DI Advanced 6.0

Table Of Contents

Welcome to Talend Training

Congratulations on choosing a Talend training module. Take a minute to review the following points to help you get the most from your experience.

Technical Difficulty

Instructor-Led

If you are following an instructor-led training (ILT) module, there will be periods for questions at regular intervals. However, if you need an answer in order to proceed with a particular lab, or if you encounter a situation with the software that prevents you from proceeding, don’t hesitate to ask the instructor for assistance so it can be resolved quickly.

Self-Paced

If you are following a self-paced, on-demand training (ODT) module, and you need an answer in order to proceed with a particular lab, or you encounter a situation with the software that prevents you from proceeding with the training module, a Talend professional consultant can provide assistance. Double-click the **Live Expert** icon on your desktop to go to the Talend Live Support login page (you will find your login and password in your ODT confirmation email). The consultant will be able to see your screen and chat with you to determine your issue and help you on your way. Please be considerate of other students and only use this assistance if you are having difficulty with the training experience, not for general questions.

Exploring

Remember that you are interacting with an actual copy of the Talend software, not a simulation. Because of this, you may be tempted to perform tasks beyond the scope of the training module. Be aware that doing so can quickly derail your learning experience, leaving your project in a state that is not readily usable within the tutorial, or consuming your limited lab time before you have a chance to finish. For the best experience, stick to the tutorial steps! If you want to explore, feel free to do so with any time remaining after you've finished the tutorial (but note that you cannot receive consultant assistance during such exploration).

Additional Resources

After completing this module, you may want to refer to the following additional resources to further clarify your understanding and refine and build upon the skills you have acquired:

- Talend product documentation (help.talend.com)
- Talend Forum (talendforge.org/)
- Documentation for the underlying technologies that Talend uses (such as Apache) and third-party applications that complement Talend products (such as MySQL Workbench)

Begin

Start your module now with the Introduction.

Introduction

Overview

Talend Data Integration Advanced 6.0 is an introduction to Talend Data Integration, focusing on the additional features and capabilities of the Enterprise product as compared to Talend Open Studio for Data Integration.

Purpose

One of the primary differences between Talend Data Integration and Talend Open Studio for Data Integration is the addition of distributed, collaborative development features as well as change data capture capabilities. While the Talend Open Studio product is primarily designed for a single developer working on a single computer, the Talend Data Integration product provides a distributed architecture that allows multiple developers to share resources and combine efforts on a single project. This training module introduces you to the most significant of those differences.

Intended Audience

This training module is for anyone who wants to become familiar with the features of Talend Data Integration.

Prerequisites

Completion of the *Talend Studio Basics 6.0* class or equivalent experience.

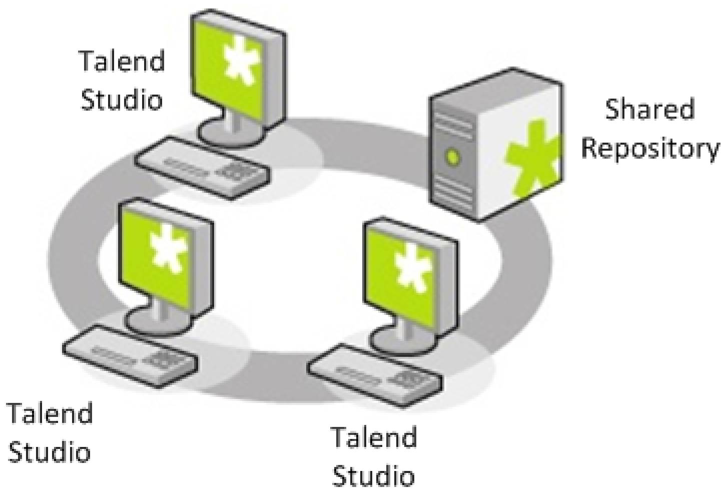
Next

This module begins by making a connection to a remote Repository.

Connecting to a Remote Repository

Overview

This lesson guides you through the process of starting Talend Studio with a connection to a remote, shared Repository. Multiple developers may work on a single Talend project, each running an individual instance of Talend Studio. By storing Repository information, including Job designs and metadata, in a central location, you avoid duplication and errors arising from out-of-sync assets.



In the controlled training environment you will connect to a Repository that is actually local, but the procedures are identical for a Repository located on a different system.

Objectives

After completing this lesson, you will be able to:

- Manage connections: local and remote
- Create a connection to a remote Talend Repository
- Start Talend Studio with a remote connection

Before You Begin

Be sure that you are working in an environment that contains the following:

- A properly installed copy of Talend Data Integration or any other licensed Talend Studio with the same functionalities
- A properly configured Talend Repository

The first step is to start the software.

Creating a Remote Connection

Overview

Talend Data Integration allows you to centralize Job designs and metadata in a common Repository to be accessed by multiple developers. In this exercise, you start Talend Studio and connect to a remote Repository prepared in advance.

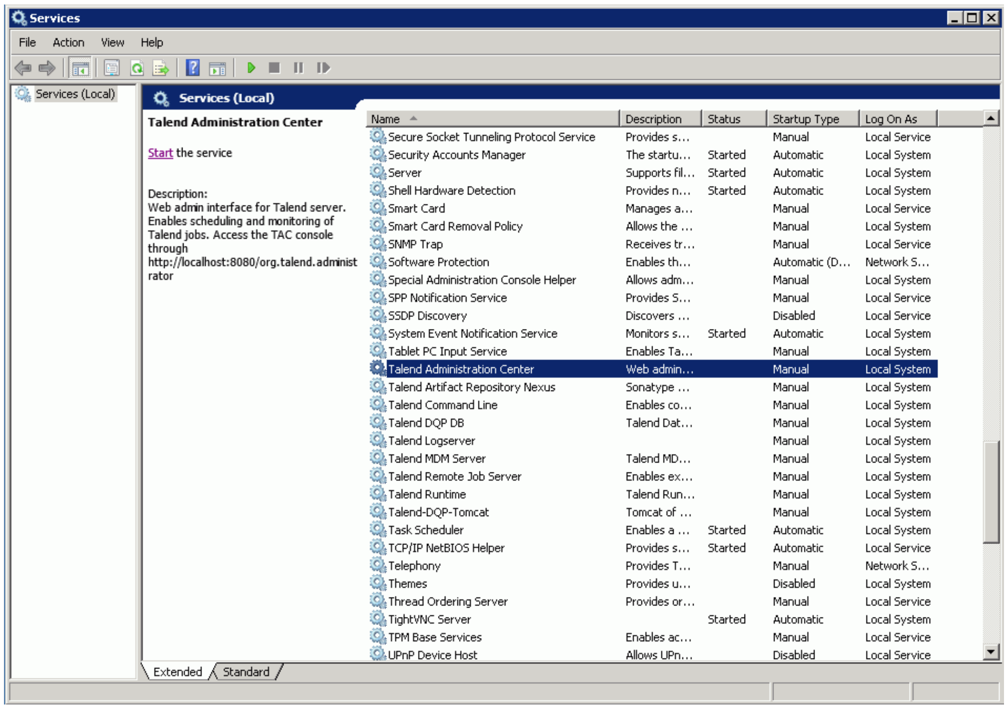
Note: the steps to create and configure a Repository are covered in the *Talend Data Integration Administration* training module.

Start services

To create a Remote Connection, you will need to start the **Talend Administration Center** first.

1. Click the **Services** icon in your Windows task bar:

2. In the **Services** list, click **Talend Administration Center**:



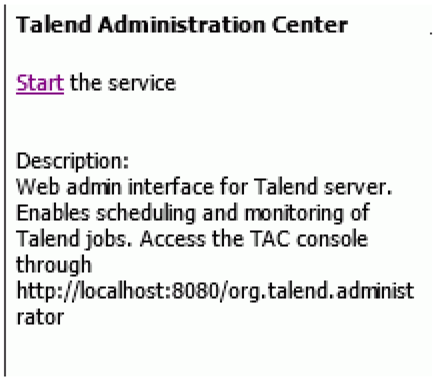
Talend Administration Center

Start the service

Description:
Web admin interface for Talend server. Enables scheduling and monitoring of Talend jobs. Access the TAC console through http://localhost:8080/org.talend.administrator

Name	Description	Status	Startup Type	Log On As
Secure Socket Tunneling Protocol Service	Provides s...		Manual	Local Service
Security Accounts Manager	The startu...	Started	Automatic	Local System
Server	Supports fil...	Started	Automatic	Local System
Shell Hardware Detection	Provides n...	Started	Automatic	Local System
Smart Card	Manages a...		Manual	Local Service
Smart Card Removal Policy	Allows the ...		Manual	Local System
SNMP Trap	Receives tr...		Manual	Local Service
Software Protection	Enables th...		Automatic (D...	Network S...
Special Administration Console Helper	Allows adm...		Manual	Local System
SPP Notification Service	Provides S...		Manual	Local Service
SSDP Discovery	Discovers ...		Disabled	Local Service
System Event Notification Service	Monitors s...	Started	Automatic	Local System
Tablet PC Input Service	Enables Ta...		Manual	Local System
Talend Administration Center	Web admin...		Manual	Local System
Talend Artifact Repository Nexus	Sonatype ...		Manual	Local System
Talend Command Line	Enables co...		Manual	Local System
Talend DQP DB	Talend Dat...		Manual	Local System
Talend Logserver			Manual	Local System
Talend MDM Server	Talend MD...		Manual	Local System
Talend Remote Job Server	Enables ex...		Manual	Local System
Talend Runtime	Talend Run...		Manual	Local System
Talend-DQP-Tomcat	Tomcat of ...		Manual	Local System
Task Scheduler	Enables a ...	Started	Automatic	Local System
TCP/IP NetBIOS Helper	Provides s...	Started	Automatic	Local Service
Telephony	Provides T...		Manual	Network S...
Themes	Provides u...		Disabled	Local System
Thread Ordering Server	Provides or...		Manual	Local Service
TightVNC Server		Started	Automatic	Local System
TPM Base Services	Enables ac...		Manual	Local Service
UPnP Device Host	Allows UPn...		Disabled	Local Service

3. Click the **Start** link:



Talend Administration Center

Start the service

Description:
Web admin interface for Talend server. Enables scheduling and monitoring of Talend jobs. Access the TAC console through http://localhost:8080/org.talend.administrator

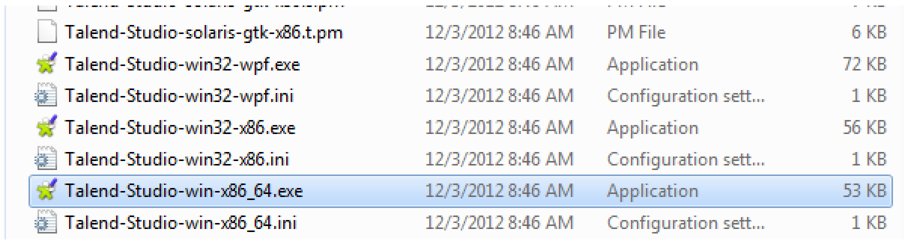
Now the Talend Administration Center is started. Next, you will create a remote connection.

Import the license

The license is not available yet in the Studio. So, before being able to connect to your project, you need to import the license in the studio as it is the first time you open the Studio.

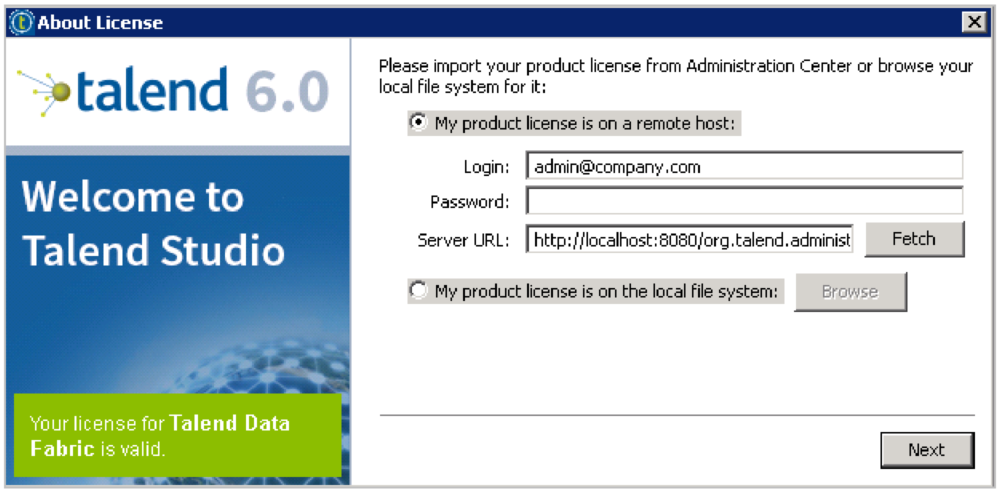
The license has already been uploaded for you in the Talend Administration Console.

1. Locate the file **Talend-Studio-win-x86_64.exe** under the directory **C:\Talend\6.0.1\studio\**.
2. Double-click the executable to start the application:



Talend-Studio-solaris-gtk-x86.t.pm	12/3/2012 8:46 AM	PM File	6 KB
Talend-Studio-win32-wpf.exe	12/3/2012 8:46 AM	Application	72 KB
Talend-Studio-win32-wpf.ini	12/3/2012 8:46 AM	Configuration sett...	1 KB
Talend-Studio-win32-x86.exe	12/3/2012 8:46 AM	Application	56 KB
Talend-Studio-win32-x86.ini	12/3/2012 8:46 AM	Configuration sett...	1 KB
Talend-Studio-win-x86_64.exe	12/3/2012 8:46 AM	Application	53 KB
Talend-Studio-win-x86_64.ini	12/3/2012 8:46 AM	Configuration sett...	1 KB

3. If the license is not set yet, you will be asked to import it from your local files or from the Talend Administration Center:



Please import your product license from Administration Center or browse your local file system for it:

(●) My product license is on a remote host:

Login: admin@company.com

Password:

Server URL: http://localhost:8080/org.talend.administ

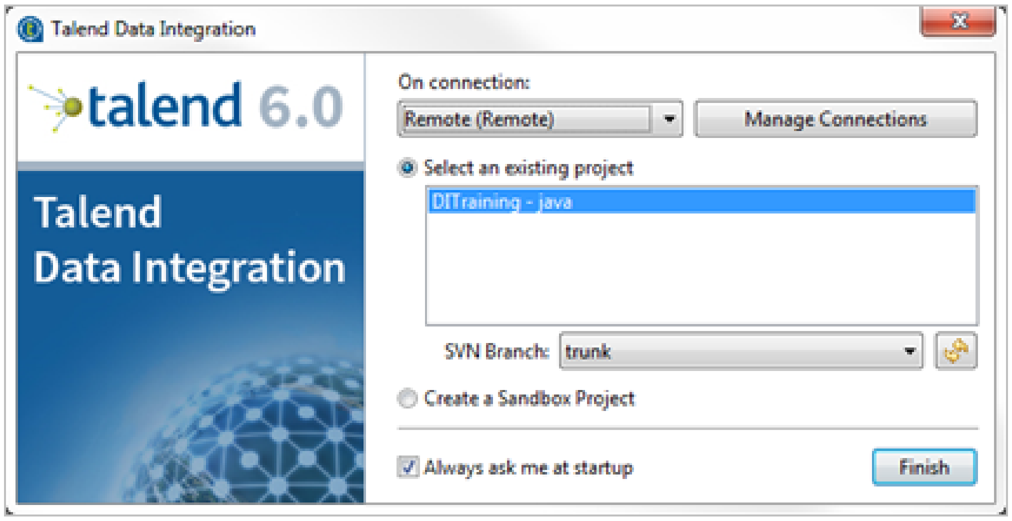
( ) My product license is on the local file system:

Select **My product license is on a remote host**.

4. In the **Password** field, enter "admin" and then click **Fetch.**
   This will import the license from the Talend Administration Center and associate it with your instance of the Studio.
5. Click **Next**.
   Your Studio will start.

Create the Connection

1. Once your Studio is started, you will have to create the Connection to your remote repository.



On connection: Remote (Remote)

(●) Select an existing project

DITraining - java

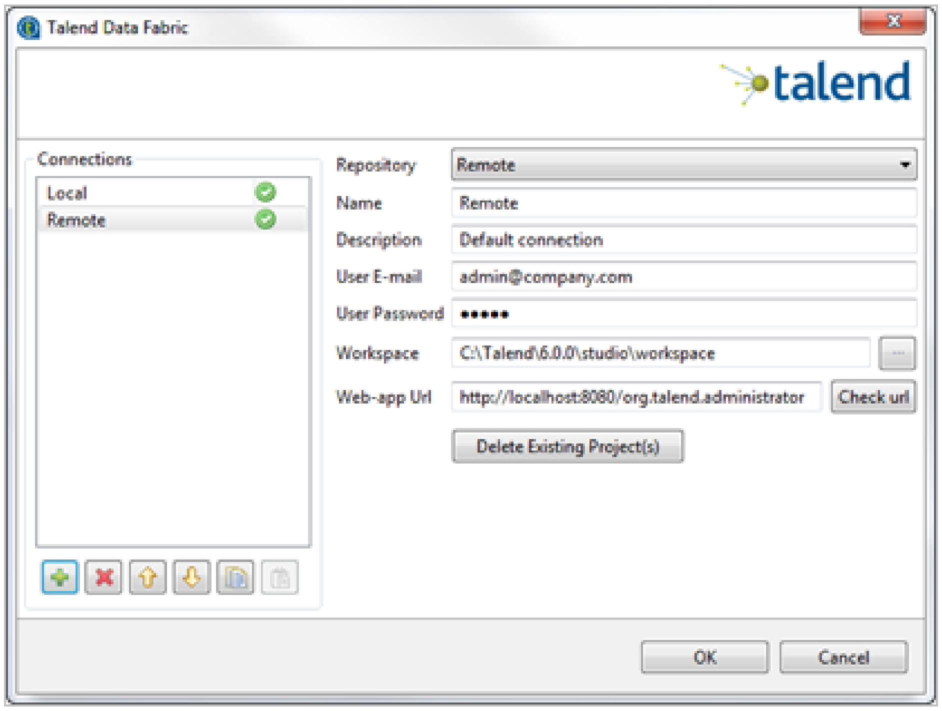
SVN Branch: trunk

( ) Create a Sandbox Project

[✓] Always ask me at startup

Note: You may have a different version of the Studio in your training environment. However, you will have exactly the same functionalities and will be able to perform all the labs.

2. Click **Manage Connections**:



Talend Data Fabric

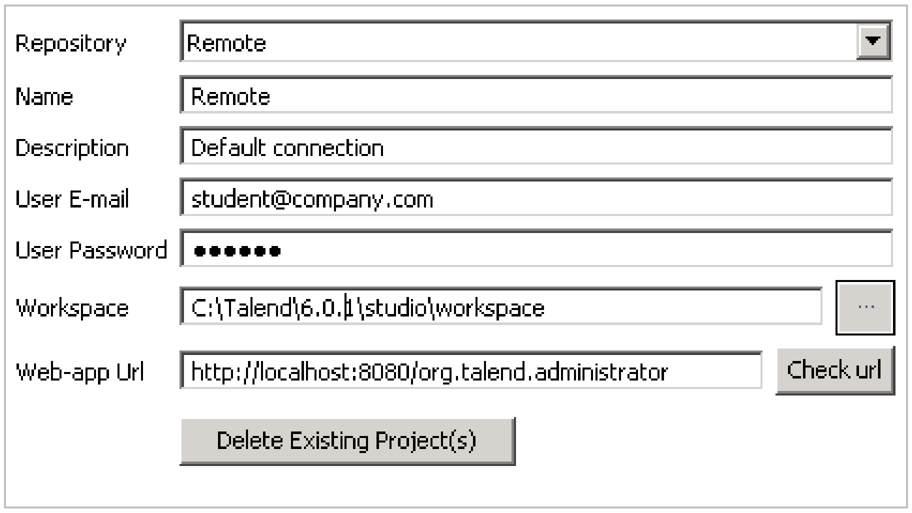
Connections
- Local
- Remote

Repository	Remote
Name	Remote
Description	Default connection
User E-mail	admin@company.com
User Password	●●●●●
Workspace	C:\Talend\6.0.0\studio\workspace
Web-app Url	http://localhost:8080/org.talend.administrator

Here you create and modify connections.

By default, a local and a remote connection are preconfigured.

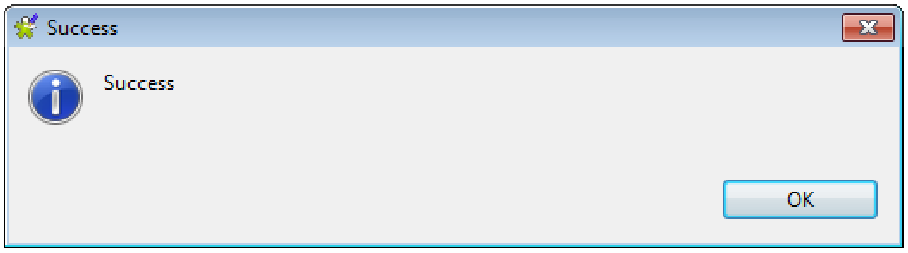
3. Enter "student@company.com" into the **User E-mail** box, "Talend" into the **User Password** box:



Repository	Remote
Name	Remote
Description	Default connection
User E-mail	student@company.com
User Password	●●●●●●
Workspace	C:\Talend\6.0.1\studio\workspace
Web-app Url	http://localhost:8080/org.talend.administrator

Note that these parameters apply to the specific Repository configured for the training environment (for example, the user "student@company.com" was created for you already). In your own environment, the parameters will of course be different. Also note that the Workspace has been configured under the default. In general, you should create a separate local workspace directory for each connection.

4. Click **Check url** to verify the parameters:



Success

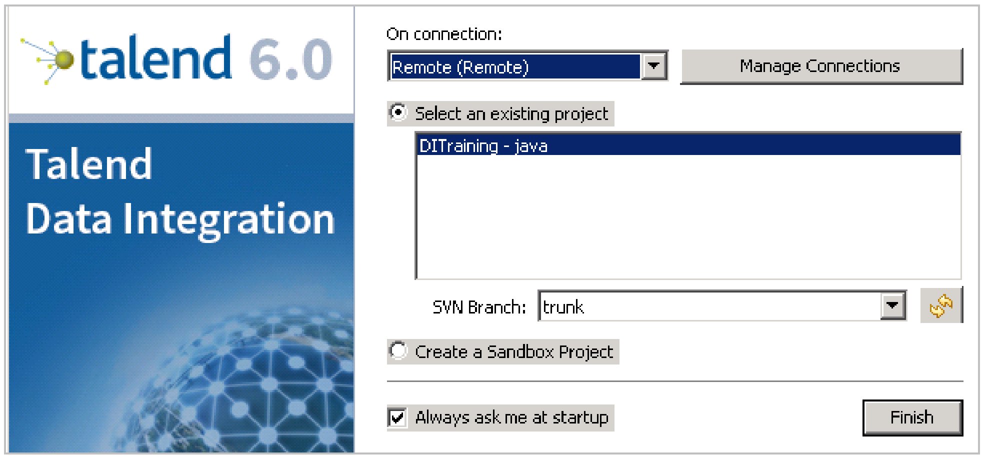
Success

If you receive an error message instead, carefully check the parameter values and then make any corrections.

5. Click **OK** in the success window and the click **OK** in the connections window.

Connect

1. Click **Remote** in the **Connection** list:



On connection: Remote (Remote)

(●) Select an existing project

DITraining - java

SVN Branch: trunk

( ) Create a Sandbox Project

☑ Always ask me at startup

This is the connection you just created. Notice the values in the **Project** and **SVN Branch** lists. This project, named **DITraining**, was configured in the trunk of the repository for the training environment. Again, the values in your environment will be different, and the steps to create a project in a central Repository are covered in the *Talend Data Integration Administration* training module.

2. Click **Finish**.
3. Click **Start Now** in the Talend Studio window.

You have now completed this lesson. You can recap your experience with the wrap-up.

Wrap-Up

Recap

In this lesson, you created a new connection that allowed you to start Talend Studio accessing an existing central Repository and opening a project within that Repository.

Further Reading

For more information about how to set up and configure a central Repository, consider the training module *Talend Data Integration Administration*. You can also find information in the *Talend Administration Console User Guide*.

Using SVN Branches

Overview

As your Talend projects move into production, you may find that you need to maintain the existing code while adding features for a new version of the project. Talend Data Integration uses the SVN source control system to help you maintain multiple versions of your Talend projects. In this lesson, you will examine another branch, copy a Job from one branch to another, and compare Jobs between branches. Note that the branch has already been created for you, because in a production environment it is likely that branching would be an administrator's function.

Objectives

After completing this tutorial, you will be able to:

- Switch between SVN branches in Talend Studio
- Copy a Job from one branch to another
- Compare the differences between two versions of the same Job

Before You Begin

Be sure that you are working in an environment that contains the following:

- A properly installed copy of Talend Data Integration
- The Jobs created in earlier lessons of this module
- A second SVN branch

The first step is to copy a Job to a branch.

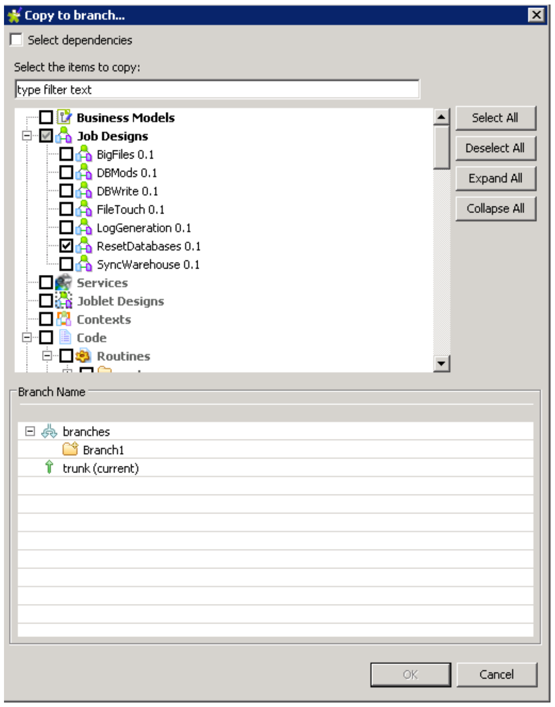
Copying a Job to a Branch

Overview

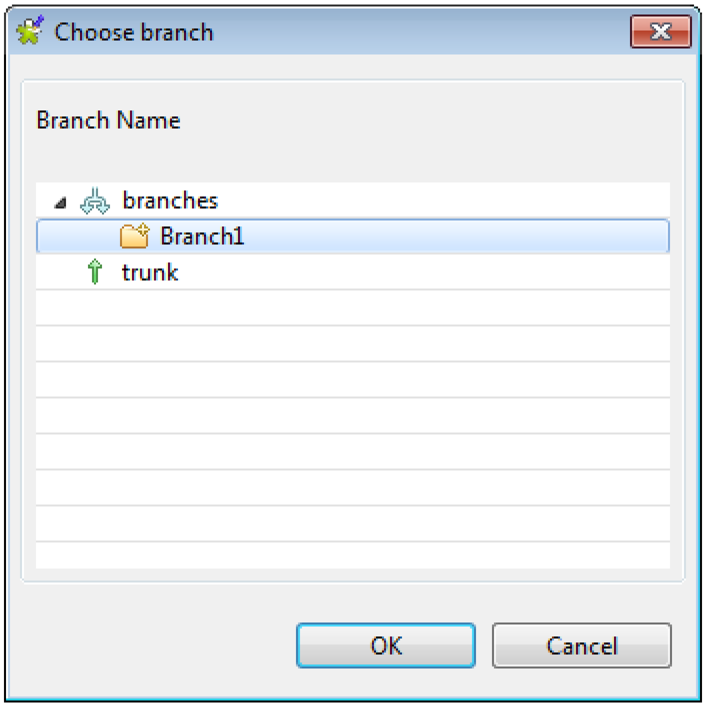
When an administrator creates a new SVN branch for a project, all of the existing Jobs in the project are copied into the new branch. This is a typical scenario for when a project team is ready to start modifying an existing version for improvements. However, it is also possible to create a Job in a sandbox branch and then copy it to the trunk when it is ready, or copy it to another branch altogether. Here, you will explore working with a branch created for you already.

Copy to Branch

1. Import the **ResetDatabases** Job from the archive **Job Designs.zip**, in the **C:\Solutions** folder.
2. Right-click the **ResetDatabases** Job in the **Repository** and then click **Copy To Branch**:

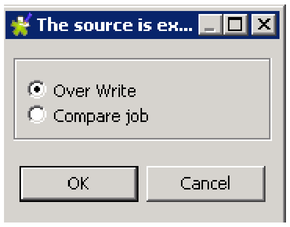


3. Click **Branch1** and then click **OK**:



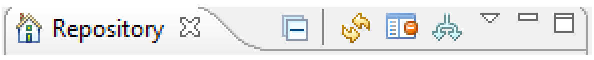
Branches were already created for you.

4. If the following message appears, select **Over Write** and then click **OK:**

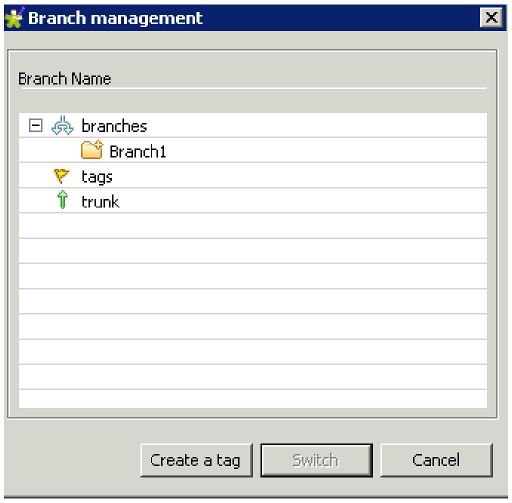


Switch Branches

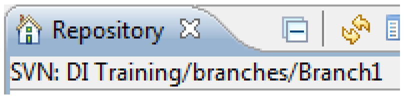
1. Wait a few seconds for the copy to complete, and then click the **Branch Management** button at the top of the **Repository**, shaped like a fork:



The **Branch Management** window appears:



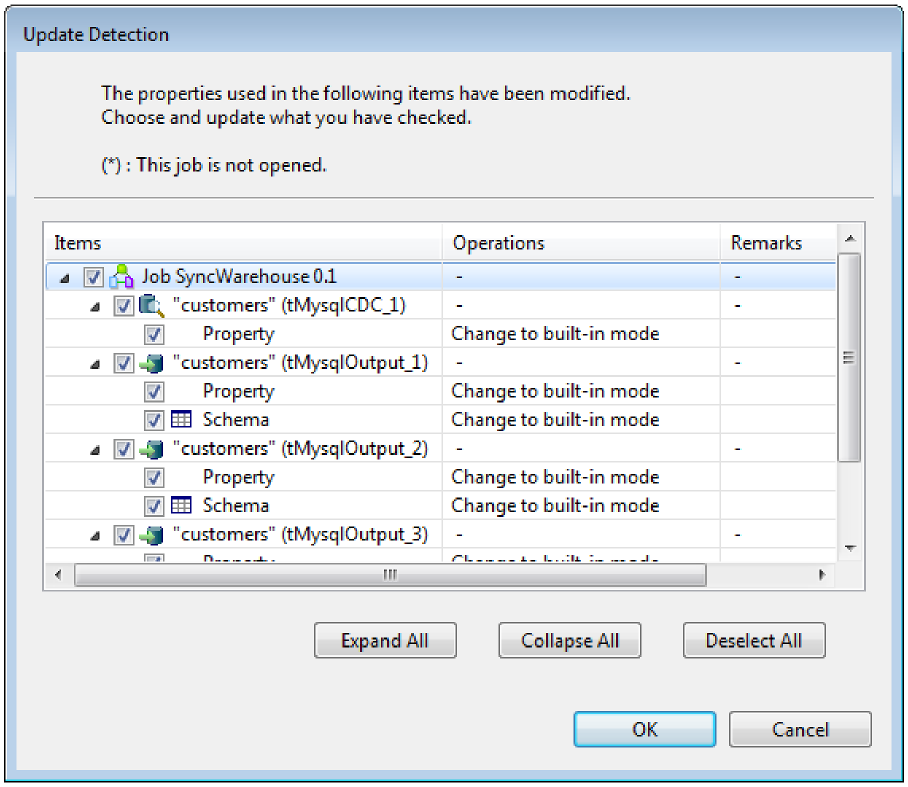
2. Click **Branch1** and then click **Switch**. Note that the label above the **Repository** has changed to reflect the SVN branch:



Note: If for some reason the branch remains at **trunk** instead of switching to **Branch1**, follow the steps in Resetting the Branch before continuing.

3. Double-click the **ResetDatabases** Job to open it.

If you copy a Job that uses metadata, a window may appear indicating that not all the metadata dependencies were copied into the branch:



Update Detection

The properties used in the following items have been modified.
Choose and update what you have checked.

(*) : This job is not opened.

Items	Operations	Remarks
☑ Job SyncWarehouse 0.1	-	-
☑ "customers" (tMysqlCDC_1)	-	-
☑ Property	Change to built-in mode	
☑ "customers" (tMysqlOutput_1)	-	-
☑ Property	Change to built-in mode	
☑ Schema	Change to built-in mode	
☑ "customers" (tMysqlOutput_2)	-	-
☑ Property	Change to built-in mode	
☑ Schema	Change to built-in mode	
☑ "customers" (tMysqlOutput_3)	-	-

If you want to ensure that all dependencies are copied, you may need to export a Job and then import it into the branch rather than simply copying it.

4. Add a **tLogRow** component and then connect it with one of the **customers** component using a **Main** row.
5. Save the changed Job, and then switch back to the trunk using the **Branch Management** button. You now have the same Job in both branches, but with slight differences.

Now you can use Talend Studio to compare the two Jobs and display the differences.

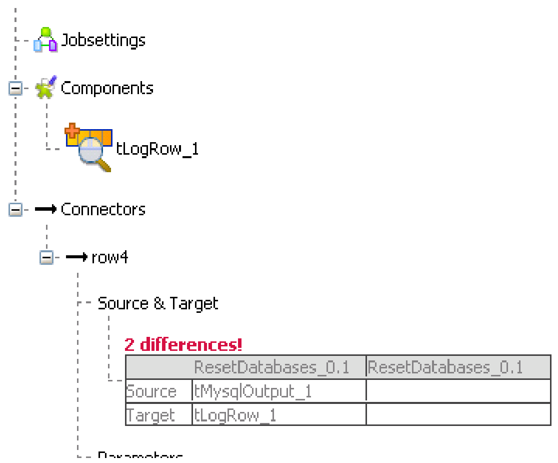
Comparing Jobs

Overview

With a Job in both the trunk and the branch, you can now compare the Jobs to determine their differences.

Compare Trunk to Branch

1. Right-click **ResetDatabases** in the **Repository** and then click **Compare Job**. The **Compare Result** view opens.
2. Click **/branches/Branch1** in **Another Branch**, click the button marked with an ellipsis next to **Another Job** and then click **ResetDatabases**, and then click **0.1** in the **Another version** list.
3. Click the **Compare** button on the right. Both Jobs open side-by-side in the design workspace and the **Compare Result** view shows a list of differences:



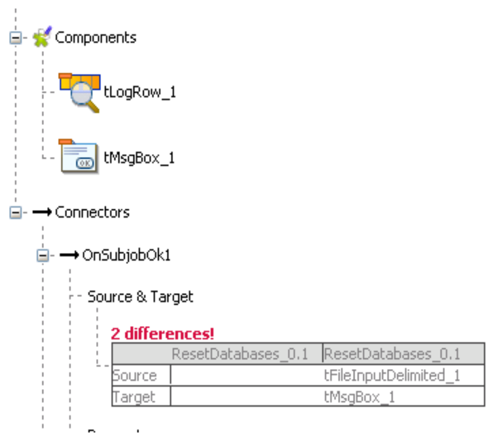
	ResetDatabases_0.1	ResetDatabases_0.1
Source	tMysqlOutput_1	
Target	tLogRow_1	

4. Examine the information shown in the **Compare Result** view.
5. Close both Jobs and the **Compare Result** view.

Compare Branch to Trunk

1. Switch to **Branch1**.
2. Open the **ResetDatabases** Job and make several changes, such as the following:
   - Delete the tLogRow component (the update).
   - Add a **tMsgBox** component below the **tFileInputDelimited** component and connect it with an **OnSubjobOK** trigger.
3. Save the Job.

4. Right-click the name of the Job in the **Repository**, click **Compare Job**, and then configure the **Compare Result** view to compare this version to the trunk version.
5. Click **Compare**, and then examine the differences between Jobs in the **Compare Result** view:



2 differences!

	ResetDatabases_0.1	ResetDatabases_0.1
Source		tFileInputDelimited_1
Target		tMsgBox_1

With the additional changes, this view displays more significant information.

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

Wrap-Up

Recap

In this lesson, you copied a Job from the trunk to an existing branch, switched Talend Studio to use that branch, and then compared Jobs between the trunk and a branch.

Further Reading

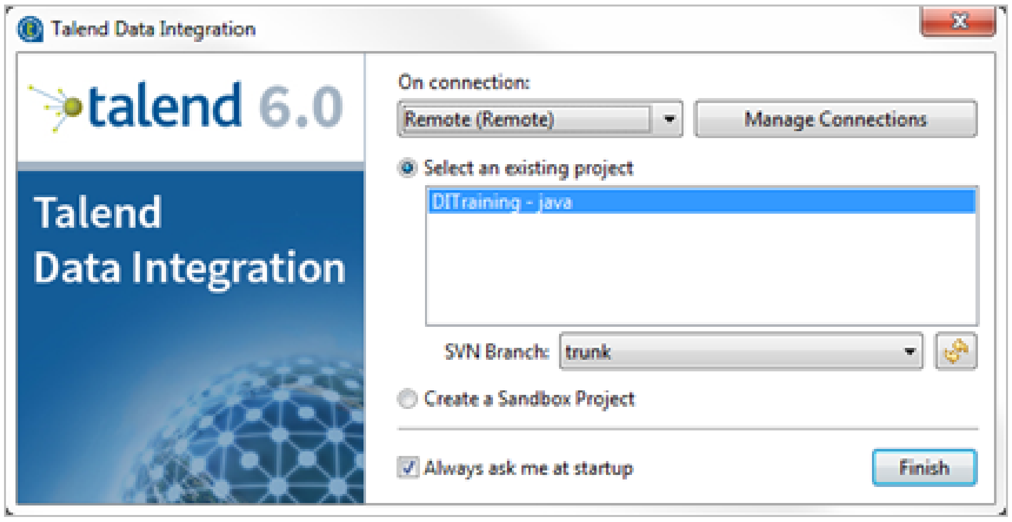
For more information about topics covered in this lesson, see the *Talend Data Integration Studio User Guide*. For more information about how to set up and configure SVN branches, consider the training module *Talend Data Integration Administration*. You can also find information in the *Talend Administration Center User Guide*.

Resetting the Branch

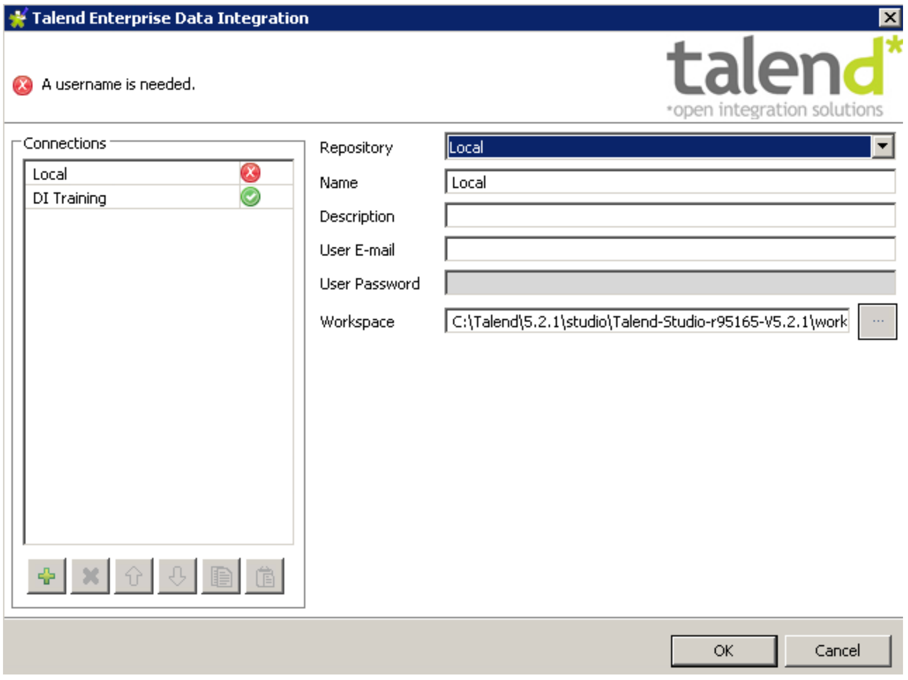
Overview

If you have difficulty switching branches, the most likely cause is that the Job copy did not complete correctly and a local copy of the project was created. Talend Studio does not permit both a local and remote copy of the same project. To recover, follow these steps.

1. From the **File** menu, click **Switch Project or Workspace**. Talend Studio restarts:



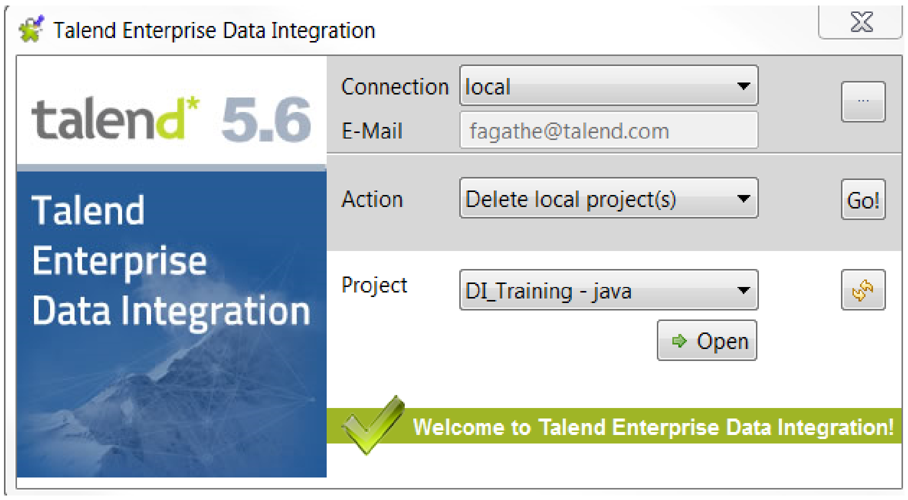
2. Click **Manage Connections**:


Talend Enterprise Data Integration
talend
open integration solutions
Connections
Local
DI Training
OK
Cancel

A username is needed.

Repository: Local
Name: Local
Description:
User E-mail:
User Password:
Workspace: C:\Talend\5.2.1\studio\Talend-Studio-r95165-V5.2.1\work

3. Enter an e-mail address into the **User E-mail** box, and then click **OK**. The start window reappears.
4. Click **Local** in the **Connection** list and then **Delete local project(s)** in the **Action** list:



Talend Enterprise Data Integration

Connection: local

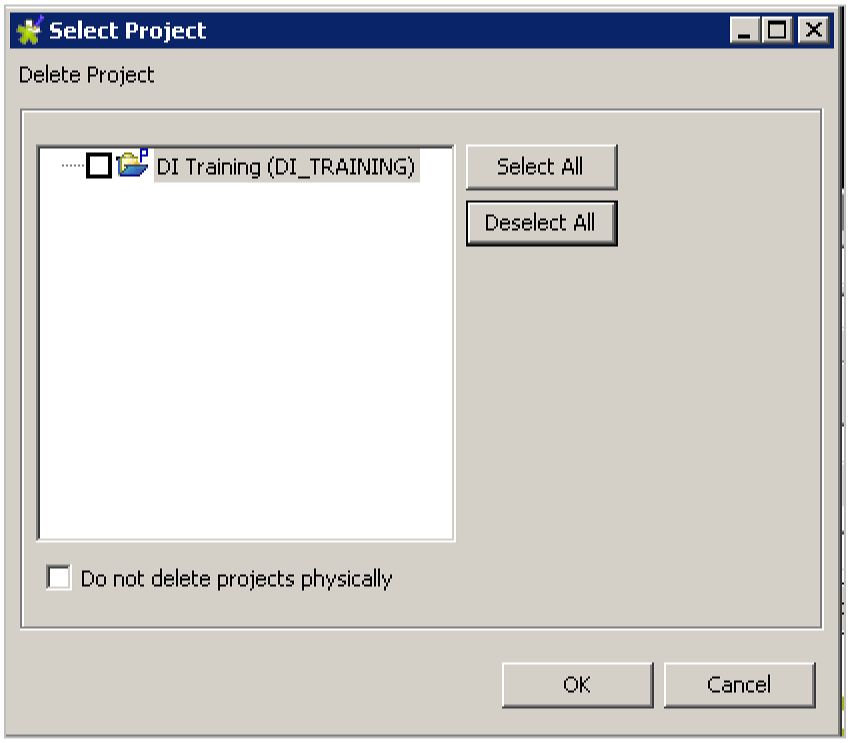
E-Mail: fagathe@talend.com

Action: Delete local project(s)

Project: DI_Training - java

talend 5.6
Talend Enterprise Data Integration
Go!
Open
Welcome to Talend Enterprise Data Integration!

5. Click **Go**:



Select Project

Delete Project

DI Training (DI_TRAINING)

Do not delete projects physically

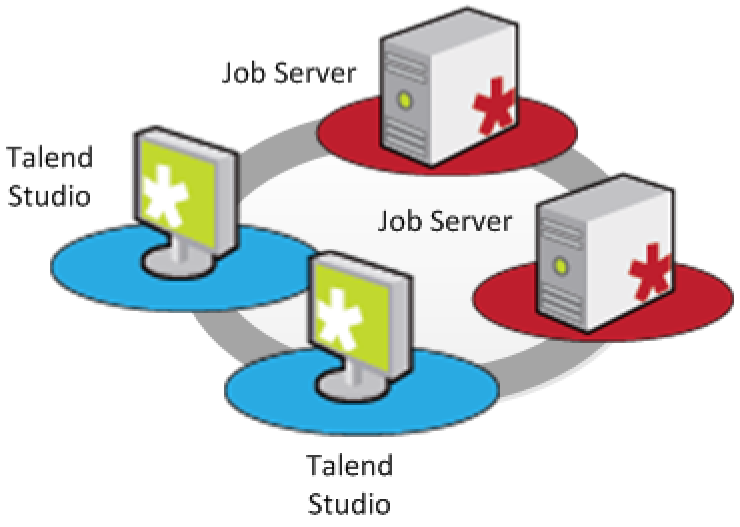
6. Click **Select All** followed by **OK**. The Start window appears again. The local copy of the Project has been deleted and you should be able to continue.
7. Click **DI Training** in the **Connection** list and then click **Open**. Once Talend Studio

has started, return to Copying a Job to a Branch and continue from the step where you switch to **Branch1**.

Running a Job Remotely

Overview

Frequently the computer you use to create a Talend Job is not the computer that will run the Job in production. With Talend Data Integration, your administrator can install Talend Job Server software on different computers without installing any of the other applications:



In this lesson, you will create a simple Job and then run the Job on a remote Job Server from your local Talend Studio. Keep in mind that in the controlled training environment, the "remote" Job Server is actually running locally, but the process is exactly the same.

Objectives

After completing this tutorial, you will be able to:

- Configure Talend Studio to identify remote Job Servers
- Run a Job from Talend Studio on a remote Job Server

Before You Begin

Be sure you are working in an environment that contains the following:

- A properly installed copy of Talend Data Integration
- A properly installed and running Talend Job Server

The first step is to create a sample Job.

Creating and Running a Job Remotely

Overview

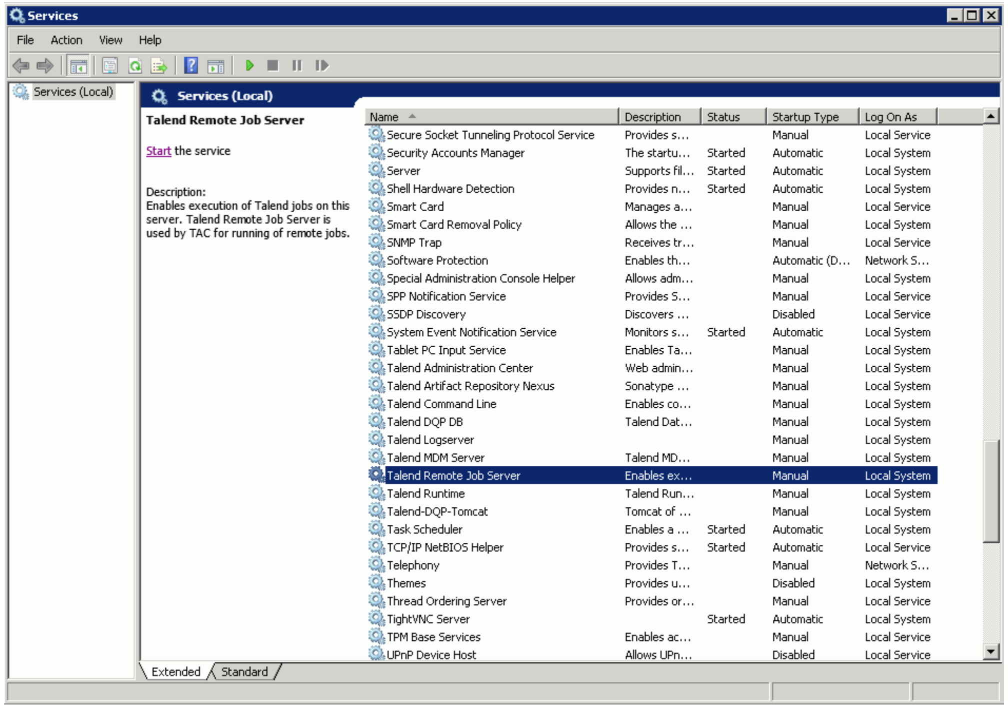
In this lesson, you are creating a Job and then running it not in Talend Studio, but on a Talend Job Server. The Job itself is trivial, because the point of the exercise is the remote execution.

Start services

To run a Job remotely, you will need to start the **Talend Remote Job Server** first.

1. Click the Services icon in your windows task bar:

2. In the **Services** list, click **Talend Remote Job Server**:



Services (Local)

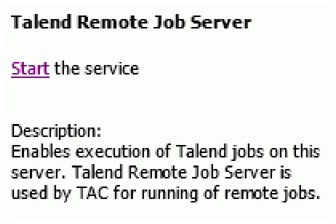
Talend Remote Job Server

Start the service

Description:
Enables execution of Talend jobs on this server. Talend Remote Job Server is used by TAC for running of remote jobs.

Name	Description	Status	Startup Type	Log On As
Secure Socket Tunneling Protocol Service	Provides s...		Manual	Local Service
Security Accounts Manager	The startu...	Started	Automatic	Local System
Server	Supports fil...	Started	Automatic	Local System
Shell Hardware Detection	Provides n...	Started	Automatic	Local System
Smart Card	Manages a...		Manual	Local Service
Smart Card Removal Policy	Allows the ...		Manual	Local System
SNMP Trap	Receives tr...		Manual	Local Service
Software Protection	Enables th...		Automatic (D...	Network S...
Special Administration Console Helper	Allows adm...		Manual	Local System
SPP Notification Service	Provides S...		Manual	Local Service
SSDP Discovery	Discovers ...		Disabled	Local Service
System Event Notification Service	Monitors s...	Started	Automatic	Local System
Tablet PC Input Service	Enables Ta...		Manual	Local System
Talend Administration Center	Web admin...		Manual	Local System
Talend Artifact Repository Nexus	Sonatype ...		Manual	Local System
Talend Command Line	Enables co...		Manual	Local System
Talend DQP DB	Talend Dat...		Manual	Local System
Talend Logserver			Manual	Local System
Talend MDM Server	Talend MD...		Manual	Local System
Talend Remote Job Server	Enables ex...		Manual	Local System
Talend Runtime	Talend Run...		Manual	Local System
Talend-DQP-Tomcat	Tomcat of ...		Manual	Local System
Task Scheduler	Enables a ...	Started	Automatic	Local System
TCP/IP NetBIOS Helper	Provides s...	Started	Automatic	Local Service
Telephony	Provides T...		Manual	Network S...
Themes	Provides u...		Disabled	Local System
Thread Ordering Server	Provides or...		Manual	Local Service
TightVNC Server		Started	Automatic	Local System
TPM Base Services	Enables ac...		Manual	Local Service
UPnP Device Host	Allows UPn...		Disabled	Local Service

3. Click the **Start** link:



Talend Remote Job Server

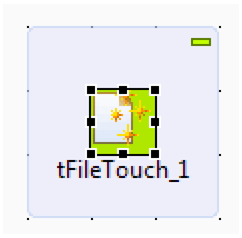
Start the service

Description:
Enables execution of Talend jobs on this server. Talend Remote Job Server is used by TAC for running of remote jobs.

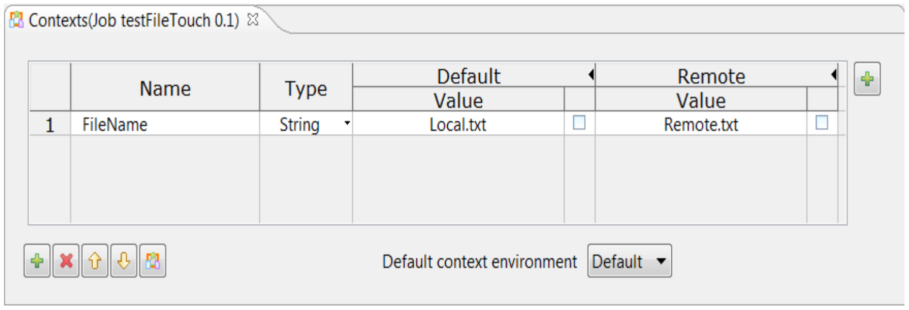
Now the Talend Remote Job Server is started. Next, you will create a Job to run it remotely.

Create a Job

1. Switch to the **Integration** Perspective, if necessary, and then create a new Job, calling it "FileTouch".
2. Add a **tFileTouch** component to the design workspace:

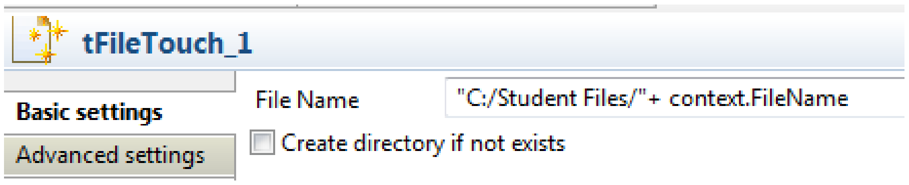
   

   This component creates an empty file or updates the timestamp of an existing file.
3. In the **Contexts** view, click the button marked with a plus sign below the table to create a new context variable named "FileName" (without quotes).
4. In the **Value** column, set the Default value to "Local.txt".
5. Click the green plus sign at the right side of the table to add a second context named "Remote" (click **New**).
6. Set the value of the variable in the **Remote** context to "Remote.txt":

	Name	Type	Default Value		Remote Value	
1	FileName	String	Local.txt	☐	Remote.txt	☐

7. Configure the **File Name** parameter of the **tFileTouch** component to be the value of the context variable under the path "C:/StudentFiles":

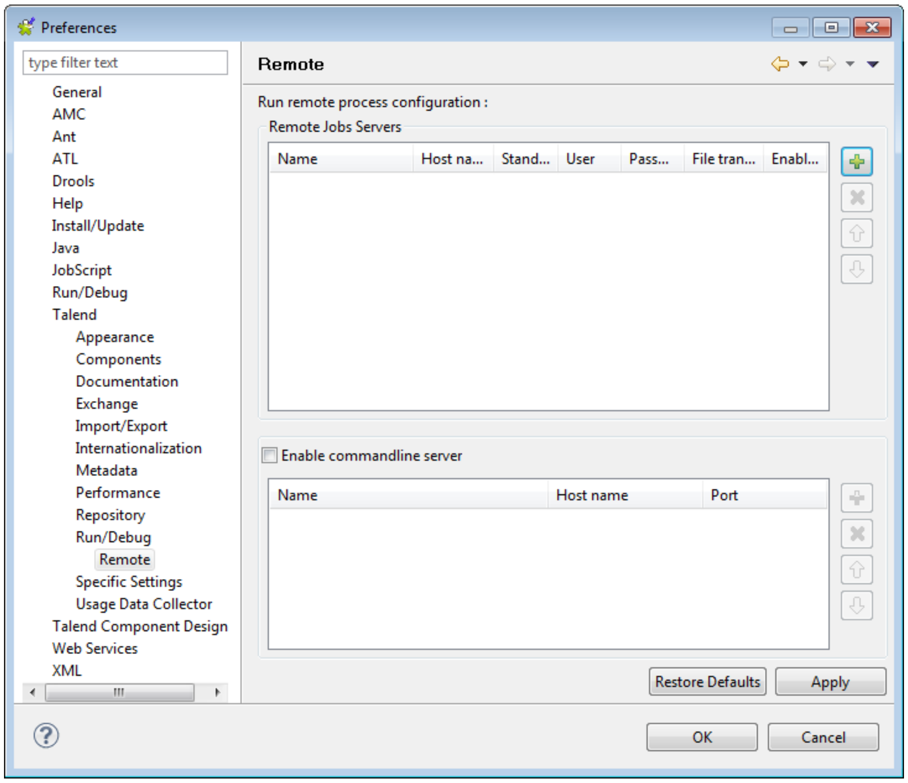
   File Name: "C:/Student Files/"+ context.FileName

   ☐ Create directory if not exists

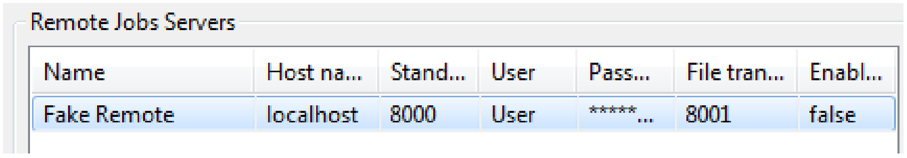
   Remember that you can access context variables by pressing Ctrl + Space.
8. Run the Job using the default context and then locate the new file.

Configure Job Server

1. From the Talend Studio **Window** menu, click **Preferences**, followed by **Talend**, **Run/Debug**, and then **Remote**:



2. Click the Add button marked with a green plus sign.
3. Change the **Host name** field to "localhost" and the **Name** to "Fake Remote", leaving the other values as-is:


Remote Jobs Servers

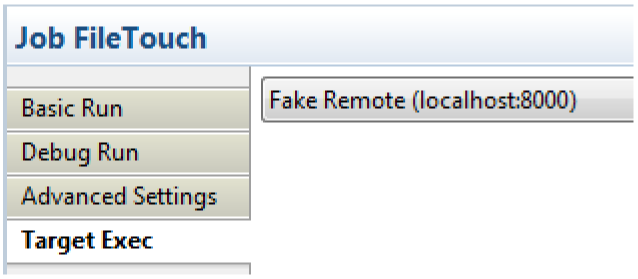
Name	Host na...	Stand...	User	Pass...	File tran...	Enabl...
Fake Remote	localhost	8000	User	*****...	8001	false

The remote server must have a Talend Job Server running. The standard installation process for Talend Data Integration also installs a Talend Job Server as a service on Windows. The training environment has a Talend Job Server running. In your environment, you would identify either your local computer or a remote computer with a running Job Server.

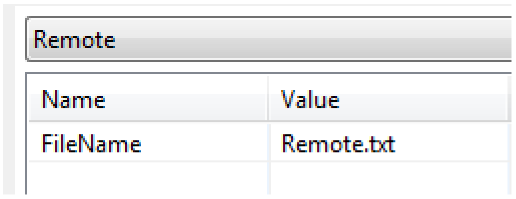
4. Click **OK**.

Run Job Remotely

1. Click the **Run** view.
2. Click the **Target Exec** tab, then click **Fake Remote** in the list:

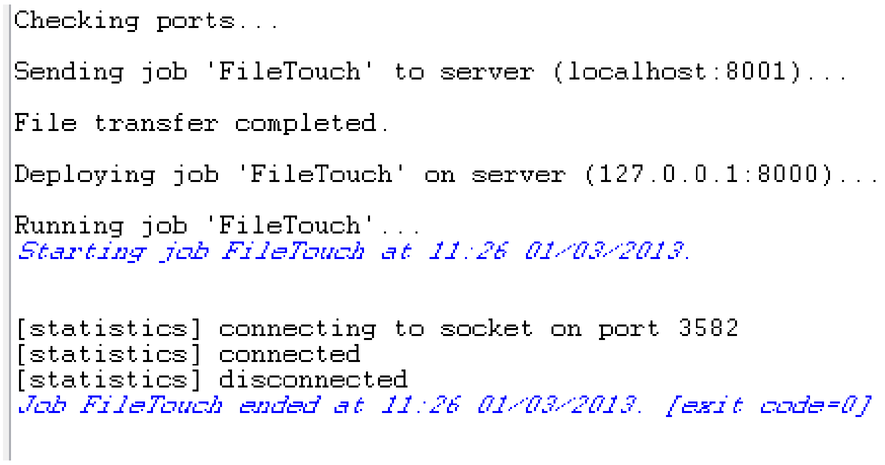


   This is the new remote Job server you just added.
3. Click the **Basic Run** tab.
4. Click **Remote** in the **Context** list:



Name	Value
FileName	Remote.txt

   Now the Job will run with the Remote value for the context variable you created earlier.
5. Click **Run** and then examine the console messages:



```
Checking ports...

Sending job 'FileTouch' to server (localhost:8001)...

File transfer completed.

Deploying job 'FileTouch' on server (127.0.0.1:8000)...

Running job 'FileTouch'...
Starting job FileTouch at 11:26 01/03/2013.


[statistics] connecting to socket on port 3582
[statistics] connected
[statistics] disconnected
Job FileTouch ended at 11:26 01/03/2013. [exit code=0]
```

   Note the messages about sending the Job and connecting to the remote server.
6. Locate the file created by the Job, named "Remote.txt". The name is different because of the change in context. When you run a Job remotely, you frequently need to use context variables to configure your Job to work properly on the remote

computer.

You have now completed this lesson and can move on to the Challenge.

Challenge

Overview

Complete this challenge to further explore the use of remote Job execution. See Solution for a possible solution to this exercise.

Second Remote Server

Modify the Job to add a second context variable that specifies the directory for the touched file. Make the directory value different for the default and remote contexts. Add a second remote Job Server and then run the Job on that server, specifying the remote context.

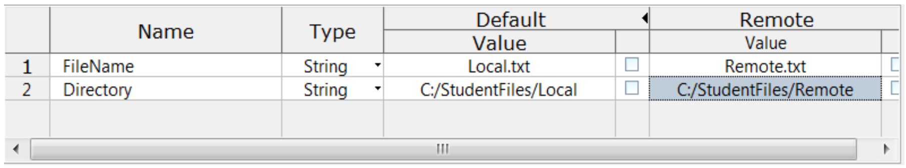
Solution

Overview

This is a possible solution to the Challenge. Note that your solution may differ and still be valid.

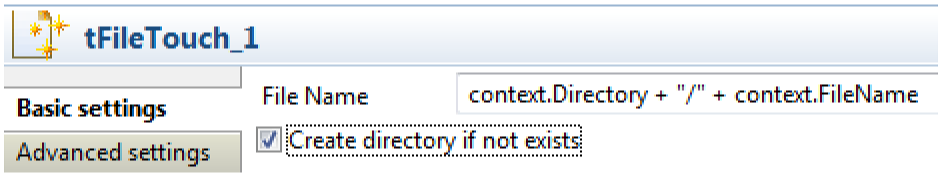
Second Remote Server

1. Add a second context variable named "Directory" to hold a value representing the directory containing the created file.
2. Set the default value to "C:/StudentFiles/Local", and the **Remote** context value to "C:/StudentFiles/Remote":



	Name	Type	Default Value		Remote Value	
1	FileName	String	Local.txt	☐	Remote.txt	☐
2	Directory	String	C:/StudentFiles/Local	☐	C:/StudentFiles/Remote	☐

3. Configure the component to use the new context variable and select **Create directory if not exists**:



tFileTouch_1

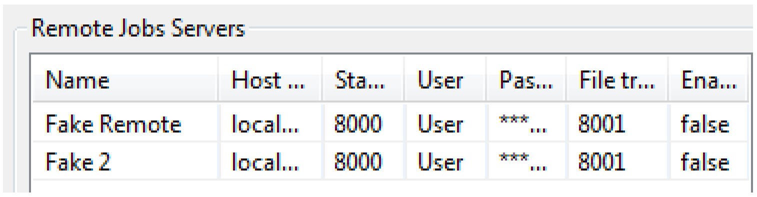
Basic settings

Advanced settings

File Name: context.Directory + "/" + context.FileName

☑ Create directory if not exists

4. Add a second remote Job server, naming it "Fake 2":

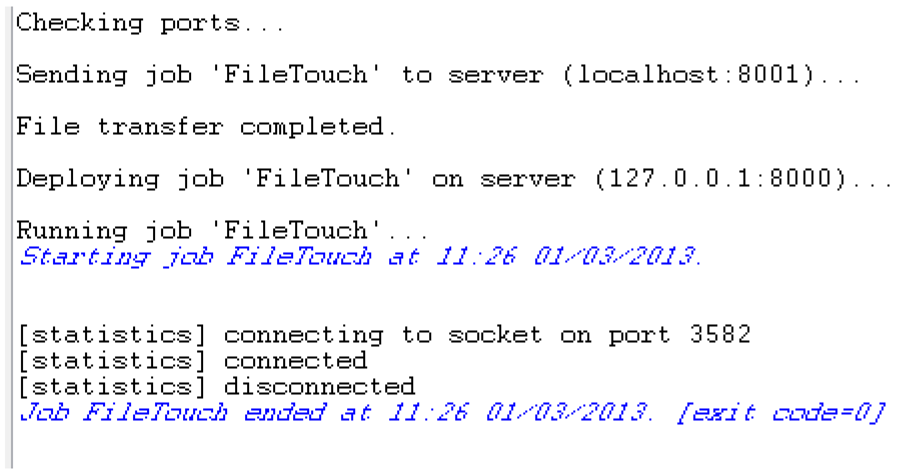


Remote Jobs Servers

Name	Host ...	Sta...	User	Pas...	File tr...	Ena...
Fake Remote	local...	8000	User	***...	8001	false
Fake 2	local...	8000	User	***...	8001	false

5. Run the Job on the remote server using the **Remote** context and then verify that

the file is created in the correct location:



```
Checking ports...

Sending job 'FileTouch' to server (localhost:8001)...

File transfer completed.

Deploying job 'FileTouch' on server (127.0.0.1:8000)...

Running job 'FileTouch'...
Starting job FileTouch at 11:26 01/03/2013.


[statistics] connecting to socket on port 3582
[statistics] connected
[statistics] disconnected
Job FileTouch ended at 11:26 01/03/2013. [exit code=0]
```

Wrap-Up

Recap

In this lesson, you explored how to identify and use remote Job Servers to execute Jobs from within Talend Studio.

Further Reading

For more information about topics covered in this lesson, see the *Talend Data Integration Installation and Upgrade Guide* and the *Talend Data Integration Administration*training module.

Monitoring Job Activity

Overview

The Talend Activity Monitoring Console (AMC) is an application that allows you to view statistics and log information about your Jobs. In this lesson, you will configure a Project and Talend Studio to capture information about your Jobs and then access AMC from Talend Studio to view that information. Note that you can also access AMC from the Talend Administration Console, a topic covered in the Administration training module.

Objectives

After completing this lesson, you will be able to:

- Configure a Talend project to capture statistics and logs
- Configure a Talend Job to capture statistics and logs
- Access the Talend Activity Monitoring Console from within Talend Studio
- List the kinds of information available in AMC

Before You Begin

Be sure you are working in an environment that contains the following:

- A properly installed copy of Talend Data Integration
- The supporting files for this lesson

The first step is to configure the project to capture statistics and logs.

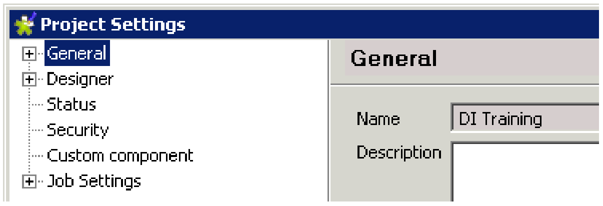
Configuring Statistics and Logging

Overview

The Activity Monitoring Console allows you to view historical information about Job execution. The AMC uses several database tables to store that information. If you want to maintain information for a particular Job, you first need to configure the Job and the project to store information in the AMC database.

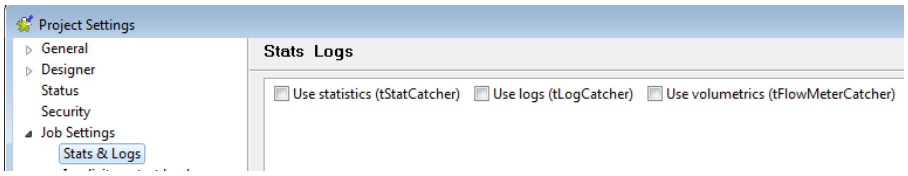
Configure Logging

1. On the **File** menu, click **Edit Project properties**:



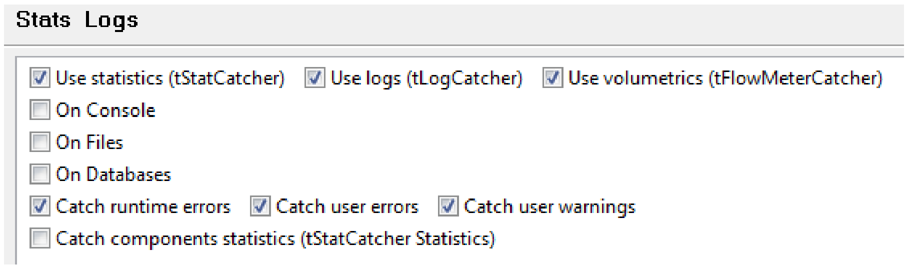
Name: DI Training

2. Expand **Job Settings** and then click **Stats & Logs**:



This is where you specify what information you want to capture about one or more Jobs in this project.

3. Select all three check boxes:

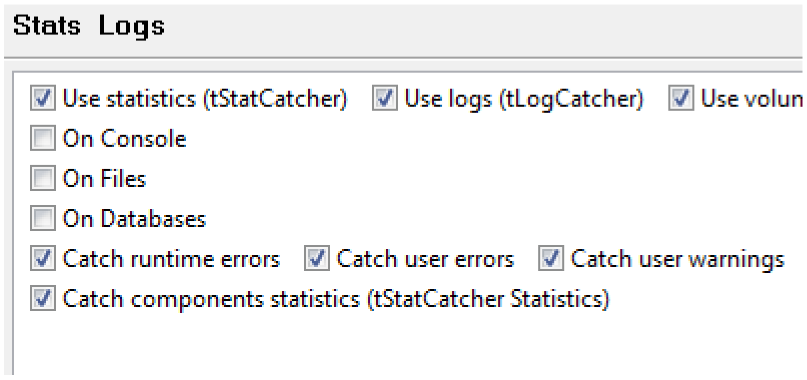


Stats Logs

☑ Use statistics (tStatCatcher) ☑ Use logs (tLogCatcher) ☑ Use volumetrics (tFlowMeterCatcher)
☐ On Console
☐ On Files
☐ On Databases
☑ Catch runtime errors ☑ Catch user errors ☑ Catch user warnings
☐ Catch components statistics (tStatCatcher Statistics)

By selecting all three, you indicate that you want to capture statistics, logs, and volume information.

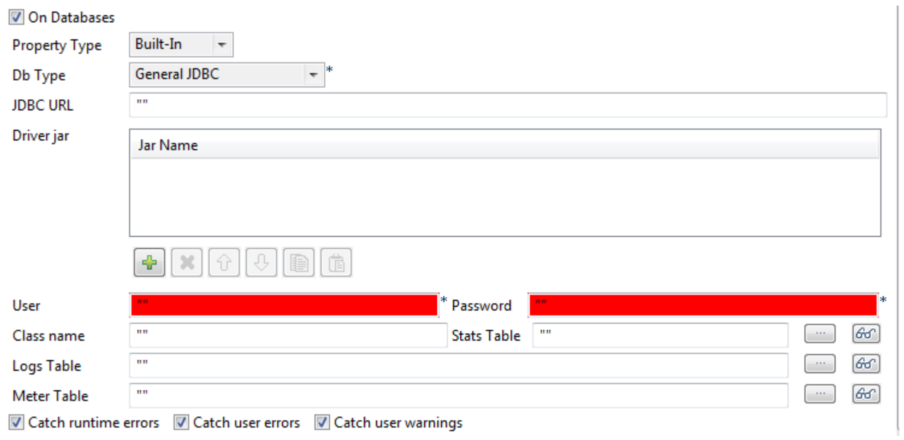
4. Select **Catch components statistics**:


Stats Logs

☑ Use statistics (tStatCatcher) ☑ Use logs (tLogCatcher) ☑ Use volun
☐ On Console
☐ On Files
☐ On Databases
☑ Catch runtime errors ☑ Catch user errors ☑ Catch user warnings
☑ Catch components statistics (tStatCatcher Statistics)

Now all events generated by Jobs will be captured, including statistics about individual components rather than just the Job as a whole.

5. Select **On Databases**:

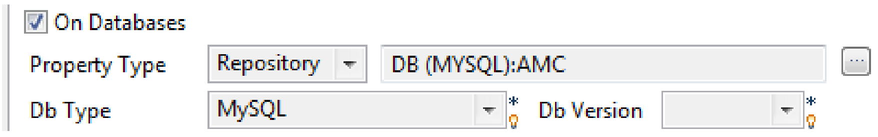

☑ On Databases

Property Type	Built-In		
Db Type	General JDBC		
JDBC URL	""		
Driver jar	Jar Name		
User	""	Password	""
Class name	""	Stats Table	""
Logs Table	""		
Meter Table	""		

☑ Catch runtime errors ☑ Catch user errors ☑ Catch user warnings

This tells Talend Studio to capture the information and store it in a database rather than in a file or display in on the Console.

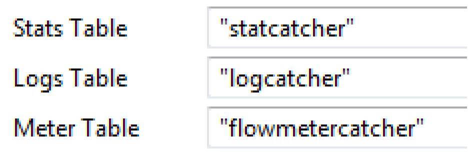
6. To specify the database tables, click **Repository** in the **Property Type** list, click the button marked with an ellipsis and then choose the **AMC** database connection:


☑ On Databases

Property Type	Repository	DB (MYSQL):AMC	
Db Type	MySQL	Db Version	

This database has already been configured in the training environment with the appropriate tables.

7. Use the buttons marked with an ellipsis next to the **Table** boxes to choose **statcatcher**, **logcatcher**, and **flowmetercatcher** in turn:



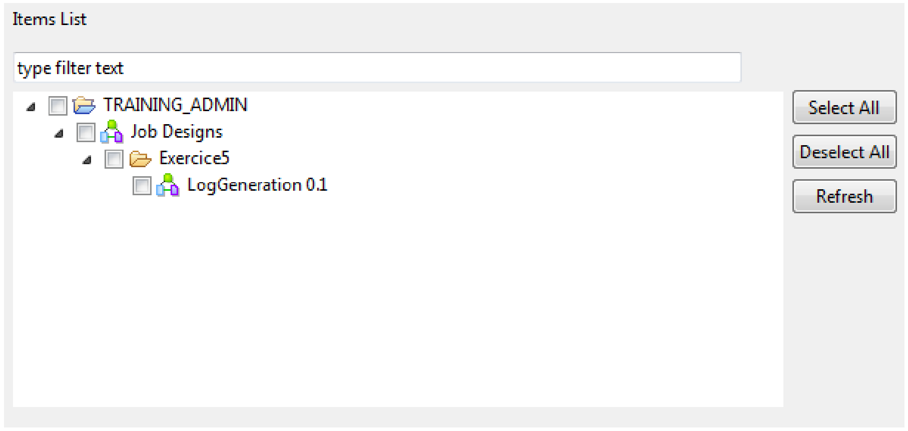
Stats Table	"statcatcher"
Logs Table	"logcatcher"
Meter Table	"flowmetercatcher"

These are the tables in the AMC database that will store the three different kinds of Job information.

8. Click **OK**.

Import Job

1. In the **Repository**, right-click **Job Designs** and then click **Import items**.
2. Click **Select archive file** and then click the **Browse** button.
3. Locate **LogGeneration.zip** under **C:\StudentFiles** and then click **Open**:



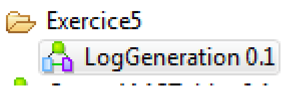
Items List

type filter text

TRAINING_ADMIN
Job Designs
Exercice5
LogGeneration 0.1

The archive file was created for you by using the Export items function of Talend Studio.

4. Click **Select All** and then click **Finish**. The Job appears in the **Repository**:



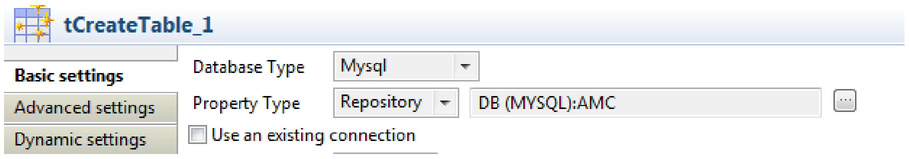
Exercice5
LogGeneration 0.1

5. Open the Job and, in the **UpdateDetection** window that appears, click **OK**.
6. Take a moment to examine the Job. This Job was designed specifically to generate a range of information that you can examine with the AMC. It issues a warning, creates a database table, generates random data, filters the data before writing it to a database, and then issues a final warning when the Job is complete.

Configure the Job

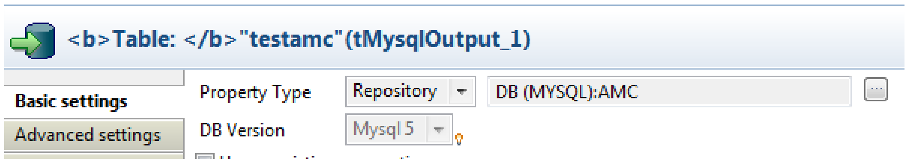
Before you can run this Job, you need to make a few modifications:

1. Double-click the **tCreateTable** component to open the Component view.
2. Change the **Property Type** from **Built-In** to **Repository**, and then use the button marked with an ellipsis to choose the **AMC** database connection:


tCreateTable_1

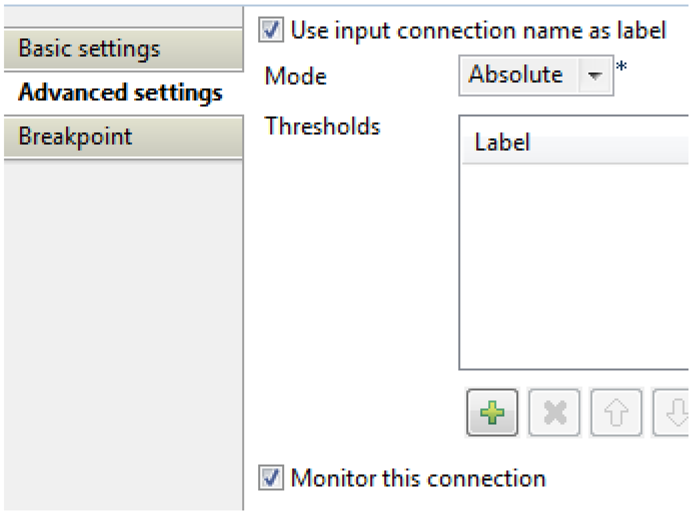
Database Type: Mysql
Property Type: Repository — DB (MYSQL):AMC
Use an existing connection

Make the same change for each of the three **tMysqlOutput** components:


<b>Table: </b>"testamc"(tMysqlOutput_1)

Property Type: Repository — DB (MYSQL):AMC
DB Version: Mysql 5

3. Click the **Advanced settings** tab for one of the **tMysqlOutput** components and note that the **tStatCatcherStatistics** check box is selected to collect information about this component.
4. Double-click one of the rows connecting the **tMap** component to one of the database components and then click the **Advanced settings** tab in the **Component** view:



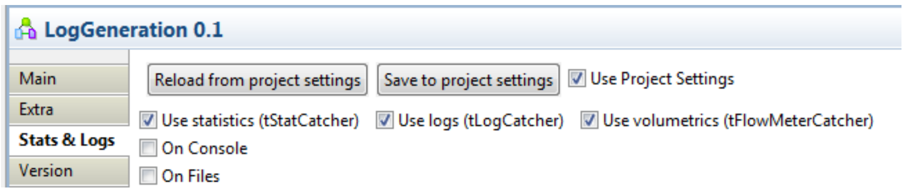
Use input connection name as label (checked)
Mode: Absolute
Thresholds: Label
Monitor this connection (checked)

Note the **Monitor** check box and the label **Monitored** in the design workspace. This indicates that the number of rows of data flowing through this connection is tracked when the Job runs, creating the flow meter information for the AMC.

Set Job Parameters

You enabled statistics and log monitoring for the project as a whole, but now you need to enable it for this specific Job:

1. Click the **Job** view and then click the **Stats & Logs** tab:



LogGeneration 0.1

☑ Use Project Settings
☑ Use statistics (tStatCatcher) ☑ Use logs (tLogCatcher) ☑ Use volumetrics (tFlowMeterCatcher)
☐ On Console
☐ On Files

2. Click the **Reload from project settings** button and then click **OK** in the warning window that appears.

   Now this Job uses the same tracking settings that you configured for the project, so that the AMC information will be stored in the three different database tables.
3. Run the Job at least five times to generate enough tracking information for you to examine.

Now that you have run a Job configured to track statistics and logging information, you can examine that information with the AMC.

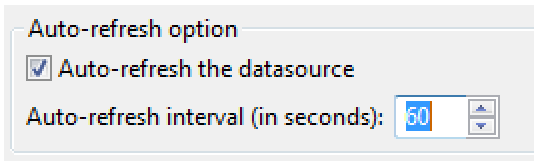
Using the Activity Monitoring Console

Overview

You've generated tracking information about a Job that is stored in the AMC database. Now you will configure Talend Studio so that you can access the AMC from that application. Note that you can also access the AMC from the Talend Administration Center, which is explored in the Talend Data Integration Administration training module.

Accessing the AMC

1. On the **Window** menu in Talend Studio, click **Preferences**.
2. Click **AMC** and then select **Auto-refresh the datasource**, setting the refresh interval to 60 seconds:

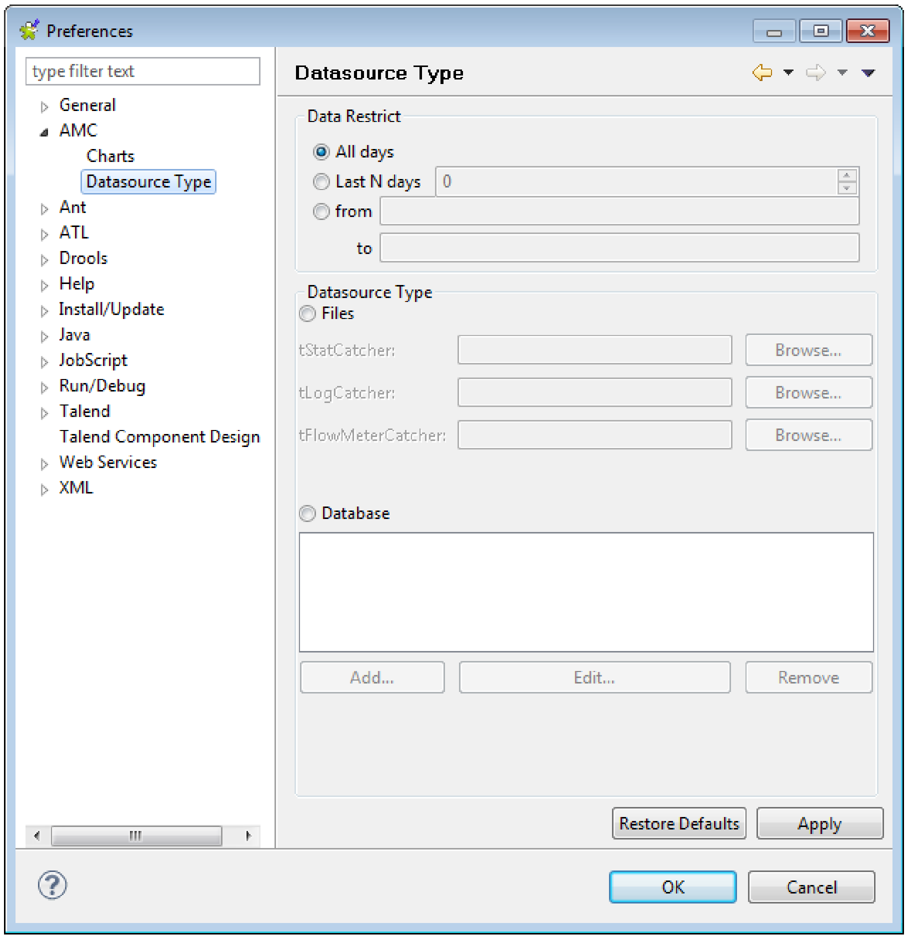
   

   Auto-refresh option

   ☑ Auto-refresh the datasource

   Auto-refresh interval (in seconds): 60

   Now the AMC will automatically check the database for new information once each minute.
3. Expand **AMC** and then click **Datasource Type**:



Datasource Type

Data Restrict

(●) All days

( ) Last N days 0

( ) from

to

Datasource Type

( ) Files

tStatCatcher: Browse...

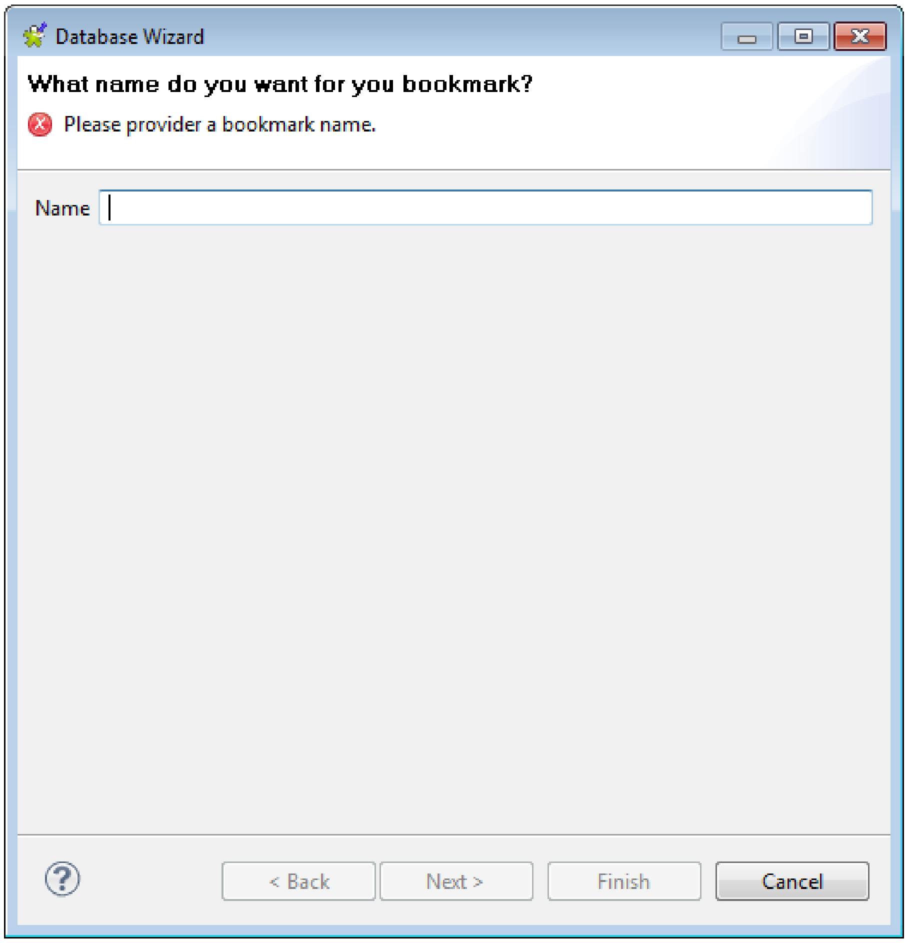
tLogCatcher: Browse...

tFlowMeterCatcher: Browse...

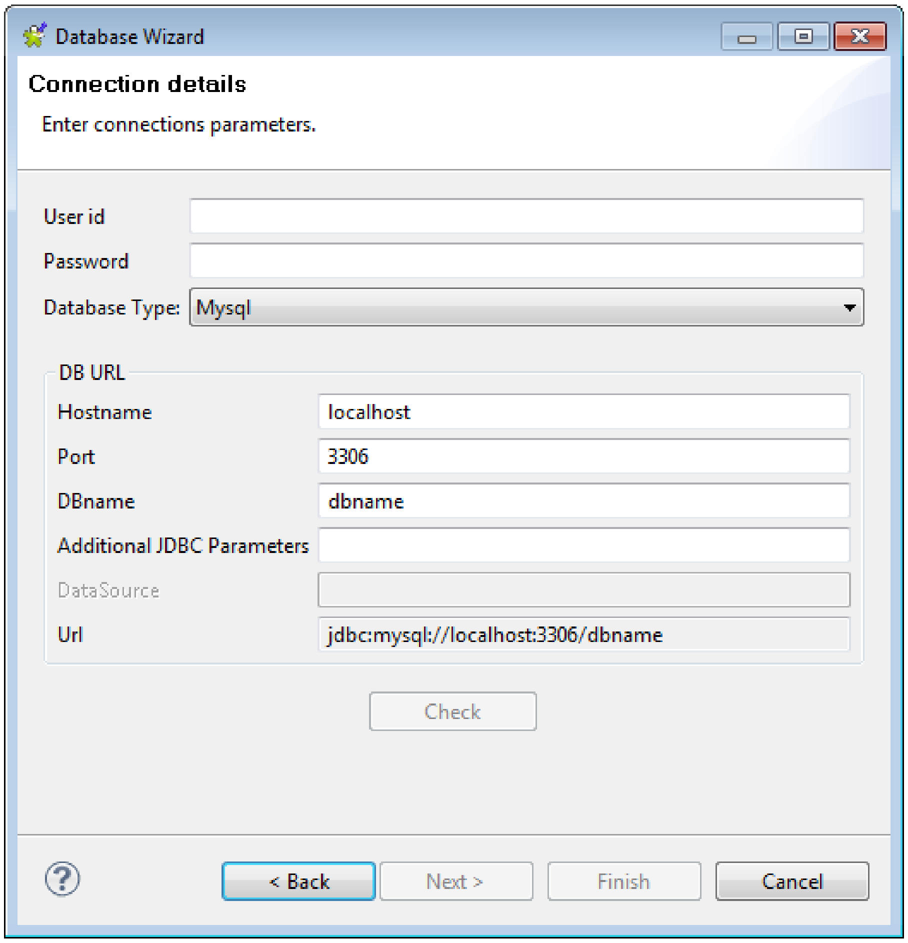
( ) Database

Add... Edit... Remove

4. Click **Database** under **Datasource Type** followed by **Add**:


Database Wizard
What name do you want for you bookmark?
Please provider a bookmark name.
Name
< Back
Next >
Finish
Cancel

5. Enter “AMC” into the **Name** box and then click **Next**:


Database Wizard

Connection details

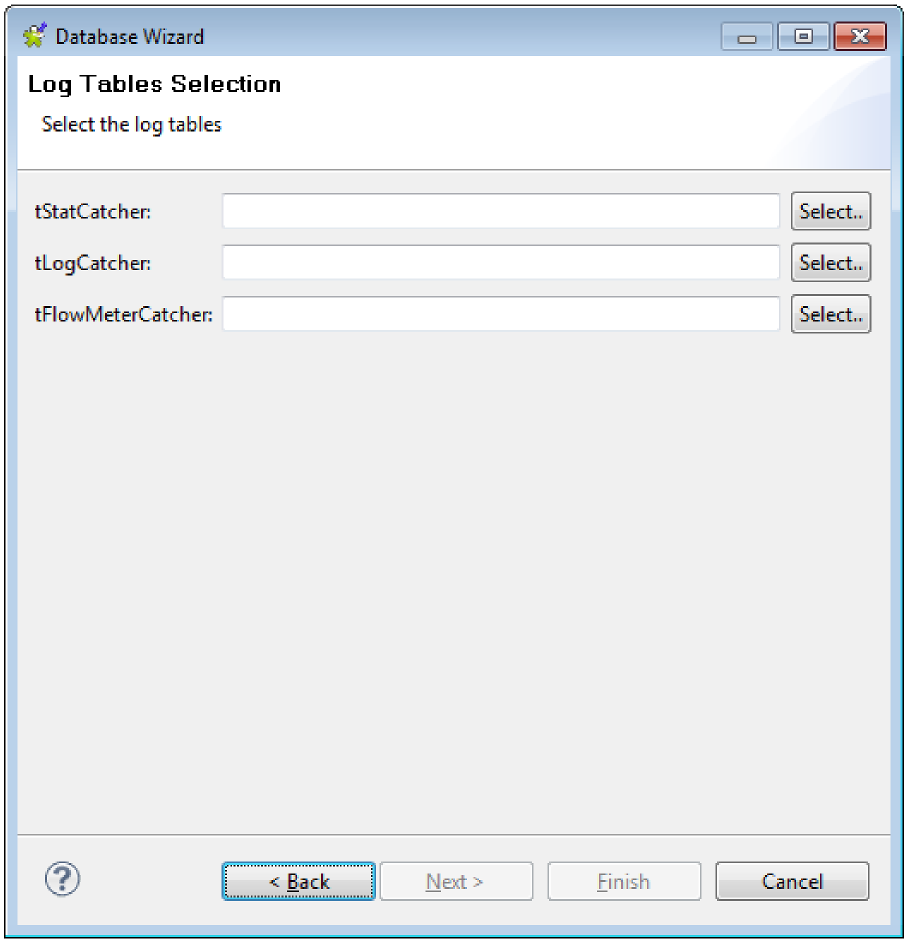
Enter connections parameters.

User id	
Password	
Database Type:	Mysql

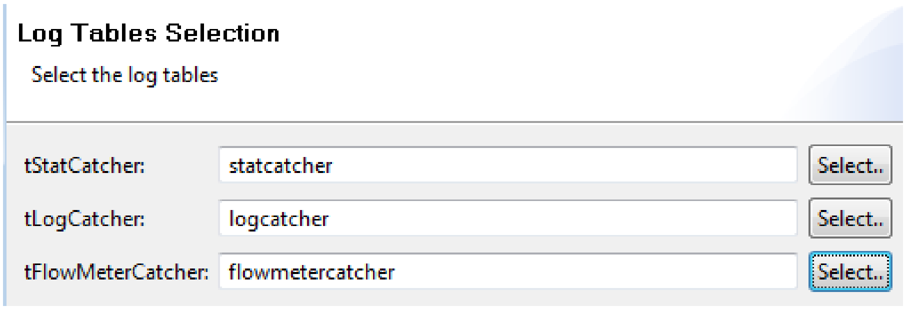
DB URL

Hostname	localhost
Port	3306
DBname	dbname
Additional JDBC Parameters	
DataSource	
Url	jdbc:mysql://localhost:3306/dbname

6. Enter the AMC database connection details into this window: "root" for both the **User id** and **Password**, and "amc" as the **DBName**. This is the database connection information the AMC uses to read the information to display. Earlier you used this same information to tell the Job where to store information. Because you can store tracking information for each Job and project separately, you need to identify the source for the AMC to read.
7. Click **Check**.
8. Click **OK** in the success window and then click **Next**:



9. Click the **Select** button next to each box in turn to choose the appropriate database tables:

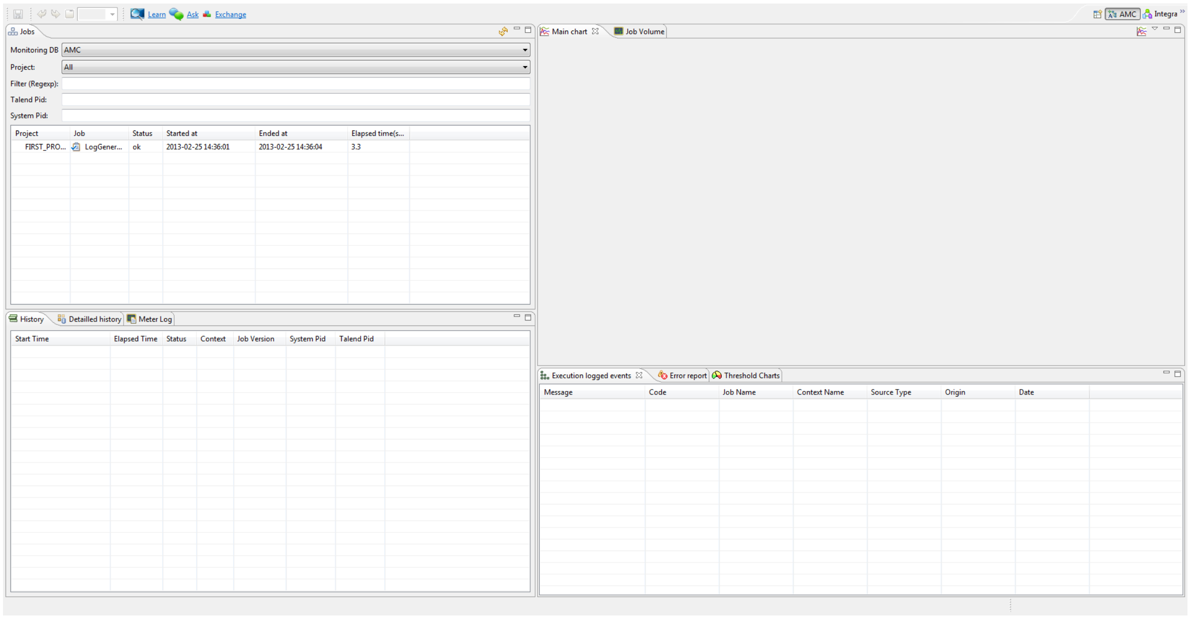


tStatCatcher:	statcatcher
tLogCatcher:	logcatcher
tFlowMeterCatcher:	flowmetercatcher

10. Click **Finish** and then click **OK**.

Examining the AMC

The **AMC Perspective** opens:



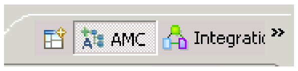
Monitoring DB: AMC
Project: All
Filter (Regexp):
Talend Pid:
System Pid:

Project	Job	Status	Started at	Ended at	Elapsed time(s...
FIRST_PRO...	LogGener...	ok	2013-02-25 14:36:01	2013-02-25 14:36:04	3.3

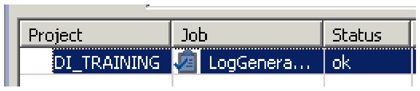
Start Time	Elapsed Time	Status	Context	Job Version	System Pid	Talend Pid

Message	Code	Job Name	Context Name	Source Type	Origin	Date

Remember that Talend Studio is built on Eclipse. A Perspective presents a different GUI through the same application. You can see which Perspective is active, and change the Perspective, from the buttons at the top right of the main window:



1. Click **Integration** and **AMC** to switch between Perspectives, and then make sure you are viewing the **AMC** Perspective before continuing.
2. In the **Jobs** view in the upper left of the window, click **LogGeneration** in the **Job** column to display the stored tracking information for that Job.



Project	Job	Status
DI_TRAINING	LogGenera...	ok

3. Explore the information available in the AMC, examining each of the tabs and being sure to click and expand each of the Job runs listed in the **LogGeneration detailed history** tab.

   As you can see, the AMC makes information available that you can use to compare a detailed history of Job runs during development to compare variations in design or configuration.

You have now completed this lesson and can move on to the Challenge.

Challenge

Overview

Complete this exercise to further explore the use of the AMC. See Solution for a possible solution to the exercise.

Introduce an Error

Switch perspectives back to **Integration** from the top right of the window, and then introduce an error into the Job. You could modify a table name in one of the output components, for example, while setting the **Action on table** to **Default** and selecting the **Die on errorcheck box**.

Run the Job and then switch to the **AMC** perspective to see what additional information is available.

Solution

Overview

This is a possible solution to the Challenge. Note that your solution may differ and still be valid.

Introduce an Error

1. Double-click one of the **tMysqlOutput** components and then change the table name to "xxx", the **Action on table** to **Default**, and then select the **Die on error** check box.
2. Run the Job.
3. Switch to the **AMC** perspective, wait for the data to refresh, and then notice the information in the **Execution logged events** and **Error report** tabs.

Wrap-Up

Recap

In this lesson, you configured the Project settings in Talend Studio to enable capturing statistics and logging information. You then imported a Job, configured it to use the Project settings, and ran the Job several times to generate tracking information. You then configured the AMC perspective and explored the information available.

Further Reading

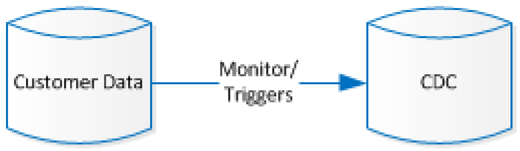
For more information about topics covered in this lesson, see the *Talend Data Integration Studio User Guide* and the *Talend Activity Monitoring Console User Guide*.

Performing Change Data Capture

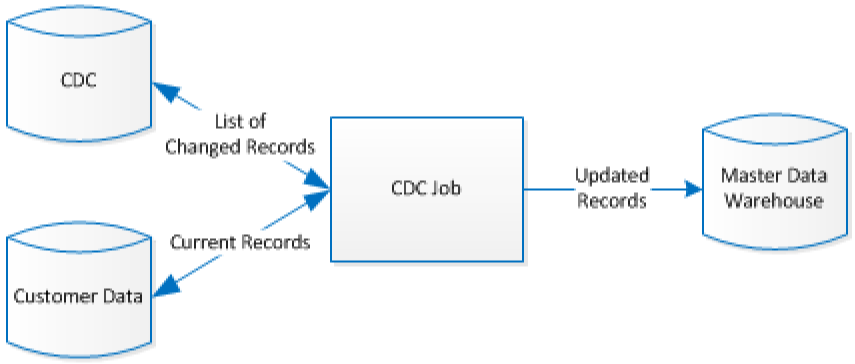
Overview

Many situations require that you keep two or more databases synchronized with each other. For example, you might have a centralized data warehouse that you need to keep current with one or more subsidiary databases. Given that today's databases are frequently massive, reloading the entire subsidiary databases into the warehouse can be impractical. A more realistic solution is to monitor the subsidiary databases for changes, then duplicate those changes in the master or warehouse database.

In this lesson, you will configure a Change Data Capture (CDC) database that monitors a different existing database of customer data for changes—record updates, deletions, and insertions:



The CDC database stores a list of the indexes of the records that have changed, the type of change, and a timestamp of when the change occurred, not the actual changes. You then create a Job that uses that list to update the master database with just the modified records from the subsidiary database:



Objectives

After completing this lesson, you will be able to:

- Examine database contents from within Talend Studio using two different mechanisms
- Configure a database table to be monitored for changes in a separate CDC database

- Create a Job that uses the information in a CDC database to update a master database table with just the changes from the monitored database table

Before You Begin

Be sure that you are familiar with the basic operations of Talend Data Integration, as covered in the training modules *Talend Studio Basics* Modules 1 and 2. Also be sure that you are working in an environment that contains the following:

- A properly installed copy of Talend Data Integration
- A properly configured set of databases
- The supporting files for this tutorial

The first step is to examine the existing databases.

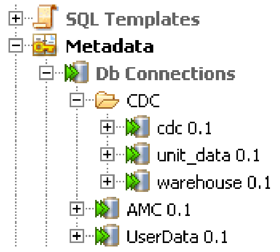
Examining the Databases

Overview

The scenario for this exercise involves a database containing customer information maintained by a single business unit and a centralized data warehouse that must be kept current with changes to the business unit database. Both of those databases have been created for you, so your first step is to examine the database connections and the data.

Identify Metadata Connections

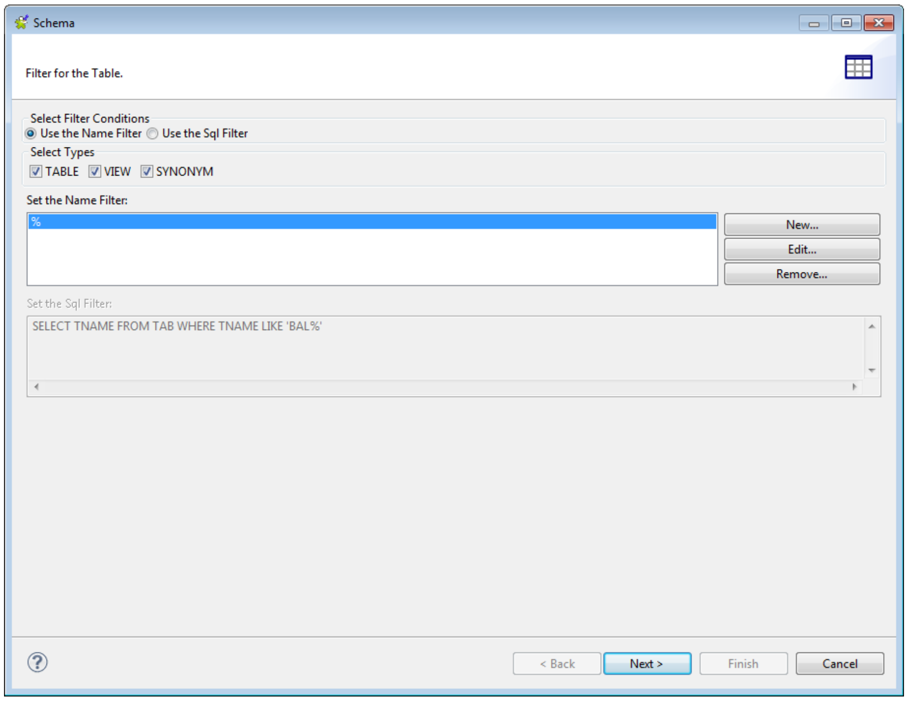
In the Talend Studio **Repository**, expand **CDC** under **Db Connections** under **Metadata**:



Note the two connections: **unit_data** for the subsidiary database and **warehouse** for the master data warehouse. In this exercise, you will be duplicating changes made to **unit_ data** in **warehouse**. You will use the **cdc** database connection later.

Retrieve Schema

1. Right-click **unit_data** in the **Repository** and then click **Retrieve Schema**:



Schema

Filter for the Table.

Select Filter Conditions

(●) Use the Name Filter ( ) Use the Sql Filter

Select Types

☑ TABLE ☑ VIEW ☑ SYNONYM

Set the Name Filter:

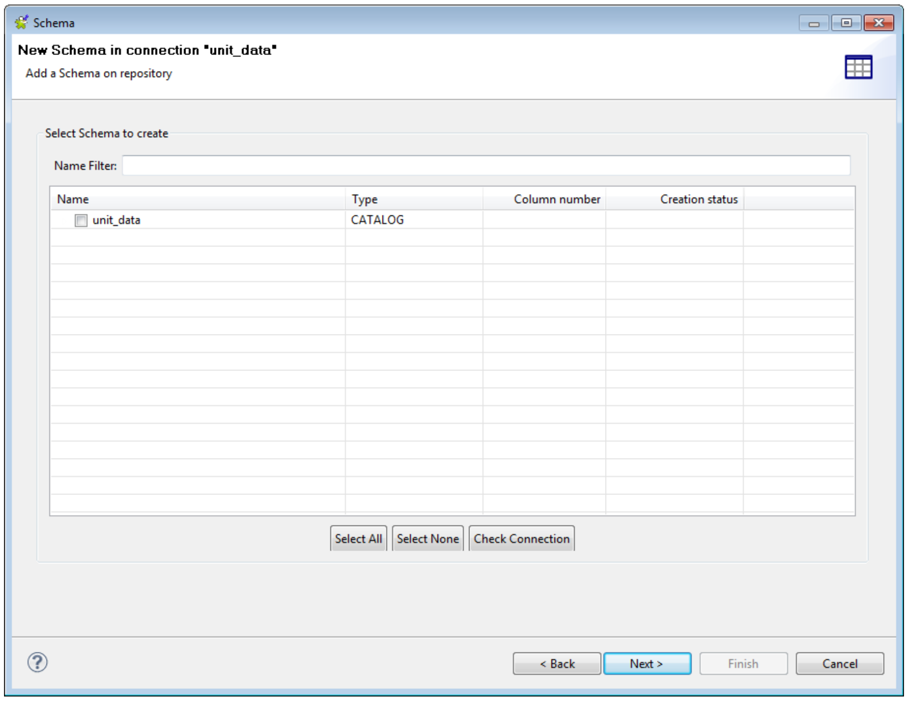
%

Set the Sql Filter:

SELECT TNAME FROM TAB WHERE TNAME LIKE 'BAL%'

You will be examining the structure of the **unit_data** database as well as reusing the table schema throughout this exercise. This window gives you an opportunity to modify which aspects of the database structure to retrieve.

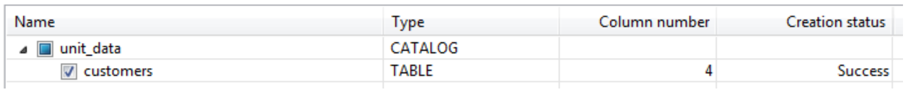
2. Leave the window as-is and then click **Next**:



Name	Type	Column number	Creation status
unit_data	CATALOG		

Here's where you choose the table(s).

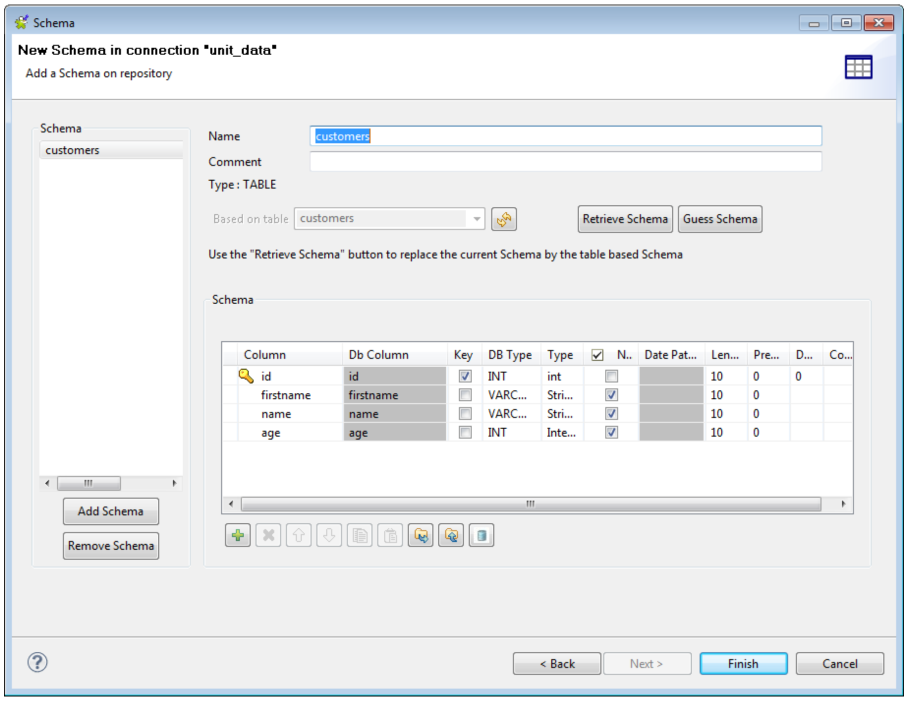
3. Expand **unit_data** and then select **customers**:



Name	Type	Column number	Creation status
unit_data	CATALOG		
☑ customers	TABLE	4	Success

This database (catalog) contains a single table.

4. Click **Next**:



Schema

New Schema in connection "unit_data"

Add a Schema on repository

Schema: customers

Name: customers

Comment:

Type : TABLE

Based on table: customers

Retrieve Schema | Guess Schema

Use the "Retrieve Schema" button to replace the current Schema by the table based Schema

Schema

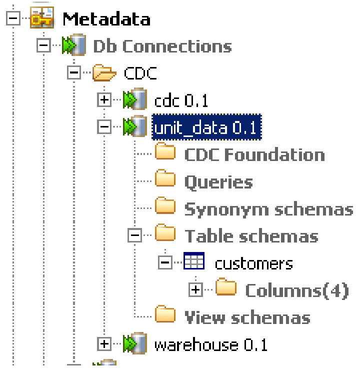
Column	Db Column	Key	DB Type	Type	✓ N..	Date Pat...	Len...	Pre...	D...	Co...
id	id	✓	INT	int	☐		10	0	0	
firstname	firstname	☐	VARC...	Stri...	✓		10	0		
name	name	☐	VARC...	Stri...	✓		10	0		
age	age	☐	INT	Inte...	✓		10	0		

Add Schema | Remove Schema

< Back | Next > | Finish | Cancel

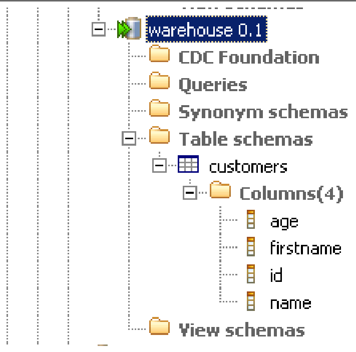
Here you can change the name you use to store the schema as well as add comments.

5. Note the names and types of the four columns in the schema, including the fact that the key is the column named **id**.
6. Click **Finish**. The schema appears under **Table schemas** in the **Repository**:



```
Metadata
  Db Connections
    CDC
      cdc 0.1
      unit_data 0.1
        CDC Foundation
        Queries
        Synonym schemas
        Table schemas
          customers
            Columns(4)
        View schemas
      warehouse 0.1
```

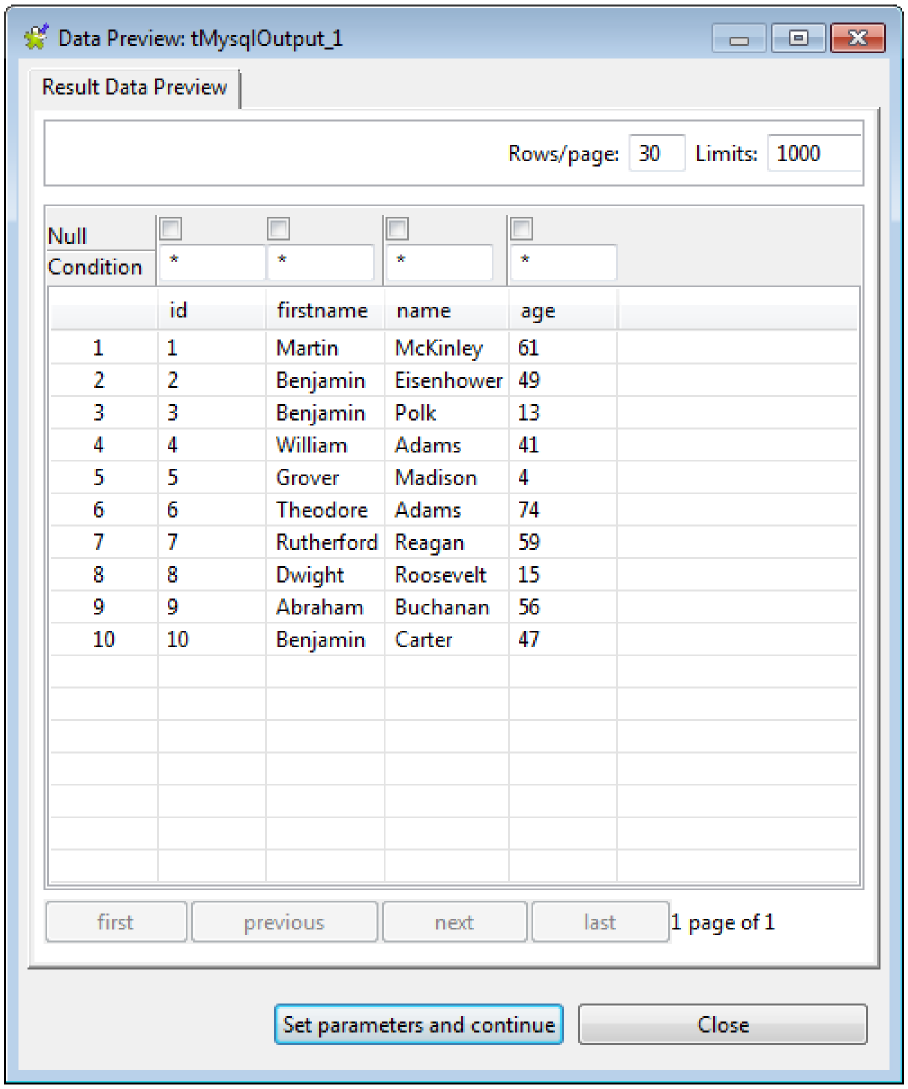
7. Repeat these steps to retrieve the schema for the **customers** table from the **warehouse** database connection:



Note that the table schema is the same in both databases.

Compare Data

1. Create a new Job, naming it "DBMods" and providing appropriate information for the **Description** and **Purpose**. Eventually this Job will make modifications to the **unit_data** database to be reflected in the **warehouse** database, but for now you are using it to examine the contents of a data table.
2. Drag the **unit_data** connection from the **Repository** onto the design workspace, choosing **tMysqlOutput** as the component type.
3. Right-click the new component and then click **Data viewer**:



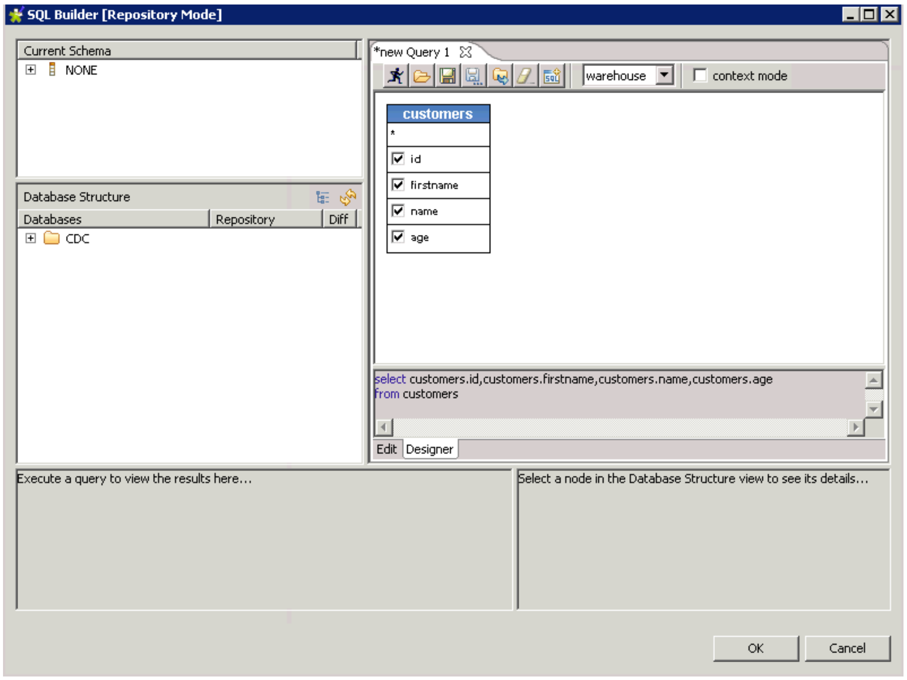
Rows/page: 30 Limits: 1000

	id	firstname	name	age
1	1	Martin	McKinley	61
2	2	Benjamin	Eisenhower	49
3	3	Benjamin	Polk	13
4	4	William	Adams	41
5	5	Grover	Madison	4
6	6	Theodore	Adams	74
7	7	Rutherford	Reagan	59
8	8	Dwight	Roosevelt	15
9	9	Abraham	Buchanan	56
10	10	Benjamin	Carter	47

1 page of 1

The data viewer allows you to examine the contents of many different types of files and databases without having to leave Talend Studio. In this case, the **customers** table contains ten rows of random data that was generated from a Talend Job.

4. Click **Close** and then save the Job.
5. Under the **warehouse** database connection in the **Repository**, right-click **customers** under **Table schemas** and then click **Edit queries**:



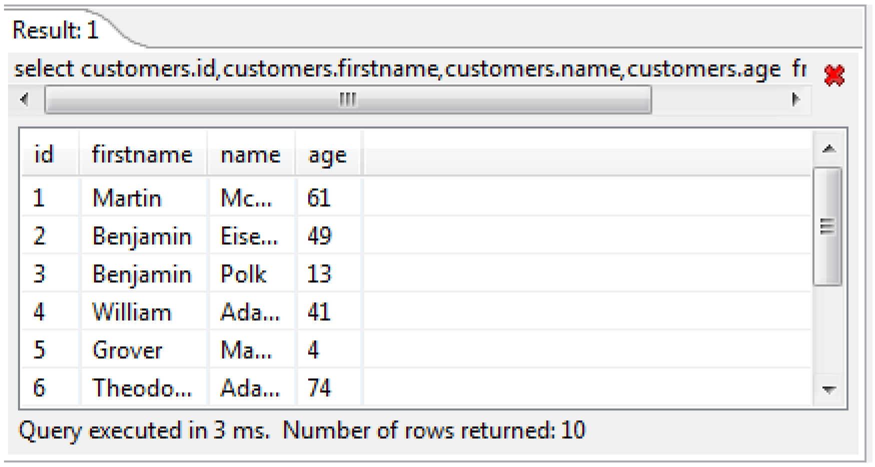
```
select customers.id,customers.firstname,customers.name,customers.age
from customers
```

Execute a query to view the results here...

Select a node in the Database Structure view to see its details...

The SQL Builder allows you to create and save a query for repeated use. Note that the default query is to retrieve all values for all columns.

6. Click the button marked with a running stick figure in the **new Query** tab to execute the query, and then examine the **Result** tab:



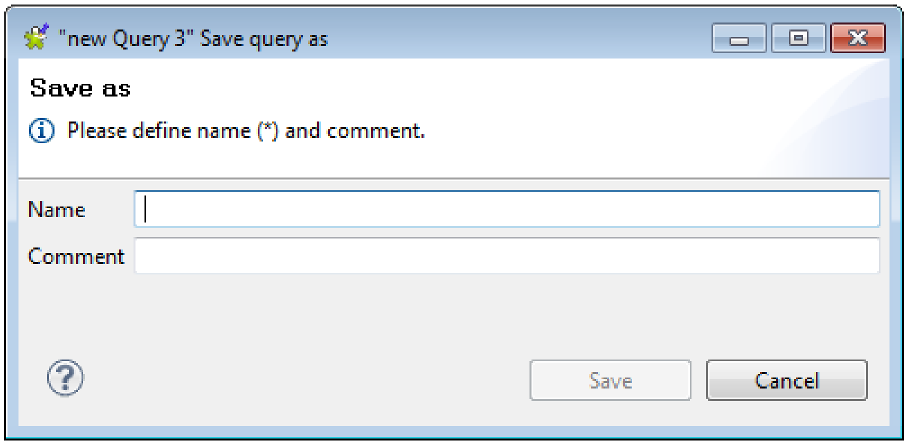
```
select customers.id,customers.firstname,customers.name,customers.age fr
```

id	firstname	name	age
1	Martin	Mc...	61
2	Benjamin	Eise...	49
3	Benjamin	Polk	13
4	William	Ada...	41
5	Grover	Ma...	4
6	Theodo...	Ada...	74

Query executed in 3 ms. Number of rows returned: 10

Note that the data in the **customers** table of the **warehouse** database is the same as the data in the **customers** table of the **unit_data** database.

7. Click **OK**. A prompt appears asking if you want to save all queries.
8. Click **Yes**:



"new Query 3" Save query as

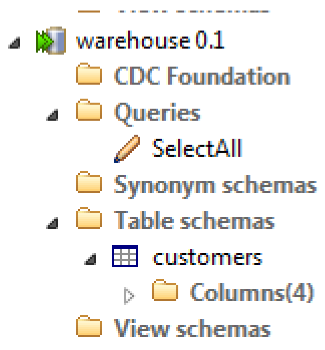
Save as

Please define name (*) and comment.

Name

Comment

9. Enter "SelectAll" into the **Name** box and appropriate text into the **Comment** box, and then click **Save**. The query apepars in the **Repository Metadata** for reuse:



```
warehouse 0.1
    CDC Foundation
    Queries
        SelectAll
    Synonym schemas
    Table schemas
        customers
            Columns(4)
    View schemas
```

Now that you've examined the existing databases, it's time to set up the CDC database.

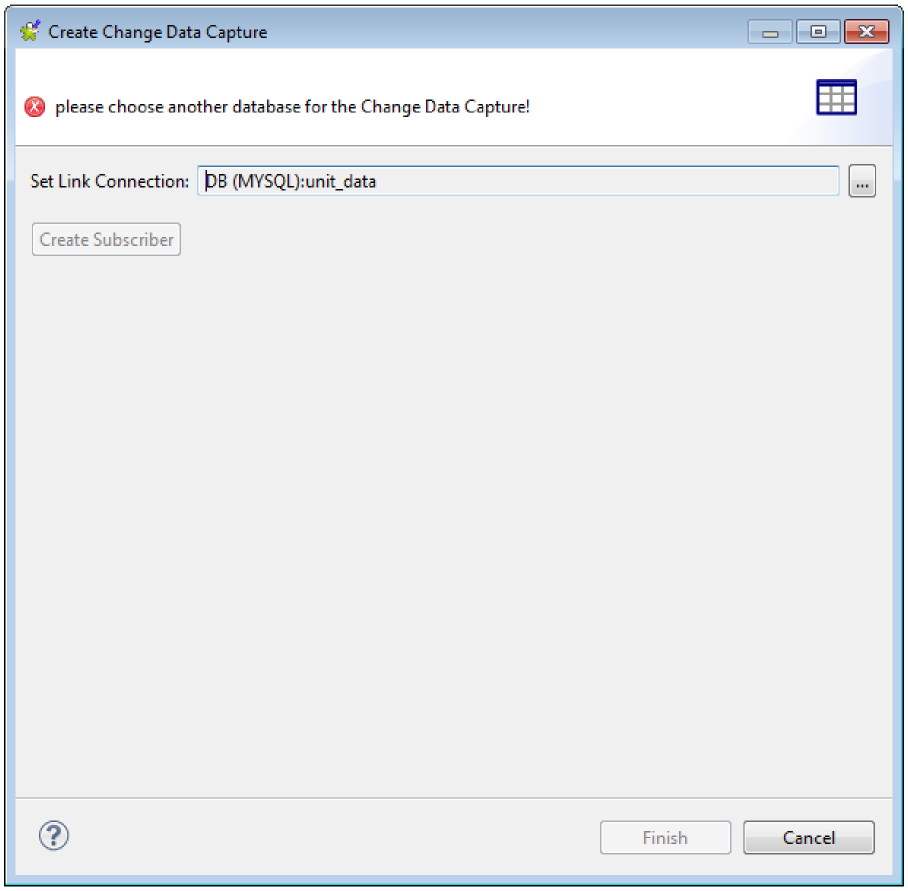
Configuring the CDC Database

Overview

The CDC database has been created for you, but is empty. You now need to set up the CDC process so that changes to the **customers** table in the **unit_data** database are recorded in the CDC database.

Define Subscribers Table

1. In the **Repository**, under the **unit_data** database connection, right-click **CDC Foundation** and then click **Create CDC**:

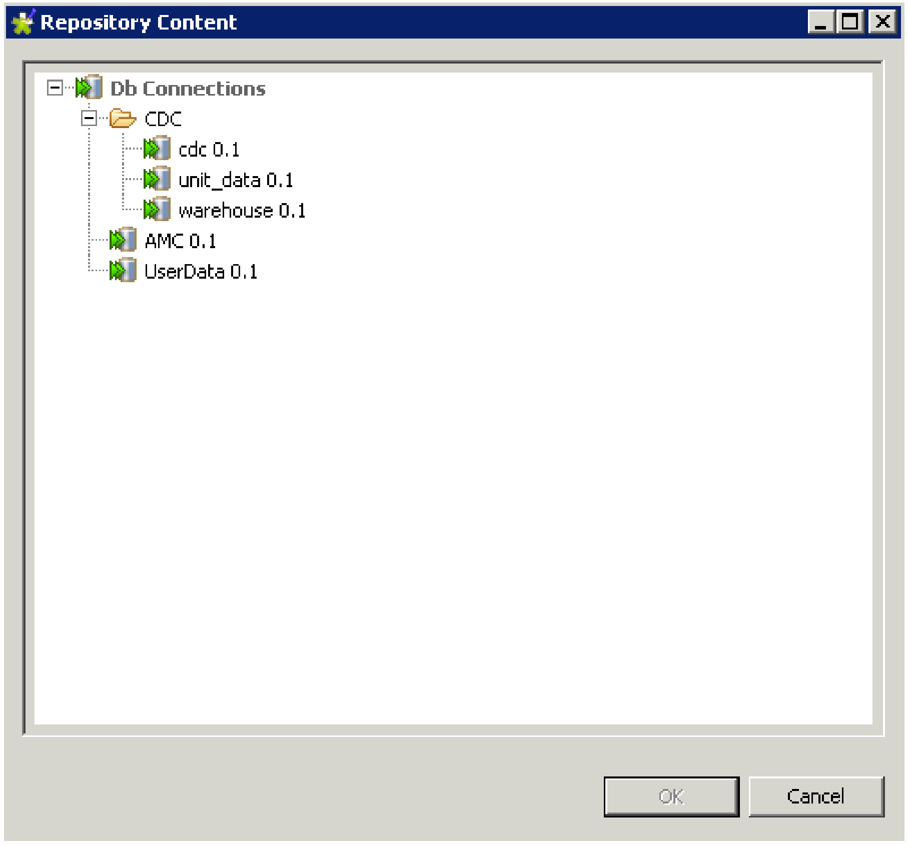


please choose another database for the Change Data Capture!

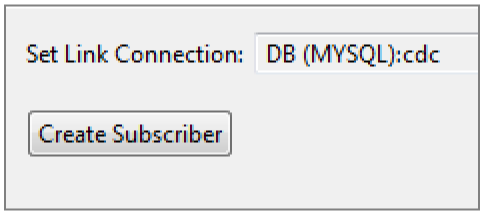
Set Link Connection: DB (MYSQL):unit_data

This is where you identify the database that will monitor changes to the **unit_data** database.

2. Click the button marked with an ellipsis next to **Set Link Connection**:

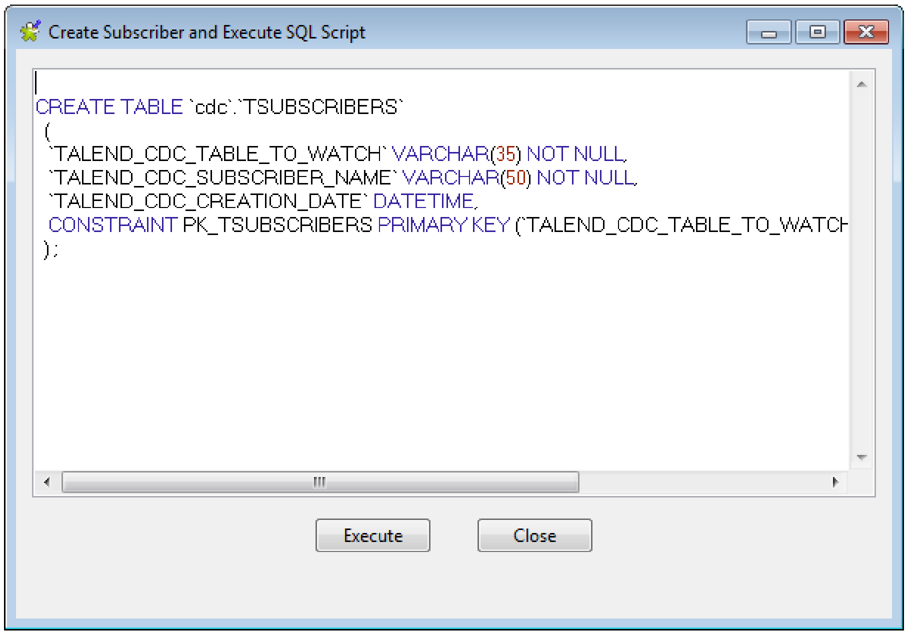


3. Expand CDC, click **cdc**, and then click **OK**. The chosen database connection appears in the window:



Set Link Connection: DB (MYSQL):cdc

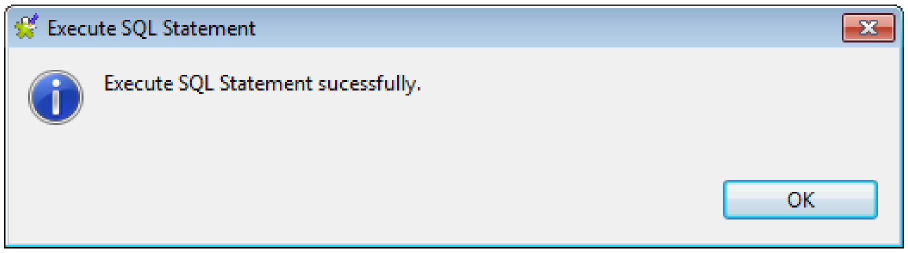
4. Click **Create Subscriber**:



```
CREATE TABLE `cdc`.`TSUBSCRIBERS`
 (
 `TALEND_CDC_TABLE_TO_WATCH` VARCHAR(35) NOT NULL,
 `TALEND_CDC_SUBSCRIBER_NAME` VARCHAR(50) NOT NULL,
 `TALEND_CDC_CREATION_DATE` DATETIME,
 CONSTRAINT PK_TSUBSCRIBERS PRIMARY KEY (`TALEND_CDC_TABLE_TO_WATCH
 );
```

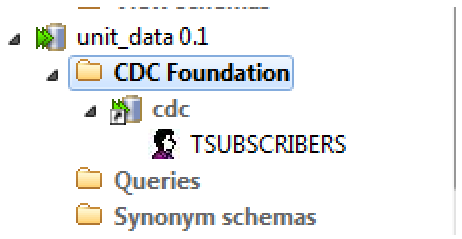
This window displays the SQL statement that will execute. Take a moment to examine the script. This creates a table named **TSUBSCRIBERS** in the **cdc** database.

5. Click **Execute**:



Execute SQL Statement sucessfully.

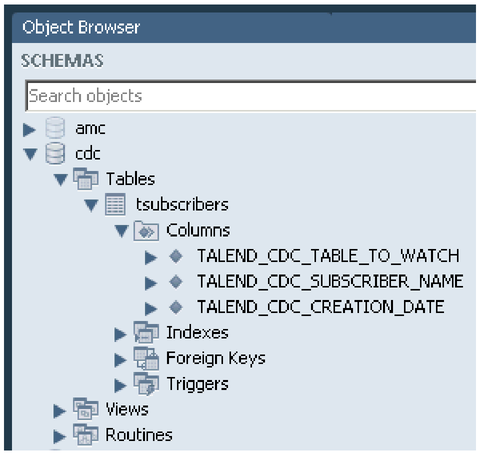
6. Click **OK** in the success window, click **Close** in the script window, and then click **Finish**. The new table appears in the **Repository**:



Note that even though the table is listed under **unit_data** in the **Repository**, it is actually in the **cdc** database. This makes it easier to see the organization when you have a single CDC database monitoring multiple tables.

7. Start the application MySQL Workbench from the shortcut on the desktop.

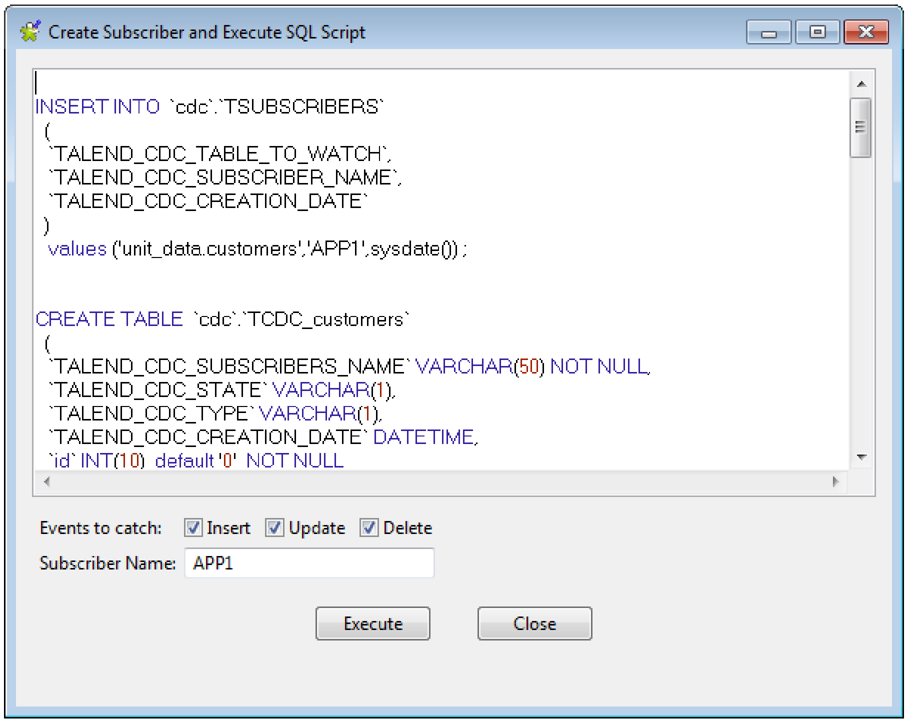
8. Under **SQL Development**, double-click **Local host**.
9. In the **Object Browser**, expand **cdc** to see the details of the new table:



The table and column names should look familiar from the SQL script you just ran from Talend Studio. These columns store information about what table to monitor.

Define CDC Table

1. In the Talend Studio **Repository**, under the **unit_data** database connection, right-click **customers** under **Table schemas** and then click **add CDC**:



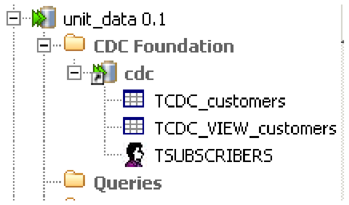
```
INSERT INTO `cdc`.`TSUBSCRIBERS`
 (
 `TALEND_CDC_TABLE_TO_WATCH`,
 `TALEND_CDC_SUBSCRIBER_NAME`,
 `TALEND_CDC_CREATION_DATE`
 )
 values ('unit_data.customers','APP1',sysdate()) ;


CREATE TABLE `cdc`.`TCDC_customers`
 (
 `TALEND_CDC_SUBSCRIBERS_NAME` VARCHAR(50) NOT NULL,
 `TALEND_CDC_STATE` VARCHAR(1),
 `TALEND_CDC_TYPE` VARCHAR(1),
 `TALEND_CDC_CREATION_DATE` DATETIME,
 `id` INT(10) default '0' NOT NULL
```

Events to catch: ☑ Insert ☑ Update ☑ Delete

Subscriber Name: APP1

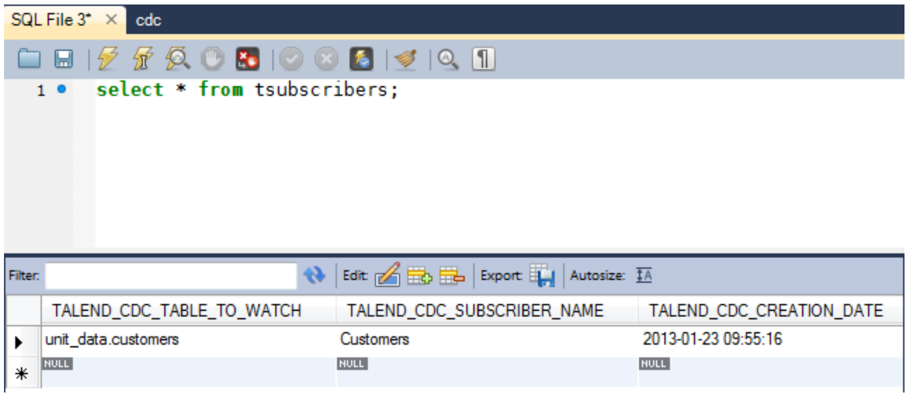
This is another SQL script that inserts data into the existing subscribers table, and creates a new table and view in the **cdc** database.

2. Change the **Subscriber Name** to "Customers" and then click **Execute**.
3. Click **OK** in the success window and then click **Close** in the script window. Note the new information in the **Repository** under **unit_data**:



Again, these are listed under **unit_data** in the **Repository** for convenience, but the tables and view are actually in the **cdc** database.

4. In MySQL Workbench, double-click **cdc** in the **Object Browser** and then execute the SQL statement "select * from tsubscribers;":

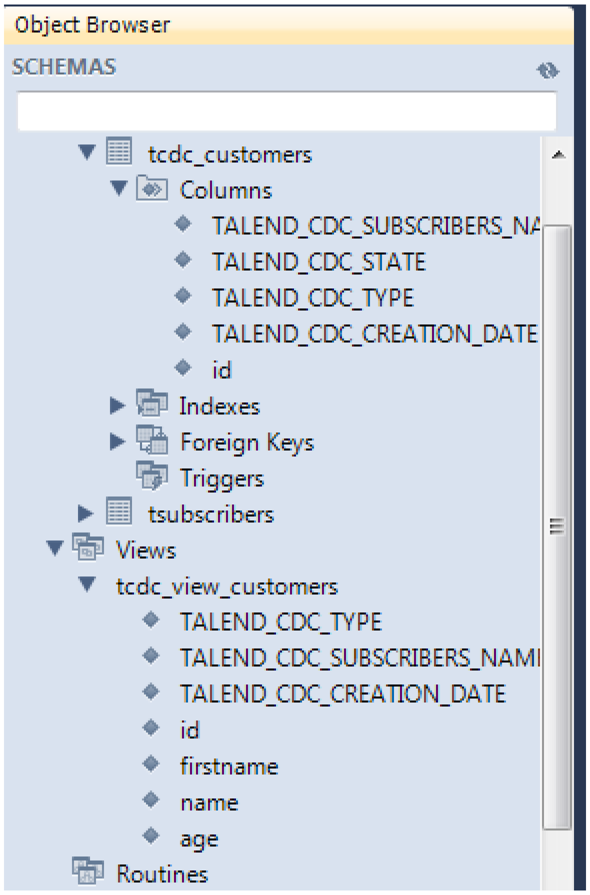


```
select * from tsubscribers;
```

	TALEND_CDC_TABLE_TO_WATCH	TALEND_CDC_SUBSCRIBER_NAME	TALEND_CDC_CREATION_DATE
▸	unit_data.customers	Customers	2013-01-23 09:55:16
*	NULL	NULL	NULL

Here you see the table being monitored (**unit_data.customers**), the name of the subscriber you just created (**Customers**), and a timestamp specifying when the subscriber was defined.

5. Still in Workbench, examine the new table and view in the **cdc** table in the **Object Browser** by right-clicking **cdc** and then clicking **Refresh All**:

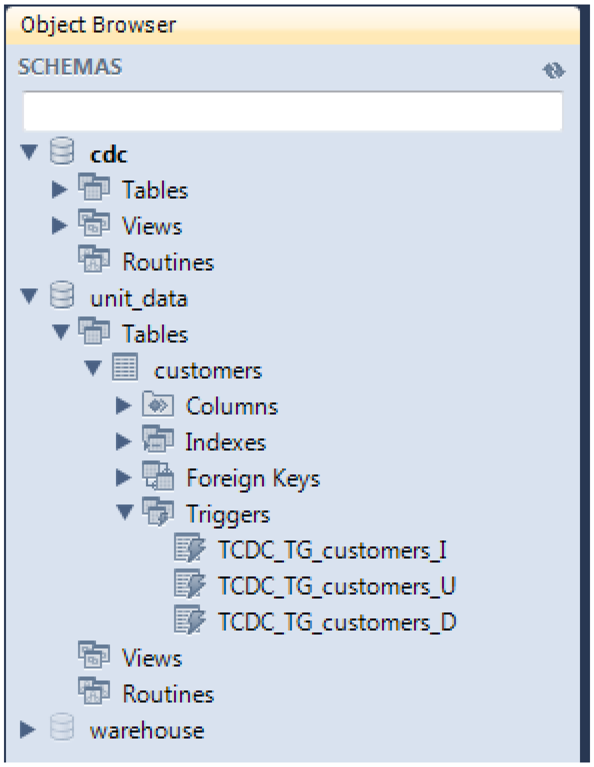


The new table **tcdc_customers** stores information about changes in the subscribed table (in this case, the **customers** table of **unit_data**). Here is an explanation of the columns in **tcdc_customers**:

- TALEND_CDC_SUBSCRIBERS_NAME: the name of the subscriber as listed in the **tsubscribers** table. In this exercise, **Customers** is the subscriber name and identifies the table being monitored.
- TALEND_CDC_STATE: a flag indicating whether or not this change has been applied.
- TALEND_CDC_TYPE: one of U, D, or I specifying the type of change (update, delete, or insert, respectively).
- TALEND_CDC_CREATION_DATE: a timestamp specifying when the record changed.
- id: the key value identifying the changed record. Note that the name of this column is specific to the table being monitored. Remember that the column **id** is the key column for the **customers** table in the **unit_data** database.

6. Also examine the triggers in **unit_data**, specifying that all three types of changes

are being monitored insert, update, delete):



Now you are ready to make some changes to the monitored table.

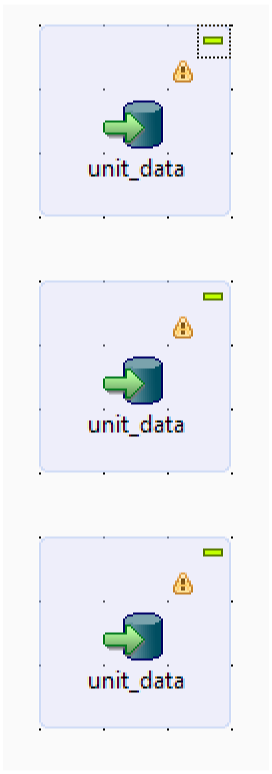
Monitoring Changes

Overview

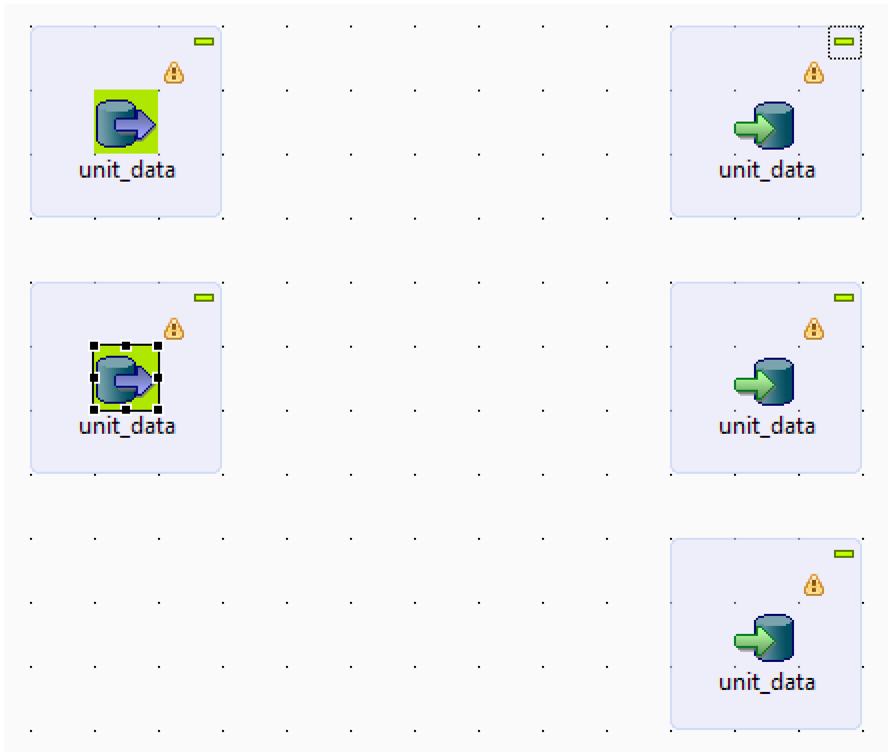
With the CDC monitoring in place, you can now make some changes to the **unit_data** database to see how CDC tracks the changes.

Build the Modify Job

1. Open the Job **DBMods** you created earlier, if it isn't already open. At this point the Job has a single **tMysqlOutput** component for the **unit_data** database. You will extend this Job so that it makes one of each type of change to the **customers** table in **unit_data**: delete, update, and insert.
2. Copy and paste the **tMysqlOutput** component twice so that you have a total of three:



3. Drag the **unit_data** database connection from the **Repository** onto the design workspace to the left of the existing components, this time choosing the **tMysqlInput** component.
4. Copy and paste the **tMysqlInput** component so that you have two:



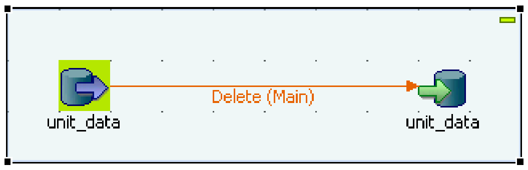
Now you can configure this Job to make the changes.

Delete a Row

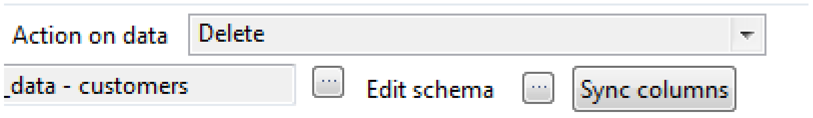
1. Double-click the first **tMysqlInput** component to open the **Component** view. You are going to use this component to choose a record for deletion.
2. Replace the text in the **Query** box so that it reads as follows, including the quotations:

```
"SELECT * FROM customers WHERE id='1'"
```

3. Connect this component to the first **tMysqlOutput** component with a **Main** row, and then name the row "Delete":



4. Double-click this first output component and change the **Action on data** to **Delete**:



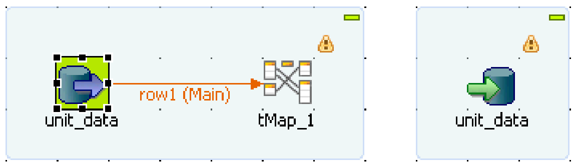
Action on data Delete

_data - customers

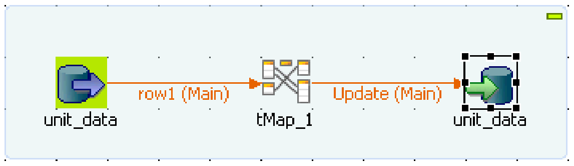
This subjob deletes the row with a value of 1 in the **id** column.

Update a Row

1. To create a second subjob to modify a row, insert a **tMap** component between the second input and output components, connecting the input component to **tMap** with a **Main** row:



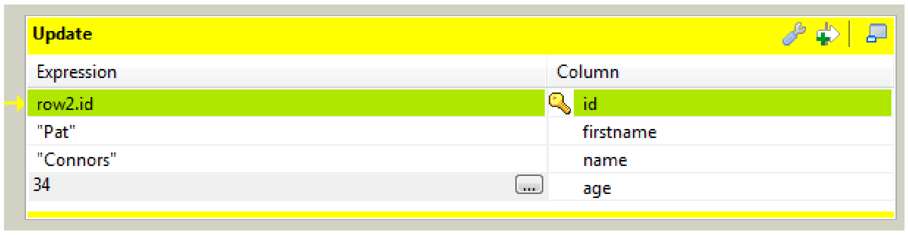
2. Connect the **tMap** component to the output component with a new output, naming it "Update". When prompted to get the schema, click **Yes**:



3. Double-click the **tMysqlInput** component and then replace the text in the **Query** box so that it reads as follows:

```
"SELECT * FROM customers WHERE id = '2'"
```

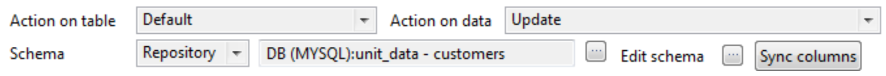
4. Double-click the **tMap** component to open the configuration window.
5. Drag the **id** column from the **row1** table to the **Update** table so that the output row will have the same value in the **id** column (you're performing an update to an existing row, not creating a new record).
6. For each of the remaining columns in the **Update** table, enter specific values into the **Expression** field: use your own first and last name and age, for example. Just be sure that you remember the values, and be sure to surround string values with quotations:



Update

Expression	Column
row2.id	id
"Pat"	firstname
"Connors"	name
34	age

7. Click **Ok**.
8. Double-click the associated **tMysqlOutput** component and then change the value of **Action on data** to **Update**:

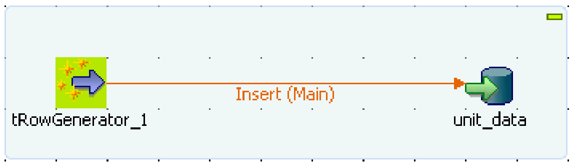


Action on table	Default	Action on data	Update
Schema	Repository	DB (MYSQL):unit_data - customers	Edit schema / Sync columns

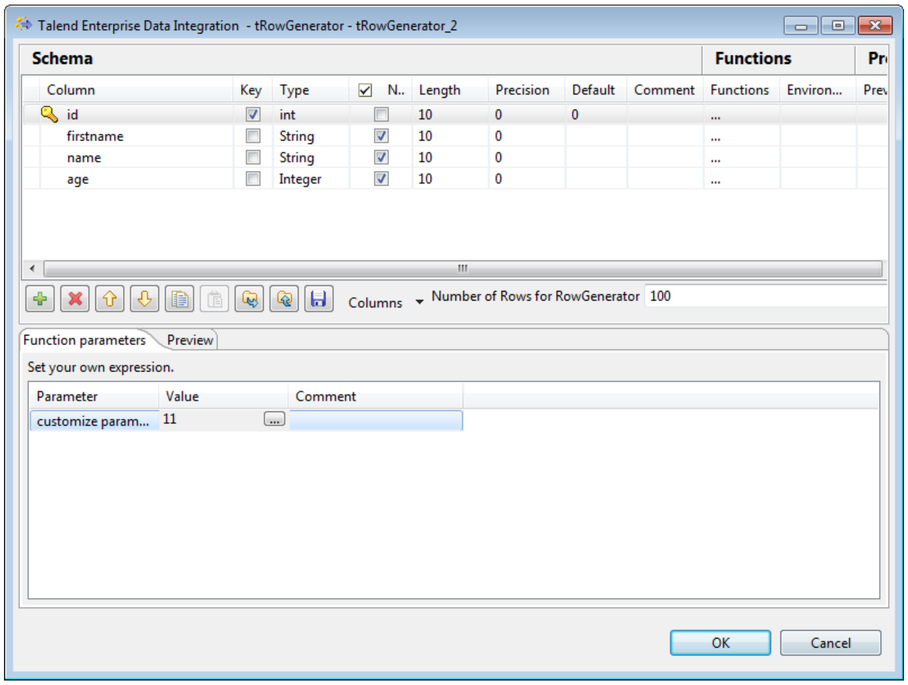
Now this subjob will update the record with 2 in the **id** column with new values for the remaining columns.

Insert a Row

1. To configure the final subjob to insert a row, add a **tRowGenerator** component to the design workspace and then connect it to the last **tMysqlOutput** component with a **Main** Row, naming it "Insert". When prompted to get the schema, click **Yes**:



2. Double-click the **tRowGenerator** component.
3. Click the **Functions** field for the **id** column and then enter "11" as the value in the **Function parameters** area (click the image to enlarge):



Talend Enterprise Data Integration - tRowGenerator - tRowGenerator_2

Column	Key	Type	N..	Length	Precision	Default	Comment	Functions	Environ...	Prev
id	✓	int		10	0	0		...		
firstname		String	✓	10	0			...		
name		String	✓	10	0			...		
age		Integer	✓	10	0			...		

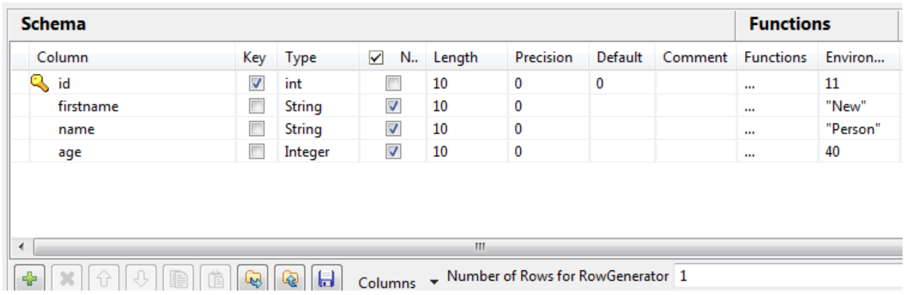
Number of Rows for RowGenerator 100

Function parameters | Preview

Set your own expression.

Parameter	Value	Comment
customize param...	11	

4. Add a custom value for each of the other columns, choosing values other than the ones you used for the row update subjob, and being sure to surround string values with quotations.
5. Change the **Number of Rows** value to 1 (click to enlarge image):



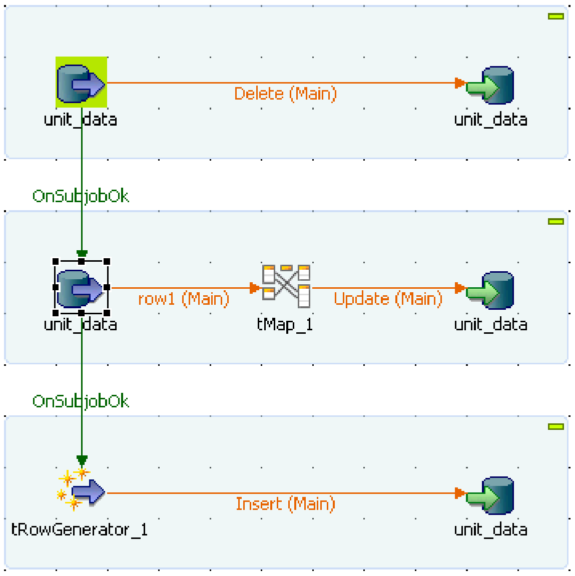
Column	Key	Type	N..	Length	Precision	Default	Comment	Functions	Environ...
id	✓	int		10	0	0		...	11
firstname		String	✓	10	0			...	"New"
name		String	✓	10	0			...	"Person"
age		Integer	✓	10	0			...	40

Number of Rows for RowGenerator 1

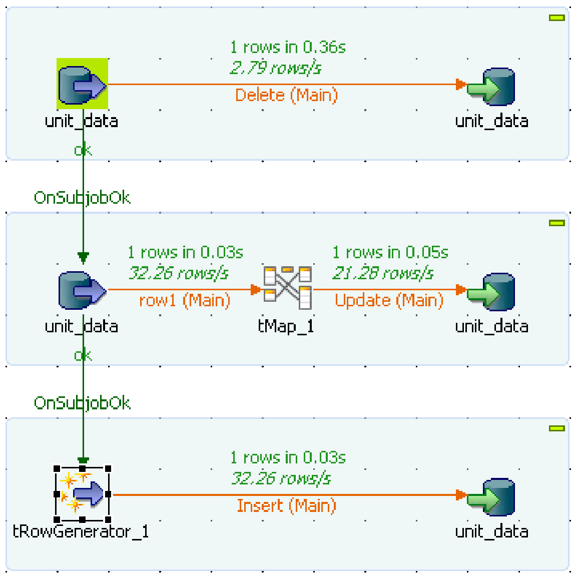
6. Click **OK**.
7. Verify that the **Action on data** value in the connected **tMysqlOutput** component is **Insert**. This subjob will now insert a new row into the table with a value of 11 in the **id** column.

Run the Job

1. Connect the subjobs with two **On SubJob Ok** triggers:



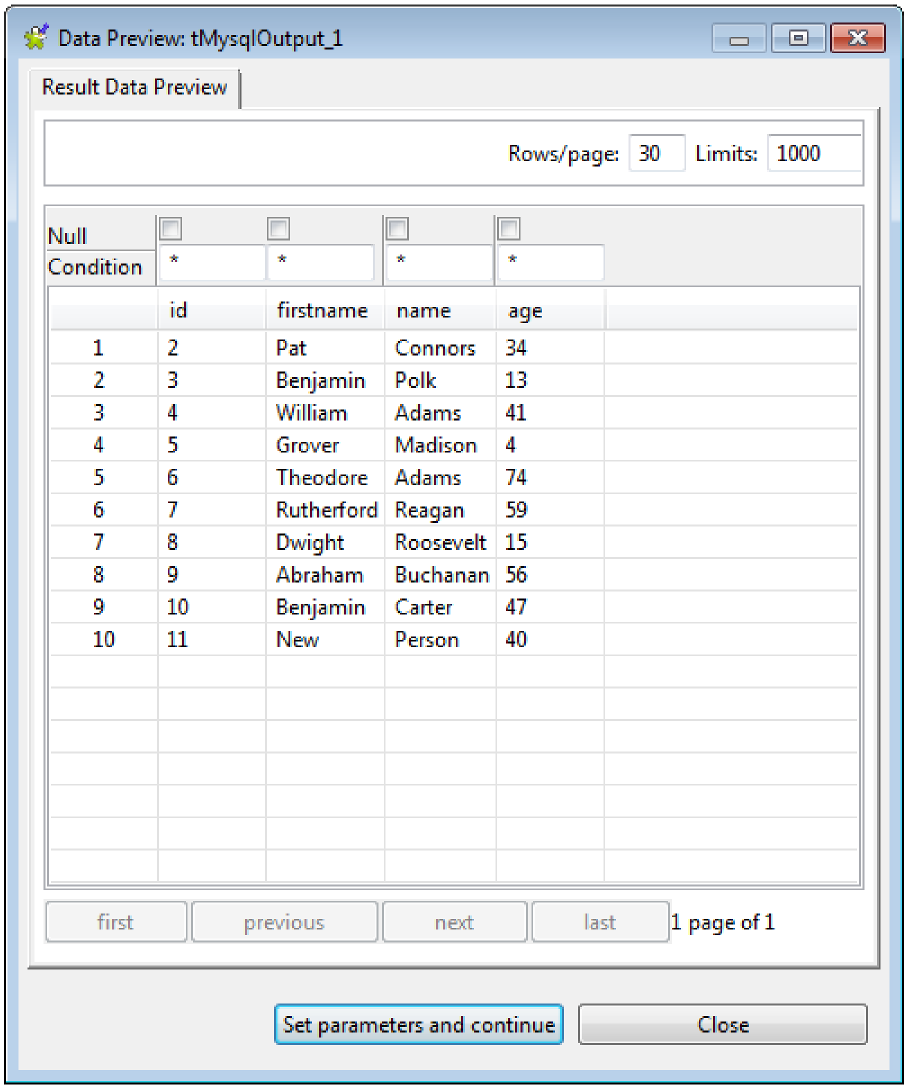
2. Run the Job.
3. Each subjob should process a single row, as displayed in the statistics:



Note: if you see any other result, you will need to follow the instructions in Resetting the Databases before making corrections and continuing.

Examine the CDC Database

1. Right-click the **tMysqlOutput** component in the last subjob and then click **Data viewer**:



Data Preview: tMysqlOutput_1

Result Data Preview

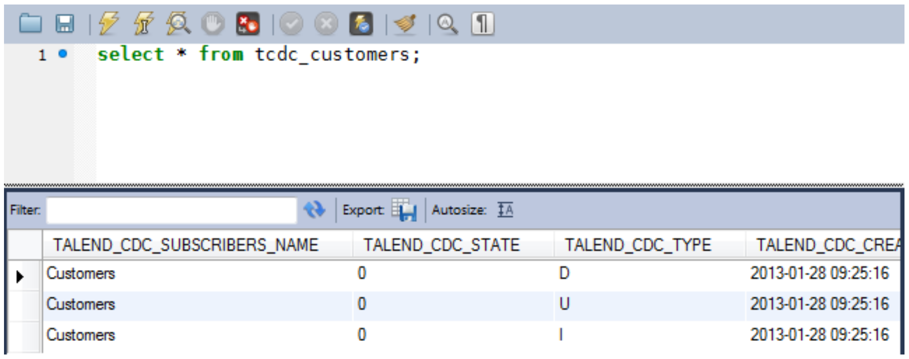
Rows/page: 30 Limits: 1000

Null Condition	*	*	*	*
	id	firstname	name	age
1	2	Pat	Connors	34
2	3	Benjamin	Polk	13
3	4	William	Adams	41
4	5	Grover	Madison	4
5	6	Theodore	Adams	74
6	7	Rutherford	Reagan	59
7	8	Dwight	Roosevelt	15
8	9	Abraham	Buchanan	56
9	10	Benjamin	Carter	47
10	11	New	Person	40

Note that the table reflects the three changes: the record with id 1 is gone, the record with id 2 has changed, and a new record with id 11 is inserted.

2. Click **Close**.
3. In MySQL Workbench, double-click **cdc** in the **Object Browser** and then execute the following query (click to enlarge image):

    Select * from tcdc_customers;



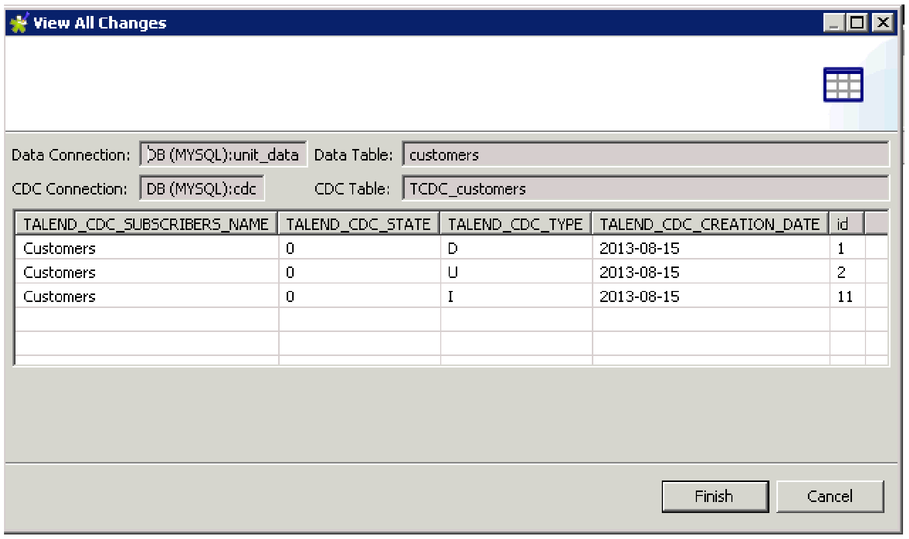
```
select * from tcdc_customers;
```

Filter: Export: Autosize:

	TALEND_CDC_SUBSCRIBERS_NAME	TALEND_CDC_STATE	TALEND_CDC_TYPE	TALEND_CDC_CREA
▸	Customers	0	D	2013-01-28 09:25:16
	Customers	0	U	2013-01-28 09:25:16
	Customers	0	I	2013-01-28 09:25:16

Note the list of three records, each tracking one of the changes that resulted from the Job you just ran. Also note the id number associated with each record (not shown in the image) specifying which record was changed. Again, if you see a different result, you will need to follow the instructions in Resetting the Databases before making corrections and continuing.

4. In the **Repository**, under the **unit_data** database connection, right-click **customers** under **Table schemas** and then click **View All Changes**:



View All Changes

Data Connection: DB (MYSQL):unit_data Data Table: customers

CDC Connection: DB (MYSQL):cdc CDC Table: TCDC_customers

TALEND_CDC_SUBSCRIBERS_NAME	TALEND_CDC_STATE	TALEND_CDC_TYPE	TALEND_CDC_CREATION_DATE	id
Customers	0	D	2013-08-15	1
Customers	0	U	2013-08-15	2
Customers	0	I	2013-08-15	11

This method allows you to view the changes from within Talend Studio.

Now that you have made changes to the table in **unit_data** and verified that those changes were tracked in the CDC database, you can build the Job that will use the cdc database to update the warehouse database.

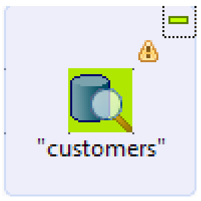
Updating the Warehouse

Overview

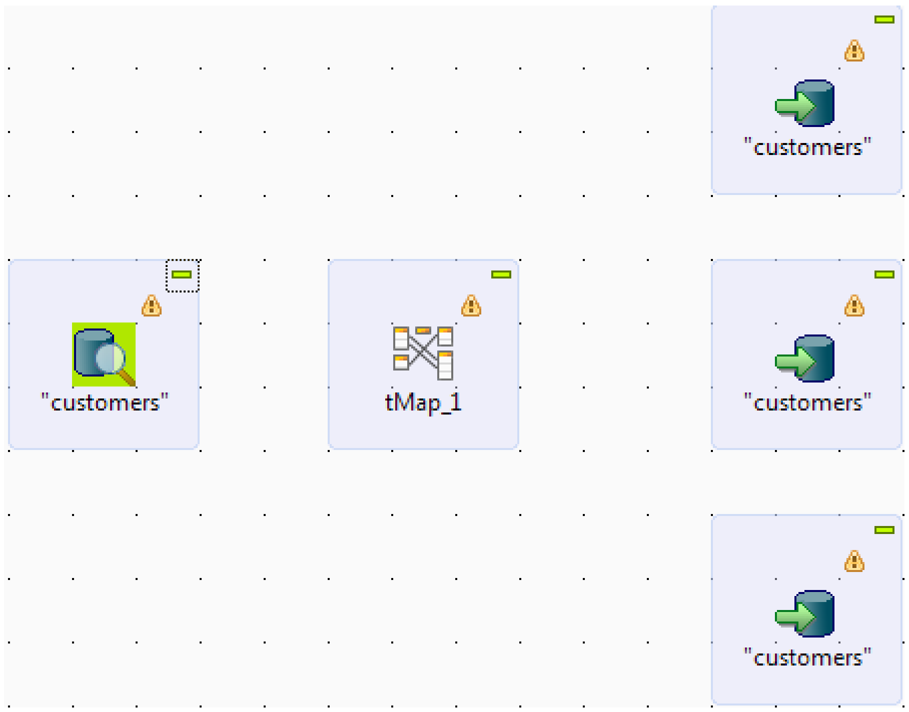
The CDC database table now contains information about changes made to the **customers** table in the **unit_data** database. You can now build a Job to update the **warehouse** database table using the CDC table.

Update the Warehouse

1. Create a new Job, naming it "SyncWarehouse".
2. Drag the **customers** schema from the **Repository** under the **unit_data** database connection onto the design workspace.
3. Click **tMysqlCDC** in the list that appears and then click **OK**:

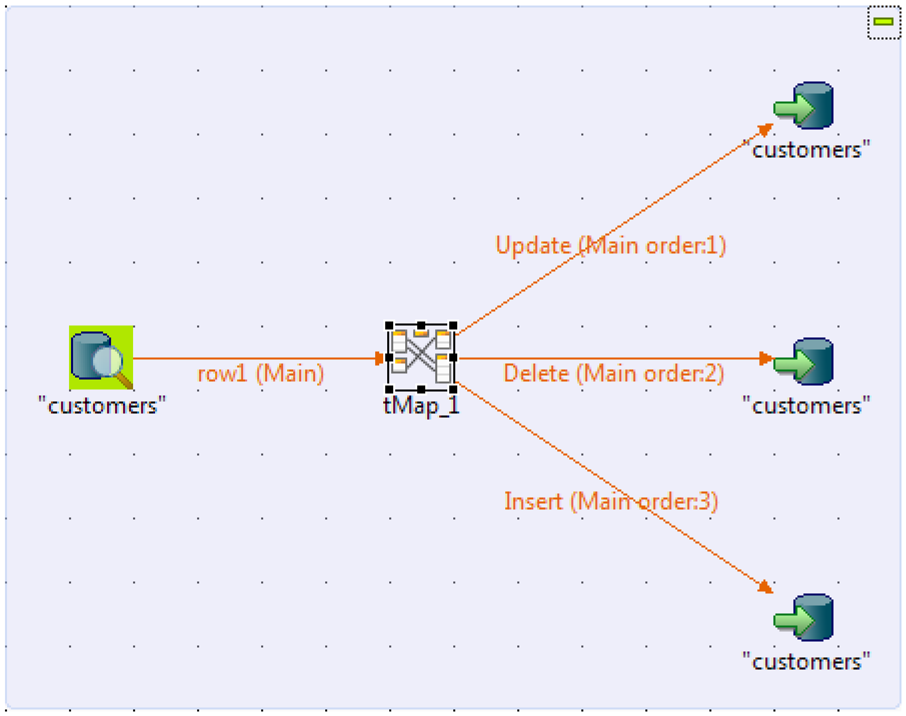
   

   This component extracts changed data from the subscribed table and makes it available for processing.
4. Double-click the component to open the **Component** view and then enter "Customers" into the **Subscriber** box, including the quotes. Remember that "Customers" is the name you used when creating the CDC subscriber.
5. Place a **tMap** component to the right of the **tMysqlCDC** component.
6. Drag the **customers** schema from the **Repository** under the **warehouse** database connection onto the design workspace, clicking **tMysqlOutput** in the list that appears. Be sure to use the **warehouse** database, not the **unit_data** database.
7. Copy and paste the new component twice for a total of three:



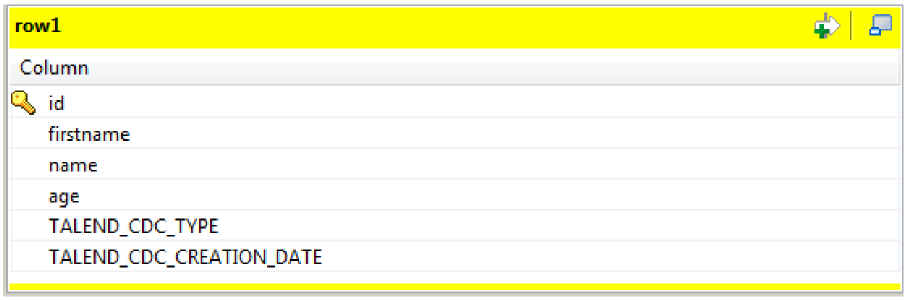
Each of these output components will submit a different type of change to the **warehouse** database: update, delete, and insert.

8. Configure the **Action on data** parameter to **Update** for the first **tMysqlOutput** component, to **Delete** for the second component, and leave the third as **Insert**.
9. Connect the **tMysqlCDC** component to the **tMap** component with a **Main** row, then connect the **tMap** component to each of the output components in turn with a new output row. Name the rows "Update", "Delete", and "Insert", respectively, each time clicking **Yes** when prompted to get the schema:



Configure tMap

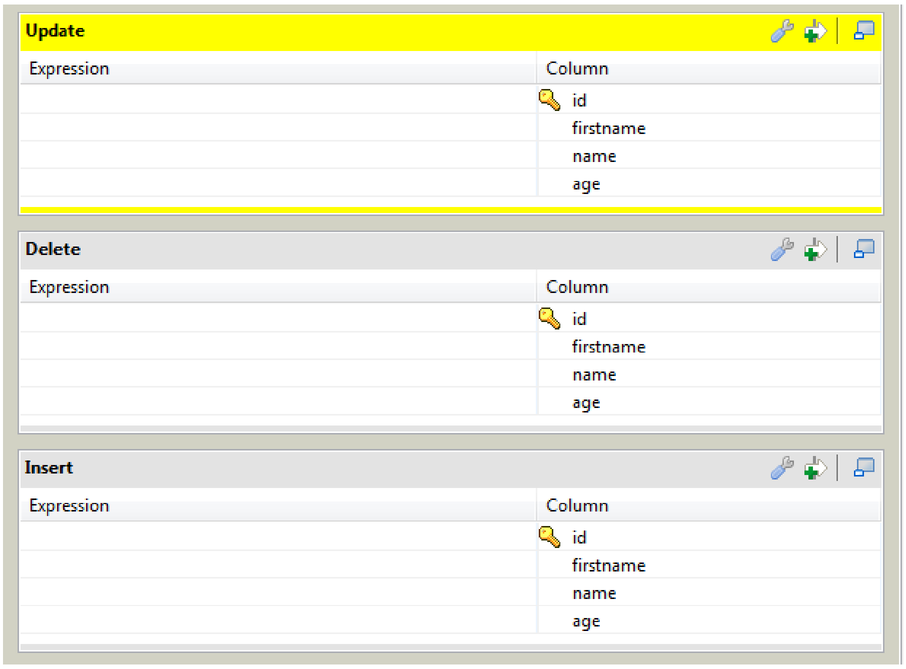
1. Double-click the **tMap** component to open the configuration window. Note that the input schema **row1** contains the columns you examined earlier from MySQL Workbench for the **cdc** database:



row1

Column
id
firstname
name
age
TALEND_CDC_TYPE
TALEND_CDC_CREATION_DATE

Also note that all three output schema tables have the correct columns for the **customers** table:



Update

Expression	Column
	id
	firstname
	name
	age

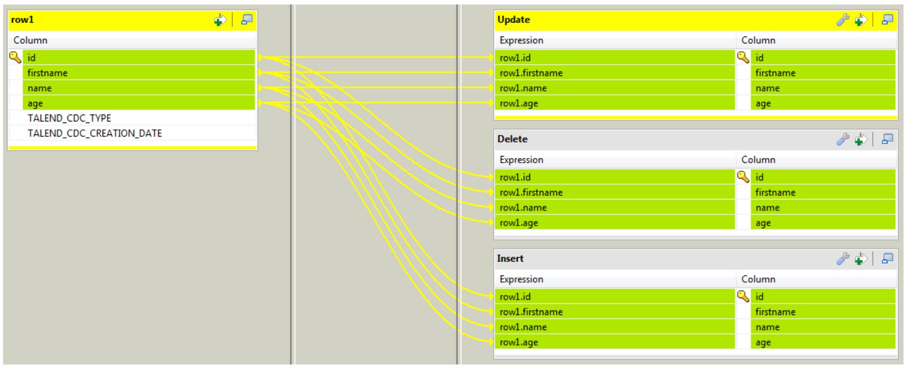
Delete

Expression	Column
	id
	firstname
	name
	age

Insert

Expression	Column
	id
	firstname
	name
	age

2. Click **Auto map!** above the output schema tables. This maps all columns from the input schema table to the **Expression** field for each of the matching columns in the output schema tables (click image to enlarge):



row1

Column
id
firstname
name
age
TALEND_CDC_TYPE
TALEND_CDC_CREATION_DATE

Update

Expression	Column
row1.id	id
row1.firstname	firstname
row1.name	name
row1.age	age

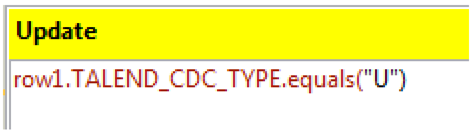
Delete

Expression	Column
row1.id	id
row1.firstname	firstname
row1.name	name
row1.age	age

Insert

Expression	Column
row1.id	id
row1.firstname	firstname
row1.name	name
row1.age	age

3. Click **Activate expression filter** (the button marked with an arrow and a plus sign) above the **Update** output table.
4. Drag **TALEND_CDC_TYPE** from the **row1** input table to the expression field and then append ".equals("U")":



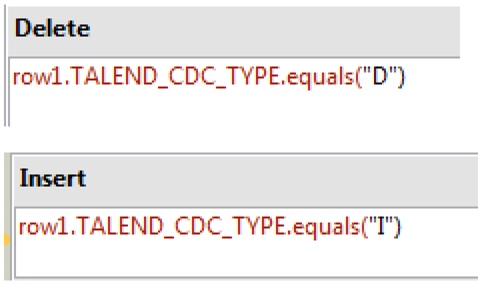
Update

```
row1.TALEND_CDC_TYPE.equals("U")
```

This expression compares the value of the column from the input row to see if it contains the letter "U", indicating an update.

5. Repeat this step for the other two output tables, replacing "U" with "D" for delete

and "I" for insert:



Delete

```
row1.TALEND_CDC_TYPE.equals("D")
```

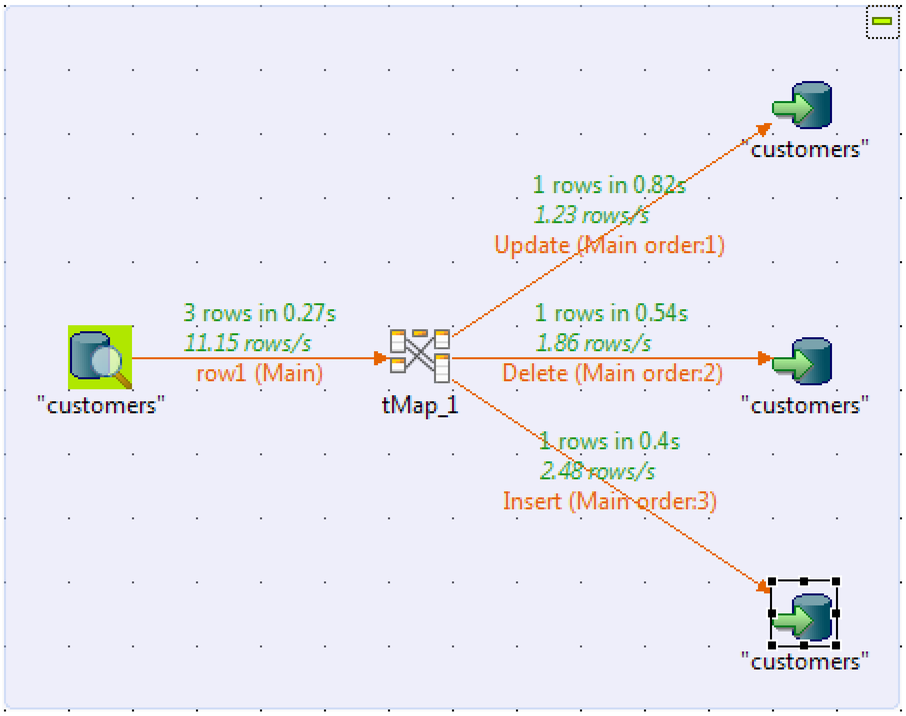
Insert

```
row1.TALEND_CDC_TYPE.equals("I")
```

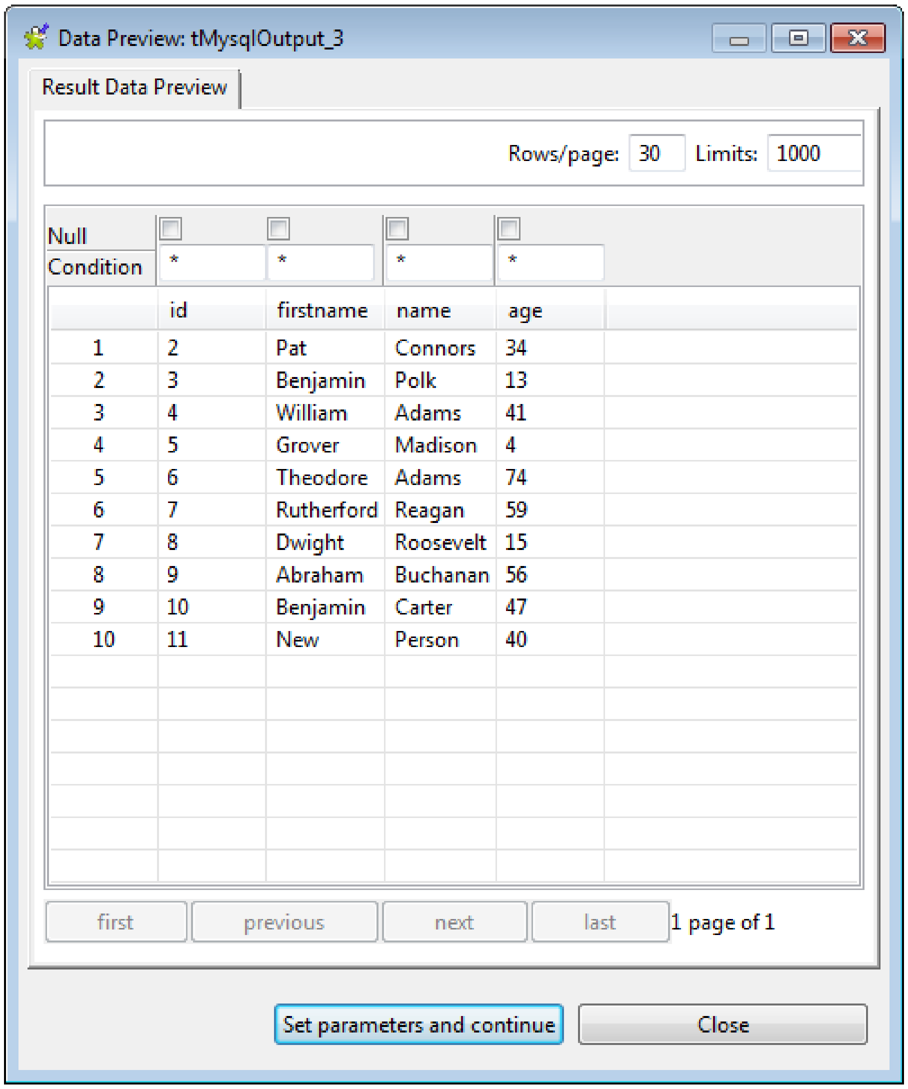
Now rows retrieved by the CDC component from the **unit_data** database will be sent to a different output depending on the type of change the row represents.

Run the Job

1. Click **Ok** to save the **tMap** changes.
2. Run the Job. The statistics should show one row processed by each output component:



3. Right-click one of the output components to examine the **warehouse** table with the data viewer to verify that it now matches the table in **unit_data**:



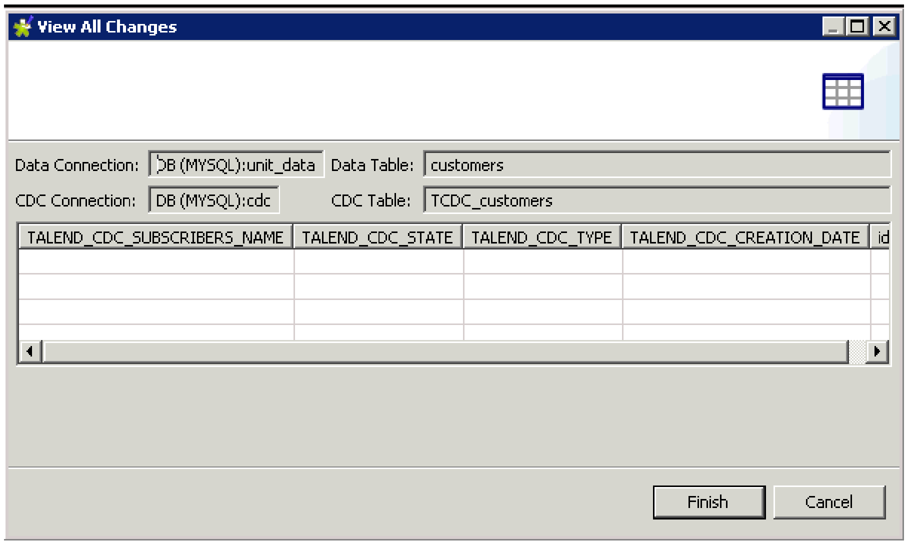
Data Preview: tMysqlOutput_3

Result Data Preview

Rows/page: 30 Limits: 1000

	id	firstname	name	age
Null Condition	*	*	*	*
1	2	Pat	Connors	34
2	3	Benjamin	Polk	13
3	4	William	Adams	41
4	5	Grover	Madison	4
5	6	Theodore	Adams	74
6	7	Rutherford	Reagan	59
7	8	Dwight	Roosevelt	15
8	9	Abraham	Buchanan	56
9	10	Benjamin	Carter	47
10	11	New	Person	40

4. Click **Close**.
5. In the **Repository**, under the **unit_data** database connection, right-click **customers** under **Table schemas** and then click **View All Changes** to see that the change records are gone now that they have been committed:



Data Connection: DB (MYSQL):unit_data Data Table: customers

CDC Connection: DB (MYSQL):cdc CDC Table: TCDC_customers

TALEND_CDC_SUBSCRIBERS_NAME	TALEND_CDC_STATE	TALEND_CDC_TYPE	TALEND_CDC_CREATION_DATE	id

Note: If your results differ in any way, you will need to follow the procedure in Resetting the Databases before attempting to make corrections.

You have now completed this lesson and can move on to the Challenge.

Challenge

Overview

Complete these exercises to further explore the use of Change Data Capture. See Solution for possible solutions to these exercises.

Multiple Changes

Create and then run a new Job that modifies the record with id 8 in **unit_data** three separate times: first to change the age to 24, then to change the first name to "Daphne", then to change the age again to 28. Use MySQL Workbench to examine the change records in the **cdc** database and then run the Job **SyncWarehouse**. How many records are processed? Ensure that the end result is what you expect.

Insert then delete

Create and then run a new Job that inserts a new record into the **unit_data customers** table, modifies it, and then deletes it. (You may find it easier to begin by duplicating the Job **DBMods**.) Use MySQL Workbench to examine the change records in the **cdc** database.

Duplicate the Job **SyncWarehouse.** Modify the output of the tMap component and the actions in each tMysqlOutput component to:

- insert the new record
- modify the record
- delete the record

Run the Job. How many records are processed? Ensure that the end result is what you expect.

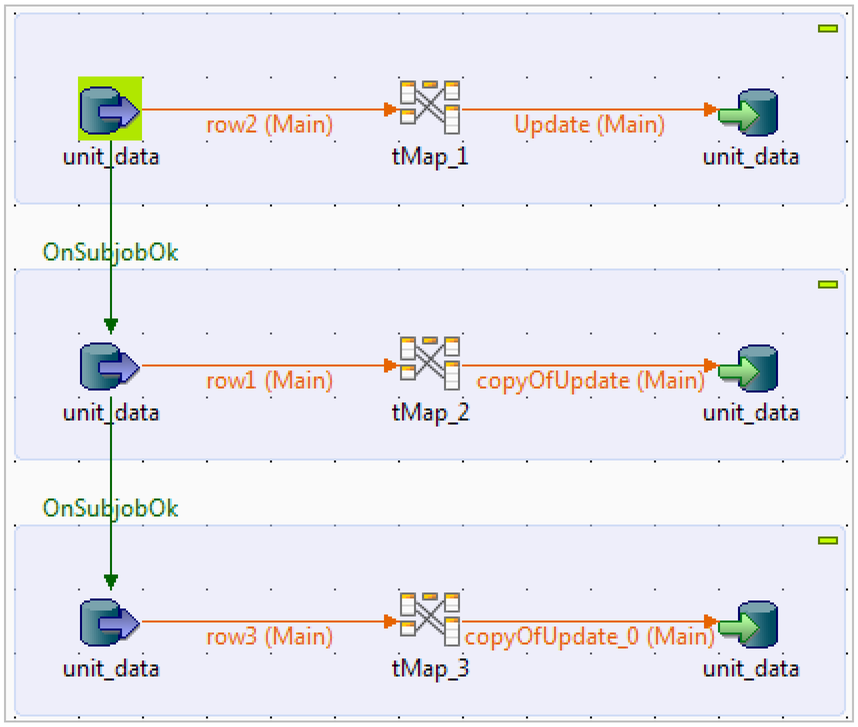
Solutions

Overview

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

Multiple Changes

The following Job contains three subjobs, each making one of the modifications:

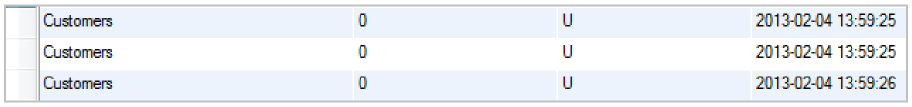


Each tMysqlInpout component is configured with the query:

"SELECT * FROM customers WHERE id = '8'"

Each subjob is a variation on the center subjob in the Job **DBMods**, with the **tMap** component configured to make each individual modification while mapping the remaining columns as-is.

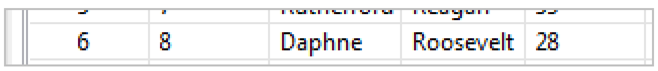
After running this Job, MySQL Workbench (or **View All Changes** from the **Repository**) shows three records in the **tcdc_customers** table :



Customers	0	U	2013-02-04 13:59:25
Customers	0	U	2013-02-04 13:59:25
Customers	0	U	2013-02-04 13:59:26

Running the Job **SyncWarehouse** processes a single update. Even though the record changed three times, the Job only needed to update the warehouse with the final, current state of the record.

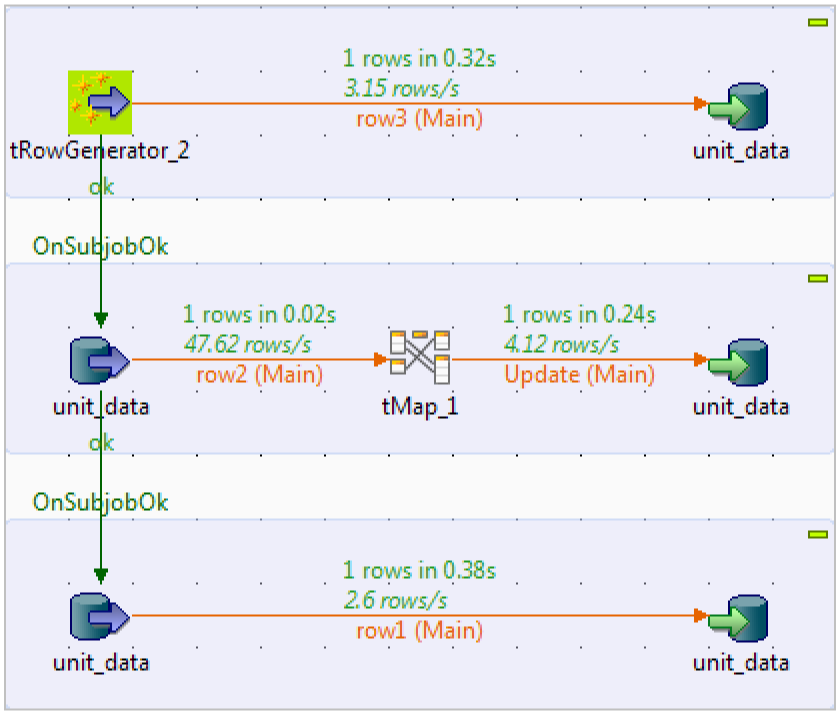
The data viewer for **warehouse** shows that the affected record has the expected changes:



6	8	Daphne	Roosevelt	28

Insert then delete

The following Job inserts, modifies, and then deletes the same record:



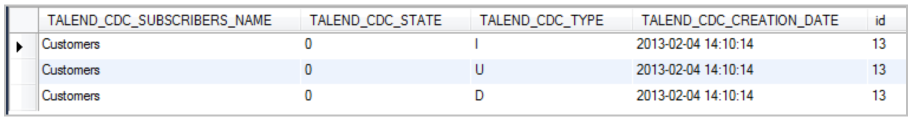
This is basically a re-ordered variation on the Job **DBMods**, with the insert subjob first.

The **tRowGenerator** component creates a record with id 12.

The two **tMysqlInput** components use the query:

"SELECT * FROM customers WHERE id = '12'"

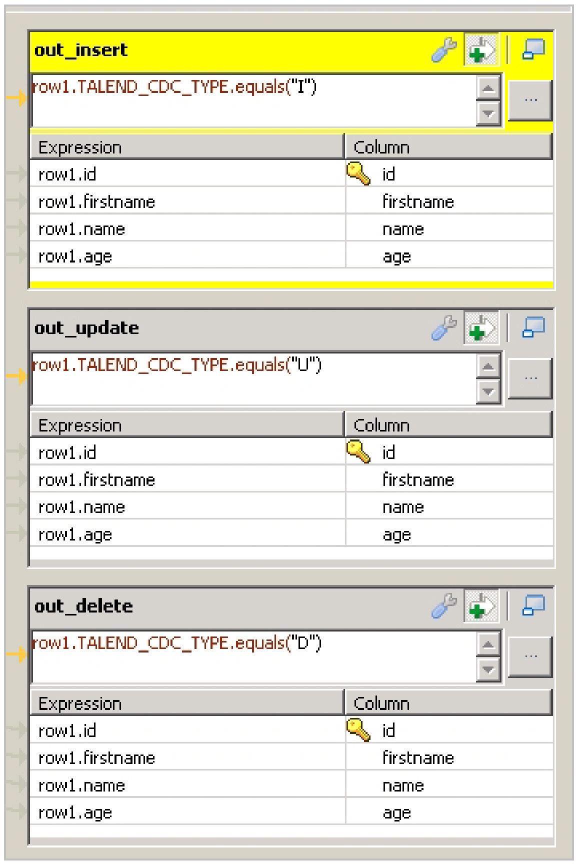
After running this Job, MySQL Workbench (or **View All Changes** from the **Repository**) shows three records in the **tcdc_customers** table:



TALEND_CDC_SUBSCRIBERS_NAME	TALEND_CDC_STATE	TALEND_CDC_TYPE	TALEND_CDC_CREATION_DATE	id
Customers	0	I	2013-02-04 14:10:14	13
Customers	0	U	2013-02-04 14:10:14	13
Customers	0	D	2013-02-04 14:10:14	13

Duplicate the Job **SyncWarehouse**.

Double-click tMap to open the editor and then modify the filters and the output names to filter "I", "U" and "D" values in the modifications list saved in the tcdc_customers table.



out_insert

```
row1.TALEND_CDC_TYPE.equals("I")
```

Expression	Column
row1.id	id
row1.firstname	firstname
row1.name	name
row1.age	age

out_update

```
row1.TALEND_CDC_TYPE.equals("U")
```

Expression	Column
row1.id	id
row1.firstname	firstname
row1.name	name
row1.age	age

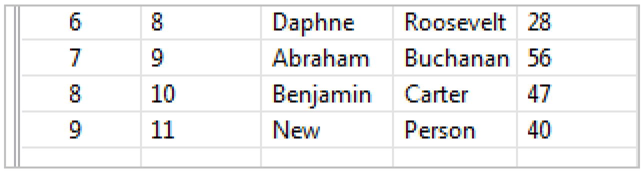
out_delete

```
row1.TALEND_CDC_TYPE.equals("D")
```

Expression	Column
row1.id	id
row1.firstname	firstname
row1.name	name
row1.age	age

Double-click the first tMysqlOutput component to open the Component view. Change the Action on Data to Insert. In the second tMysqlOutput component, change the Action on Data to Update. And set the Action on Data to Delete for the last tMysqlOutput component.

Run your Job, and check that the Job processes one of each type of change. The end result as displayed in the data viewer is no change to the table contents:



6	8	Daphne	Roosevelt	28
7	9	Abraham	Buchanan	56
8	10	Benjamin	Carter	47
9	11	New	Person	40

Wrap-Up

Recap

In this lesson, you started with two databases containing exactly the same records in matching tables. You used the Data Viewer and stored Queries to examine the contents of database tables from within Talend Studio. You then configured a third database to act as the CDC Foundation for one of the databases, tracking any changes made to the table in **unit_data**. After creating a Job to make some changes to the table, you finally built a Job that used the information in the CDC database table to update the **warehouse** database with only the changes.

Further Reading

For more information about topics covered in this lesson, see the *Talend Data Integration Studio User Guide* and the *Talend Components Reference Guide*.

Resetting the Databases

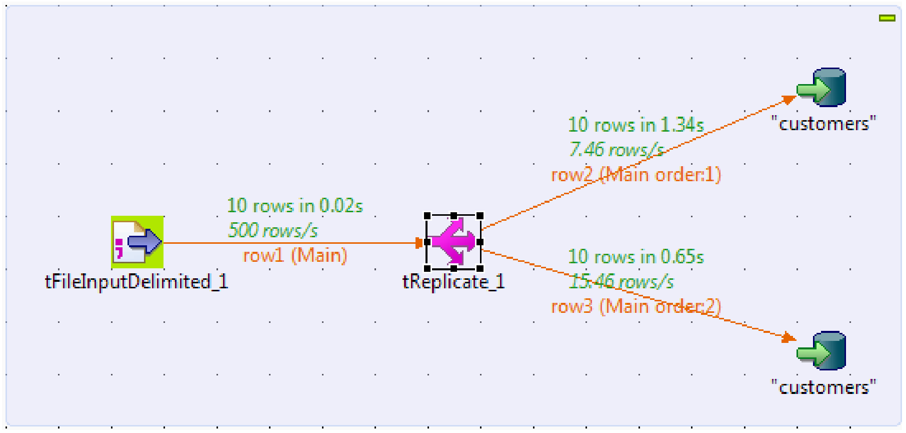
Overview

Because this lesson involves tracking changes to a database, a simple mistake can make it difficult to complete the exercises—both the mistake and your corrections are tracked in the CDC database. The steps listed here allow you to reset the databases to their original state and clear out the information in the CDC database.

Reset Table Contents

If you already have a ResetDatabases Job in your project, delete it. Then, import the **ResetDatabases** Job from the archive **Job Designs.zip**, in the **C:\Solutions** folder.

Open and run the Job **ResetDatabases** (click **OK** when the **Update Detection** window appears):

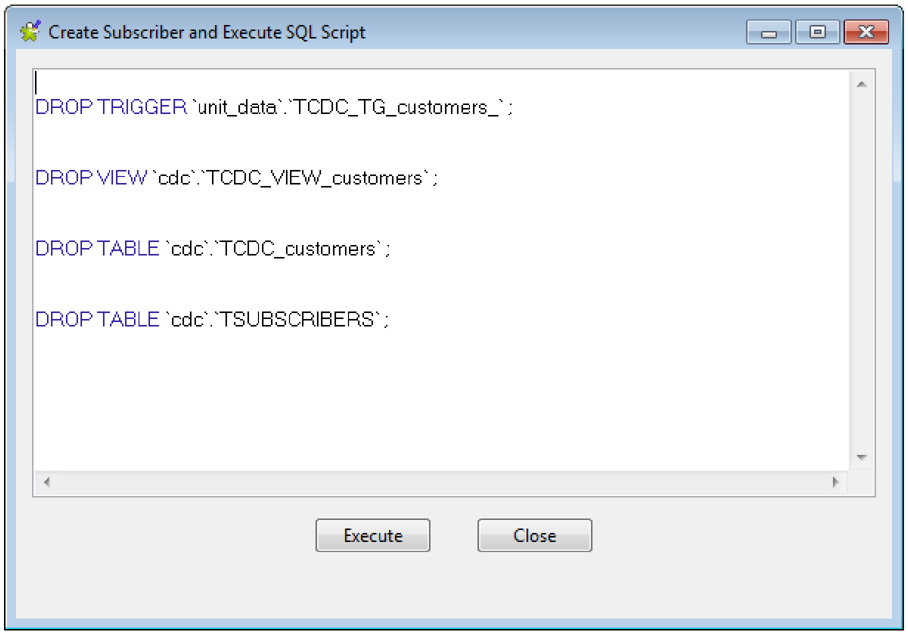


This Job drops the current contents of the **customers** table in both the **unit_data** and **warehouse** databases and loads them with the original data.

Reset CDC

Because you have CDC tracking on, the changes made in the previous step are now in the **cdc** database. To reset the **cdc** table, you need to drop it and then re-create it by following these steps:

1. In the **Repository**, under the **unit_data** database connection, right-click **cdc** under **CDC foundation** and then click **Delete CDC**:



```
DROP TRIGGER `unit_data`.`TCDC_TG_customers_` ;

DROP VIEW `cdc`.`TCDC_VIEW_customers` ;

DROP TABLE `cdc`.`TCDC_customers` ;

DROP TABLE `cdc`.`TSUBSCRIBERS` ;
```

2. Click **Execute**.
3. Click **Ignore** in the warning that appears.
4. Click **OK** in the success window, and then click **Close**.
5. Repeat the steps in Configuring the CDC Database.

Continue

If your error occurred while making changes to **unit_data**, carefully go over the steps in Monitoring Changes and make any necessary corrections before running the Job again and then continuing.

If your error occurred while updating **warehouse**, run the Job **DBMods** again to make the changes, then carefully go over the steps in Updating the Warehouse and make any necessary corrections before running the Job again.

Using Parallel Execution

Overview

Talend Data Integration provides several different mechanisms for using parallel execution to speed up certain kinds of Jobs. You can specify that subjobs within a Job be executed using multiple threads, use a specialized component to control parallel execution, and even specify parallel execution within certain database components. In this lesson, you will create Jobs that write large amounts of data to files or a database and then compare the execution times when run in sequence or in parallel. You will also perform multithreading using dedicated components or using the automatic multithreading feature of the Studio.

Objectives

After completing this tutorial, you will be able to:

- Configure a Job to use multi-threaded execution
- Configure an individual component to use parallel execution
- Use a Talend component to run subjobs in parallel
- Use Talend components to split your data in multiple threads and then perform multi-threaded execution.

Before You Begin

Be sure you are working in an environment that contains the following:

- A properly installed copy of Talend Data Integration
- The supporting files for this lesson
- Multiple processors or cores

The first step is to create a Job that consumes a significant amount of resources.

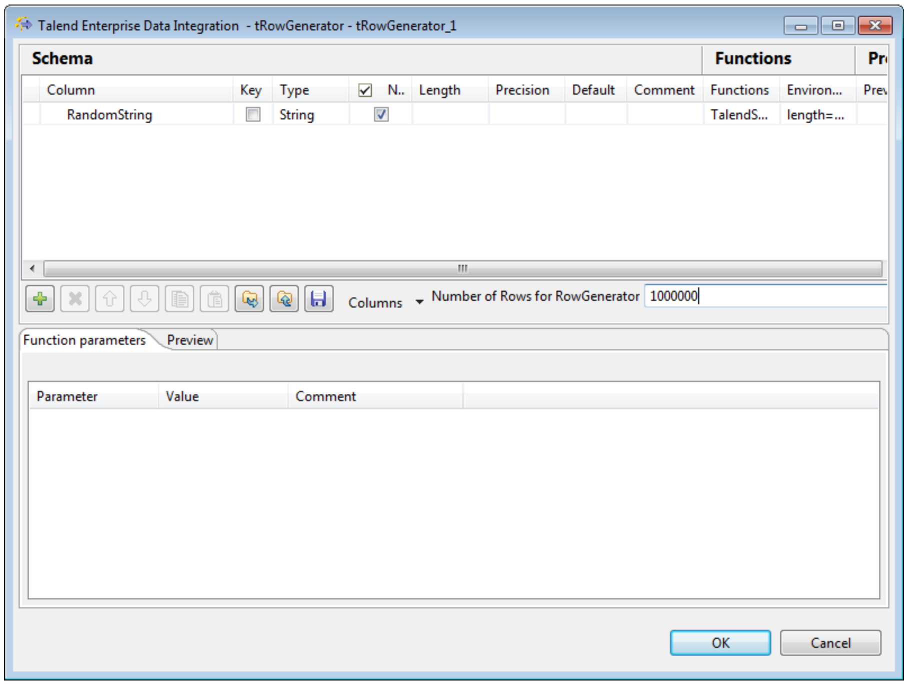
Writing Large Files

Overview

To demonstrate the different methods of executing Talend Jobs in parallel, you will be creating a Job that writes a million rows of random data to multiple files.

Create the Job

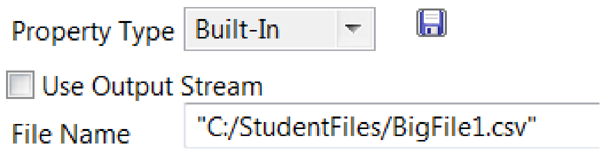
1. Create a new Job, naming it "BigFiles".
2. Add a **tRowGenerator** component to the design workspace and then configure it to create one million rows, each with a single column containing a randomly generated string:


Talend Enterprise Data Integration - tRowGenerator - tRowGenerator_1
Schema
Functions
Function parameters
Preview
OK
Cancel

Column	Key	Type	N..	Length	Precision	Default	Comment	Functions	Environ...
RandomString	☐	String	✓					TalendS...	length=...

Number of Rows for RowGenerator 1000000

3. Add a **tFileOutputDelimited** component and then connect the two components with a **Main** Row.
4. Configure the output component to write to the file "C:/StudentFiles/BigFile1.csv":

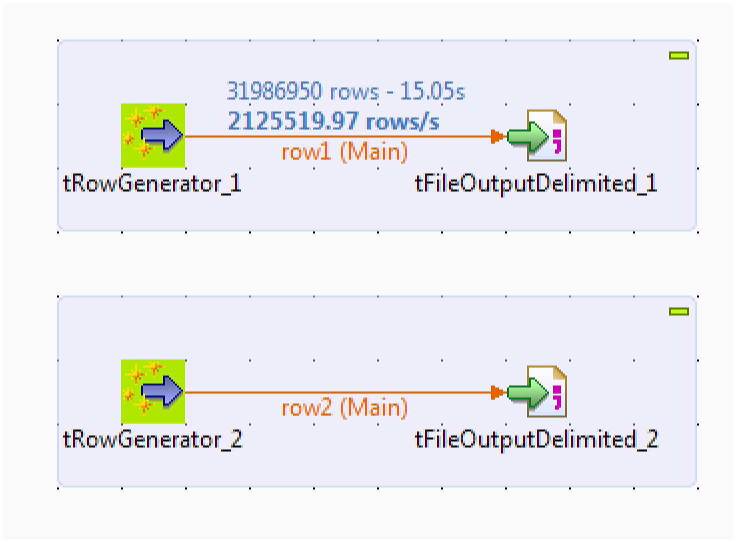


Property Type: Built-In
☐ Use Output Stream
File Name: "C:/StudentFiles/BigFile1.csv"

5. Copy the entire subjob and paste it so that the Job has two identical subjobs.
6. Configure the output component in the second subjob to write to a file named

"BigFile2.csv".

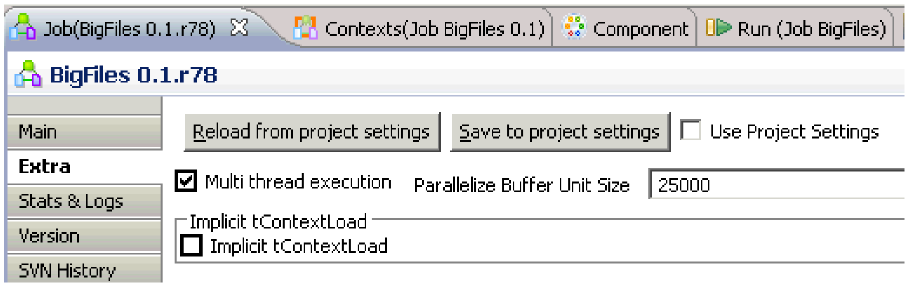
7. Run the Job. Note that the first subjob executes first, followed by the second:



By default, the subjobs execute in the order in which they were created.

Run the Subjobs in Parallel

1. Click the **Job** view.
2. Click the **Extra** tab, clear Use Project Settings, and then select **Multi thread execution**:



☐ Use Project Settings

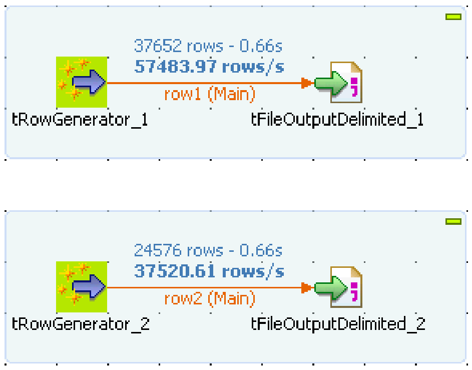
☑ Multi thread execution

Parallelize Buffer Unit Size 25000

Implicit tContextLoad

☐ Implicit tContextLoad

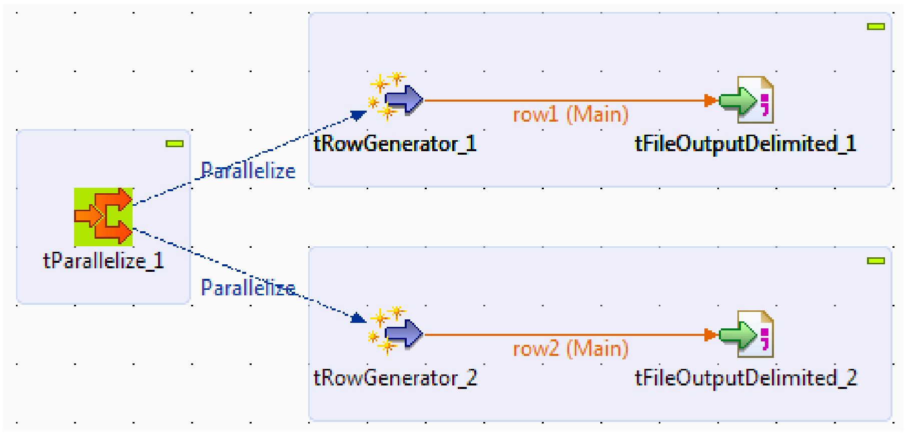
3. Run the Job again, noting that both subjobs run simultaneously:



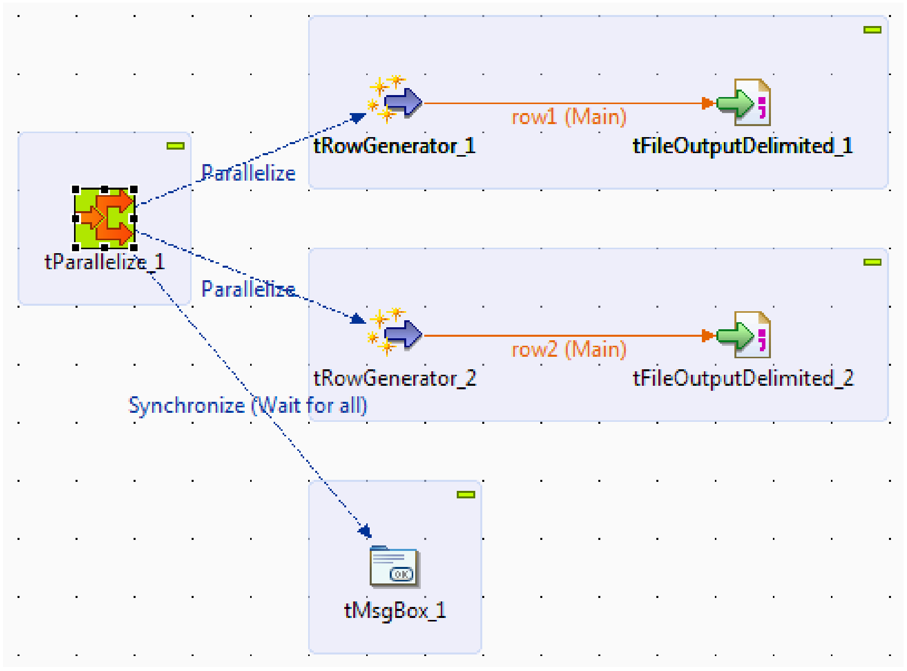
Note that on a single-processor system, turning on multi-thread execution might actually increase the total execution time.

Use the tParallelize Component

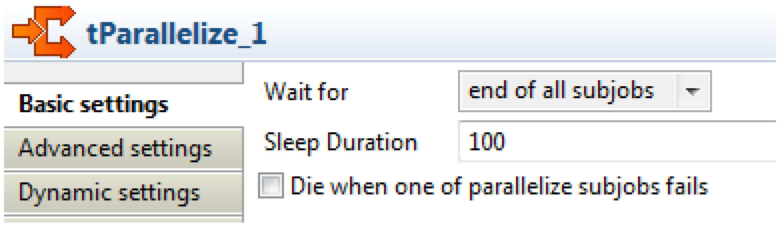
1. Add a **tParallelize** component to the left of the two subjobs.
2. Connect the component to each subjob with a **Parallelize** trigger:



3. In the **Job** view, clear the **Multi thread execution** check box.
4. Run the Job. Again the subjobs run in parallel. The advantage of the **tParallelize** component is that it allows you to run several subjobs in parallel and then synchronize another subjob to execute after completion.
5. Add a **tMsgBox** component to the bottom of the design workspace and then configure it to display the message "Files written".
6. Connect the **tParallelize** component to the **tMsgBox** component with a **Synchronize** trigger:



7. Run the Job again. Note that the message box appears after both subjobs complete. Obviously, the message box in this example is not terribly useful, but sending a notification after a very lengthy execution might be.
8. Examine the **Component** view of the **tParallelize** component:



Wait for: end of all subjobs

Sleep Duration: 100

☐ Die when one of parallelize subjobs fails

Note that you can specify whether the synchronization subjob executes after the first parallel subjob finishes or after all subjobs complete (the default), how long to wait between checks to see if the parallel subjobs are complete, and whether or not the entire Job should exit if one of the subjobs encounters an error.

Now that you have seen how to perform parallel execution with subjobs, you can move on to performing parallel execution while working with databases.

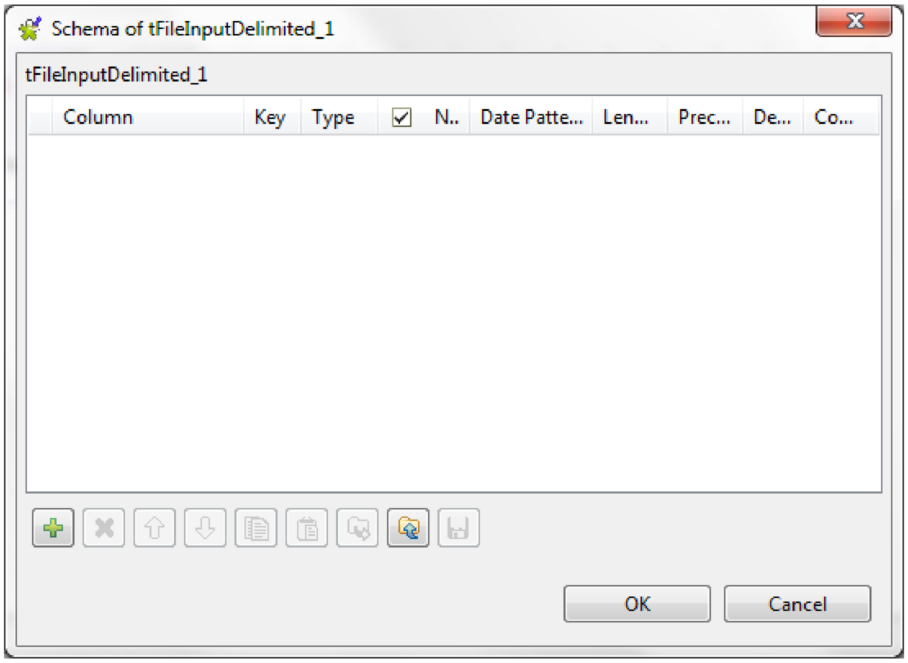
Writing to Databases

Overview

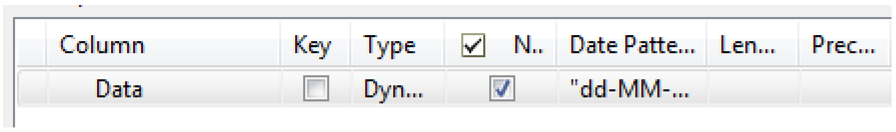
In the previous exercise, you ran subjobs in parallel. Many Talend components can be configured to execute in parallel, particularly those that write to databases. For a complete list, see the *Talend Component Reference Guide*. In this exercise, you will create a Job that writes a large number of rows to a database and then compare execution times without and then with parallel execution.

Create the Job

1. Create a new Job, naming it "DBWrite".
2. Add a **tFileInputDelimited** component to the design workspace.
3. Configure the component to read the file "C:/StudentFiles/CustData.csv" and enter "1" into the **Header** box.
4. Click the **Edit schema** button:



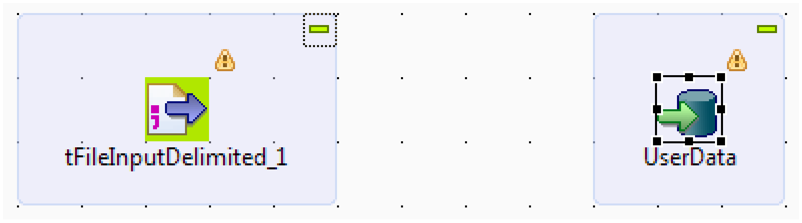
5. Add a single column, naming it "Data" and clicking **Dynamic** as the **Type**:



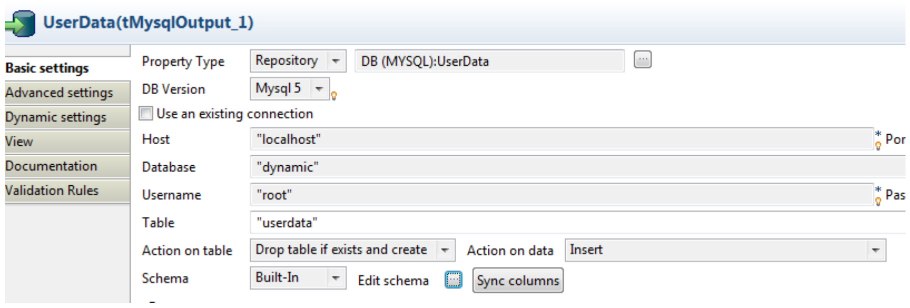
Column	Key	Type	✓ N..	Date Patte...	Len...	Prec...
Data	☐	Dyn...	✓	"dd-MM-...		

Dynamic is an opaque data type allowing you to pass data through without knowing the actual columns in the file or database. It will capture all columns not explicitly named.

6. Drag the **UserData** database connection metadata from the **Repository** to the design workspace, clicking **tMysqlOutput** as the component:



7. Connect the two components with a **Main** Row.
8. Configure the **tMysqlOutput** component, entering "userdata" into the **Table** box and clicking **Drop table if exists and create** in the **Action on table** list:

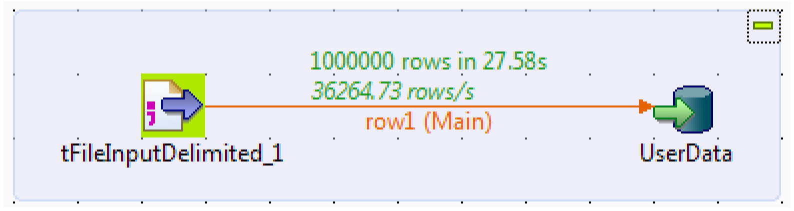


UserData(tMysqlOutput_1)

Basic settings
Advanced settings
Dynamic settings
View
Documentation
Validation Rules

Property Type	Repository	DB (MYSQL):UserData	
DB Version	Mysql 5		
Use an existing connection			
Host	"localhost"	Por	
Database	"dynamic"		
Username	"root"	Pas	
Table	"userdata"		
Action on table	Drop table if exists and create	Action on data	Insert
Schema	Built-In	Edit schema	Sync columns

9. Run the Job and then make a note of the total time the Job took to execute:



In this example, the Job took 27.58 seconds.

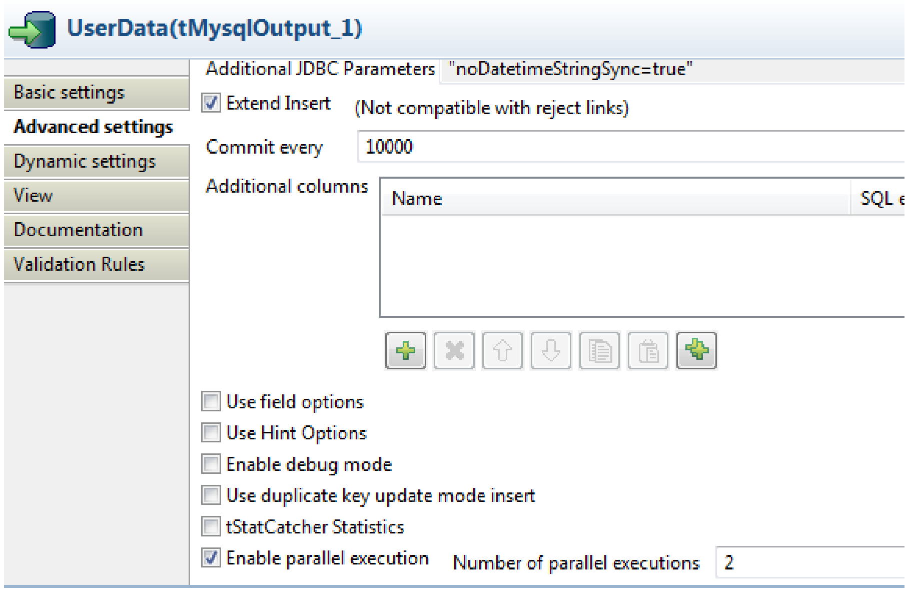
10. Use MySQL Workbench to examine the database by double-clicking **dynamic** in the **Object Browser** and then executing the following SQL statement:

```
select * from userdata;
```

Note that the database table has three columns: the table is constructed correctly from the column definitions in the input file even though you did not specify the schema anywhere in your Job.

Configure Parallel Execution

1. In the **Component** view for the **tFileInputDelimited** component, change the Schema from **Built-In** to **Repository** and then specify the generic schema metadata under **UserData** (you need to retrieve the schema first). When prompted to propagate changes, click **Yes**.
2. In the **Component** view for the **tMysqlOutput** component, click the **Advanced settings** tab and then select the **Enable parallel execution** check box:



UserData(tMysqlOutput_1)

Additional JDBC Parameters "noDatetimeStringSync=true"

✓ Extend Insert (Not compatible with reject links)

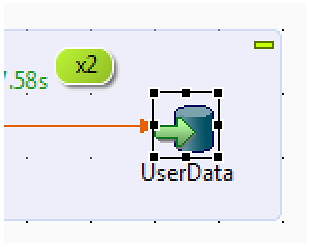
Commit every 10000

Additional columns

Name	SQL e

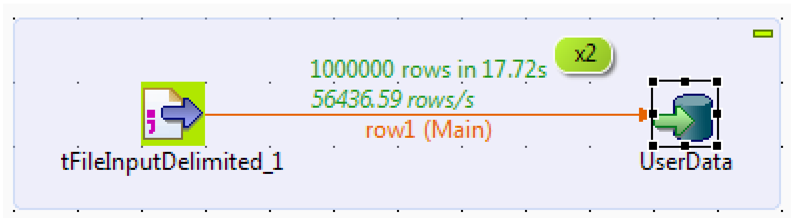
☐ Use field options
☐ Use Hint Options
☐ Enable debug mode
☐ Use duplicate key update mode insert
☐ tStatCatcher Statistics
✓ Enable parallel execution Number of parallel executions 2

3. Note the component display in the design workspace:



The "x2" indicates that the component will use two processors or cores for the Job. When doing this in your home environment, enter a number into the **Number of parallel executions** box that corresponds to the number of processors or cores you want to use for the Job.

4. Run the Job again, noting the execution time:



As shown in this example, the amount of time has dropped due to the parallel execution.

You can continue exploring parallel executions with the next labs that will show you how to use dedicated components to partition your data in multiple threads or how to enable automatically parallel execution.

Partitioning

Another way to parallelize executions is to use components such as **tPartitioner** and **tCollector**. You will build a Job that generates an important amount of data, sort the data and then write the sorted result in a file.

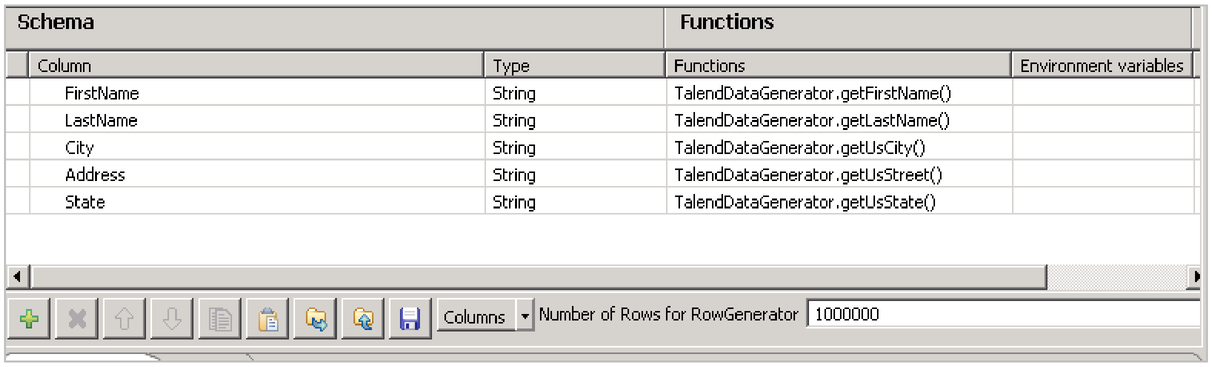
Generate data

First, you will generate 1 million lines of random data using the tRowGenerator.

1. Create a new Job and name it **JobPartitioner**.
2. Add a **tRowGenerator** component and open the **Component** view.
3. Using the **green plus sign**, add 5 columns and name them:
    - FirstName
    - LastName
    - City
    - Address
    - State

    The type of each column is String and you will configure your tRowGenerator to generate random first and last names, random city, address and State.
4. Edit the **FirstName** column **Function** and select **TalendDataGenerator.getFirstName()** in the list.
5. Edit the **LastName** column **Function** and select **TalendDataGenerator.getLastName()** in the list.
6. Edit the **City** column **Function** and select **TalendDataGenerator.getUsCity()** in the list.
7. Edit the **Address** column **Function** and select **TalendDataGenerator.getUsStreet()** in the list.
8. Edit the **State** column **Function** and select **TalendDataGenerator.getUsState()** in the list.

9. In the **Number of Rows** for RowGenerator field, enter 1000000.
   Your configuration should be similar to the following:



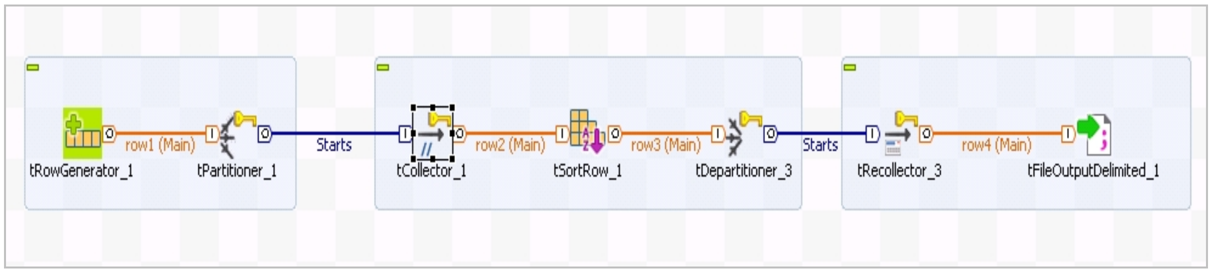
Column	Type	Functions	Environment variables
FirstName	String	TalendDataGenerator.getFirstName()	
LastName	String	TalendDataGenerator.getLastName()	
City	String	TalendDataGenerator.getUsCity()	
Address	String	TalendDataGenerator.getUsStreet()	
State	String	TalendDataGenerator.getUsState()	

Number of Rows for RowGenerator: 1000000

10. Click **OK** to save your configuration and close the window.

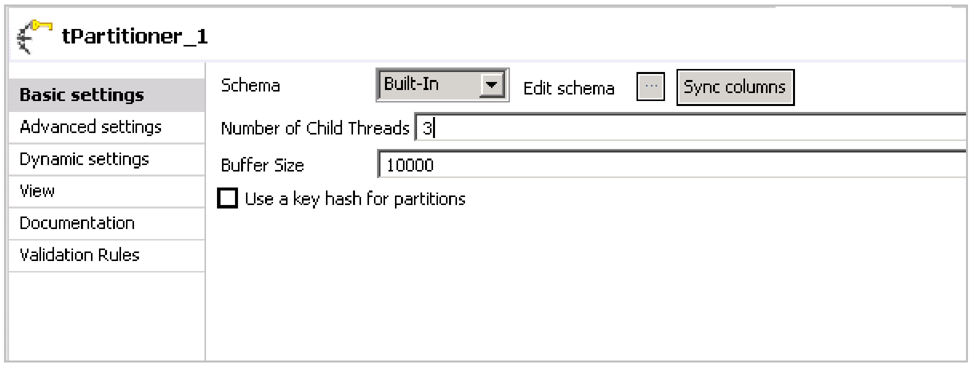
Partition and collect data

The tPartitioner component dispatches the input records into a specific number of threads. You will use this component to start the parallel execution in your Job. Next to the tPartitioner, you will use a tCollector component. This component sends the threads, generated with tPartitioner, to its following components for parallel execution. At the end of this lab, your Job should be similar to the following:



1. Add a **tPartitioner** component to the right side of **tRowGenerator** and connect it with a **Main** row.
2. Double-click **tPartitioner** to open the **Component** view.
   This is where you will specify the number of threads.

3. In the **Number of Child Threads** box, enter **3**.



tPartitioner_1

Basic settings | Advanced settings | Dynamic settings | View | Documentation | Validation Rules

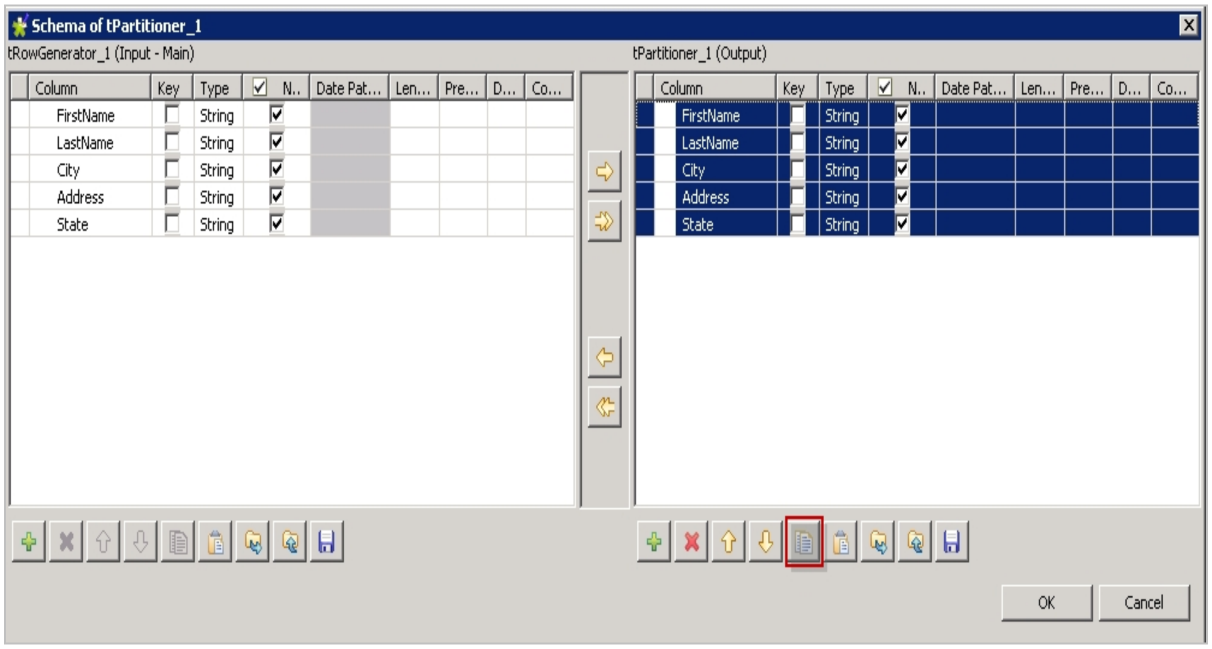
Schema: Built-In — Edit schema — Sync columns

Number of Child Threads: 3

Buffer Size: 10000

☐ Use a key hash for partitions

4. Edit the schema clicking the **(...)** button.
5. Select the five columns and then click the button to copy the schema:



Schema of tPartitioner_1

tRowGenerator_1 (Input - Main)

Column	Key	Type	N..	Date Pat...	Len...	Pre...	D...	Co...
FirstName	☐	String	✓					
LastName	☐	String	✓					
City	☐	String	✓					
Address	☐	String	✓					
State	☐	String	✓					

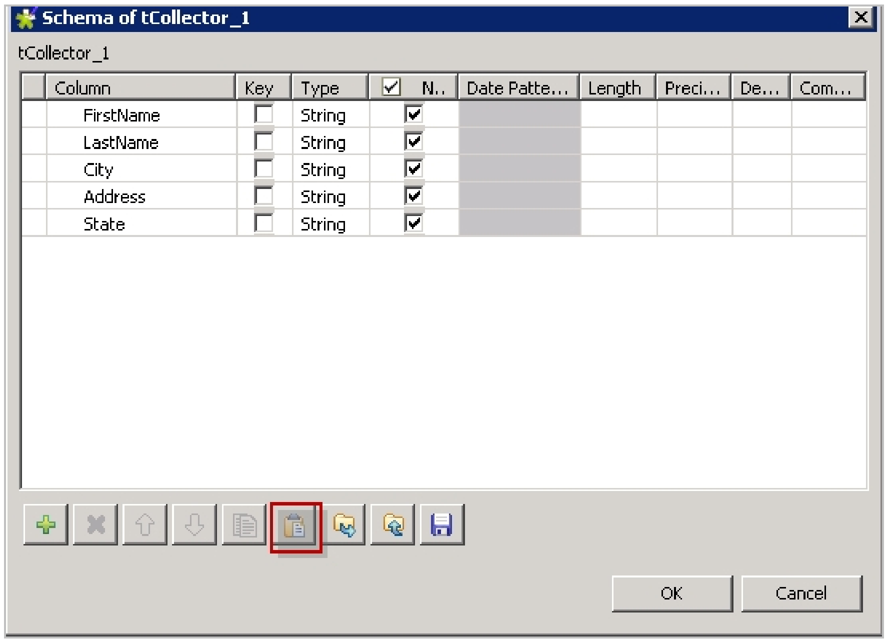
tPartitioner_1 (Output)

Column	Key	Type	N..	Date Pat...	Len...	Pre...	D...	Co...
FirstName	☐	String	✓					
LastName	☐	String	✓					
City	☐	String	✓					
Address	☐	String	✓					
State	☐	String	✓					

Click **OK** to close the schema window.

6. Add a **tCollector** component.
7. Connect **tPartitioner** to **tCollector** with the **Starts Trigger** and open the **Component** view of tCollector.

8. Edit the schema and click the button to paste the schema copied in tPartitioner.



Schema of tCollector_1

tCollector_1

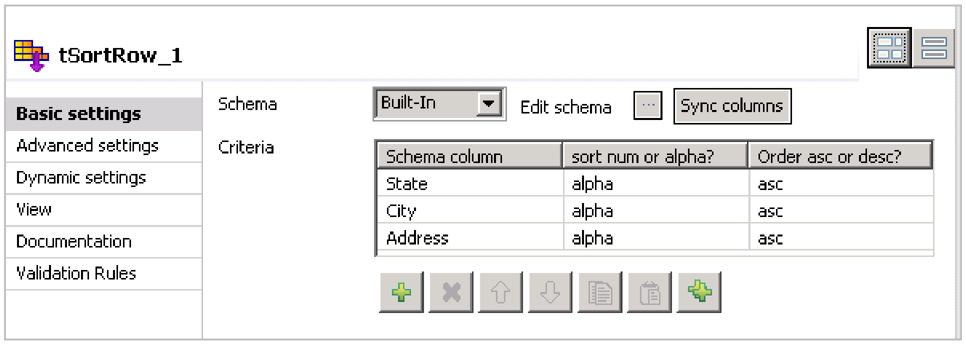
Column	Key	Type	N..	Date Patte...	Length	Preci...	De...	Com...
FirstName	☐	String	✓					
LastName	☐	String	✓					
City	☐	String	✓					
Address	☐	String	✓					
State	☐	String	✓					

Click **OK** to close the schema window.

Sort data

The data will be sorted according to the State, City and Address columns alphabetic order. You will use the tSortRow component to achieve this. As a huge amount of data will be sorted, you will use advanced settings of tSortRow to allow sort on disk and to allow temporary data to be saved.

1. Add a **tSortRow** component. Connect **tPartitioner** to **tSortRow** with a **Main** Row.
2. Open the **Component** view.
3. Use the **green plus sign** to add 3 lines in the **Criteria** table and configure them to sort on State, City and Address columns, in alphabetic order. Your configuration should look like this:



tSortRow_1

Schema: Built-In

Criteria

Schema column	sort num or alpha?	Order asc or desc?
State	alpha	asc
City	alpha	asc
Address	alpha	asc

4. Click the **Advanced settings** tab.
5. Select **Sort on disk**.

   Some more lines will appear in the window. You will be able to set the path of the temporary data directory. As multiple threads have been created by the tPartitioner component, multiple temporary data directory will be needed to avoid overwriting occurs during the Job execution.

To create as many folders as threads, the name of the folder will be created using the thread ID available in a context variable.

6. In the **Temp data directory path** box, enter:
   ```
   "C:/Studentfiles/temp"+
   ```
7. Still in the **Temp data directory path** box, press **Ctrl+Space** to get the following variable : **tCollector_1_THREAD_ID**.
   You should get the following line in the Temp data directory path field:
   ```
   "C:/StudentFiles/temp"+((Integer)globalMap.get("tCol-
   lector_1_THREAD_ID"))
   ```

Finalize Job

Your data are now sorted, but they are still splitted in multiple threads. You will have to collect them before writing them in a file. You will use tDepartitioner and tRecollector components to finalize your Job.

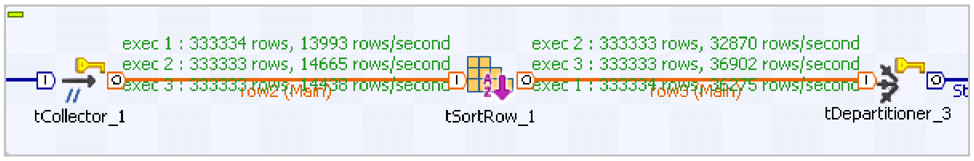
The tDepartitioner component regroups the outputs of the processed parallel threads and then, the tRecollector component captures the output of a given tDepartitioner component and sends the captured data to the next component.

1. Add a **tDepartitioner** component and then, connect **tSortRow** to **tDepartitioner** with a **Main** row.
2. Open the **Component** view of **tDepartitioner** and check that the schema has been updated.
3. Add a **tRecollector** component and then, connect **tDepartitioner** to **tRecollector** with the **Starts Trigger**.
4. Open the **Component** view of **tRecollector** and click **(...)** to edit the **Schema**.
5. Paste the schema copied earlier in tPartitioner.
   The last step is to write your sorted data in a file.
6. Add a **tFileOutputDelimited** component and then connect **tRecollector** to **tFileOutputDelimited** with a **Main** row.
7. Open the **Component** view of **tFileOutputDelimited** and then, in the **File Name** box, enter:
   ```
   "C:/StudentFiles/SortedData.csv"
   ```

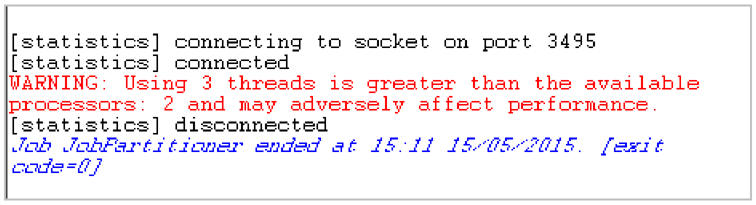
Your Job is now ready to run.

Run the Job

In the Run view, click Run and then observe the statistics in he Designer view.
As the Job was built to use 3 threads, you will see the number of rows processed on each thread. You should see one third of the 1 million initial number of rows, processed on each thread:


exec 1 : 333334 rows, 13993 rows/second
exec 2 : 333333 rows, 14665 rows/second
exec 3 : 333333 rows, 14438 rows/second
row2 (Main)
exec 2 : 333333 rows, 32870 rows/second
exec 3 : 333333 rows, 36902 rows/second
exec 1 : 333334 rows, 35275 rows/second
row3 (Main)
tCollector_1
tSortRow_1
tDepartitioner_3

You may see a Warning in the Console if the number of threads is higher than the number of available processors. In fact, if you are asking for more threads than processors, this may lead to a performance degradation and you will loose the advantage of multi-threading:



```
[statistics] connecting to socket on port 3495
[statistics] connected
WARNING: Using 3 threads is greater than the available
processors: 2 and may adversely affect performance.
[statistics] disconnected
Job JobPartitioner ended at 15:11 15/05/2015. [exit
code=0]
```

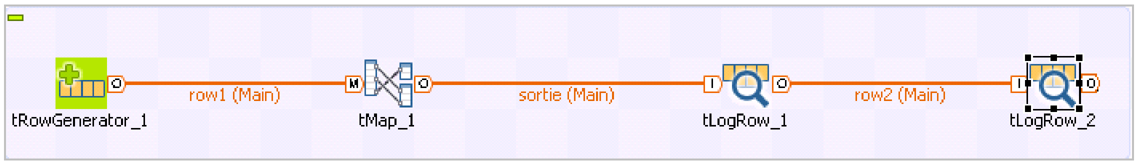
Now it's time to move to the last Job on parallelization which will show you how to enable parallelization automatically.

Automatic Parallelization

The last lab will show you how to enable automatic parallelization in your Jobs without the use of parallelization components. It is recommended to use the automatic approach compared to the use of components.

When you have to develop a Job to process a huge amount of data, you can enable or disable parallelization by one single click, and the Studio automates the implementation across the Job.

At the end of the Lab, your Job should be similar to this:



Generate data

You will create a Job that will generate data. Then, you will enrich the data with a tMap component and display the rows with 2 tLogRow components.

1. Create a **new Job** and name it **AutoParallel**. Add a Description and Purpose.
2. Add a **tRowGenerator** in the **Designer**.

   The tRowGenerator will create 100 rows of data composed of 3 columns : Id, FirstName and LastName
3. Double-click **tRowGenerator**.
4. Use the **green plus sign** to add 3 columns to the schema and then, name the 3 columns: **Id**, **FirstName** and **LastName**.
5. Configure the **Id** column using an **Integer Type** and in the **Functions** list, select:

   `Numeric.sequence(String,int,int)`

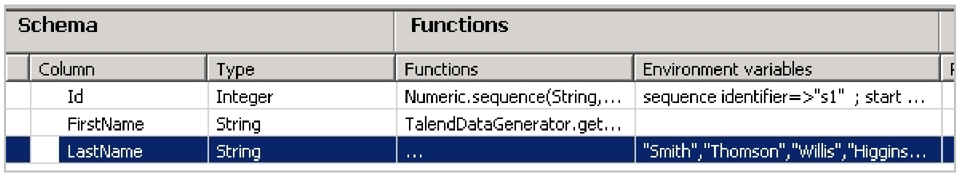
   You can use the default parameters, to generate the Ids.
6. Configure the **FirstName** column using a **String Type** and in the **Functions** list select:

   `TalendDataGenerator.getFirstName()`
7. Configure the **LastName** column using a **String Type** and in the **Functions** list select **(...)**.

   The last name value will be chosen in a list of predefined names.
8. Click the **LastName** column in the **Schema** and then, in the **Function Parameters** tab, in the **Value** box, enter:

   `"Smith","Thomson","Willis","Higgins","Harris"`

Your configuration should be as follows:



Schema		Functions	
Column	Type	Functions	Environment variables
Id	Integer	Numeric.sequence(String,...	sequence identifier=>"s1" ; start ...
FirstName	String	TalendDataGenerator.get...	
LastName	String	...	"Smith","Thomson","Willis","Higgins...

Next, you will enrich the data with a tMap component.

Enrich and display data

The component will enrich the data with the Thread ID. This will help you to understand how data is partitioned between the different threads. The Thread ID is already available as a variable.

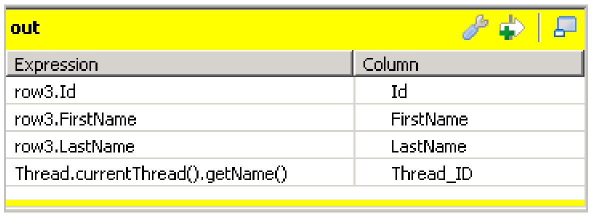
To display the data, you will use 2 tLogRow components. The first one will display the data processed by each thread and the last one will show you the data after merging the threads.

1. Add a **tMap** component on the right side of tRowGenerator. Connect **tRowGenerator** to **tMap** with the **Main** row.
2. Double-click **tMap** to open the editor.
3. Create a **new output** and name it **out**.
4. Drag **Id**, **FirstName** and **LastName** columns to the **out** table.
5. Use the **green plus sign** below the out schema table to add a **new column** and name it **Thread_ID**.
6. Click **(...)** to edit the Thread_ID **Expression** and enter the following in the **Expression** box:

   ```
   Thread.currentThread().getName()
   ```

   This is an existing variable in the Studio that provides the Thread ID.

   The configuration of out should be as follows:

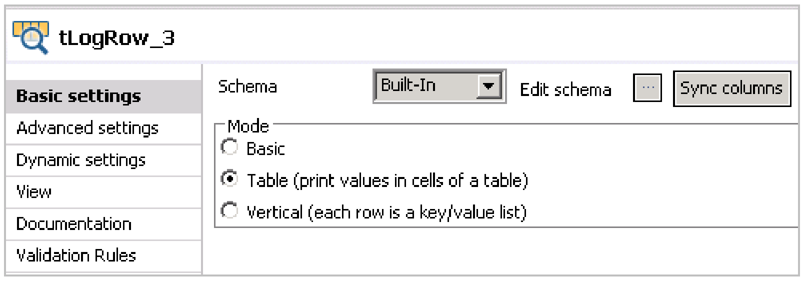
   

   out

Expression	Column
row3.Id	Id
row3.FirstName	FirstName
row3.LastName	LastName
Thread.currentThread().getName()	Thread_ID

7. Click **OK** to save the configuration.
8. Add a **tLogRow** component and connect it with the **out** Output of tMap.
9. Add a second **tLogRow** component and connect to the first tLogRow with the **Main** row.

10. Configure both **tLogRow** components in **Table Mode**:



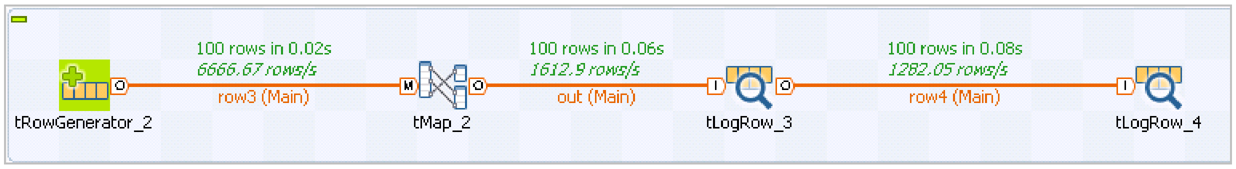
Schema: Built-In

Mode: ○ Basic [● Table (print values in cells of a table)] ○ Vertical (each row is a key/value list)

First Run

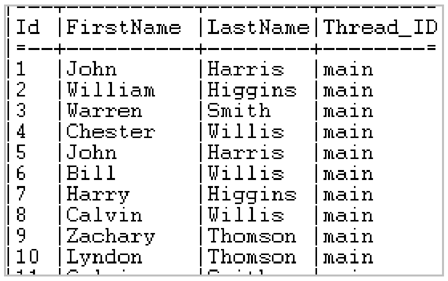
You will run your Job and observe the data displayed in the Console.

1. In the **Run View**, click **Run**.
2. Observe the statistics in the **Designer** view:



   As expected, 100 rows are processed.

3. Observe the results in the **Console**:



```
|Id |FirstName |LastName|Thread_ID
|=--+----------+--------+---------=
|1  |John      |Harris  |main
|2  |William   |Higgins |main
|3  |Warren    |Smith   |main
|4  |Chester   |Willis  |main
|5  |John      |Harris  |main
|6  |Bill      |Willis  |main
|7  |Harry     |Higgins |main
|8  |Calvin    |Willis  |main
|9  |Zachary   |Thomson |main
|10 |Lyndon    |Thomson |main
```

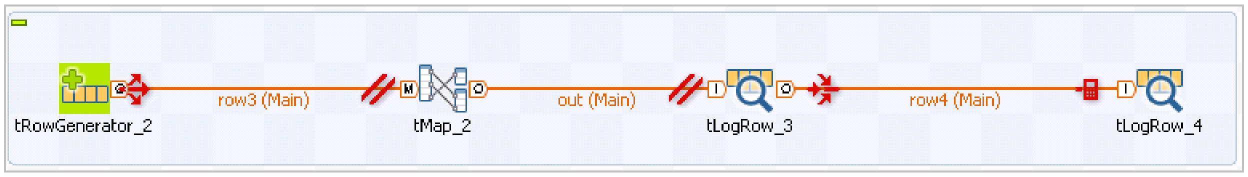
   As the parallelization is not enabled yet, the Thread ID is "Main" and both tLogRow components display the same data.

Enable Parallelization

Now you will enable the parallelization and execute the Job with the default parallelization parameters.

1. Right-click **tRowGenerator** and select **Set Parallelization** in the contextual menu.

   This will enable the parallelization in your Job, and add some red symbols in the designer.



The symbols correspond to the Partitioning, Collecting, Departitioning and Recollecting steps used in the previous Lab.

The first symbol ( ) corresponds to Partitioning. In this step, the data is split into a given number of thread.

The second symbol ( )corresponds to Collecting. In this step, the Studio collects the split threads and sends them to a given component.

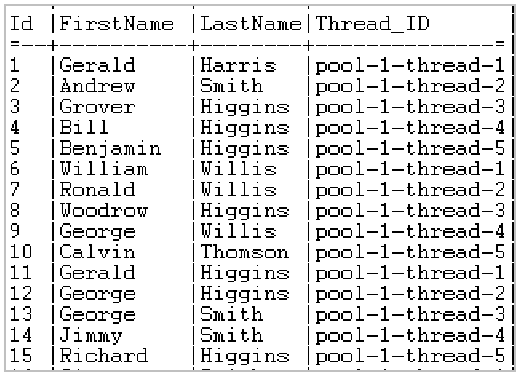
The third symbol ( ) corresponds to Departitioning. In this step, the Studio groups the outputs of the parallel executions of the split threads.

The last symbol ( ) corresponds to Recollecting. In this step, the Studio captures the grouped execution results and outputs them to a given component.

2. **Run** the Job and observe the output in the **Console**.

   For the first tLogRow component, 5 output tables are displayed, one per thread processed.

   For the second tLogRow component, there is only one output table which should be similar to this one:



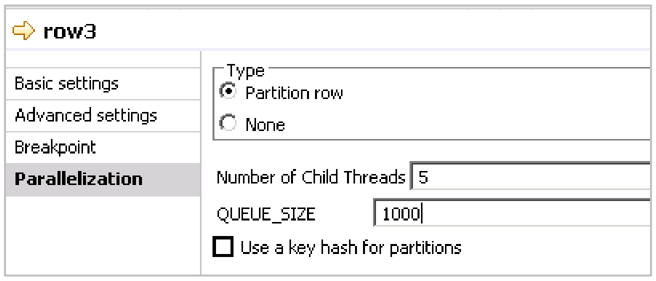
```
Id |FirstName |LastName|Thread_ID      |
=--+----------+--------+--------------=|
1  |Gerald    |Harris  |pool-1-thread-1|
2  |Andrew    |Smith   |pool-1-thread-2|
3  |Grover    |Higgins |pool-1-thread-3|
4  |Bill      |Higgins |pool-1-thread-4|
5  |Benjamin  |Higgins |pool-1-thread-5|
6  |William   |Willis  |pool-1-thread-1|
7  |Ronald    |Willis  |pool-1-thread-2|
8  |Woodrow   |Higgins |pool-1-thread-3|
9  |George    |Willis  |pool-1-thread-4|
10 |Calvin    |Thomson |pool-1-thread-5|
11 |Gerald    |Higgins |pool-1-thread-1|
12 |George    |Higgins |pool-1-thread-2|
13 |George    |Smith   |pool-1-thread-3|
14 |Jimmy     |Smith   |pool-1-thread-4|
15 |Richard   |Higgins |pool-1-thread-5|
```

3. The results are explained by the default values used for the parallelization.

   To see the default values applied, right-click the **output row of tRowGenerator**, then select **Settings** in the contextual menu.

   This will open the **Component** view.

4. Click the **Parallelization** tab to find the default Parallelization settings:



row3

Basic settings
Advanced settings
Breakpoint
Parallelization

Type
Partition row [✓]
None

Number of Child Threads: 5
QUEUE_SIZE: 1000
☐ Use a key hash for partitions

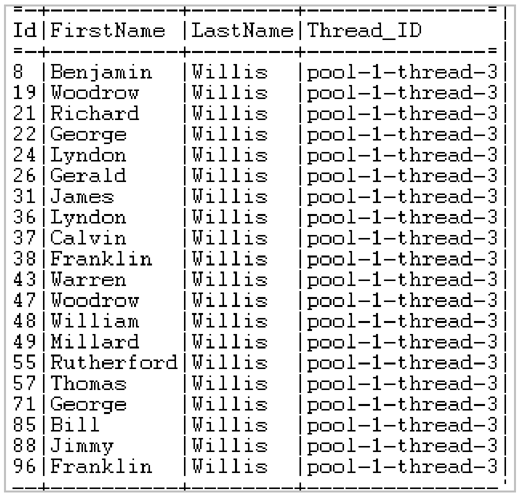
In the settings, the number of child threads is set to 5. That's why you had 5 output tables for the first tLogRow component.

And, as a default, the way the rows are attributed to each thread is Round-robin. This means that records are dispatched one-by-one to each thread, in a circular fashion, until the last record is dispatched. If you look at the output of the second tLogRow, you will observe that the Id 1 was assigned to Thread 1, Id 2 to Thread 2 etc...

Change Parallelization settings

Now, you will personalize the Parallelization settings.

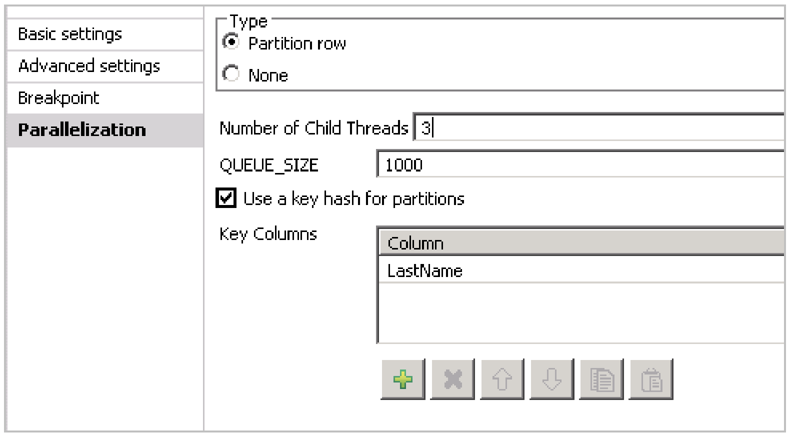
1. In the **Parallelization tab**, check the **Use a key hash for partitions** option.
   This means that the partition will be based on the values in the selected columns. All records that meet the same criteria are dispatched into the same thread.
2. Click the **green plus sign** below the **Key Columns** table then in the **Column** list, select **LastName**.
   The LastName are chosen in a predifined list of 5 names. So using this criteria should lead to have the rows dispatched on the 5 threads depending on the LastName value.
3. **Run** the Job and observe the result in the **Console**.
   The output of the first tLogRow is still 5 tables, but this time, you will find a unique LastName value per table:



```
=-+----------+--------+---------------=|
Id|FirstName |LastName|Thread_ID       |
=-+----------+--------+---------------=|
8 |Benjamin  |Willis  |pool-1-thread-3|
19|Woodrow   |Willis  |pool-1-thread-3|
21|Richard   |Willis  |pool-1-thread-3|
22|George    |Willis  |pool-1-thread-3|
24|Lyndon    |Willis  |pool-1-thread-3|
26|Gerald    |Willis  |pool-1-thread-3|
31|James     |Willis  |pool-1-thread-3|
36|Lyndon    |Willis  |pool-1-thread-3|
37|Calvin    |Willis  |pool-1-thread-3|
38|Franklin  |Willis  |pool-1-thread-3|
43|Warren    |Willis  |pool-1-thread-3|
47|Woodrow   |Willis  |pool-1-thread-3|
48|William   |Willis  |pool-1-thread-3|
49|Millard   |Willis  |pool-1-thread-3|
55|Rutherford|Willis  |pool-1-thread-3|
57|Thomas    |Willis  |pool-1-thread-3|
71|George    |Willis  |pool-1-thread-3|
85|Bill      |Willis  |pool-1-thread-3|
88|Jimmy     |Willis  |pool-1-thread-3|
96|Franklin  |Willis  |pool-1-thread-3|
--+----------+--------+---------------'
```

A possible result, is to have Thomson assigned to Thread 1, Harris assigned to Thread 2, Willis assigned to Thread 3, Smith assigned to Thread 4 and Higgins assigned to Thread 5.

4. Go back to the **Component** view of the **output row of tRowGenerator**. In the **Parallelization tab**, set the **Number of Child Threads** to `3`:



Type: (● Partition row) ○ None

Number of Child Threads: 3

QUEUE_SIZE: 1000

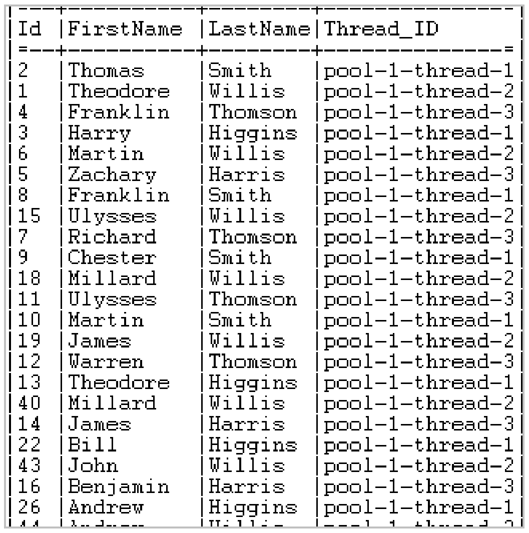
☑ Use a key hash for partitions

Key Columns:

Column
LastName

5. **Run** the Job and observe the result in the **Console**.
   This time, only 3 output tables are created for the first tLogRow component.
   A possible result is to have Higgins and Smith assigned to Thread 1, Willis assigned to Thread 2 and Thomson and Harris assigned to Thread 3.
   The output of the second tLogRow is a unique output table displaying all the

records dispatched on the 3 created Threads:

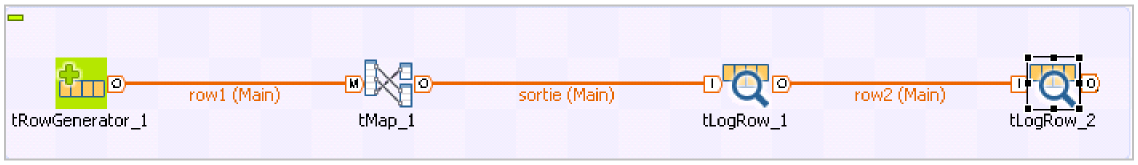


```
|---+------------+---------+----------------|
|Id |FirstName   |LastName |Thread_ID       |
|=--+------------+---------+---------------=|
|2  |Thomas      |Smith    |pool-1-thread-1|
|1  |Theodore    |Willis   |pool-1-thread-2|
|4  |Franklin    |Thomson  |pool-1-thread-3|
|3  |Harry       |Higgins  |pool-1-thread-1|
|6  |Martin      |Willis   |pool-1-thread-2|
|5  |Zachary     |Harris   |pool-1-thread-3|
|8  |Franklin    |Smith    |pool-1-thread-1|
|15 |Ulysses     |Willis   |pool-1-thread-2|
|7  |Richard     |Thomson  |pool-1-thread-3|
|9  |Chester     |Smith    |pool-1-thread-1|
|18 |Millard     |Willis   |pool-1-thread-2|
|11 |Ulysses     |Thomson  |pool-1-thread-3|
|10 |Martin      |Smith    |pool-1-thread-1|
|19 |James       |Willis   |pool-1-thread-2|
|12 |Warren      |Thomson  |pool-1-thread-3|
|13 |Theodore    |Higgins  |pool-1-thread-1|
|40 |Millard     |Willis   |pool-1-thread-2|
|14 |James       |Harris   |pool-1-thread-3|
|22 |Bill        |Higgins  |pool-1-thread-1|
|43 |John        |Willis   |pool-1-thread-2|
|16 |Benjamin    |Harris   |pool-1-thread-3|
|26 |Andrew      |Higgins  |pool-1-thread-1|
```

Disable Parallelization

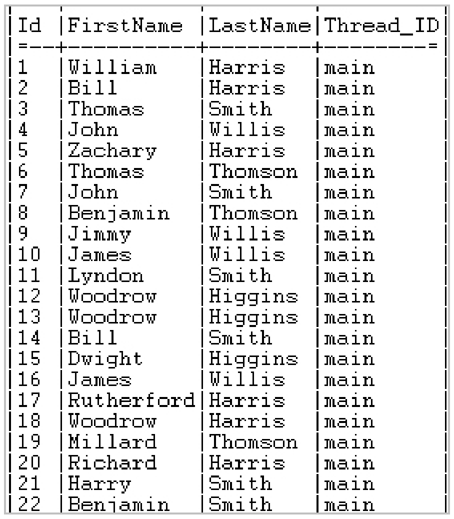
Disabling parallelization is as easy as enabling it.

1. Right-click **tRowGenerator** and then click **Disable Parallelization** in the contextual menu:

   This will delete the Parallelization icons.

2. **Run** your Job and check in the **Console** that the Parallelization has been disabled:

```
|Id |FirstName   |LastName |Thread_ID|
|=--+------------+---------+--------=|
|1  |William     |Harris   |main     |
|2  |Bill        |Harris   |main     |
|3  |Thomas      |Smith    |main     |
|4  |John        |Willis   |main     |
|5  |Zachary     |Harris   |main     |
|6  |Thomas      |Thomson  |main     |
|7  |John        |Smith    |main     |
|8  |Benjamin    |Thomson  |main     |
|9  |Jimmy       |Willis   |main     |
|10 |James       |Willis   |main     |
|11 |Lyndon      |Smith    |main     |
|12 |Woodrow     |Higgins  |main     |
|13 |Woodrow     |Higgins  |main     |
|14 |Bill        |Smith    |main     |
|15 |Dwight      |Higgins  |main     |
|16 |James       |Willis   |main     |
|17 |Rutherford  |Harris   |main     |
|18 |Woodrow     |Harris   |main     |
|19 |Millard     |Thomson  |main     |
|20 |Richard     |Harris   |main     |
|21 |Harry       |Smith    |main     |
|22 |Benjamin    |Smith    |main     |
```

You should see only one output table for the first tLogRow component. And you should find a unique Thread ID, named main.

You have covered the last Lab on parallelization. Now it's time to Wrap Up.

Wrap-Up

Recap

In this lesson, you set multi-thread execution on a Job as a whole to have the subjobs run in parallel, used the tParallelize component to run subjobs in parallel and then run an additional subjob after completion, and enabled parallel execution for an individual database output component.

You also used components dedicated to partitioning and collecting data to achieve multi-thread processing. You achieved the same task enabling automatically parallelization in your Job.

You can use these methods to take advantage of a multi-core or multi-processor system to speed execution of your Jobs.

And a best practice is to use the automatic implementation of parallelization across a Job without the use of dedicated components.

Further Reading

For more information about topics covered in this lesson, see the *Talend Components Reference Guide*.

Joblet

Overview

A Joblet is a specific component that replaces Job component Groups. It factorizes recurrent processing or complex transformation steps to ease the reading of a complex Job. Joblets can be reused in different Jobs or several times in the same Job.

Available Joblets appear in the Repository under the Joblet Designs section.

At runtime, the Joblet code is integrated in the Job code itself. This way, the Joblet does not impact the performance of your Job. The performances are exactly the same as if you included directly the subjob in your design.

Given that the Joblet code is automatically included in the Job code at run-time, unlike tRunJob components, (covered in the DI Basics course) it uses less resources. Also, the Joblet uses the same context variables as the Job itself.

Objectives

After completing this tutorial, you will be able to:

- Create a Joblet from scratch
- Create a Joblet from an existing Job
- Create a Joblet that allows triggered executions
- Use a Joblet in a Job

Before You Begin

Be sure you are working in an environment that contains the following:

- A properly installed copy of Talend Data Integration
- The supporting files for this lesson

The first step is to open the Job you will use for this Lab.

Joblet creation from an existing Job

Overview

This lab will show you how to create a Joblet from an existing Job.

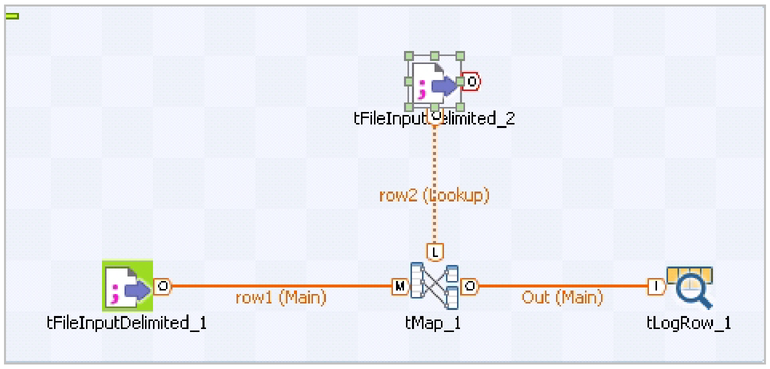
As an example, you will use a basic Job that reads data from a two different files and join them before displaying the result in the Console. In the first file, you will find information on a list of customers composed of their first and last name, their detailed address that is recorded in the Number, Street, City and State columns of the file.

The second file is the list of State codes with the corresponding State name.

You will run the Job to see how it works and next you will refactor a part of this Job as a Joblet.

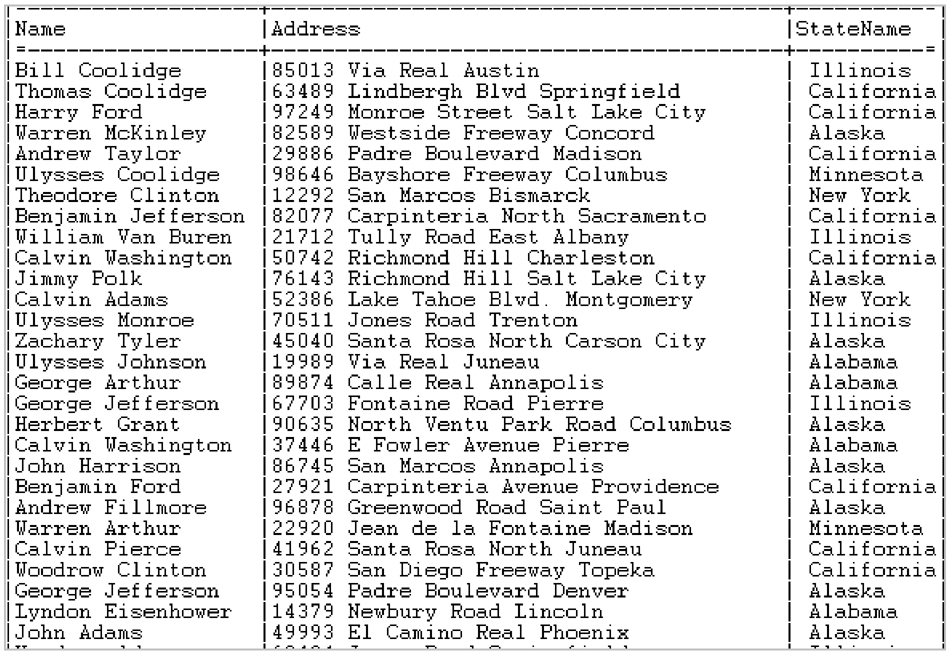
Run the original Job

First, you will import and run the original Job. This Job is saved as a zip file in the C:\StudentFiles folder.

1. In the **Repository**, right-click **Job Designs** and then select **Import Items** in the contextual menu.
2. Click **Select Archive file** in the **Import items** window.
3. Navigate to **C:\StudentFiles** and then select **JoinData_original.zip**
4. Select everything in the archive file and then click **Finish** to import the Job and its dependencies.
5. The Job JoinData_original appears in the **Repository**.
   Open the Job. It looks like the following:
   

6. Duplicate the Job and name it JoinData.

7. Open JoinData and then run the Job and observe the result in the Console:



```
|-------------------+----------------------------------------+-----------|
|Name               |Address                                 |StateName  |
|===================+========================================+===========|
|Bill Coolidge      |85013 Via Real Austin                   | Illinois  |
|Thomas Coolidge    |63489 Lindbergh Blvd Springfield        | California|
|Harry Ford         |97249 Monroe Street Salt Lake City      | California|
|Warren McKinley    |82589 Westside Freeway Concord          | Alaska    |
|Andrew Taylor      |29886 Padre Boulevard Madison           | California|
|Ulysses Coolidge   |98646 Bayshore Freeway Columbus         | Minnesota |
|Theodore Clinton   |12292 San Marcos Bismarck               | New York  |
|Benjamin Jefferson |82077 Carpinteria North Sacramento      | California|
|William Van Buren  |21712 Tully Road East Albany            | Illinois  |
|Calvin Washington  |50742 Richmond Hill Charleston          | California|
|Jimmy Polk         |76143 Richmond Hill Salt Lake City      | Alaska    |
|Calvin Adams       |52386 Lake Tahoe Blvd. Montgomery       | New York  |
|Ulysses Monroe     |70511 Jones Road Trenton                | Illinois  |
|Zachary Tyler      |45040 Santa Rosa North Carson City      | Alaska    |
|Ulysses Johnson    |19989 Via Real Juneau                   | Alabama   |
|George Arthur      |89874 Calle Real Annapolis              | Alabama   |
|George Jefferson   |67703 Fontaine Road Pierre              | Illinois  |
|Herbert Grant      |90635 North Ventu Park Road Columbus    | Alaska    |
|Calvin Washington  |37446 E Fowler Avenue Pierre            | Alabama   |
|John Harrison      |86745 San Marcos Annapolis              | Alaska    |
|Benjamin Ford      |27921 Carpinteria Avenue Providence     | California|
|Andrew Fillmore    |96878 Greenwood Road Saint Paul         | Alaska    |
|Warren Arthur      |22920 Jean de la Fontaine Madison       | Minnesota |
|Calvin Pierce      |41962 Santa Rosa North Juneau           | California|
|Woodrow Clinton    |30587 San Diego Freeway Topeka          | California|
|George Jefferson   |95054 Padre Boulevard Denver            | Alaska    |
|Lyndon Eisenhower  |14379 Newbury Road Lincoln              | Alabama   |
|John Adams         |49993 El Camino Real Phoenix            | Alaska    |
```

The next step is to create a Joblet that corresponds to the mapping task in the Job.

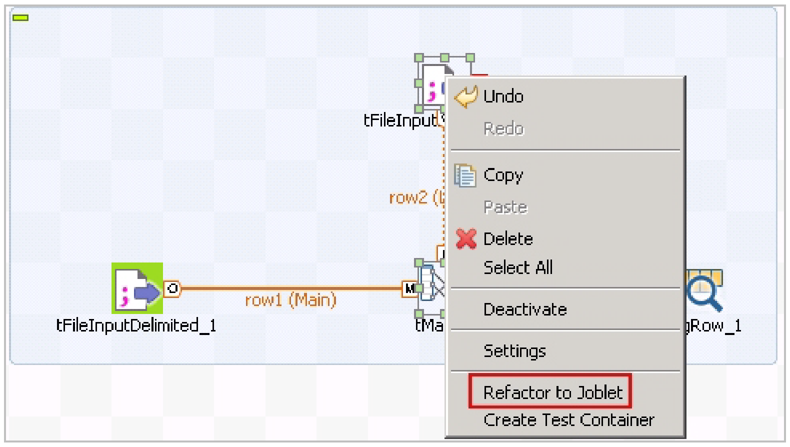
Refactor mapping to Joblet

Creating a Joblet from an existing Job is easy to do. The procedure is to select the portion of Job you want to refactor and then ask for the refactoring.

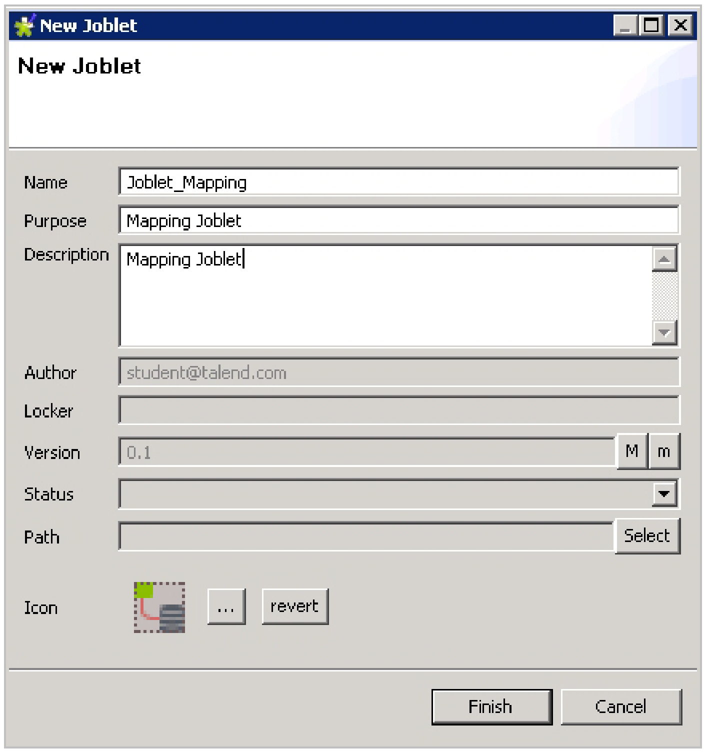
The Joblet you are about to create will include two components : tFileInputDelimited_2 and tMap.

1. In the duplicated JoinData Job, select **tMap** and **tFileInputDelimited_2** in the Designer view.

2. Right-click the selected components and then select **Refactor to Joblet** in the contextual menu:



3. A window appears where you will be able to give a name to your Joblet.
   In the **Name** box, enter **Joblet_Mapping**, provide a **Purpose** and a **Description**:

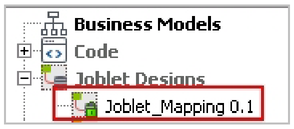


New Joblet

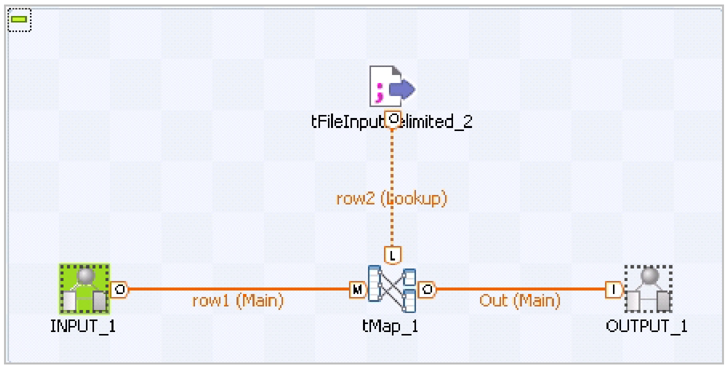
New Joblet

Name	Joblet_Mapping
Purpose	Mapping Joblet
Description	Mapping Joblet
Author	student@talend.com
Locker	
Version	0.1
Status	
Path	

4. Click **Finish** to save the Joblet.
   The Joblet will appear under Joblet Designs in the Repository:



5. The Joblet is also opened in the Designer View:

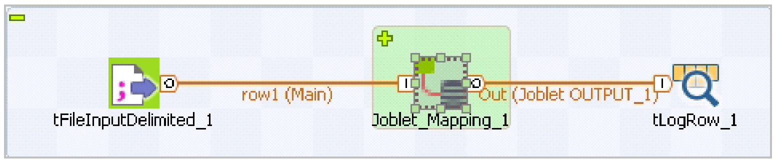


   Observe the INPUT_1 and OUTPUT_1 components in the design of Joblet_Mapping. These components are necessary to have input and output connections to your Joblet.

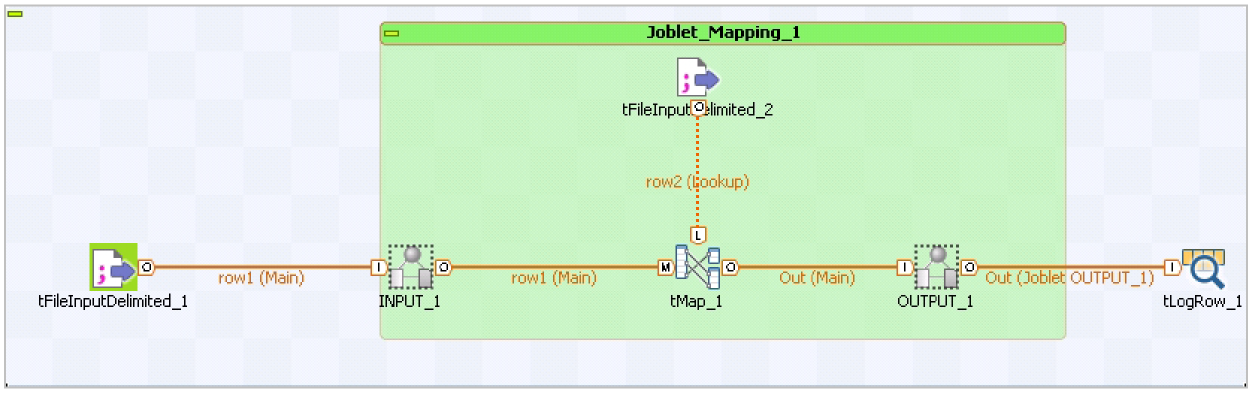
   Save Joblet_Mapping.

6. Open **JoinData** in the **Designer** view.

   The tMap and tFileInputDelimited components have disappeared and have been replaced by the Joblet_Mapping component. This is the Joblet you just created:

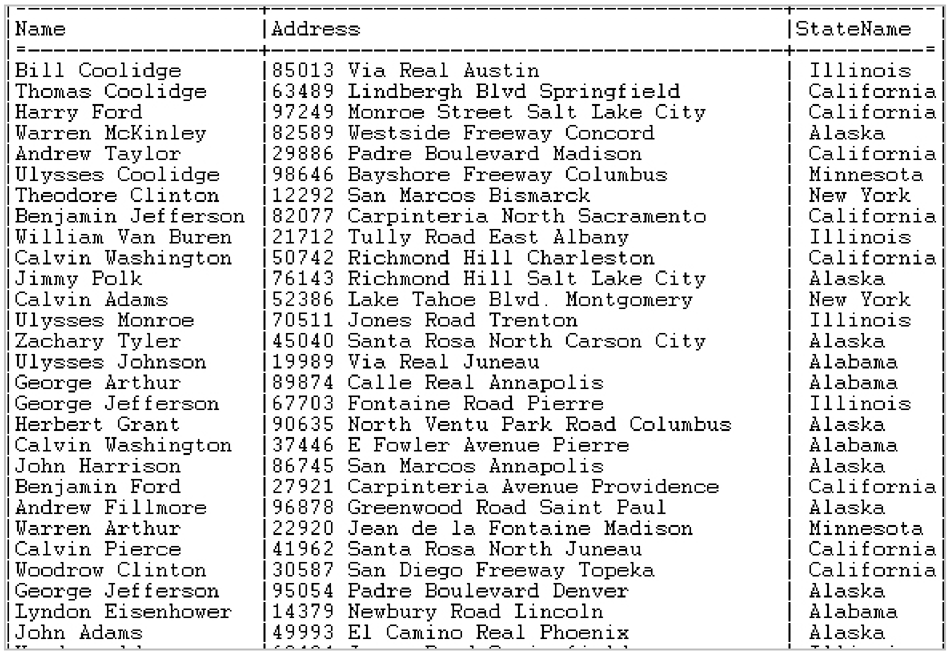


7. Click the **green plus sign** in the top left corner of the **Joblet_Mapping** component:



   This will expand the Joblet_Mapping design inside the JoinData Job design.

8. Run the Job and observe the Result in the Console:



```
|-------------------+----------------------------------------+-----------|
|Name               |Address                                 |StateName  |
|===================+========================================+==========|
|Bill Coolidge      |85013 Via Real Austin                   | Illinois  |
|Thomas Coolidge    |63489 Lindbergh Blvd Springfield        | California|
|Harry Ford         |97249 Monroe Street Salt Lake City      | California|
|Warren McKinley    |82589 Westside Freeway Concord          | Alaska    |
|Andrew Taylor      |29886 Padre Boulevard Madison           | California|
|Ulysses Coolidge   |98646 Bayshore Freeway Columbus         | Minnesota |
|Theodore Clinton   |12292 San Marcos Bismarck               | New York  |
|Benjamin Jefferson |82077 Carpinteria North Sacramento      | California|
|William Van Buren  |21712 Tully Road East Albany            | Illinois  |
|Calvin Washington  |50742 Richmond Hill Charleston          | California|
|Jimmy Polk         |76143 Richmond Hill Salt Lake City      | Alaska    |
|Calvin Adams       |52386 Lake Tahoe Blvd. Montgomery       | New York  |
|Ulysses Monroe     |70511 Jones Road Trenton                | Illinois  |
|Zachary Tyler      |45040 Santa Rosa North Carson City      | Alaska    |
|Ulysses Johnson    |19989 Via Real Juneau                   | Alabama   |
|George Arthur      |89874 Calle Real Annapolis              | Alabama   |
|George Jefferson   |67703 Fontaine Road Pierre              | Illinois  |
|Herbert Grant      |90635 North Ventu Park Road Columbus    | Alaska    |
|Calvin Washington  |37446 E Fowler Avenue Pierre            | Alabama   |
|John Harrison      |86745 San Marcos Annapolis              | Alaska    |
|Benjamin Ford      |27921 Carpinteria Avenue Providence     | California|
|Andrew Fillmore    |96878 Greenwood Road Saint Paul         | Alaska    |
|Warren Arthur      |22920 Jean de la Fontaine Madison       | Minnesota |
|Calvin Pierce      |41962 Santa Rosa North Juneau           | California|
|Woodrow Clinton    |30587 San Diego Freeway Topeka          | California|
|George Jefferson   |95054 Padre Boulevard Denver            | Alaska    |
|Lyndon Eisenhower  |14379 Newbury Road Lincoln              | Alabama   |
|John Adams         |49993 El Camino Real Phoenix            | Alaska    |
```

You should have exactly the same result as before.

You just created a Joblet from an existing Job. In the next section you will create a Joblet from scratch and use it in the JoinData Job.

Joblet creation from scratch

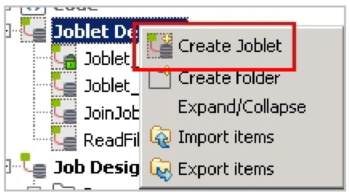
Overview

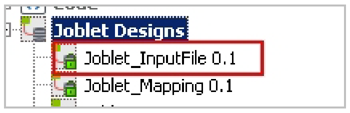
It is also possible to create a Joblet from a blank Designer. The procedure is very similar to a standard Job creation.

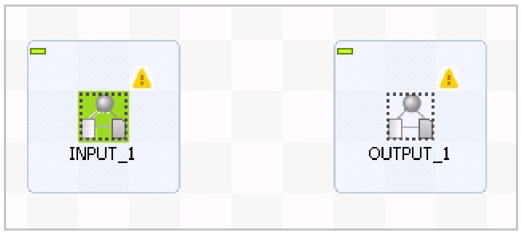
You will create a startable Joblet. That means that this Joblet can be used to start a Job, without any input link on this Joblet. This Joblet will read the Customers file. And then, you will use this Joblet in the JoinData Job.

Creating a Joblet

In the Repository, there is a Joblet Designs folder. This is where you will create your Joblet.

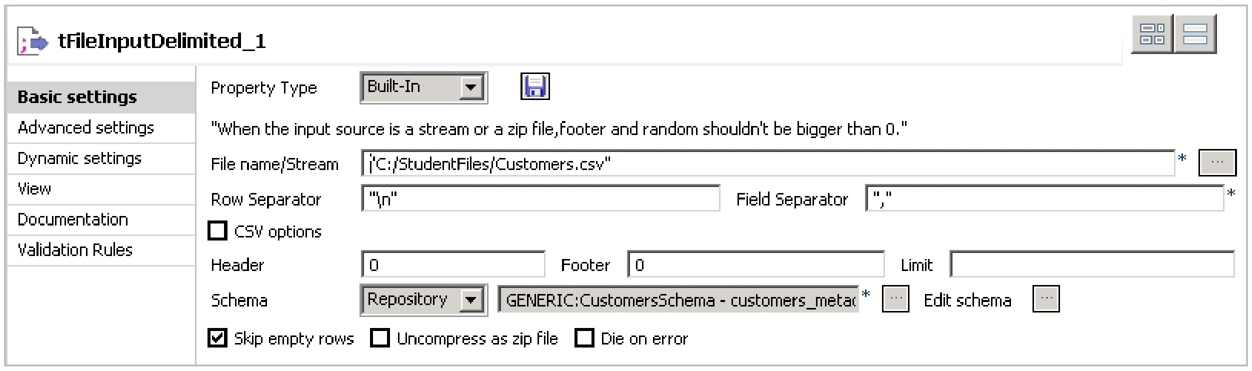
1. In the **Repository**, right-click **Joblet Designs** and then click **Create Joblet**:


2. Name your Joblet **Joblet_InputFile** and then click **Finish**:


The Joblet opens in the Designer:


As a default, you will get an Input and an Output component, so that your Joblet could be connected to other components.
3. Copy **tFileInputDelimited** component, in the **JoinData** Job and paste it between **INPUT_1** and **OUTPUT_1**.
It should be configured to read the Customers.csv file in the C:\StudentFiles folder.

4. Open the **Component** view. Your configuration should be as follows:



tFileInputDelimited_1

Basic settings | Advanced settings | Dynamic settings | View | Documentation | Validation Rules

Property Type: Built-In

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

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

Row Separator: "\n" Field Separator: ","

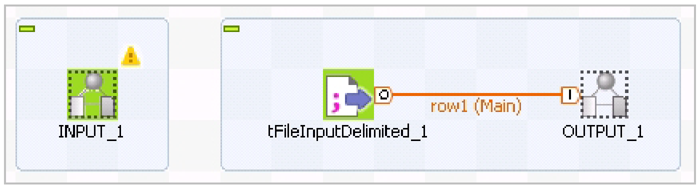
☐ CSV options

Header: 0 Footer: 0 Limit:

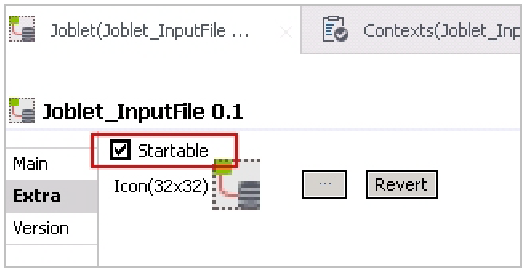
Schema: Repository GENERIC:CustomersSchema - customers_metac Edit schema

☑ Skip empty rows ☐ Uncompress as zip file ☐ Die on error

5. Connect the **tFileInputDelimited** component to the **OUTPUT** component with the **Main** row:



6. Click the **Joblet view** (next to the Run and Context views) and then click the **Extra** tab:



Joblet_InputFile 0.1

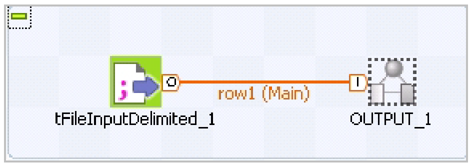
☑ Startable

Icon(32x32)

This is where you can choose to have a Joblet Startable or not.

As the Joblet will be used to read the Customers file, it may not be useful to keep the INPUT_1 component.

7. Delete **INPUT_1** and then save the Joblet:

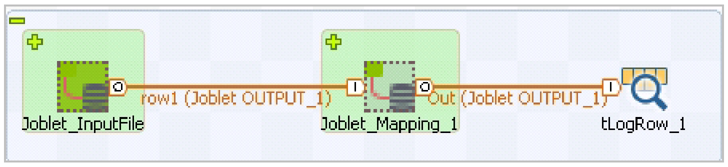


Next, you will use this Joblet in the JoinData Job.

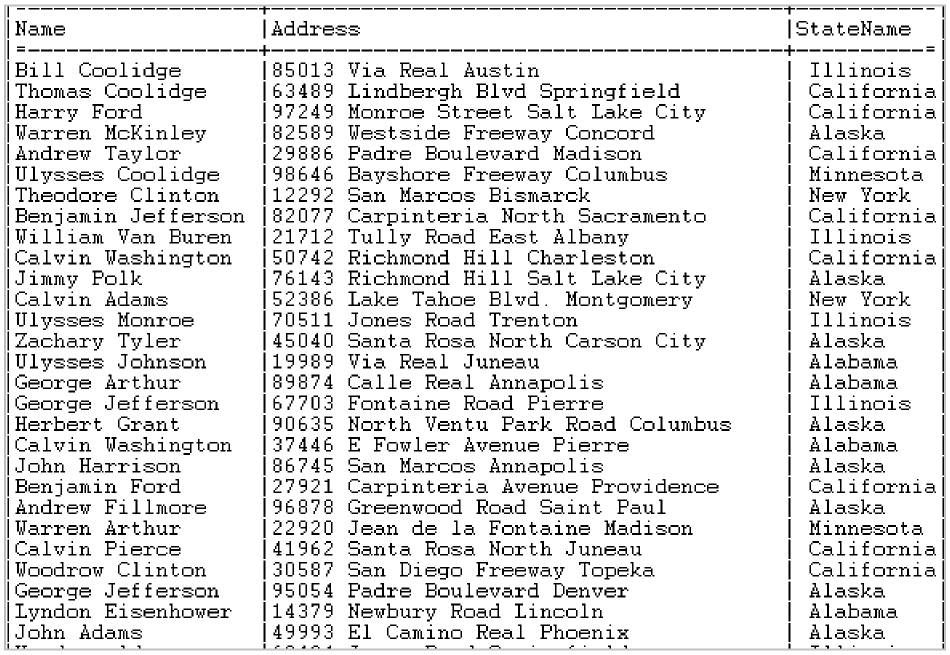
Update the JoinData Job

You will update the Job with this new Joblet and then run it to check if it's still working correctly.

1. Open the **JoinData** Job in the **Designer**.
2. Delete the **tFileInputDelimited** component.
3. Click **Joblet_InputFile** under **Joblet Designs** in the **Repository** and then drag it to the **Designer**.
4. Connect **Joblet_InputFile** to **Joblet_Mapping**:



5. Run the Job and check the result in the **Console**:



```
|=-------------------+-----------------------------------------+----------=|
|Name                |Address                                  |StateName  |
|=-------------------+-----------------------------------------+----------=|
|Bill Coolidge       |85013 Via Real Austin                    | Illinois  |
|Thomas Coolidge     |63489 Lindbergh Blvd Springfield         | California|
|Harry Ford          |97249 Monroe Street Salt Lake City       | California|
|Warren McKinley     |82589 Westside Freeway Concord           | Alaska    |
|Andrew Taylor       |29886 Padre Boulevard Madison            | California|
|Ulysses Coolidge    |98646 Bayshore Freeway Columbus          | Minnesota |
|Theodore Clinton    |12292 San Marcos Bismarck                | New York  |
|Benjamin Jefferson  |82077 Carpinteria North Sacramento       | California|
|William Van Buren   |21712 Tully Road East Albany             | Illinois  |
|Calvin Washington   |50742 Richmond Hill Charleston           | California|
|Jimmy Polk          |76143 Richmond Hill Salt Lake City       | Alaska    |
|Calvin Adams        |52386 Lake Tahoe Blvd. Montgomery        | New York  |
|Ulysses Monroe      |70511 Jones Road Trenton                 | Illinois  |
|Zachary Tyler       |45040 Santa Rosa North Carson City       | Alaska    |
|Ulysses Johnson     |19989 Via Real Juneau                    | Alabama   |
|George Arthur       |89874 Calle Real Annapolis               | Alabama   |
|George Jefferson    |67703 Fontaine Road Pierre               | Illinois  |
|Herbert Grant       |90635 North Ventu Park Road Columbus     | Alaska    |
|Calvin Washington   |37446 E Fowler Avenue Pierre             | Alabama   |
|John Harrison       |86745 San Marcos Annapolis               | Alaska    |
|Benjamin Ford       |27921 Carpinteria Avenue Providence      | California|
|Andrew Fillmore     |96878 Greenwood Road Saint Paul          | Alaska    |
|Warren Arthur       |22920 Jean de la Fontaine Madison        | Minnesota |
|Calvin Pierce       |41962 Santa Rosa North Juneau            | California|
|Woodrow Clinton     |30587 San Diego Freeway Topeka           | California|
|George Jefferson    |95054 Padre Boulevard Denver             | Alaska    |
|Lyndon Eisenhower   |14379 Newbury Road Lincoln               | Alabama   |
|John Adams          |49993 El Camino Real Phoenix             | Alaska    |
```

You should get the same results as before.

You have learned how to build a Joblet and how to make it startable or not. The next step is to create a Joblet than can be triggered, with an OnSubjobOK trigger, for example.

Triggered Joblet

Overview

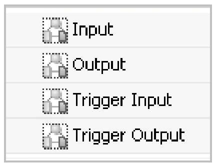
It is possible to use a Joblet as a step in a Job. You can start the execution of a Joblet after the execution of a subjob or start the execution of a subjob after the execution of a Joblet.

You create a new Joblet which will allow using Triggers to connect to it.

Create a Triggered Joblet

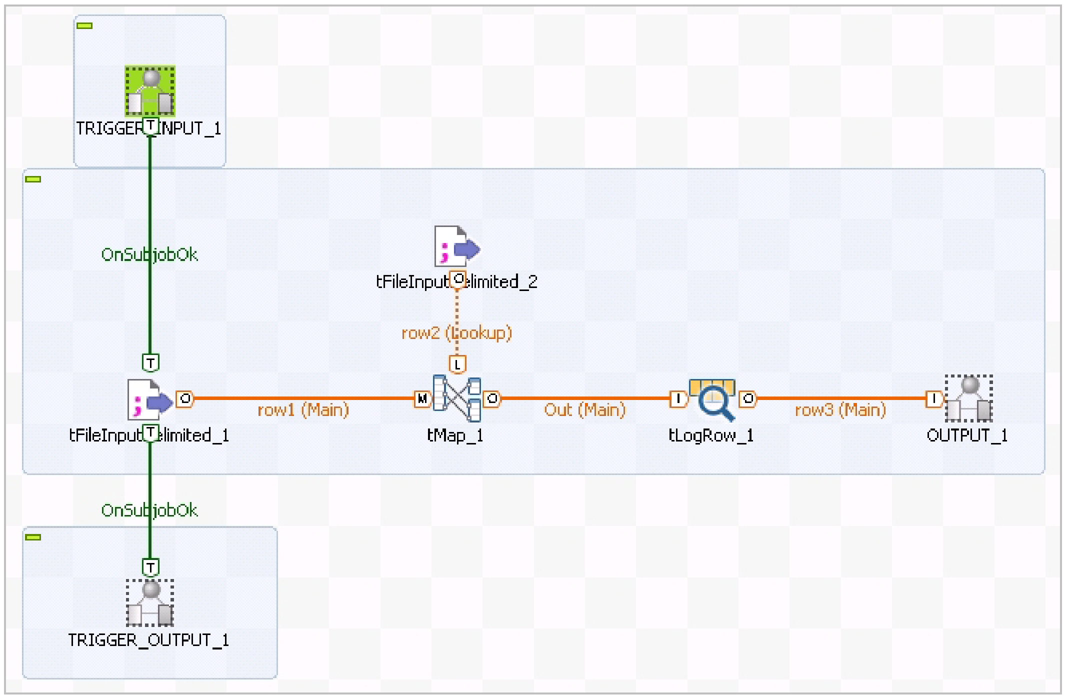
You will use the JoinData_original Job to create a Joblet and then add triggers component to this Joblet.

1. Open the **JoinData_original** Job.
2. Create a new Joblet and then name it **Joblet_trigger**.
3. Copy the **JoinData_original** Job components and paste them in **Joblet_trigger**.
4. When you edit a Joblet, components specific to Joblets appear in top of the Palette:

   Add a **Trigger Input** component above the **tFileInputDelimited** component.
5. Connect **TRIGGER_INPUT_1** to **tFileInputDelimited** with an **OnSubjobOk** trigger.
6. Add a **Trigger Output** component below the **tFileInputDelimited** component.
7. Connect **tFileInputDelimited** to **TRIGGER_OUTPUT_1** with an **OnSubjobOk** trigger.

8. Add an **OUTPUT** component and connect it to **tLogRow** with a **Main** row. Your Joblet should look like the following:



9. Save your Joblet

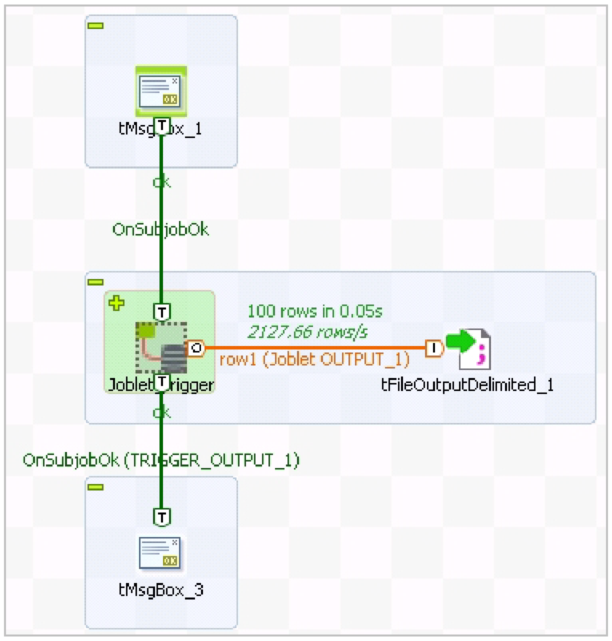
Now, you will build a new Job to test this Joblet.

Create test Job

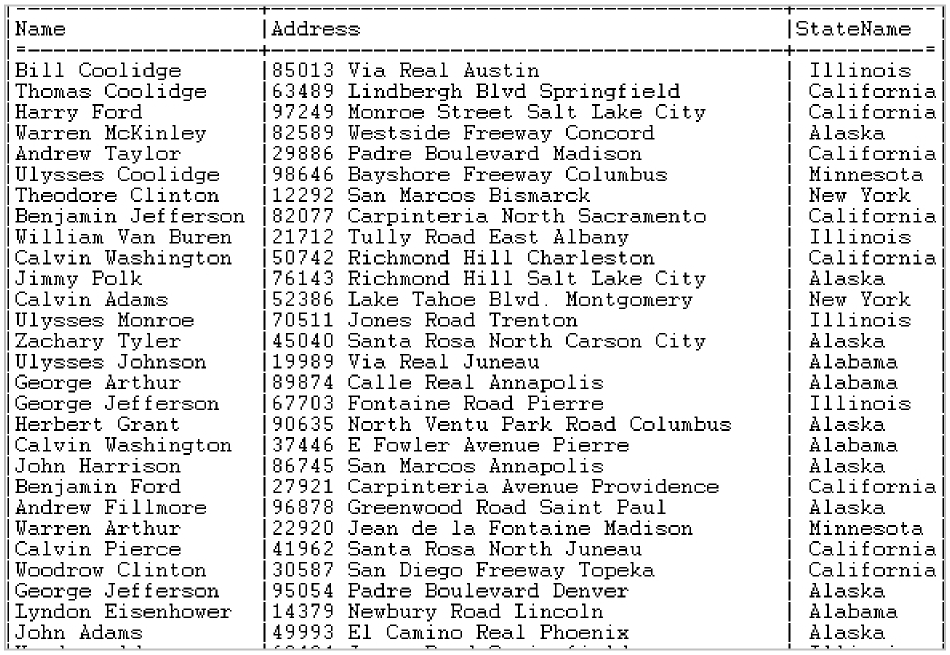
You will build a Job that will generate an information message, launch the Joblet and then generate another information message.

1. Create a new Job and name it **JoinData_trigger**.
2. Add a **tMsgBox** component and double-click to open the **Component** view.
3. In the **Message** box, enter "Start".
4. Drag **Joblet_trigger** from the **Repository**. Place it below **tMsgBox**.
5. Connect **tMsgBox** to **Joblet_trigger** with an **OnSubjobOk** trigger.
6. Add a **tFileOutputDelimited** component at the right side of **Joblet_trigger.**
7. Connect **Joblet_trigger** to **tFileOutputDelimited** with the **Joblet OUTPUT_1 row**.
8. Open the **Component** view and then configure **tFileOutputDelimited** to write the file under "C:\Studentfiles\Customers_out.csv".
9. Add a **tMsgBox** below **Joblet_trigger** and connect it with an **OnSubjobOk** trigger.

10. Double-click **tMsgBox** to open the **Component** view and then, enter "Processing Done!" in the **Message box**.
    Your Job should look like the following:



11. Run your Job and check the results in the Console.
    A first popup message appears and once you click OK, the Joblet executes. At the end of the Joblet execution, the second popup message appears. Click **OK** to end the Job.
    You should have the same results as previously in the Console:



```
|--------------------+----------------------------------------+----------|
|Name                |Address                                 |StateName |
|====================+========================================+==========|
|Bill Coolidge       |85013 Via Real Austin                   | Illinois |
|Thomas Coolidge     |63489 Lindbergh Blvd Springfield        | California|
|Harry Ford          |97249 Monroe Street Salt Lake City      | California|
|Warren McKinley     |82589 Westside Freeway Concord          | Alaska   |
|Andrew Taylor       |29886 Padre Boulevard Madison           | California|
|Ulysses Coolidge    |98646 Bayshore Freeway Columbus         | Minnesota |
|Theodore Clinton    |12292 San Marcos Bismarck               | New York |
|Benjamin Jefferson  |82077 Carpinteria North Sacramento      | California|
|William Van Buren   |21712 Tully Road East Albany            | Illinois |
|Calvin Washington   |50742 Richmond Hill Charleston          | California|
|Jimmy Polk          |76143 Richmond Hill Salt Lake City      | Alaska   |
|Calvin Adams        |52386 Lake Tahoe Blvd. Montgomery       | New York |
|Ulysses Monroe      |70511 Jones Road Trenton                | Illinois |
|Zachary Tyler       |45040 Santa Rosa North Carson City      | Alaska   |
|Ulysses Johnson     |19989 Via Real Juneau                   | Alabama  |
|George Arthur       |89874 Calle Real Annapolis              | Alabama  |
|George Jefferson    |67703 Fontaine Road Pierre              | Illinois |
|Herbert Grant       |90635 North Ventu Park Road Columbus    | Alaska   |
|Calvin Washington   |37446 E Fowler Avenue Pierre            | Alabama  |
|John Harrison       |86745 San Marcos Annapolis              | Alaska   |
|Benjamin Ford       |27921 Carpinteria Avenue Providence     | California|
|Andrew Fillmore     |96878 Greenwood Road Saint Paul         | Alaska   |
|Warren Arthur       |22920 Jean de la Fontaine Madison       | Minnesota |
|Calvin Pierce       |41962 Santa Rosa North Juneau           | California|
|Woodrow Clinton     |30587 San Diego Freeway Topeka          | California|
|George Jefferson    |95054 Padre Boulevard Denver            | Alaska   |
|Lyndon Eisenhower   |14379 Newbury Road Lincoln              | Alabama  |
|John Adams          |49993 El Camino Real Phoenix            | Alaska   |
```

You have discovered how to create and use Joblets. Now, it's time to Wrap Up.

Wrap Up

Recap

In this chapter, you discovered Joblets which allow you to factorize a portion of Job. Joblets can be used in Jobs as any other component without impacting the performances of your Job.

Further Reading

To learn more about Joblets, see the *Talend Data Integration Studio User Guide.*

Survey

Please take a few minutes to provide feedback on your training experience by completing this very short survey.

Copyright