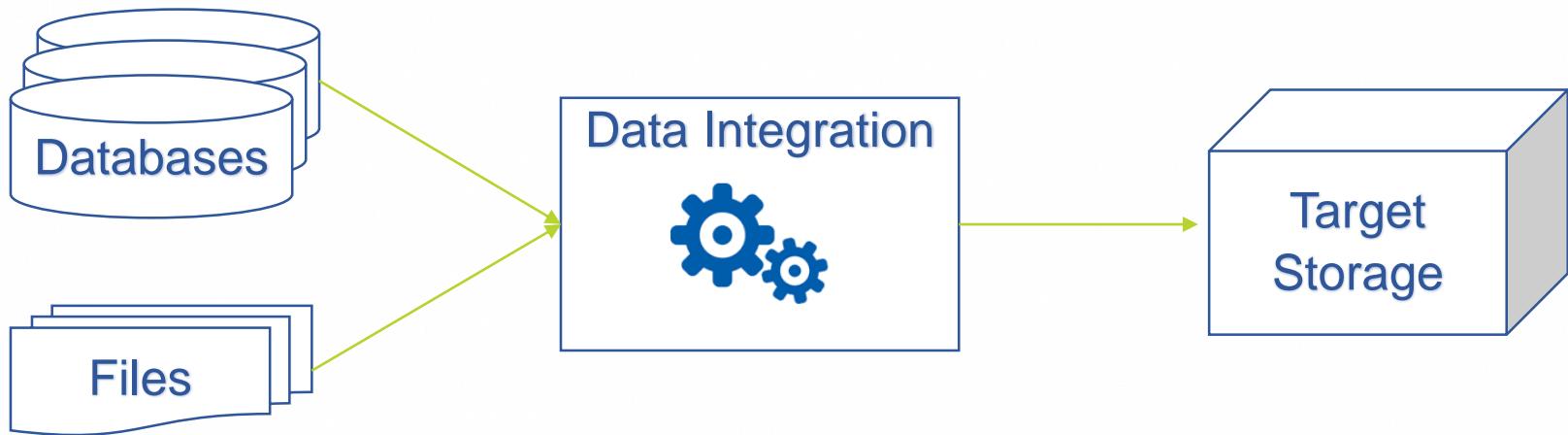
Adds value to data

- 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

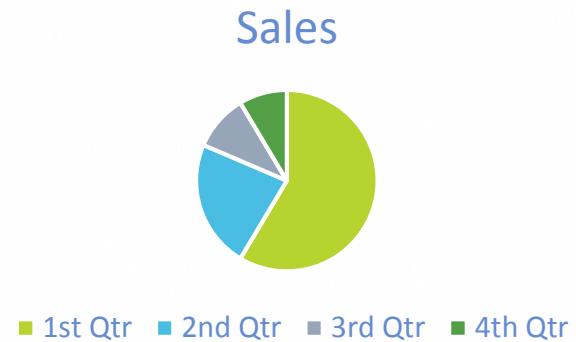
- Data Migration
- Data Warehousing
- Data Consolidation
- Data Synchronization





Typical Use Cases

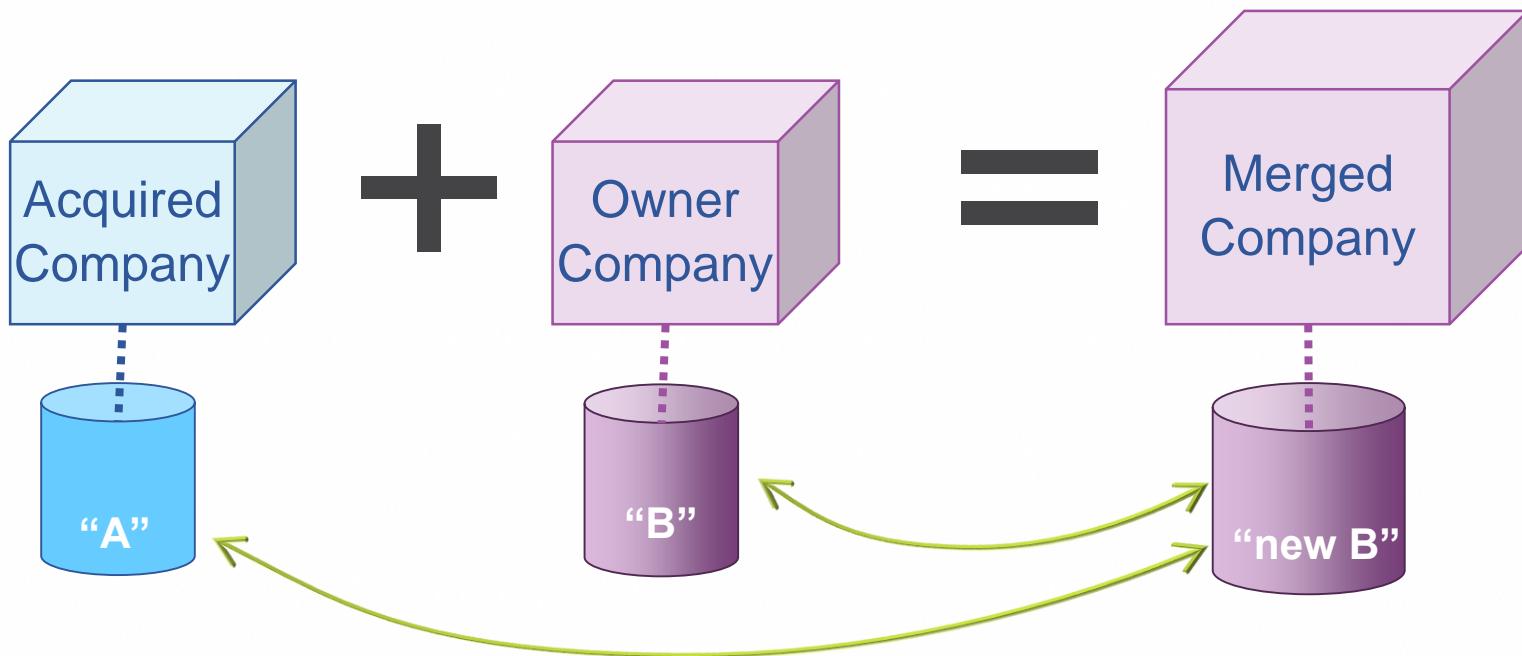
- Data Migration
- **Data Warehousing**
- Data Consolidation
- Data Synchronization





Typical Use Cases

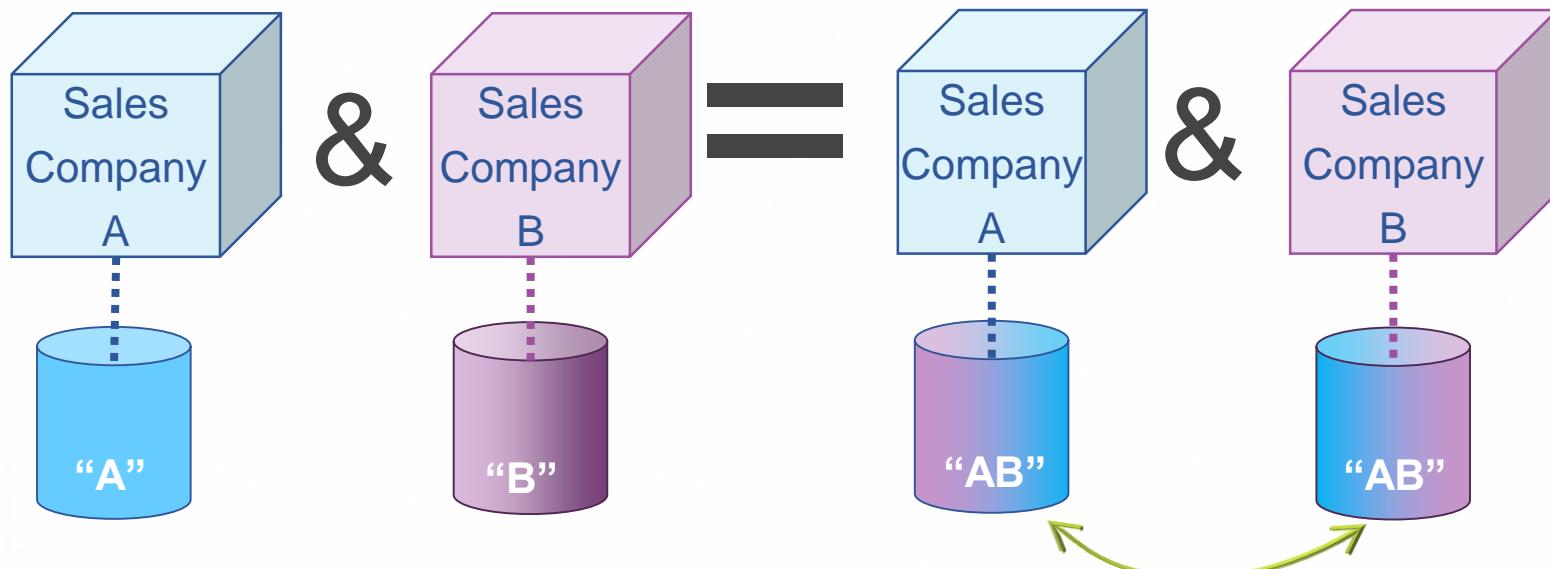
- Data Migration
- Data Warehousing
- Data Consolidation
- Data Synchronization





Typical Use Cases

- Data Migration
- Data Warehousing
- Data Consolidation
- Data Synchronization

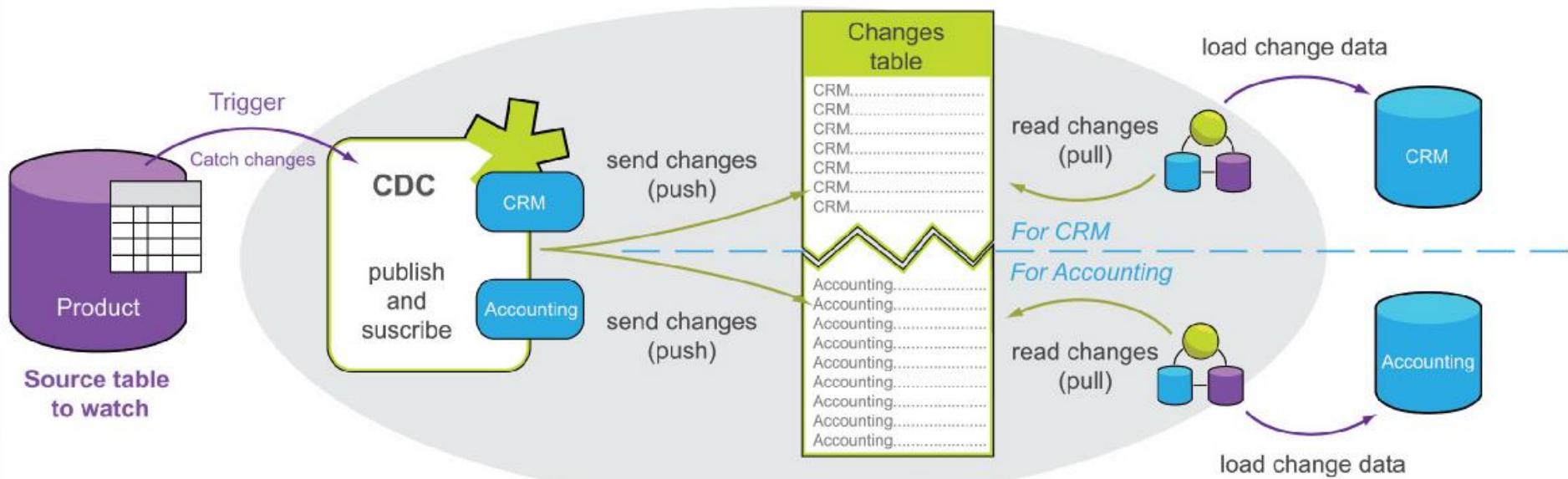




Integration Using Delta Only

Change Data Capture Scenario

- The Change Data Capture is based on the DB Journal or triggers
- A Changes table is used to track changes
- Publish / Subscribe mechanism





Key benefits of Data Integration

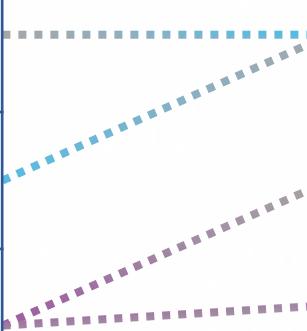
- Connectivity to source and target data systems
- Scalability and performance
- Transformation flexibility
- Data Cleansing components
- Logging and Exception Handling
- Easy integration with Web Services



Data Integration Power

- Splitting or merging fields
 - Over 800 components to assist you

First Name	John
Last Name	Smith
Credit Card	VISA 12 3456 7891



Full Name	John Smith
Card Type	VISA
Card Number	12 3456 7891



Data Integration Power

- Data Standardization
 - Dates
 - Units of measurement

Source systems



Product	Length	Date
Cable 1	5 m	25/01/2015



Product	Length	Date
Cable 2	38.2 Ft	2015-16-02

Target system



Product	Length	Date
Cable 1	5 m	25 Jan 2015
Cable 2	10 m	16 Feb 2015



Data Integration Power

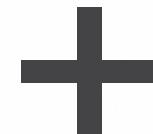
- Join Data
 - Complete the data with information from external sources

Source system



Product	Quantity	Zip Code	Customer
P167987	13	10004	C987674
P167987	16	10005	C987856

Reference file / WS



Zip Code, City
10005, New York
10004, New York
....

Target system



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



Data Integration Power

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

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



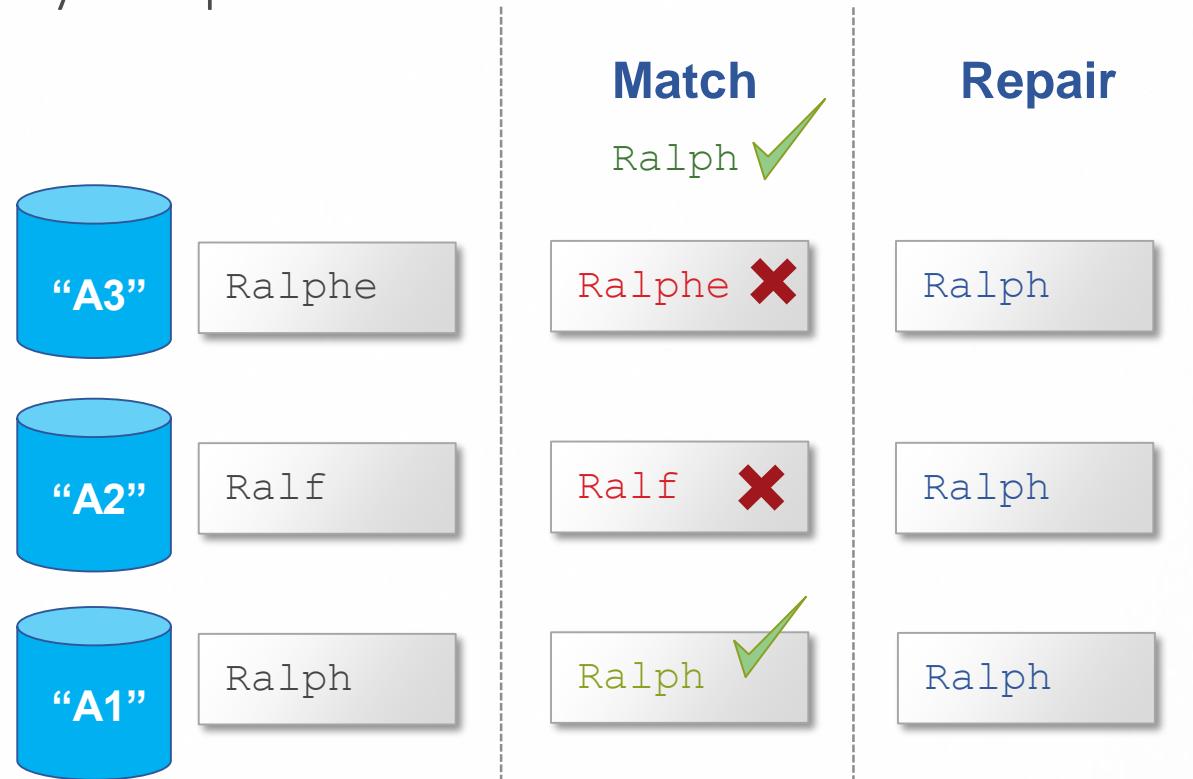
Aggregated data to be analyzed
(grouped by city and by product)

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

Design Faster



Collaborate Better



Cleanse Earlier



Manage More



Scale Easier



Use Talend Studio to design data integration jobs with a drag and drop user interface

Enable teams of developers to collaborate better using the Team Repository

Use profiling and data matching to understand and cleanse data

Centralize deployment and manage projects through a web-based Administration Console

Scale up with a high availability environment





Getting Started

Talend Studio for Data Integration



Objectives

By the end of this lesson, you will be able to:

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



Talend Studio Main Window

The screenshot shows the Talend Studio interface. The main window title is "Job HelloWorld 0.1". The central area is labeled "Designer" and displays a graphical job flow diagram:

```
graph LR; InputMessage[InputMessage] --> row1["row1 (Main)"]; row1 --> tLogRow_1[tLogRow_1]
```

A callout box highlights the "Designer" label.

Below the diagram, a callout box highlights the text "Displays the Job in graphical mode".

The bottom left pane shows the component tree with nodes: "tFixedFlowInput_1 (InputMessage)" and "tLogRow_1".

The bottom right pane is the "Properties" palette for the selected "InputMessage(tFixedFlowInput_1)". The "Basic settings" tab is active, showing:

- Schema: Built-In
- Number of rows: 1
- Mode: Use Single Table
- Values table:

Column	Value
Message	"Hello World"

The right side of the interface features a vertical "Favorites" sidebar with various categories like Business Models, Job Designs, Services, etc., and a "Palette" sidebar with a search bar and component categories such as Favorites, Recently Used, Big Data, Business Intelligence, Business, Cloud, Custom Code, Data Quality, Databases, DotNET, ETL, ESB, File, Internet, Logs & Errors, Misc, Orchestration, Processing, System, Talend Cloud, Talend MDM, Unstructured, and XML.



Talend Studio Main Window

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

- Top Bar:** File, Edit, View, Window, Help.
- Toolbar:** Learn, Ask, Exchange, Integration, Mediation, MDM.
- Left Sidebar:** LOCAL: DI_B, Business Models, Job Designs (Standard, Big Data Streaming, Big Data Batch), Services, Joblet Designs, Contexts, Code, SQL Templates, Metadata, Documentation, Recycle bin.
- Designer Area:** *Job HelloWorld 0.1*. It contains a flow with an **InputMessage** component connected to a **tLogRow_1** component. A tooltip says: "You can drag and drop components from the **Palette** to the Designer".
- Component Editor:** InputMessage(tFixedFlowInput_1). It shows basic settings like Schema (Built-In), Number of rows (1), Mode (Use Single Table), and Values (Message: "Hello World").
- Palette:** Shows a search bar "Find component..." and a list of categories: Favorites, Recently Used, Big Data, Business Intelligence, Business, Cloud, Custom Code, Data Quality, Databases, DotNET, ETL, ESB, File, Internet, Logs & Errors, Misc, Orchestration, Processing, System, Talend Cloud, Talend MDM, Unstructured, XML. A box highlights the "Palette" button.



Talend Studio Main Window

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

- Top Bar:** File, Edit, View, Window, Help.
- Toolbar:** Learn, Ask, Exchange, Integration, Mediation, MDM.
- Left Sidebar:** LOCAL: DI_B, Business Models, Job Designs (Standard, Big Data Streaming, Big Data Batch), Services, Joblet Designs, Contexts, Code, SQL Templates, Metadata, Documentation, Recycle bin.
- Middle Area:** *Job HelloWorld 0.1 X. It contains a flow diagram with an InputMessage component connected to a tLogRow_1 component. A blue box highlights the text "Run View with console output".
- Bottom Area:** Designer, Code, Jobscrip tabs. Below the tabs is a toolbar with Job(WS), Component, Run (Job HelloWorld), and Views buttons. A red arrow points from the "Views" button to the "Component View" window.
- Component View:** Shows details for the selected InputMessage component. It includes sections for Basic settings (Advanced settings, Dynamic settings, View, Documentation, Validation Rules), Schema (Built-In, Edit schema), Number of rows (1), Mode (Use Single Table selected), and Values table.
- Right Sidebar:** Palette, Find component..., Favorites (Recently Used, Big Data, Business Intelligence, Business, Cloud, Custom Code, Data Quality, Databases, DotNET, ETL, ESB, File, Internet, Logs & Errors, Misc, Orchestration, Processing, System, Talend Cloud, Talend MDM, Unstructured, XML).

Component View – configurable details of selected components



Talend Studio Main Window

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

- Top Bar:** File, Edit, View, Window, Help.
- Toolbar:** Learn, Ask, Exchange, Integration, Mediation, MDM.
- Left Sidebar (Repository):** LOCAL: DI_B, Business Models, Job Designs (highlighted), Standard, HelloWorld 0.1, Big Data Streaming, Big Data Batch, Services, Joblet Designs, Contexts, Code, SQL Templates, Metadata, Documentation, Recycle bin. A red arrow points to the "HelloWorld 0.1" item under Job Designs.
- Middle Area (Job Designer):** *Job HelloWorld 0.1*. The main area displays the text: "Organize Jobs, Metadata, and Context Variables ...". Below it are two components: InputMessage and tLogRow_1. The InputMessage component is selected. The Job Designer tab is active.
- Bottom Left (InputMessage Component Panel):** Basic settings (selected), Advanced settings, Dynamic settings, View, Documentation, Validation Rules. The Schema dropdown is set to Built-In. Number of rows: 1. Mode: Use Single Table. Values table:

Column	Value
Message	"Hello World"
- Bottom Right (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 MDM, Unstructured, XML.



Talend Studio Main Window

The screenshot shows the Talend Studio interface with several key components highlighted:

- Quick Access Toolbar:** Located at the top, it contains various icons for file operations like Save, Open, and Print, as well as links to "Learn", "Ask", and "Exchange".
- Repository View:** On the left, it shows the project structure under "LOCAL: DI_B". It includes sections for Business Models, Job Designs (with "HelloWorld 0.1" selected), Big Data Streaming, Big Data Batch, Services, Joblet Designs, Contexts, Code, SQL Templates, Metadata, Documentation, and Recycle bin.
- Job Perspective:** The central workspace displays a job named "HelloWorld 0.1". The job flow consists of an "InputMessage" component connected to a "tLogRow_1" component. A tooltip over the job area says: "Find a Job, Create a Job, Run a Job, ...".
- Palette:** On the right, the "Palette" view is open, showing a search bar "Find component..." and a list of categories: Favorites, Recently Used, Big Data, Business Intelligence, Business, Cloud, Custom Code, Data Quality, Databases, DotNET, ETL, ESB, File, Internet, Logs & Errors, Misc, Orchestration, Processing, System, Talend Cloud, Talend MDM, Unstructured, and XML.
- Bottom Panel:** The bottom panel shows the details for the currently selected component, "InputMessage(tFixedFlowInput_1)". The "Basic settings" tab is active, displaying options for Schema (Built-In), Number of rows (1), Mode (Use Single Table selected), and Values (a table with a single row: Message -> "Hello World").

Annotations:

- A red arrow points from the "Integration" icon in the Repository view to the "Integration" icon in the Top Bar.
- A red arrow points from the "Integration" icon in the Top Bar to the "Integration" icon in the Bottom Panel.
- A large blue box highlights the central workspace with the text: "Change Perspective – initial set and layout of views Integration perspective displayed here".
- A white box with a black border highlights the "Integration" icon in the Top Bar.
- A white box with a black border highlights the "Integration" icon in the Bottom Panel.



User Interface - Components

The screenshot shows the Talend Palette interface. On the left is a sidebar with a search bar at the top labeled "Find component..." with a magnifying glass icon. Below the search bar is a list of "Favorites" and "Recently Used" components. The main area is titled "File" and contains a tree view of component families. The "Input" family is expanded, showing various sub-components like tApacheLogInput, tFileInputARFF, etc. A red bracket on the right side of the palette groups the search bar and the component list.

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

Components can be found in the **Palette** using their names or keywords.

Components are organized by **family**:

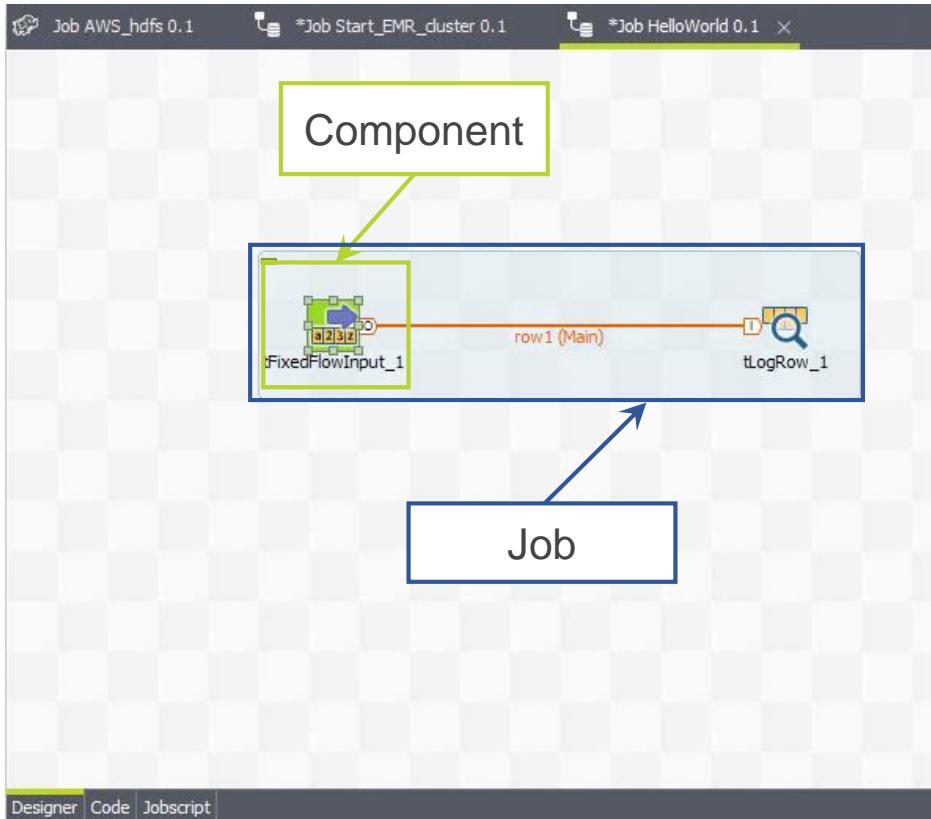
- File
- Databases
- Processing
- Logs & Errors
- Orchestration
- ...

Types inside of families:

- Input
- Output
- ...



What Is a Talend Job?



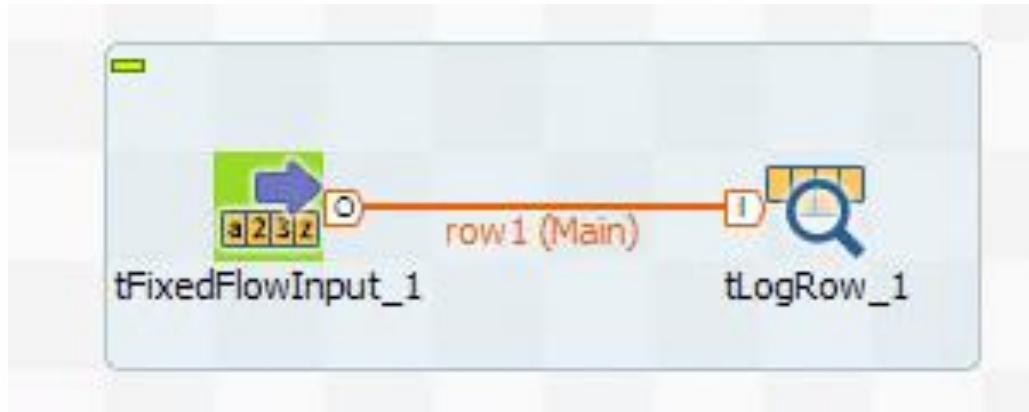
- **Component** – Executable part of a Job. Talend provides 900+ pre-built components to accomplish specific tasks.
- **Job** – One or more components connected together (graphically) that generate java code



Build a Job

Example *Hello World* Job

Display a Hello World message



- **tFixedFlowInput**

Used to define a message for an internal variable

- **tLogRow**

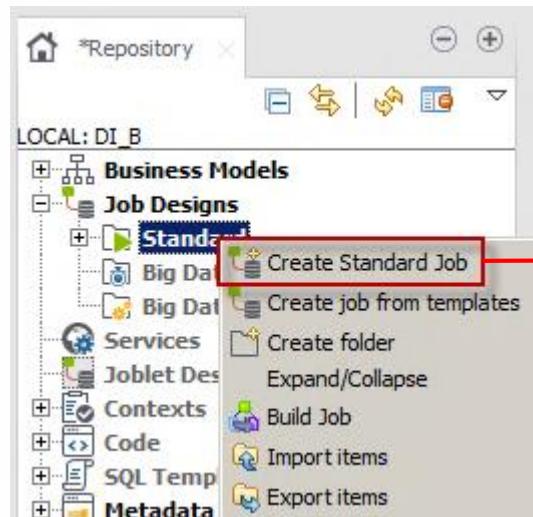
Used to display the message in the execution console



Build a Job

Create a Job

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



New Job

Add a job in the repository

Name	HelloWorld
Purpose	First DI Job
Description	DI Basic Training
Author	user@talend.com
Locker	
Version	0.1
Status	
Path	Select

Finish Cancel

- **Name:** no spaces or special characters
- Fill out both **Purpose** and **Description**



Build a Job

Add components

Two options:

1. Type the first characters of the component in the design space
 - Auto-complete function provides a list of choices (select the component)

2. Search for the component in the Palette
 - Drag and drop it to the Designer





Build a Job

Connect components

- To create a direct link between two components:
 - Right click the component
 - Select Row > Main
 - Click on the second component



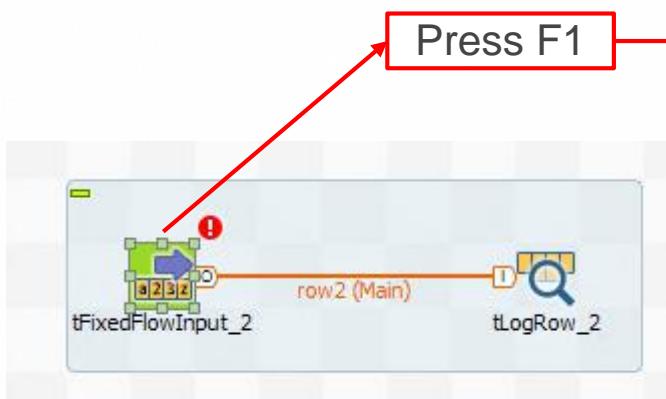
- Main rows can also be created by **holding a right click** from the first component to the second one



Build a Job

Online context sensitive help

- To open a component's Help page:
 - Select component
 - Press F1



A screenshot of the Talend Components Reference Guide. The title is 'tFixedFlowInput' under 'Chapter 20. Misc group components'. The page includes navigation links for 'Prev' and 'Next', and a search bar. Below the title, there is a section for 'tFixedFlowInput' with a blue arrow icon and the text 'tFixedFlowInput'. A table provides detailed information about the component:

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.

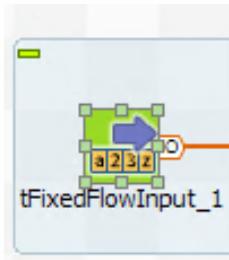


Build a Job

Rename a component

- Use the **Component View**

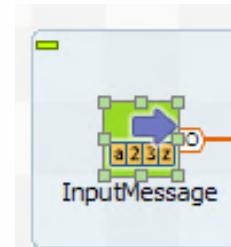
- View tab
- Change the component's name in the **Label format** box



Component

tFixedFlowInput_1

Basic settings	Label format	<code>__UNIQUE_NAME__</code>
Advanced settings	Hint format	<code>__UNIQUE_NAME__</code>
Dynamic settings	Connection format	row
View		
Documentation		
Validation Rules		



Component

InputMessage(tFixedFlowInput_1)

Basic settings	Label format	<code>InputMessage</code>
Advanced settings	Hint format	<code>InputMessage</code>
Dynamic settings	Connection format	row
View		
Documentation		
Validation Rules		

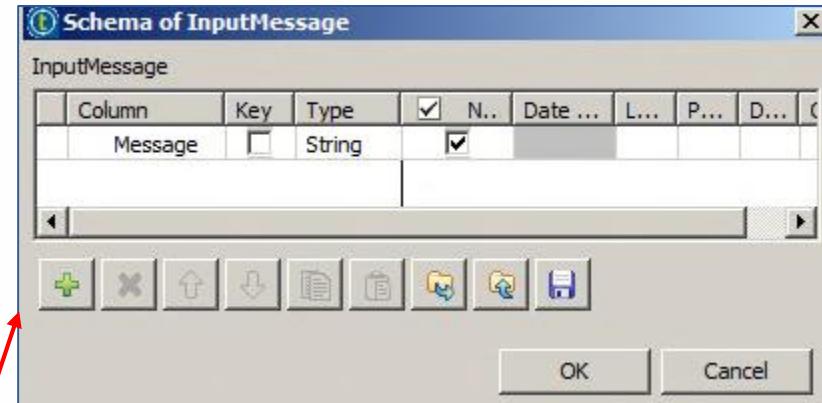


Build a Job

Configure components

- Use the **Component View**
- **Settings** are specific to each component

The screenshot shows the 'Component' view with the 'InputMessage(tFixedFlowInput_1)' component selected. The 'Basic settings' tab is active. The 'Schema' dropdown is set to 'Built-In'. A red box highlights the 'Edit schema' button, which is a small square icon with three dots. Below it, the 'Number of rows' is set to '1'. Under the 'Mode' section, 'Use Single Table' is selected. The 'Values' table contains one row with 'Message' in the Column column and '"Hello World"' in the Value column.



Example :

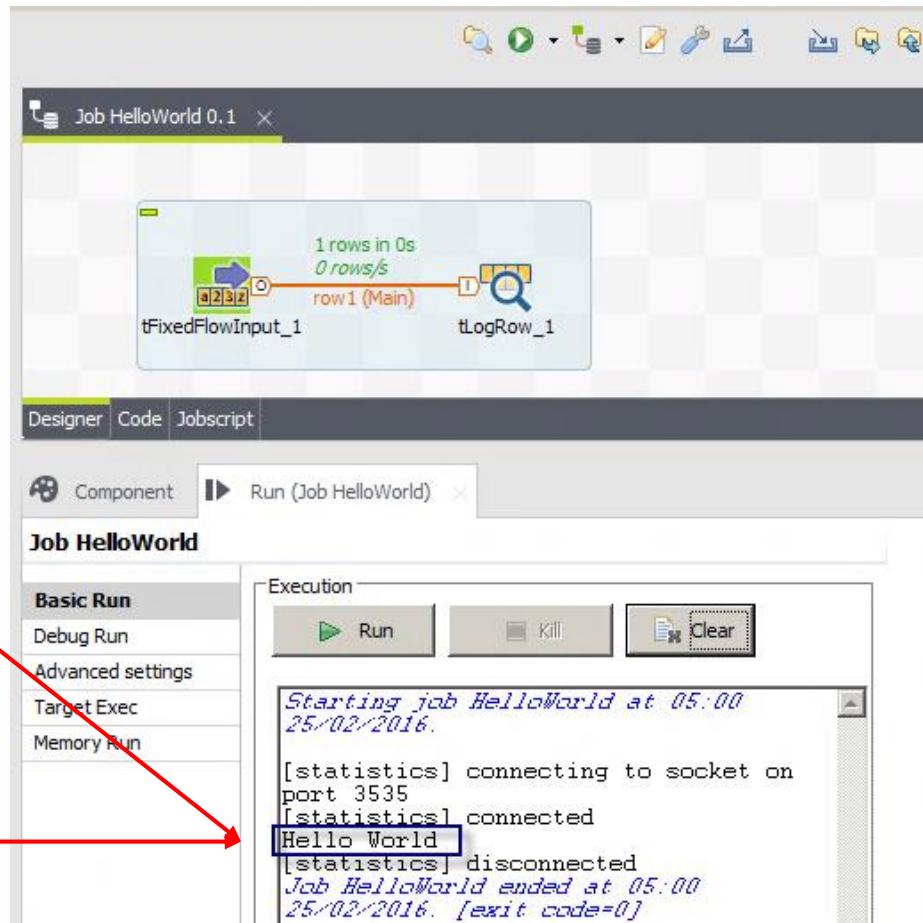
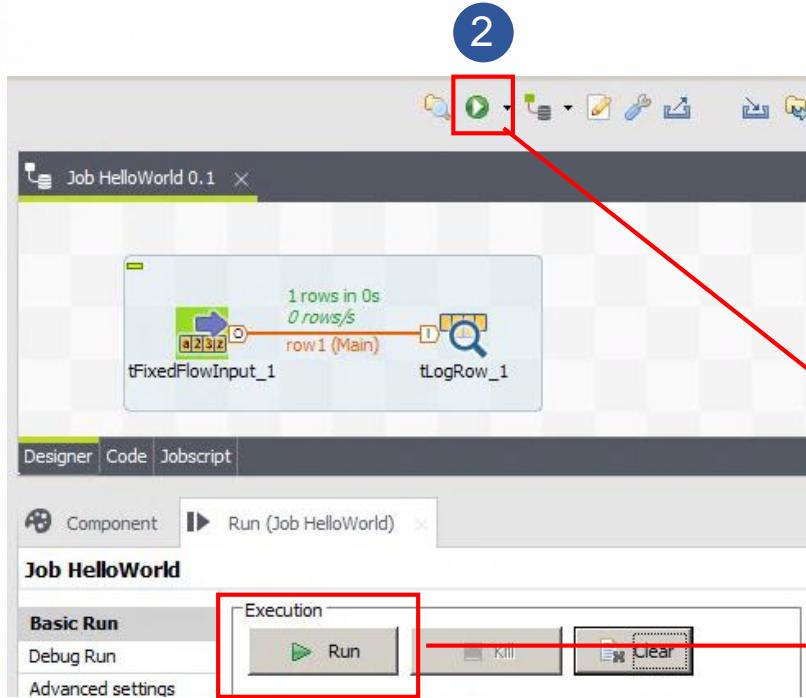
tFixedFlowInput settings

- Add a new Column to the Schema
- Define the value



Run a Job

- Two options
 - 1. Use the Run View
 - 2. Use quick access Run button





Lab overview

Discover Talend Studio and build a first Job to display a “Hello World” message

- Start Talend Studio
- Build the following Job





Lesson summary

- Talend Studio Main Window
 - Designer
 - Palette
 - Views
 - Repository
 - Quick Access Toolbar
- Build a Job in the **Designer** space
 - Use the **Palette** to find components
 - Configure components in the **Component View**
 - Use the **Run View** to explore Job's execution





Working with Files



Objectives

By the end of 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



Use Case

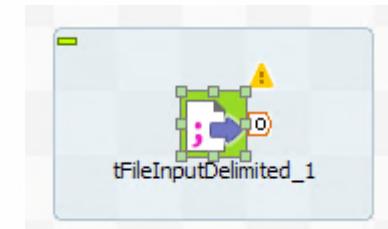
- Apply basic transformations to an input file
 - Transform a column to uppercase
 - Concatenate first name and last name into full name





Read a Delimited File

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



A screenshot of the Talend component configuration interface. The title bar shows 'Customers File(tFileInputDelimited_1)'. The left sidebar lists configuration tabs: Basic settings (selected), Advanced settings, Dynamic settings, View, Documentation, Validation Rules, Schema, and a footer section with checkboxes for 'Skip empty rows', 'Uncompress as zip file', and 'Die on error'. The 'Basic settings' tab contains the following configuration:

- Property Type: Built-In
- File name/Stream: "C:/StudentFiles/Custs.csv" (highlighted with a red box)
- Row Separator: "\n"
- Field Separator: ";" (highlighted with a red box)
- Header: 1 (highlighted with a red box)
- Footer: 0
- Limit: (empty)
- CSV options checkbox: unchecked



File Delimited Schema

- The Schema defines the structure of the file
 - Built-In Type
Schema defined locally for the selected component only (**tFileInputDelimited**)

Customers File(tFileInputDelimited_1)

Basic settings

Property Type: Built-In

"When the input source is a stream or a zip file, footer and random:

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

Row Separator: "\n"

CSV options

Header: 1 Footer: 0

Schema: Built-In **Edit schema**

Skip empty rows Uncompress as zip file Die on error

Schema of Customers File

Customers File

Column	Key	Type	N..	Date...	L...	P...	D...
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

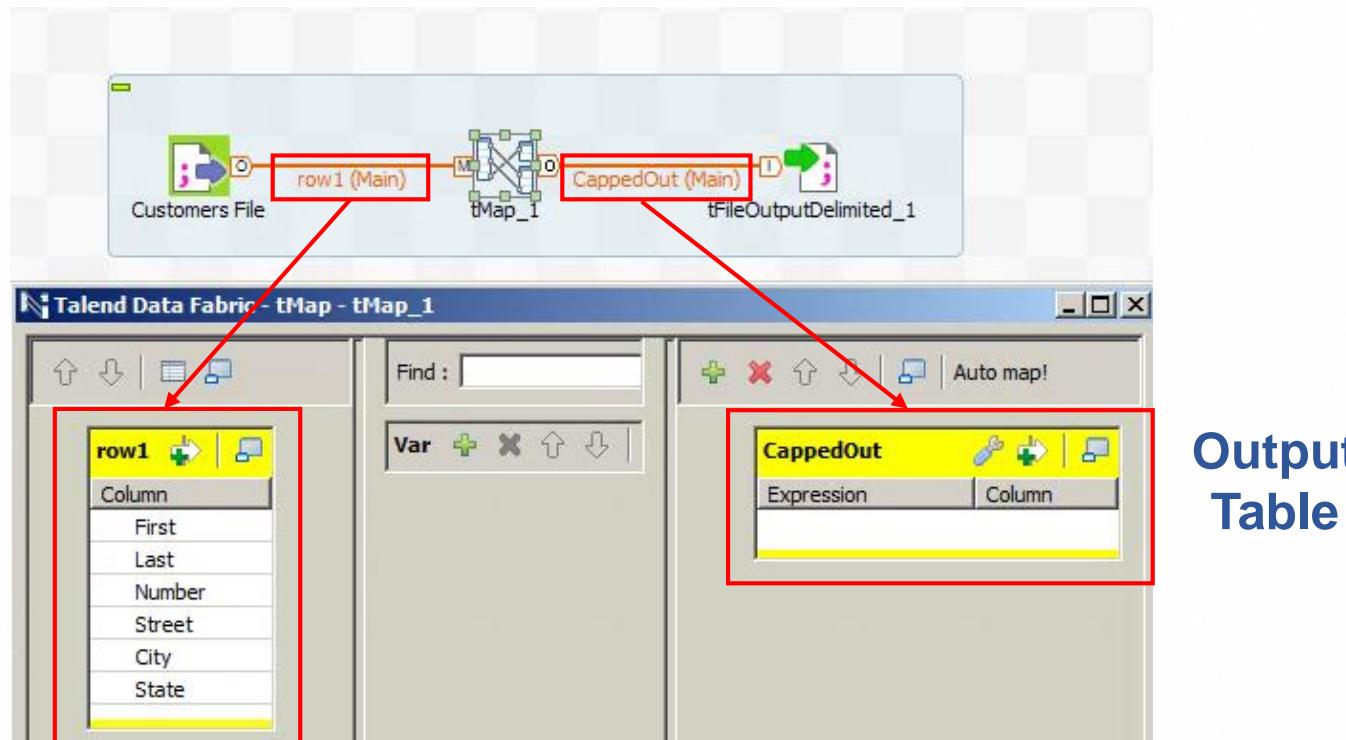
OK Cancel



Transforming Data

Input & Output Table

- Use a **tMap** component
 - Always between an input and an output component
 - Table name = name of the Main Row



**Input
Table**

**Output
Table**



Transforming Data

Map rows

- Drag and drop input columns to the **Output Table**

The screenshot shows the Talend Data Fabric interface for a tMap component named 'tMap_1'. The interface is divided into two main sections: 'Input' (left) and 'Output' (right).

Input Section: Contains a table titled 'row1' with columns: First, Last, Number, Street, City, State.

Output Section: Contains a table titled 'CappedOut' with columns: First, Number, Street, City, State.

Bottom Navigation: Buttons include: Schema editor (highlighted with a red box), Expression editor, Apply, Ok, Cancel.

Schema Editor (Left): A table titled 'row1' defining the schema for the input data. It has columns: Column, Key, Type, N.., Dat..., L..., P... with values: First, String, checked, 15; Last, String, checked, 15; Number, String, checked, 10; Street, String, checked, 20; City, String, checked, 20; State, String, checked, 2.

Output Table Schema (Right): A table titled 'CappedOut' defining the schema for the output data. It has columns: Column, Key, Type, N.., Da..., L..., P..., D with values: First, String, checked, 15; Nu..., String, checked, 10; Street, String, checked, 20; City, String, checked, 20; State, String, checked, 2.

- **Schema Editor –** allows schema updates



Transforming Data

Map Rows - Build Expressions

- Use pre-defined functions to build expressions

The screenshot illustrates the process of building an expression in the Talend Expression Builder. On the left, the 'CappedOut' component map shows a row for 'row1.State' with a red box around the '...' button. An arrow points from this button to the 'Expression Builder' window on the right.

Expression Builder Window:

- Expression:** StringHandling.UPCASE(row1.State)
- Test:** Var: row1.State, Value: NULL
- Categories:** Numeric, Relational, SQLite, **StringHandling** (selected), TalendDataGenerator, TalendDate, TalendString
- Functions:** SPACE(int i) : id_String - String, SQUOTE(String string) : id_String, STR(String string, int int) : id_String, TRIM(String string) : id_String, **UPCASE(String string) : id_String** (selected)
- Help:** Converts all lowercase letters in an ex...
- Buttons:** Test!, Clear, Ok, Cancel



Transforming Data

Map Rows - Build Expressions

- Use operators and input columns to build expressions

The diagram illustrates the process of building expressions for a mapping table. A red arrow points from the Expression Builder window to the 'Expression' column of the 'CappedOut' table.

CappedOut

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

Expression Builder

Expression: `row1.First + " " + row1.Last`

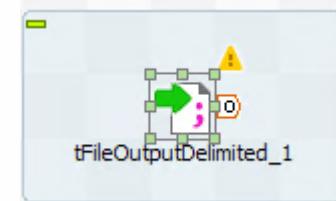
Operators:

- + - * /
- = = < <= != > = >
- and or not ()



Write to a Delimited File

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



A screenshot of the Talend component configuration interface for "tFileOutputDelimited_1". The interface has a tabbed navigation bar at the top: Component, Run (Job UpperCase), and two icons for saving and closing. Below this is a toolbar with a component icon, a save icon, and a refresh icon. The main area is titled "tFileOutputDelimited_1" and contains several settings sections:

- Basic settings**: Property Type is set to "Built-In".
 - Use Output Stream
- Dynamic settings**: File Name is set to "C:/StudentFiles/CappedOut.csv". The "File Name" input field is highlighted with a red border.
- View**: Row Separator is set to "\n" and Field Separator is set to ";".
- Documentation**: Append is unchecked, Include Header is checked, and Compress as zip file is unchecked.
- Validation Rules**: Schema is set to "Built-In". Buttons for Edit schema and Sync columns are available.



Data Viewer

- Available for both Input and Output components
- Allows quick access to data

The screenshot shows the Talend Data Integration environment. On the left, a component palette displays a 'Main' job with a 'tFileOutputDelimited' component. A context menu is open over the 'tFileOutputDelimited' component, listing options like 'Row', 'Trigger', 'Undo', 'Redo', 'Copy', 'Paste', 'Delete', 'Select All', 'Deactivate' (for the component and current subjob), 'Add breakpoint', 'Settings', 'Hide subjob', 'Find Component in Jobs', 'Data viewer' (which is highlighted with a red box), 'Create Test Case', and 'Refactor to Joblet'. A red arrow points from the 'Data viewer' menu option to the 'Data Preview' window on the right. The 'Data Preview' window title is 'Data Preview: tFileOutputDelimited_1'. It has tabs for 'Result Data Preview' (selected) and 'File Content'. It shows a preview of 30 rows per page with columns: Null, Condition, Name, Number, Street, City, and State. The data includes historical US presidents. Navigation buttons at the bottom include 'First', 'previous', 'next', 'last', and '1 page of 4'. Buttons at the bottom right are 'Set parameters and continue' and 'Close'.

Null	Condition	Name	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
	*	Timmy Polk	76143	Richmond Hill	Salt Lake City	AK



Transforming data

- Results

The diagram illustrates a data transformation process. On the left, a window titled "Data Preview: tFileInputDelimited_1" shows a table of historical U.S. Presidents with columns: Null, Condition, First, Last, Number, Street, City, and State. The "State" column is highlighted with a red border. On the right, a window titled "Data Preview: tFileOutputDelimited_1" shows the same data, but the columns are rearranged: Null, Condition, Name, Number, Street, City, and State. The "Name" column is highlighted with a red border. A large green arrow points from the input window to the output window, indicating the transformation mapping.

Data Preview: tFileInputDelimited_1

Null	Condition	First	Last	Number	Street	City	State
1	*	Bill	Coolidge	85013	Via Real	Austin	IL
2	*	Thomas	Coolidge	63489	Lindbergh Blvd	Springfield	ca
3	*	Harry	Ford	97249	Monroe Street	Salt Lake City	ca
4	*	Warren	McKinley	82589	Westside Freeway	Concord	ak
5	*	Andrew	Taylor	29886	Padre Boulevard	Madison	CA
6	*	Ulysses	Coolidge	98646	Bayshore Freeway	Columbus	MN

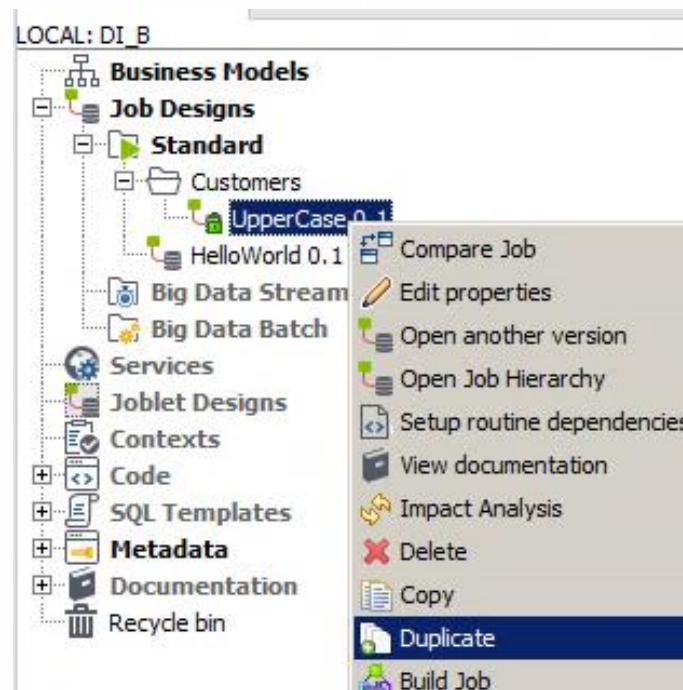
Data Preview: tFileOutputDelimited_1

Null	Condition	Name	Number	Street	City	State
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



Duplicate a Job

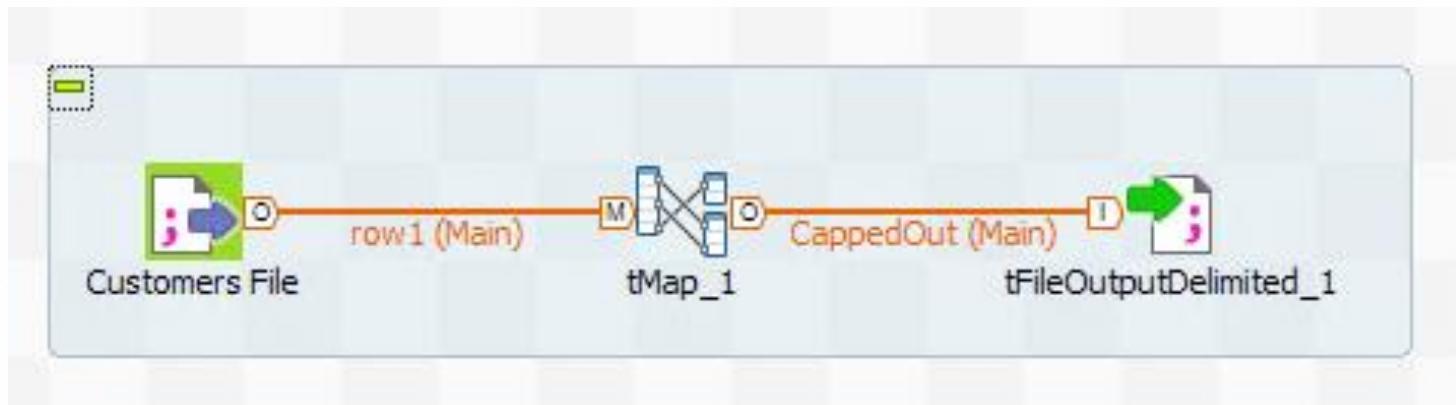
- Allow reuse of simple Jobs in a more complex scenario
- Right-click the Job and select **Duplicate**





Lab overview

Build the following Job:



- Run Job and examine/confirm results
- Duplicate the Job



Lesson summary

- **tFileInputDelimited**

- Reads a delimited file

- **tFileOutputDelimited**

- Writes a delimited file

- **tMap**

- Allows data transformation using operators and predefined functions





Joining Data Sources

Multiple Data Sources



Lesson Objectives

By the end of this lesson, you will be able to:

- Store metadata centrally for use in other components/Jobs
- Use metadata
- Join two data sources
- Troubleshoot Join – handle rejects
- Log join rejects in the console



Create Metadata

The screenshot shows the Talend Data Integration interface. In the top left, there's a 'Repository' tab. Below it, the 'LOCAL: DI_B' workspace is visible. On the left, a tree view shows various connection types: SQL Templates, Metadata, Db Connections, SAP Connections, File delimited, File positions, File regex, File xml, File Excel, File Idif, File Json, LDAP, Salesforce, UN/EDIFACT, Generic schemas, Talend MDM, Rules Management, Hierarchical Mapper, NoSQL Connections, Hadoop Cluster, Web Service, Validation Rules, FTP, and HL7. The 'File delimited' node is currently selected, and a context menu is open over it. The menu items are: Create file delimited (which is highlighted in blue), Import connections from CSV, Create folder, Expand/Collapse, Import items, and Export items.

- **Metadata**

stores **configuration information** that can be **reused** in multiple components

- **File Delimited - stores file information:**

- File Name
- Separator & Header information
- Schema (fields, types, length)

This screenshot shows the 'File delimited' node expanded in the tree view. Under 'File delimited', there is a sub-node named 'StateCodes 0.1'. This node has a child node named 'StateCodesSchema'. Other nodes under 'File delimited' include 'File positions' and 'File positional'.

Description of the Schema						
Column	Key	Type	N..	D...	Length	
StateCode		String	<input checked="" type="checkbox"/>		2	
StateName		String	<input checked="" type="checkbox"/>		20	



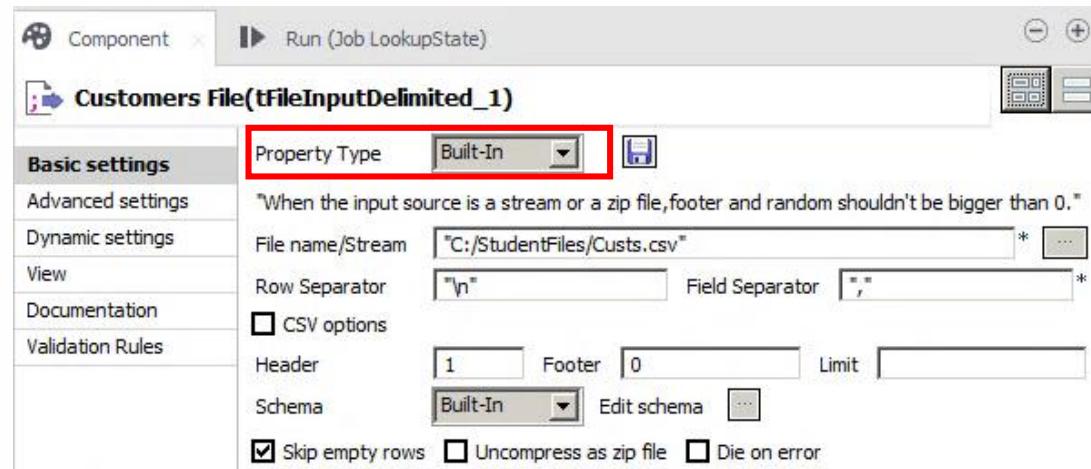
Use Metadata

Repository Schema Type

- Schema/Property types:

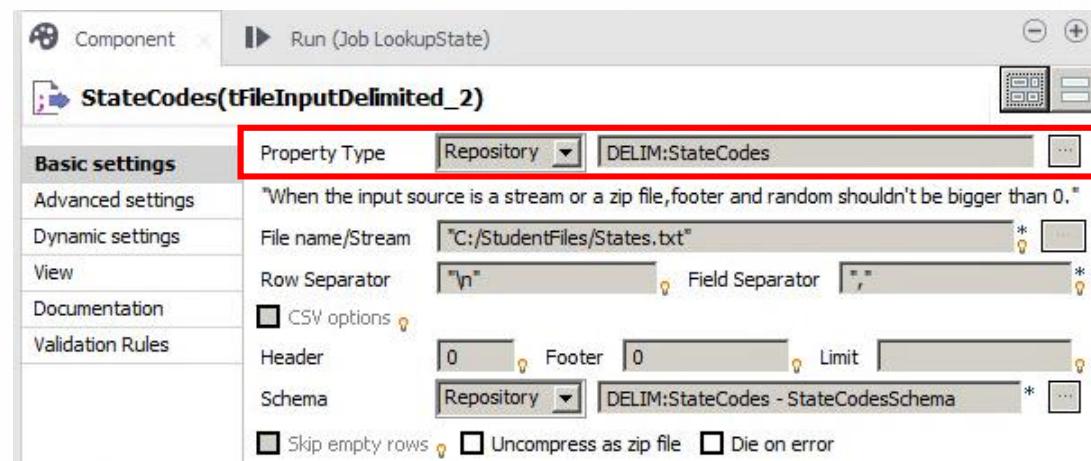
- **Built-In**

- Schema is defined **locally** for the components
- Details are defined **locally**



- **Repository**

- Metadata is referenced
- Details are predefined

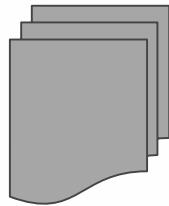




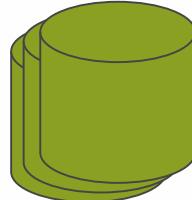
Joining Data Sources

Use tMap to join and transform multiple inputs to multiple outputs

1 or more files



Input



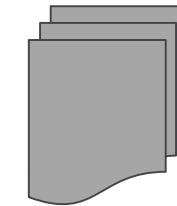
1 or more databases

Map/transform

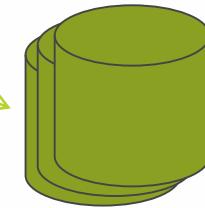
- Multiple inputs
- Multiple outputs
- Table joins
- Join failures



tMap is often part of a solution to various data challenges



Output (transformed)





Joining Data Sources

Join Types

- Join - Combines rows from two or more tables

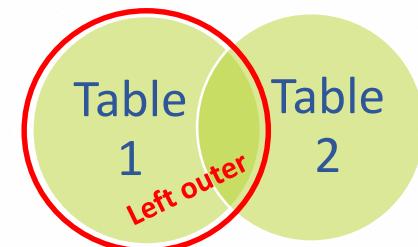
- Inner Join

- Returns rows that are in *both* tables



- Left Outer Join (or left join)

- Returns *all records* from the left table (Table 1) even if a match is not found in the right table (Table 2)... but with NULL in the column for right table





Joining Data sources

Inner Join or Left Outer Join?

Id	Name	Gender
1	Shong	M
2	Ross	M
3	Sabrina	F
4	Elisa	F



Inner Join

Id	Name	Email
1	Shong	shong@talend.com
4	Elisa	elisa@talend.com

Left Outer Join

User_id	Name	Country	Email
1	Shong	CN	shong@talend.com
4	Elisa	FR	elisa@talend.com

Id	Name	Email
1	Shong	shong@talend.com
2	Ross	null
3	Sabrina	null
4	Elisa	elisa@talend.com



Join data using tMap

Define Multiple Input Sources

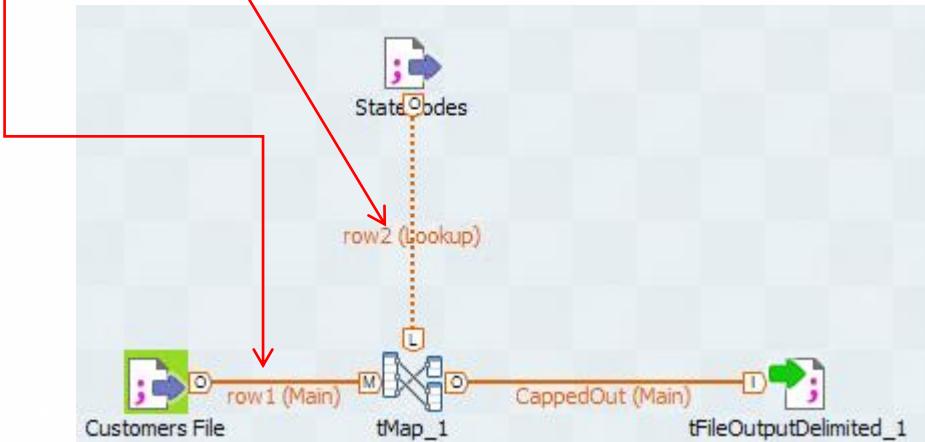
Configure Join

Define Multiple Output Tables

Handle Rejects

Display Rejects

- tMap accepts multiple input sources to join
 - Main flow (only **one** can be Main row)
 - Lookup flow (the rest of the input rows)





Join data using tMap

Define Multiple Input Sources

Configure Join

Define Multiple Output Tables

Handle Rejects

Display Rejects

1. Join data
2. Expand Model properties
3. Define Join Model: Inner Join or Left Outer Join

The screenshot shows the Talend Data Fabric interface for a tMap component named 'tMap_1'. The interface is divided into several sections:

- Row Definitions:** 'row1' contains columns for First, Last, Number, Street, City, and State. 'row2' contains columns for Property and Value.
- Join Model Configuration:** The 'Property' column of 'row2' has a red border around its rows, and the 'Value' column has a red border around its first two rows. A blue circle labeled '1' points to the 'Join Model' row in 'row2', which is set to 'Inner Join'. A blue circle labeled '2' points to the 'Value' column header of 'row2'. A blue circle labeled '3' points to the 'Value' column header of 'row2'.
- Expr. key:** A mapping section where 'row1.State' is mapped to 'StateCode' and 'StateName'.



Join data using tMap

Define Multiple Input Sources

Configure Join

Define Multiple Output Tables

Handle Rejects

Display Rejects

Add
Output
Table

The screenshot shows the Talend tMap component configuration window. It features two main sections: 'CappedOut' and 'JoinFailures'.
CappedOut: This section contains a table mapping expressions to columns. The mappings are:

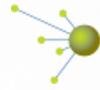
Expression	Column
row1.First + " " + row1.Last	Name
row1.Number	Number
row1.Street	Street
row1.City	City
StringHandling.UPCASE(row1.State)	State
row2.StateName	StateName

JoinFailures: This section contains a table with properties and their values. The 'Schema Type' property is highlighted with a red box.

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

Output table Schema Type can be:

- Built-in
- Repository (reference metadata)



Join data using tMap

Define Multiple Input Sources

Configure Join

Define Multiple Output Tables

Handle Rejects

Display Rejects

- Inner Join Rejects can be automatically caught using the following property:

Example failures/rejects output table

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

Expression	Column
row1.Last	Last
row1.State	State

Options

true
false

OK Cancel



Join data using tMap

Define Multiple Input Sources

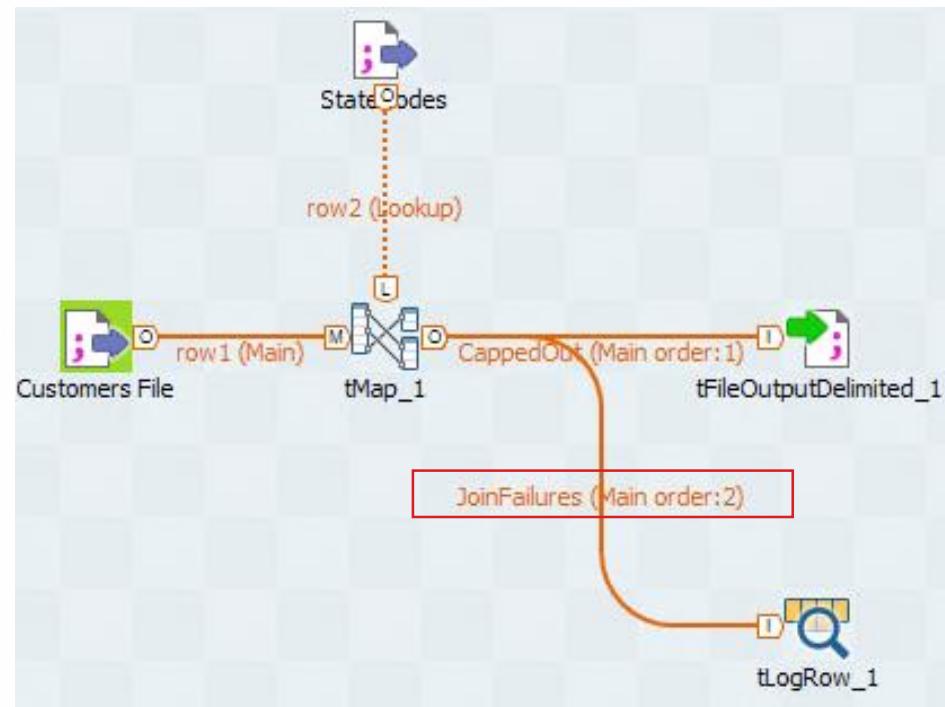
Configure Join

Define Multiple Output Tables

Handle Rejects

Display Rejects

- Inner Join Rejects can be displayed in a separate output component



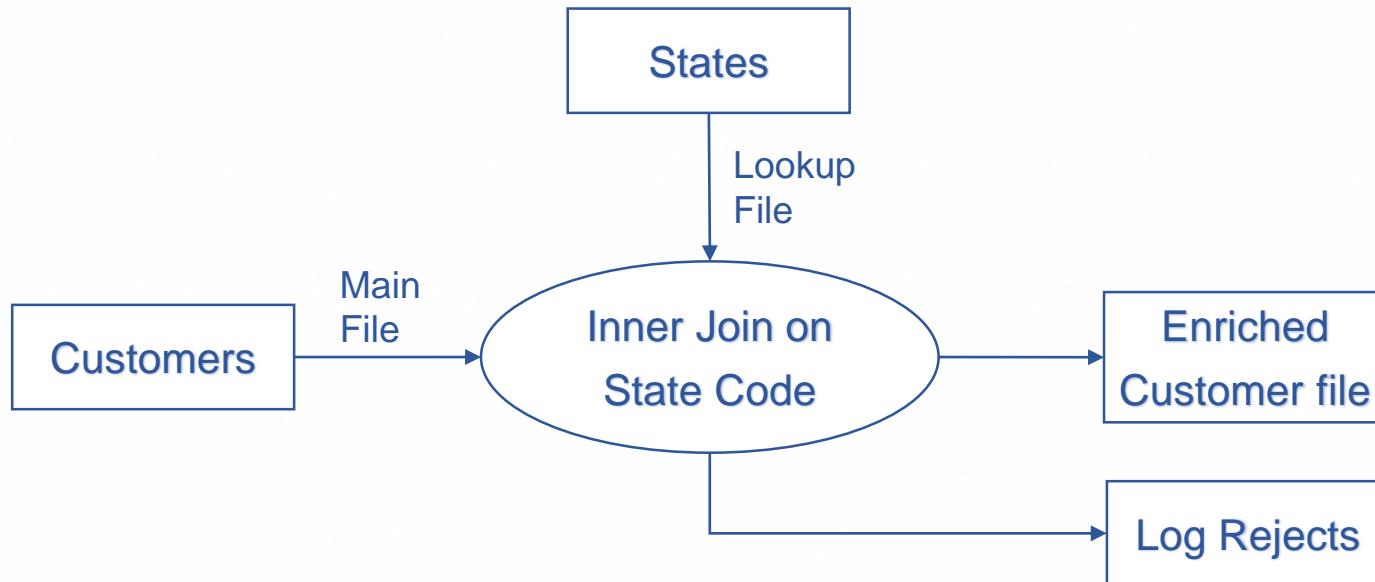


Lab overview

Joining multiple data sources

Enrich customer data file with state names coming from an external file:

- Create file delimited repository metadata for states file
- Build the following Job:



- Run Job and examine/confirm results



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)
- Can **Join** data sources (inner, left outer)
- Can perform **complex transformations**
- Catches **rejects**
- Output tables accept both **Built-In & Repository** Schemas





Filtering Data



Objectives

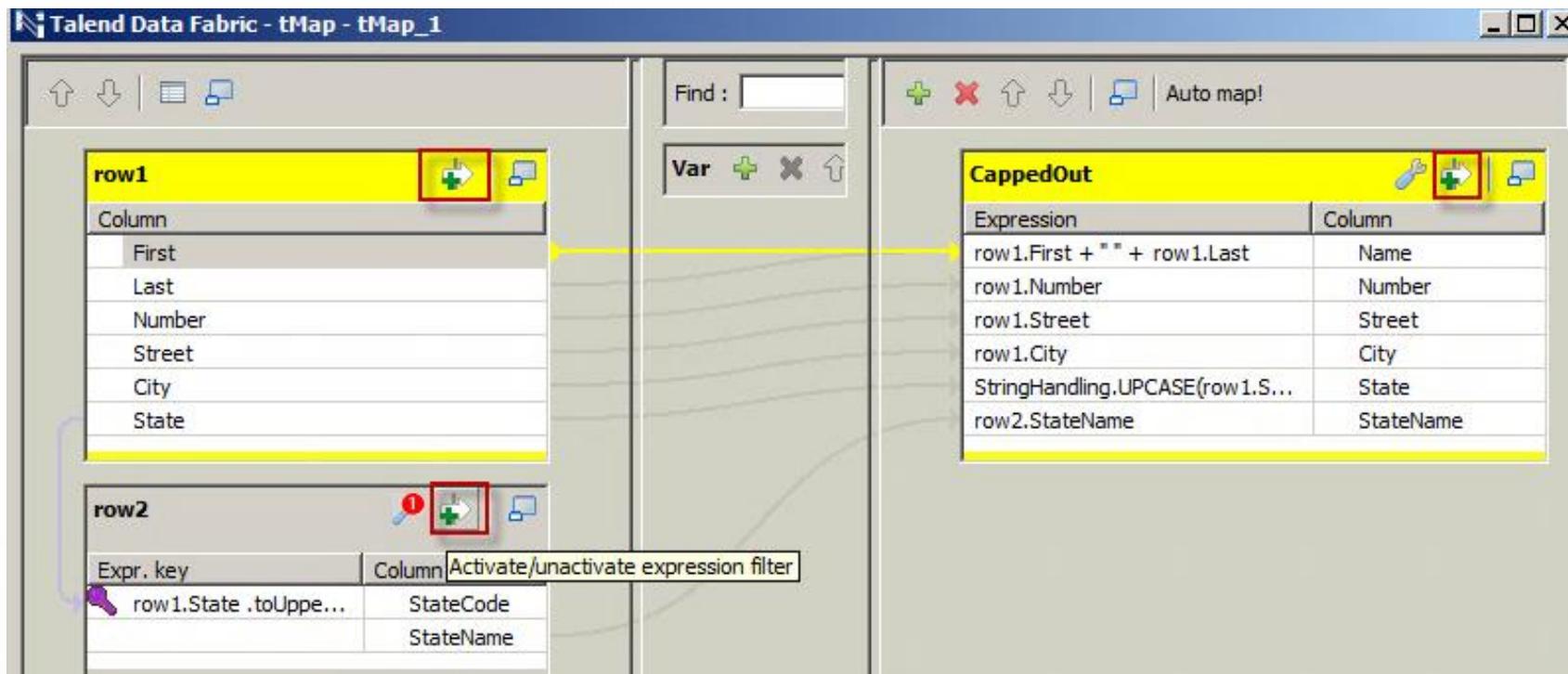
By the end of this lesson, you will be able to:

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



Filtering Data

- Retrieve a limited set of data based on conditions
- The **tMap** component allows data filtering
 - Filters can be applied on any **input** or **output** table





Configure Data Filtering

Add a Filter

1. Activate the filtering option.
2. Set the filter expression
3. Use the **Expression Builder** to assist if needed

The screenshot illustrates the process of configuring a data filter. On the left, a configuration interface shows a table with columns 'Expression' and 'Column'. A row contains the expression "'CA'.equals(row2.StateCode)". Above this row is a toolbar with a green plus icon (highlighted with a red box and number 1) and a '...' button (highlighted with a red box and number 2). An arrow points from the '...' button to the Expression Builder window on the right. The Expression Builder window has a red border and displays the expression "'CA'.equals (row2.StateCode)' in the 'Expression' field. It includes a toolbar with 'Wrap', 'Undo(Ctrl + Z)', 'Clear', 'Test!', and 'Clear' buttons. Below the expression are operators (+, -, *, /, ==, <, <=, !=, >, >=) and logical operators (and, or, not). To the right of the expression are 'Var' and 'Test' panes. At the bottom are 'Add' and 'Remove' buttons. On the left side of the Expression Builder are 'Categories' (e.g., *All, User Defined, DataOperation, Data Quality, Mathematical, MDM, Numeric, Relational) and 'Functions' (e.g., ABS, ACOS, addDate, addRepeatingElement, ALPHA, ASIN). A help pane on the right shows the text "return an incremented". Buttons for 'Ok' and 'Cancel' are at the bottom right.



Filtering Data

Example filtering column data

Input Data

	First	Last	Number	Street	City	State
1	Bill	Coolidge	85013	Via Real	Austin	IL
2	Thomas	Coolidge	63489	Lindbergh Blvd	Springfield	ca
3	Harry	Ford	97249	Monroe Street	Salt Lake City	ca
4	Warren	McKinley	82589	Westside Freeway	Concord	ak
5	Andrew	Taylor	29886	Padre Boulevard	Madison	CA
6	Ulysses	Coolidge	98646	Bayshore Freeway	Columbus	MN
7	Theodore	Clinton	12292	San Marcos	Bismarck	NY
8	Benjamin	Jefferson	82077	Carpinteria North	Sacramento	ca
9	William	Van Buren	21712	Tully Road East	Albany	IL
10	Calvin	Washington	50742	Richmond Hill	Charleston	ca
11	Jimmy	Polk	76143	Richmond Hill	Salt Lake City	AK
12	Calvin	Adams	52386	Lake Tahoe Blvd.	Montgomery	NY
13	Ulysses	Monroe	70511	Jones Road	Trenton	IL
14	Zachary	Tyler	45040	Santa Rosa North	Carson City	AK
15	Ulysses	Johnson	19989	Via Real	Juneau	AL
16	George	Arthur	89874	Calle Real	Annapolis	AL
17	George	Jefferson	67703	Fontaine Road	Pierre	IL
18	Herbert	Grant	90635	North Ventu Park Road	Columbus	AK
19	Calvin	Washington	37446	E Fowler Avenue	Pierre	al
20	John	Harrison	86745	San Marcos	Annapolis	AK

Filtered & Transformed Data

	Name	Number	Street	City	State
1	Thomas Coolidge	63489	Lindbergh Blvd	Springfield	CA
2	Harry Ford	97249	Monroe Street	Salt Lake City	CA
3	Andrew Taylor	29886	Padre Boulevard	Madison	CA
4	Benjamin Jefferson	82077	Carpinteria North	Sacramento	CA
5	Calvin Washington	50742	Richmond Hill	Charleston	CA

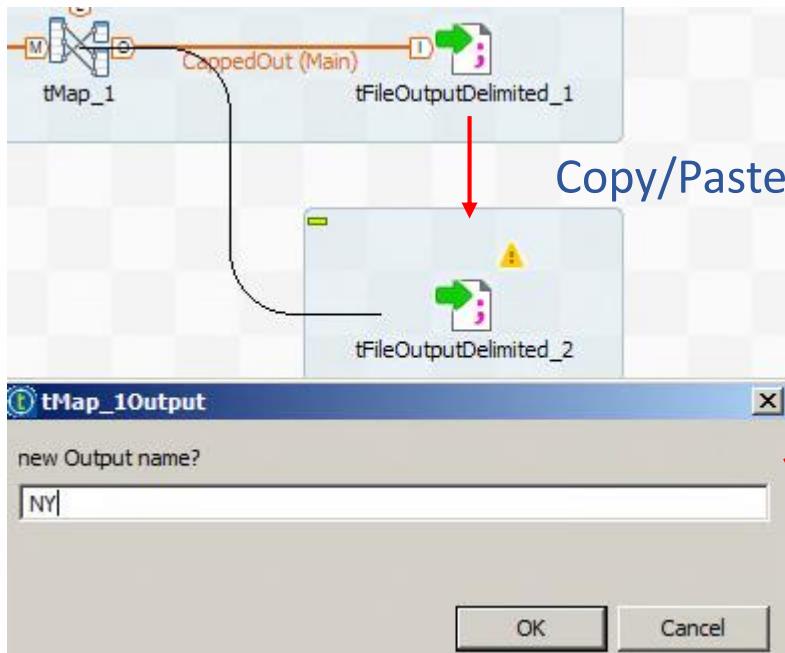
Filtered – California only
Transformed – Upper case



Duplicate Output Flows

Duplicate Component

- Copy/Paste output component
- Create connection with **tMap**
 - Schema is propagated to the **tMap** component
 - Mapping configuration is **empty** by default



The screenshot shows two component configuration windows. The top window is for 'CappedOut' with the expression "'CA'.equals(row2.StateCode)'. It contains a mapping table:

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

The bottom window is for 'NY' and is currently empty, with its mapping table also highlighted with a red box.

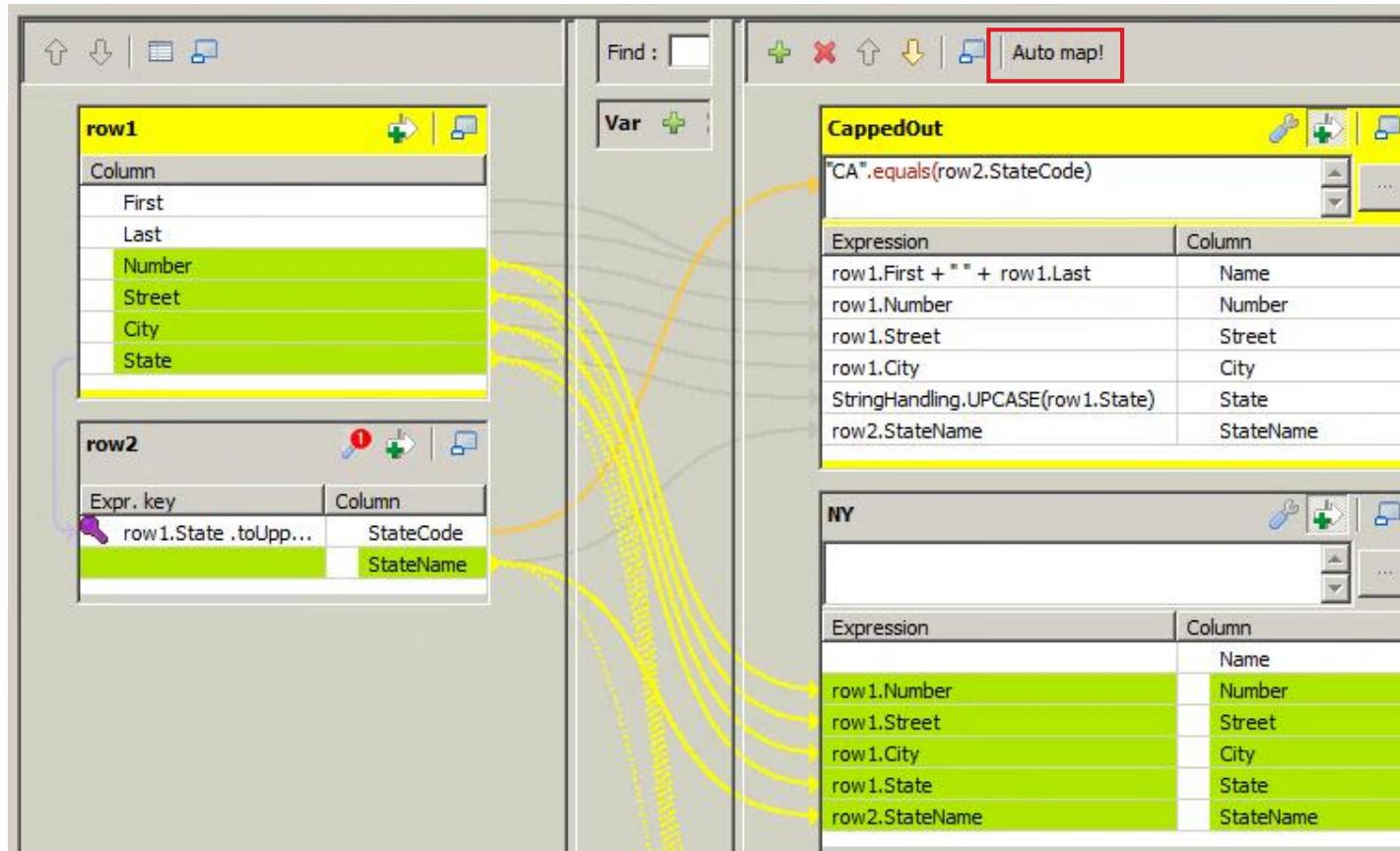


Duplicate Output Flows

Use Auto Map to duplicate output flows within tMap

- The **Auto map!** function:

- Maps **input columns with the same name** (highlighted in green)





Duplicate Output Flows

Finish tMap configuration

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

The screenshot shows two tMap configurations side-by-side.

CappedOut:

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

NY:

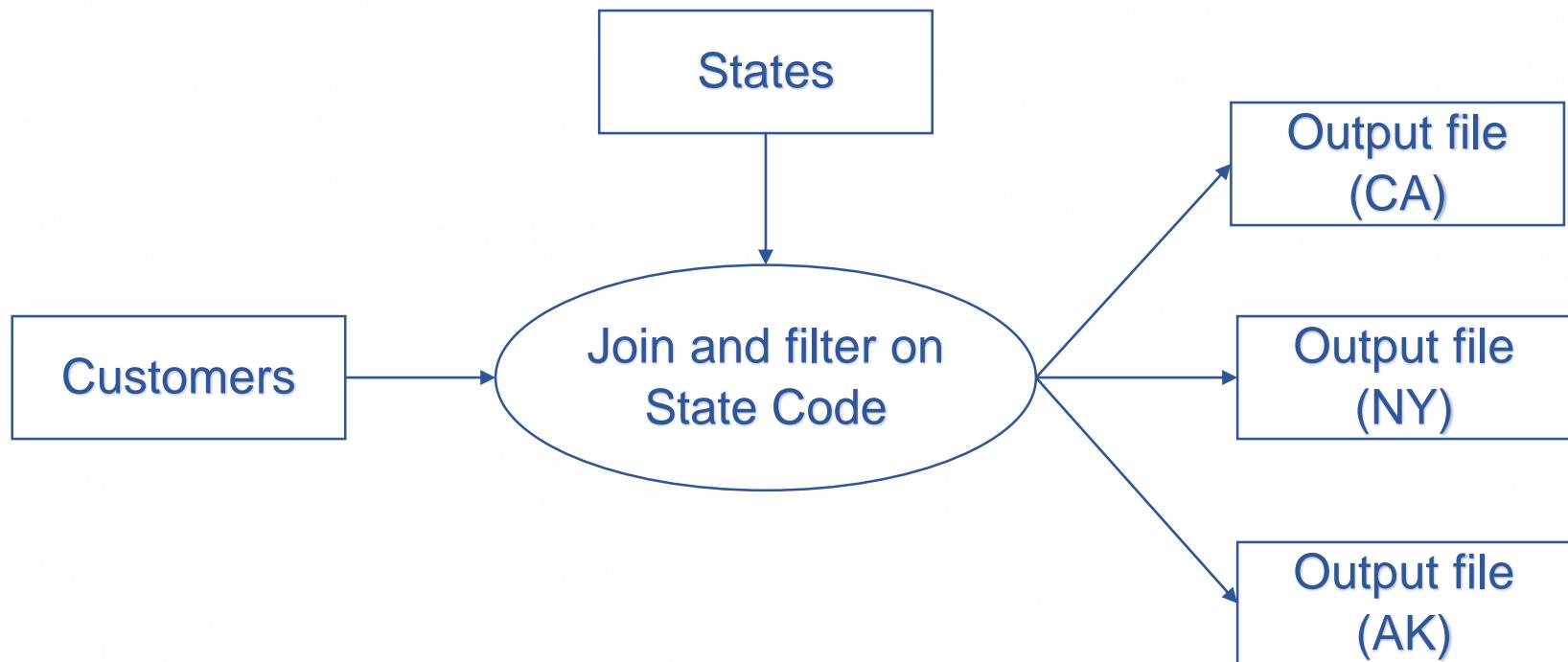
Expression	Column
"NY".equals(row2.StateCode)	
row1.First + " " + row1.Last	Name
row1.Number	Number
row1.Street	Street
row1.City	City
StringHandling.UPCASE(row1.State)	State
row2.StateName	StateName

In both tables, the first row's expression is highlighted with a red box. In the NY table, the last three rows' expressions are also highlighted with red boxes.



Lab overview

- Build a Job that generates three different output files (by filtering input data per state):



- Run Job and examine output files



Lesson summary

tMap

- Allows **data filtering** in input and output tables
- Allows **Multiple output flows** (to files, databases, ...)

Remember that **tMap** also allows

- Multiple Input flows
- Complex Data transformation
- Joining data
- Catching rejects





Using Context Variables



Objectives

By the end of this lesson, you will be able to:

- Create Built-in Context Variables, specific for a Job
- Run a Job using Built-in Context Variables
- Use Context Variables
- Create Repository Context Variables, available for all Jobs
- Run a Job using Repository Context Variables



What is a Context Variable?

- **Context Variables**

- Can take different values depending on the context

Context Variable	Development	Test	Production
InputDir	C:/Dev/Input	C:/Test/Input	C:/Prod/Input
OutputDir	C:/Dev/Output	C:/Test/Output	C:/Prod/Output

- A Job is always run in a **specific context**
- **Goal:** Specify the **value to use** by changing the **context** in which you run your Job



Define Context Variables

Context View

- Add variable(s)

- Add context(s)
- Set default context environment
- Set pre-defined values

Context View is available for each Job

The screenshot shows the 'Contexts(UpperCase_Context)' tab in a software interface. The title bar also displays 'Component' and 'Run (Job UpperCase_Context)'. Below the title bar is a toolbar with icons for adding (+), deleting (-), and other operations. A red arrow points from the 'Add' button on the toolbar to the 'OutputDir' row in the table. The table has columns for Name, Type, Default, and Value. The 'OutputDir' entry has 'String' selected as the type and a small input field for the value. A large green '+' button is located in the top right corner of the table area.

	Name	Type	Default	
			Value	
1	OutputDir	String		<input type="text"/>

Default context environment: Default



Define Context Variables

Context View

- Add variable(s)
- **Add context(s)**
- Set default context environment
- Set pre-defined values

The screenshot shows a software interface for defining context variables. The title bar reads "Contexts(UpperCase_Context)" and "Component Run (Job UpperCase_Context)". The main area is a table with columns: Name, Type, Default, and Production. The "Default" column has two rows: "Value" and "Production". The "Production" row is highlighted with a red box. In the bottom right corner of the table's data area, there is a green plus icon with a white cross inside. A blue callout box with the text "New context added" points to this icon. At the bottom of the interface, there are several icons: a green plus, a red minus, a yellow up arrow, a blue down arrow, and a grey save icon. To the right of these icons, it says "Default context environment" followed by a dropdown menu set to "Default".

	Name	Type	Default	Production
			Value	
1	OutputDir	String		

New context added



Define Context Variables

Context View

- Add variable(s)
- Add context(s)
- Set default context environment
- Set pre-defined values

The screenshot shows a software interface for defining context variables. The title bar reads "Contexts(UpperCase_Context)" and includes tabs for "Component" and "Run (Job UpperCase_Context)". The main area is a table with columns: Name, Type, Default, and Production. A single row is present with "OutputDir" as the name, "String" as the type, and two empty cells under "Default" and "Production". In the bottom right corner of the table, there is a red box highlighting a dropdown menu labeled "Default context environment" which has "Default" and "Production" options.

	Name	Type	Default		Production	
			Value		Value	
1	OutputDir	String				

Default context environment

Default

Production



Define Context Variables

Context View

- Add variable(s)
- Add context(s)
- Set default context environment
- **Set pre-defined values**

The screenshot shows a software interface for managing contexts. At the top, there are tabs for 'Job(UpperCase_Context...)', 'Contexts(UpperCase_Co...', 'Run (Job UpperCase_Co...', and 'Component'. Below the tabs is a toolbar with icons for adding (+), deleting (-), and navigating (up, down, left, right). A large table is the central element, divided into sections for 'Default' and 'Production'. The 'Default' section has columns for 'Name', 'Type', 'Value', and 'Prompt'. The 'Production' section has columns for 'Value' and 'Prompt'. A row in the 'Default' section is highlighted with a red border and contains the value 'C:/StudentFiles/' in the 'Value' column and a checked checkbox in the 'Prompt' column. An arrow points from the text 'Prompt Checkbox' to this checked checkbox. The table also includes a '+' button in the bottom right corner.

	Name	Type	Default		Production		
			Value	Prompt	Value	Prompt	
1	OutputDir	String	"C:/StudentFiles/"	<input checked="" type="checkbox"/>	OutputDir?	"C:/StudentFiles/Production"	<input type="checkbox"/>

Update pre-defined values during execution by using the **Prompt checkbox**



Repository Context Variables

- Defined in the **Repository -> Contexts**
 - Similar to Context Variables defined in the Job's **Contexts** tab (Built-In)
- Can be reused in several Jobs
 - Import them in the Job's **Contexts Tab**
 - Values are grayed out - Can only be changed in the **Repository**

The screenshot shows the 'Contexts' tab of a job configuration. The title bar includes tabs for 'Job(UpperCase_Context...)', 'Contexts(UpperCase_Co...', 'Run (Job_UpperCase_Co...', 'Component', and buttons for minus, plus, and a green plus sign.

The main area is a table with columns: Name, Type, Default, and Production. The 'Default' column has a dropdown arrow. The 'Production' column also has a dropdown arrow. A red box highlights rows 2 and 3, which correspond to the 'LocationContextVariables' and 'RepoOutputDir' entries in the table below.

	Name	Type	Default		Production	
			Value		Value	
1	OutputDir	String	"C:/StudentFiles/"	<input type="checkbox"/>	"C:/StudentFiles/Production"	<input type="checkbox"/>
2	LocationContextVariables			<input type="checkbox"/>		<input type="checkbox"/>
3	RepoOutputDir	String	"C:/StudentFiles/"	<input type="checkbox"/>	"C:/StudentFiles/New_Production"	<input type="checkbox"/>

At the bottom, there are icons for adding (+), deleting (-), and sorting (up and down arrows). To the right, it says 'Default context environment' and 'Default' with a dropdown arrow. A red arrow points from the 'LocationContextVariables' entry in the table to the delete (-) icon at the bottom.



Use Context Variables

Access the Context Variable values

- All Context Variables (Built-In or Repository) are accessed the same way
- Script code to access a context variable:

context.<VariableName>

tFileOutputDelimited_1

Basic settings	Property Type Built-In	
Advanced settings	<input type="checkbox"/> Use Output Stream	
Dynamic settings	File Name context.OutputDir + "CappedOut.csv"	
View	Row Separator "\n"	Field Separator ";"
Documentation	<input type="checkbox"/> Append <input checked="" type="checkbox"/> Include Header <input type="checkbox"/> Compress as zip file	
Validation Rules	Schema Built-In	Edit schema ... Sync columns



Use Context Variables

Change Context While Running a Job

- Specify execution context in the Run View
 - The default context and values are displayed
 - When the context is changed, pre-defined values are displayed
 - Pre-defined values can not be updated in the Run View

The screenshot shows the Talend Run View interface for a job named "JobUpperCase_Context". The left panel displays basic run options like Debug Run, Advanced settings, Target Exec, and Memory Run. The right panel shows the execution interface with a "Run" button and a log window. A context dropdown menu is open, showing "Production" selected. Below it, a table lists context variables:

Name	Value
OutputDir	"C:/StudentFiles/Production"
RepoOutputDir	"C:/StudentFiles/New_Production/"

Annotations on the right side of the screenshot identify the context dropdown and the table as "Context" and "Variables" respectively.



Lab overview

Create variables to provide different values in two different contexts, **development** and **production** environments:

- Create **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 repository context variable's pre-defined value



Lesson summary

Context Variable types

- **Built-In**
 - Defined for each Job in the **Contexts View**
 - Can be updated in the Job's context view
- **Repository**
 - Defined in the Repository -> Contexts
 - Can be reused in several Jobs
 - Can only be updated in the Repository

Execution Context can be set up in the Run View





Error Handling



Lesson Objectives

By the end of this lesson, you will be able to:

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

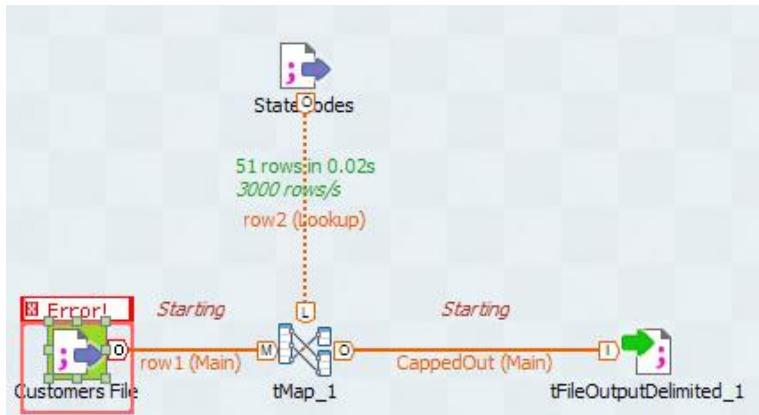


Error Handling

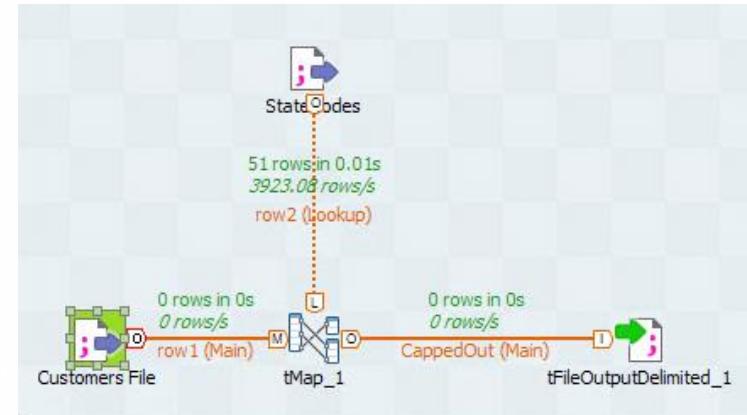
Overview

- In the event that a component throws an error
Example: The input file does not exist
- Job execution options:

1. The Job stops when error is encountered



2. The Job completes execution despite the error





Error Handling

Kill a Job on component Error

- **Die On Error** option - Available for certain components

Check Die On Error option

Job stops execution when error is encountered

The screenshot shows the configuration window for a 'Customers File(tFileInputDelimited_1)' component. The 'Basic settings' tab is active. In the 'Validation Rules' section, the 'Die on error' checkbox is checked and highlighted with a red box. To the right, a small Talend job diagram is shown with a red box around the 'Customers File' component, indicating where the 'Die on error' setting would be applied.

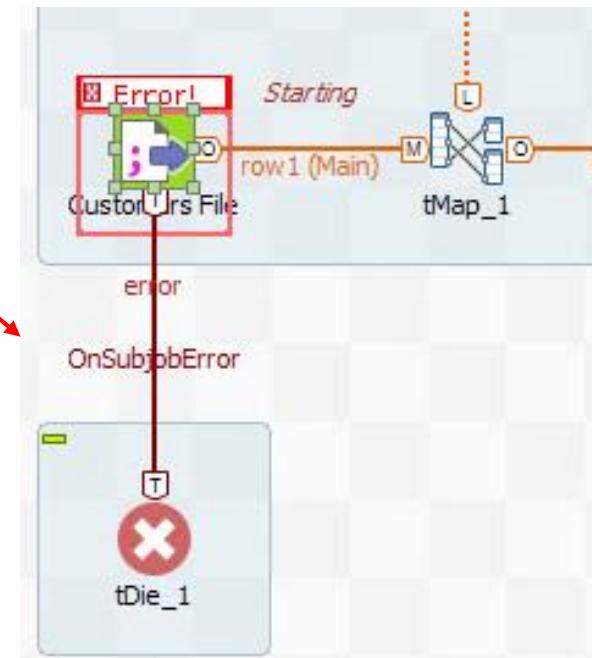
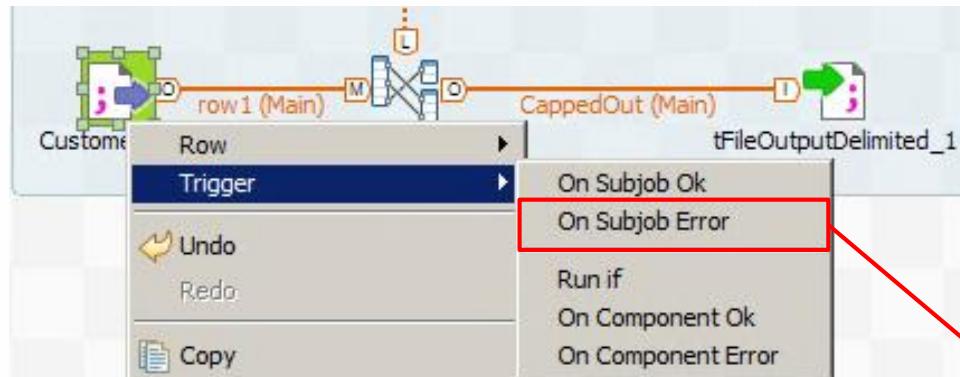
- Some components always die on error, so the option is not available



Error Handling

Triggers

- Triggers allow implementation of different execution paths:
 - The status of components/subjobs
 - Specific conditions



- Example: Define a message in case of an error
 - Use **tDie** component to set the Die Message



Error Handling

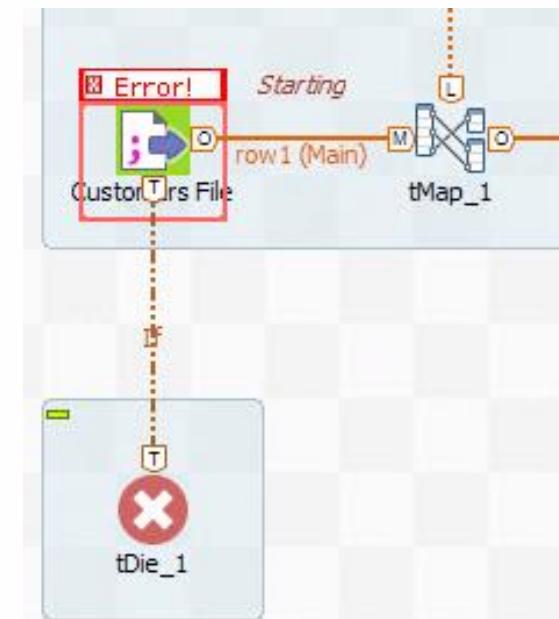
Kill a Job under specific conditions

Requirement:

- Kill a Job under specific conditions (no technical error)
- Ex: The input file has less than a certain number of lines

Solution:

- Use the **tDie** component to kill the Job
- Use Run If trigger to specify the condition
 - The Job execution stops if the condition is met





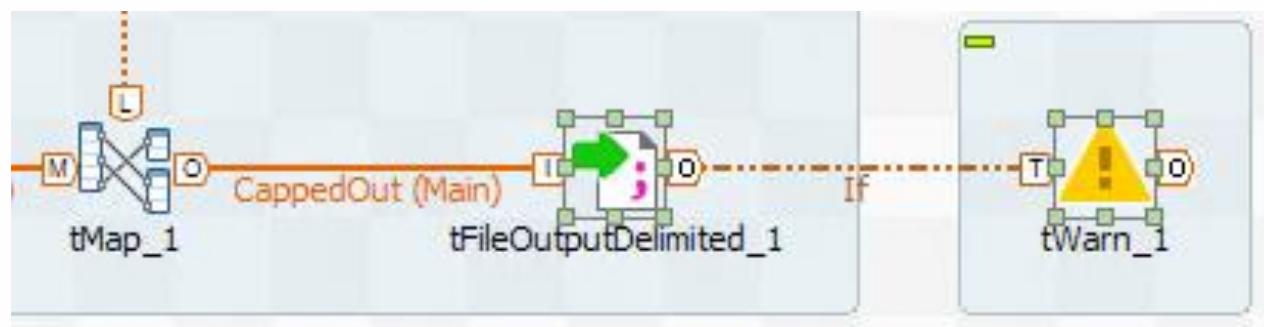
Raise a Warning

Need:

- Raise a Warning when a business rule is not met
- Example: The output file does not have the expected number of lines

Solution:

- Use the **tWarn** component to **raise a warning**
- Use **Run If** trigger to **specify the condition**
 - A warning is raised if the condition is met





Log Level

Configure the log level

• log4jLevel

- Available under the **Advanced Settings tab** of the **Run View**
- Specifies the types of **log** messages to be displayed

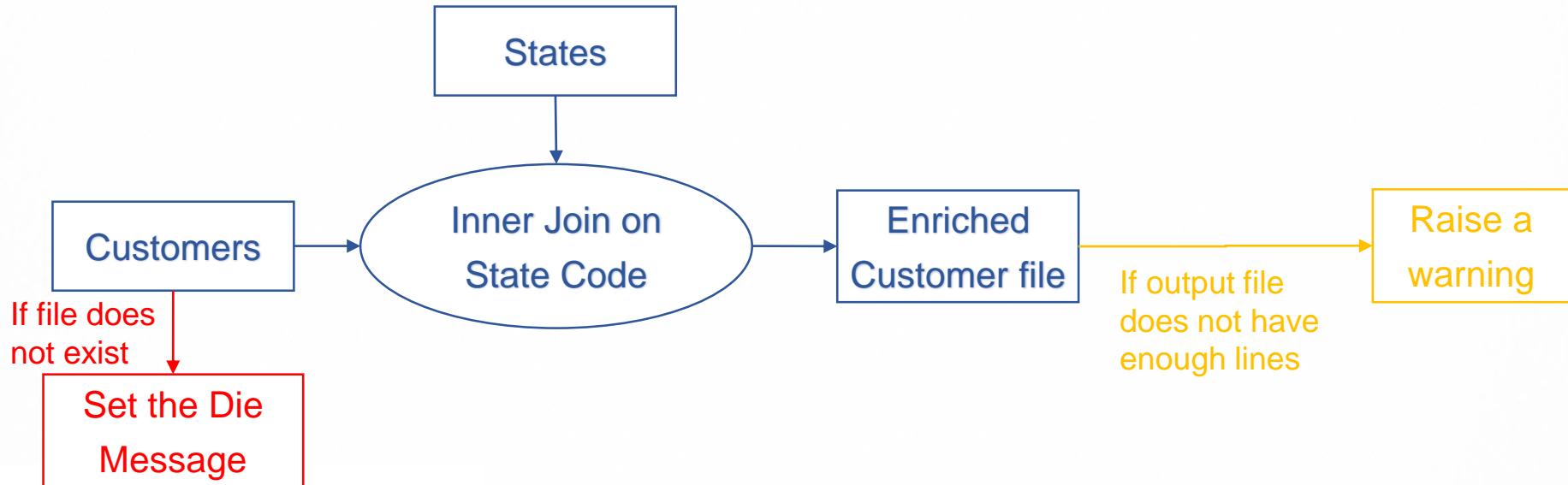
The screenshot shows the Apache Nifi Run View interface. On the left, there is a sidebar with tabs: Component, Run (Job Error_LookupState), and a close button. Below the tabs, there are several run configurations: Basic Run, Debug Run, Advanced settings (which is selected), Target Exec, and Memory Run. Under the Advanced settings tab, there are several checkboxes: Statistics (checked), Save Job before execution (checked), Exec time (unchecked), Clear before run (checked), and log4jLevel (checked). A red box highlights the log4jLevel checkbox and the dropdown menu next to it. The dropdown menu lists several log levels: Trace, Debug, Info, Warn (selected), Error, Fatal, and Off. Below the dropdown, there are sections for JVM Setting, Job Run VM arguments, and Argument, each with their respective configurations. A red arrow points from the highlighted log4jLevel setting in the configuration to the corresponding log output in the main pane. The main pane is titled 'Execution' and contains three buttons: Run (highlighted in blue), Kill, and Clear. It displays the log output for the job 'Error_LookupState'. The log output shows the job starting at 02:46 on 18/03/2016, connecting to a socket on port 3536, and then disconnecting. A red box highlights the 'Warn' log entry: '[WARN]: di_b.error_lookupstate_0_1.Error_LookupState - tWarn_1 - Message: Less than 100 rows written. Code: 42'. The log concludes with the job ending at 02:46 on 18/03/2016 with an exit code of 0.

```
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

1. Create a technical error: input file does not exist
 - Kill the Job
 - Set the Die message
2. Create a business error: 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
 - To kill a Job under certain conditions
- Raise a Warning
 - **tWarn** component can be used
- Configure **log level** in the Advanced setting of the **Run view**





Generic Schemas



Lesson Objectives

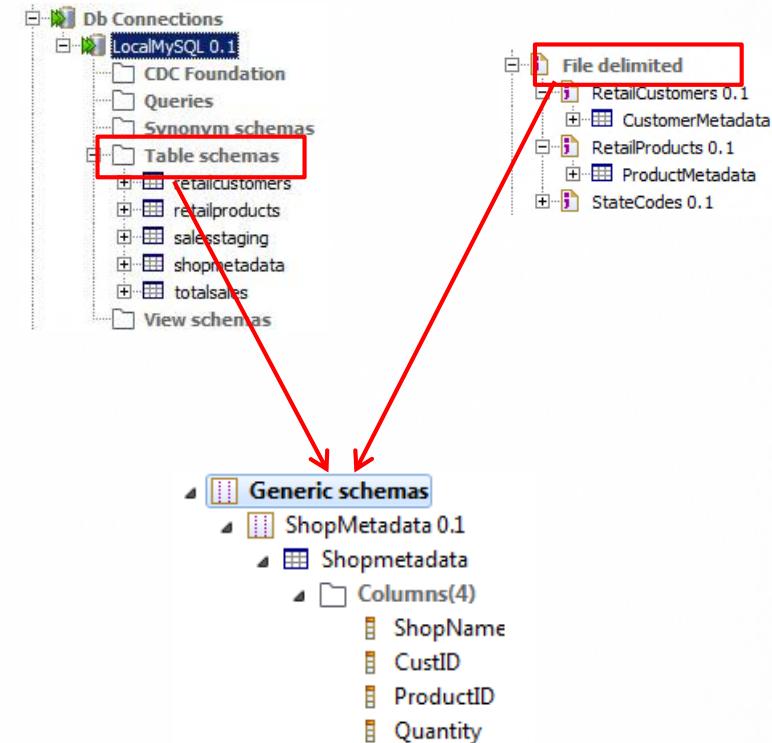
By the end of this lesson, you will be able to:

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



Generic schemas - Definition

It is possible to create schema metadata for specific types of Files (Delimited, XML, JSON, etc). But this schema metadata can't be used interchangeably as they are linked to a type of file.

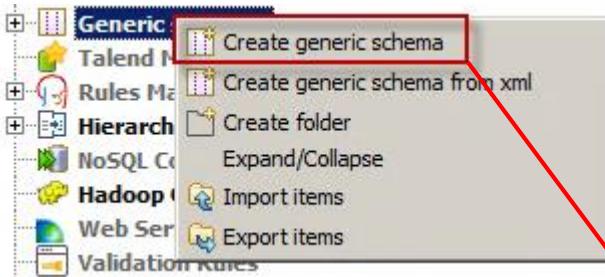


A **generic schema** is one that can be used in any component requiring a schema and is **not linked to any type of source** (file, database, etc.).

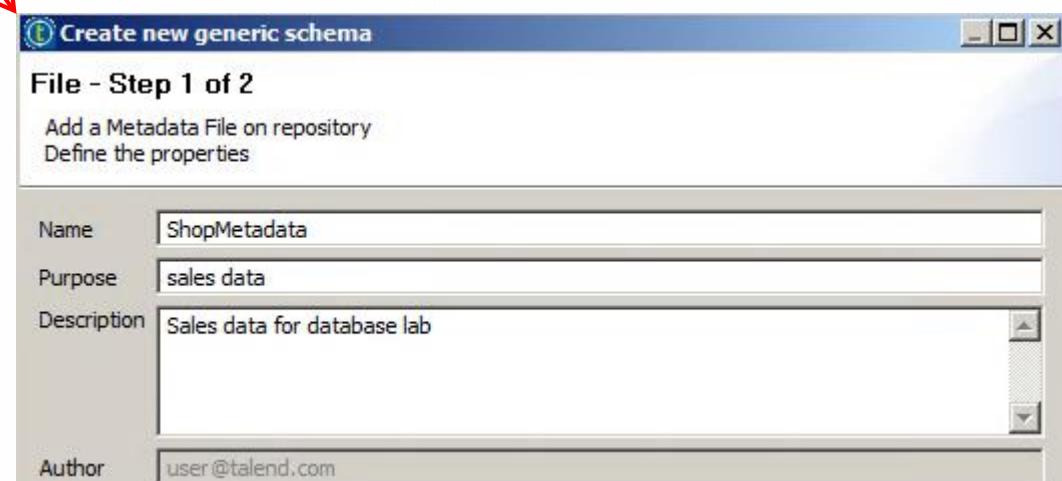


Create New Generic Schema

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



Provide Name, Purpose
and Description.





Create New Generic Schema

- Specify the Column's Name and Type

File - Step 2 of 2
Edit an existing Metadata File on repository
Update the properties

Name Shopmetadata
Comment

Select the database mapping type

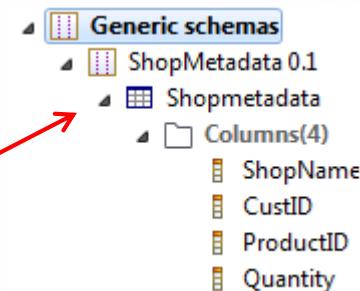
Schema

Description of the Schema

Column	Key	Type	N..	Date...	Length	Precision
ShopName	<input type="checkbox"/>	String	<input checked="" type="checkbox"/>		10	0
CustID	<input type="checkbox"/>	Integer	<input checked="" type="checkbox"/>		0	
ProductID	<input type="checkbox"/>	Integer	<input checked="" type="checkbox"/>		0	
Quantity	<input type="checkbox"/>	Integer	<input checked="" type="checkbox"/>		0	

< Back Cancel

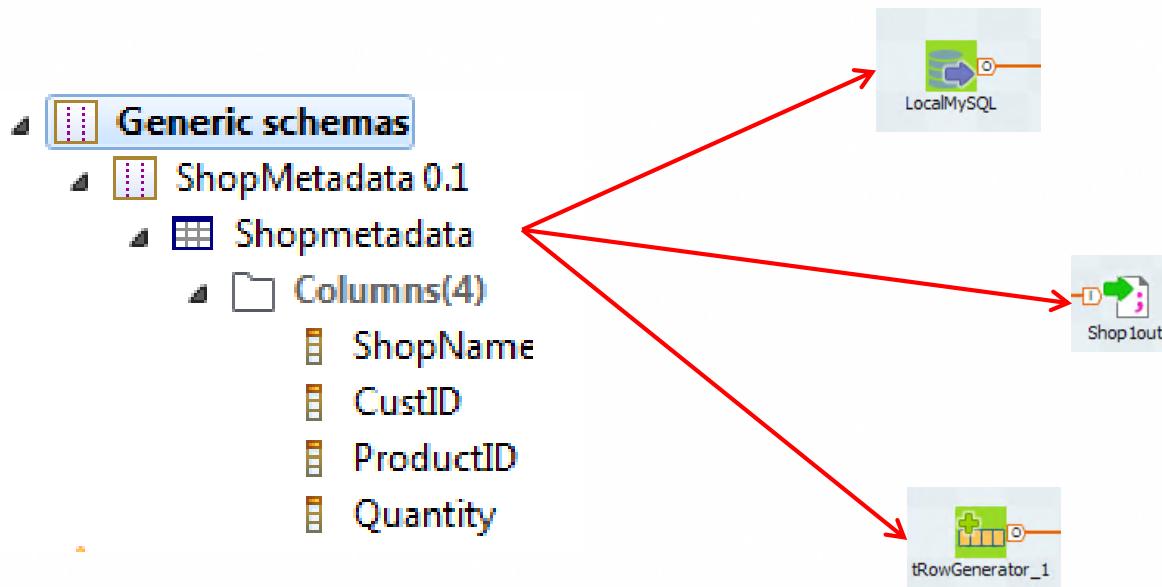
The Generic schema appears in the Repository





Where are Generic schemas used?

- Generic schemas can be used in various components:
 - Input components: **tFileInputDelimited**, **tMySqlInput**...
 - Output components: **tFileOutputDelimited**, **tMySqlOutput**...
 - Miscellaneous components: **tRowGenerator**...





Generating Random Data

Using the tRowGenerator component

Define schema

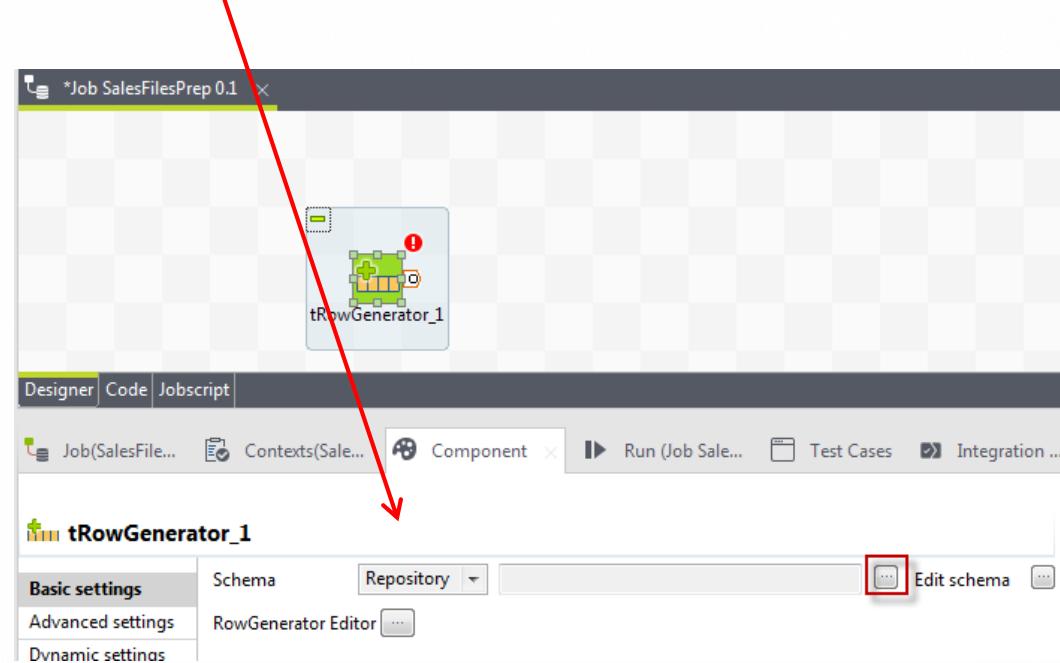
Set number of rows

Select Functions

Set environment variable values

Preview values

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





Generating Random Data

Using the tRowGenerator component

Define schema

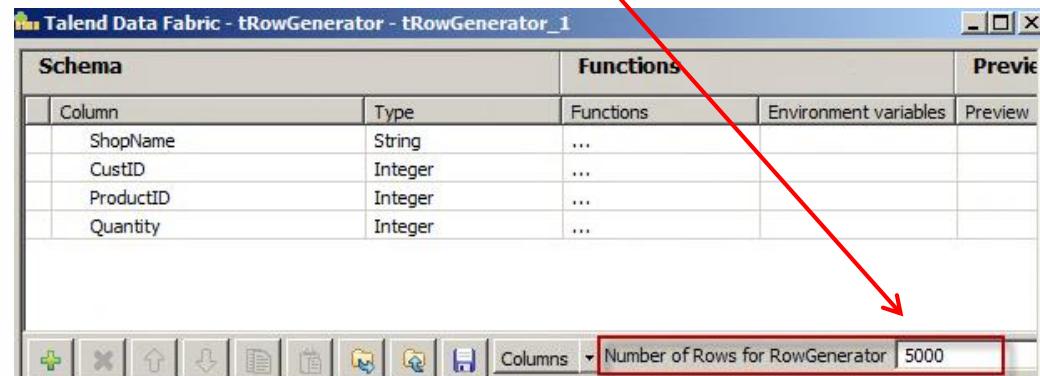
Set number of rows

Select Functions

Set environment variable values

Preview values

- Specify the number of rows to generate in the tRowGenerator editor





Generating Random Data

Using the tRowGenerator component

Define schema

Set number of rows

Select Functions

Set environment variable values

Preview values

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

The screenshot shows the Talend Data Fabric interface for the tRowGenerator component. On the left, there's a sidebar with tabs: 'Select Functions' (which is active), 'Set environment variable values', and 'Preview values'. The main area has a title bar 'Talend Data Fabric - tRowGenerator - tRowGenerator_1'. Below the title bar is a toolbar with icons for adding columns, deleting columns, moving columns up and down, and saving. The main workspace contains a table titled 'Schema' with columns 'Column', 'Type', 'Functions', 'Environment variables', and 'Preview'. There are four rows in the table:

Column	Type	Functions	Environment variables	Preview
ShopName	String	...	"Shop1"	
CustID	Integer	Numeric.random(int...)	min value=>1 ; max value=>100 ;	
ProductID	Integer	Numeric.random(int...)	min value=>1 ; max value=>80 ;	
Quantity	Integer	Numeric.random(int...)	min value=>1 ; max value=>1000 ;	

To the right of the table is a vertical scrollable list of functions. The 'Numeric.random(int,int)' function is highlighted with a blue selection bar and a red arrow points from the 'Functions' column in the table to this item in the list. Other visible functions include Mathematical.BITAND, Mathematical.BITNOT, Mathematical.BITOR, Mathematical.BITXOR, Mathematical.INT, Mathematical.NUM, Mathematical.SCMP, Mathematical.SDIV, StringHandling.COUNT, StringHandling.INDEX, StringHandling.LEN, TalendDate.compareDate, TalendDate.compareDateWithFormat, TalendDate.diffDateFloor, and TalendDate.getPartOfDay.



Generating Random Data

Using the tRowGenerator component

Define schema

Set number of rows

Select Functions

Set environment variable values

Preview values

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

Talend Data Fabric - tRowGenerator - tRowGenerator_1

Column	Type	Functions	Environment variables
ShopName	String	...	"Shop1"
CustID	Integer	Numeric.random(int...)	min value=>0 ; ma...
ProductID	Integer	...	
Quantity	Integer	...	

Function parameters Preview

return a random int between min and max

Parameter	Value	Comment
min value	1	
max value	100	

OK Cancel

Talend Data Fabric - tRowGenerator - tRowGenerator_1

Column	Type	Functions	Environment variables	Preview
ShopName	String	...	"Shop1"	
CustID	Integer	...		
ProductID	Integer	...		
Quantity	Integer	...		

Function parameters Preview

Set your own expression...

Parameter	Value	Comment
customize parameter	"Shop1"	

OK Cancel



Generating Random Data

Using the tRowGenerator component

Define schema

Set number of rows

Select Functions

Set environment variable values

Preview values

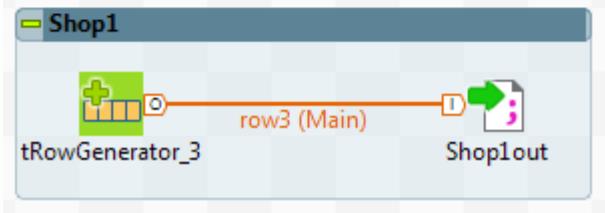
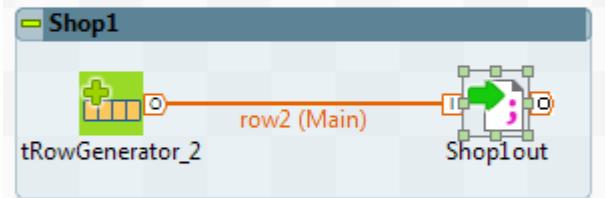
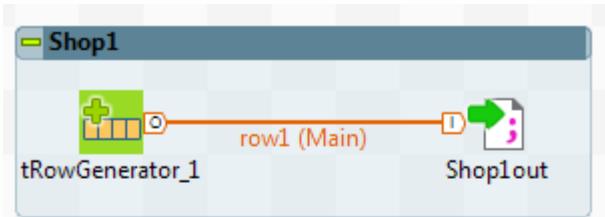
- Verify the configuration using the Preview feature of the tRowGenerator component

ShopName	CustID	ProductID	Quantity
1 Shop1	75	11	335
2 Shop1	27	52	715
3 Shop1	24	42	673
4 Shop1	40	10	45
5 Shop1	51	47	655
6 Shop1	28	25	567
7 Shop1	86	70	246
8 Shop1	52	76	456
9 Shop1	3	47	800
10 Shop1	73	19	383



What is a Subjob ?

- A collection of connected components within a Job

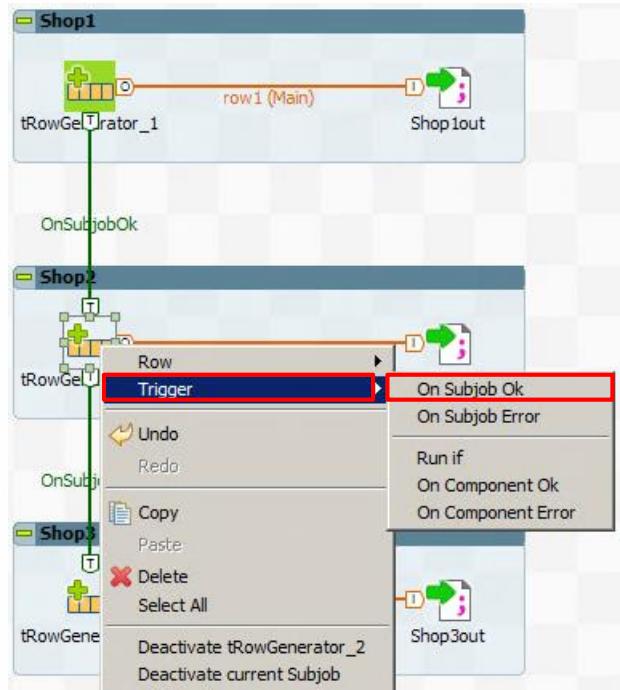


Each subjob appears in a blue rectangle



Connecting Subjobs with Triggers

- A subjob is connected to another subjob with a **Trigger**, available on the *first* component of the Subjob.



The green links (Triggers) are **not** flows

- No data is transferred from one component to another
- They make it possible to **sequence** the execution of Job Design processing



Subjobs – Activation/Deactivation

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

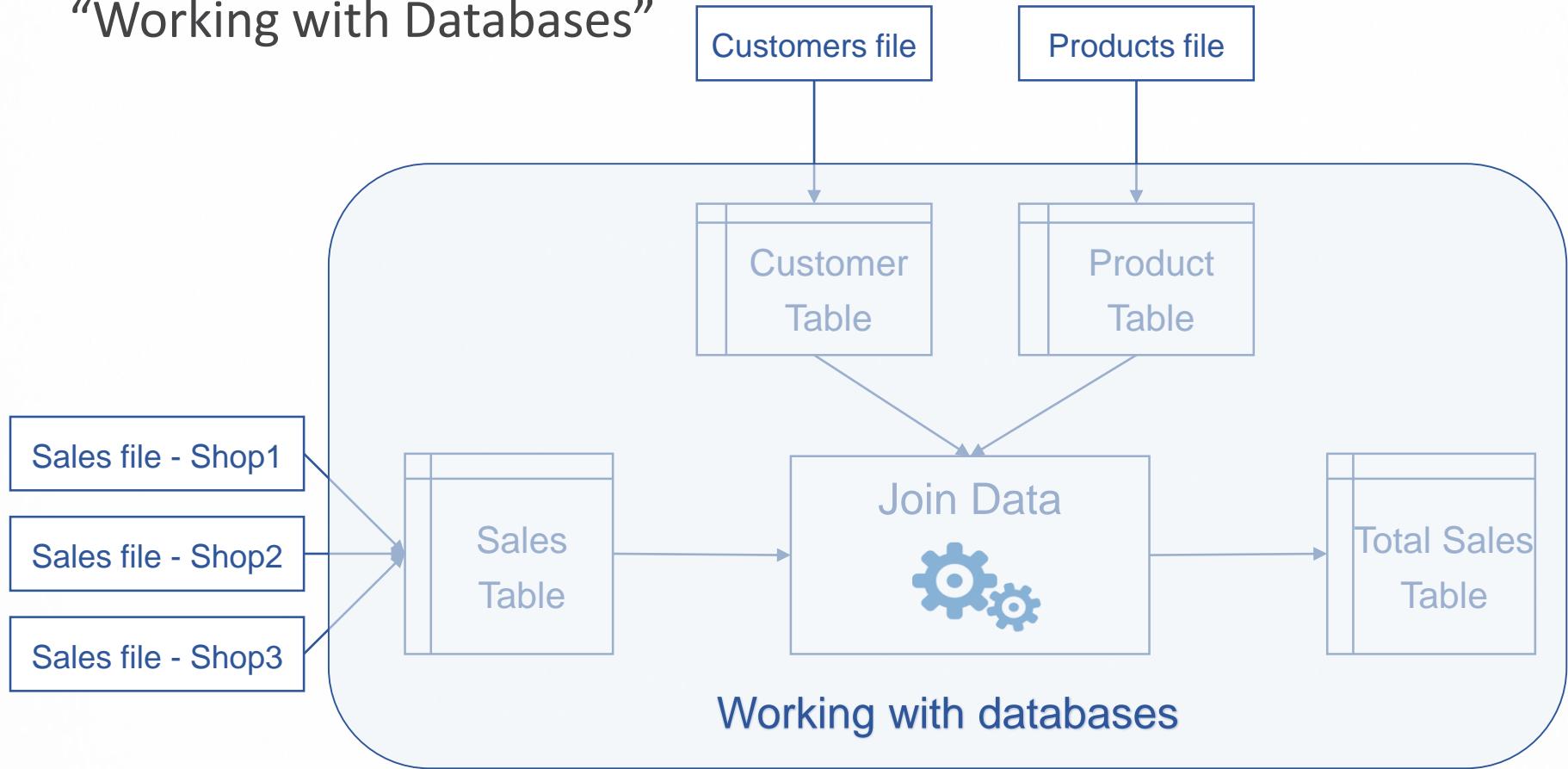




Lab overview

Generic Schemas

- Prepare schemas, files, and metadata for the next lesson
“Working with Databases”





Lesson summary

- **Generic Schemas** are reusable schemas not linked to any type of source
- **tRowGenerator** paired with Generic Schemas can be used to produce sample data
 - Great for testing
 - Easy to test edge case values (e.g. boundaries such as low/high limits)
- **Subjobs with Triggers** - Process data based on different flows
 - When things work properly (On Subjob OK)
 - When things don't (On Subjob Error)





Working With Databases



Objectives

By the end of 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
- Filter unique data rows
- Perform aggregation
- Write data to an XML file



Database Connection

Creating Metadata

- Database connection metadata can be found under **Repository > Metadata > Db connections**

The screenshot shows the SAP BusinessObjects interface. On the left, there is a navigation tree under 'Metadata' with 'Db Connections' selected. A red arrow points from the 'Create connection' option in this menu to the 'Create connection' button in the main dialog window. The main window is titled 'Database Connection' and displays 'New Database Connection on repository - Step 1/2'. It asks to 'Define the properties' for a connection named 'LocalMySQL'. The fields filled in are:

Name	LocalMySQL
Purpose	Create Connection MySQL
Description	DI BASIC Training
Author	student@company.com
Locker	
Version	0.1
Status	
Path	

At the bottom of the dialog are buttons for '< Back', 'Next >', 'Finish', and 'Cancel'.

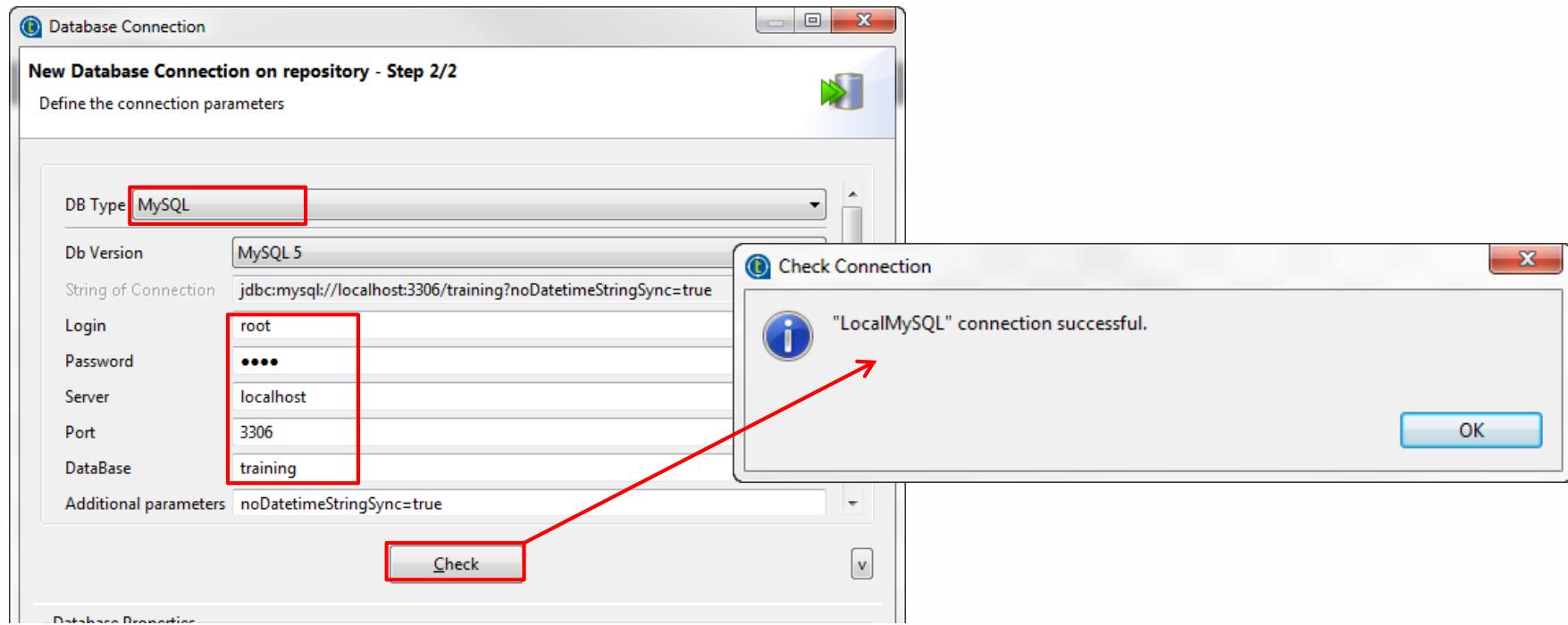
Provide Name, Purpose,
and Description



Database Connection

Creating Metadata

- Specify **key connection information** and check your connection

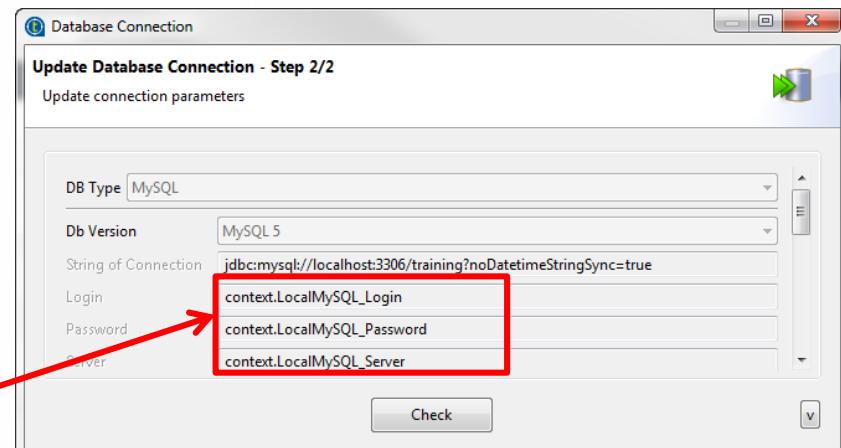
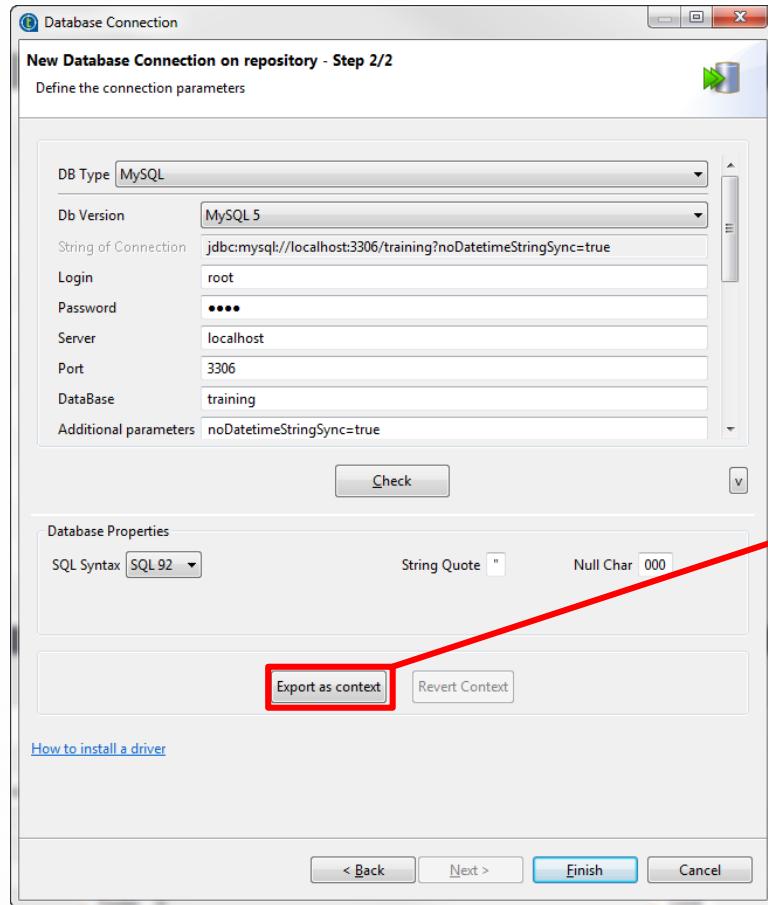




Database Connection

Export as Context

- Save connection info as context variables





Database Connection

Using Metadata

The screenshot illustrates the process of using metadata for database connections in Talend. A red arrow points from the 'Db Connections' node in the Repository tree to the 'tCreateTable_1' component in the Designer. Another red arrow points from the 'tMysqlOutput' component in the 'Components' dialog to the configuration panel for the 'LocalMySQL(tMysqlOutput_1)' component.

Repository Tree:

- LOCAL: DI_B
- Contexts
- Code
- SQL Templates
- Metadata
 - Db Connections
 - LocalMySQL 0.1
 - CDC Foundation
 - Queries
 - Synonym schemas
 - Table schemas
 - View schemas
 - SAP Connections
 - File delimited
 - File positional
 - File regex
 - File Excel
 - File Idif
 - File Json
 - LDAP

Designer View:

- *Job CustPrep2Table 0.1
- tCreateTable_1
- RetailCustomers
- OnComponentOk

Components Dialog:

- Choose one component to create.
 - tMemSQLInput
 - tMysqlOutput
 - tMemSQLOutput
 - tAmazonMysqlInput
 - tAmazonMysqlOutput
- OK
- Cancel

Component Configuration Panel:

Basic settings

Property Type: Repository (highlighted)
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:

Action on table: Default Action on data: Insert

Schema: Built-In Edit schema ... Sync columns

- Drag the database connection metadata to the Designer
- Select a component in the list
- The component is configured with the connection information metadata



Writing data to a Database

tMysqlOutput

- To write data to a MySQL database, you can use the **tMysqlOutput** component

The screenshot shows the configuration interface for the tMysqlOutput component. The title bar indicates the job name is "Job(CustPrep2Table 0.1)". The main panel is titled "LocalMySQL(tMysqlOutput_1)". On the left, there is a sidebar with tabs: "Basic settings" (selected), "Advanced settings", "Dynamic settings", "View", "Documentation", and "Validation Rules". The "Basic settings" tab contains the following fields:

- Property Type: Repository (dropdown menu)
- DB Version: Mysql 5 (dropdown menu)
- Use an existing connection
- Host: context.LocalMySQL_Server (with an asterisk and a yellow warning icon)
- Database: context.LocalMySQL_Database
- Username: context.LocalMySQL_Login (with an asterisk and a yellow warning icon)
- Password: context.LocalMySQL_Password (with an asterisk and a yellow warning icon)
- Table: "RetailCustomers" (highlighted with a red border)
- Action on table: Default (dropdown menu)
- Action on data: Insert
- Schema: Repository (dropdown menu) / DELIM:RetailCustomers - CustomerMetadata (text input)

- You should also provide the **Table Name** and the **Schema**



Writing data to a Database

tMysqlOutput – Action on table

- Select the **Action on table** in the drop down list

The screenshot shows the configuration window for a tMysqlOutput component named "LocalMySQL(tMysqlOutput_1)". The "Basic settings" tab is selected. The "Table" field contains the value "CustomerMetadata". The "Action on table" dropdown menu is open and highlighted with a red box, showing the following options:

- Default
- Drop and create table
- Create table
- Create table if does not exist
- Drop table if exists and create
- Specify a data source
- Clear table
- Truncate table

Other visible fields include "Repository" set to "DB (MYSQL):LocalMySQL", "DB Version" set to "Mysql 5", and "Host" set to "context.LocalMySQL_Server". The "Action on data" dropdown is set to "Insert".

- The table can be created first (if needed) then written to



Writing data to a Database

tMysqlOutput – Action on data

- Select the **Action on data** in the drop down list

The screenshot shows the configuration interface for a tMysqlOutput component named "LocalMySQL(tMysqlOutput_1)". The "Basic settings" tab is selected. The "Action on table" dropdown is set to "Default". A red box highlights the "Action on data" dropdown, which is currently set to "Insert". A second red box highlights the dropdown's menu, which includes options: Insert, Update, Insert or update, Update or insert, Delete, Replace, Insert or update on duplicate key or unique index, and Insert Ignore.

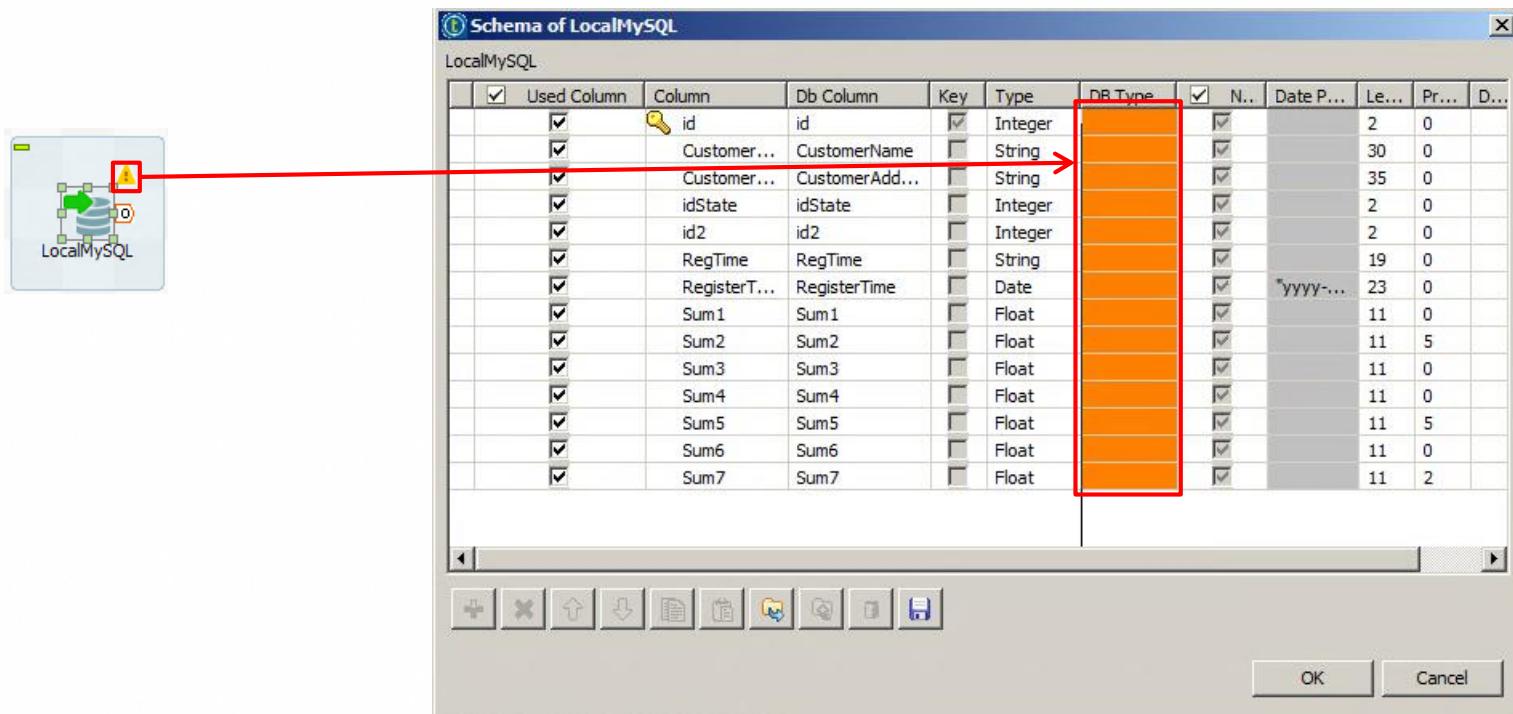
Basic settings	Property Type	Repository	DB (MYSQL):LocalMySQL
Advanced settings	DB Version	Mysql 5	
Dynamic settings	<input type="checkbox"/> Use an existing connection		
View	Host	context.LocalMySQL_Server	* Port context.LocalMySQL_Port
Documentation	Database	context.LocalMySQL_Database	
Validation Rules	Username	context.LocalMySQL_Login	* Password context.LocalMySQL_Password
	Table	"CustomerMetadata"	
	Action on table	Default	Action on data
	Schema	Repository	DELIM:RetailCustomers - Customer
Data source	This option only applies when deploying and running in the Talend Runtime		
	<input type="checkbox"/> Specify a data source alias		
	<input type="checkbox"/> Die on error		



Writing data to a Database

tMysqlOutput – Fix schema

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





Writing data to a Database

tMysqlOutput – Fix schema

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

Schema of LocalMySQL

Column	Db Column	Key	Type	DB Type		N..	Date Patt...	Length	Prec...	De...	Com..
id	id	<input checked="" type="checkbox"/>	Inte...	<input checked="" type="checkbox"/>				2	0		
CustomerName	CustomerName	<input type="checkbox"/>	String	<input checked="" type="checkbox"/>				30	0		
CustomerAddress	CustomerAddress	<input type="checkbox"/>	String	<input checked="" type="checkbox"/>				35	0		
idState	idState	<input type="checkbox"/>	Inte...	<input checked="" type="checkbox"/>				2	0		
id2	id2	<input type="checkbox"/>	Inte...	<input checked="" type="checkbox"/>							
RegTime	RegTime	<input type="checkbox"/>	String	<input checked="" type="checkbox"/>							
RegisterTime	RegisterTime	<input type="checkbox"/>	Date	<input checked="" type="checkbox"/>							
Sum1	Sum1	<input type="checkbox"/>	Float	<input checked="" type="checkbox"/>							
Sum2	Sum2	<input type="checkbox"/>	Float	<input checked="" type="checkbox"/>							
Sum3	Sum3	<input type="checkbox"/>	Float	<input checked="" type="checkbox"/>							
Sum4	Sum4	<input type="checkbox"/>	Float	<input checked="" type="checkbox"/>							
Sum5	Sum5	<input type="checkbox"/>	Float	<input checked="" type="checkbox"/>							
Sum6	Sum6	<input type="checkbox"/>	Float	<input checked="" type="checkbox"/>							
Sum7	Sum7	<input type="checkbox"/>	Float	<input checked="" type="checkbox"/>							

Schema of LocalMySQL

Column	Db Column	Key	Type	DB Type		N..	Date Patt...	Length	Prec...	De...	Com..
id	id	<input checked="" type="checkbox"/>	Inte...	<input checked="" type="checkbox"/>				2	0		
CustomerName	CustomerName	<input type="checkbox"/>	String	<input checked="" type="checkbox"/>				30	0		
CustomerAddress	CustomerAddress	<input type="checkbox"/>	String	<input checked="" type="checkbox"/>				35	0		
idState	idState	<input type="checkbox"/>	Inte...	<input checked="" type="checkbox"/>				2	0		
id2	id2	<input type="checkbox"/>	Inte...	<input checked="" type="checkbox"/>				2	0		
RegTime	RegTime	<input type="checkbox"/>	String	<input checked="" type="checkbox"/>				19	0		
RegisterTime	RegisterTime	<input type="checkbox"/>	Date	<input checked="" type="checkbox"/>				yyyy-mm...	23	0	
Sum1	Sum1	<input type="checkbox"/>	Float	<input checked="" type="checkbox"/>				FLOAT	11	0	
Sum2	Sum2	<input type="checkbox"/>	Float	<input checked="" type="checkbox"/>				FLOAT	11	5	
Sum3	Sum3	<input type="checkbox"/>	Float	<input checked="" type="checkbox"/>				FLOAT	11	0	
Sum4	Sum4	<input type="checkbox"/>	Float	<input checked="" type="checkbox"/>				FLOAT	11	0	
Sum5	Sum5	<input type="checkbox"/>	Float	<input checked="" type="checkbox"/>				FLOAT	11	5	
Sum6	Sum6	<input type="checkbox"/>	Float	<input checked="" type="checkbox"/>				FLOAT	11	0	
Sum7	Sum7	<input type="checkbox"/>	Float	<input checked="" type="checkbox"/>				FLOAT	11	2	

Click the **Reset DB types** button to fix the DB Types



Reading from a Database

tMysqlInput

To read data from a MySQL database, you can use the **tMysqlInput** component. This component requires a **SQL Query** to read the data from the MySQL database.

Click the **Guess Query** button to build the SQL Query based on the Table Name and the Schema

ProductDataIn(tMysqlInput_2)

Basic settings

Property Type: Repository, DB (MySQL):LocalMySQL

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

Schema: Repository, DELIM:RetailProducts - ProductMetadata

Table Name: **RetailProducts***

Query Type: Built-In, Guess Query, Guess schema

Query:

```
SELECT
`RetailProducts`.`id`,
`RetailProducts`.`Product`
FROM `RetailProducts`
```

Schema of ProductDataIn

Column	Db Column	Key	Type	DB Type	N...	Date P...
id	id		Integer	INT	<input checked="" type="checkbox"/>	
Product	Product		String	VARCHAR	<input checked="" type="checkbox"/>	10



Processing Data

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

Name: tSortRow



Purpose:

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

Name: tUniqRow



Purpose: Filters out duplicate entries

Name: tAggregateRow



Purpose:

- Aggregates input data based on one or more columns
- Sample operations: min, max, avg, sum, first, last,



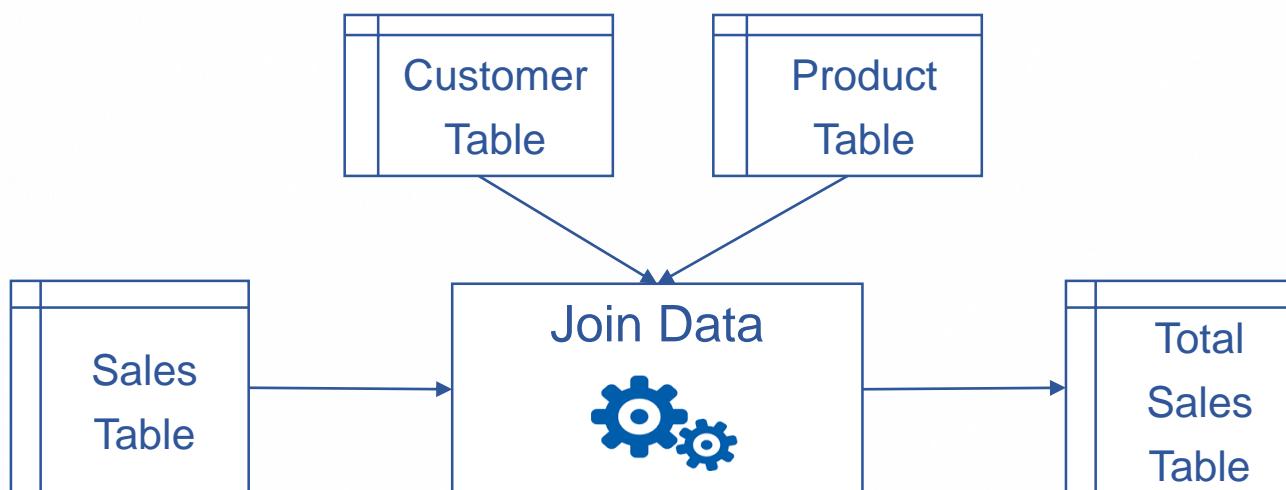
Lab overview

Working with Databases

- First, you will create Customer, Product, and Sales tables.



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





Lesson summary

- Create database connection information as reusable metadata
- Read/Write to Databases
 - tCreateTable
 - tMysqlInput
 - tMysqlOutput
- Processing data with components
 - tSortRow
 - tAggregateRow
 - tUniqRow





Master Jobs



Objectives

By the end of this lesson, you will be able to:

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



Job Orchestration - Master Jobs

Master Job – Calls and orchestrates child jobs





Job Orchestration - Master Jobs

When, Why and How

- When?

- Complex Jobs that need to run in sequence

- Why?

- Easy to use orchestration layer within the Talend Studio

- How?

- Largely through **tRunJob** component





Job Orchestration - Master Jobs

tRunJob – Placement in the design workspace

- tRunJob is placed like any other component
- It can also be dragged/dropped from the Repository





Job Orchestration - Master Jobs

tRunJob – Basics settings

The screenshot shows the Talend Studio interface with the 'Component' tab selected. A specific component, 'CustPrep2Table(tRunJob_1)', is highlighted. The 'Basic settings' tab is active, displaying various configuration options. A red box highlights the 'Job' dropdown set to 'CustPrep2Table', the 'Version' dropdown set to 'Latest', and the 'Context' dropdown set to 'Default'. Below these, there are checkboxes for 'Use dynamic job', 'Use an independent process to run subjob', and 'Transmit whole context', with 'Die on child error' checked. A 'Context Param' table is shown with two columns: 'Parameters' and 'Values'. At the bottom, there are several small icons for managing components.

- Specify the Job to run and the version of the Job
- Change the context it should run in (if needed)



Job Orchestration - Master Jobs

tRunJob - Overriding context

The value of **any context variable** can be modified

Job(Maste... Contexts... Component Run (Job ... Tes

▶ ProductPrep2Table(tRunJob_2)

Basic settings	Schema	Built-In	Edit schema	Copy Child Job
Advanced settings	<input type="checkbox"/> Use dynamic job			
Dynamic settings	Job	ProductPrep2Table	...	Version
View	<input type="checkbox"/> Use an independent process to run subjob			
Documentation	<input checked="" type="checkbox"/> Die on child error			
Validation Rules	<input type="checkbox"/> Transmit whole context			
	Context Param	Parameters	Values	
	RetailProducts_File	"C:/StudentFiles/toto.csv"		

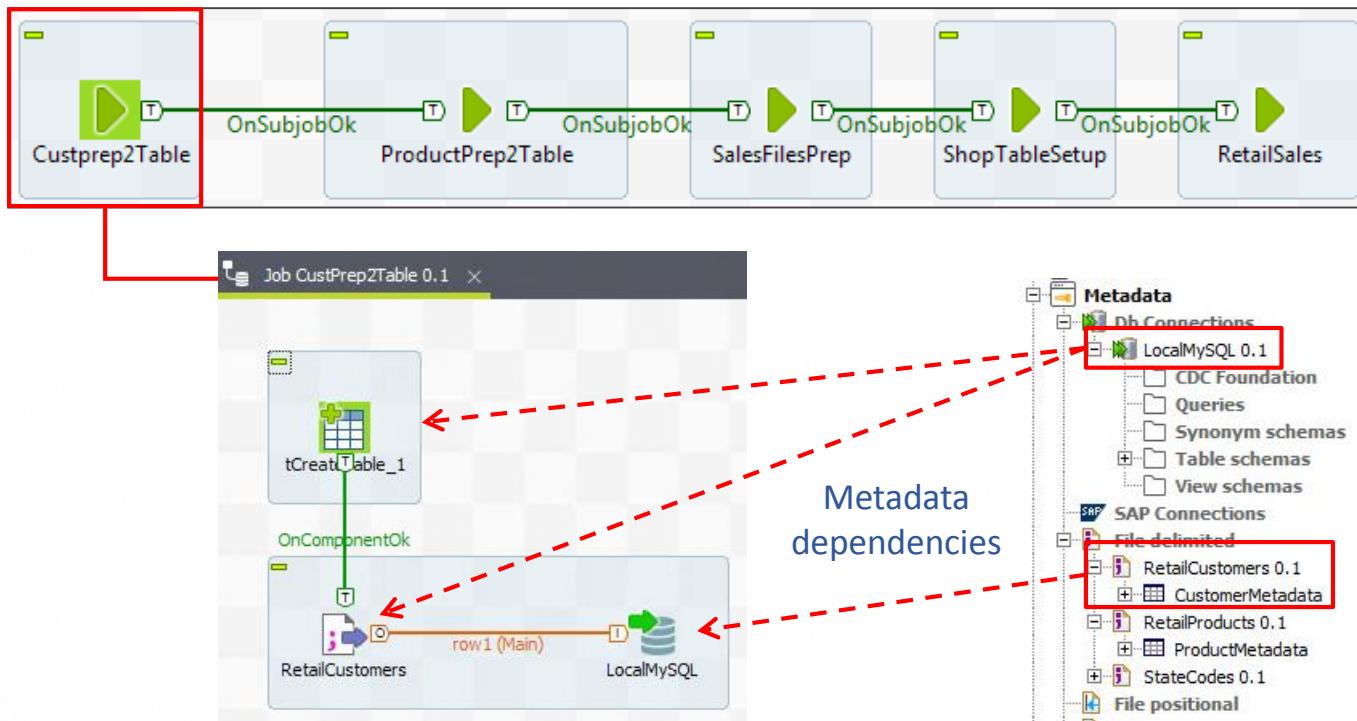
A change here is *only available* for this instance of tRunJob



Exporting Jobs

Scenario

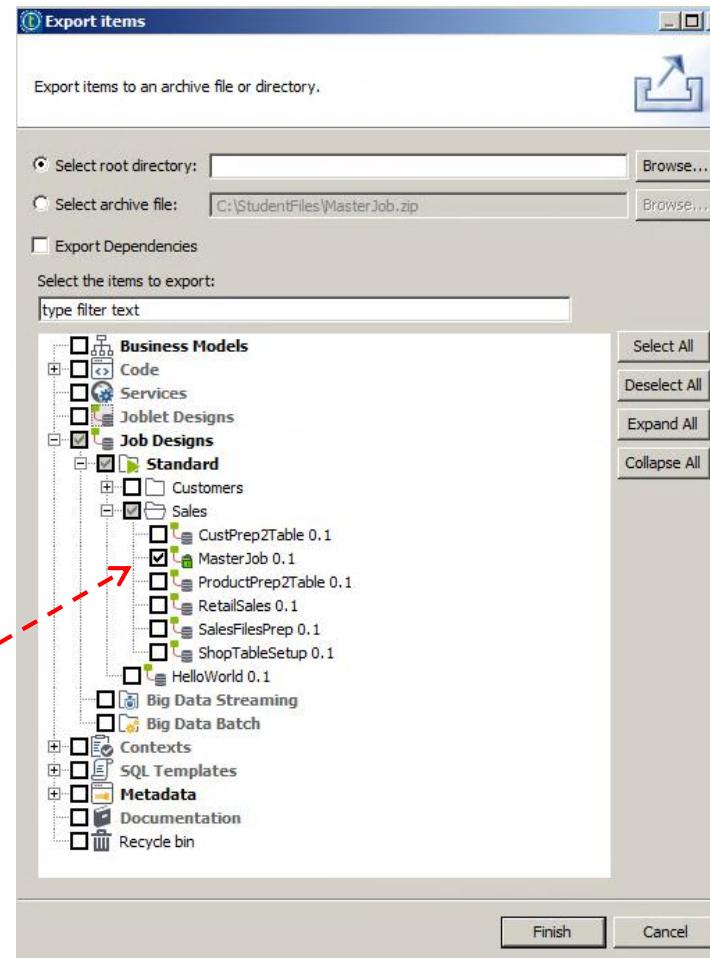
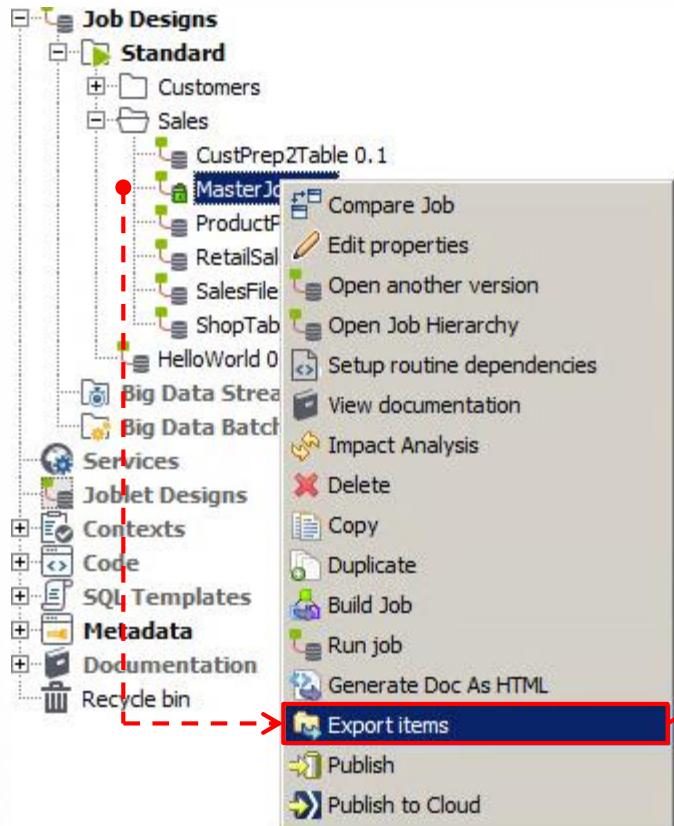
You need to share the Master Job with colleagues. It needs to be *packaged with all dependencies* in order to make it work in another Talend Studio instance:





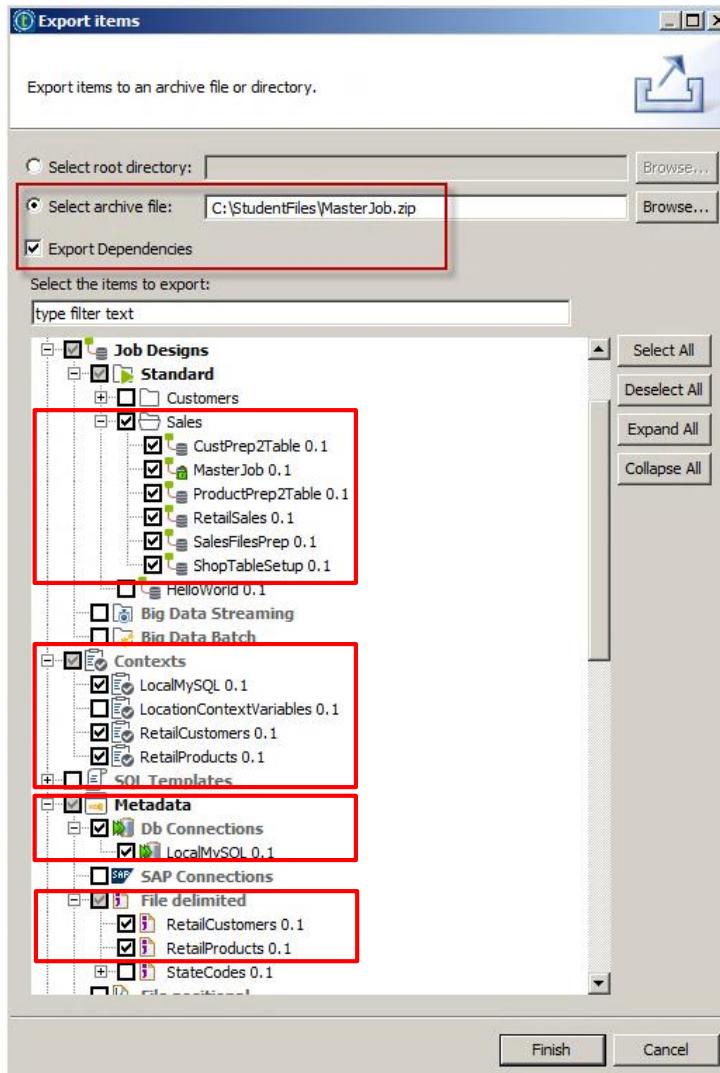
Exporting Jobs

Jobs can be exported from the Repository





Exporting Jobs



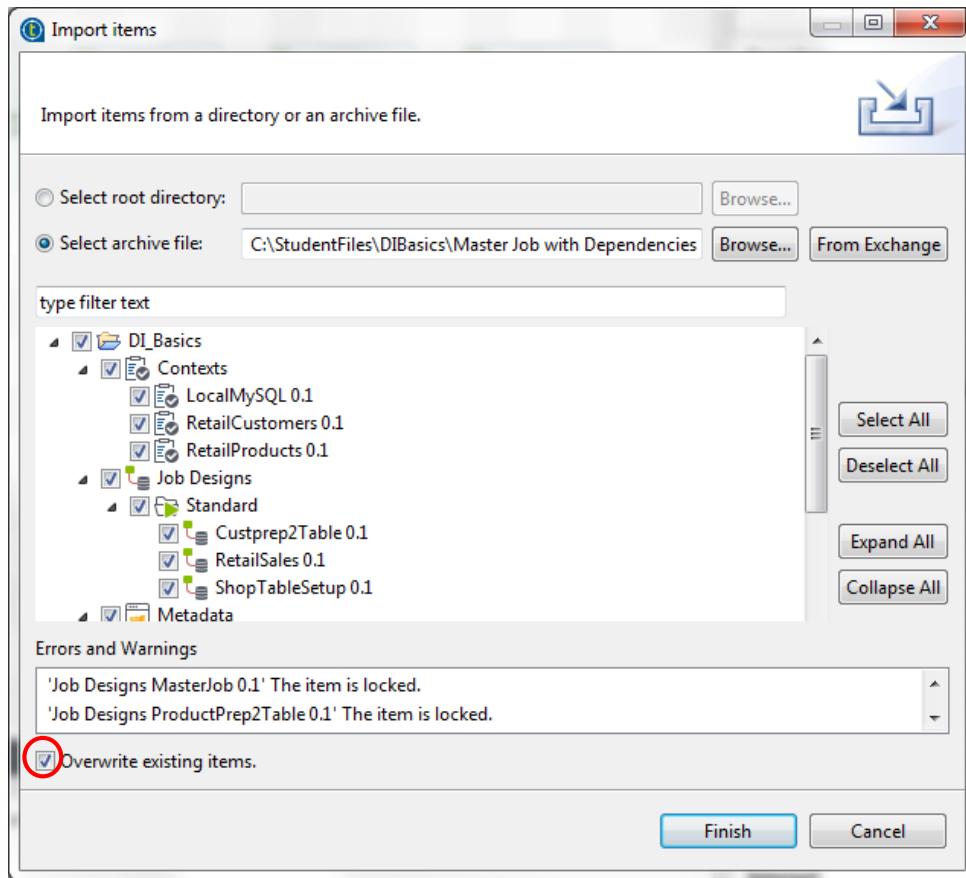
Jobs can be exported:

- As a .zip file
- With all dependencies



Importing Jobs

Jobs can be imported from the Repository



- **Select/unselect items** prior to import
- **Select Overwrite** as needed to clear Errors and Warnings (e.g. if some items already exist in the Repository)
- **Finish** is not actionable until Errors are resolved



Lab overview

Master Jobs

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



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



Lesson summary

- **tRunJob** is a key component to orchestrate Jobs
- Triggers connect Subjobs based on a condition
 - On Subjob Ok
 - On SubJob Error
- Use Export Items/Import Items to share work with others





Web Services

DI Jobs that work with Web Services



Objectives

By the end of this lesson, you will be able to:

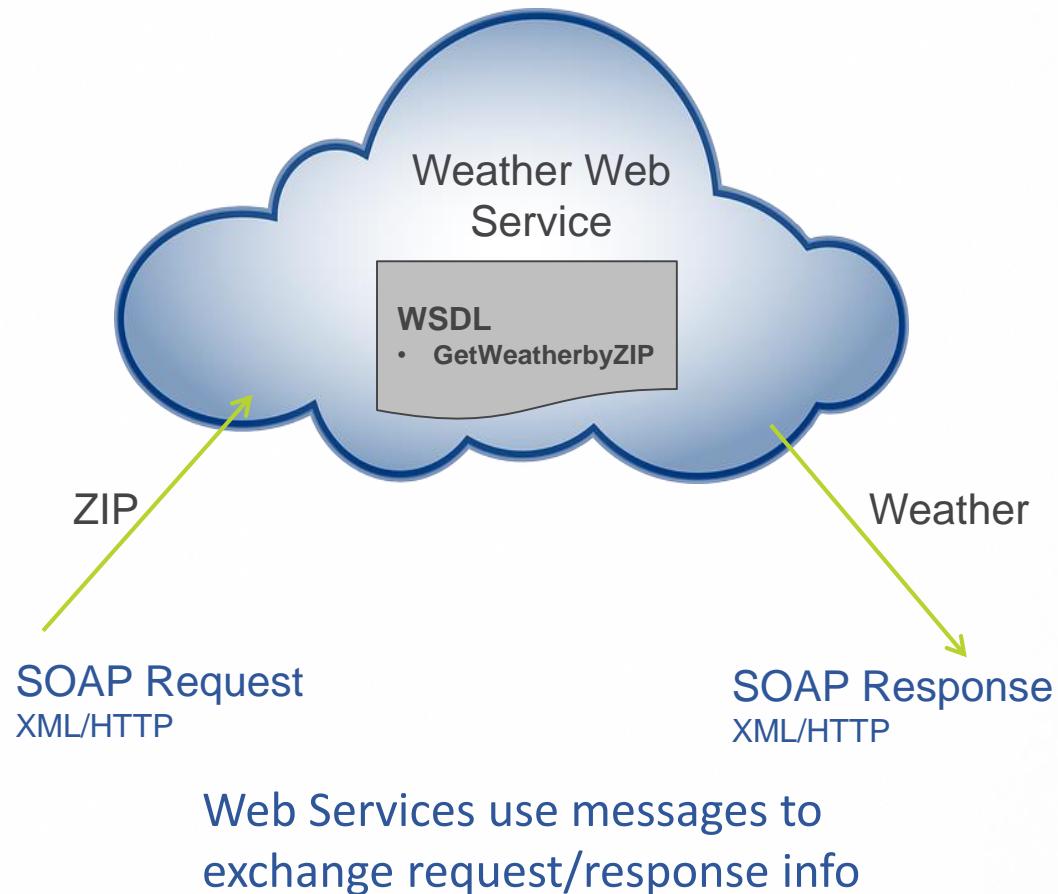
- Use a Talend component to access a Web Service method
- Extract specific elements from a Web Service reply
- Store Web Service WSDL Schemas for use in a tXMLMap component



Web Services

What are 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





Web Services Description Language

What is WSDL?

- 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

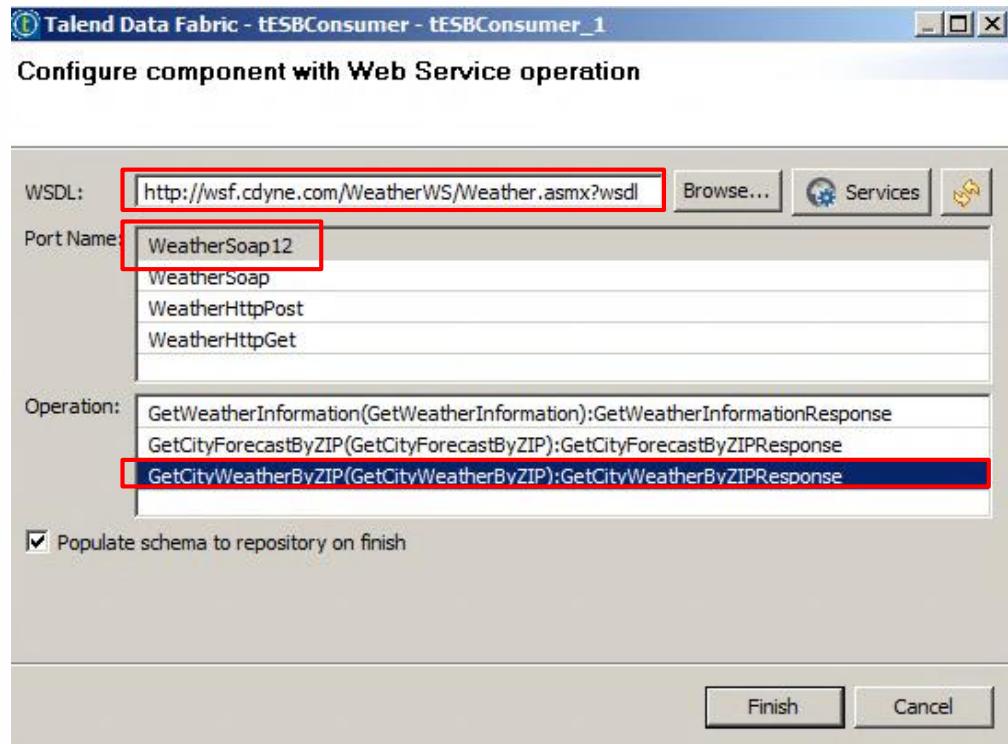


tESBConsumer

Example component configuration for the *request*

Configure the consumer to issue a SOAP request to get city weather by ZIP code

- Enter the WSDL URL
- Select Port Name
- Select Operation

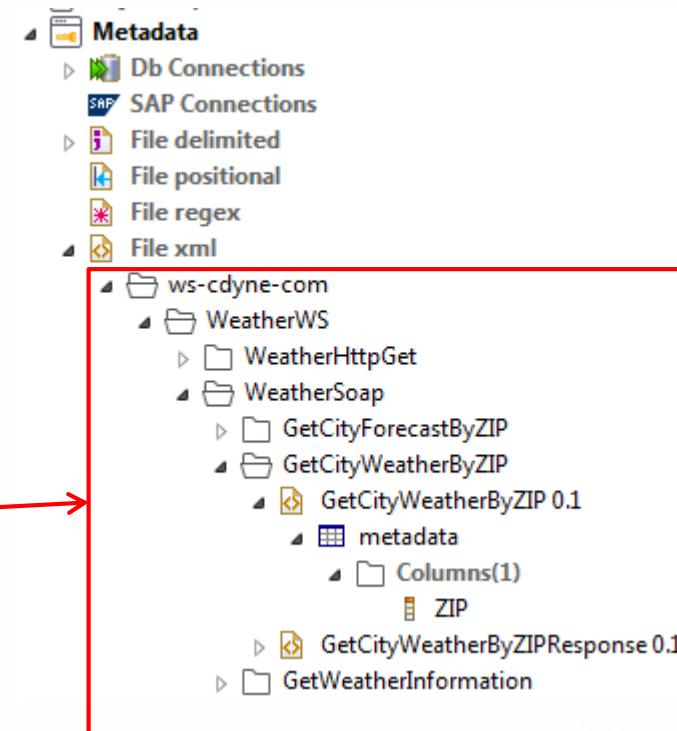
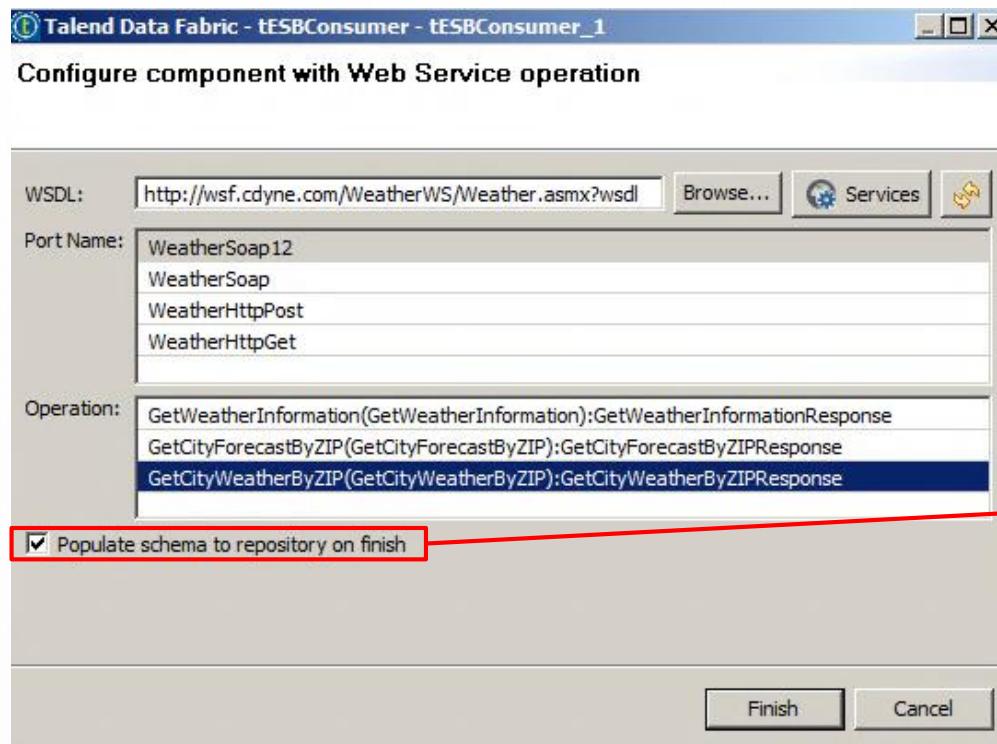




tESBConsumer

Example component configuration for processing the *response* schema

- Although the schema of the Web Service response can be complex, the **tESBConsumer** component allows you to retrieve the schema and save it in the Repository



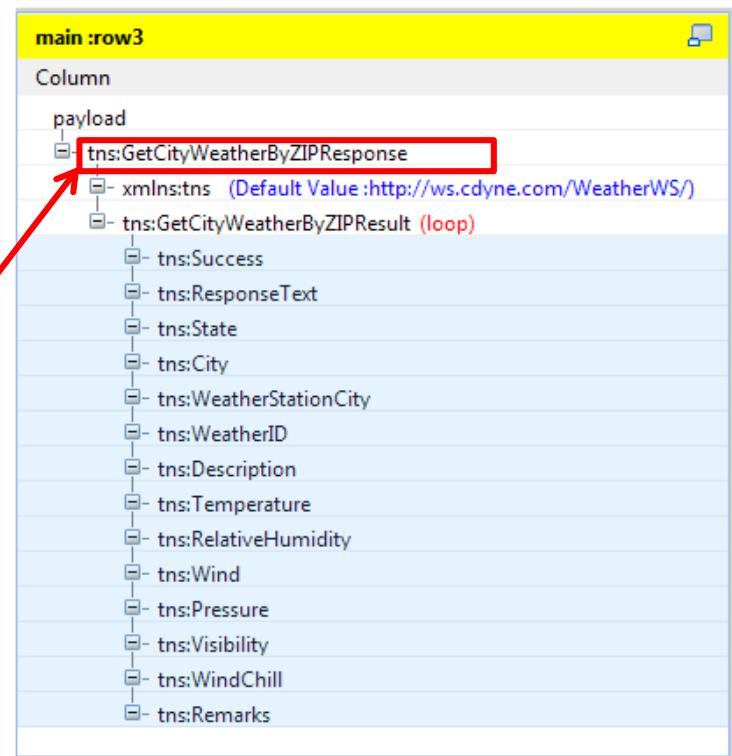
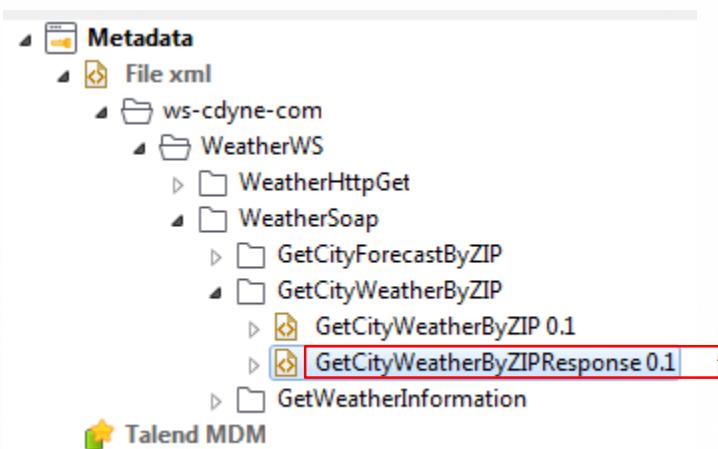


tXMLMap (1 of 2)

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





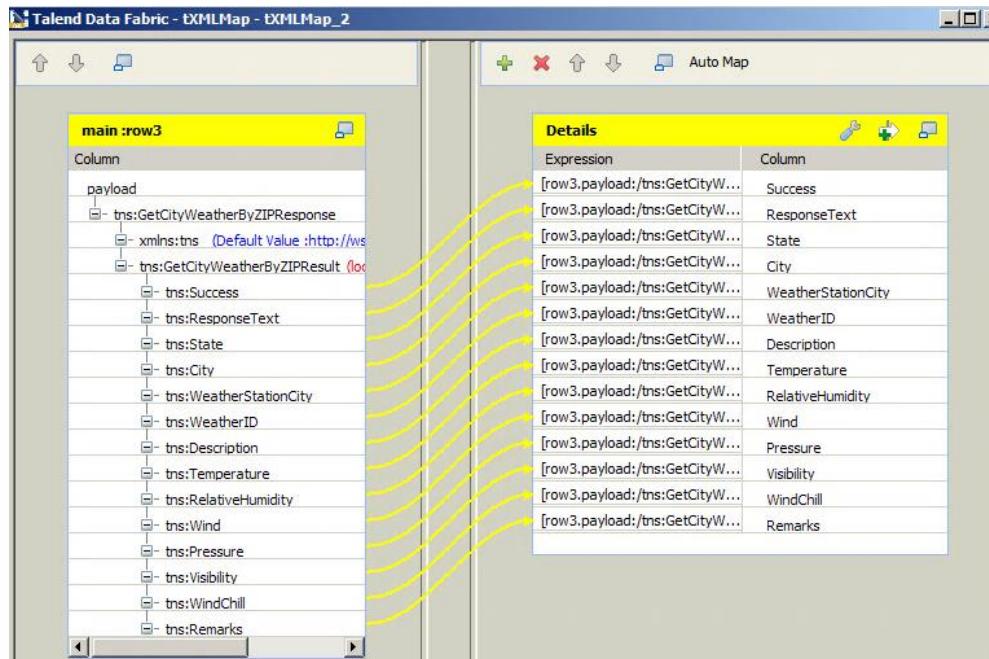
tXMLMap (2 of 2)

Process the XML Web Service response

The tXMLMap editor is similar to tMap's

The tXMLMap editor is similar to tMap's when processing the response data flow

- Route and transform data from/to one or more data sources

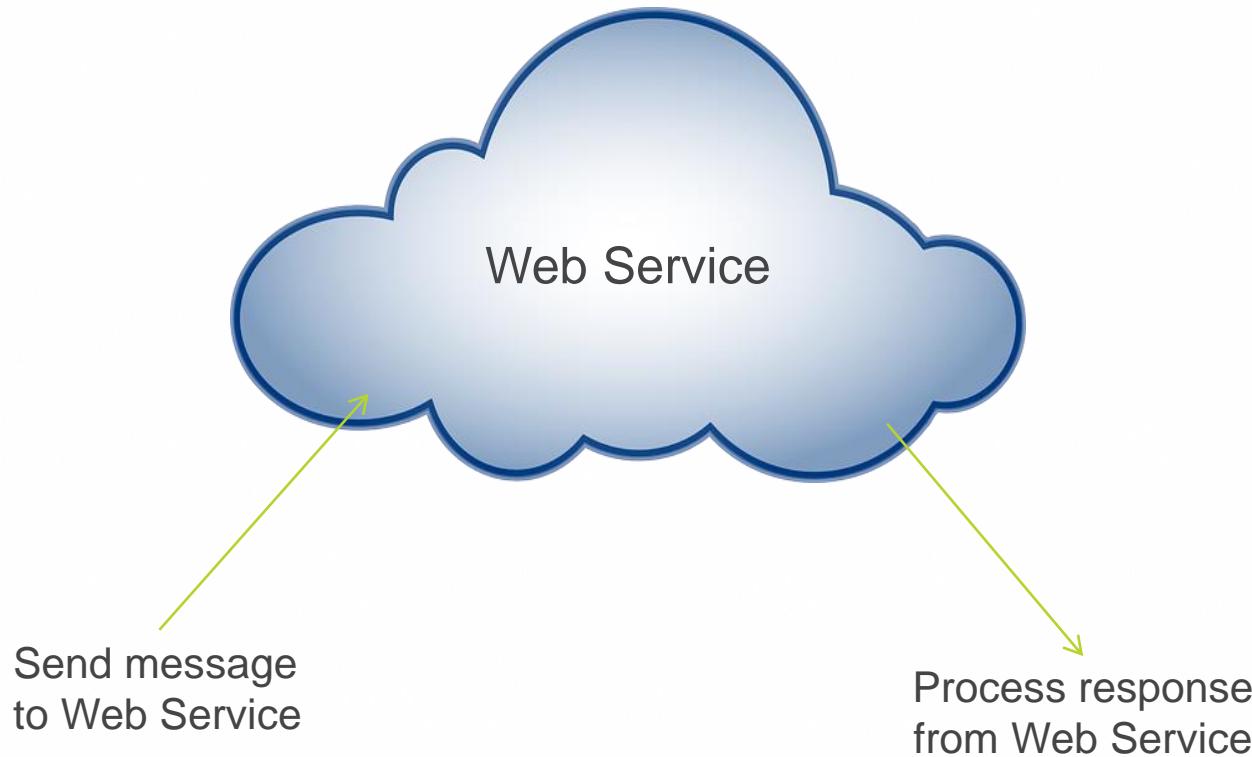




Lab overview

Web Services

- You will create a DI 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

Build and run Jobs away from the studio



Objectives

By the end of this lesson, you will be able to:

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



Building a Job

- Export Job
 - Export a Job and its dependencies
 - *Only* usable within the Studio



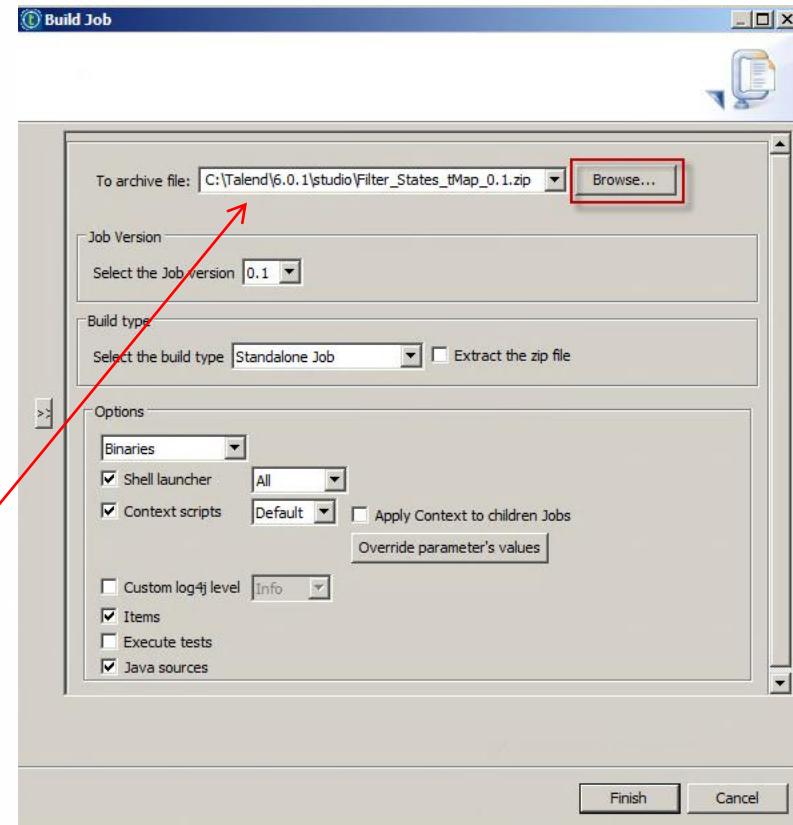
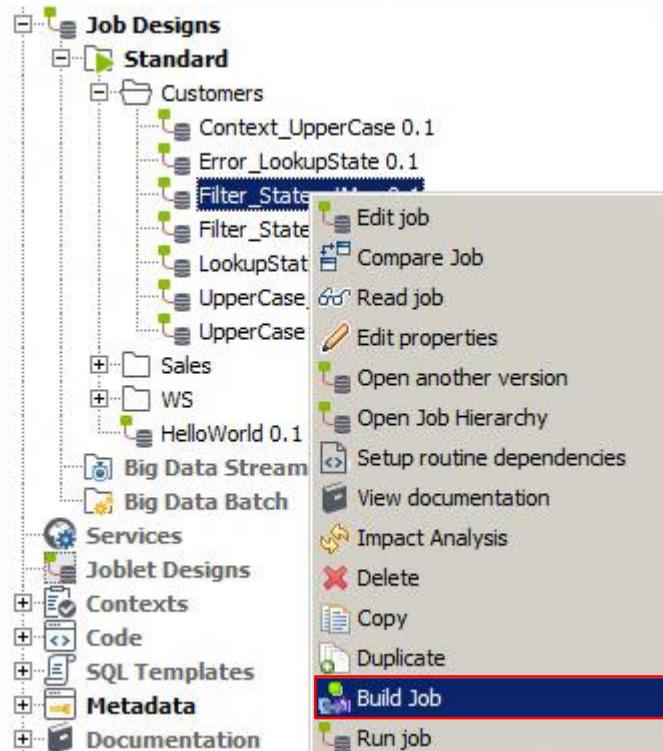
- Build Job
 - Build and deploy a Job
 - Run on *any* Server (independent of Studio)





Building a Job

- Pick the Job in the Repository and select Build Job



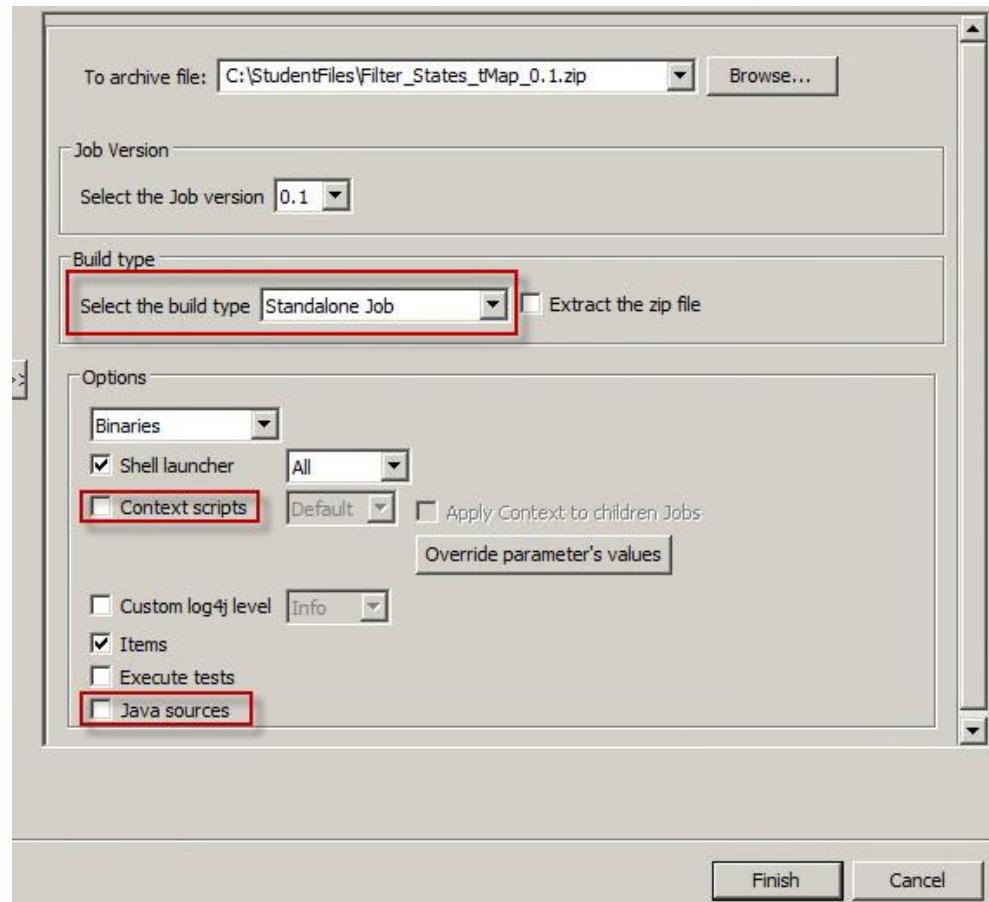


Building a Job

- Select the Build type

Select **Standalone Job**. The following will be included in the built .zip archive:

- **JAR files**
 - Java Archive Files
 - Package of files needed for deployment
- ***.bat files** - Windows batch files (for launching)
- ***.sh files** – Unix shell script files

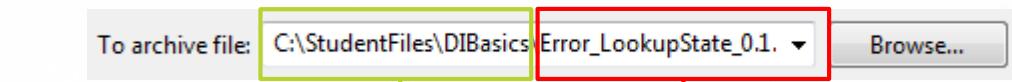




Extract and Run

Extract

- Go to Archive folder
- Extract .zip



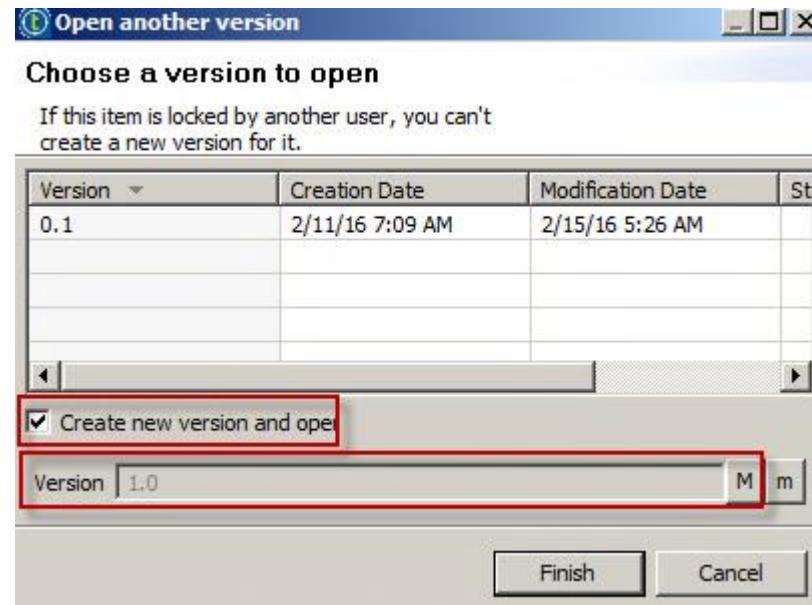
Run

- Invoke executable
 - Error_LookupState_0.1.bat (Windows)
 - Error_LookupState_0.1.sh (Linux)



Job Versioning

- Studio offers basic version control (Major.minor)
 - Create Job
 - Open another version

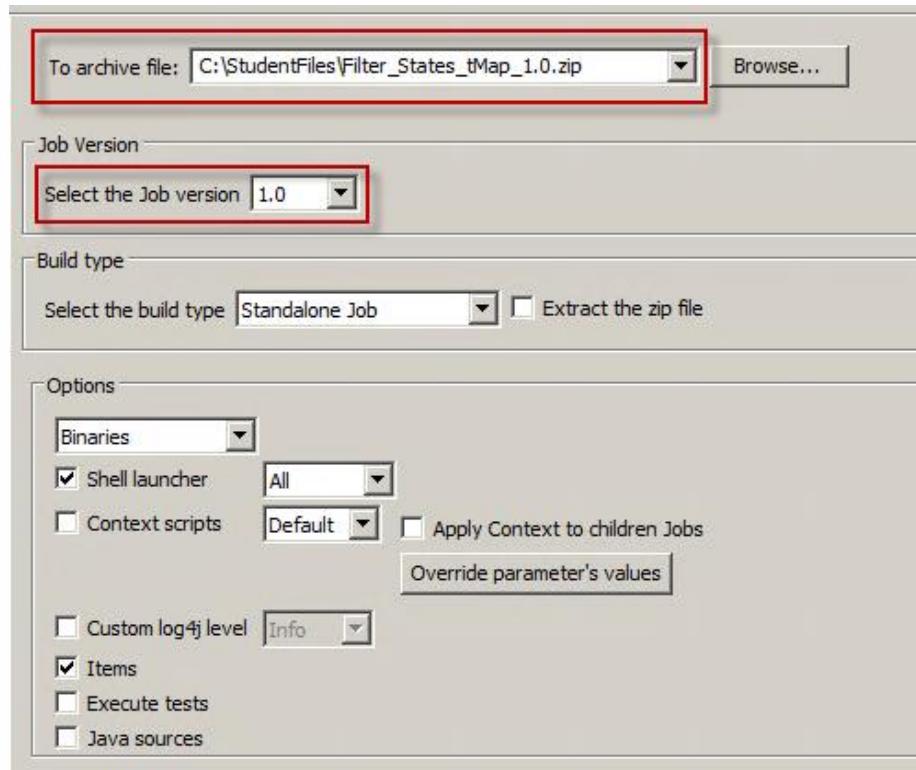


Increment Major/minor



Job Versioning

- Studio offers basic version control (Major.minor)
 - Build another version

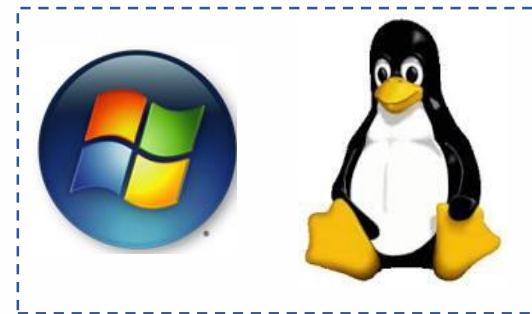




Lab overview

Stand alone Jobs

- Build Jobs that can be run outside of the Talend Studio
- Use basic version control





Lesson summary

- **Build Job** is different than **Export Job**
- Built Jobs can be run outside of the Studio
- Basic version control is easy to implement
 - Create new Job
 - Open another Version
 - Build Job





Best Practices & Documentation

Plan – Develop - Document



Objectives

At the end of this lesson, you will be able to:

- 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

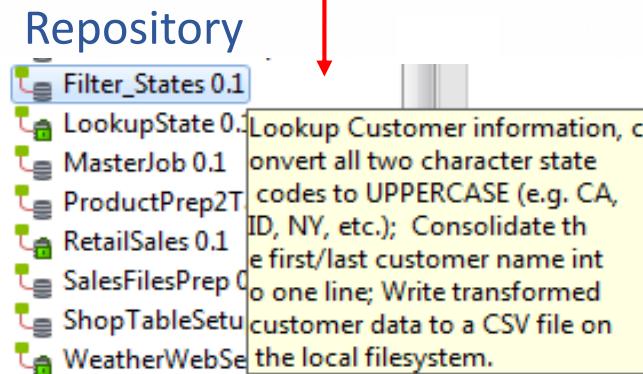
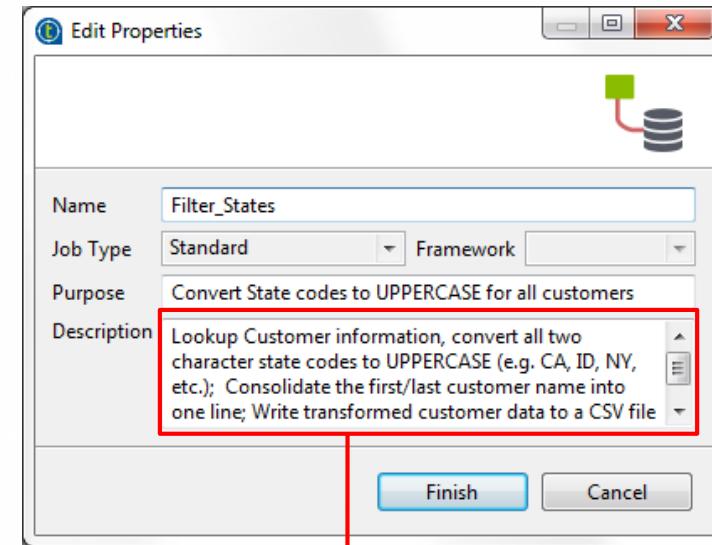
- There are a lot of inconsistencies and no documentation on Jobs developed by your team.
- Your new manager wants to implement several best practices for the development team. Most are documentation related, from Job and component naming conventions to generating HTML documentation for others departments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



Best Practices - Jobs

Name, purpose and description are all key properties that help document your Job

- Create a New Job
 - **Name** – Recommend a naming convention
 - **Purpose** – OK to keep brief
 - **Description** – More detailed, shows in repository hover-text
- Modify anytime with **Edit Properties** (Right-click Job in repository)



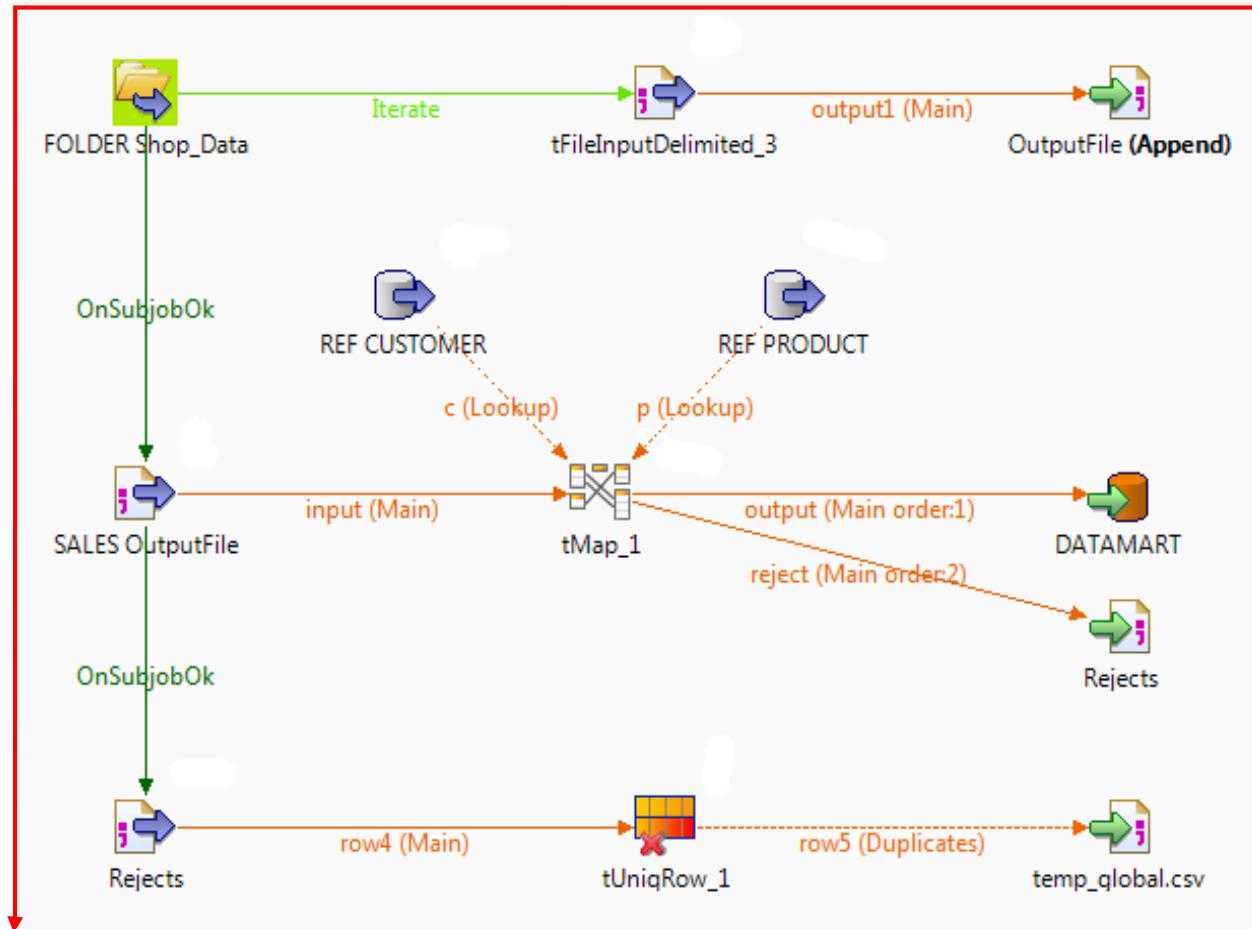


Best Practices - Jobs

Guidelines for Job layout

Work from the top. Place and connect components from left to right...

... and
top to
bottom





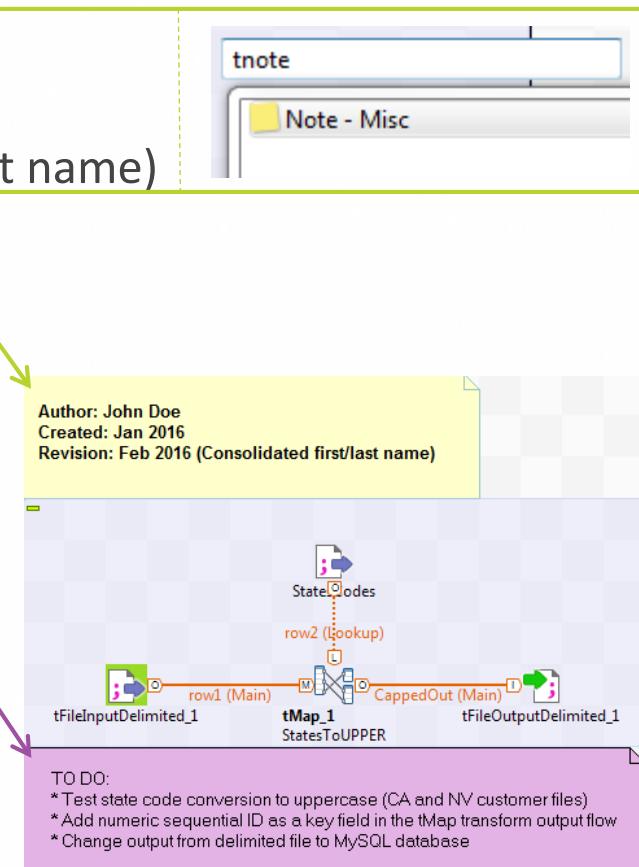
Best Practices - Jobs

Use a **Note** component to help document your Job in the design workspace

- Place a **Note** in the design workspace
- Possible format and content conventions:

- Author: John Doe
- Created: Jan 2016
- Revision: Feb 2016 (Consolidated first and last name)
- To Do list: Test state code ...

- Explore basic format options
 - Font, size, **bold**, *italics*
 - Opacity
 - Colors (text, border, fill)
 - Vertical/Horizontal alignment





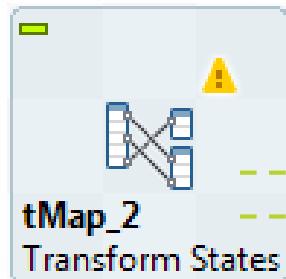
Best Practices - Components

Customizing the component label improves readability in the design workspace

Field	Description	Default Value
Label format	Text label shown in design workspace	<code>__UNIQUE_NAME__</code>

Customized example:

```
<b>__UNIQUE_NAME__</b><br>Transform States
```



At a glance:

- Icon (develop familiarity)
- Component Name (in bold)
- Primary function



Best Practices - Components

The Hint format provides helpful component information in the workspace

Field	Description	Default Value
Hint format	Hidden tooltip. Visible when hover-over component <i>and</i> Documentation “show information” selected	__UNIQUE_NAME__ __COMMENT__

- Documentation > Show information

Label format `__UNIQUE_NAME__
Customers from Alaska`

Hint format `Only output where the State = "AK" (Alaska) in the Customer table`

Basic settings Show Information

Advanced settings

Dynamic settings

View

Documentation

Validation Rules

Comment

Only output where the State = "AK" (Alaska) in the Customer table

Customers from Alaska



View Documentation

Use View documentation in the Studio during the development phase of your Job

- Repository > *JobName* > View documentation
 - Opens a tab with HTML documentation for review



The screenshot shows a browser window titled "Filter_States_tMap_1.0.html". The page is titled "Job Documentation" and includes the Talend logo. It states "Generated by Talend Data Fabric". Below this is a table with project metadata:

Project Name	DI_Basics	GENERATION DATE	Feb 10, 2016 4:36:57 PM
AUTHOR	user@talend.com	Talend Data Fabric VERSION	6.1.1.20151214_1327

Under the "Summary" section, there is a list of hyperlinks:

- [Project Description](#)
- [Description](#)
- [Preview Picture](#)
- [Settings](#)
- [Context List](#)
- [Component List](#)
- [Components Description](#)

Actionable hyperlinks with additional content

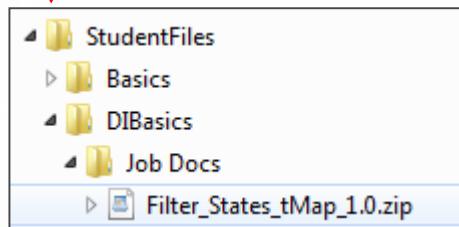
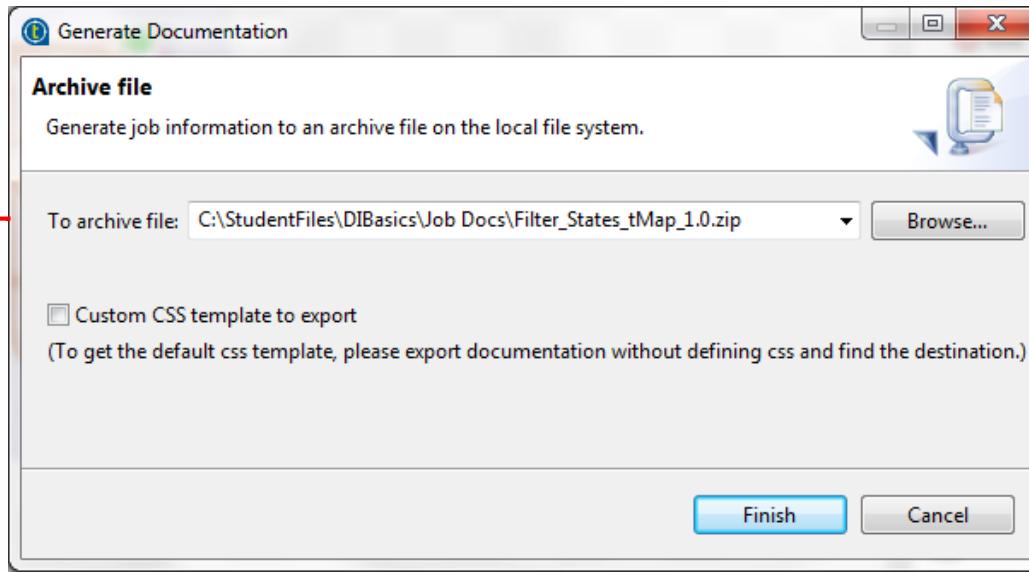
Actionable
hyperlinks with
additional content



Generate Documentation as HTML

Once the development phase for your Job is completed, generate the documentation

Repository > *JobName* > Generate Doc as HTML



Unzip
→

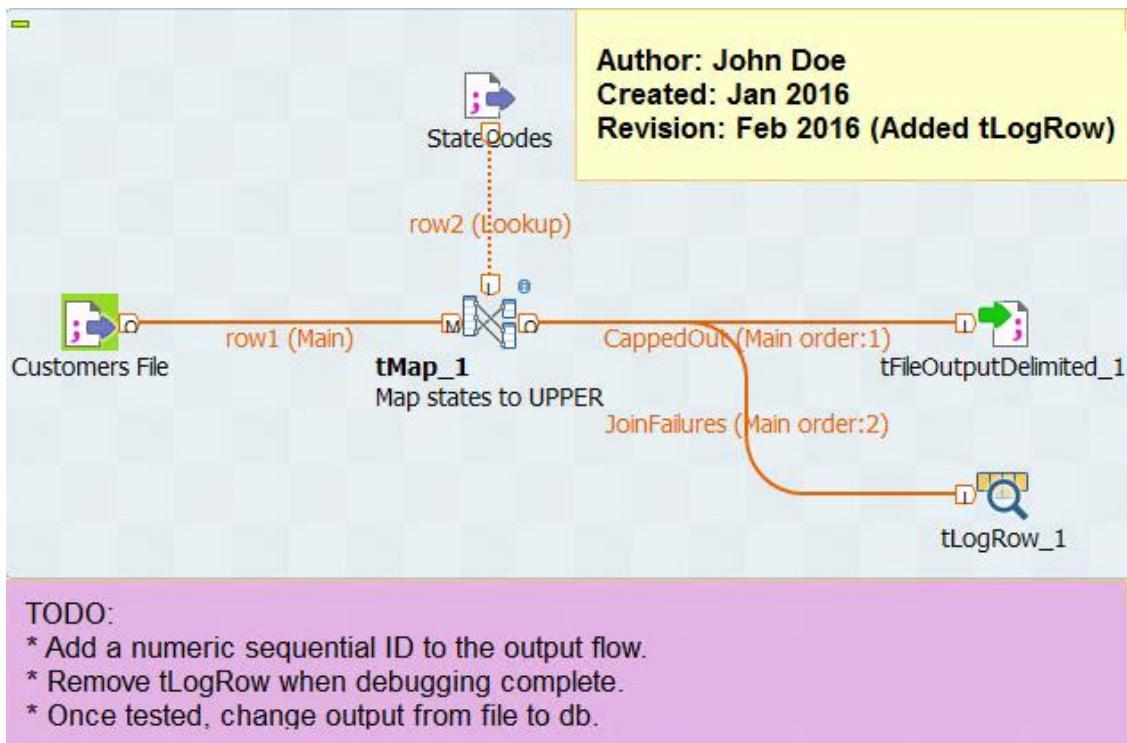
Name	Date modified	Type	Size
pictures	2/10/2016 4:49 PM	File folder	
default.css	2/10/2016 4:47 PM	Cascading Style S...	2 KB
Filter_States_tMap_1.0.html	2/10/2016 4:47 PM	Firefox HTML Doc...	81 KB
Filter_States_tMap_1.0.xml	2/10/2016 4:47 PM	XML Document	30 KB
tMap_1.xml	2/10/2016 4:47 PM	XML Document	6 KB



Lab overview

Best Practices and Documentation

- Use several best practices while developing a Job
- View HTML Job documentation in the Studio
- Generate and view documentation external to the Studio





Lesson summary

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

1. Planning

- Business Model

2. Development

- Naming conventions, labels and hints

3. Documentation

- Generate and View HTML

