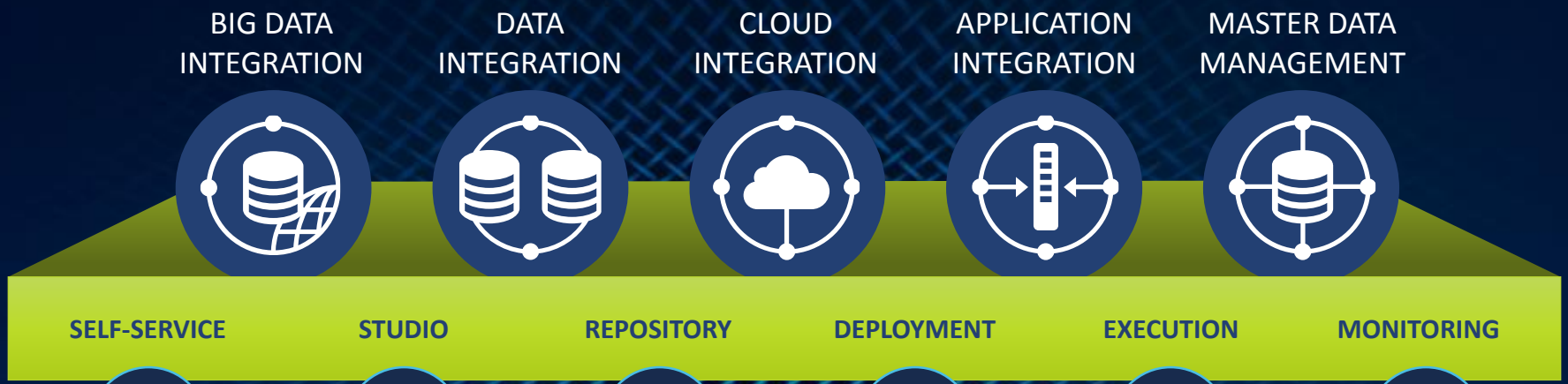
Data Integration Basics

Version 6.3

talend | Data Fabric



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

talend | This course focuses on:

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

Course Table of Contents

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

Introduction to Talend Data Integration

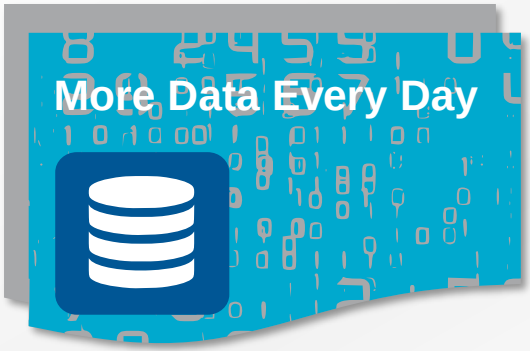
Version 6.3

Objectives

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

Why Data Integration?

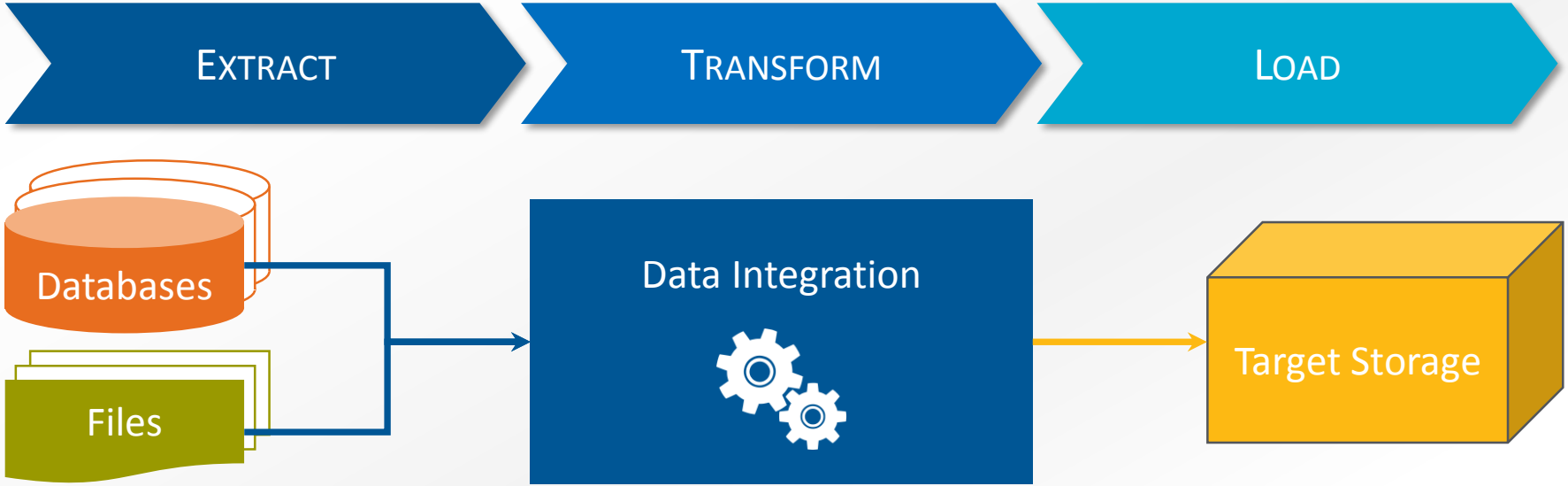
- Different formats
- Different database structures and models
- Different standards
- Different systems



- Removes mistakes and corrects data
- Adjusts data from multiple sources to be used together
- Structures and aggregates data to be used by BI tools

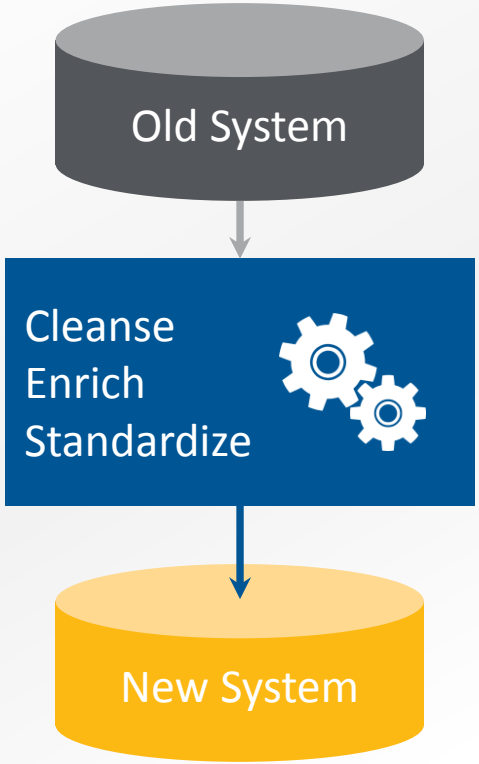
What is Data Integration?

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



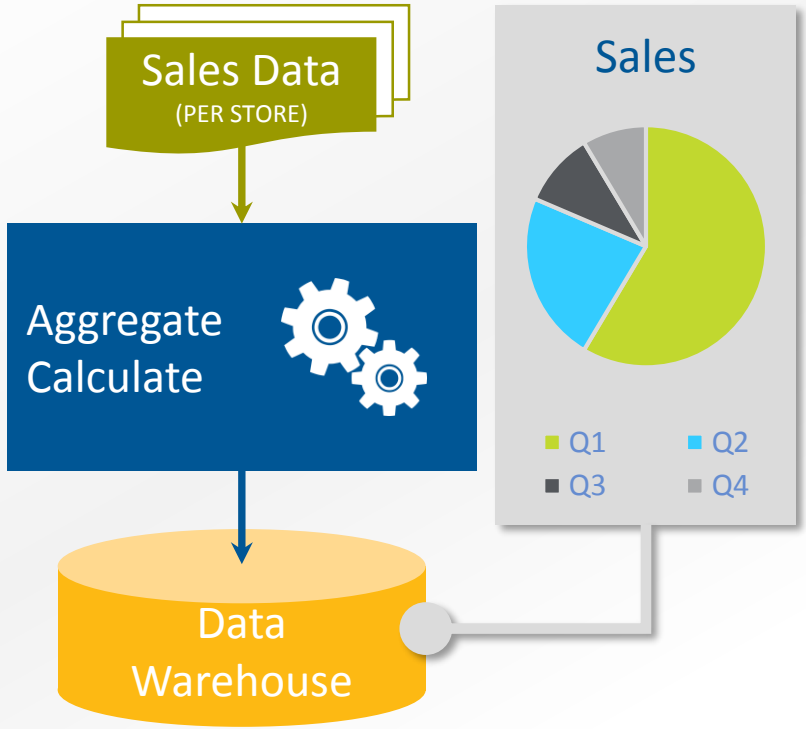
Typical Use Cases

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



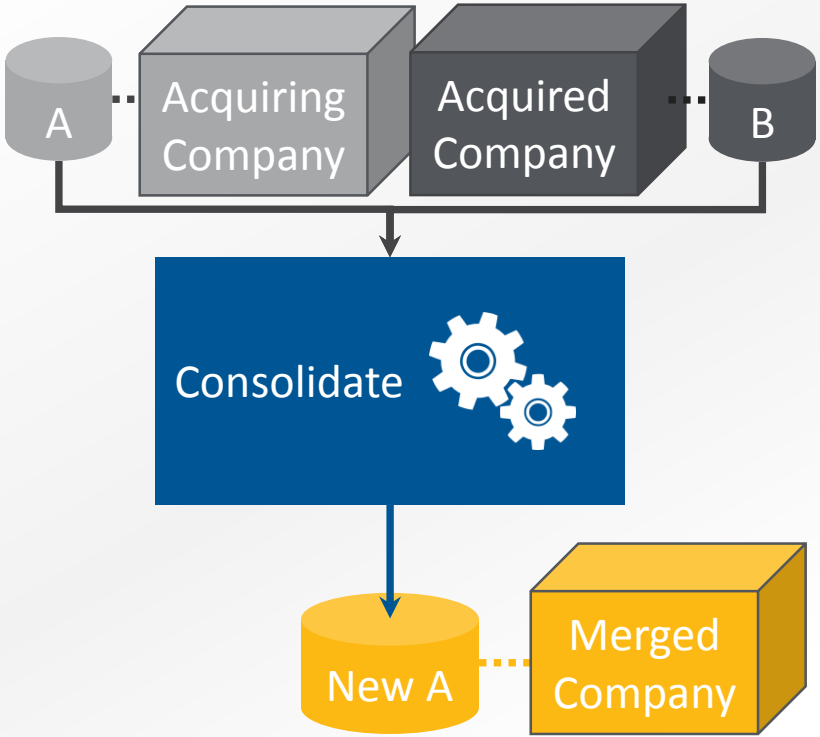
Typical Use Cases

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



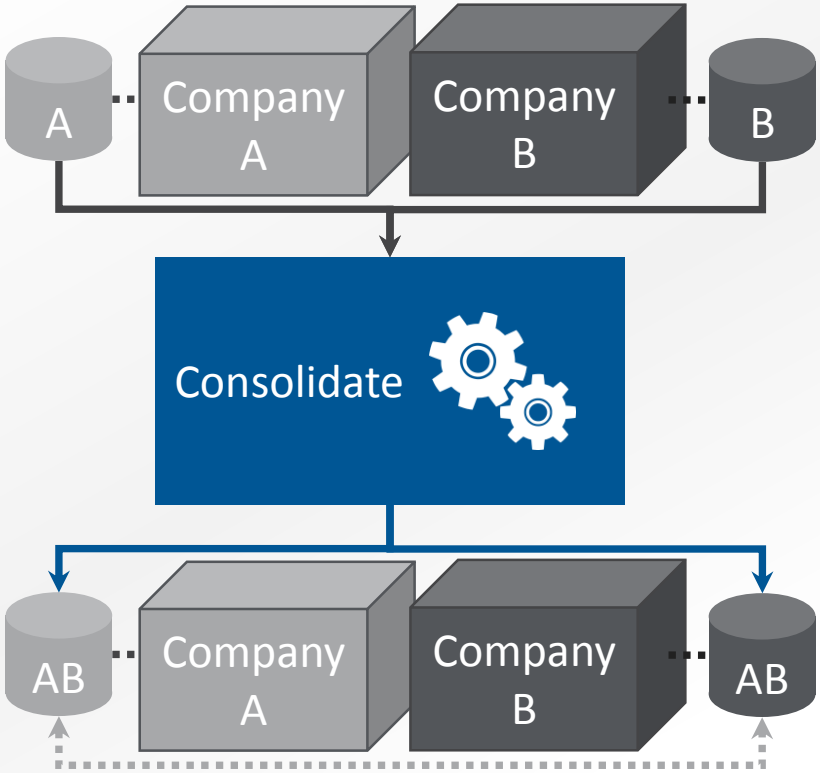
Typical Use Cases

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



Typical Use Cases

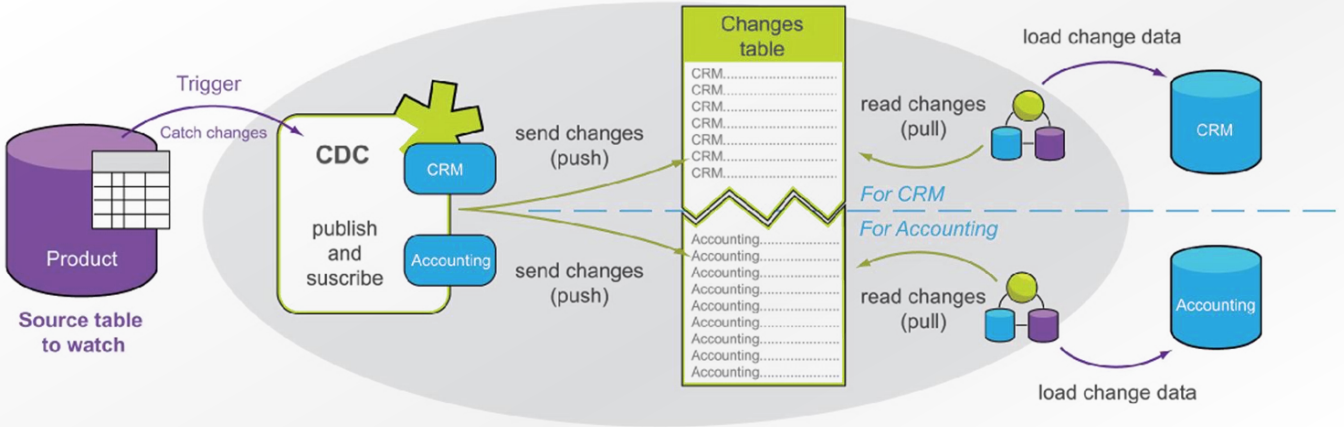
- Data Migration
- Data Warehousing
- Data Consolidation
- Data Synchronization



Integration Using Delta Only

Change Data Capture (CDC) Scenario

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

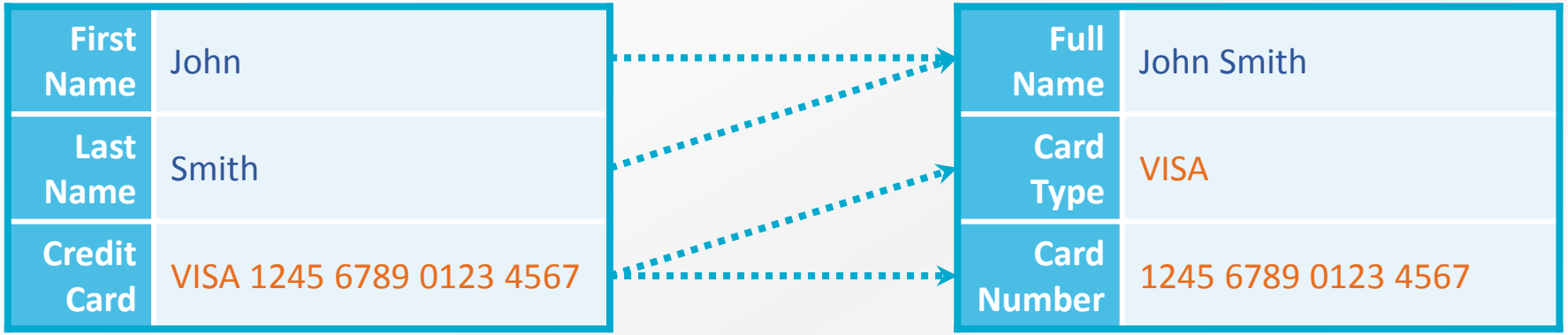


Key benefits of Data Integration

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

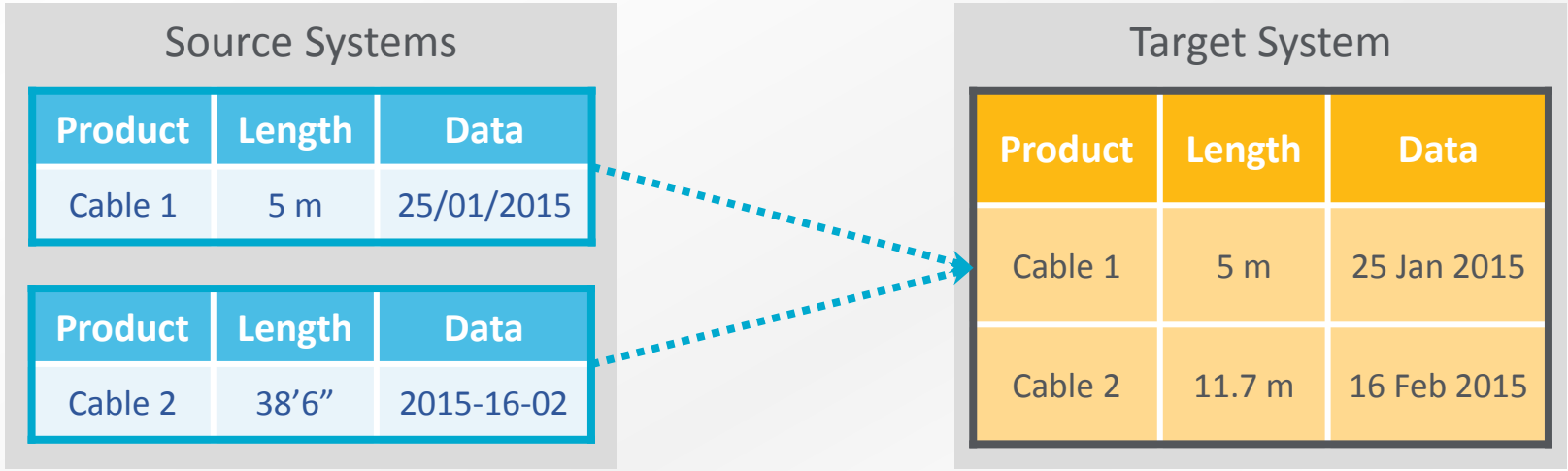
Data Integration Power

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



Data Integration Power

- Data standardization
  - Dates
  - Units of measurement



Data Integration Power

- Join data
  - Complete data with information from external sources

Source System

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

```
Zip Code,City
10004,New York,NY
10005,New York,NY
10006,…
```

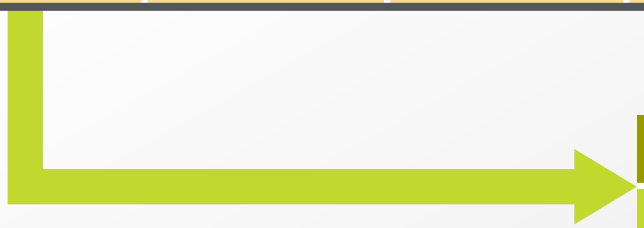
Target System

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

Data Integration Power

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

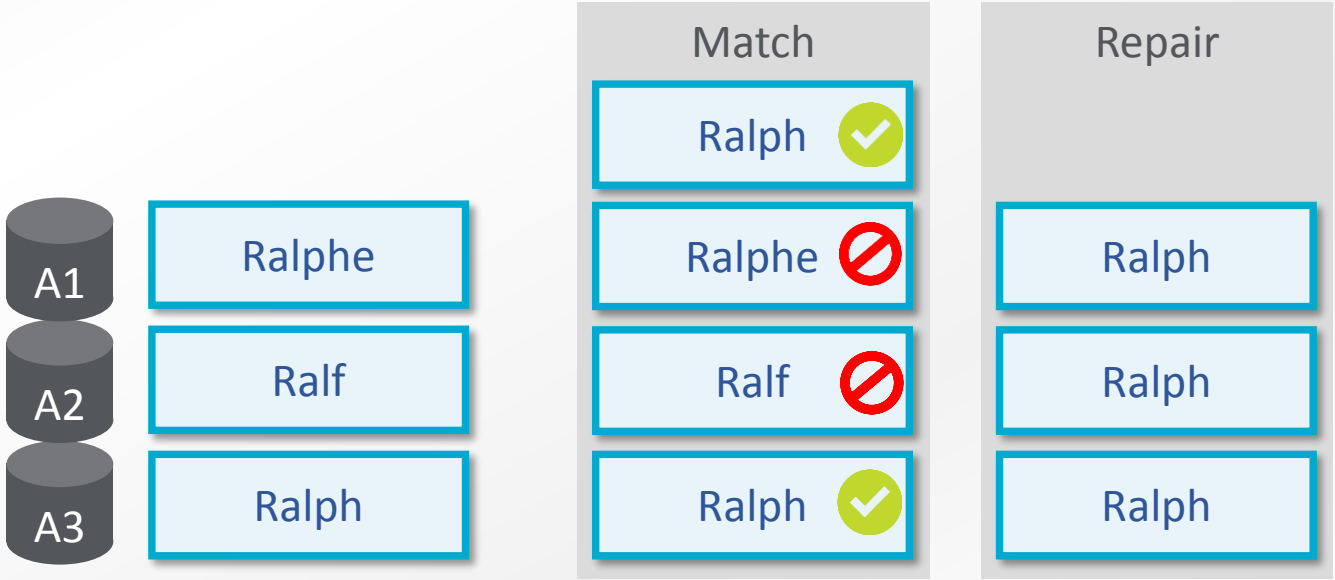
Product	Quantity	Zip Code	City	State	Customer
P167987	13	10004	**New York**	**NY**	C987674
P167987	16	10005	**New York**	**NY**	C987856



Product	Quantity	City
P167987	29	New York

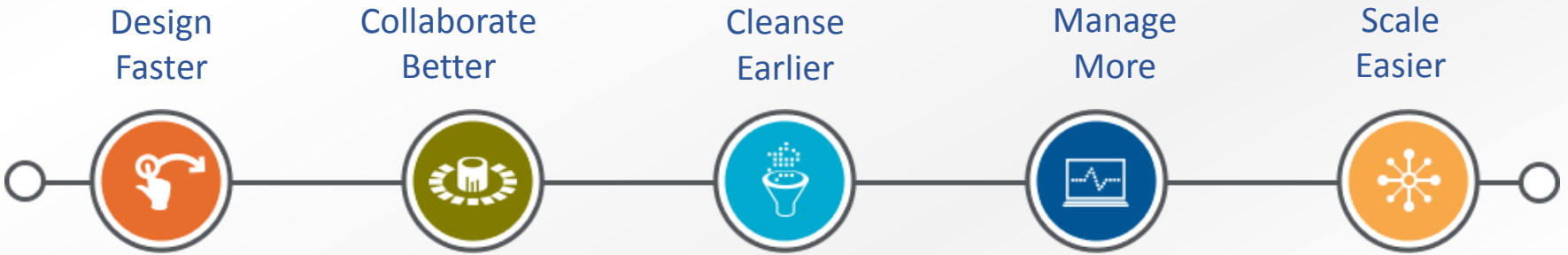
Data Integration Power

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



Why Talend Data Integration?

Respond Faster to Business Needs



Getting Started

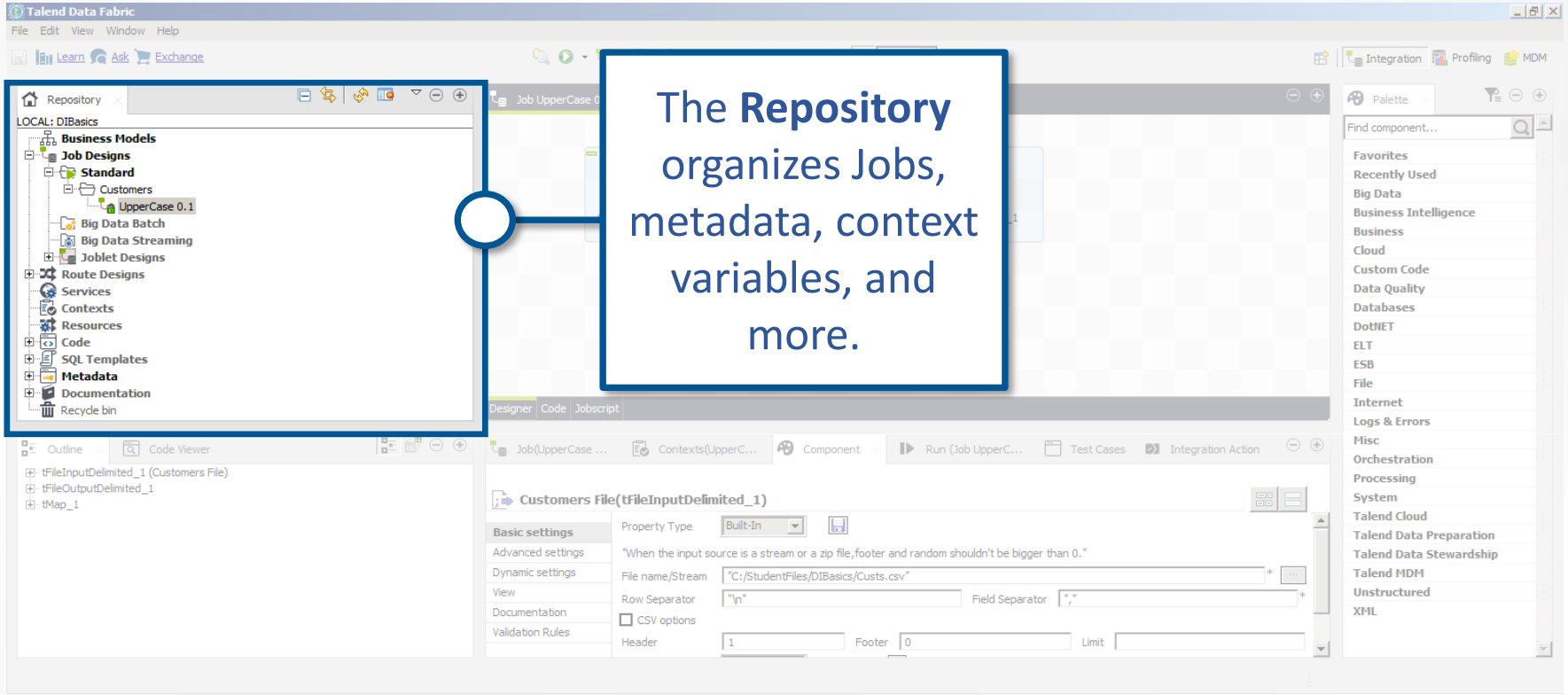
Talend Studio for Data Integration

Objectives

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

Talend Studio Main Window

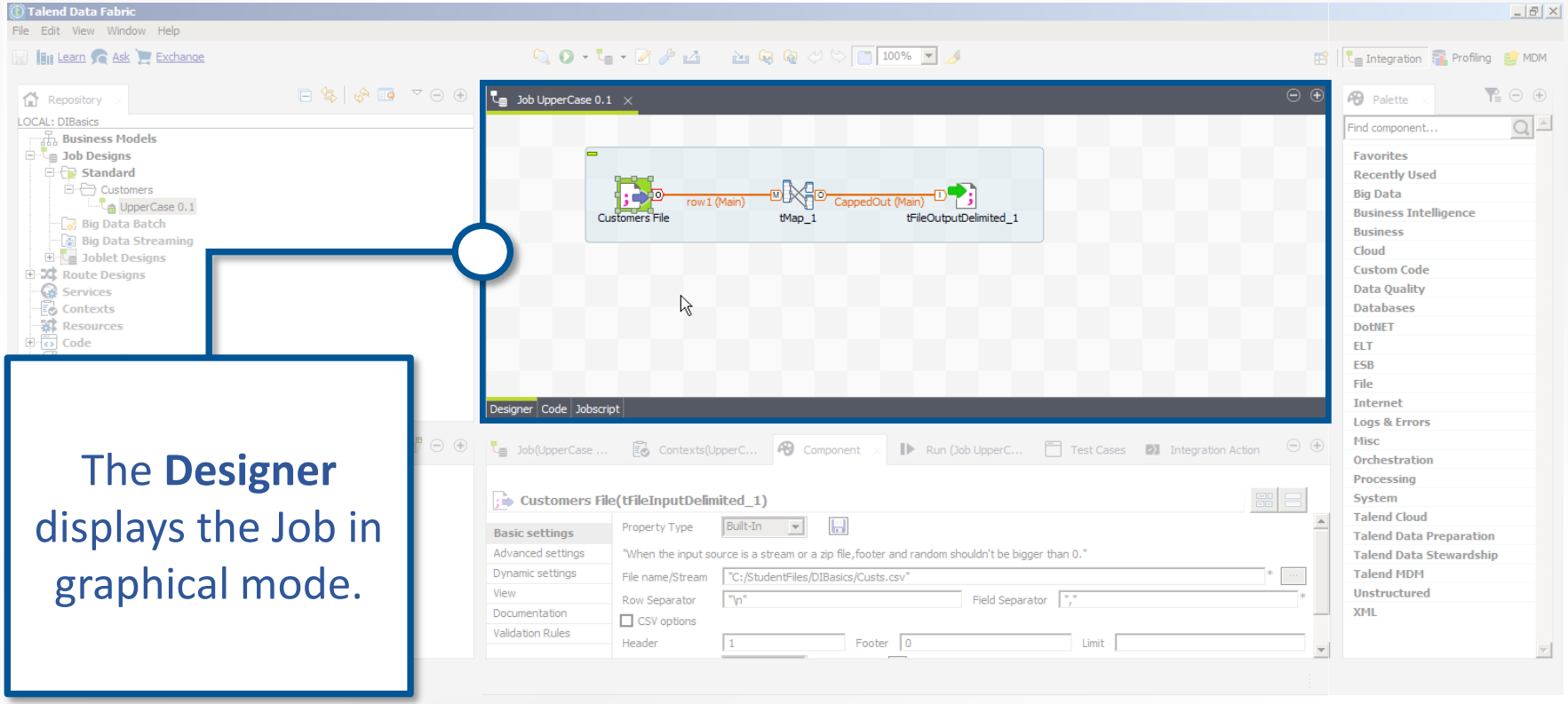
Repository



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

Talend Studio Main Window

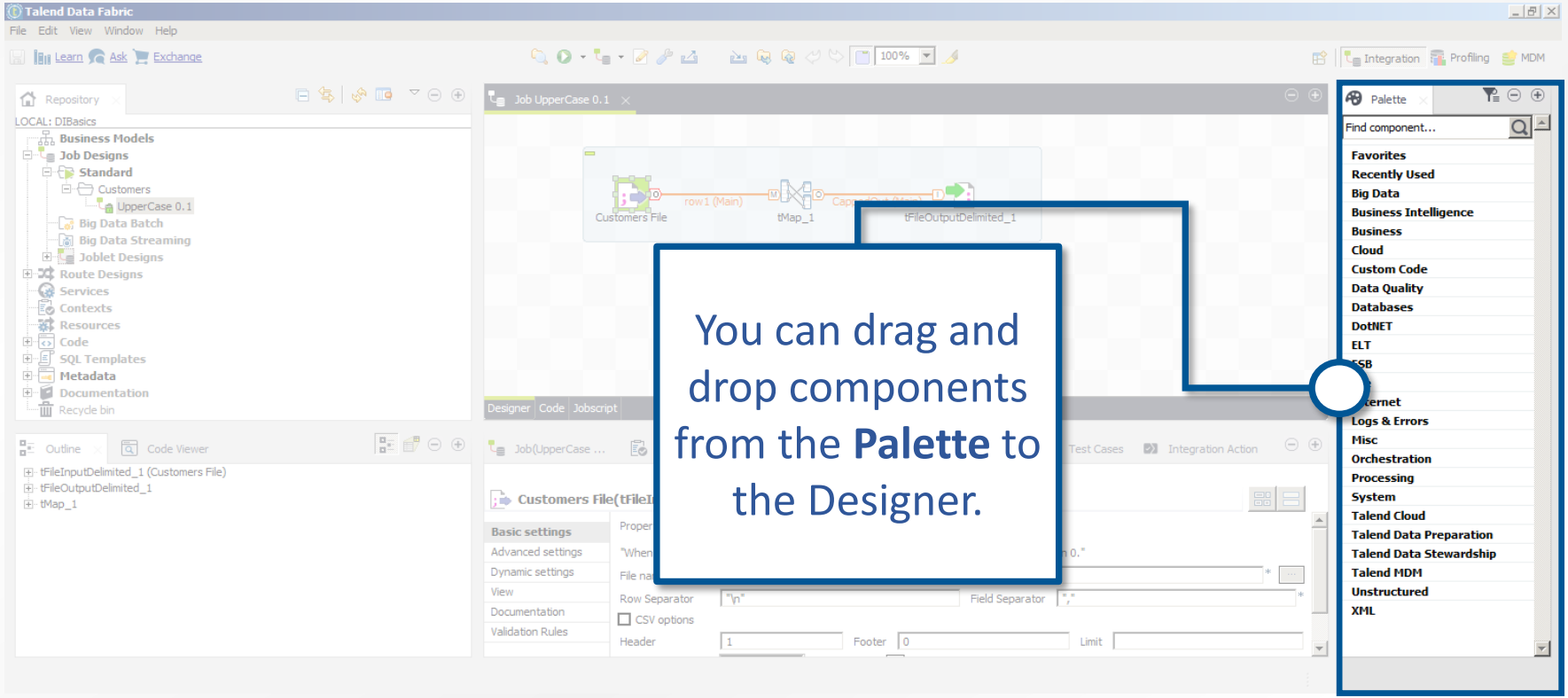
Designer



The **Designer** displays the Job in graphical mode.

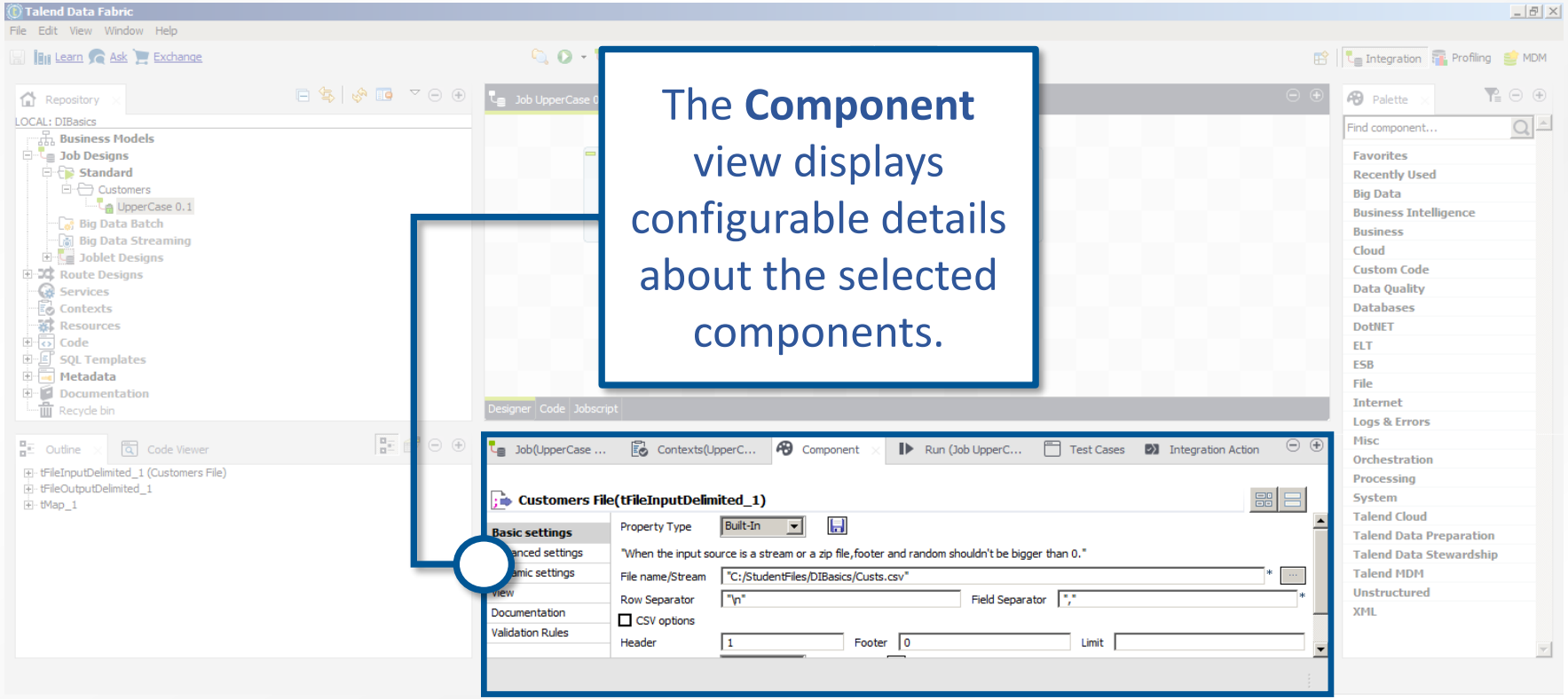
Talend Studio Main Window

Palette



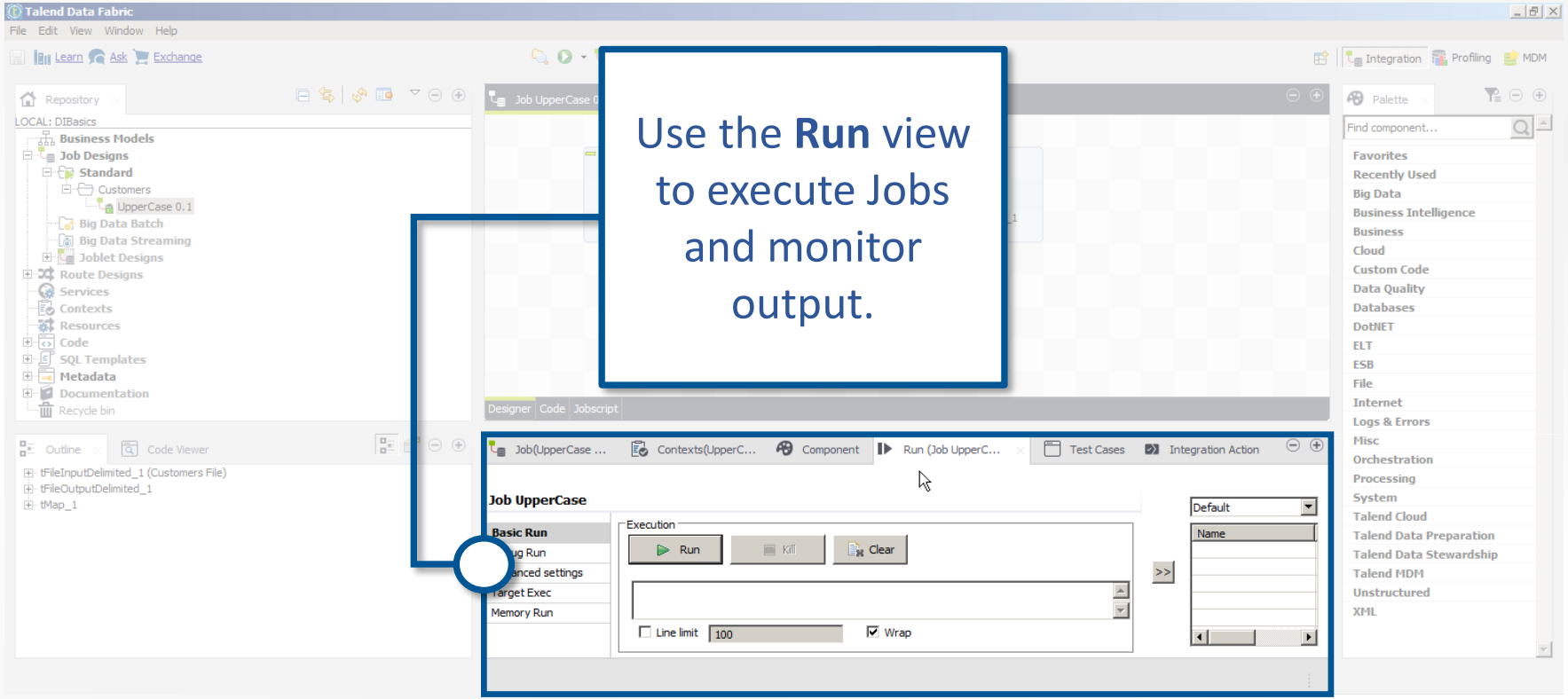
Talend Studio Main Window

Component View



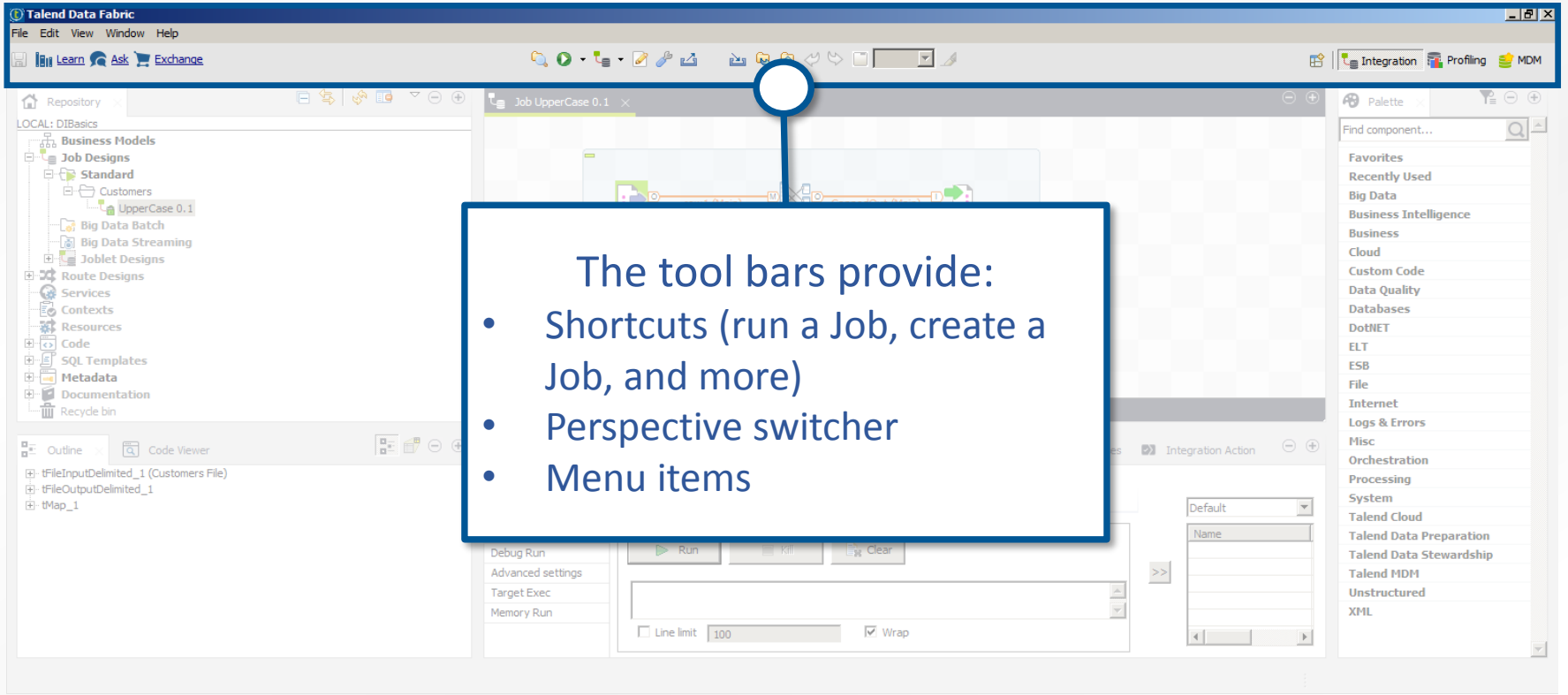
Talend Studio Main Window

Run View



Talend Studio Main Window

Run View

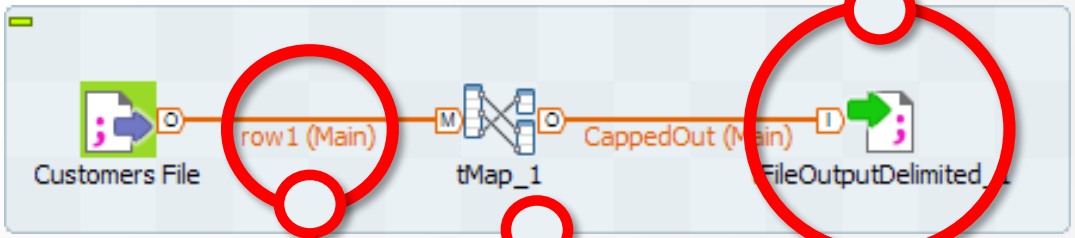


The tool bars provide:

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

What Is a Talend Job?

A **component** is the executable part of a Job. Talend provides hundreds of ready-to-use components to accomplish specific tasks.



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

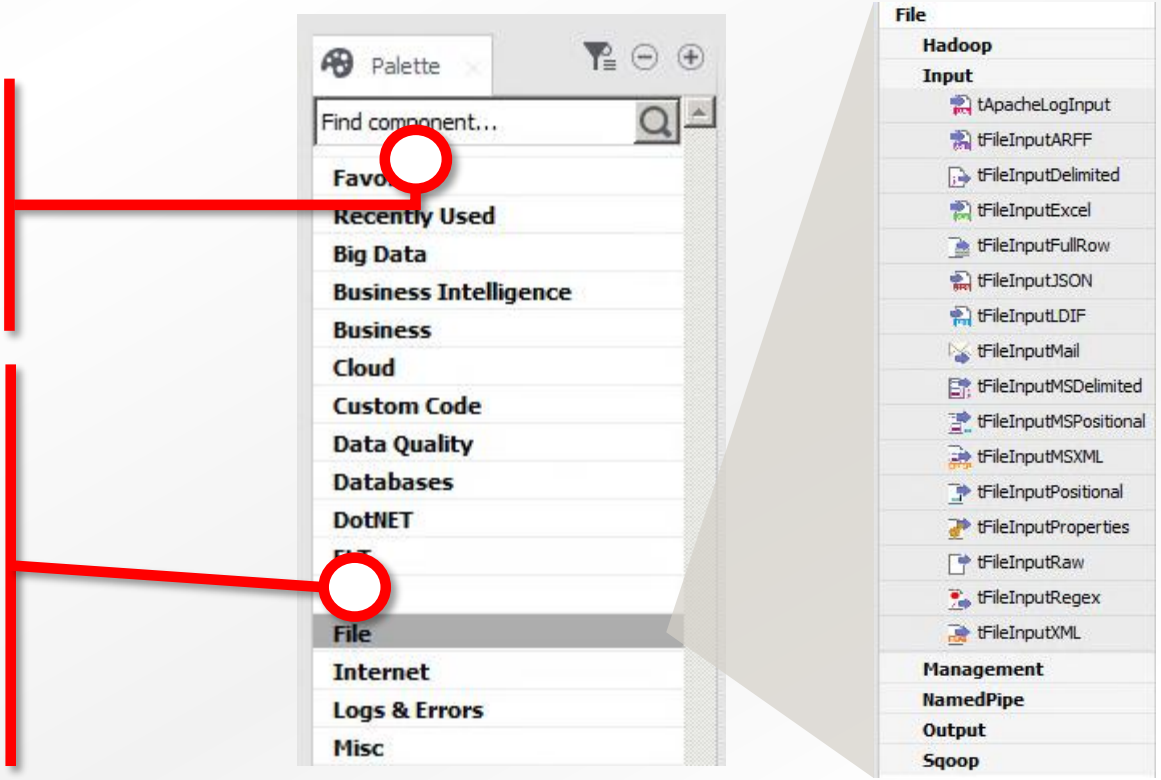
A **Job** consists of one or more connected components that generates Java code.

Components

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

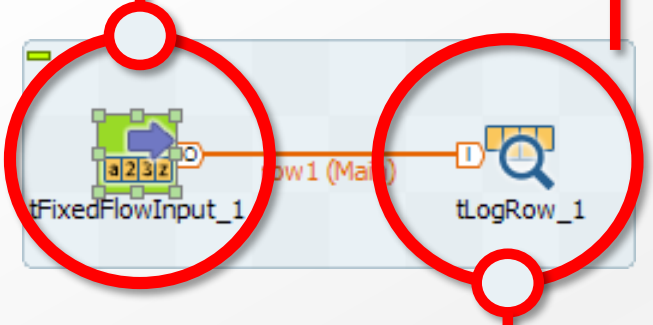
Components are organized by family. For example:

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



Build a Job

Example **Hello World** Job



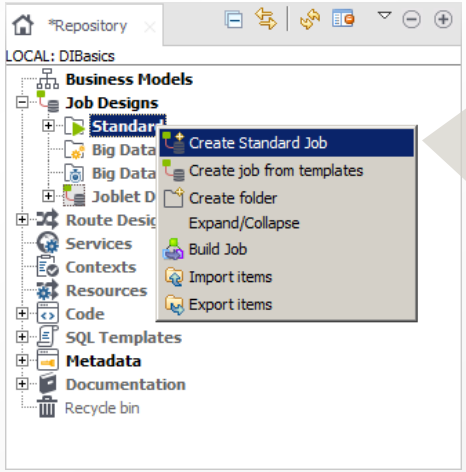
A **tFixedFlowInput** component is used to define a message to pass on to another component.

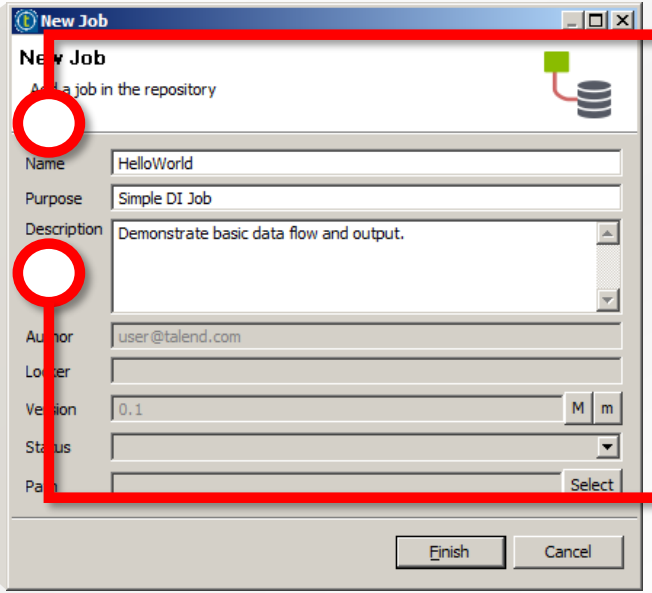
A **tLogRow** component is used to display messages in the Job execution console.

Build a Job

Create a Job

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





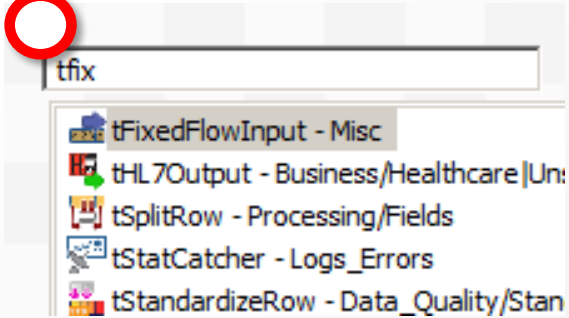
Name cannot contain spaces or special characters.

Fill out both **Purpose** and **Description**.

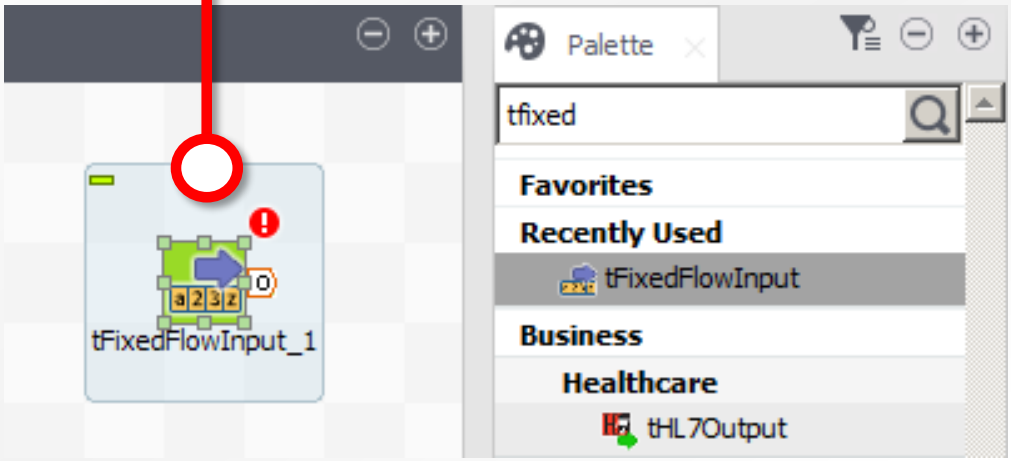
Build a Job

Options for adding components

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



Drag and drop components from **Palette**

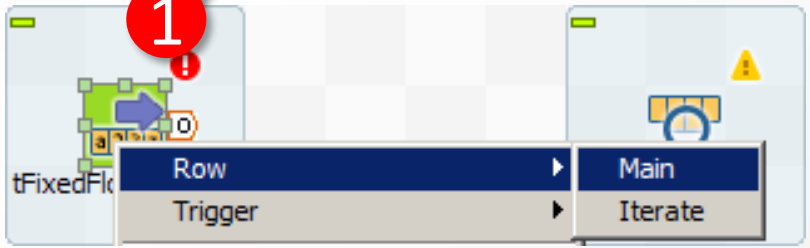


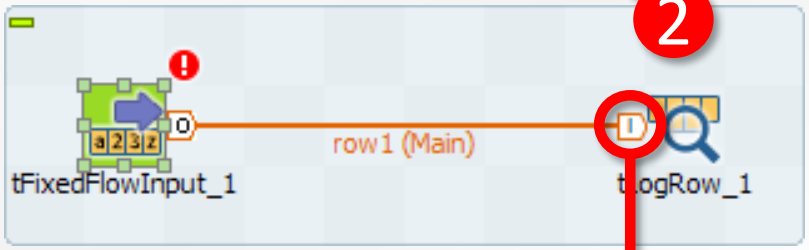
Build a Job

Connecting components

Right-click the component and select **Row > Main**

Click the second component to complete the link



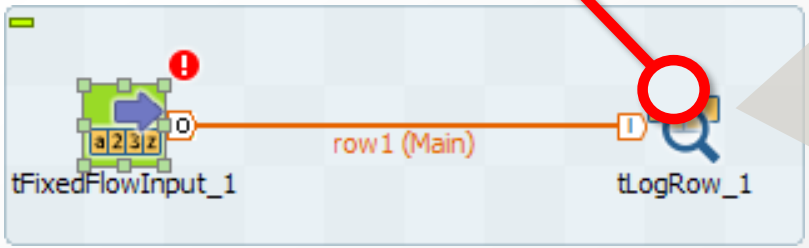


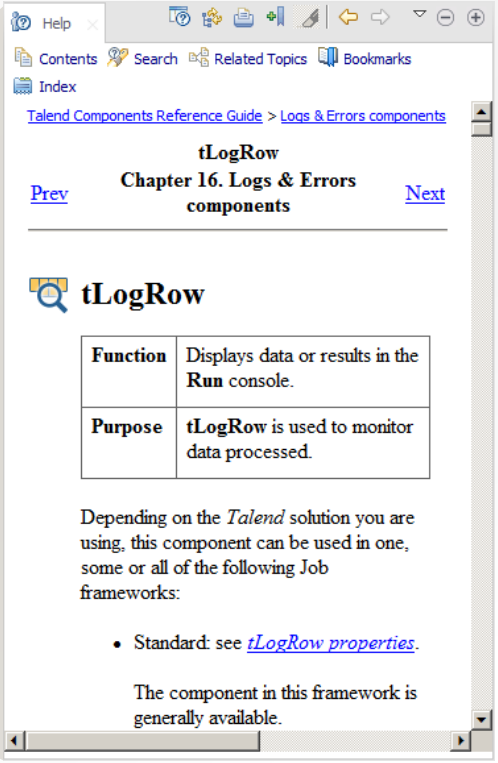
The arrow indicates the direction of the data flow in the link.

Build a Job

Online help

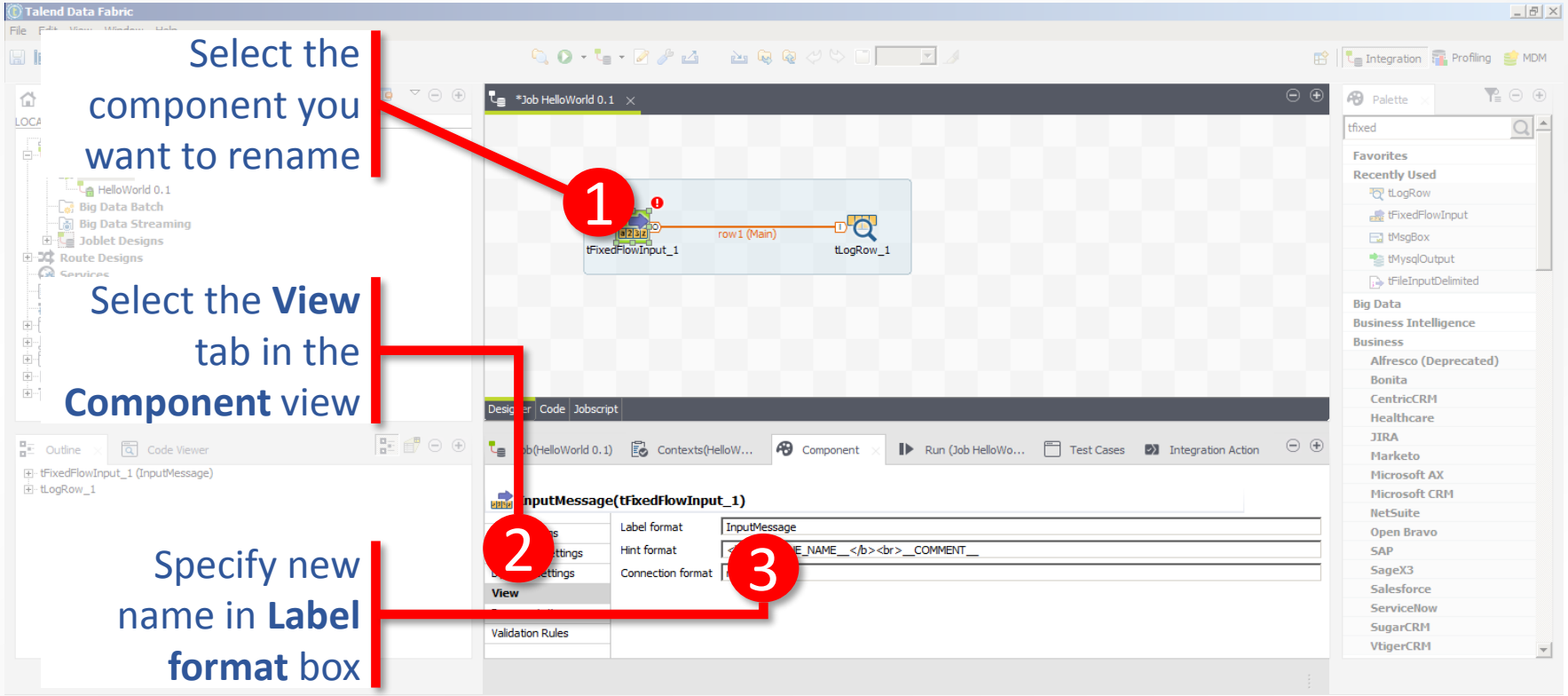
Select the component and press **F1**





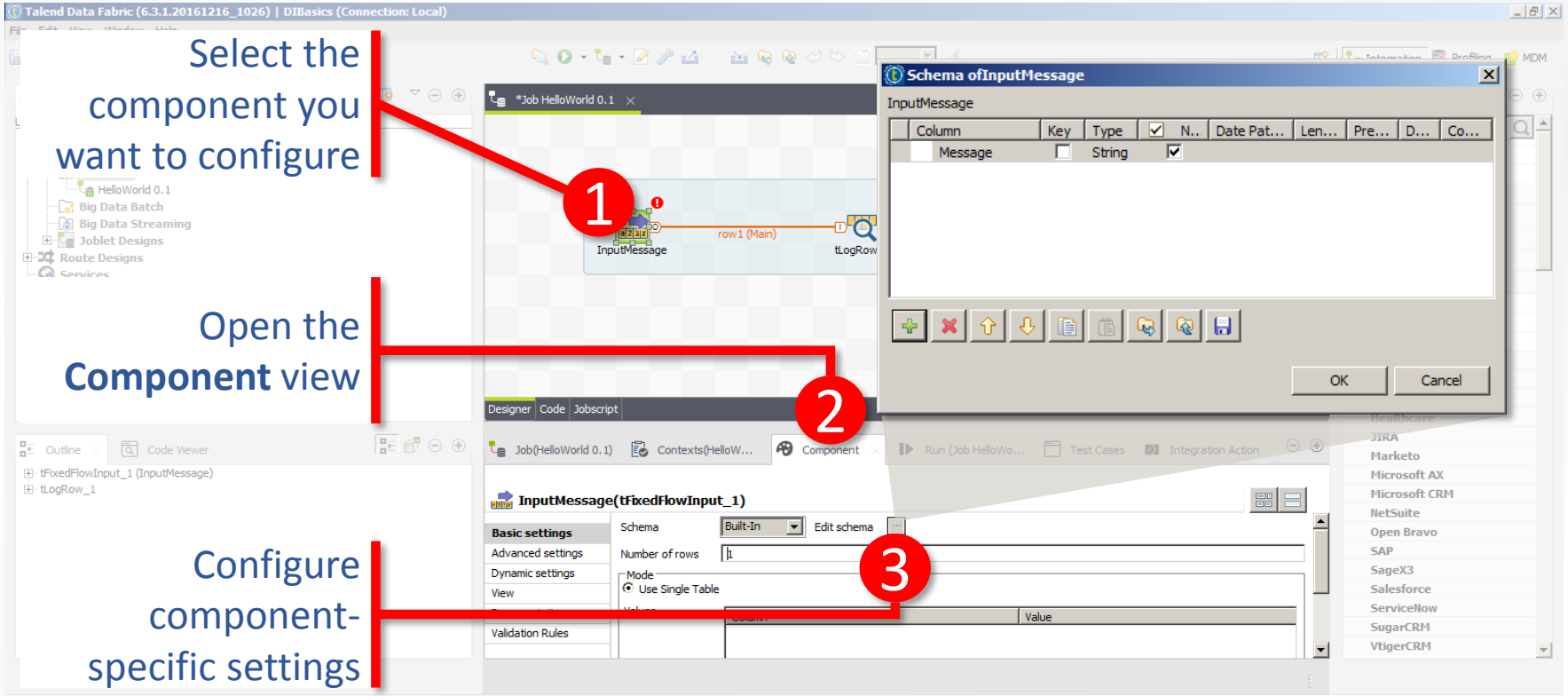
Build a Job

Rename a component



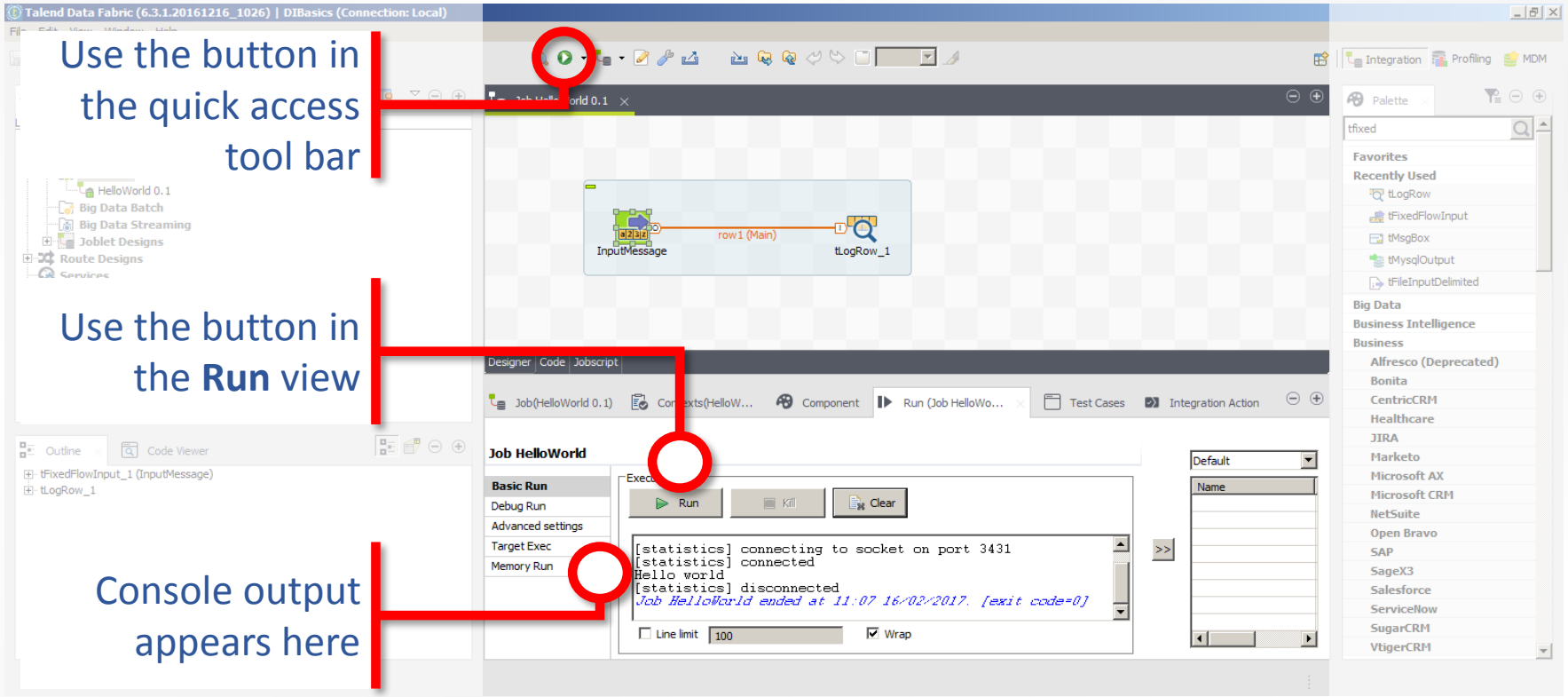
Build a Job

Configure a component



Run a Job

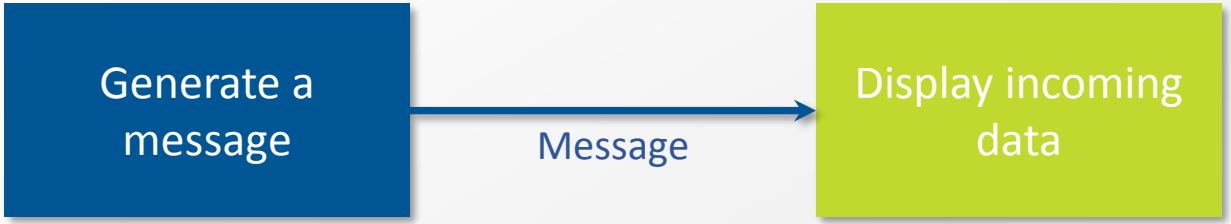
Options for running a Job



Lab Overview

Discover Talend Studio, build and run your first Job

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



Lesson Summary

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

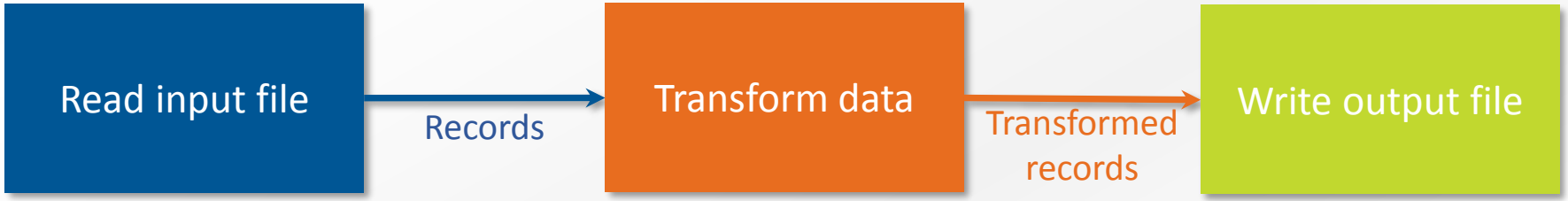
Working with Files

Objectives

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

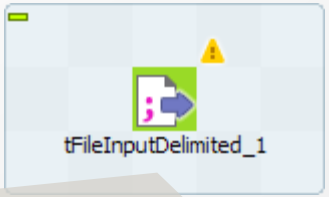
Use Case

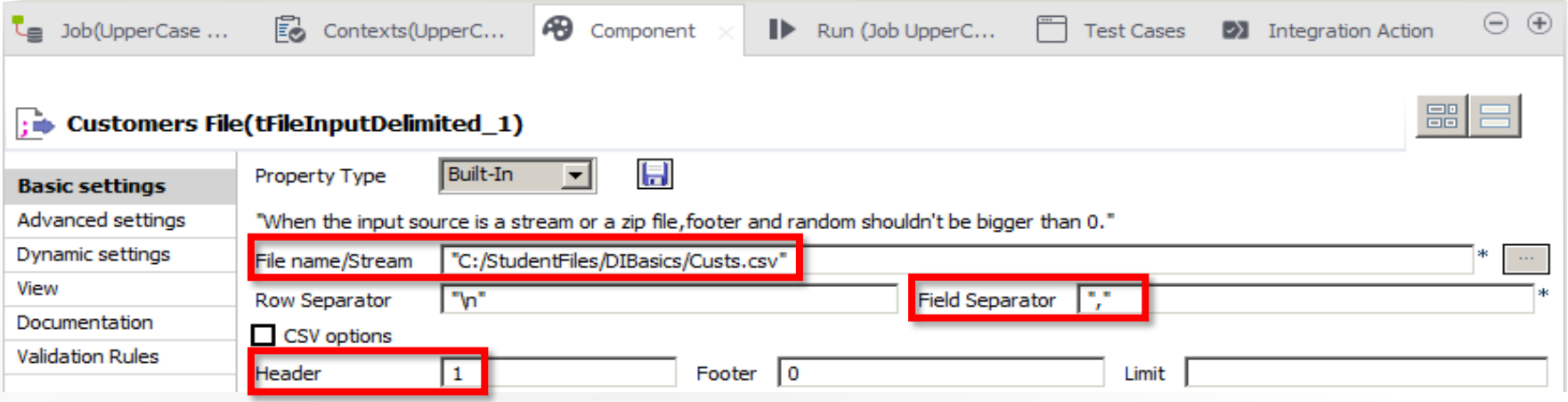
- Apply basic transformations to an input file
  - Transform a column to uppercase
  - Concatenate first name and last name into full name



Read a Delimited File

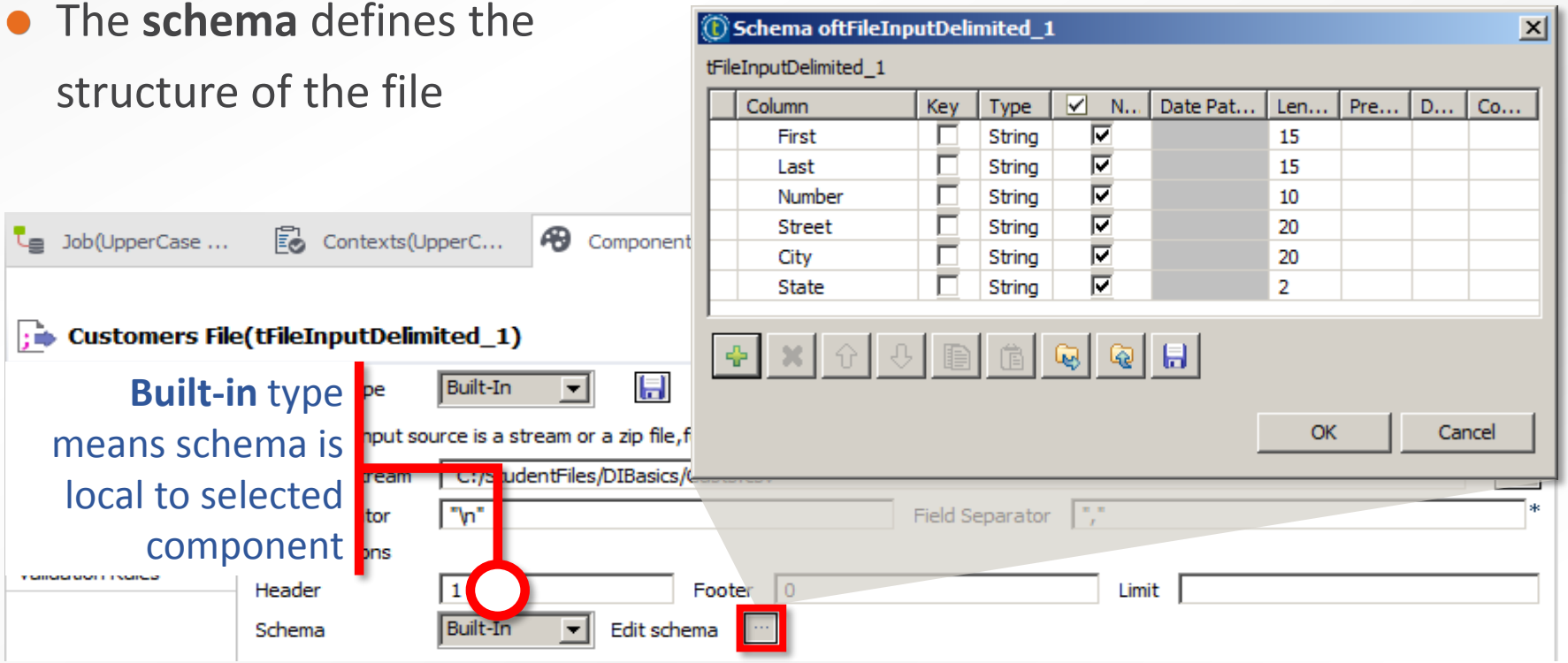
- Use a **tFileInputDelimited** component
  - Configure **file name**
  - Check **field separator**
  - Specify **number of header lines**





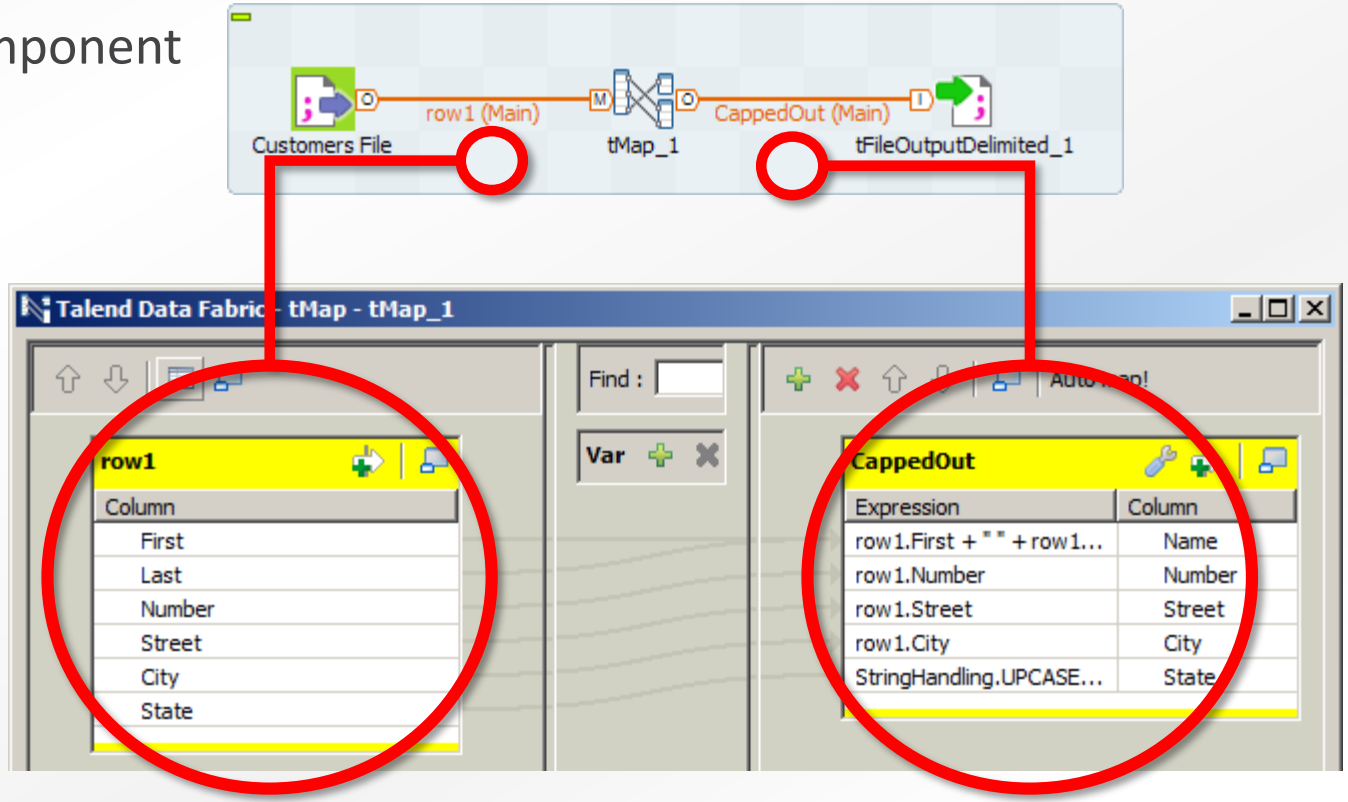
File Delimited Schema

- The **schema** defines the structure of the file



Transforming Data

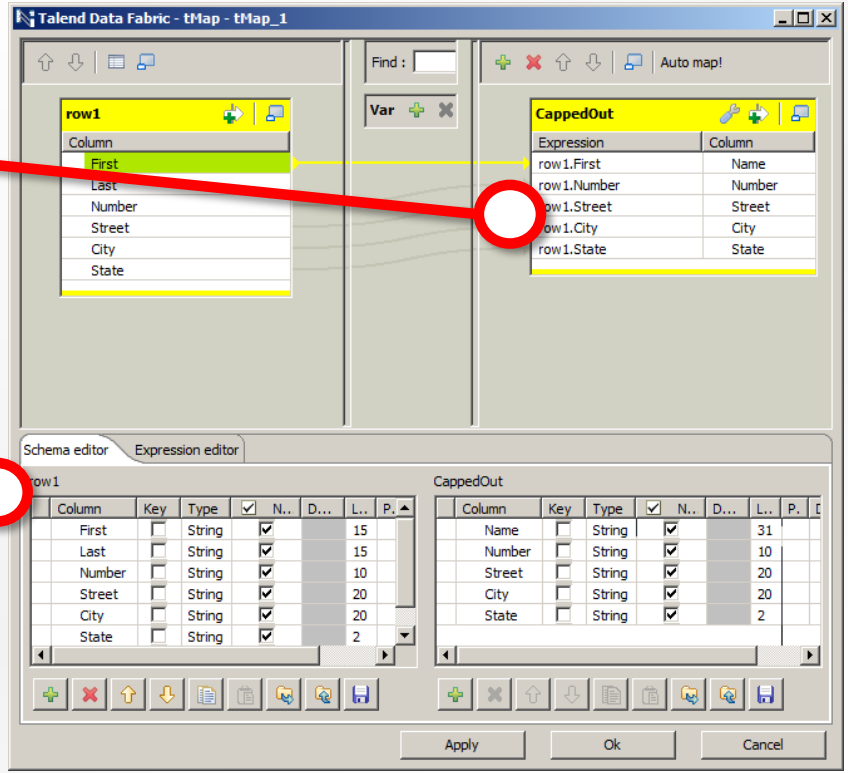
- Use a **tMap** component



Transforming Data

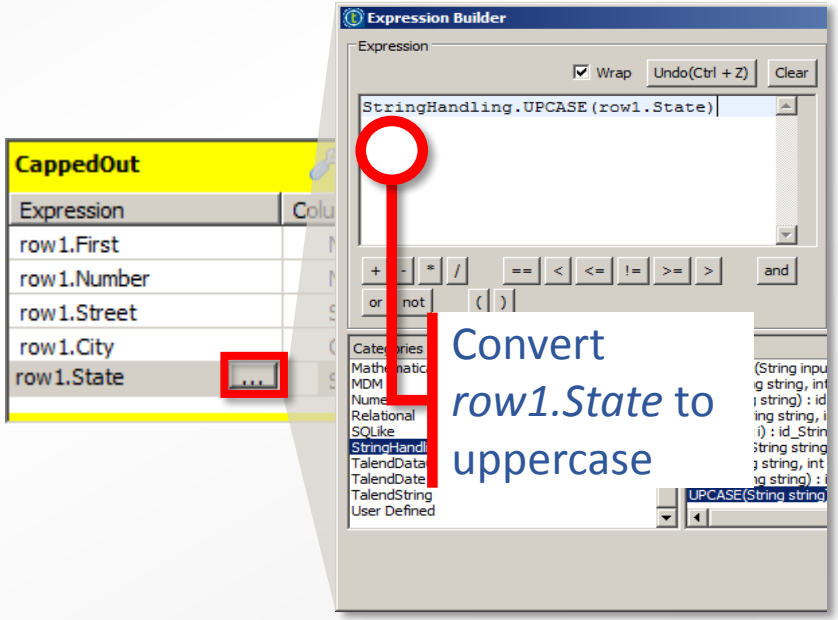
Mapping editor allows you to drag and drop input columns to the output tables

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

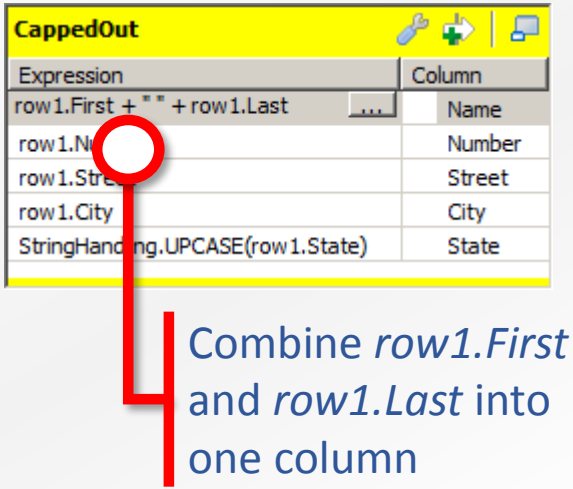


Transforming Data

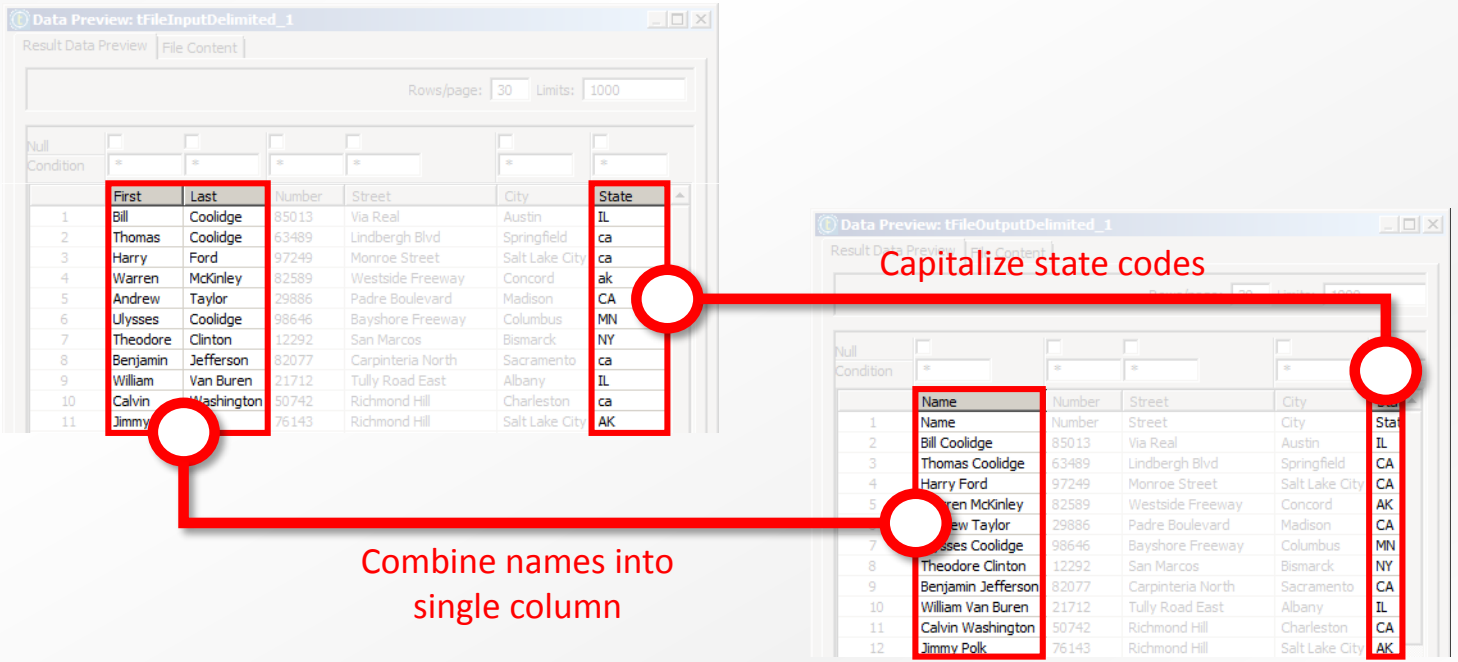
- Use predefined functions to build expressions



- Use operators with input columns to build expressions

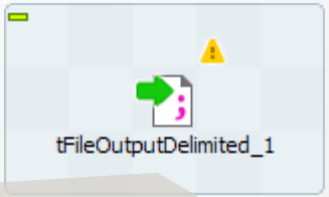


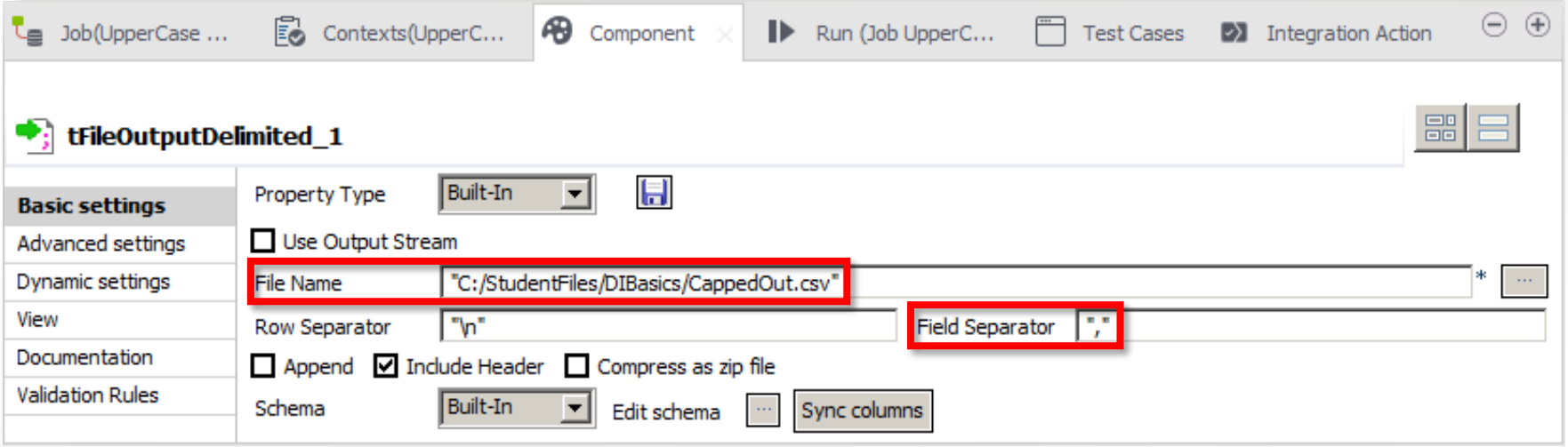
Transforming Data



Write to a Delimited File

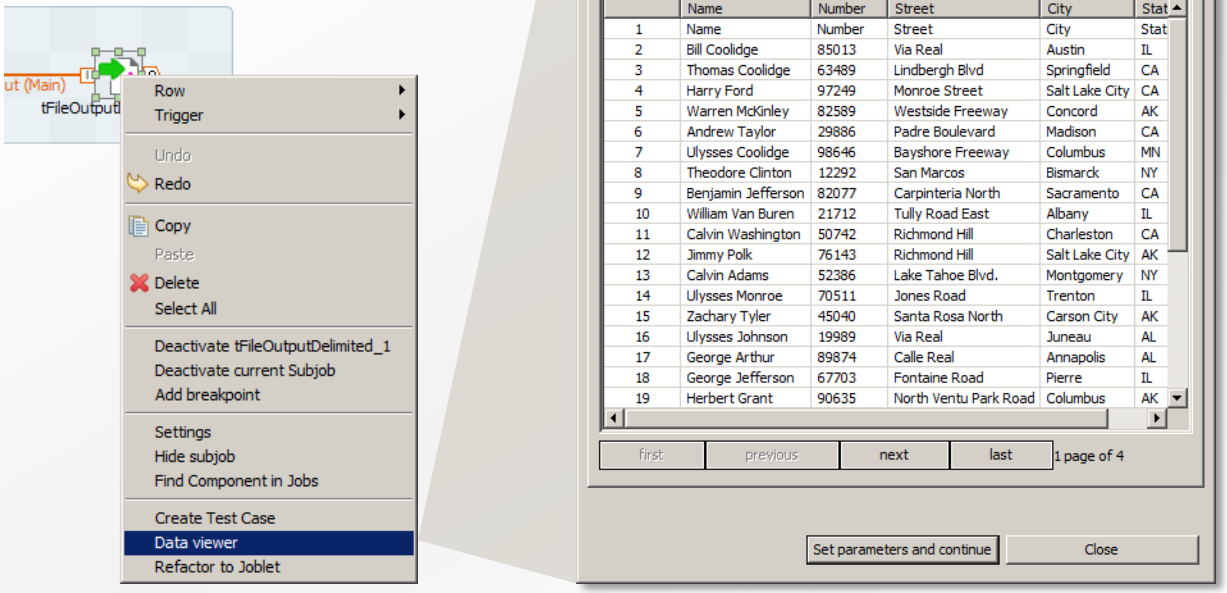
- Use a **tFileOutputDelimited** component
  - Configure **file name**
  - Check **field separator**





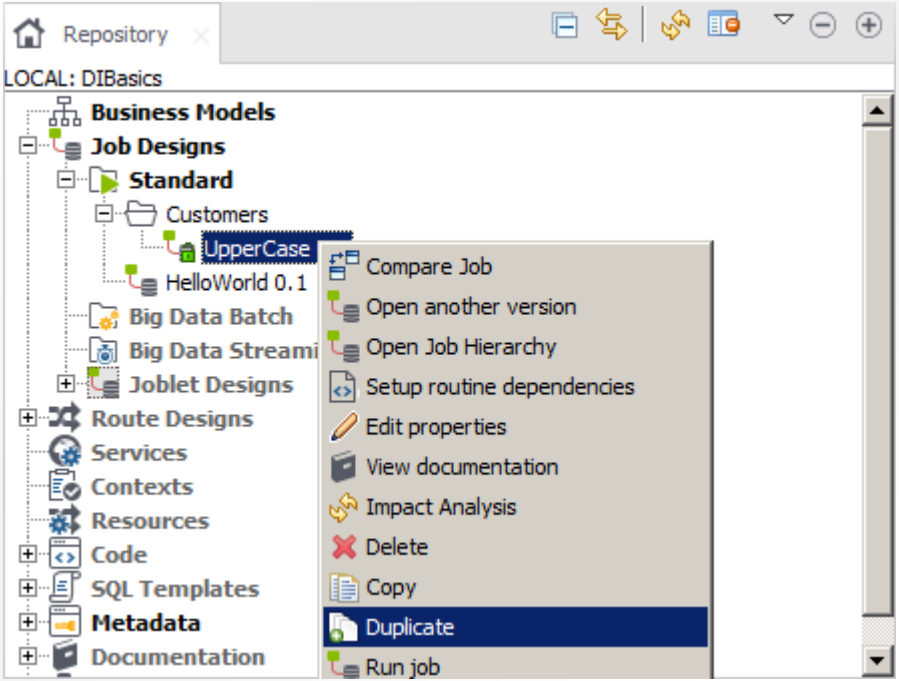
Data Viewer

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



Duplicate a Job

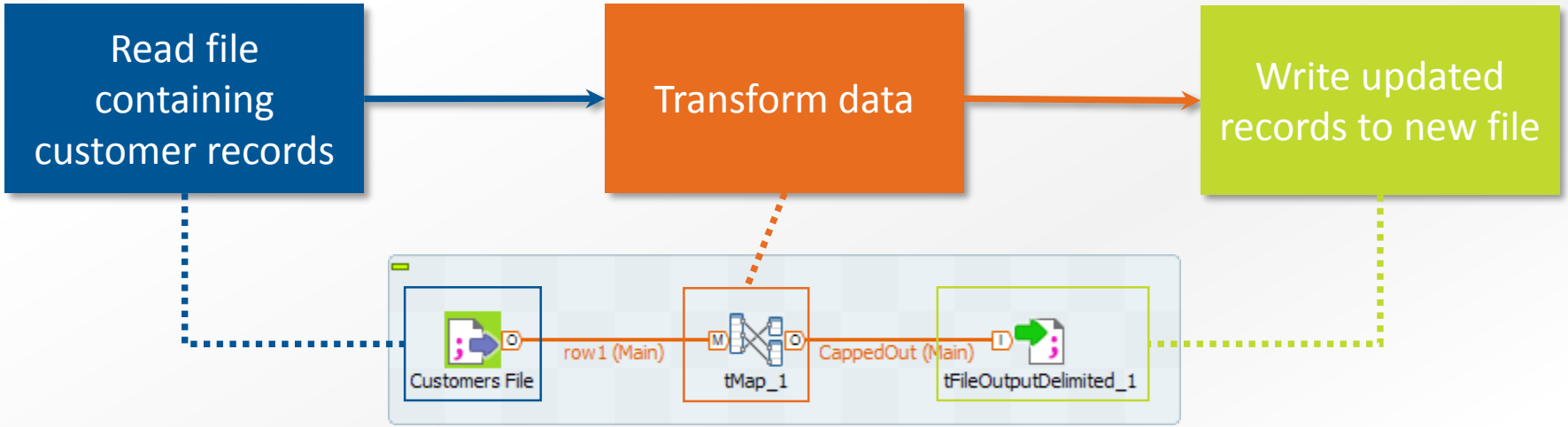
- Allows reuse of Jobs
- Right-click the Job and select **Duplicate**



Lab Overview

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

- Build and run the following Job:



Lesson Summary

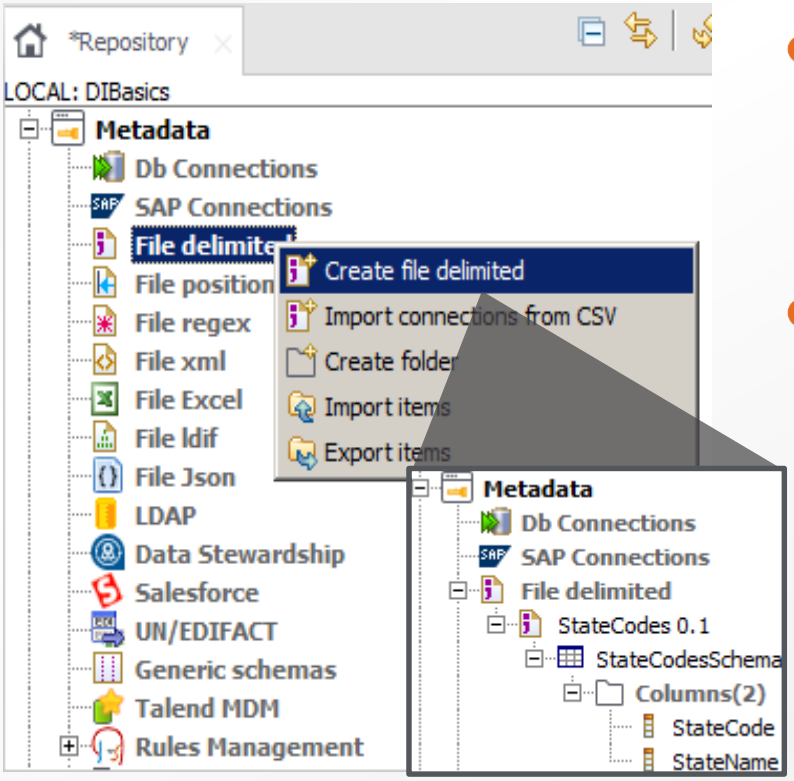
- **tFileInputDelimited**
  - Reads a delimited file
- **tFileOutputDelimited**
  - Writes a delimited file
- **tMap**
  - Transforms data using operators and predefined functions

Joining Data Sources

Objectives

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

Create Metadata

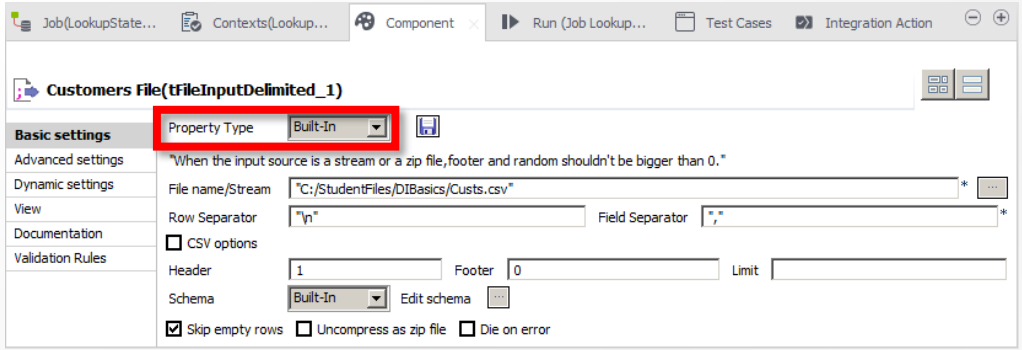


- **Metadata** stores configuration information that can be reused in multiple components
- **File Delimited** stores file information:
  - File Name
  - Separator and header information
  - Schema (fields, types, length)

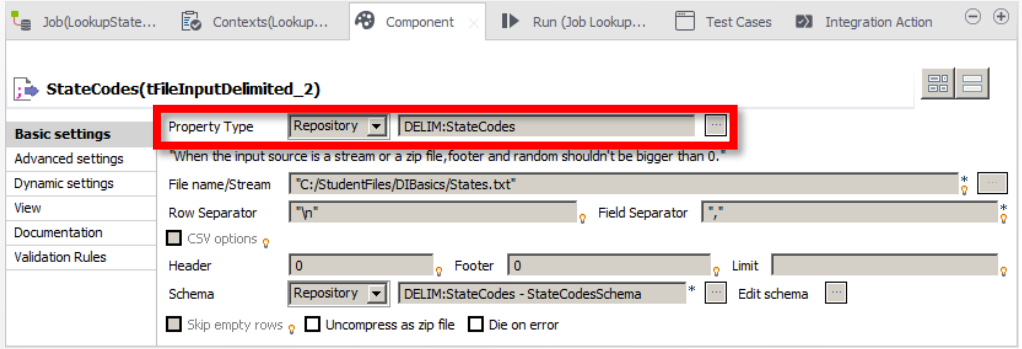
Use Metadata

Schema/Property Types

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



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



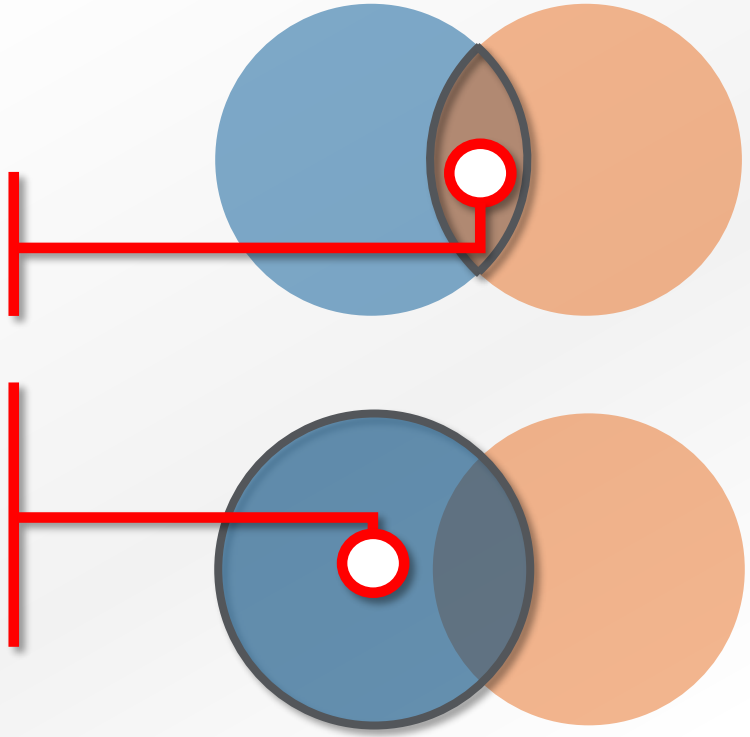
Joining Data Sources

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

Joining Data Sources

Join Types

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

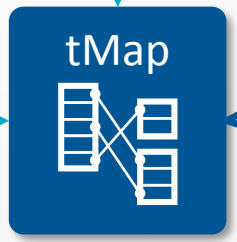


Joining Data Sources

Comparing Join Types

ID	Name	Country	Email
1	Shong	China	shong@talend.com
4	Elisa	France	elisa@talend.com

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

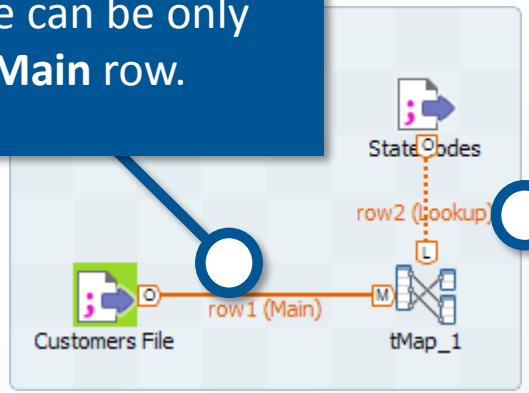
Left Outer Join

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

Join Data Using tMap

- **tMap** accepts multiple input sources

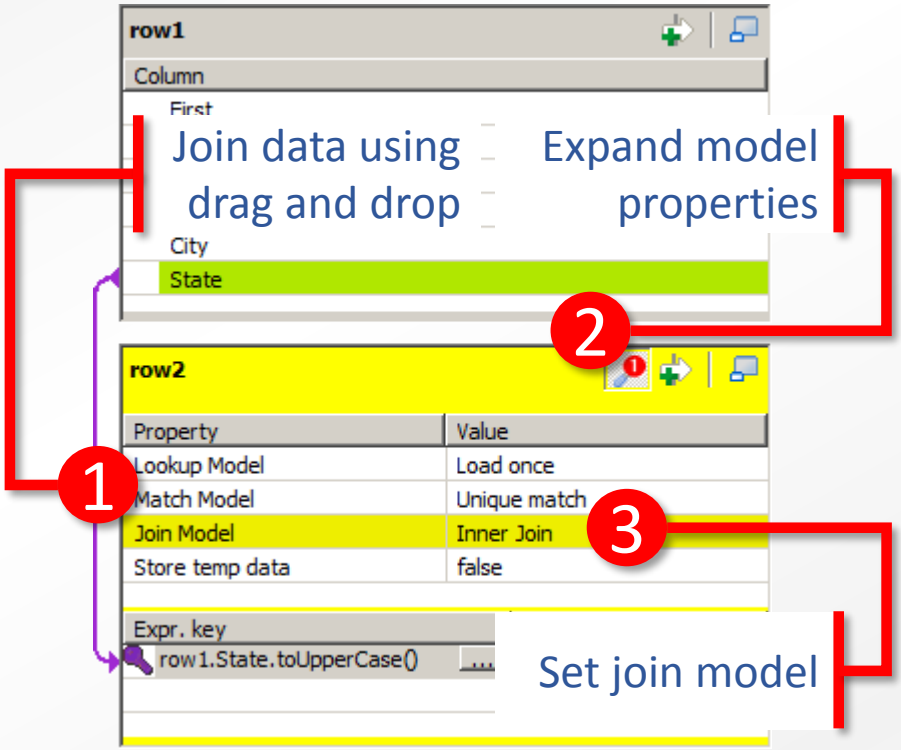
There can be only one **Main** row.



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

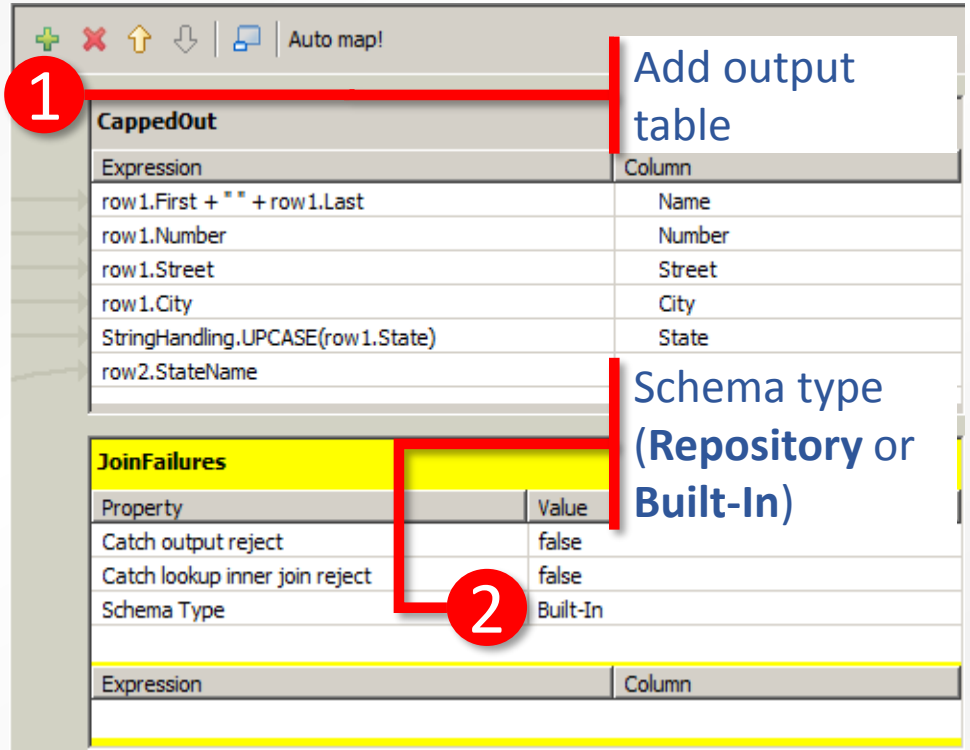
1. Define Multiple Input Sources
2. Configure Join
3. Define Multiple Output Sources
4. Handle Rejects

Join Data Using tMap



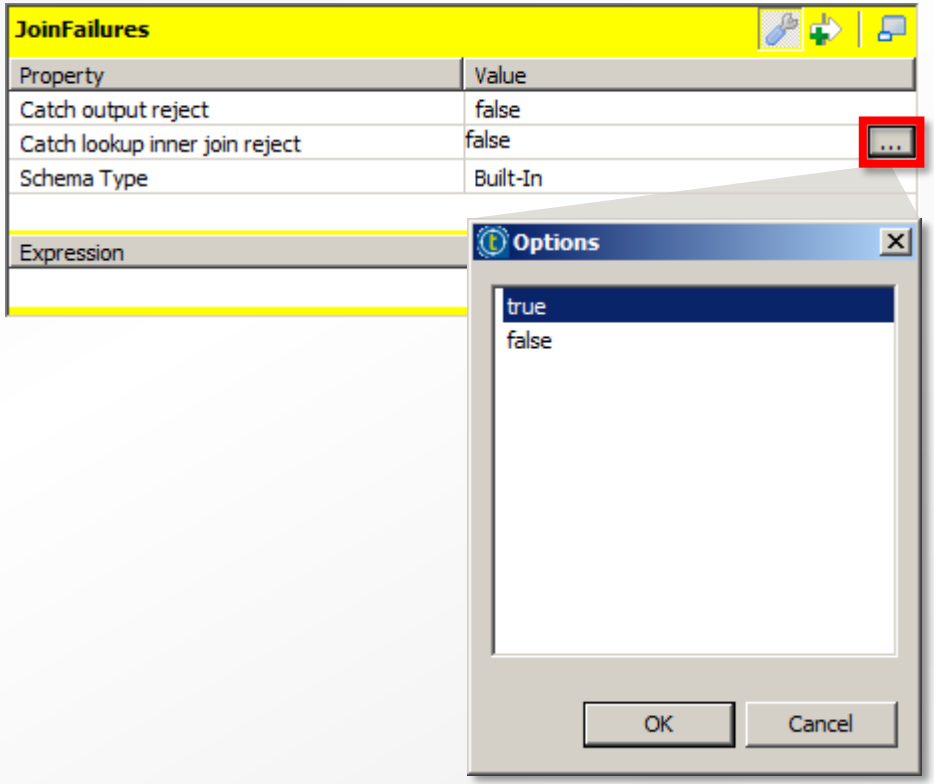
1. Define Multiple Input Sources
2. Configure Join
3. Define Multiple Output Sources
4. Handle Rejects

Join Data Using tMap



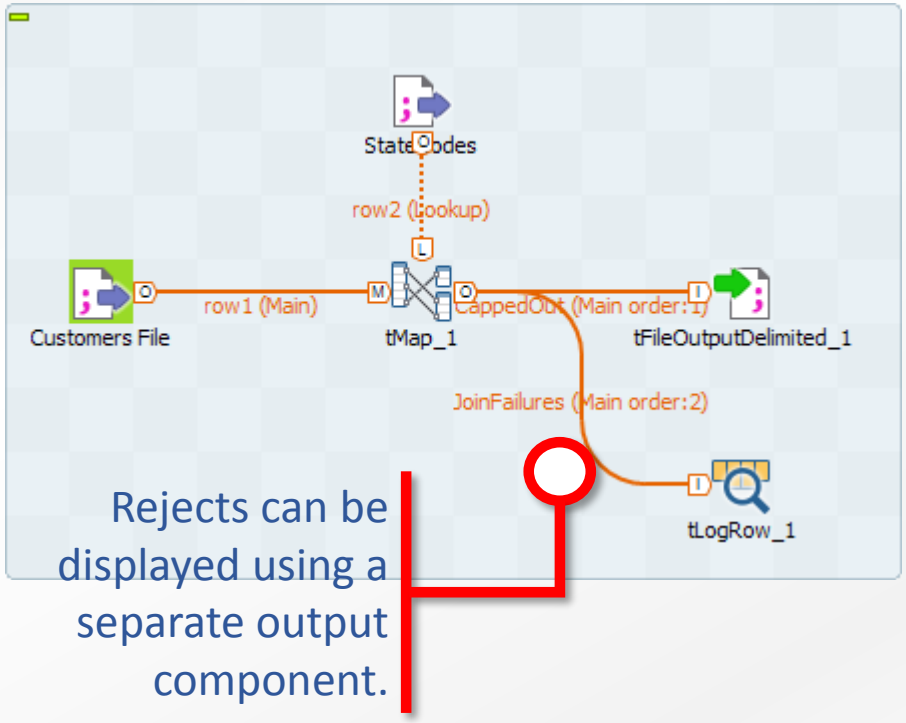
1. Define Multiple Input Sources
2. Configure Join
3. Define Multiple Output Sources
4. Handle Rejects

Join Data Using tMap



1. Define Multiple Input Sources
2. Configure Join
3. Define Multiple Output Sources
4. Handle Rejects

Join Data Using tMap

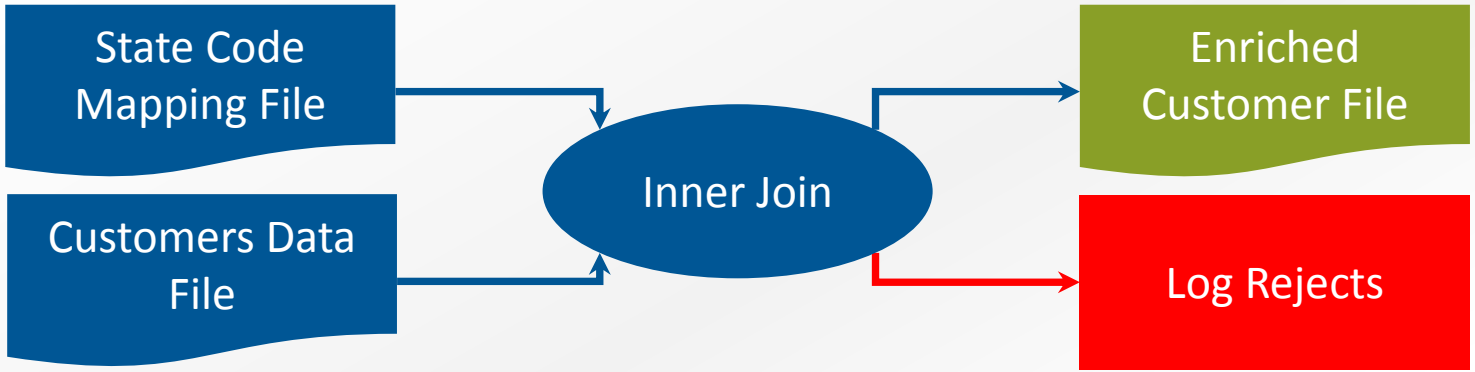


1. Define Multiple Input Sources
2. Configure Join
3. Define Multiple Output Sources
4. Handle Rejects

Lab Overview

Joining multiple data sources

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



Lesson Summary

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

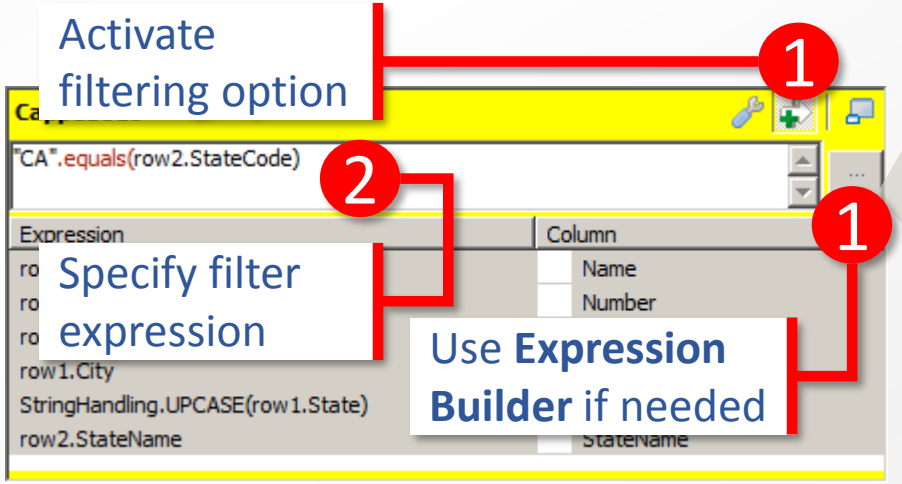
Filtering Data

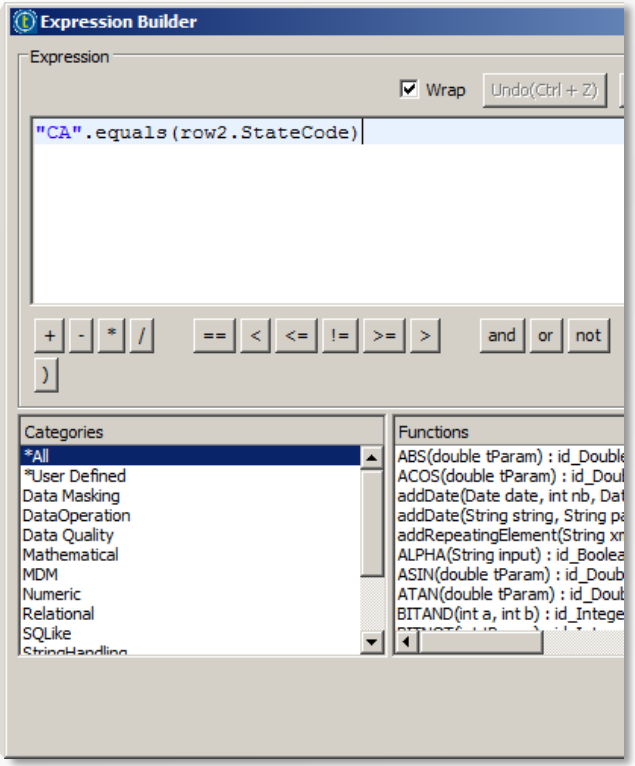
Objectives

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

Filtering Data

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





Filtering Data

Example Filtering and Transforming Column Data

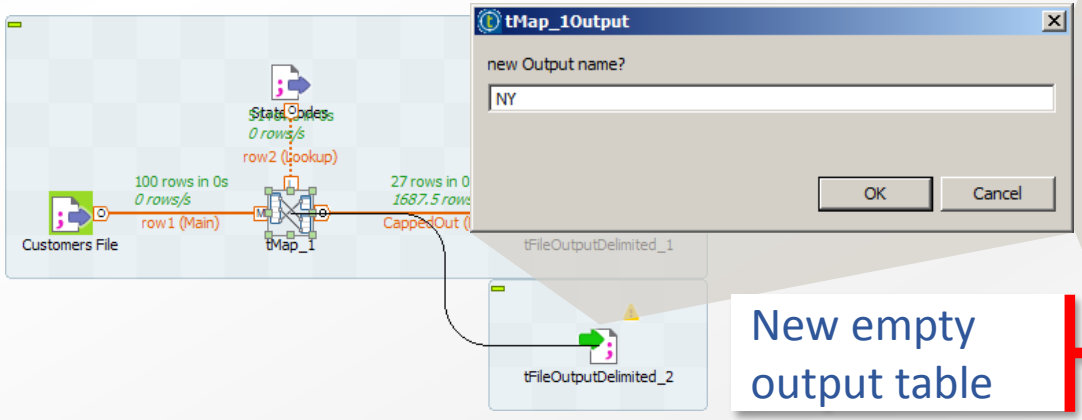
	First	Last	Number	Street	City	State
1	Bill	Coolidge	85013	Via Real	Austin	IL
2	Thomas	Coolidge	63489	Lindbergh Blvd	Springfield	ca
3	Harry	Ford	97249	Monroe Street	Salt Lake City	ca
4	Warren	McKinley	82589	Westside Freeway	Concord	ak
5	Andrew	Taylor	29886	Padre Boulevard	Madison	CA
6	Ulysses	Coolidge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

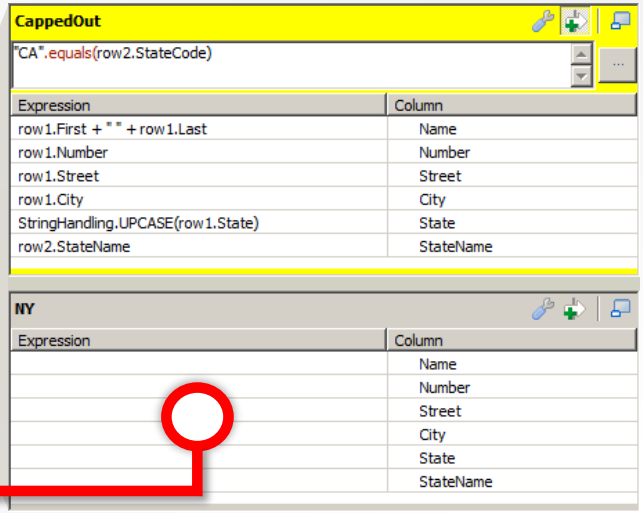
- Filtered (California records only)
- Transformed (capitalize state codes

		Number	Street	City	State
	Number	Street	City	State	
	s Coolidge	63489	Lindbergh Blvd	Springfield	CA
	Ford	97249	Monroe Street	Salt Lake City	CA
4	Andrew Taylor	29886	Padre Boulevard	Madison	CA
5	Benjamin Jefferson	82077	Carpinteria North	Sacramento	CA
6	Calvin Washington	50742	Richmond Hill	Charleston	CA
7	Benjamin Ford	27921	Carpinteria Avenue	Providence	CA
8	Calvin Pierce	41962	Santa Rosa North	Juneau	CA
9	Woodrow Clinton	30587	San Diego Freeway	Topeka	CA
10	George Adams	40011	South Highway	Concord	CA
11	Dwight Pierce	24382	North Atherton Street	Providence	CA
12	Jimmy Coolidge	19658	North Ventu Park Road	Sacramento	CA

Duplicate Output Flows

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

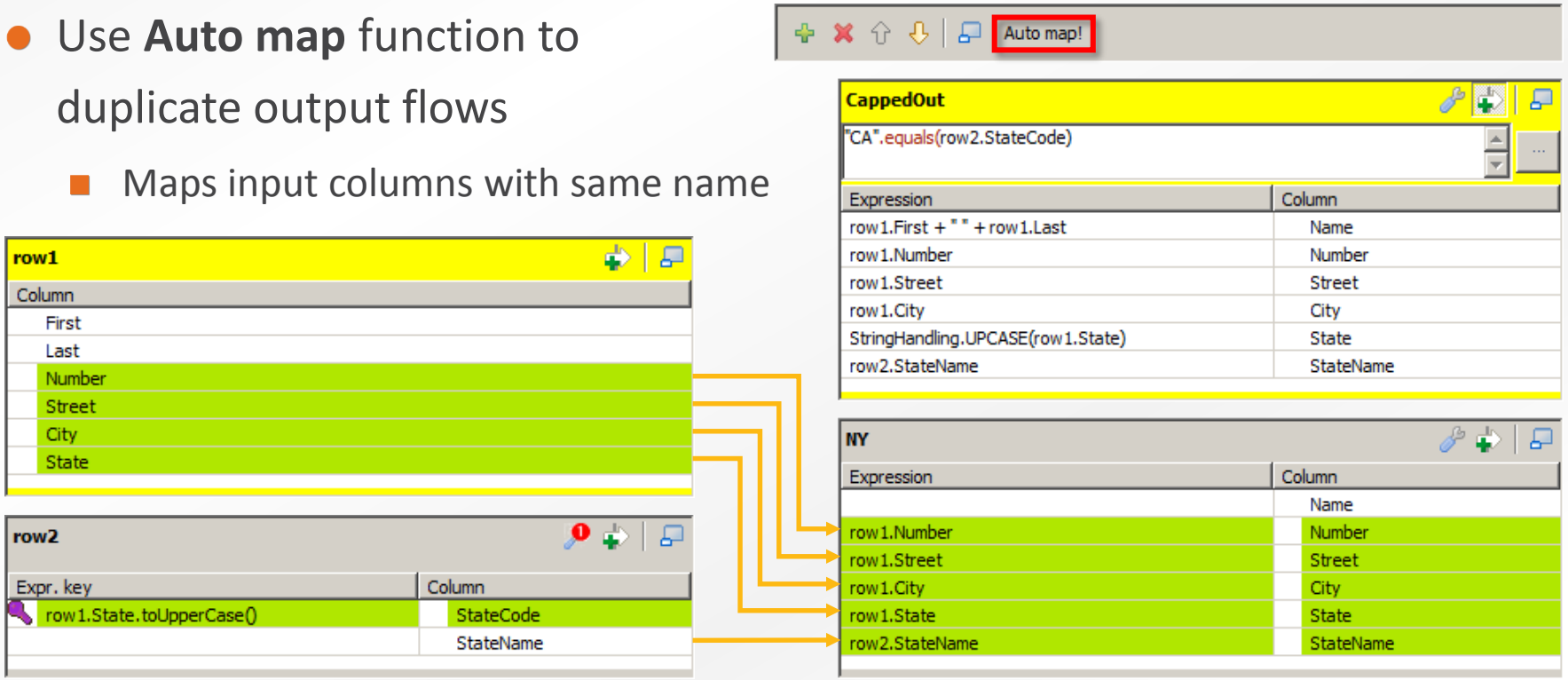




New empty output table

Duplicate Output Flows

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

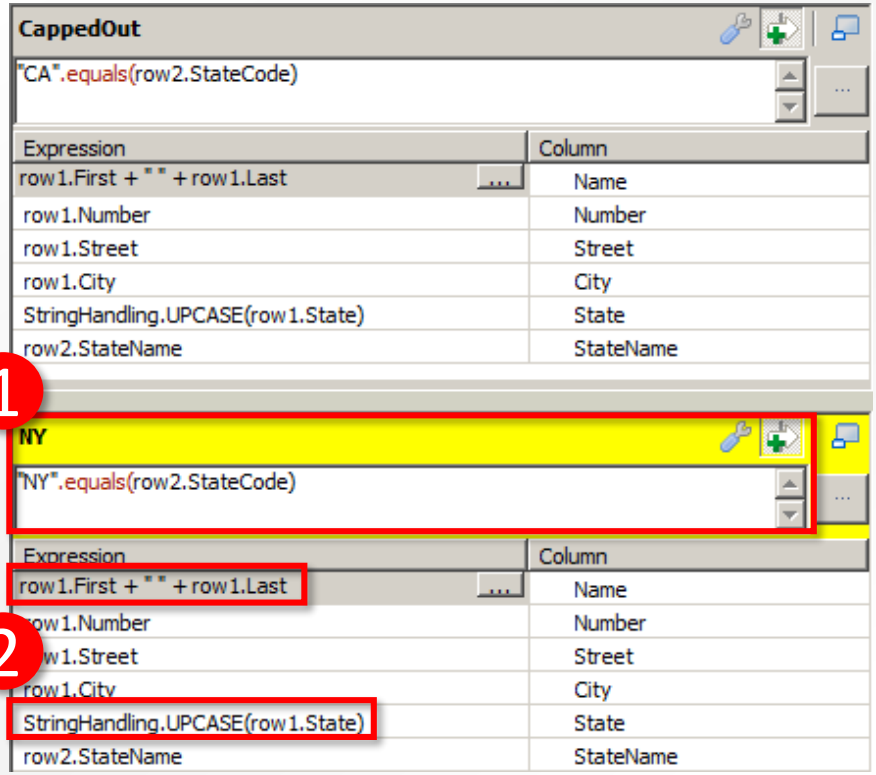


Duplicate Output Flows

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

Add filter expression

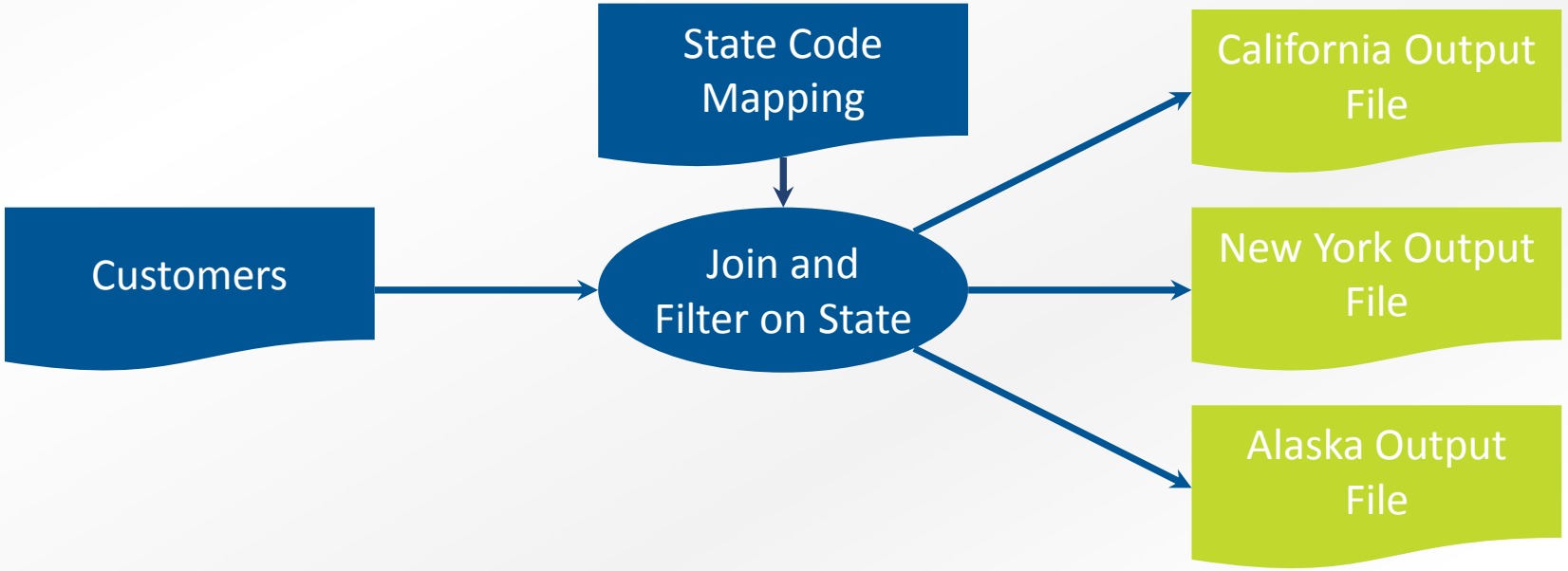
Apply transformation rules



Lab Overview

Redirecting data to different output flows based on input criteria

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



Lesson Summary

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

Using Context Variables

Objectives

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

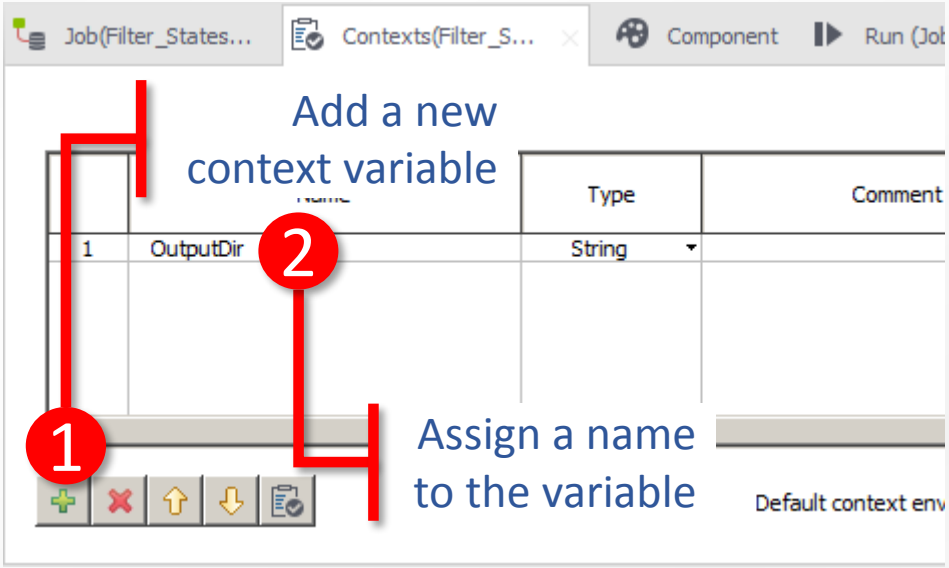
What is a Context Variable?

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

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

Define Context Variables

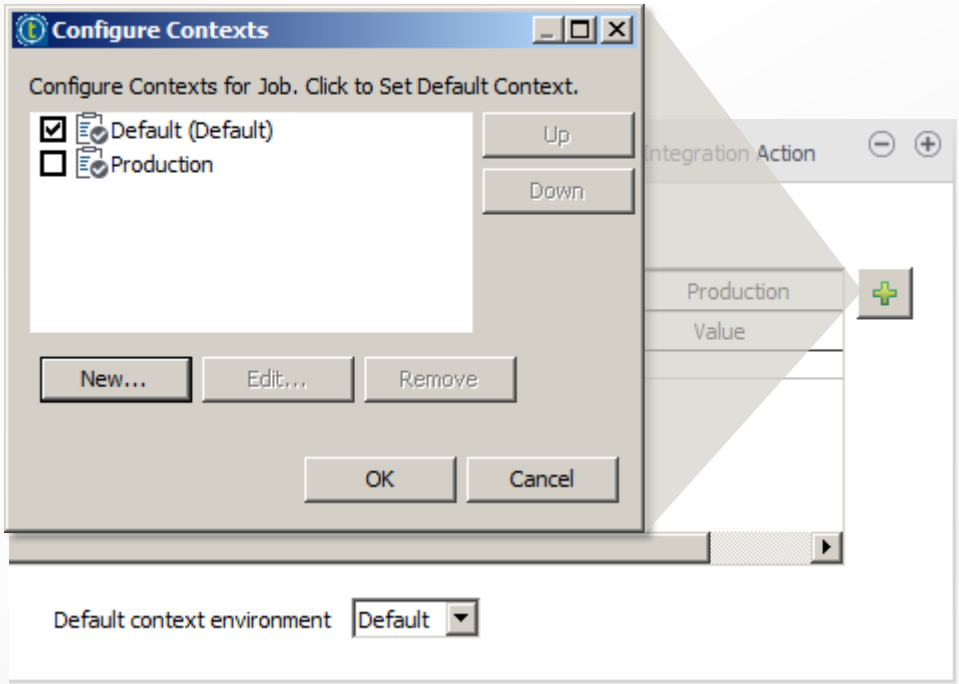
- Manage context variables in the **Contexts** view



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

Define Context Variables

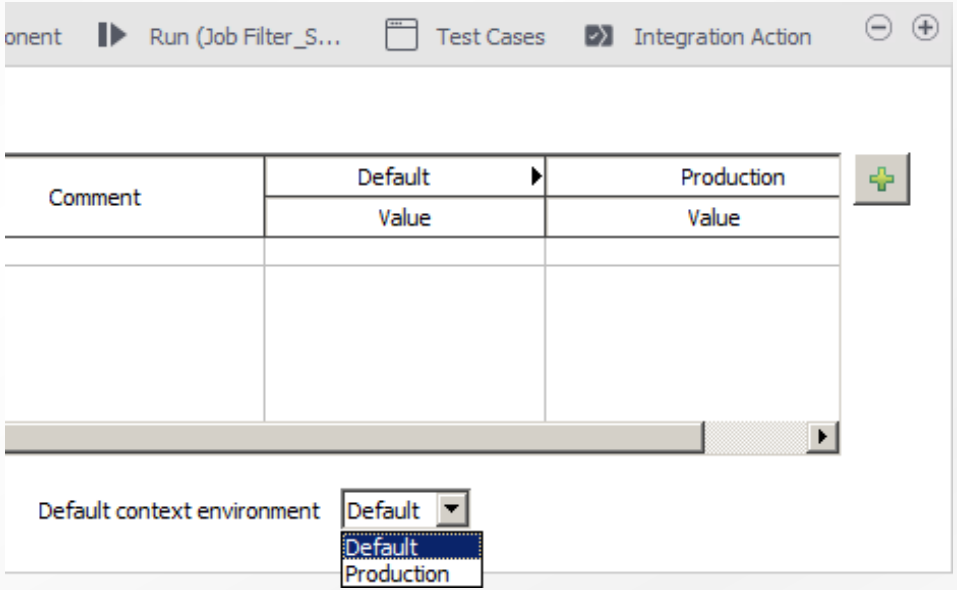
- Manage contexts in the **Contexts** view



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

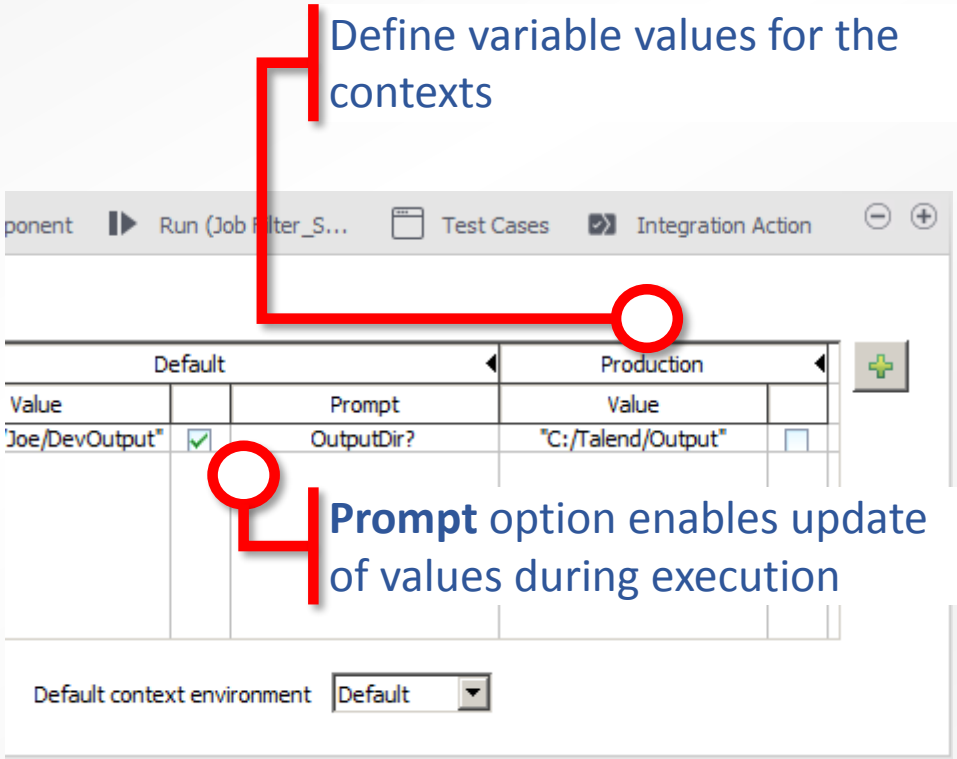
Define Context Variables

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



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

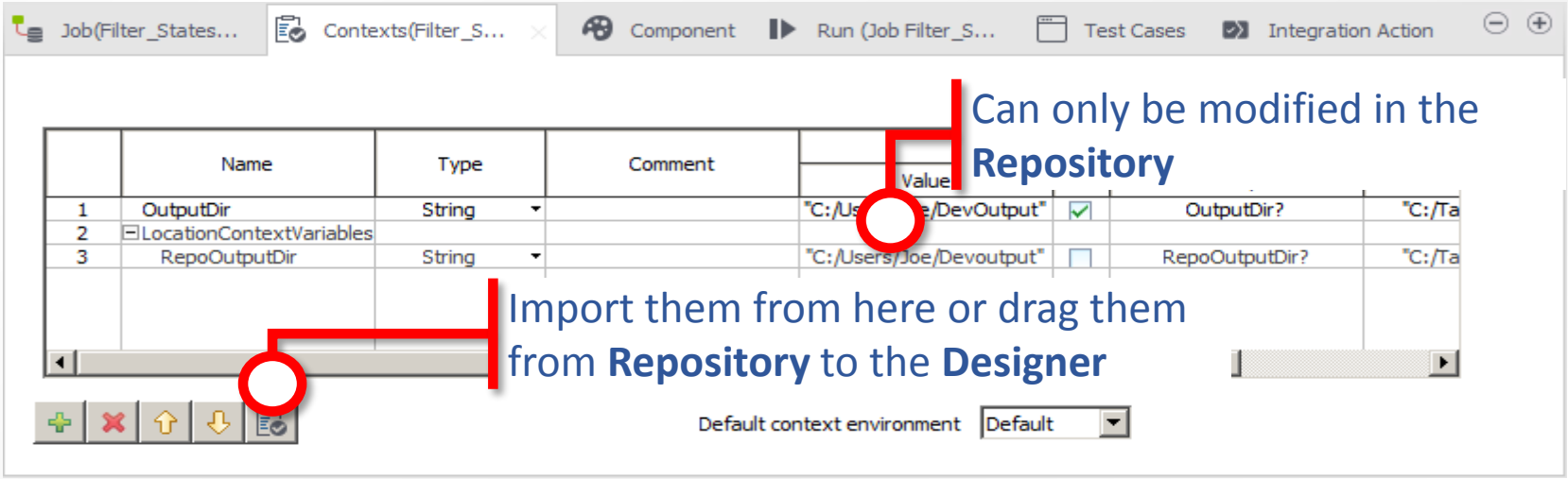
Define Context Variables



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

Repository Context Variables

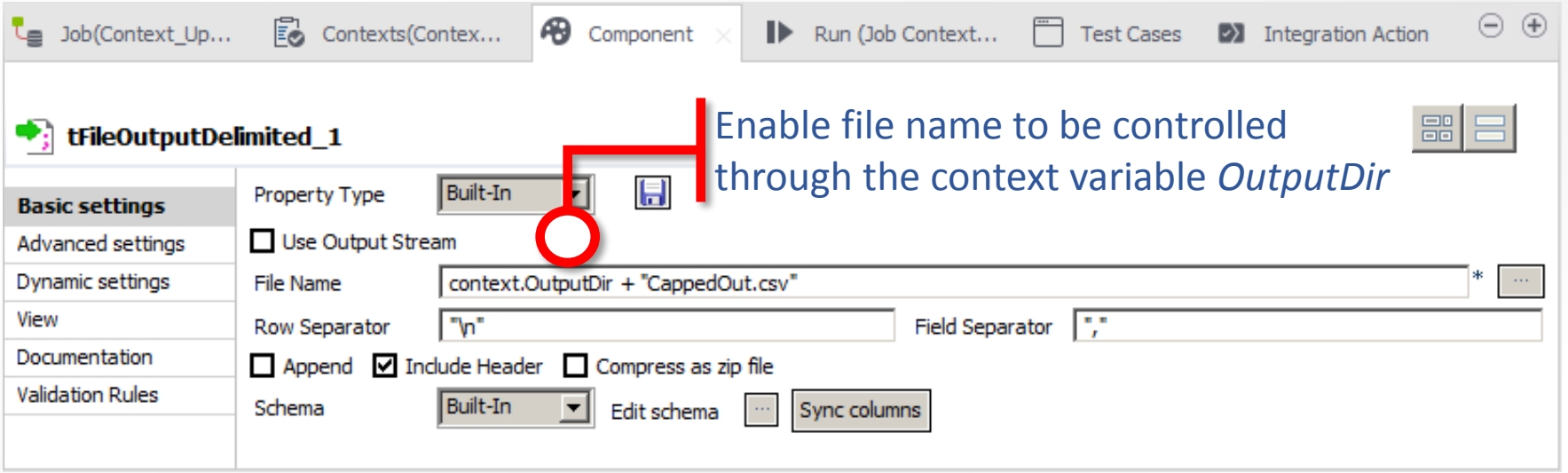
- Defined in **Repository -> Contexts**
  - Repository version of the context variables defined in the Job's **Contexts** view
  - Can be used across multiple Jobs



Using Context Variables

Access the Context Variables from the Component of a Job

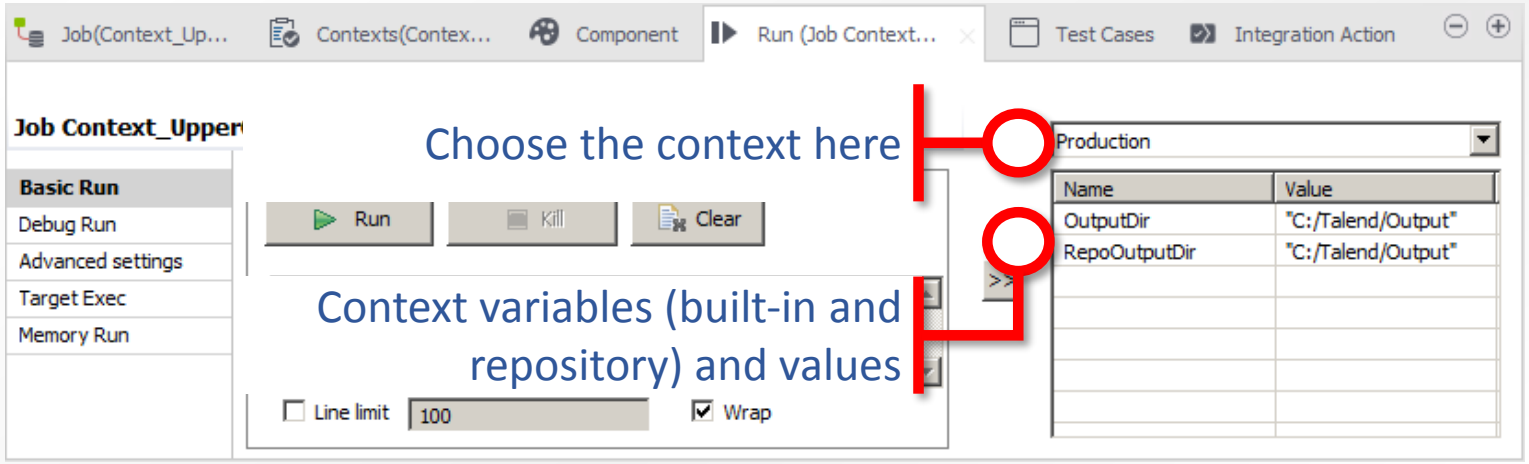
- All Context variables (built-in or repository) are accessed the same way
  - Example below illustrates an expression to access the *OutputDir* variable



Using Context Variables

Set the Context when Running a Job

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



Lab Overview

Create Variables to Provide Different Values in Two Different Contexts

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

Lesson Summary

Context Variable Types

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

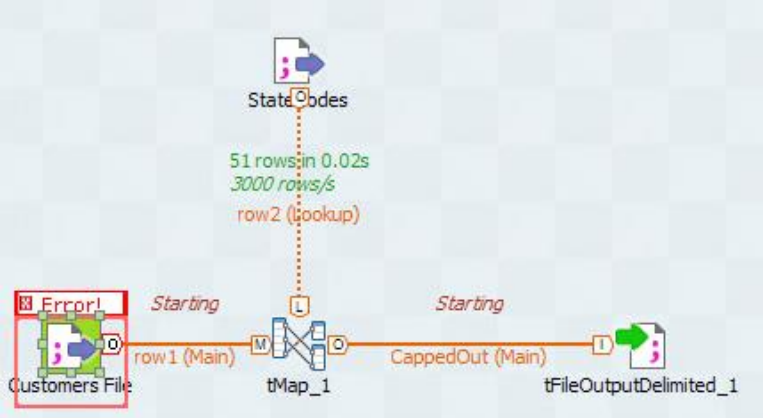
Error Handling

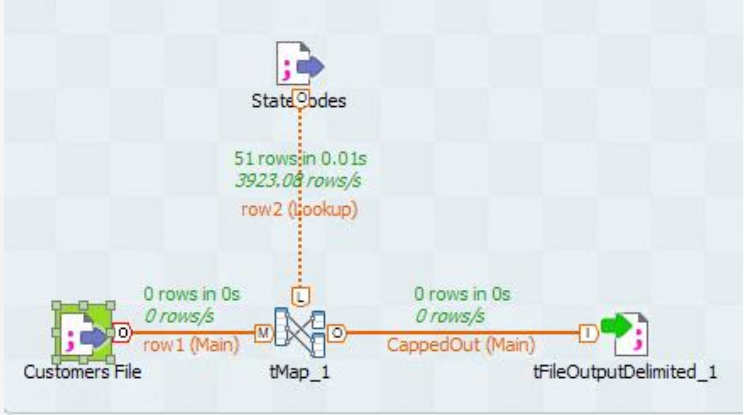
Objectives

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

Error Handling

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

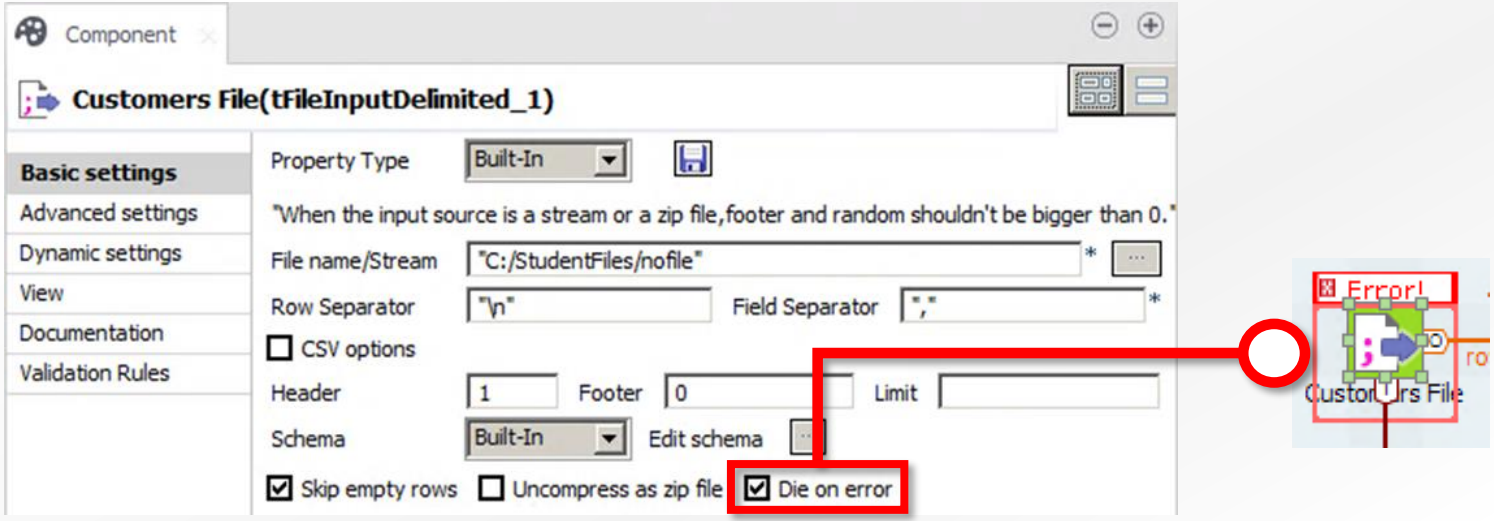




Error Handling

Die on Error option

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



Error Handling

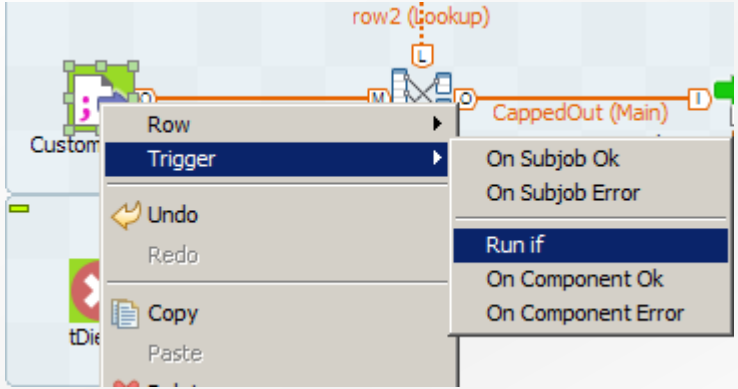
Triggers

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

Error Handling

Kill a Job Under Specific Conditions

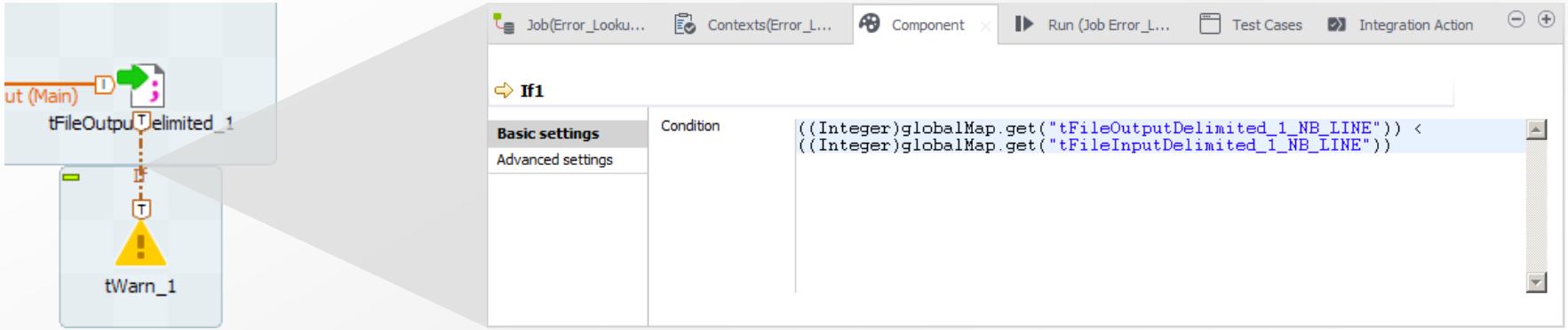
- Requirement:
  - Kill a Job under specific conditions (even if no technical error occurred)
  - For example: input file has less than a certain number of lines
- Solution:
  - Use **tDie** to kill the Job
  - Connect with **Run if** trigger
  - Configure the **condition**



Component view for the link enables you to configure the trigger expression

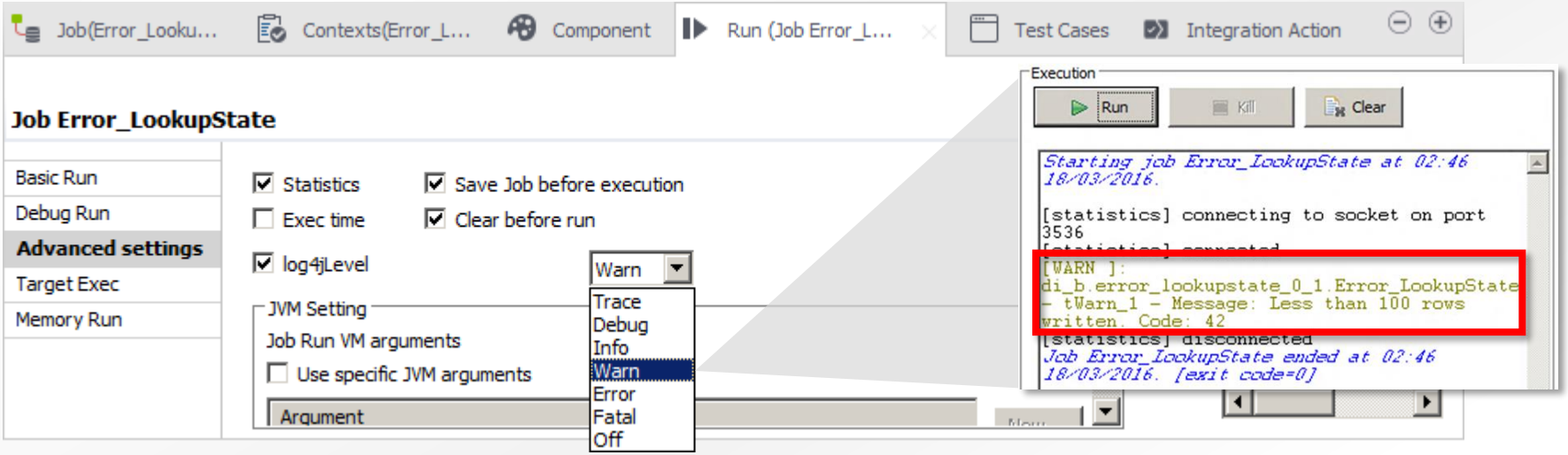
Raise a Warning

- Requirement:
  - Raise a warning when a business rule is not met
  - Example: the output file has fewer lines than the input
- Solution:
  - Use **tWarn** to raise the warning
  - Connect with **Run if** trigger
  - Configure the **condition**



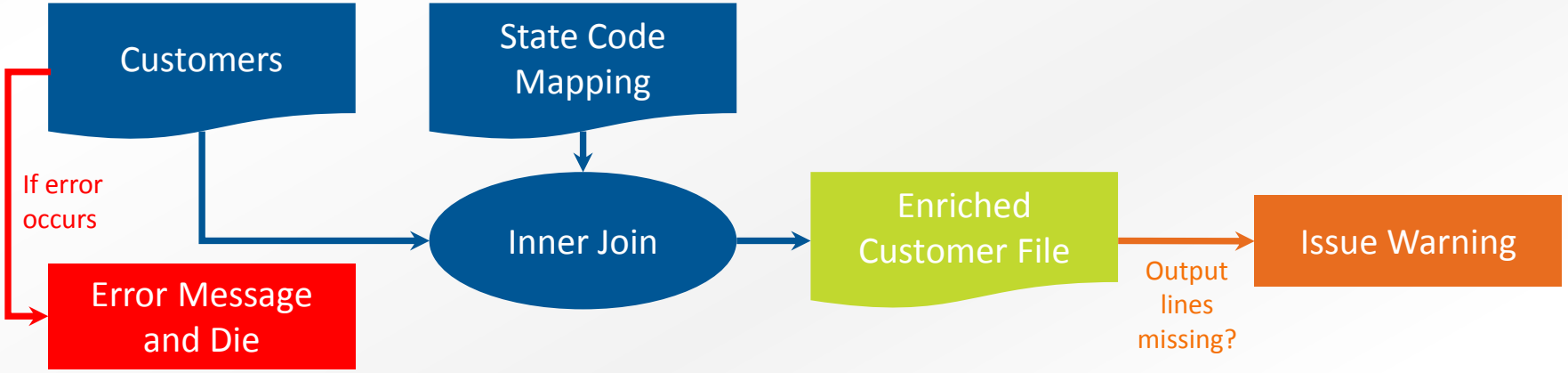
Logging Level

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



Lab Overview

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



Lesson Summary

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

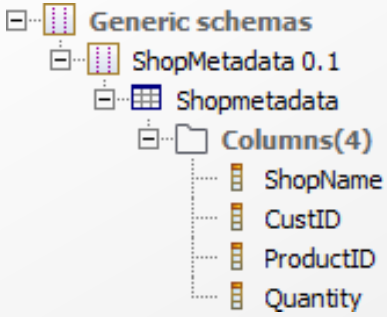
Generic Schemas

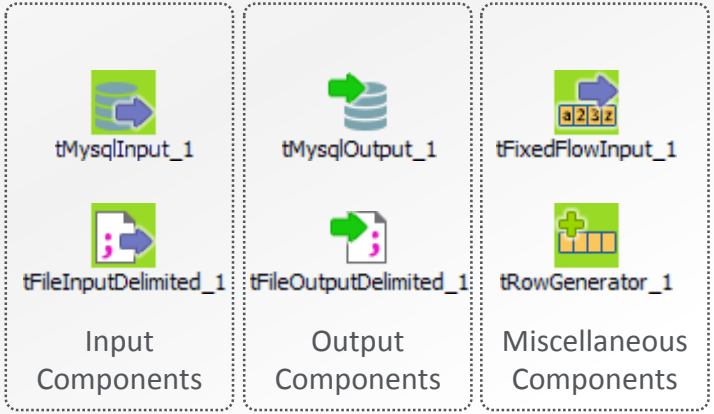
Objectives

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

Generic Schemas

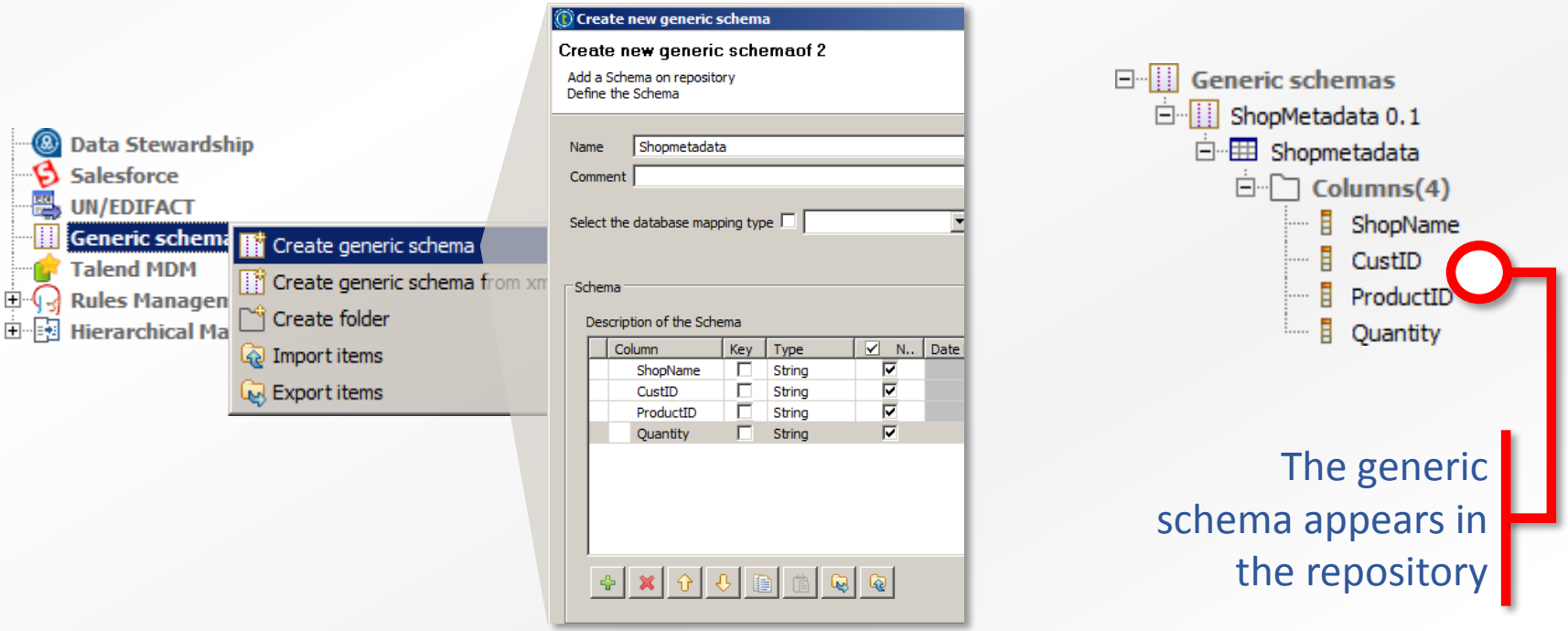
- The schema metadata covered previously was bound to a specific file type
  - Delimited, XML, JSON, and more
  - Cannot be applied universally
- A **generic schema** is not linked to a resource type and can be used by any component
  - Some examples shown on right





Create New Generic Schema

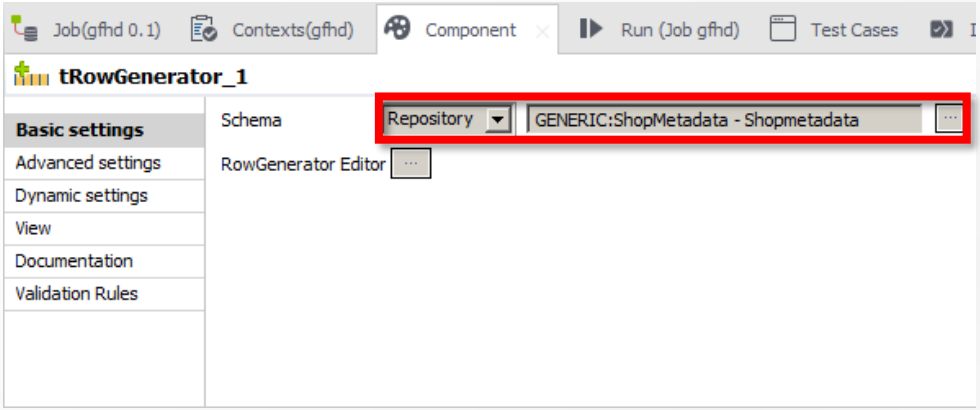
- Generic schemas can be found under **Repository > Metadata**



Generating Random Data

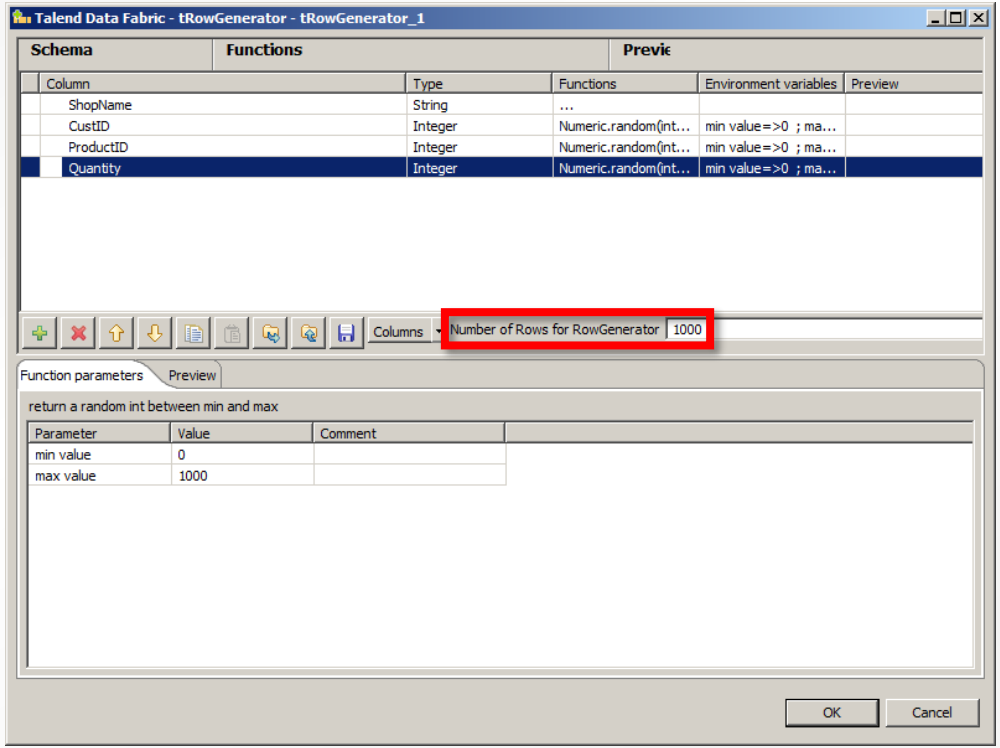
Use the **tRowGenerator** Component

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



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

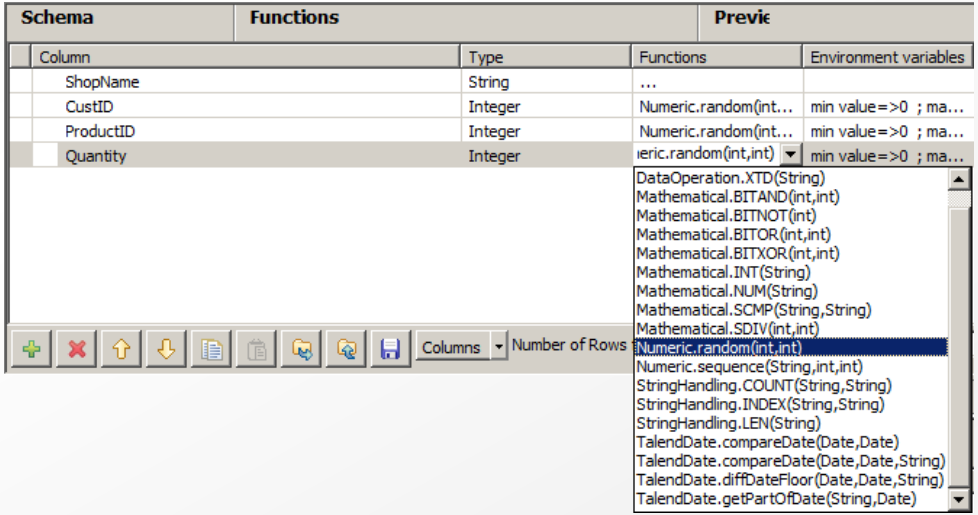
Generating Random Data



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

Generating Random Data

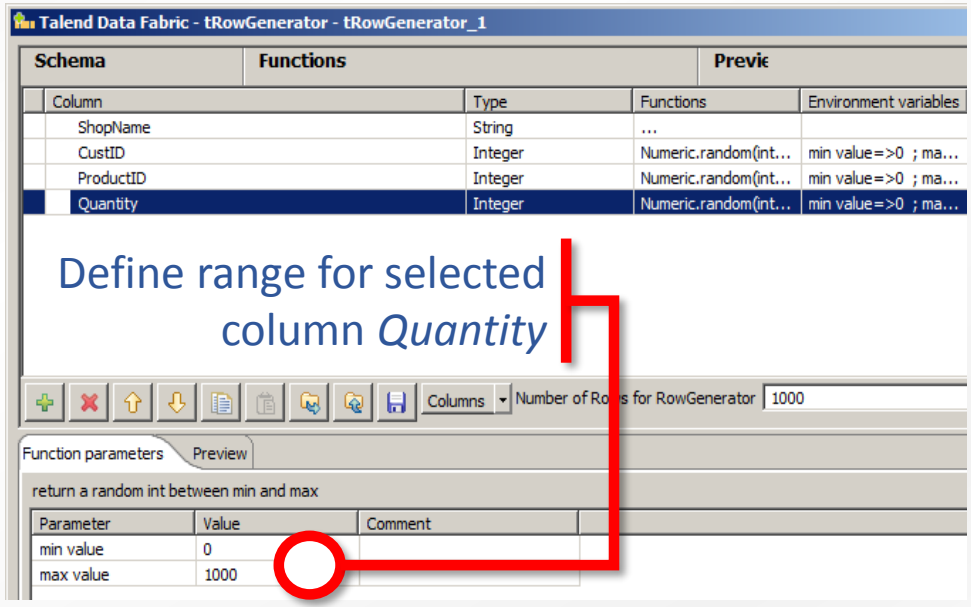
- For each column in the schema, select the function that will be used to generate the data.



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

Generating Random Data

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

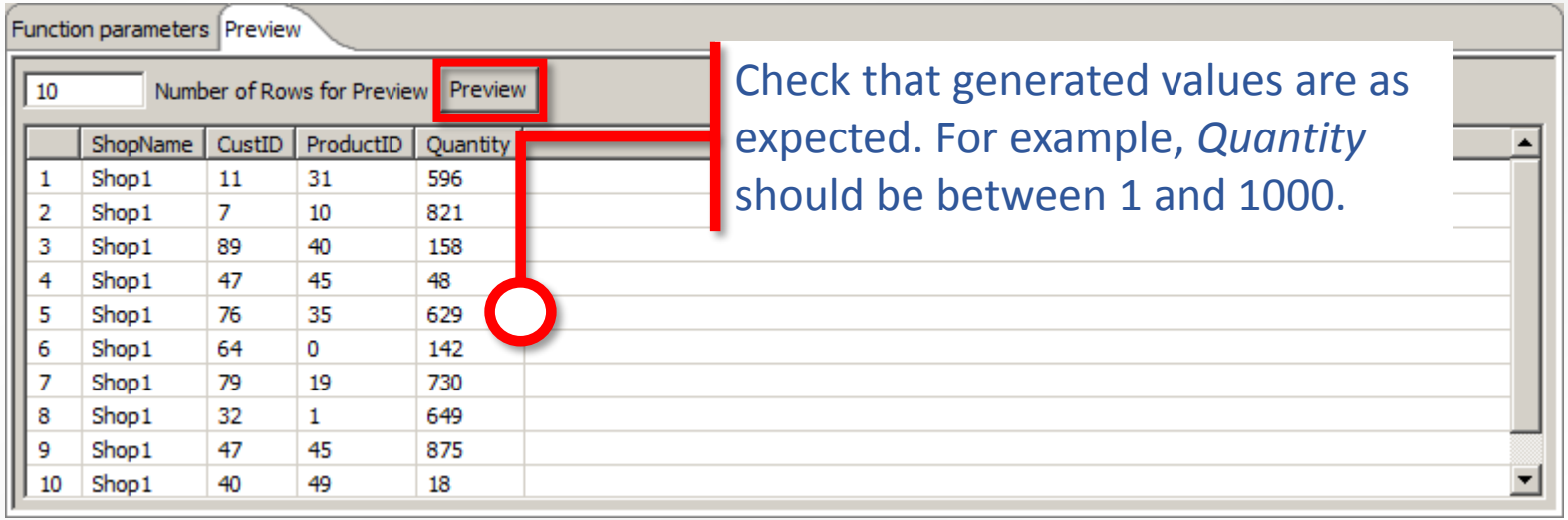


1. Define Schema
2. Set Number of Rows
3. Select Functions
4. Set Environment Variable Values

Generating Random Data

Preview the Data

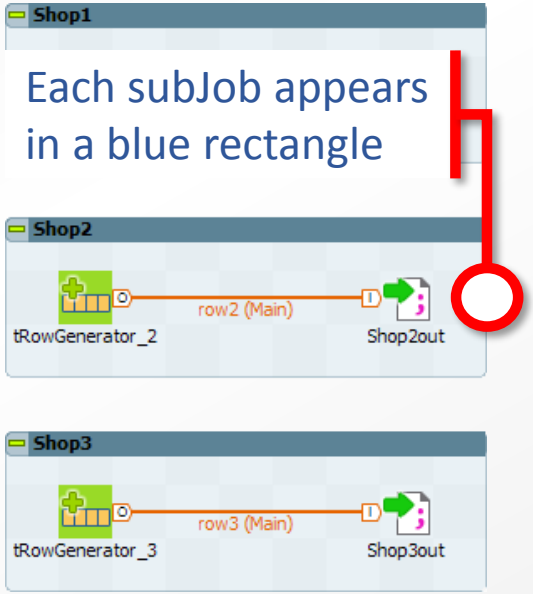
- Verify the configuration using the **Preview** feature



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

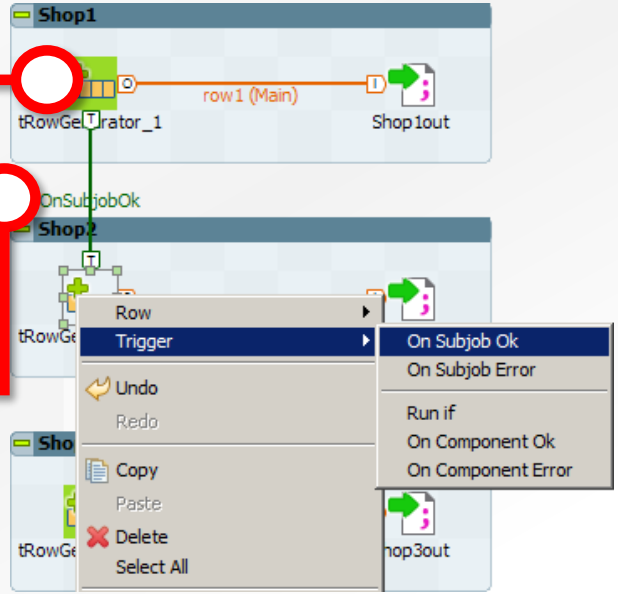
What is a Subjob?

- A collection of connected components within a Job
- SubJobs are connected with triggers



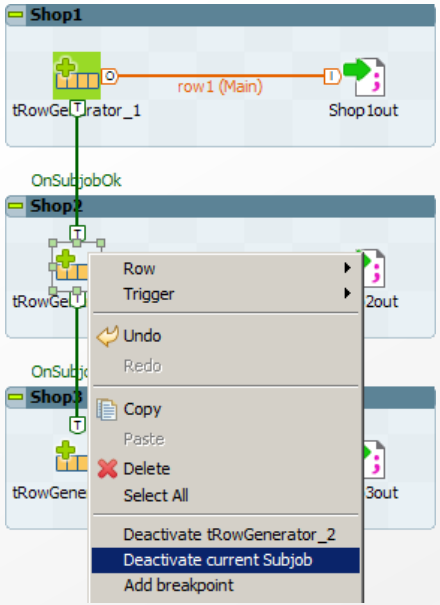
Triggers are connected to first component in subJob

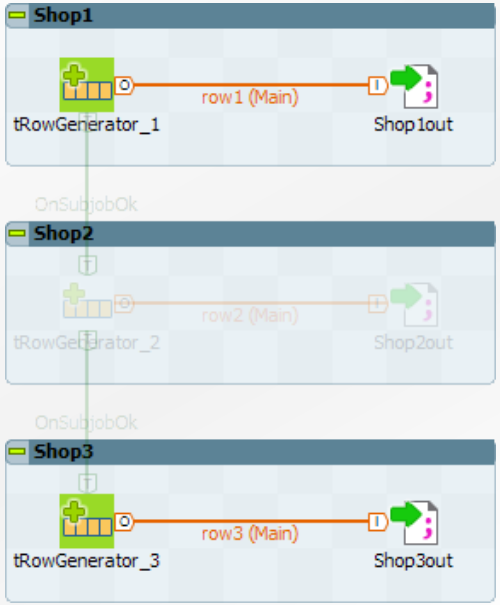
- Triggers are not flows (no data transferred)
- Used for sequencing subJob execution



Deactivating SubJobs

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

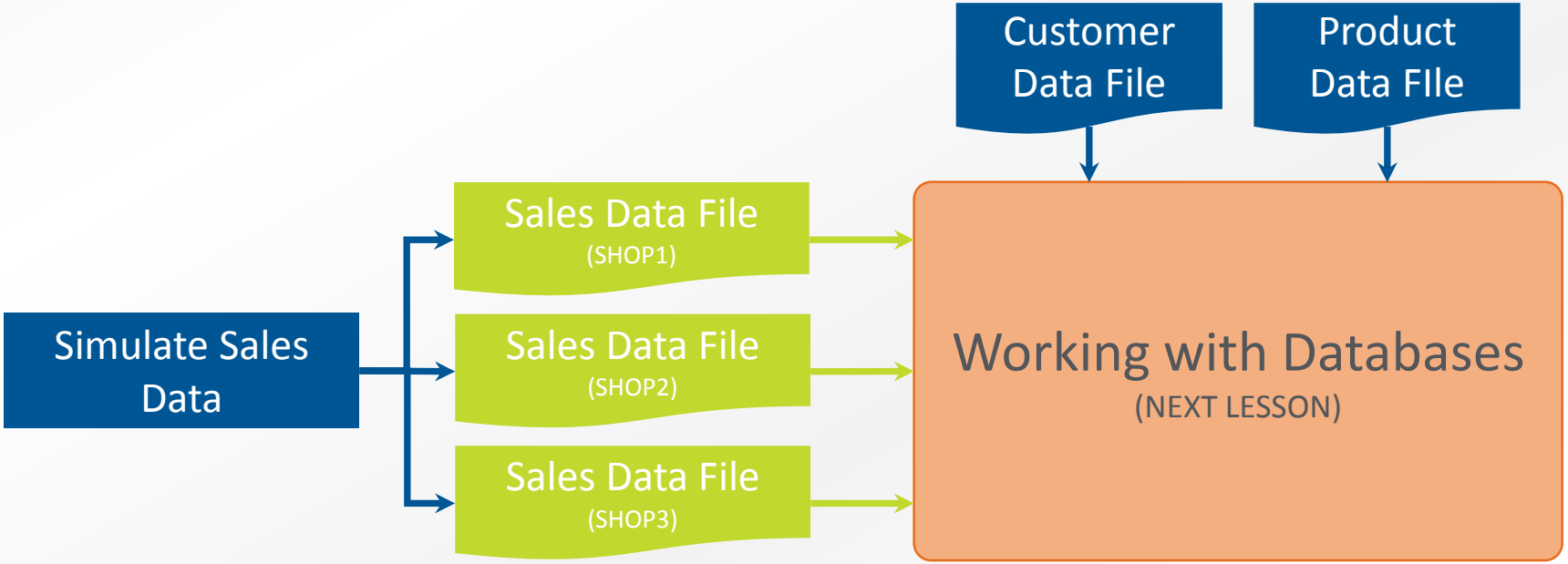




Lab Overview

Generic Schemas

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



Lesson Summary

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

Working With Databases

Objectives

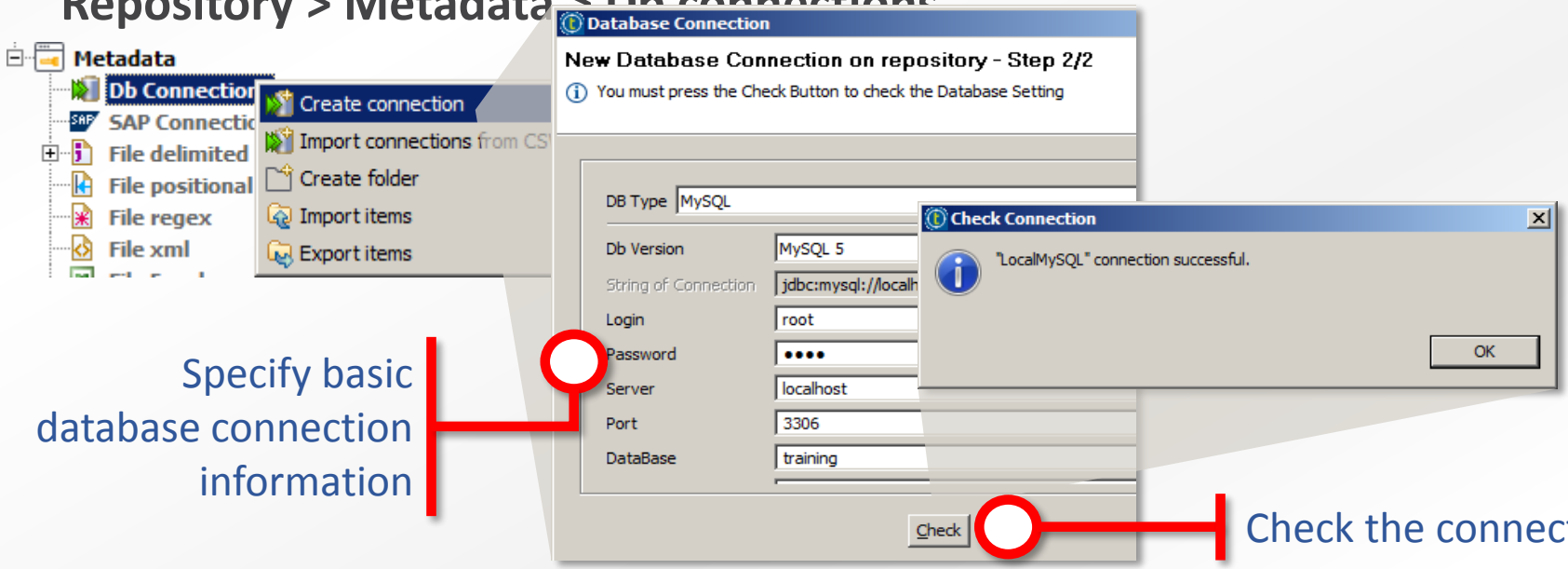
- Connect to a database from a Talend Job
- Use a component to create a database table
- Write to and read from a database table
- Filter unique data rows
- Aggregate data and perform calculations
- Write data to an XML file

Database Connections

Creating Metadata

- Database connection metadata is found under

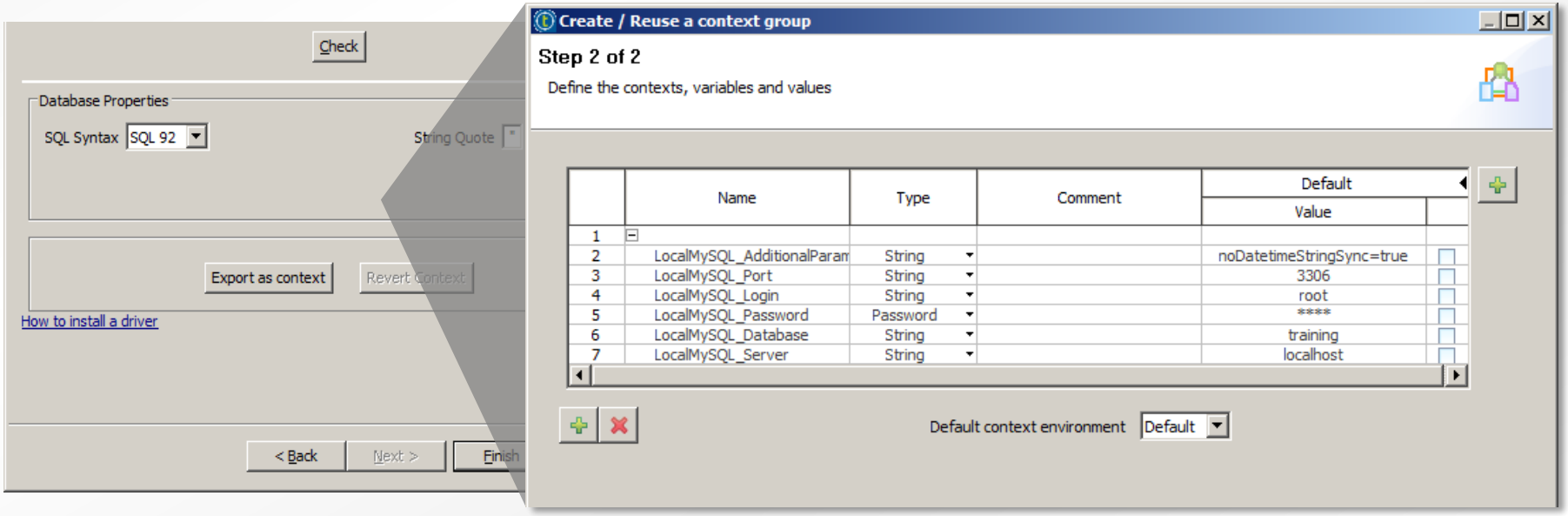
  Repository > Metadata > Db connections



Database Connections

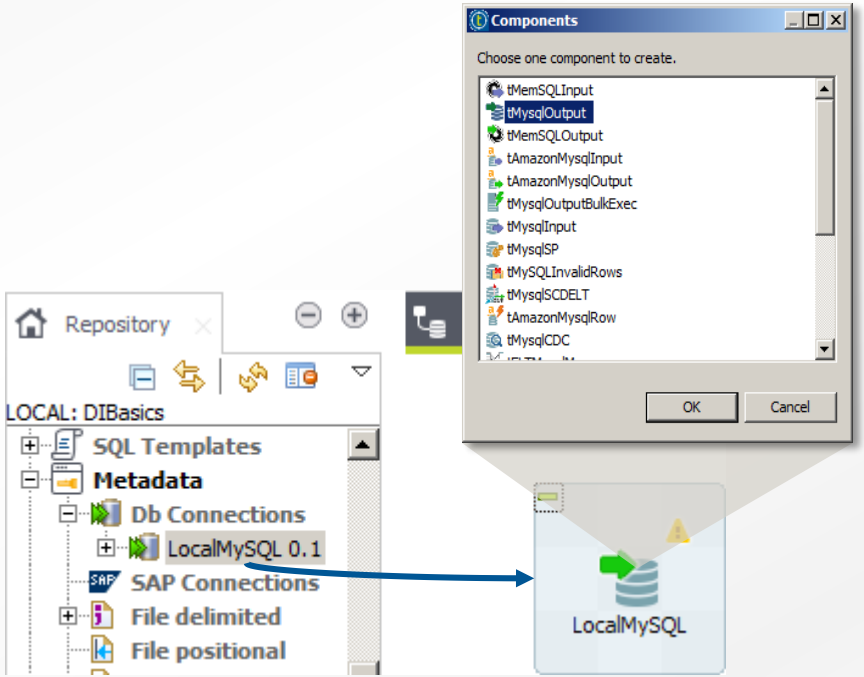
Export as Context

- Save connection information as context variables

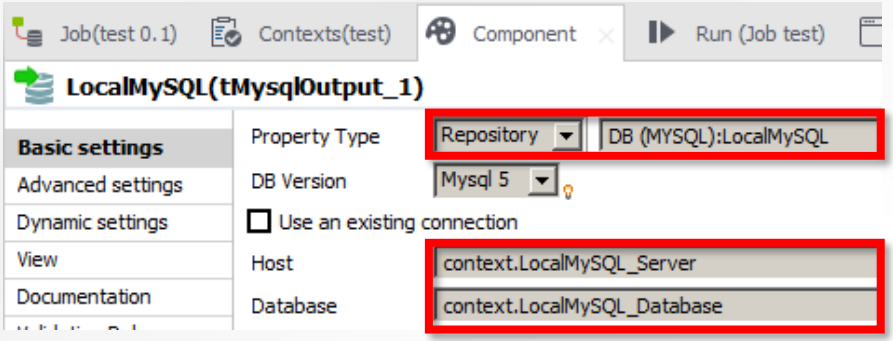


Database Connections

Using the Metadata in your Jobs



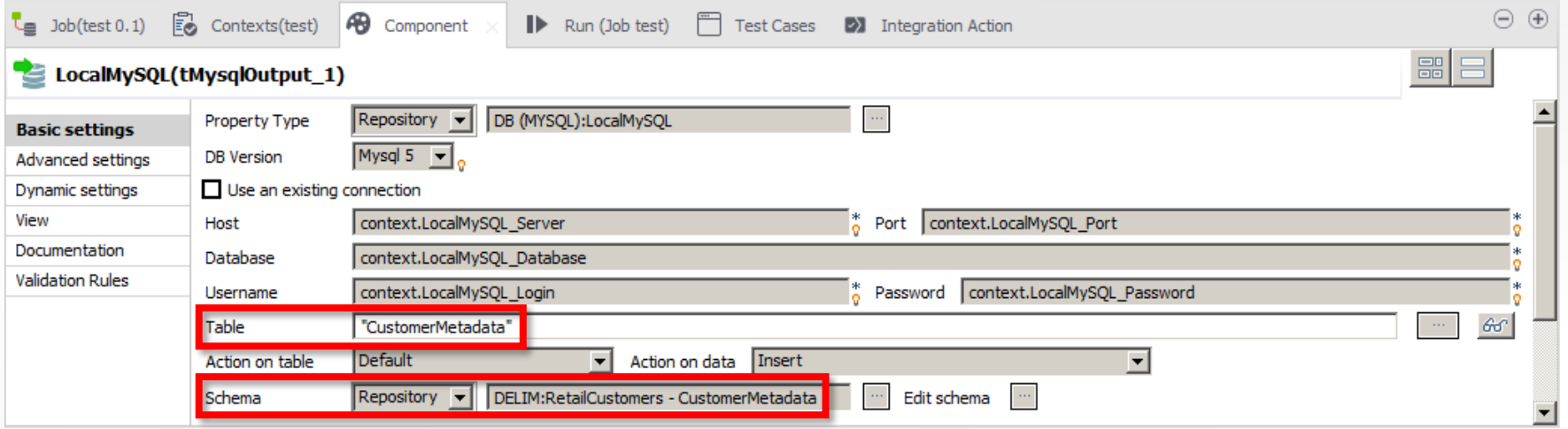
- Drag the database connection metadata to the **Designer**
  - Select a component from the list
  - Automatically configured with the connection information metadata



Writing to a Database

Specifying Table and Schema

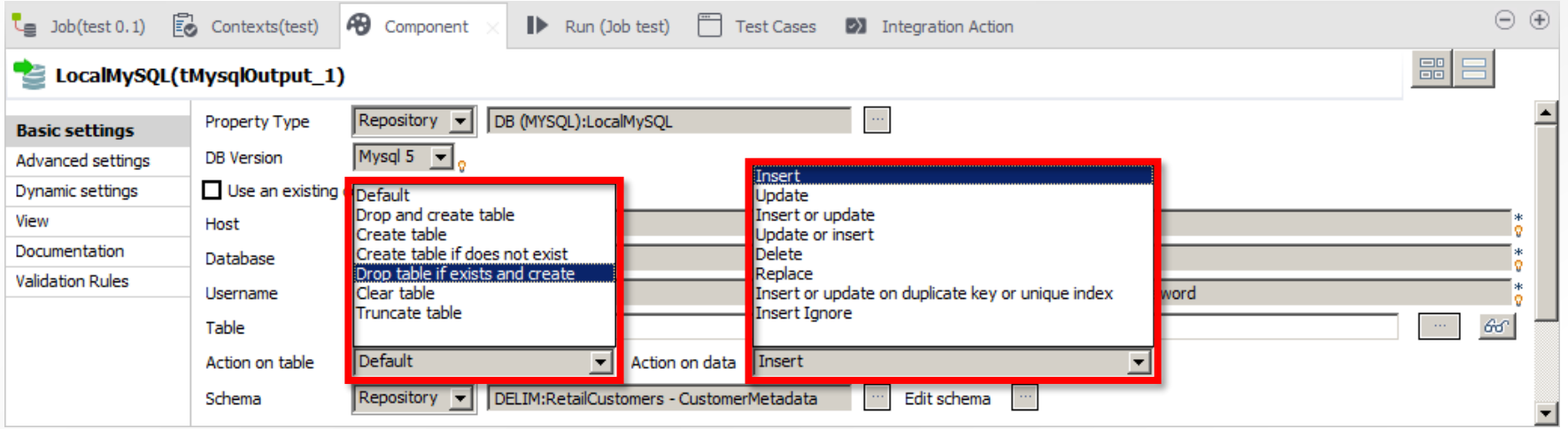
- To write to a MySQL database, use the **tMysqlOutput** component
  - Must also specify the **table name** and **schema**



Writing to a Database

Specifying Table and Data Action

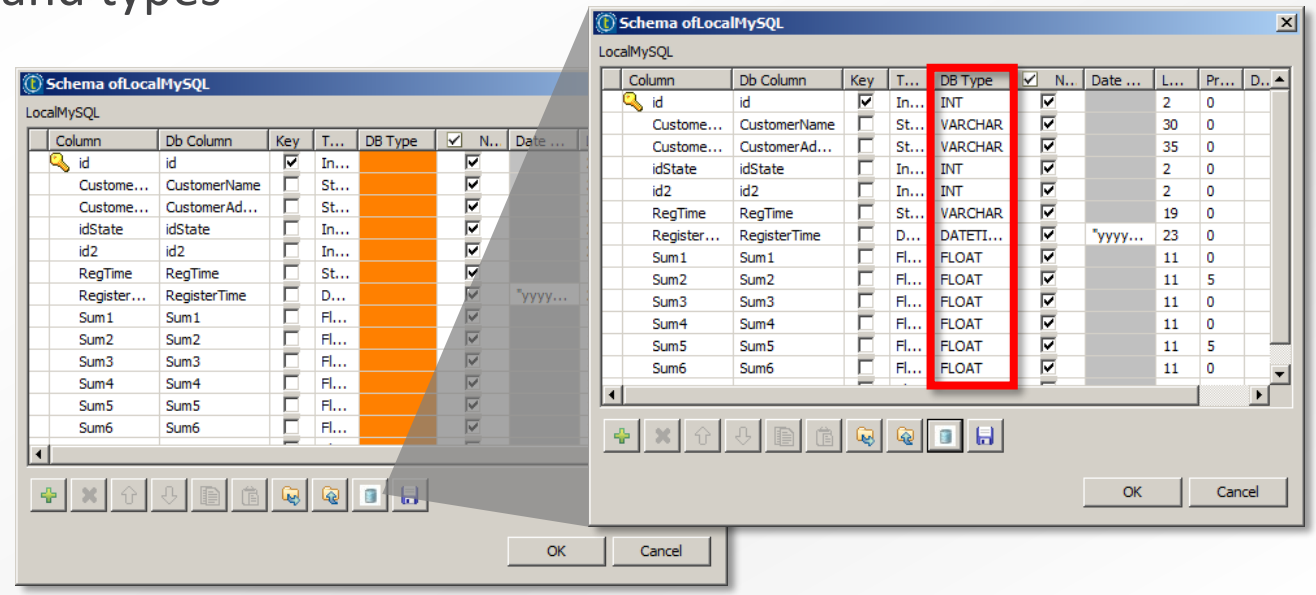
- Specify **Action on table** and **Action on data** as needed
  - Appropriate settings depend on Job requirements and database privileges



Writing to a Database

Fixing the Schema

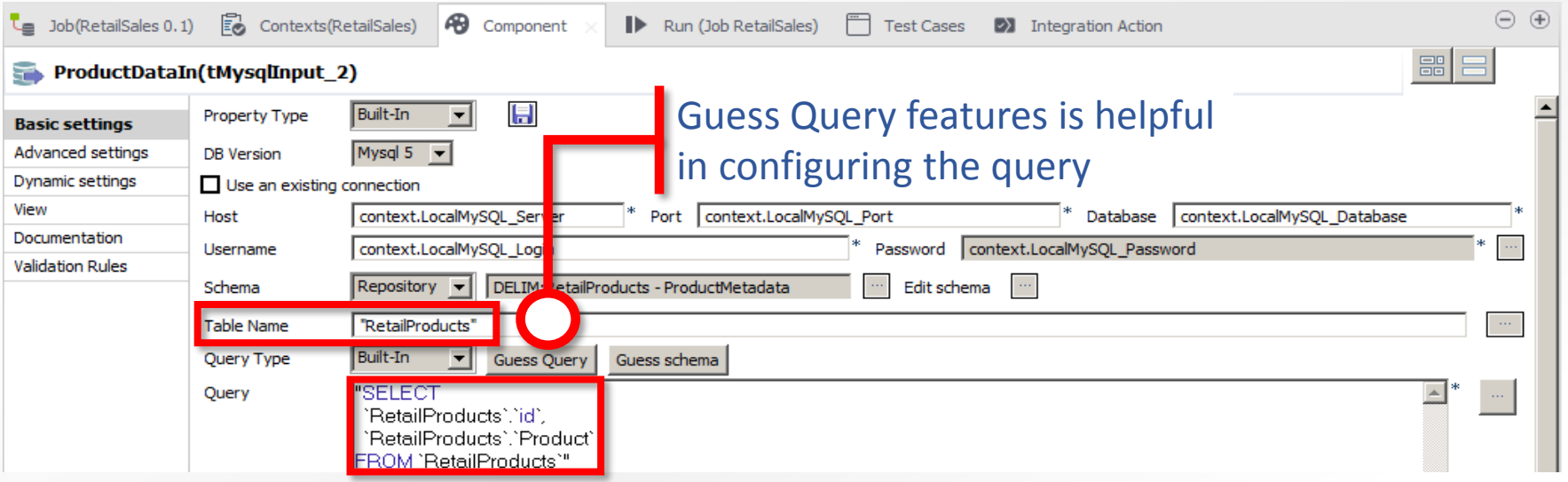
- The schema in a **tMysqlOutput** component requires Database column names and types



Reading from a Database

Specifying the Query

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



Processing Data

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

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

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

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

Lab Overview

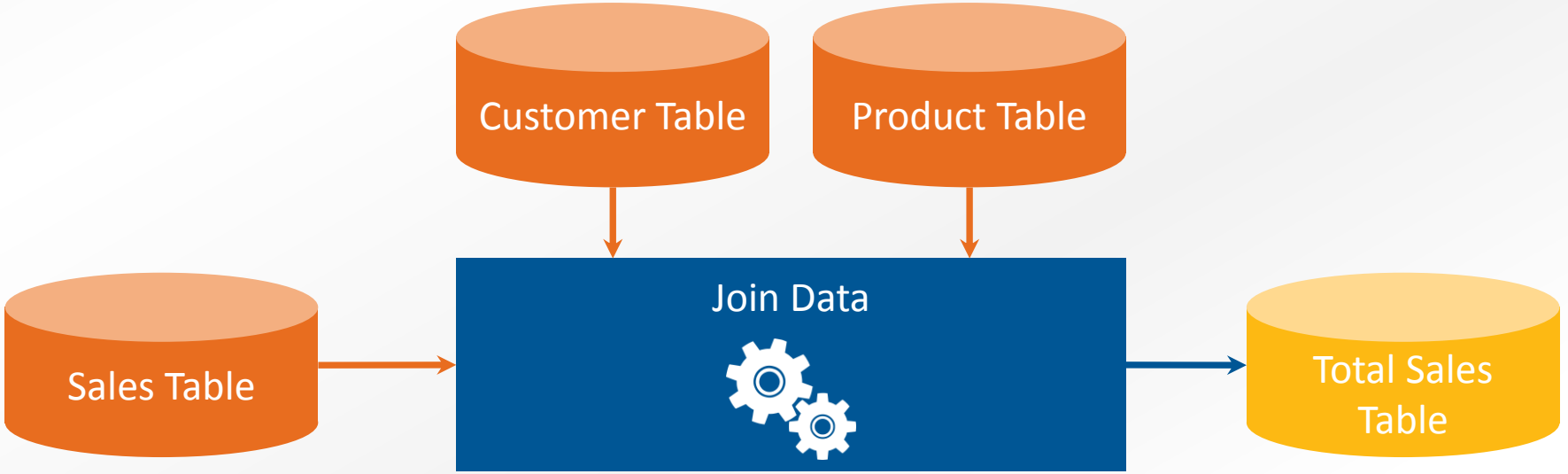
Working with Databases

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

Lab Overview

Working with Databases

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



Lesson Summary

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

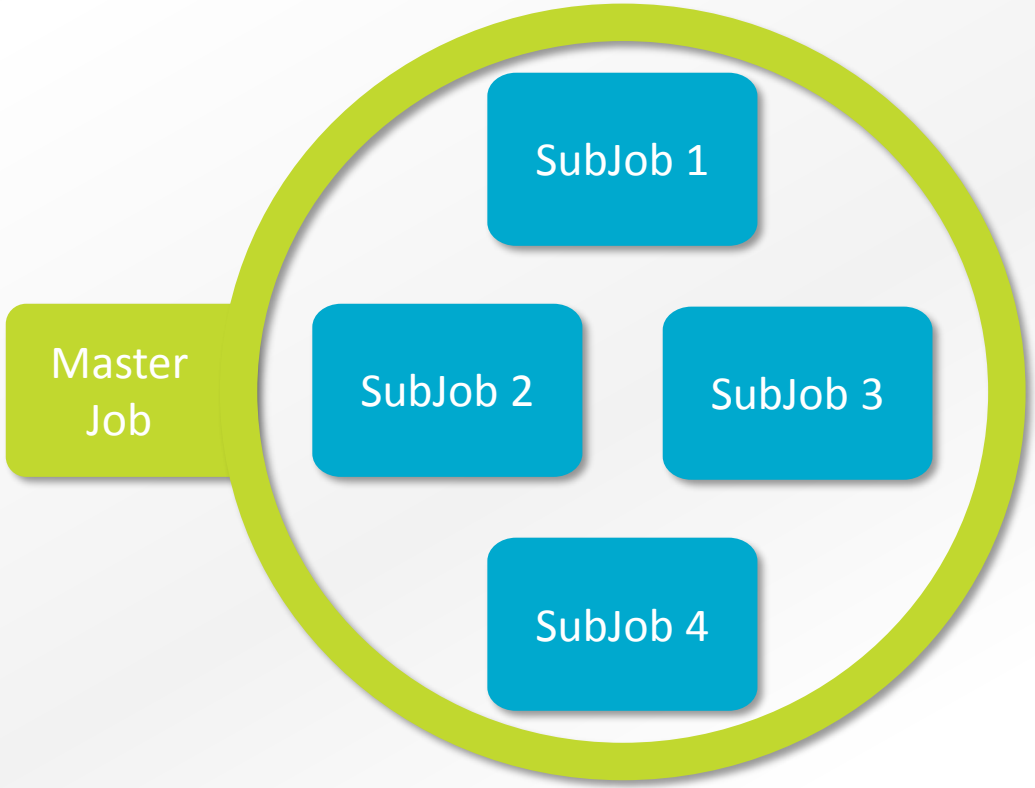
Master Jobs

Objectives

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

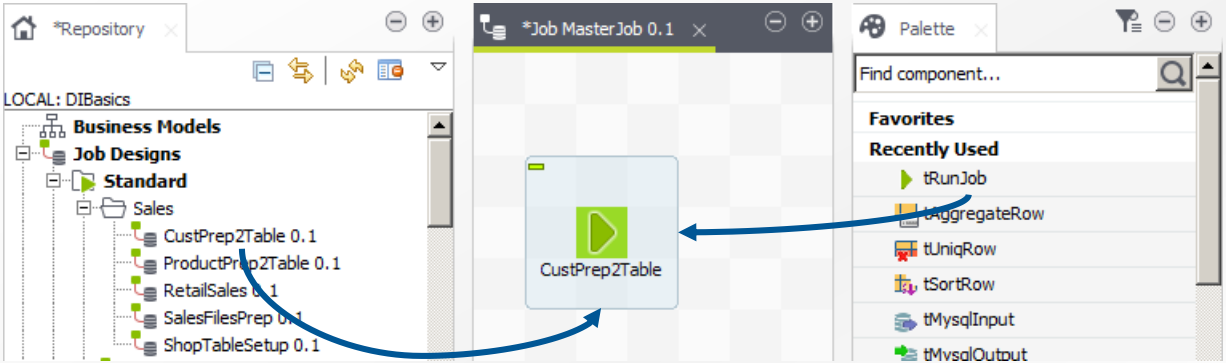
Master Jobs

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



Master Jobs

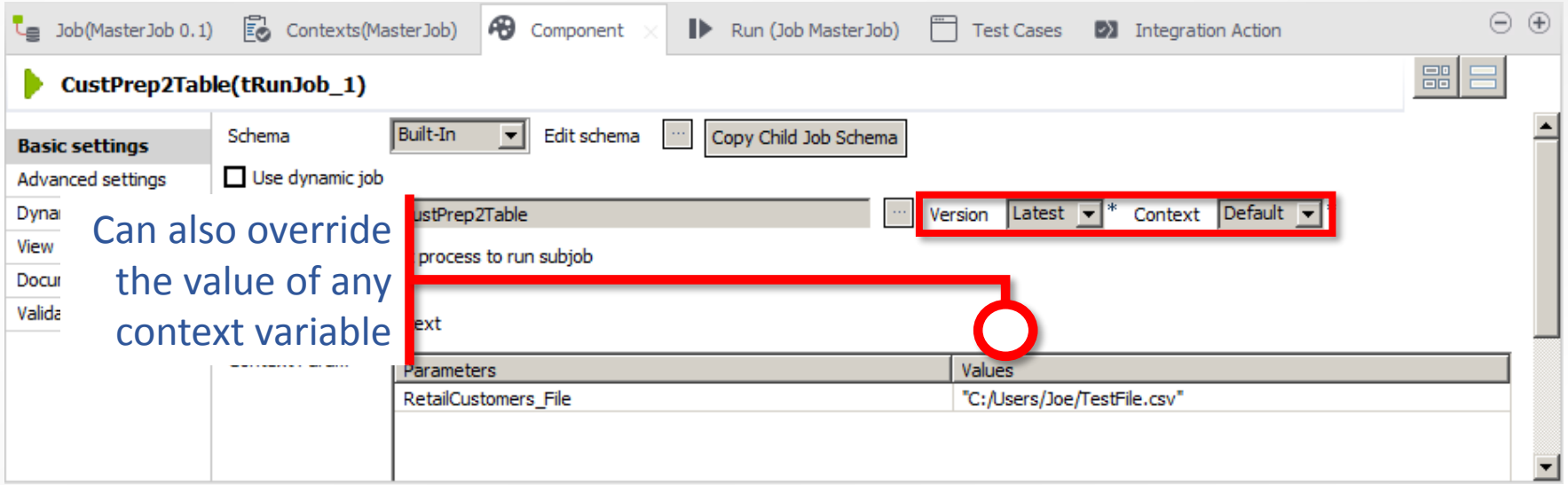
- **tRunJob** components are placed like any other component
  - They can also be placed by dragging and dropping Jobs from the repository



Master Jobs

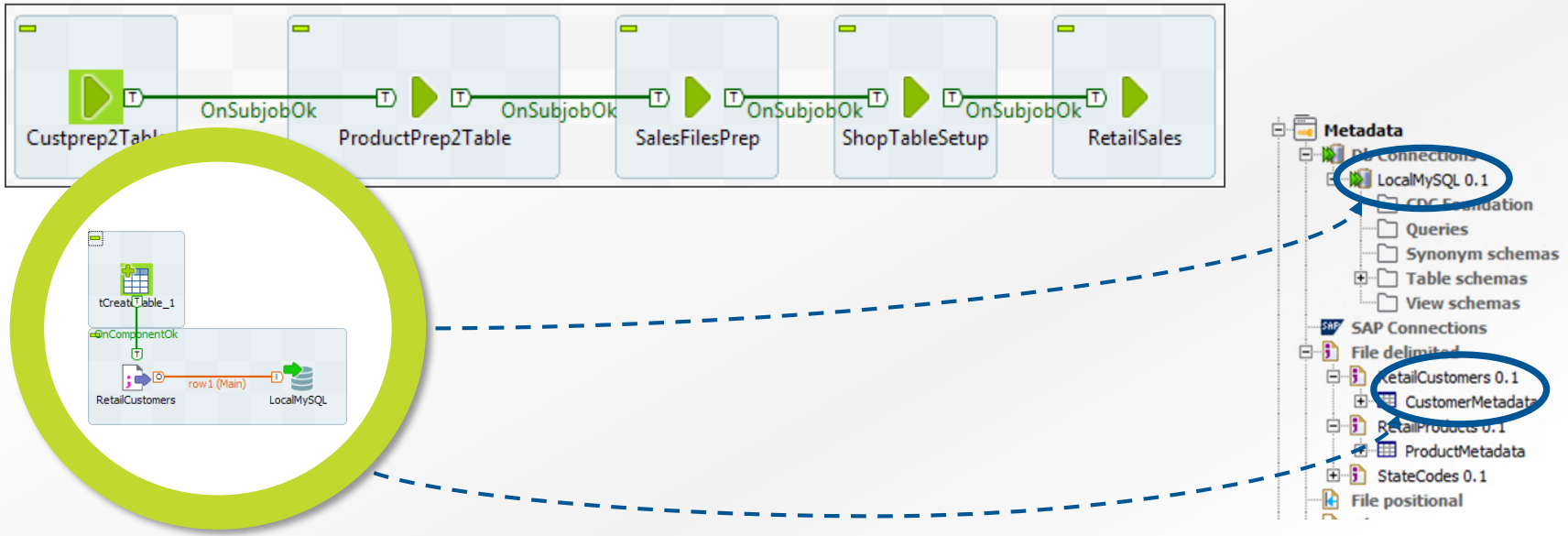
Configuration Settings

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

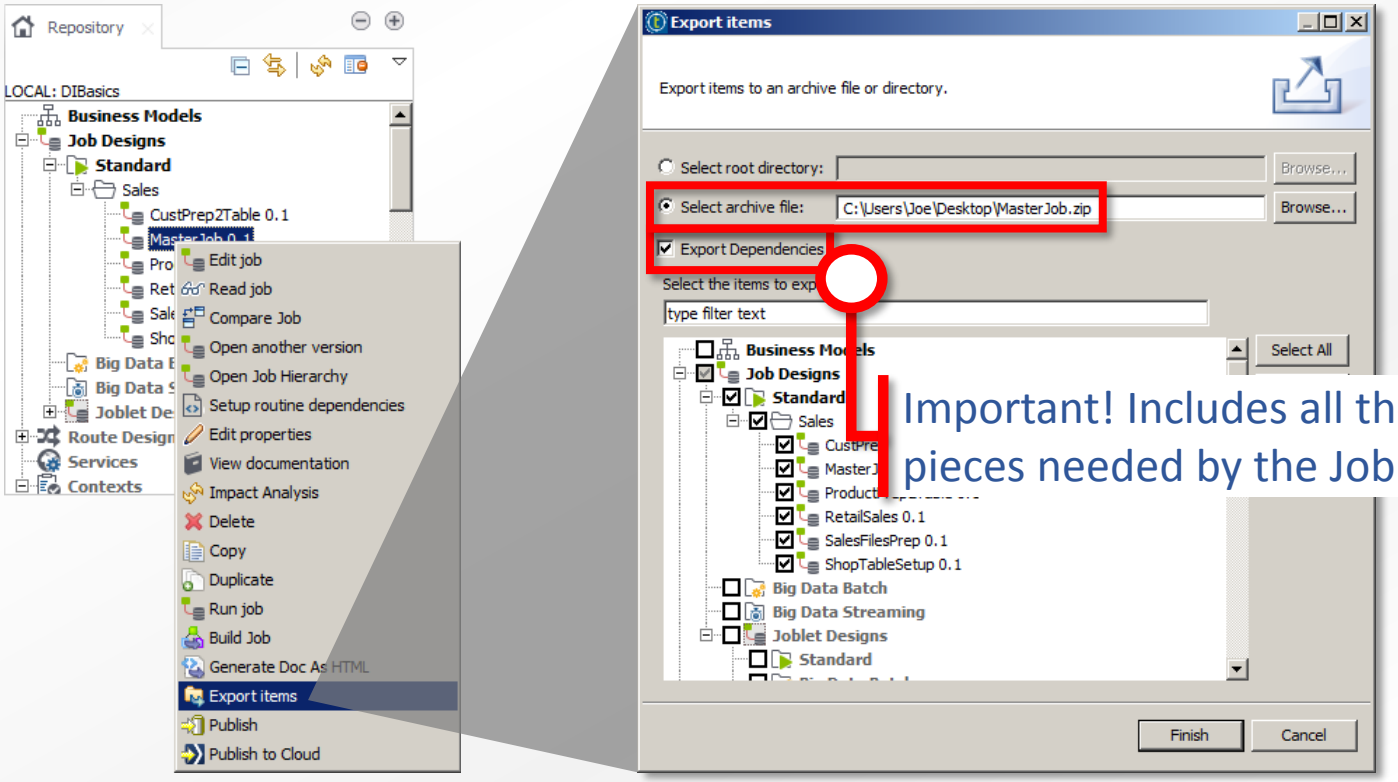


Exporting Jobs

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



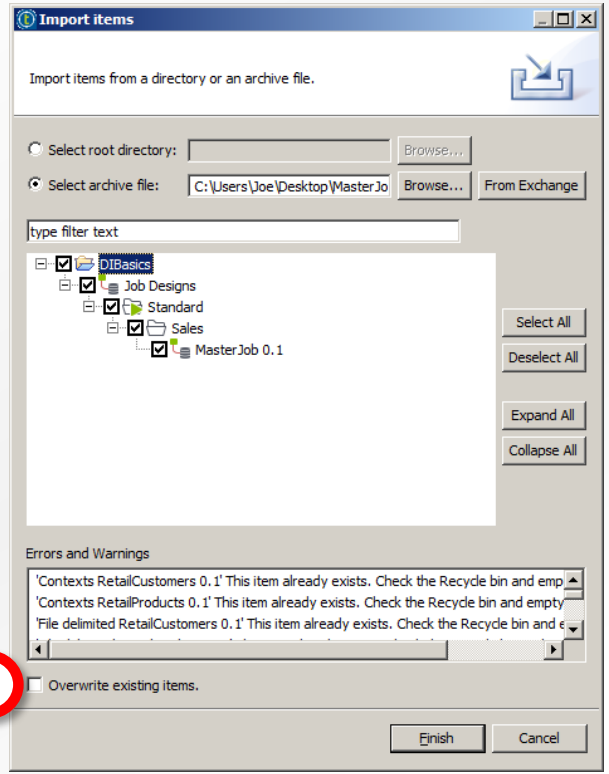
Exporting Jobs



Importing Jobs

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

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



Lab Overview

Master Jobs

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

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

Lesson Summary

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

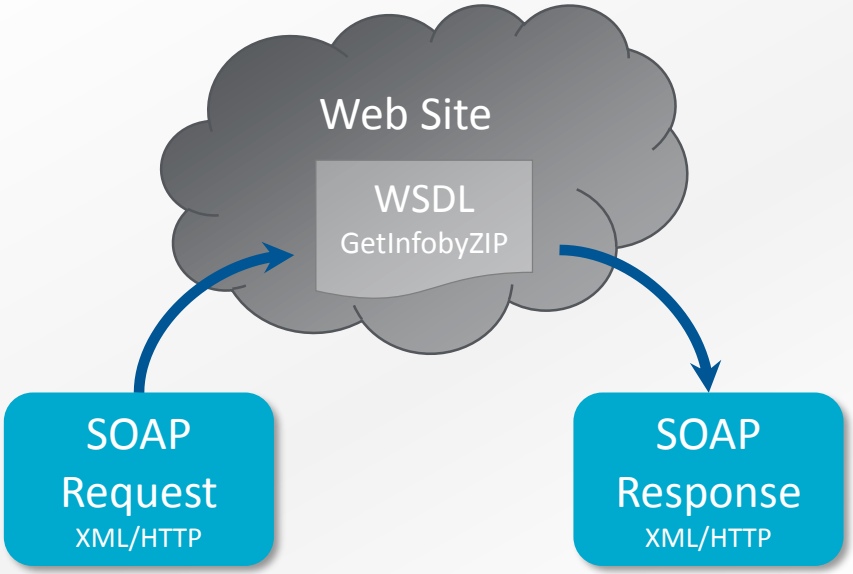
Web Services

Objectives

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

Web Services

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



Web Service Description Language

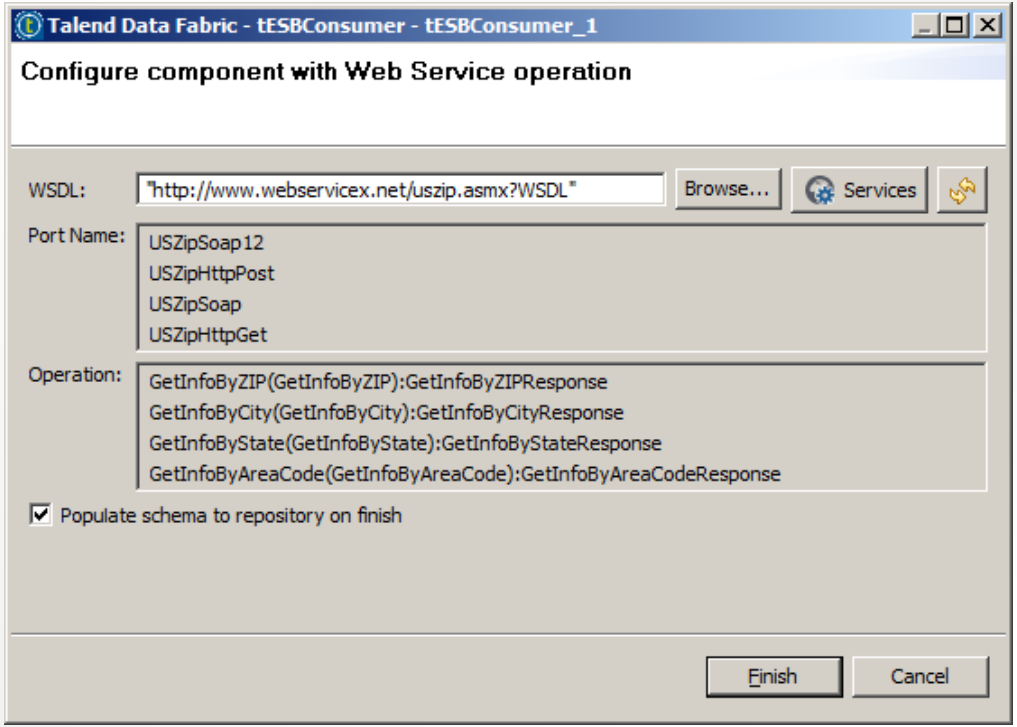
- Web Service Description Language (WSDL) is a contract between the Web Service provider and consumer

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

The tESBConsumer Component

Example Component Configuration for the Request

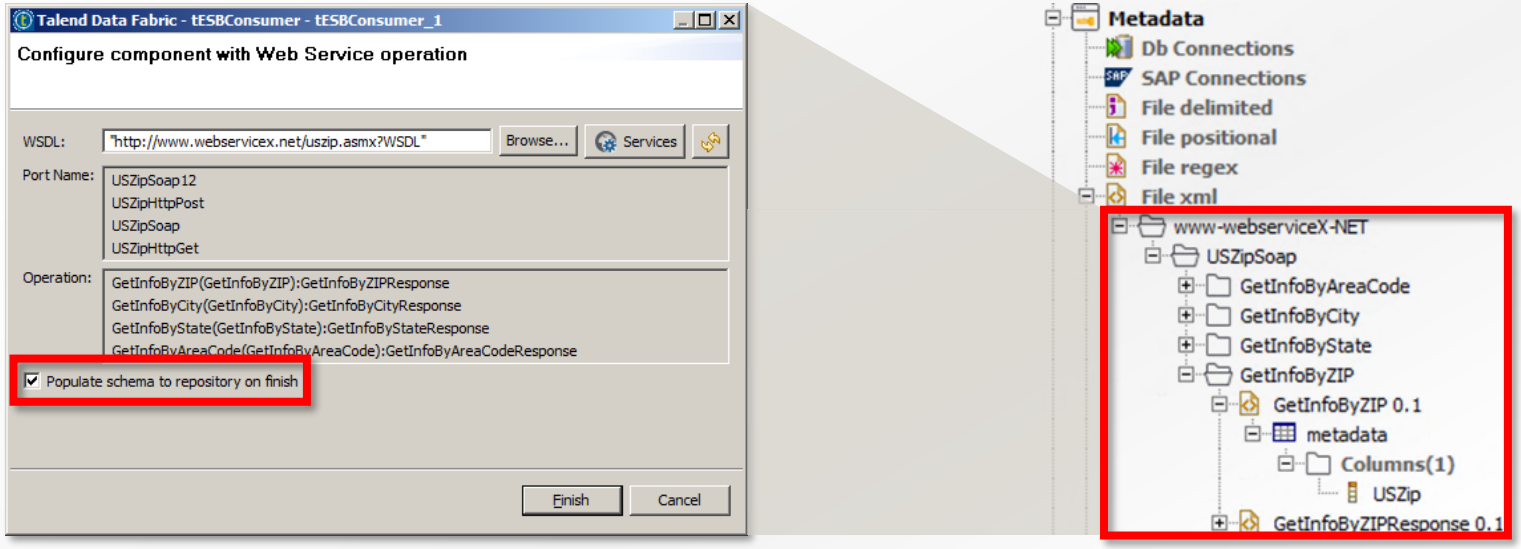
- Configure the consumer to issue a SOAP request to get city information by ZIP code
  - Specify **WSDL** URL
  - Select **Port Name** and **Operation**



The tESBConsumer Component

Example Component Configuration for the Processing the Response Schema

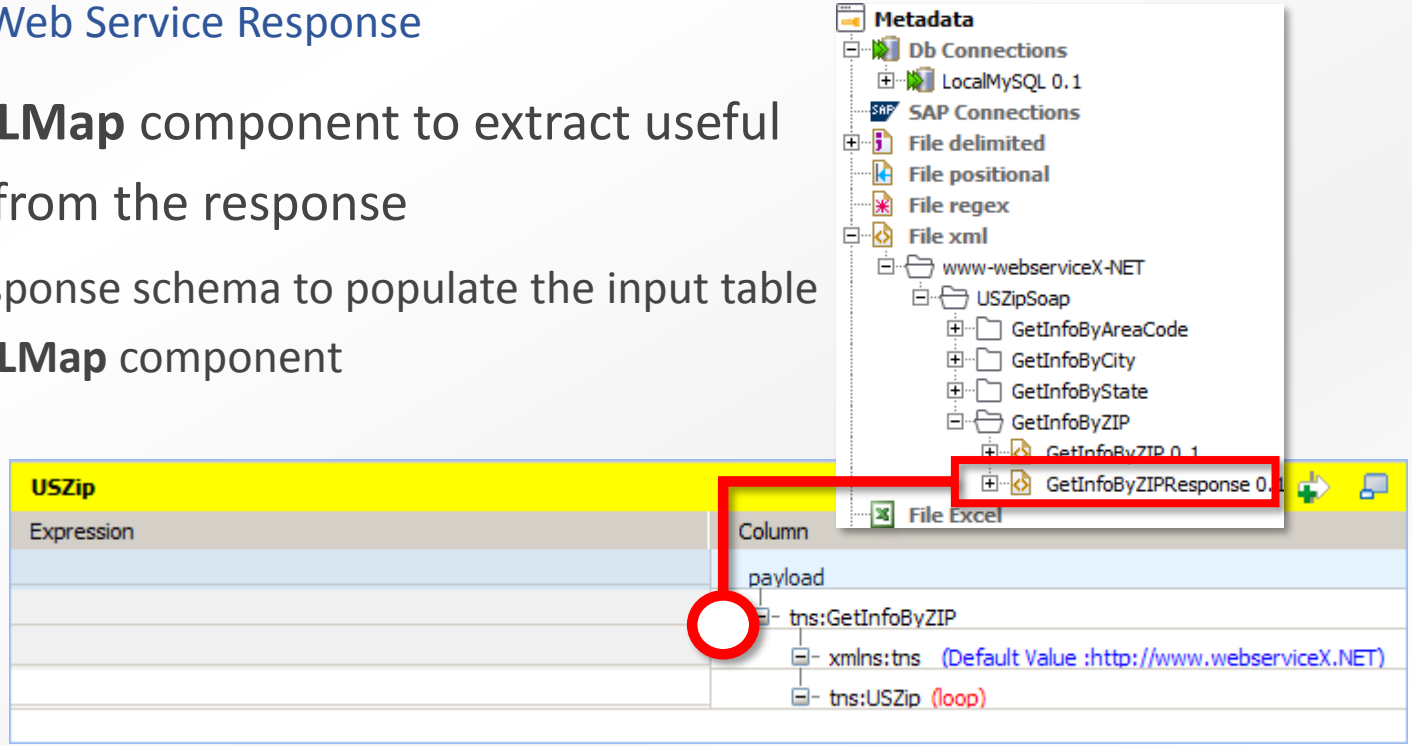
- The schema of the Web Service response can be complex
  - **tESBConsumer** allows you to retrieve the schema and save it



The tXMLMap Component

Process the XML Web Service Response

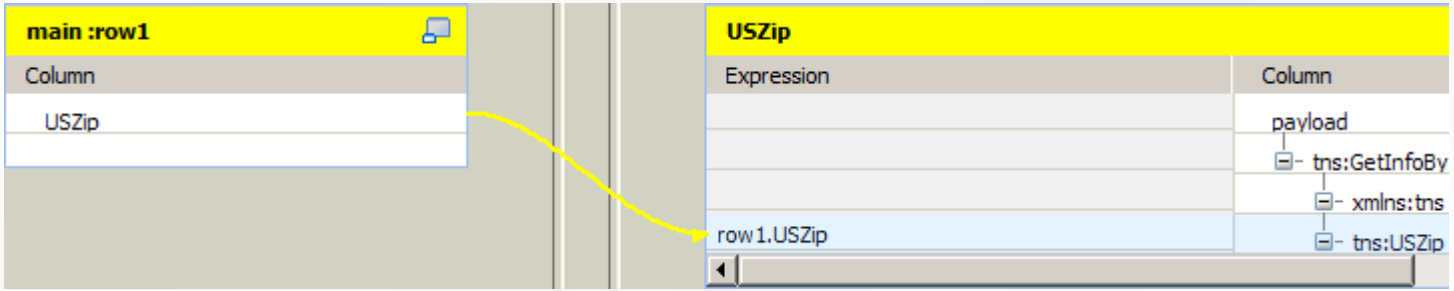
- Use the **tXMLMap** component to extract useful information from the response
  - Use the response schema to populate the input table of the **tXMLMap** component



The tXMLMap Component

Process the XML Web Service Response

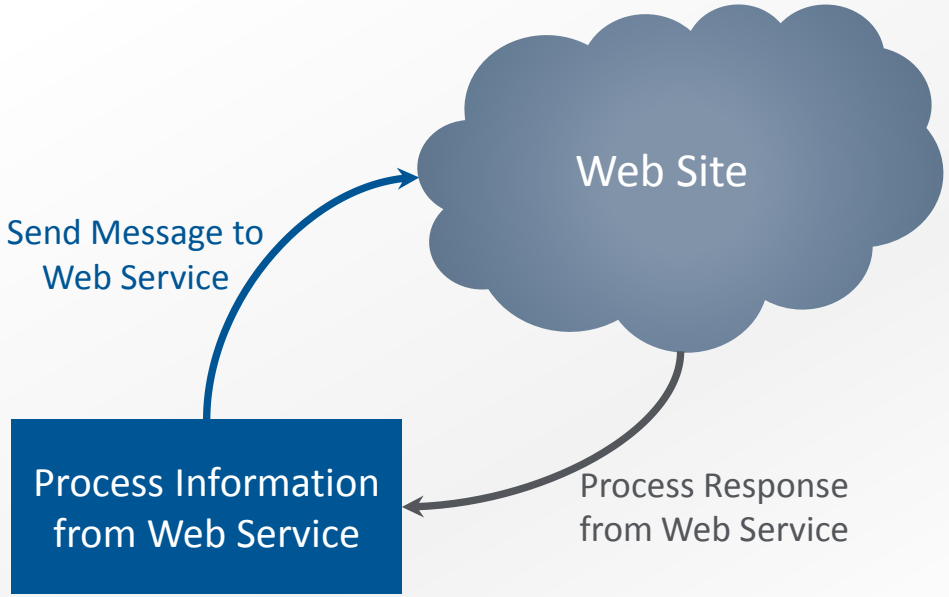
- The **tXMLMap** editor is similar to the **tMap** editor when processing the response data flow
  - Provides similar capabilities for routing and transforming data from one or more data sources to one or more outputs



Lab Overview

Working with Web Services

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



Lesson Summary

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

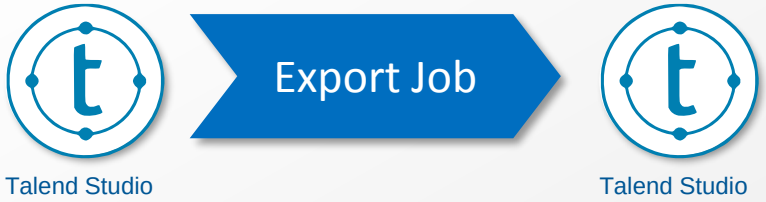
Standalone Jobs

Objectives

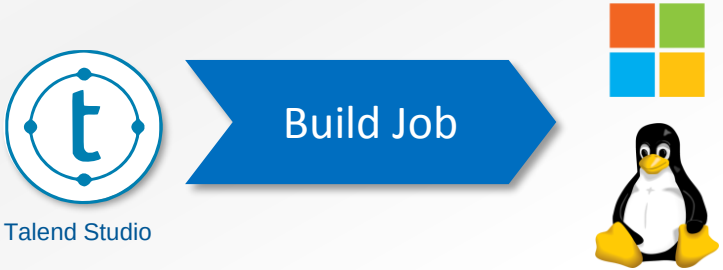
- Build and export a Job
- Run an exported Job independently of the Talend Studio
- Create a new version of an existing Job

Building a Job

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

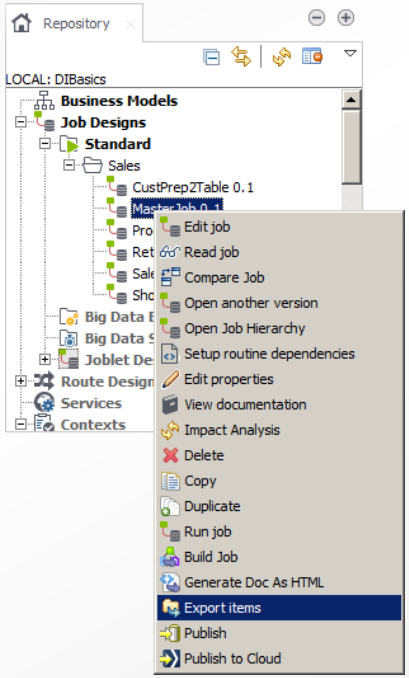


- Building a Job
  - Bundles a Job and its dependencies into a standalone format
  - Job can be run anywhere (outside of Studio)
  - Delivers flexible deployment options

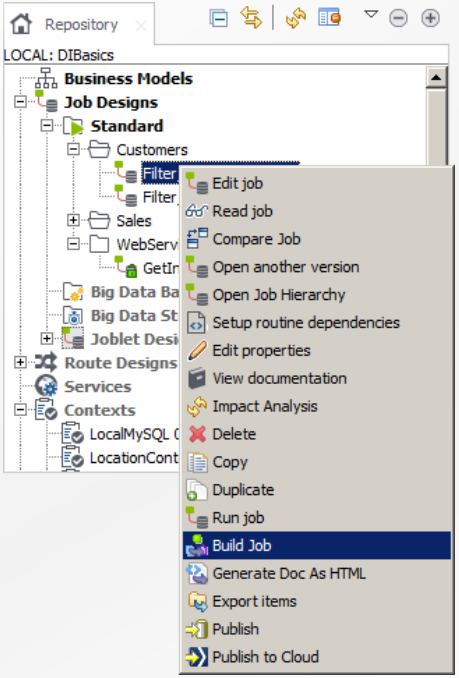


Building a Job

- Exporting a Job



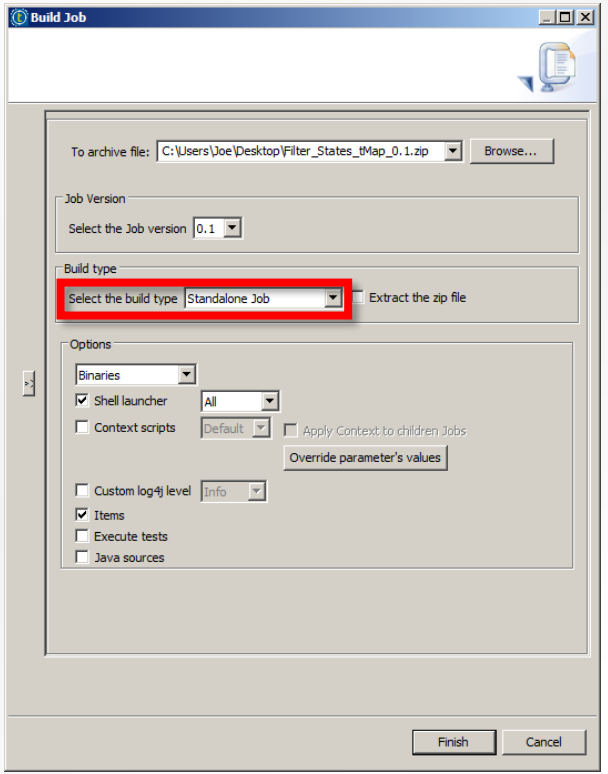
- Building a Job



Building a Job

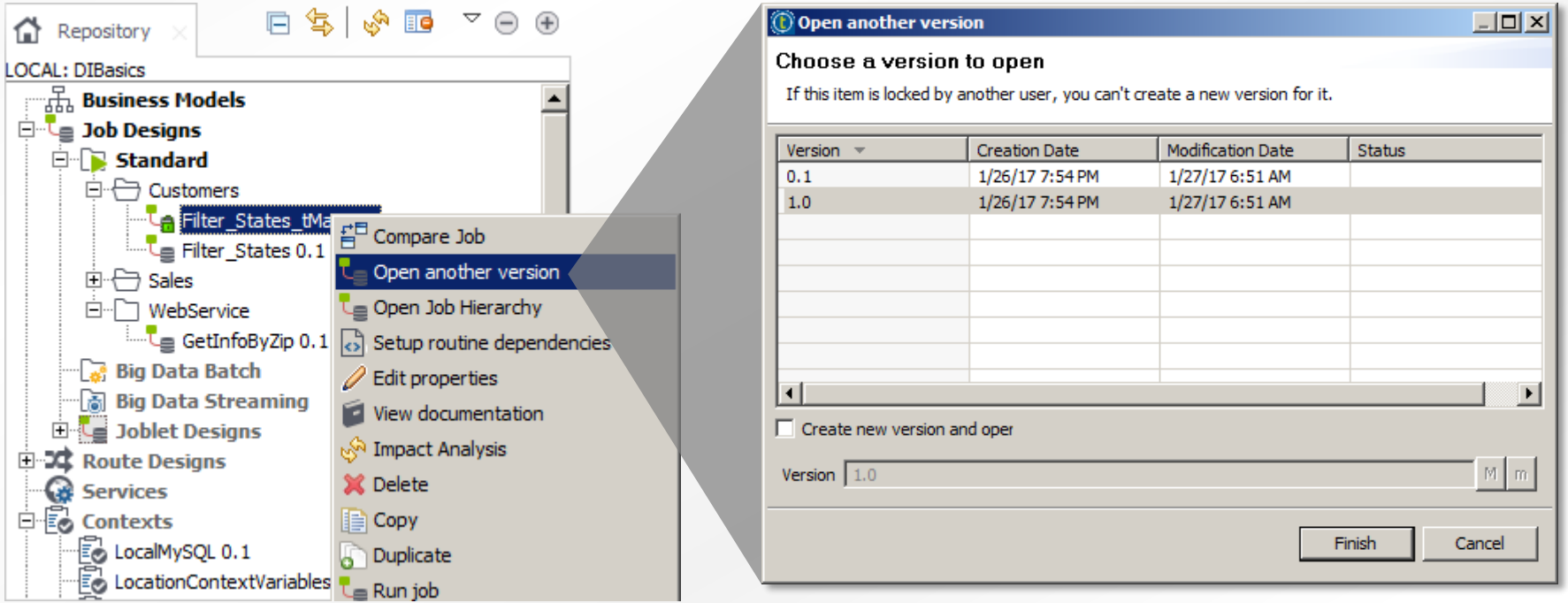
Build Type

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



Job Versions

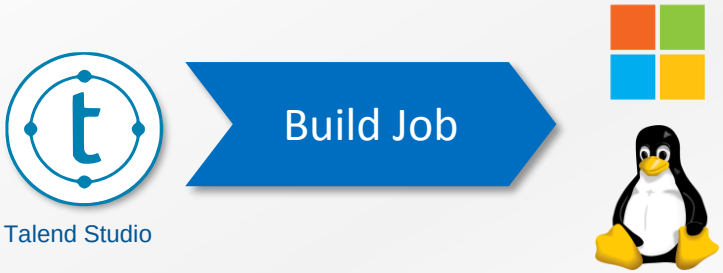
- Studio offers basic version control (**major.minor** numbering scheme)



Lab Overview

Standalone Jobs

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



Lesson Summary

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

Best Practices and Documentation

Objectives

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

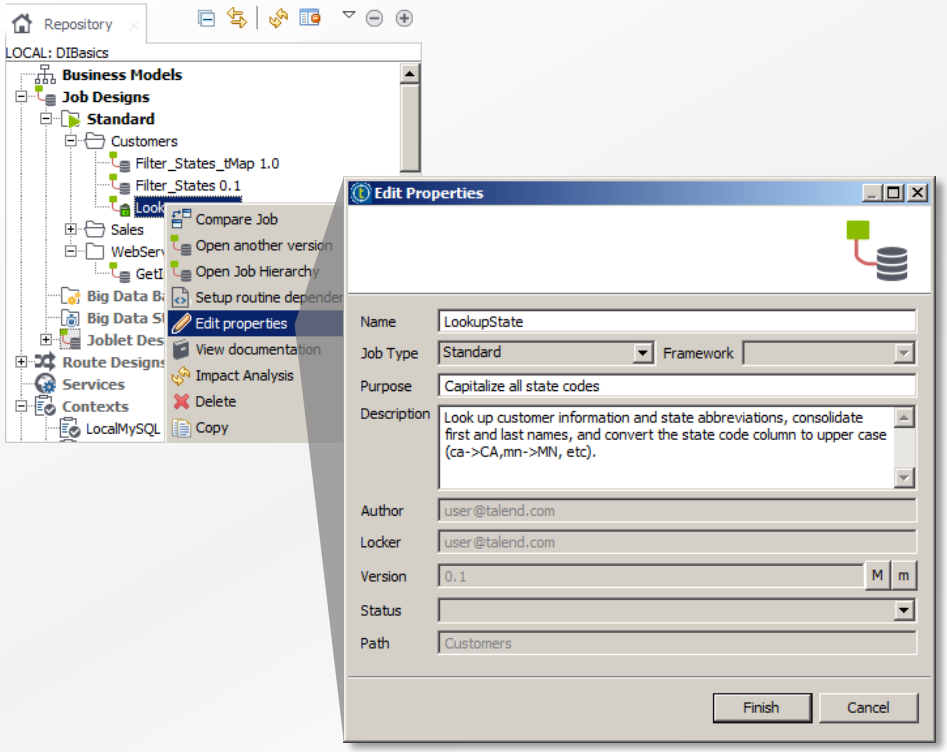
Scenario

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

Best Practices

Naming and Describing Your Jobs

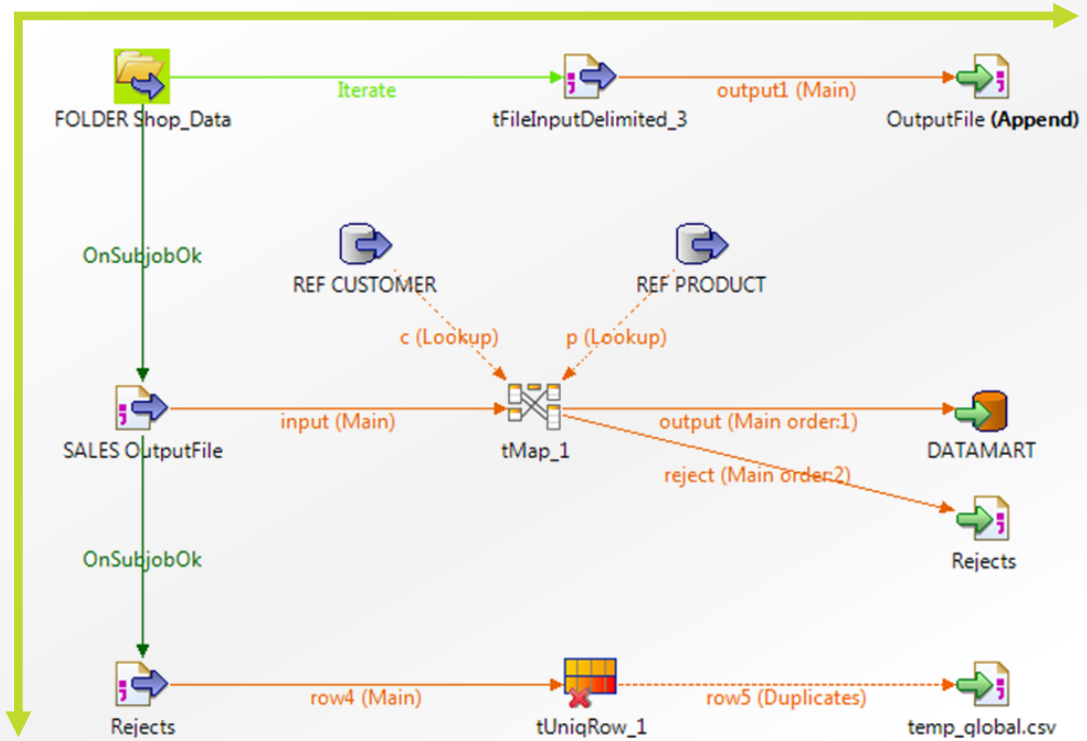
- Specify important information when creating new Jobs
  - **Name**: adopt and follow a consistent naming convention
  - **Purpose:** brief description
  - **Description:** detailed description that appears in repository hover text
- Can also add or modify at any time after creation



Best Practices

Job Layout

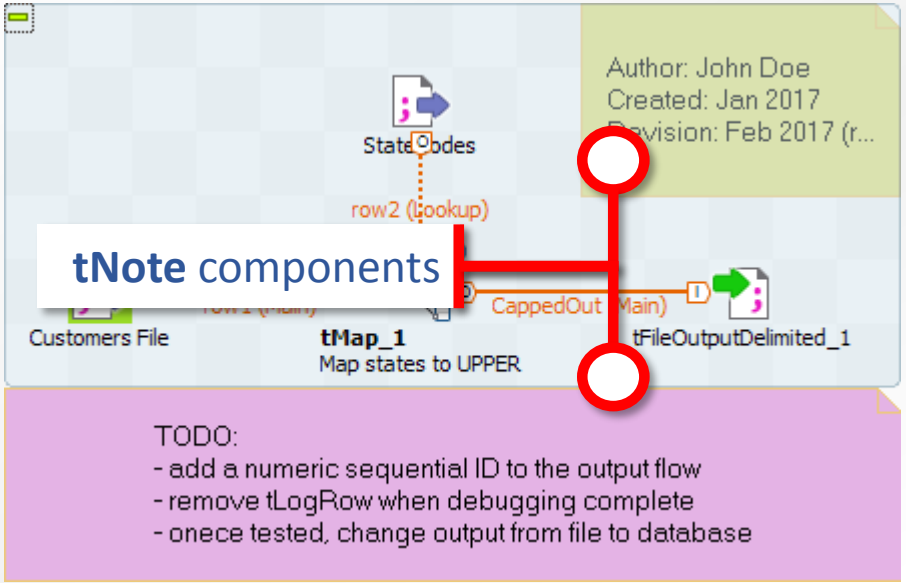
- Work from the top-left corner.
  - Place and connect components from left to right.
  - Work also from top to bottom.



Best Practices

Annotating Jobs with **tNote**

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

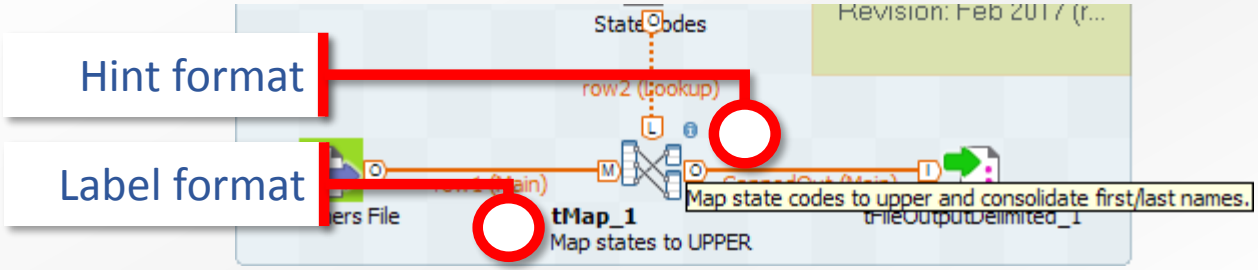


Best Practices

Labelling Components

- Customizing the component label improves readability in the **Designer**

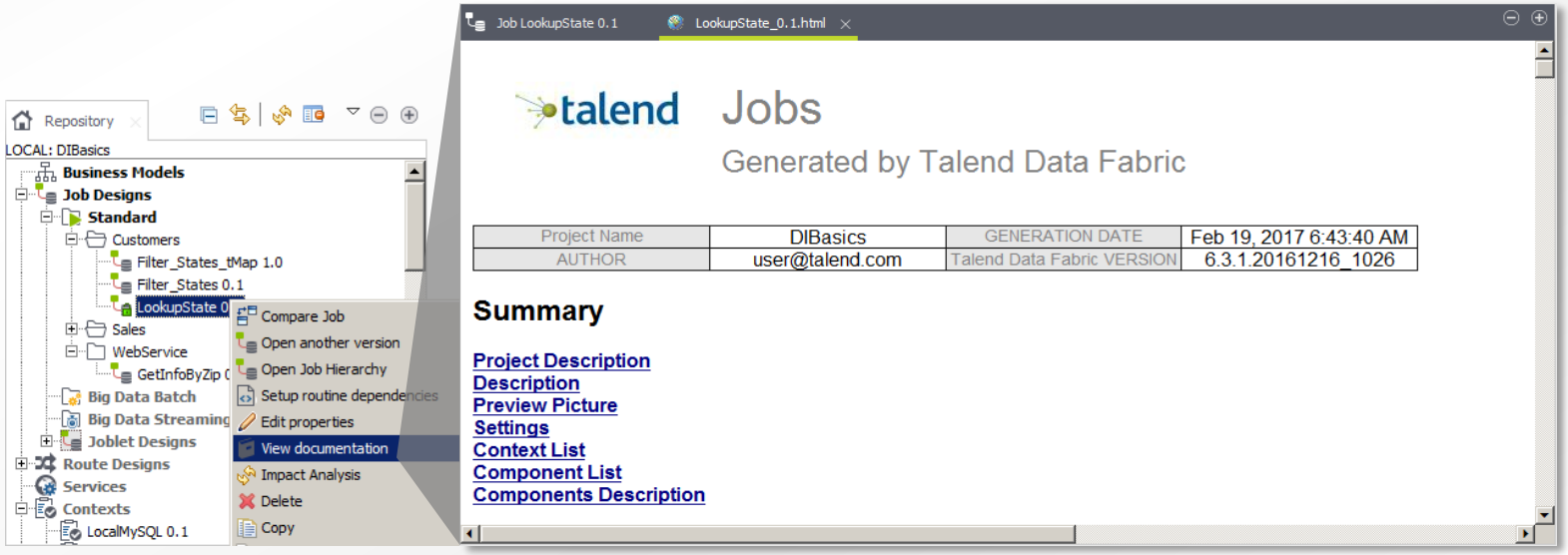
Field	Description	Default Value
Label format	Text label shown in **Designer** (component name)	__UNIQUE_NAME__
Hint format	Hidden tooltip. Visible when hovering over component and **Show Information** option is selected.	__UNIQUE_NAME__ __COMMENT__



Documentation

View Documentation in the Studio During Development

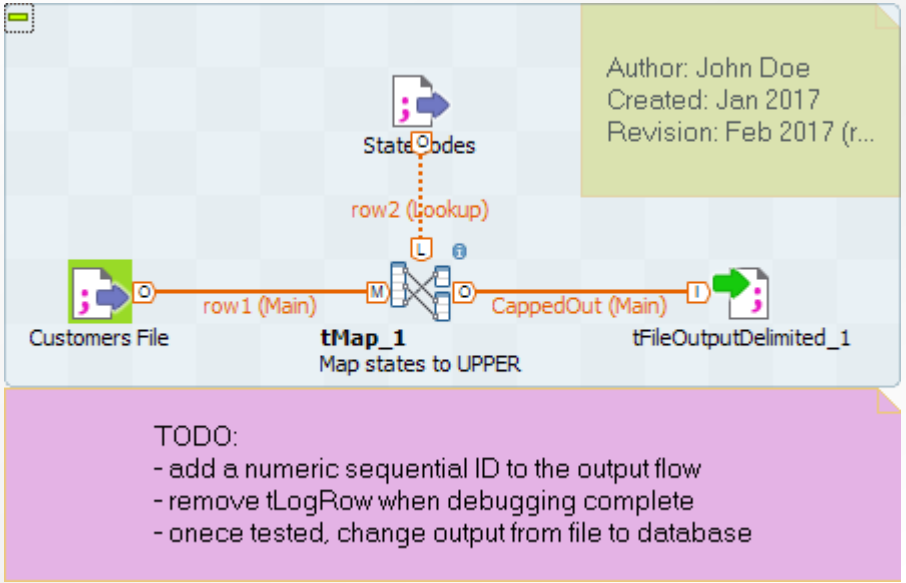
- Open a tab with HTML documentation for review
  - When development is done, generate a permanent version of the documentation



Lab Overview

Best Practices and Documentation

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



Lesson Summary

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

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