

Talend Open Studio

for ESB

User Guide

5.1_b

Talend Open Studio : User Guide

Adapted for Talend Open Studio for ESB v5.1.x. Supersedes previous User Guide releases.

Copyleft

This documentation is provided under the terms of the Creative Commons Public License (CCPL).

For more information about what you can and cannot do with this documentation in accordance with the CCPL, please read: <http://creativecommons.org/licenses/by-nc-sa/2.0/>

Notices

Talend, Talend Integration Factory, Talend Service Factory, and Talend ESB are trademarks of Talend, Inc.

Apache CXF, CXF, Apache Karaf, Karaf, Apache Camel, Camel, Apache Maven, Maven, Apache Archiva and Archiva are trademarks of The Apache Foundation.

SoapUI is a trademark of SmartBear Software.

All other brands, product names, company names, trademarks and service marks are the properties of their respective owners.

Table of Contents

Preface	vii
1. General information	vii
1.1. Purpose	vii
1.2. Audience	vii
1.3. Typographical conventions	vii
2. History of changes	vii
3. Feedback and Support	ix
Chapter 1. Enterprise Service Bus and Talend Studio	1
1.1. Versatile and flexible Web services.....	2
1.2. Dynamic mediation and intelligent routing	2
1.3. Flexible deployment	2
1.4. Live statistics and real-time debugging	2
1.5. System monitoring	3
1.6. Service security	3
1.7. Failover and load balancing	3
1.8. <i>Talend Open Studio for ESB</i> functional architecture	3
Chapter 2. Getting started with Talend Studio	7
2.1. Important concepts in <i>Talend Open Studio for ESB</i>	8
2.2. Launching <i>Talend Open Studio for ESB</i>	9
2.2.1. How to launch the Studio for the first time	9
2.2.2. How to set up a project	13
2.3. Working with different workspace directories	13
2.3.1. How to create a new workspace directory	14
2.4. Working with projects	14
2.4.1. How to create a project	15
2.4.2. How to import the demo project	17
2.4.3. How to import projects	18
2.4.4. How to open a project	20
2.4.5. How to delete a project	21
2.4.6. How to export a project	21
2.4.7. Migration tasks	22
2.5. Setting <i>Talend Open Studio for ESB</i> preferences	23
2.5.1. Java Interpreter path	23
2.5.2. External or User components.....	24
2.5.3. Exchange preferences	25
2.5.4. Language preferences	25
2.5.5. Debug and Job execution preferences	26
2.5.6. Designer preferences	27
2.5.7. Adding code by default	28
2.5.8. Performance preferences	29
2.5.9. Documentation preferences	30
2.5.10. Displaying special characters for schema columns	30
2.5.11. SQL Builder preferences	30
2.5.12. Schema preferences	31
2.5.13. Libraries preferences	32
2.5.14. Type conversion	33
2.5.15. Usage Data Collector preferences	33
2.6. Customizing project settings	35
2.6.1. Palette Settings	36
2.6.2. Version management	37
2.6.3. Status management	38
2.6.4. Job Settings	39
2.6.5. Stats & Logs	40

2.6.6. Context settings	41
2.6.7. Project Settings use	42
2.6.8. Status settings	43
2.6.9. Security settings	45
2.7. Filtering entries listed in the Repository tree view	45
2.7.1. How to filter by Job name	45
2.7.2. How to filter by user	47
2.7.3. How to filter by job status	49
2.7.4. How to choose what repository nodes to display	49
Chapter 3. Designing a Business Model	53
3.1. What is a Business Model	54
3.2. Opening or creating a Business Model	54
3.2.1. How to open a Business Model	55
3.2.2. How to create a Business Model	55
3.3. Modeling a Business Model	56
3.3.1. Shapes	56
3.3.2. Connecting shapes	57
3.3.3. How to comment and arrange a model	59
3.3.4. Business Models	61
3.4. Assigning repository elements to a Business Model	63
3.5. Editing a Business Model	64
3.5.1. How to rename a Business Model	64
3.5.2. How to copy and paste a Business Model	64
3.5.3. How to move a Business Model	64
3.5.4. How to delete a Business Model	64
3.6. Saving a Business Model	64
Chapter 4. Designing a Job	67
4.1. What is a Job design	68
4.2. Getting started with a basic Job design	68
4.2.1. How to create a Job	68
4.2.2. How to drop components to the workspace	71
4.2.3. How to search components in the Palette	73
4.2.4. How to connect components together	73
4.2.5. How to drop components in the middle of a Row link	74
4.2.6. How to define component properties	75
4.2.7. How to run a Job	81
4.2.8. How to customize your workspace	87
4.3. Using connections	92
4.3.1. Connection types	92
4.3.2. How to define connection settings	96
4.4. Using the Metadata Manager	98
4.4.1. How to centralize the Metadata items	98
4.4.2. How to centralize contexts and variables	99
4.4.3. How to use the SQL Templates	110
4.5. Handling Jobs: advanced subjects	110
4.5.1. How to map data flows	110
4.5.2. How to create queries using the SQLBuilder	111

4.5.3. How to download/upload Talend Community components	114
4.5.4. How to install external modules	121
4.5.5. How to launch a Job periodically	122
4.5.6. How to use the tPrejob and tPostjob components	124
4.6. Handling Jobs: miscellaneous subjects	125
4.6.1. How to share a database connection	125
4.6.2. How to define the Start component	126
4.6.3. How to handle error icons on components or Jobs	126
4.6.4. How to add notes to a Job design	128
4.6.5. How to display the code or the outline of your Job	129
4.6.6. How to manage the subjob display	130
4.6.7. How to define options on the Job view	132
4.6.8. How to find components in Jobs	133
4.6.9. How to set default values in the schema of a component	135
Chapter 5. Designing a Route	137
5.1. What is a Route	138
5.2. Accessing the Mediation perspective....	138
5.2.1. Switching to the Mediation perspective	138
5.2.2. Managing quick access icons for different perspectives	139
5.3. Getting started with a basic Route	140
5.3.1. How to create a Route	141
5.3.2. How to drop components to the workspace	143
5.3.3. How to define component properties	143
5.3.4. How to use Camel components in a Route	146
5.3.5. How to add notes to a Route....	148
5.3.6. How to run a Route	148
5.4. Using connections	151
5.4.1. Connection types	151
5.4.2. How to define connection settings	153
5.5. Using Beans	154
5.5.1. How to create a Bean	154
5.5.2. How to use a Bean	155
Chapter 6. Designing a Service	157
6.1. What is a Service	158
6.2. Getting started with a basic Service....	158
6.2.1. How to create a Service	158
6.2.2. How to edit a WSDL file	163
6.2.3. How to associate data service Jobs with a Service	186
6.2.4. How to export a Service for deployment	190
6.3. Handling Services: miscellaneous subjects	191
6.3.1. How to import WSDL schemas	191
6.3.2. How to set the Runtime options	192
Chapter 7. Managing Jobs, Routes and Services	195
7.1. Activating/Deactivating a Job or a sub-job or a Route	196
7.1.1. How to disable a Start component	196
7.1.2. How to disable a non-Start component	196
7.2. Importing/exporting items, Routes, Services or Jobs	197
7.2.1. How to import items	197
7.2.2. How to export Jobs	199
7.2.3. How to export Routes	207
7.2.4. How to export items	208
7.2.5. How to change context parameters in Jobs and Routes	210
7.3. Managing repository items	211
7.3.1. How to handle updates in repository items	211
7.4. Searching a Job in the repository	213
7.5. Managing Job and Route versions	215
7.6. Documenting a Job	216
7.6.1. How to generate HTML documentation	216
7.6.2. How to update the documentation on the spot	217
7.7. Handling Job execution	217
7.7.1. How to deploy a Job on SpagoBI server	217
Chapter 8. Mapping data flows	221
8.1. tMap and tXMLMap interfaces	222
8.2. tMap operation	223
8.2.1. Setting the input flow in the Map Editor	224
8.2.2. Mapping variables	231
8.2.3. Using the expression editor	232
8.2.4. Mapping the Output setting	237
8.2.5. Setting schemas in the Map Editor	242
8.2.6. Solving memory limitation issues in tMap use	245
8.2.7. Handling Lookups	247
8.3. tXMLMap operation	248
8.3.1. Using the document type to create the XML tree	249
8.3.2. Defining the output mode	260
8.3.3. Editing the XML tree schema....	265
Chapter 9. Managing Metadata	267
9.1. Objectives	268
9.2. Setting up a DB connection	269
9.2.1. Step 1: General properties	269
9.2.2. Step 2: Connection	270
9.2.3. Step 3: Table upload	271
9.2.4. Step 4: Schema definition	274
9.3. Setting up a JDBC schema	275
9.3.1. Step 1: General properties	275
9.3.2. Step 2: Connection	275
9.3.3. Step 3: Table upload	277
9.3.4. Step 4: Schema definition	277
9.4. Setting up a SAS connection	278
9.4.1. Prerequisites	278
9.4.2. Step 1: General properties	278
9.4.3. Step 2: Connection	278
9.5. Setting up a File Delimited schema	280
9.5.1. Step 1: General properties	280
9.5.2. Step 2: File upload	281
9.5.3. Step 3: Schema definition	281
9.5.4. Step 4: Final schema	283
9.6. Setting up a File Positional schema	284
9.6.1. Step 1: General properties	285
9.6.2. Step 2: Connection and file upload	285
9.6.3. Step 3: Schema refining	286
9.6.4. Step 4: Finalizing the end schema	286
9.7. Setting up a File Regex schema	287

9.7.1. Step 1: General properties	287	11.3.2. How to access a system SQL template	358
9.7.2. Step 2: File upload	287	11.3.3. How to create user-defined SQL templates	360
9.7.3. Step 3: Schema definition	288	11.3.4. A use case of system SQL templates	361
9.7.4. Step 4: Finalizing the end schema	288	Appendix A. GUI	367
9.8. Setting up an XML file schema	288	A.1. Main window	368
9.8.1. Setting up an XML schema for an input file	289	A.2. Menu bar and Toolbar	369
9.8.2. Setting up an XML schema for an output file	296	A.2.1. Menu bar of <i>Talend Open Studio for ESB</i>	369
9.9. Setting up a File Excel schema	305	A.2.2. Toolbar of <i>Talend Open Studio for ESB</i>	370
9.9.1. Step 1: General properties	306	A.3. Repository tree view	371
9.9.2. Step 2: File upload	306	A.4. Design workspace	373
9.9.3. Step 3: Schema refining	307	A.5. Palette	373
9.9.4. Step 4: Finalizing the end schema	308	A.6. Configuration tabs	374
9.10. Setting up a File LDIF schema	309	A.7. Outline and code summary panel	376
9.10.1. Step 1: General properties	309	A.8. Shortcuts and aliases	376
9.10.2. Step 2: File upload	309	Appendix B. Theory into practice: Data service and routing examples	379
9.10.3. Step 3: Schema definition	310	B.1. Data service example	380
9.10.4. Step 4: Finalizing the end schema	311	B.1.1. How to build a data service	380
9.11. Setting up an LDAP schema	311	B.1.2. How to deploy a data service....	393
9.11.1. Step 1: General properties	312	B.1.3. How to build a Route using a data service	395
9.11.2. Step 2: Server connection	312	B.2. Job and Route example	401
9.11.3. Step 3: Authentication and DN fetching	313	B.2.1. Discovering the scenario	401
9.11.4. Step 4: Schema definition	314	B.2.2. Building a DI Job	401
9.11.5. Step 5: Finalizing the end schema	315	B.2.3. Building a Route	402
9.12. Setting up a Salesforce connection.....	315	B.2.4. Viewing the code and executing the Route and the Job	406
9.12.1. Step 1: General properties	316	Appendix C. SQL template writing rules	409
9.12.2. Step 2: Connection to a Salesforce account	316	C.1. SQL statements	410
9.12.3. Step 3: Retrieving Salesforce modules	317	C.2. Comment lines	410
9.12.4. Step 4: Retrieving Salesforce schemas	318	C.3. The <% . . . %> syntax	410
9.12.5. Step 5: Finalizing the end schema	319	C.4. The <%= . . . %> syntax	411
9.13. Setting up a Generic schema	321	C.5. The </ . . . /> syntax	411
9.13.1. Step 1: General properties	321	C.6. Code to access the component schema elements	412
9.13.2. Step 2: Schema definition	321	C.7. Code to access the component matrix properties	412
9.14. Setting up an MDM connection	322	Appendix D. System routines	415
9.14.1. Step 1: Setting up the connection	322	D.1. Numeric Routines	416
9.14.2. Step 2: Defining MDM schema	324	D.1.1. How to create a Sequence	416
9.15. Setting up a Web Service schema	338	D.1.2. How to convert an Implied Decimal	416
9.15.1. Setting up a simple schema.....	338	D.2. Relational Routines	417
9.16. Setting up an FTP connection	341	D.3. StringHandling Routines	417
9.16.1. Step 1: General properties	341	D.3.1. How to store a string in alphabetical order	418
9.16.2. Step 2: Connection	342	D.3.2. How to check whether a string is alphabetical	419
9.17. Exporting Metadata as context	344	D.3.3. How to replace an element in a string	419
Chapter 10. Managing routines	345	D.3.4. How to check the position of a specific character or substring, within a string	419
10.1. What are routines	346	D.3.5. How to calculate the length of a string	419
10.2. Accessing the System Routines	346	D.3.6. How to delete blank characters	420
10.3. Customizing the system routines	347	D.4. TalendDataGenerator Routines	420
10.4. Managing user routines	348	D.4.1. How to generate fictitious data	421
10.4.1. How to create user routines.....	348	D.5. TalendDate Routines	421
10.4.2. How to edit user routines	350	D.5.1. How to format a Date	422
10.4.3. How to edit user routine libraries	350	D.5.2. How to check a Date	423
10.5. Calling a routine from a Job	352	D.5.3. How to compare Dates	423
10.6. Use case: Creating a file for the current date	352	D.5.4. How to configure a Date	423
Chapter 11. Using SQL templates	355		
11.1. What is ELT	356		
11.2. Introducing Talend SQL templates....	356		
11.3. Managing Talend SQL templates	356		
11.3.1. Types of system SQL templates	357		

D.5.5. How to parse a Date	424
D.5.6. How to retrieve part of a Date..	424
D.5.7. How to format the Current Date	424
D.6. TalendString Routines	425
D.6.1. How to format an XML string..	425
D.6.2. How to trim a string	426
D.6.3. How to remove accents from a string	426

Preface

1. General information

1.1. Purpose

This User Guide explains how to manage *Talend Open Studio for ESB* functions in a normal operational context.

Information presented in this document applies to *Talend Open Studio for ESB* releases beginning with **5.1.x**.

1.2. Audience



This guide is for users and administrators of *Talend Open Studio for ESB*.



The layout of GUI screens provided in this document may vary slightly from your actual GUI.

1.3. Typographical conventions

This guide uses the following typographical conventions:

- text in **bold**: window and dialog box buttons and fields, keyboard keys, menus, and menu and options,
- text in **[bold]**: window, wizard, and dialog box titles,
- text in `courier`: system parameters typed in by the user,
- text in *italics*: file, schema, column, row, and variable names,
- The  icon indicates an item that provides additional information about an important point. It is also used to add comments related to a table or a figure,
- The  icon indicates a message that gives information about the execution requirements or recommendation type. It is also used to refer to situations or information the end-user need to be aware of or pay special attention to.

2. History of changes

The following table lists changes made in the *Talend Open Studio for ESB User Guide*.

Version	Date	History of Changes
v5.0_a	12/12/2011	Updates in <i>Talend Open Studio for ESB User Guide</i> include: <ul style="list-style-type: none">• Post-migration restructuring

Version	Date	History of Changes
		<ul style="list-style-type: none"> Updated documentation to reflect new product names. For further information on these changes, see Talend's website. Removed the prerequisite chapter. Updated chapter: Getting Started with <i>Talend Open Studio for ESB</i> Updated chapter: Designing a Job Updated chapter: Designing a Route Added chapter: Designing a Service Updated chapter: Managing Jobs, Routes and Services Updated chapter: Mapping data flows Updated chapter: Managing Metadata Updated appendix: Theory into practice: Data Service example Updated appendix: Camel components
v5.0_b	13/02/2012	<p>Updates in <i>Talend Open Studio for ESB User Guide</i> include:</p> <ul style="list-style-type: none"> Separated the appendix for Camel components from the User Guide to form a new <i>Talend Open Studio for ESB Mediation Components Reference Guide</i>. Updated sections: How to set the Runtime options and How to edit a WSDL file. Added legal notices to the User Guide. Updated the formatting of part of the User Guide.
v5.1_a	28/05/2012	<p>Updates in <i>Talend Open Studio for ESB User Guide</i> include:</p> <ul style="list-style-type: none"> Updated the <i>Enterprise Service Bus and Talend Studio</i> chapter. Added descriptions about multiple loop elements and setting the root element as loop element in using tXMLMap. Updated screenshots and descriptions about setting up a Copybook connection. Updated appendix: Theory into practice: Data Service example.
v5.1_b	05/07/2012	<p>Updates in <i>Talend Open Studio for ESB User Guide</i> include:</p> <ul style="list-style-type: none"> Updated section: tXMLMap operation. Added section: How to use Camel components in a Route. Updated appendix: Theory into practice: Data service and routing examples.

3. Feedback and Support

Your feedback is valuable. Do not hesitate to give your input, make suggestions or requests regarding this documentation or product and find support from the **Talend** team, on **Talend**'s Forum website at:

<http://talendforge.org/forum>

Chapter 1. Enterprise Service Bus and Talend Studio

Powered by the leading Apache open source integration projects, *Talend ESB* is a versatile and flexible, enterprise service bus (ESB) that allows organizations to address any integration challenge – from simple departmental projects to complex, heterogeneous IT environments. *Talend ESB* makes enterprise-class integration accessible by delivering a cost-effective and easy-to-use way to integrate and expand systems and applications.

Talend ESB is a standards-based connectivity layer used to integrate distributed systems across functional, enterprise, and geographic boundaries. Capabilities include messaging, Web services, intelligent routing, and data transformation. Its modular, pluggable architecture allows it to be easily expanded to suit most enterprise requirements.

Talend ESB is also the first to combine application integration with data management to allow businesses to cope with the volume of data in an increasingly connected world. *Talend ESB*, through *Talend Open Studio for ESB*, enables developers to easily build reliable, scalable and secure REST, Web and data services to quickly integrate heterogeneous IT environments.

Talend Open Studio for ESB is an innovative, Eclipse-based tooling environment for modeling, configuring, and deploying integration solutions using the Apache-based open source Enterprise Service Bus, *Talend ESB*. *Talend Open Studio for ESB* speeds time to deployment by making developers more productive, allowing them to rapidly respond to integration requirements.

From *Talend Open Studio for ESB*, developers can increase the success of projects by identifying issues through automated tests during development, use Enterprise Integration Patterns to mediate messages between services without knowledge of deployment or container configuration details and quickly deploy Web services, data services, REST applications and mediation routes and manage upgrades through a centralized deployment console.

1.1. Versatile and flexible Web services

Talend Open Studio for ESB is made of the **Talend ESB Integration** perspective allowing users to effortlessly combine data integration with a Web service using a graphical palette of components and connectors, and to quickly develop, build, test and publish secure Java Web services and REST applications through an easy-to-use, drag-and-drop graphical interface. Users can leverage a WSDL-first approach to integration to efficiently deploy multiple data services behind a single interface on a common ESB runtime. *Talend Open Studio for ESB* provides access to a library of over 450 connectors supporting all types of sources and targets for data integration, migration or synchronization. For more information on how to create Services, see [Chapter 6, *Designing a Service*](#). For more information about **Talend** components and more specifically about ESB components, see *Talend Open Studio Components Reference Guide*.

1.2. Dynamic mediation and intelligent routing

Integrated business operations involve many complex relationships among various systems and applications. The decision on what to process, when and where, can change rapidly. *Talend ESB* provides dynamic mediation and routing capabilities based on enterprise integration patterns (EIPs) to meet the demands of these ever-changing systems. Mediation and routing decisions are based on architecture, business rules, and deployed enterprise integration patterns. *Talend ESB* enables routing decisions at the endpoint. This eliminates bottlenecks and results in a scalable, high performance solution.

Talend Open Studio for ESB is made of the **Talend ESB Mediation** perspective for the design of integration solutions based on Enterprise Integration Patterns (EIPs) – a standard set of integration templates used to address standard integration needs. For use with *Talend ESB* products or the Apache Camel project, the **Mediation** perspective allows users to drag-and-drop EIPs from a Palette gathering all standard EIPs and configure them, eliminating the need to write code or use a domain specific language (DSL). Visibility into live statistics of message flow activity within a specific EIP enables rapid prototyping for developers, dramatically reducing development and testing cycles. For more information on how to create and manage Routes, see [Chapter 5, *Designing a Route*](#) and [Chapter 7, *Managing Jobs, Routes and Services*](#). And for more information about EIPs, see *Talend Open Studio for ESB Mediation Components Reference Guide*.

1.3. Flexible deployment

Talend ESB integrates and enhances proven open source technologies including Apache ESB software, Apache Camel, and Apache CXF, distributed as a single package, pre-configured out-of-the-box for common deployment in all popular Java environments. This way, *Talend ESB* saves users' time and effort, and minimizes the chance of set-up errors. For more information about deployment, see [Section 6.2.4, “How to export a Service for deployment”](#) and [Section 7.2.2.4, “How to export Jobs as OSGI Bundle For ESB”](#).

1.4. Live statistics and real-time debugging

Talend Open Studio for ESB includes powerful testing, debugging and tuning features that allow the real-time tracking of data flowing through the whole transformation processes, including execution statistics and an advanced trace mode.

Talend Open Studio for ESB also streamlines development by providing visibility into live statistics of message flow activity within a specific messaging route. You can simulate messaging routes from within Eclipse and

observe the execution flow, with statistics on each aspect of the route that is executed for each message. This enables rapid prototyping for developers, dramatically reducing development and testing cycles.

And, of course, all code generated by *Talend Open Studio for ESB*, regardless of the target language, is always visible and accessible from the design environment.

1.5. System monitoring

Talend ESB includes **Service Activity Monitoring** to support the drilldown into inter-process application and data service events; this event-based monitoring capability tracks a message across multiple service invocations and allows users to perform in-depth analysis to address service level agreements (SLAs) for synchronous and asynchronous messaging patterns. This can be used to analyze service response times, identify traffic patterns, perform determine root cause analysis and more. For more information about Service Activity Monitoring options in *Talend Open Studio for ESB*, see the [Section 6.3.2, “How to set the Runtime options”](#) of the present guide and for more detailed informations about Service Activity Monitoring, see the *Infrastructure Services Configuration Guide*.

1.6. Service security

Talend ESB allows ‘trust’ to be established between parties using the latest Security standards for Web Services. Through the STS Framework, clients and services securely and transparently authenticate during connections – without custom coding. Abstracting authentication through brokers allows trust to be established through a common trusted entity in situations where parties cannot establish trust directly. For more information about Security Token Service options in *Talend Open Studio for ESB*, see the [Section 6.3.2, “How to set the Runtime options”](#) of the present guide and for more detailed informations about Security Token Service, see the *Infrastructure Services Configuration Guide* and *Security Token Service User Guide*.

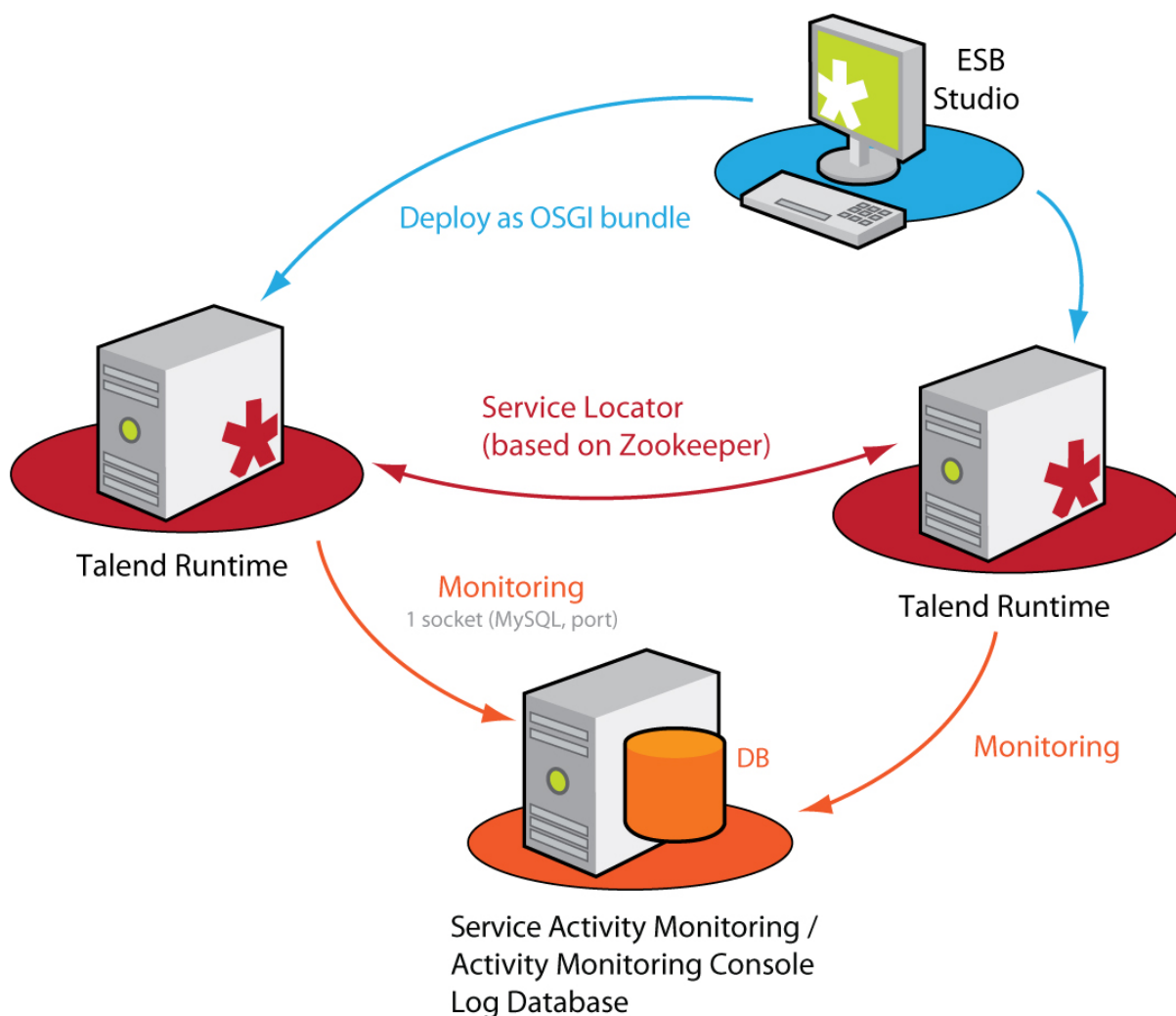
1.7. Failover and load balancing

Through the Service Locator, *Talend ESB* provides automatic and transparent failover and load balancing via dynamic endpoint registration and lookup through Apache Zookeeper. Service Locator maintains the availability of the service to help meet demands and service level agreements (SLAs). For more information about Service Locator options in *Talend Open Studio for ESB*, see the [Section 6.3.2, “How to set the Runtime options”](#) of the present guide and for more detailed informations about Service Locator, see the *Infrastructure Services Configuration Guide*.

1.8. *Talend Open Studio for ESB* functional architecture

Talend Open Studio for ESB functional architecture is an architectural model that identifies *Talend Open Studio for ESB* functions, interactions and corresponding IT needs. The overall architecture has been described by isolating specific functionalities in different functional blocks.

The below chart illustrates the main architectural functional blocks explored within *Talend Open Studio for ESB*.



Three different types of functional block are defined:

- The **blue block** represents a Studio API where you can carry out data integration or data service processes, mediation routes and services. For more information, see their respective chapters of the present user guide.
- The **red blocks** represent one or more Talend Runtimes (execution container) deployed inside your information system. Talend Runtime enables you to deploy and execute the Jobs, Routes and Services created in the Studio. For more information on how to deploy items in Talend Runtime, see the [Section 6.2.4, “How to export a Service for deployment”](#) and for more information about Talend Runtime itself, see the *Talend ESB Infrastructure Services Configuration Guide*.

If you have several Talend Runtimes on which to deploy the Services and Routes, you will be able to load balance their execution according to your needs. All instances of Talend Runtime will communicate between each other via the Service Locator to identify the one more likely to deploy and execute them.

- The **orange block** represents a monitoring database gathering log information of the execution of your data processes and service activity.

Data processes log information can be captured with the use of the **tFlowMeterCatcher**, **tStatCatcher**, **tLogCatcher** components. For more information, see the *Talend Open Studio Components Reference Guide*. And to automate the functionalities of the **tFlowMeterCatcher**, **tStatCatcher**, **tLogCatcher** components without using them, you can use the Stats & Logs tab. For more information regarding Stats & Logs, see the [Section 4.6.7.1, “How to automate the use of statistics & logs”](#) of the present guide.

The Service Activity Monitoring allows the end-users to monitor service calls. It provides monitoring and consolidated event information that the end-user can use to understand the underlying requests and replies

that compose the event, monitor faults that may be unexpectedly generated and support the system management decisions. For more information on the Service Activity Monitoring, see its corresponding chapter in *Talend ESB Infrastructure Services Configuration Guide*.

For more information on the installation of all these components, see the *Talend ESB Getting Started Guide* or see the Installation Guide available on the Talend ESB download page <http://www.talend.com/download.php>.



Chapter 2. Getting started with Talend Studio

This chapter introduces *Talend Open Studio for ESB*. It provides basic configuration information required to get started with *Talend Open Studio for ESB*.

The chapter guides you through the basic steps in creating local projects. It also describes how to set preferences and customize the workspace in *Talend Open Studio for ESB*.

Before starting any data integration processes, you need to be familiar with *Talend Open Studio for ESB* Graphical User Interface (GUI). For more information, see [Appendix A, GUI](#).

2.1. Important concepts in *Talend Open Studio for ESB*

When working with *Talend Open Studio for ESB*, you will often come across words such as repository, project, workspace, Job, data service Job, Route, Service, component and item.

Understanding the concept behind each of these words is crucial to grasping the functionality of *Talend Open Studio for ESB*.

What is a repository? A repository is the storage location *Talend Open Studio for ESB* uses to gather data related to all of the technical items that you use either to describe business models or to design Jobs.

What is a project? Projects are structured collections of technical items and their associated metadata. All of the Jobs and business models you design are organized in Projects.

You can create as many projects as you need in a repository. For more information about projects, see [Section 2.4, “Working with projects”](#).

What is a workspace? A workspace is the directory where you store all your project folders. You need to have one workspace directory per connection (repository connection). *Talend Open Studio for ESB* enables you to connect to different workspace directories, if you do not want to use the default one.

For more information about workspaces, see [Section 2.3, “Working with different workspace directories”](#).

What is a Job? A Job is a graphical design, of one or more components connected together, that allows you to set up and run dataflow management processes. It translates business needs into code, routines and programs. Jobs address all of the different sources and targets that you need for data integration processes and all other related processes.

What is a data service Job? A data service Job is a graphical design, of one or more components connected together, that allows you to set up and run data service processes. It translates business needs into code, routines and programs. Jobs address all of the different sources and targets that you need for data integration processes and combine it with Web services.

For detailed information about how to design data integration processes in *Talend Open Studio for ESB*, see [Chapter 4, *Designing a Job*](#).



Data service Jobs will simply be referred to as Jobs in the following documentation.

What is a Route? A camel Route is a graphical design, based on Apache Camel framework, of two or more components connected together that allows you to set up and run routing and mediation rules. A routing rule defines how messages will be moved from one service (or endpoint) to another.

What is a Service? A Service is a graphical design, of several WSDL objects (service, binding, port type and so on) linked together, that allows you to set up and implement Web services. A Service is associated with one or more data service Jobs as the service provider and can be consumed by consumer Jobs.

What is a component? A component is a preconfigured connector used to perform a specific data integration operation, no matter what data sources you are integrating: databases, applications, flat files, Web services, etc. A component can minimize the amount of hand-coding required to work on data from multiple, heterogeneous sources.

Components are grouped in families according to their usage and displayed in the **Palette** of the *Talend Open Studio for ESB* main window.

For detailed information about components types and what they can be used for, see the *Talend Open Studio for ESB* Reference Guide.

What is an item? An item is the fundamental technical unit in a project. Items are grouped, according to their types, as: Job Design, Business model, Context, Code, Metadata, etc. One item can include other items. For example, the business models and the Jobs you design are items, metadata and routines you use inside your Jobs are items as well.

2.2. Launching *Talend Open Studio for ESB*

2.2.1. How to launch the Studio for the first time

To open *Talend Open Studio for ESB* for the first time, complete the following:

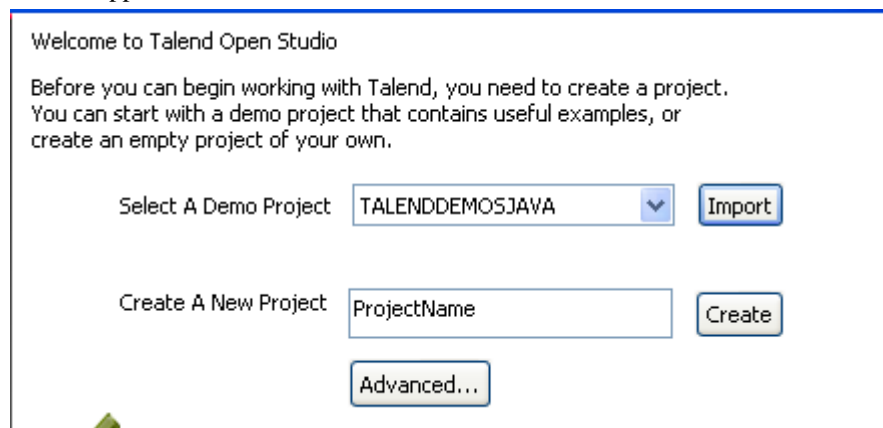
1. Unzip the *Talend Open Studio for ESB* zip file and, in the folder, double-click the executable file corresponding to your operating system.



The Studio zip archive contains binaries for several platforms including Mac OS X and Linux/Unix.

2. In the **[License]** window that appears, read and accept the terms of the end user license agreement to continue.

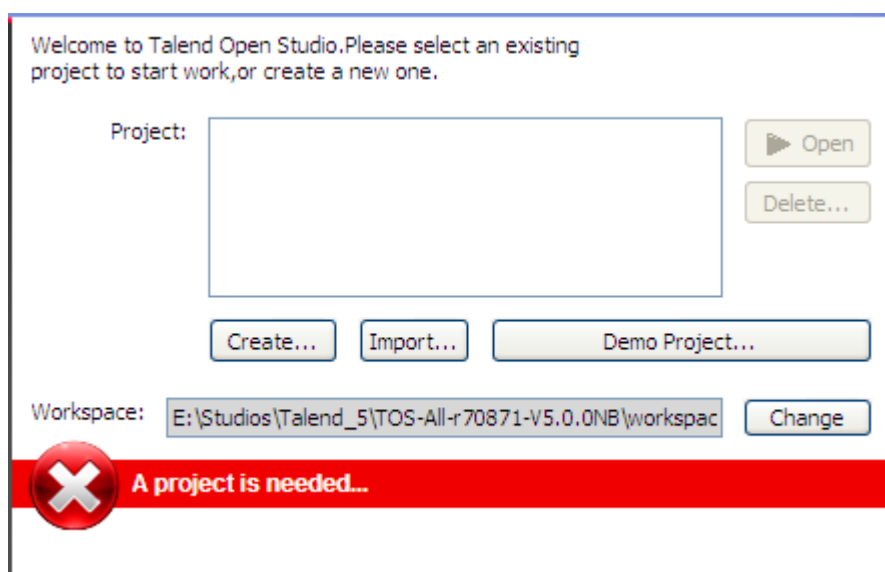
The startup window appears.



This screen appears only when you launch the *Talend Open Studio for ESB* for the first time or if all existing projects have been deleted.

3. Click the **Import** button to import the selected demo project, or type in a project name in the **Create A New Project** field and click the **Create** button to create a new project, or click the **Advanced...** button to go to the Studio login window.

In this procedure, click **Advanced...** to go to the Studio login window. For more information about the other two options, see [Section 2.4.2, “How to import the demo project”](#) and [Section 2.4.1, “How to create a project”](#) respectively.

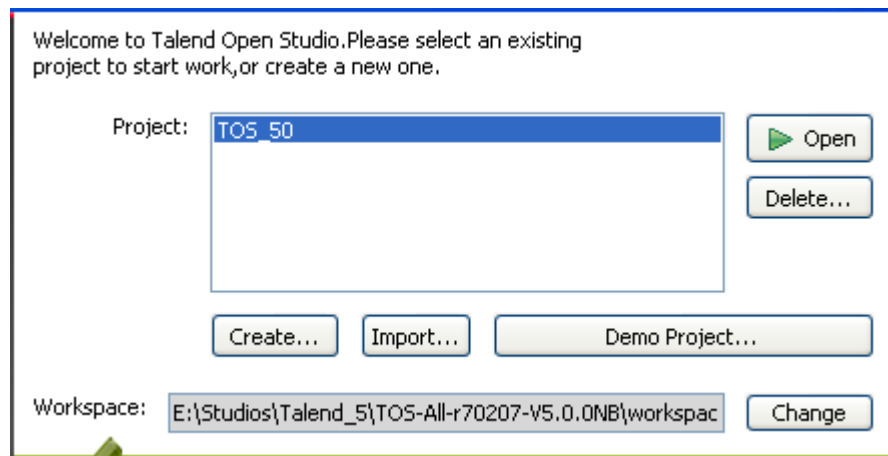


4. From the Studio login window:

Click...	To...
Create...	create a new project that will hold all Jobs and Business models designed in the Studio. For more information, see Section 2.4.1, “How to create a project” .
Import...	import one or more existing projects. For more information, see Section 2.4.3, “How to import projects” .
Demo Project...	import the Demo project including numerous samples of ready-to-use Jobs. This Demo project can help you understand the functionalities of different Talend components. For more information, see Section 2.4.2, “How to import the demo project” .
Open	open the selected existing project. For more information, see Section 2.4.4, “How to open a project” .
Delete...	open a dialog box in which you can delete any created or imported project that you do not need anymore. For more information, see Section 2.4.5, “How to delete a project” .

As the purpose of this procedure is to create a new project, click **Create...** to open the **[New project]** dialog box.

5. In the dialog box, enter a name for your project and click **Finish** to close the dialog box. The name of the new project is displayed in the **Project** list.



6. Select the project, and click **Open**.

The **Connect to TalendForge** page appears, inviting you to connect to the **Talend** Community so that you can check, download, install external components and upload your own components to the **Talend** Community to share with other **Talend** users directly in the **Exchange** view of your Job designer in the Studio.

To learn more about the **Talend** Community, click the **read more** link. For more information on using and sharing community components, see [Section 4.5.3, “How to download/upload Talend Community components”](#).

7. If you want to connect to the **Talend** Community later, click **Skip** to continue.
8. If you are working behind a proxy, click **Proxy setting** and fill in the **Proxy Host** and **Proxy Port** fields of the **Network setting** dialog box.
9. By default, the Studio will automatically collect product usage data and send the data periodically to servers hosted by **Talend** for product usage analysis and sharing purposes only. If you do not want the Studio to do so, clear the **I want to help to improve Talend by sharing anonymous usage statistics** check box.

You can also turn on or off usage data collection in the Usage Data Collector preferences settings. For more information, see [Section 2.5.15, “Usage Data Collector preferences”](#).

10. Fill in the required information, select the **I Agree to the TalendForge Terms of Use** check box, and click **Create Account** to create your account and connect to the **Talend** Community automatically. If you already have created an account at <http://www.talendforge.org>, click the **or connect on existing account** link to sign in.



Be assured that any personal information you may provide to **Talend** will never be transmitted to third parties nor used for any purpose other than joining and logging in to the **Talend** Community and being informed of **Talend** latest updates.

Connect to TalendForge

Connect your studio to TalendForge, the Talend online community.

- Download new components and connectors from Talend Exchange
- Access the most recent documentation and tech articles from the Talend social knowledgebase.
- See the latest messages in the Talend discussion forums

Create an account (or connect on existing account):

Username: testuser *

Email: testuser@company.com *

Password: ***** *

Password(again): ***** *

Country: United States ▼

☒ I Agree to the TalendForge Terms of Use

☒ I want to help to improve Talend by sharing anonymous usage statistics

[\(read more...\)](#)

Create Account

Proxy settings... Skip



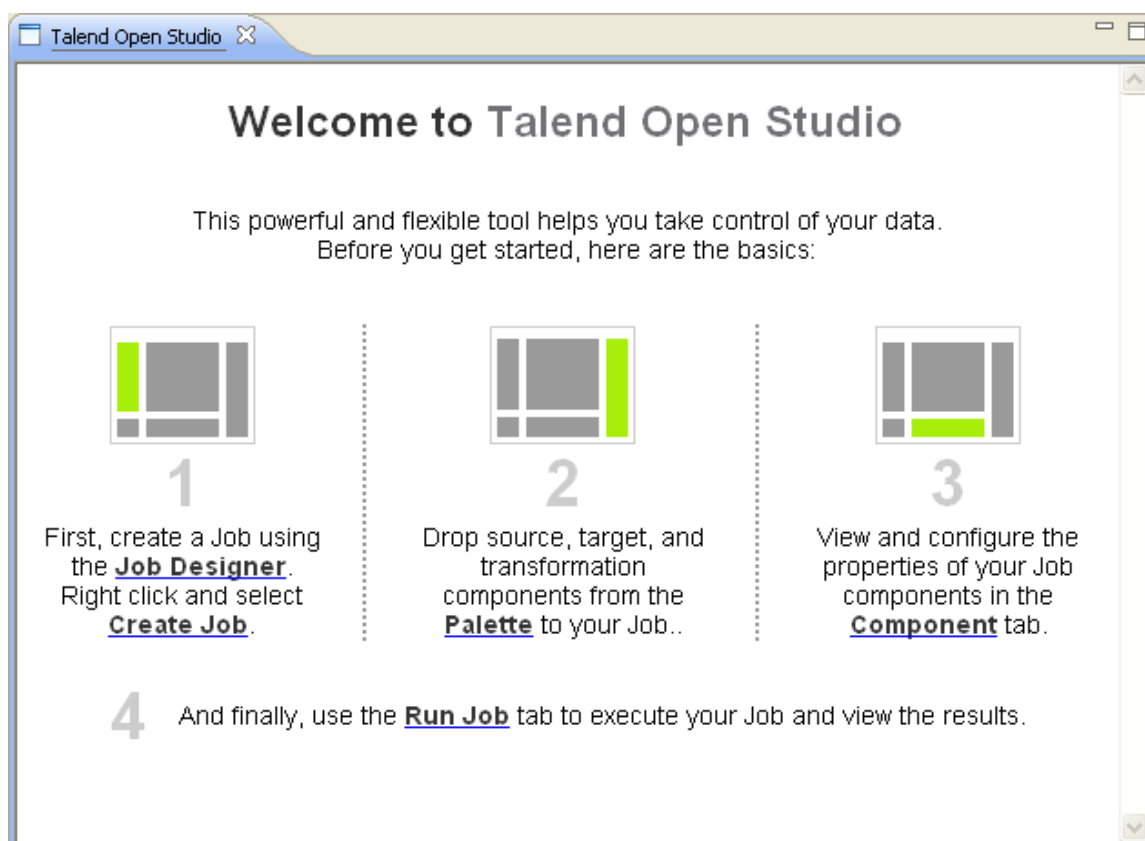
This page will not appear again at Studio startup once you successfully connect to the **Talend** Community or if you click **Skip** too many times. You can show this page again from the **[Preferences]** dialog box. For more information, see [Section 2.5.3, “Exchange preferences”](#).

A progress information bar and a welcome window display consecutively. From this page you have direct links to user documentation, tutorials, **Talend** forum, **Talend Exchange** and **Talend** latest news.

11. Click **Start now!** to open *Talend Open Studio for ESB* main window.

The main window opens on a welcome page which has useful tips for beginners on how to get started with the Studio. Clicking an underlined link brings you to the corresponding tab view or opens the corresponding dialog box.

For more information on how to open a project, see [Section 2.4.4, “How to open a project”](#).



2.2.2. How to set up a project

To open the *Talend Open Studio for ESB* main window, you must first set up a project.

You can set up a project by:

- creating a new project. For more information, see [Section 2.4.1, “How to create a project”](#).
- importing one or more projects you already created in other sessions of *Talend Open Studio for ESB*. For more information, see [Section 2.4.3, “How to import projects”](#).
- importing the Demo project. For more information, see [Section 2.4.2, “How to import the demo project”](#).

2.3. Working with different workspace directories

Talend Open Studio for ESB makes it possible to create many workspace directories and connect to a workspace different from the one you are currently working on, if necessary.

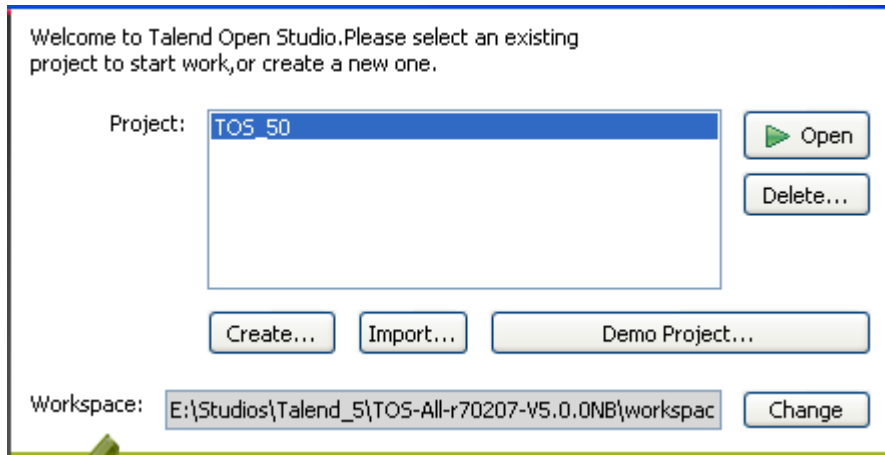
This flexibility enables you to store these directories wherever you want and give the same project name to two or more different projects as long as you store the projects in different directories.

2.3.1. How to create a new workspace directory

Talend Open Studio for ESB is delivered with a default workspace directory. However, you can create as many new directories as you want and store your project folders in them according to your preferences.

To create a new workspace directory:

1. In the project login window, click **Change** to open the dialog box for selecting the directory of the new workspace.



2. In the dialog box, set the path to the new workspace directory you want to create and then click **OK** to close the view.

On the login window, a message displays prompting you to restart the Studio.

3. Click **Restart** to restart the Studio.
4. On the re-initiated login window, set up a project for this new workspace directory.

For more information, see [Section 2.2.2, “How to set up a project”](#).

5. Select the project from the **Project** list and click **Open** to open *Talend Open Studio for ESB* main window.

All business models or Jobs you design in the current instance of the Studio will be stored in the new workspace directory you created. .

When you need to connect to any of the workspaces you have created, simply repeat the process described in this section.

2.4. Working with projects

In *Talend Open Studio for ESB*, the highest physical structure for storing all different types of data integration Jobs and business models, metadata, routines, etc. is the “project”.

From the login window of *Talend Open Studio for ESB*, you can:

- import the Demo project to discover the features of *Talend Open Studio for ESB* based on samples of different ready-to-use Jobs. When you import the Demo project, it is automatically installed in the workspace directory of the current session of the Studio.

For more information, see [Section 2.4.2, “How to import the demo project”](#).

- create a local project. When connecting to *Talend Open Studio for ESB* for the first time, there are no default projects listed. You need to create a project and open it in the Studio to store all the Jobs and business models you create in it. When creating a new project, a tree folder is automatically created in the workspace directory on your repository server. This will correspond to the **Repository** tree view displaying on *Talend Open Studio for ESB* main window.

For more information, see [Section 2.4.1, “How to create a project”](#).

- import projects you have already created with previous releases of *Talend Open Studio for ESB* into your current *Talend Open Studio for ESB* workspace directory by clicking **Import...** .

For more information, see [Section 2.4.3, “How to import projects”](#).

- open a project you created or imported in the Studio.

For more information, see [Section 2.4.4, “How to open a project”](#).

- delete local projects that you already created or imported and that you do not need any longer.

For more information, see [Section 2.4.5, “How to delete a project”](#).

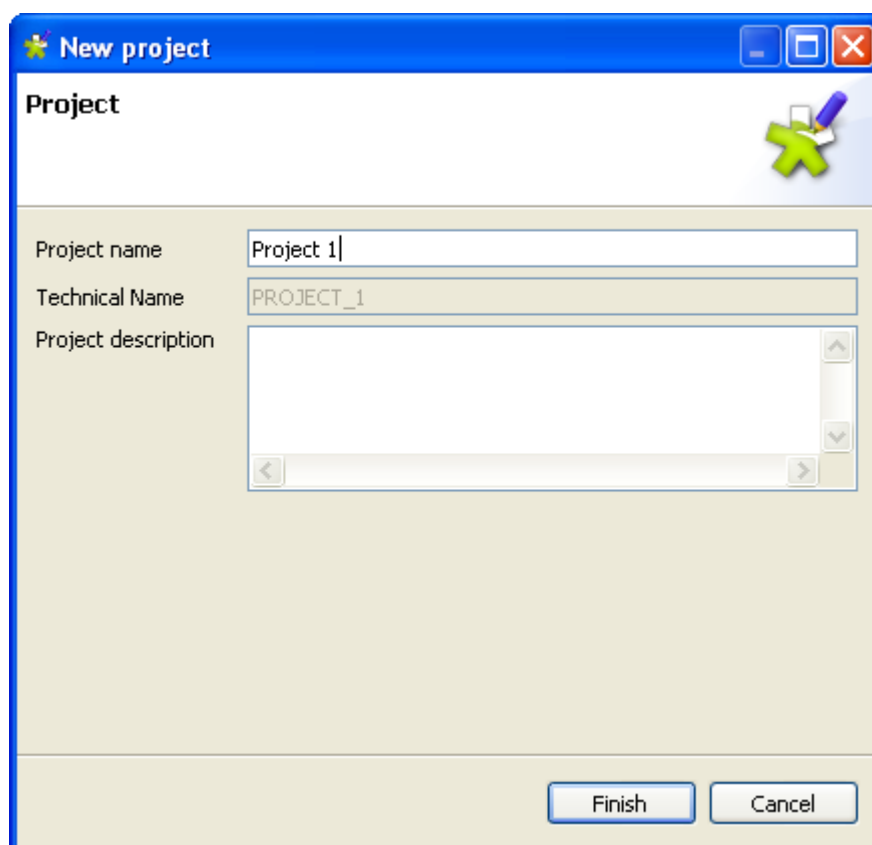
Once you launch *Talend Open Studio for ESB*, you can export the resources of one or more of the created projects in the current instance of the Studio. For more information, see [Section 2.4.6, “How to export a project”](#).

2.4.1. How to create a project

When you launch the Studio for the first time, there are no default projects listed. You need to create a project that will hold all data integration Jobs and business models you design in the current instance of the Studio.

To create a project:

1. Launch *Talend Open Studio for ESB*.
2. Use either of the following two options:
 - Enter a project name in the **Create A New Project** field and click **Create** to open the **[New project]** dialog box with the **Project name** field filled with the specified name.
 - Click **Advanced**, and then from the login window click **Create...** to open the **[New project]** dialog box with an empty **Project name** field.



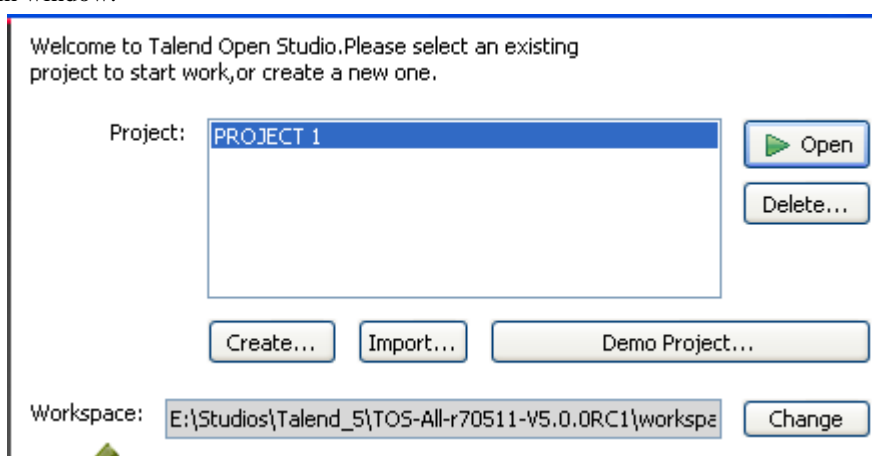
3. In the **Project name** field, enter a name for the new project, or change the previously specified project name if needed. This field is mandatory.

A message shows at the top of the wizard, according to the location of your pointer, to inform you about the nature of data to be filled in, such as forbidden characters



The read-only “technical name” is used by the application as file name of the actual project file. This name usually corresponds to the project name, upper-cased and concatenated with underscores if needed.

4. Click **Finish**. The name of the newly created project is displayed in the **Project** list in *Talend Open Studio for ESB* login window.



From version 5.0 onwards, Java is the only language generated.

To open the newly created project in *Talend Open Studio for ESB*, select it from the **Project** list and then click **Open**. A generation engine initialization window displays. Wait till the initialization is complete.

Later, if you want to switch between projects, on the Studio menu bar, use the combination **File > Switch Project**.

If you already used *Talend Open Studio for ESB* and want to import projects from a previous release, see [Section 2.4.3, “How to import projects”](#).

2.4.2. How to import the demo project

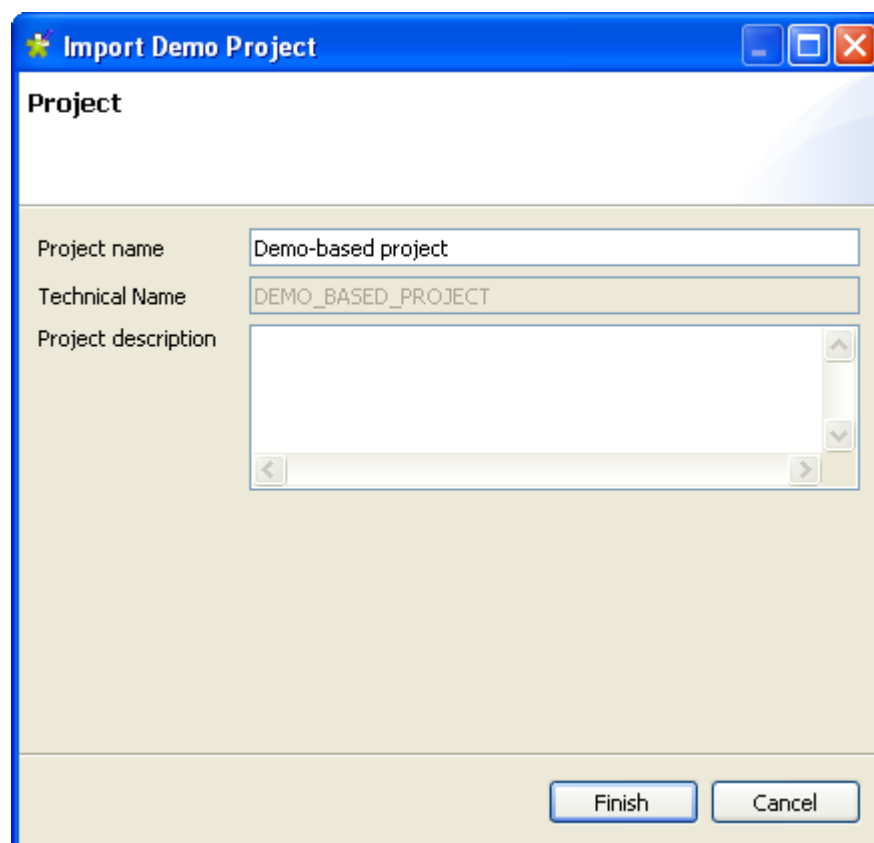
In *Talend Open Studio for ESB*, you can import the demo project that includes numerous samples of ready to use Jobs. This demo project can help you understand the functionalities of different **Talend** components.

At the first launch of *Talend Open Studio for ESB*, you can:

- create a new project in your repository using the demo project as a template,
- import the demo project *TALENDDDEMOSJAVA* into your repository.

To create a new project based on the demo project:

1. Click the **Import** button next to the **Select A Demo Project** list box. The **[Import Demo Project]** dialog box opens.



2. Type in a name for the new project, and click **Finish** to create the project.

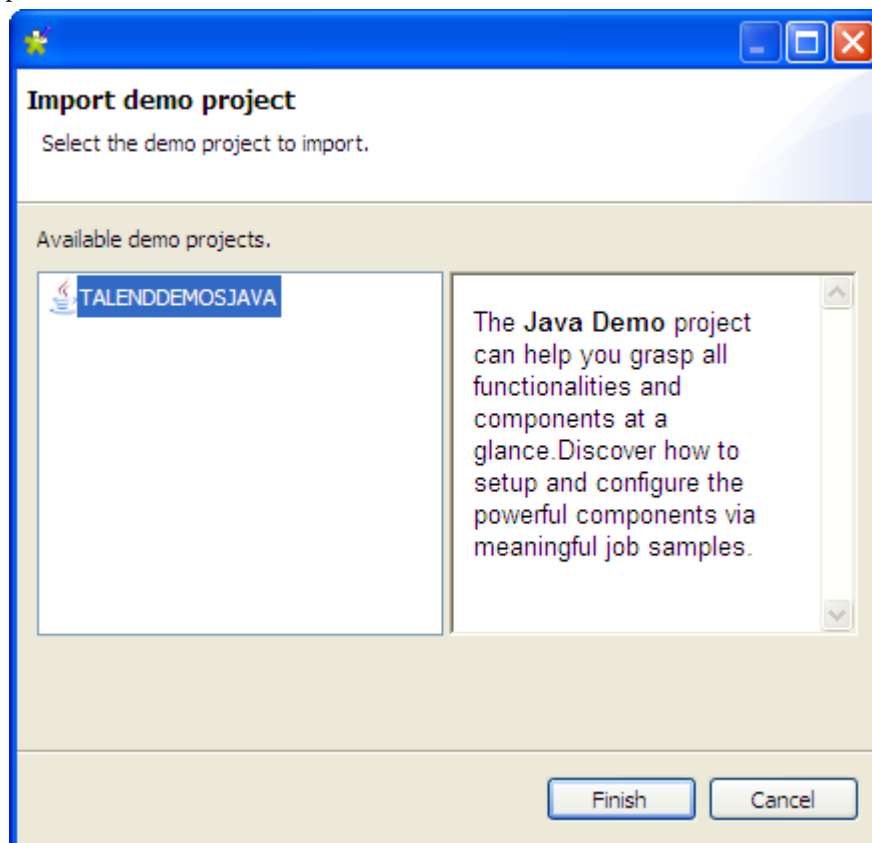
A confirmation message is displayed, informing you that the demo project has been successfully imported in the current instance of the Studio.

3. Click **OK** to close the confirmation message.

All the samples of the demo project are imported into the newly created project, and the name of the new project is displayed in the **Project** list on the login screen.

To import the demo project *TALENDDEMOSJAVA* into your repository:

1. Click **Advanced...**, and then from the login window click **Demo Project...**. The **[Import demo project]** dialog box opens.



2. Select the demo project and then click **Finish** to close the dialog box.

A confirmation message is displayed, informing you that the demo project has been successfully imported in the current instance of the Studio.

3. Click **OK** to close the confirmation message.

The imported demo project displays in the **Project** list on the login window.

To open the imported demo project in *Talend Open Studio for ESB*, select it from the **Project** list and then click **Open**. A generation engine initialization window displays. Wait till the initialization is complete.

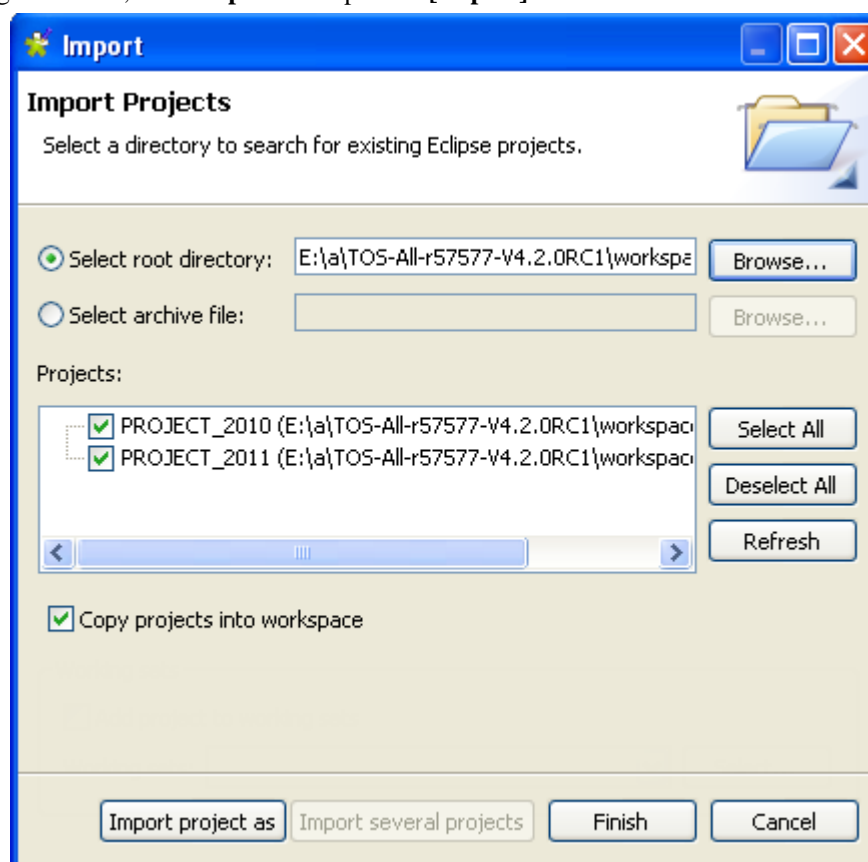
The Job samples in the open demo project are automatically imported into your workspace directory and made available in the **Repository** tree view under the **Job Designs** folder.

You can use these samples to get started with your own Job design.

2.4.3. How to import projects

In *Talend Open Studio for ESB*, you can import projects you already created with previous releases of the Studio.

1. If you are launching *Talend Open Studio for ESB* for the first time, click **Advanced...** to open to the login window.
2. From the login window, click **Import...** to open the **[Import]** wizard.



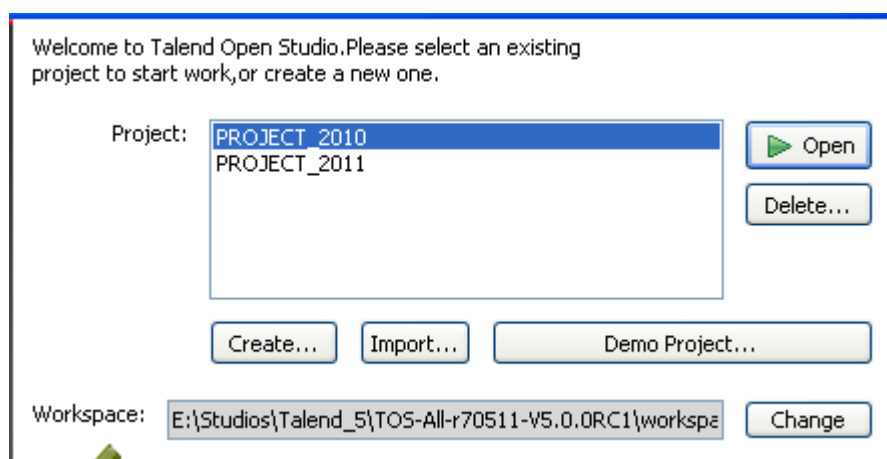
3. Click **Import several projects** if you intend to import more than one project simultaneously.
4. Click **Select root directory** or **Select archive file** depending on the source you want to import from.
5. Click **Browse...** to select the workspace directory/archive file of the specific project folder. By default, the workspace in selection is the current release's one. Browse up to reach the previous release workspace directory or the archive file containing the projects to import.
6. Select the **Copy projects into workspace** check box to make a copy of the imported project instead of moving it.



If you want to remove the original project folders from the *Talend Open Studio for ESB* workspace directory you import from, clear this check box. But we strongly recommend you to keep it selected for backup purposes.

7. From the **Projects** list, select the projects to import and click **Finish** to validate the operation.

In the login window, the names of the imported projects now appear on the **Project** list.



You can now select the imported project you want to open in *Talend Open Studio for ESB* and click **Open** to launch the Studio.



A generation initialization window might come up when launching the application. Wait until the initialization is complete.

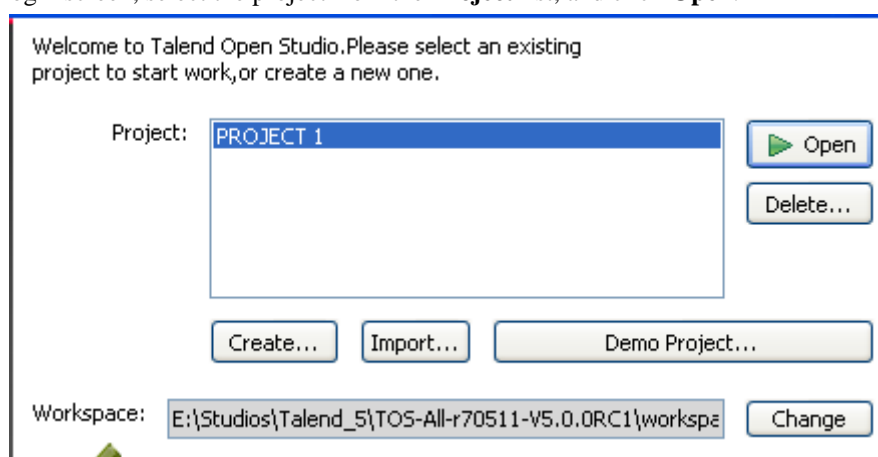
2.4.4. How to open a project



*When you launch Talend Open Studio for ESB for the first time, no project names are displayed on the **Project** list. First you need to create a project or import a Demo project in order to populate the **Project** list with the corresponding project names that you can then open in the Studio.*

To open a project in *Talend Open Studio for ESB*:

On the Studio login screen, select the project from the **Project** list, and click **Open**.



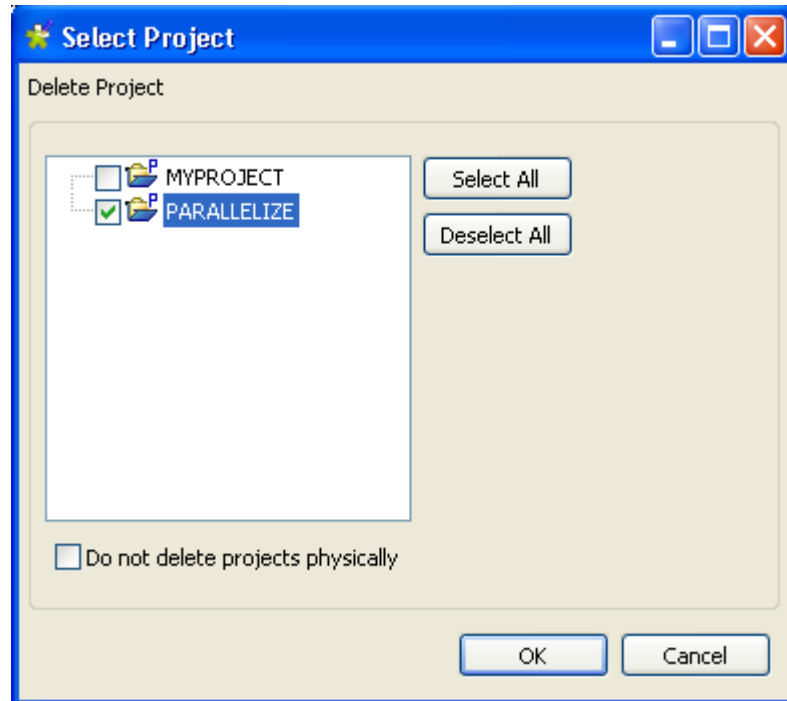
A progress bar appears, and the *Talend Open Studio for ESB* main window opens. A generation engine initialization dialog box displays. Wait till initialization is complete.



When you open a project imported from a previous version of the Studio, an information window pops up to list a short description of the successful migration tasks. For more information, see [Section 2.4.7, “Migration tasks”](#).

2.4.5. How to delete a project

1. On the login screen, click **Delete...** to open the [Select Project] dialog box.



2. Select the check box(es) of the project(s) you want to delete.
3. Click **OK** to validate the deletion.

The project list on the login window is refreshed accordingly.



*Be careful, this action is irreversible. When you click **OK**, there is no way to recuperate the deleted project(s).*

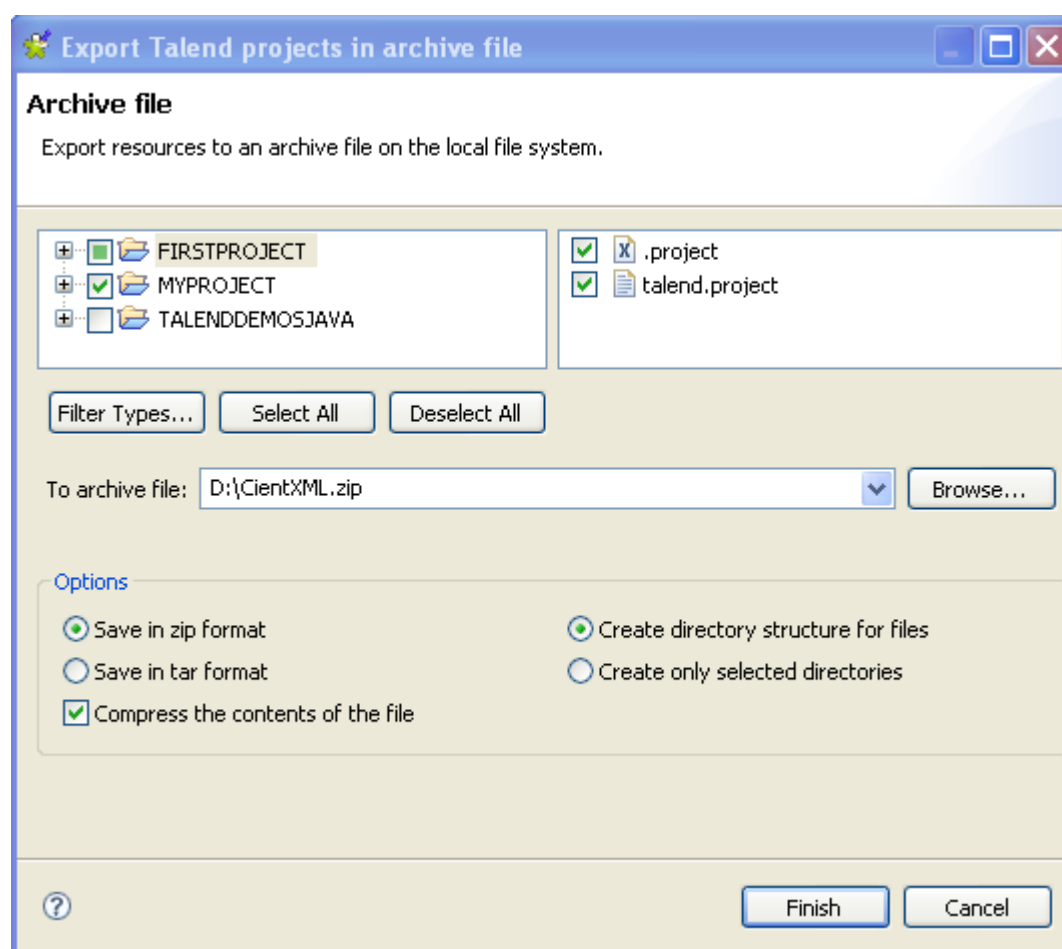


If you select the **Do not delete projects physically** check box, you can delete the selected project(s) only from the project list and still have it/them in the *workspace* directory of *Talend Open Studio for ESB*. Thus, you can recuperate the deleted project(s) any time using the **Import existing project(s) as local** option on the **Project** list from the login window.

2.4.6. How to export a project

Talend Open Studio for ESB, allows you to export projects created or imported in the current instance of *Talend Open Studio for ESB*.

1. On the toolbar of the Studio main window, click  to open the [Export Talend projects in archive file] dialog box.



2. Select the check boxes of the projects you want to export. You can select only parts of the project through the **Filter Types...** link, if need be (for advanced users).
3. In the **To archive file** field, type in the name of or browse to the archive file where you want to export the selected projects.
4. In the **Option** area, select the compression format and the structure type you prefer.
5. Click **Finish** to validate the changes.

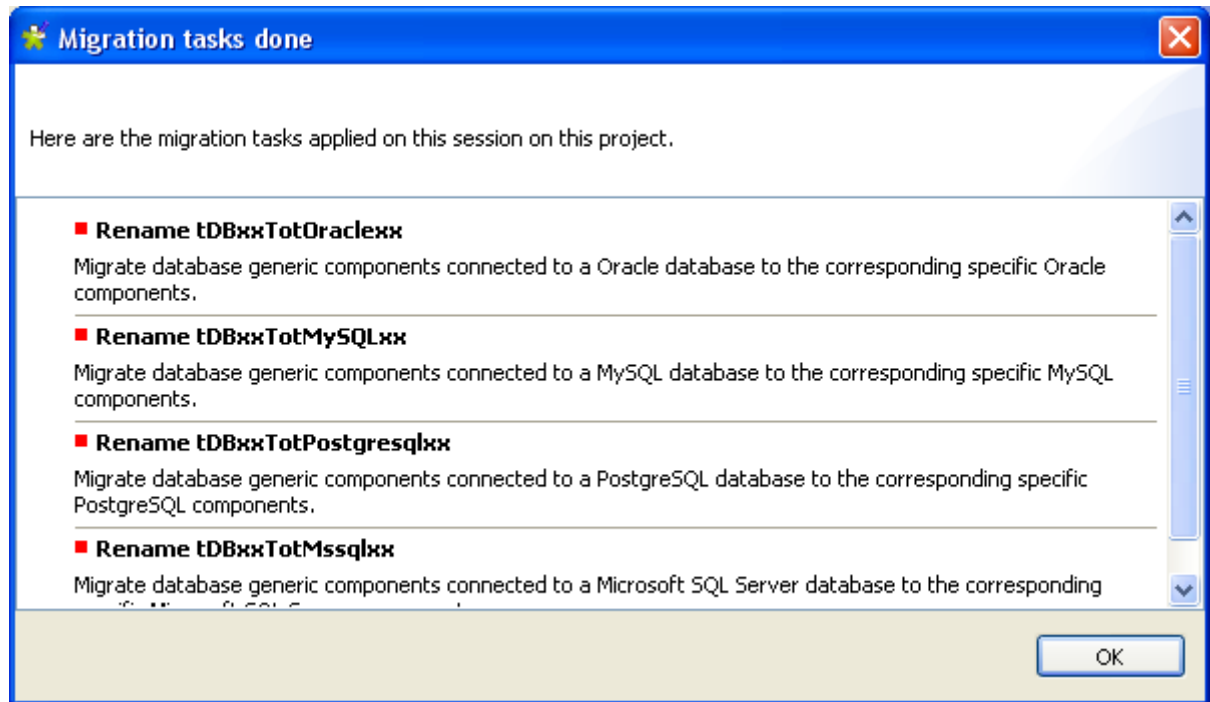
The archived file that holds the exported projects is created in the defined place.

2.4.7. Migration tasks

Migration tasks are performed to ensure the compatibility of the projects you created with a previous version of *Talend Open Studio for ESB* with the current release.

As some changes might become visible to the user, we thought we'd share these update tasks with you through an information window.

This information window pops up when you launch the project you imported (created) in a previous version of *Talend Open Studio for ESB*. It lists and provides a short description of the tasks which were successfully performed so that you can smoothly roll your projects.



Some changes that affect the usage of *Talend Open Studio for ESB* include, for example:

- **tDBInput** used with a MySQL database becomes a specific **tDBMySQLInput** component the aspect of which is automatically changed in the Job where it is used.
- **tUniqRow** used to be based on the Input schema keys, whereas the current **tUniqRow** allows the user to select the column to base the unicity on.

2.5. Setting *Talend Open Studio for ESB* preferences

You can define various properties of *Talend Open Studio for ESB* main design workspace according to your needs and preferences.

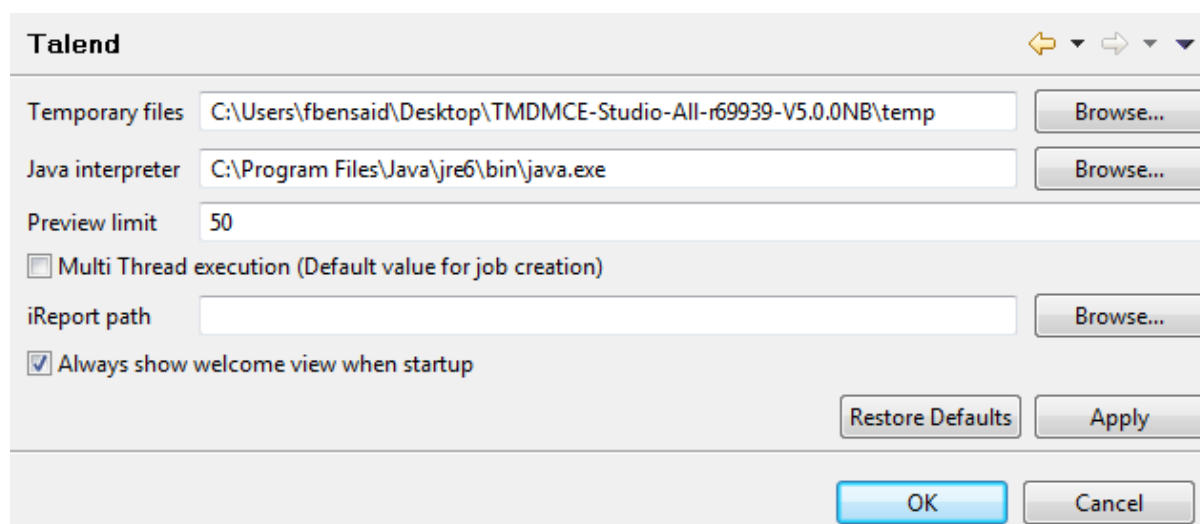
Numerous settings you define can be stored in the **Preference** and thus become your default values for all new Jobs you create.

The following sections describe specific settings that you can set as preference.

First, click the **Window** menu of your *Talend Open Studio for ESB*, then select **Preferences**.

2.5.1. Java Interpreter path

The Java Interpreter path is set default in the Java file of your computer (by default Program Files\Java\jre6\bin\java.exe).



To customize your Java Interpreter path:

1. If needed, click the **Talend** node in the tree view of the **[Preferences]** dialog box.
2. Enter a path in the **Java interpreter** field if the default directory does not display the right path.

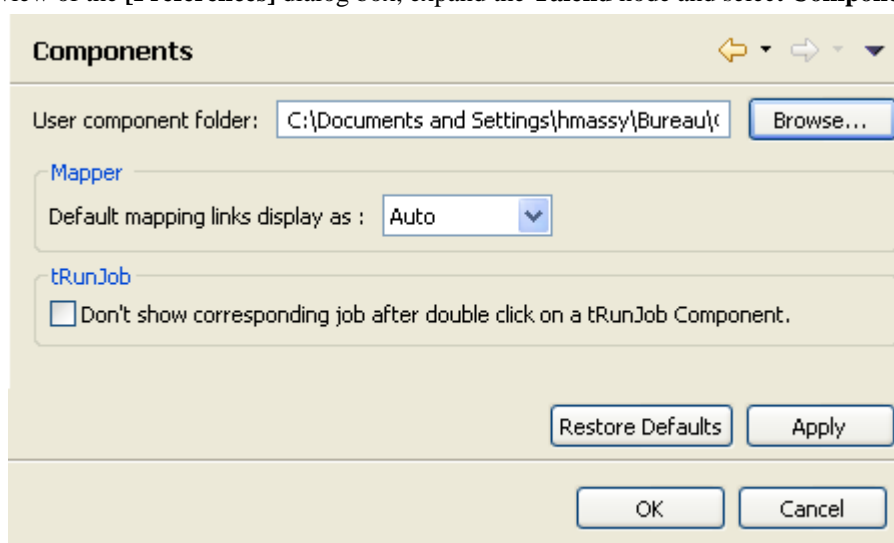
On the same view, you can also change the preview limit and the path to the temporary files or the OS language.

2.5.2. External or User components

You can create and develop your own components for use in *Talend Open Studio for ESB*.

For further information about the creation and development of user components, refer to the component creation tutorial on our wiki at http://www.talendforge.org/wiki/doku.php?id=component_creation.

1. In the tree view of the **[Preferences]** dialog box, expand the **Talend** node and select **Components**.



2. Enter the **User components folder** path or browse to the folder that holds the components to be added to the *Talend Open Studio for ESB Palette*.
3. From the **Default mapping links display as** list, select the mapping link type you want to use in the **tMap**.

- Under **tRunJob**, select the check box if you do not want the corresponding Job to open upon double clicking a **tRunJob** component.



You will still be able to open the corresponding Job by right clicking the **tRunJob** component and selecting **Open tRunJob Component**.

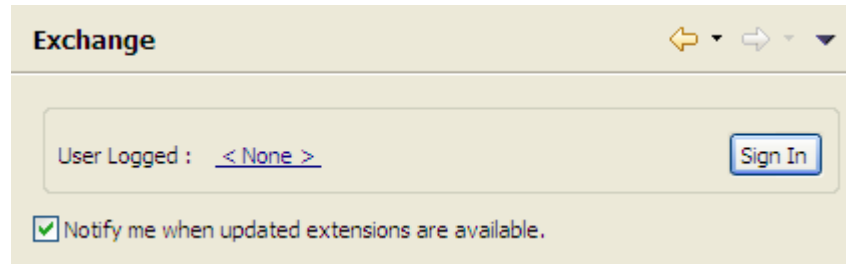
- Click **Apply** and then **OK** to validate the set preferences and close the dialog box.

The external components are added to the **Palette**.

2.5.3. Exchange preferences

You can set preferences related to your connection with **Talend** Exchange, which is part of the **Talend** Community, in *Talend Open Studio for ESB*. To do so:

- From the menu bar, click **Window > Preferences** to open the **[Preferences]** dialog box.
- Expand the **Talend** node and click **Exchange** to display the **Exchange** view.



- Set the Exchange preferences according to your needs:
 - If you are not yet connected to the **Talend** Community, click **Sign In** to go to the **Connect to TalendForge** page to sign in using your **Talend** Community credentials or create a **Talend** Community account and then sign in.

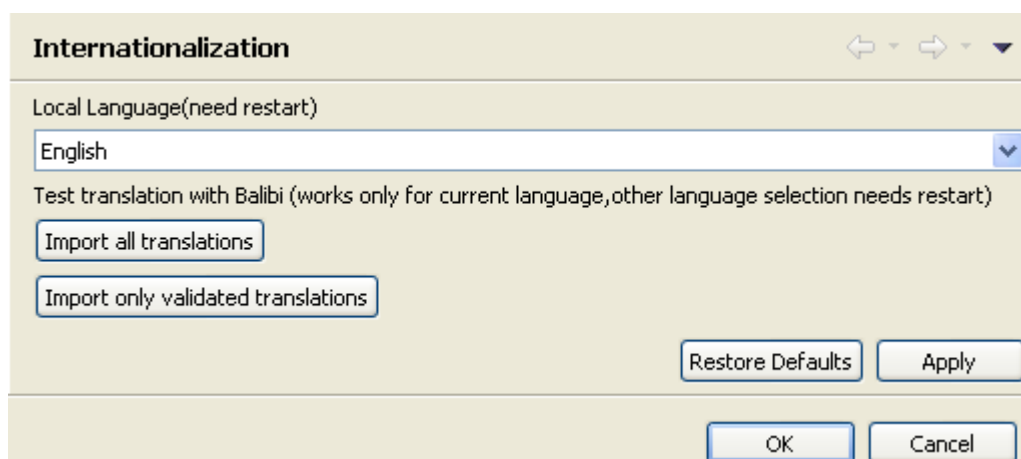
If you are already connected to the **Talend** Community, your account is displayed and the **Sign In** button becomes **Sign Out**. To get disconnected from the **Talend** Community, click **Sign Out**.
 - By default, while you are connected to the **Talend** Community, whenever an update to an installed community extension is available, a dialog box appears to notify you about it. If you often check for community extension updates and you do not want that dialog box to appear again, clear the **Notify me when updated extensions are available** check box.

For more information on connecting to the **Talend** Community, see [Section 2.2, “Launching Talend Open Studio for ESB”](#). For more information on using community extensions in the Studio, see [Section 4.5.3, “How to download/upload Talend Community components”](#).

2.5.4. Language preferences

You can set language preferences in *Talend Open Studio for ESB*. To do so:

- From the menu bar, click **Window > Preferences** to open the **[Preferences]** dialog box.
- Expand the **Talend** node and click **Internationalization** to display the relevant view.

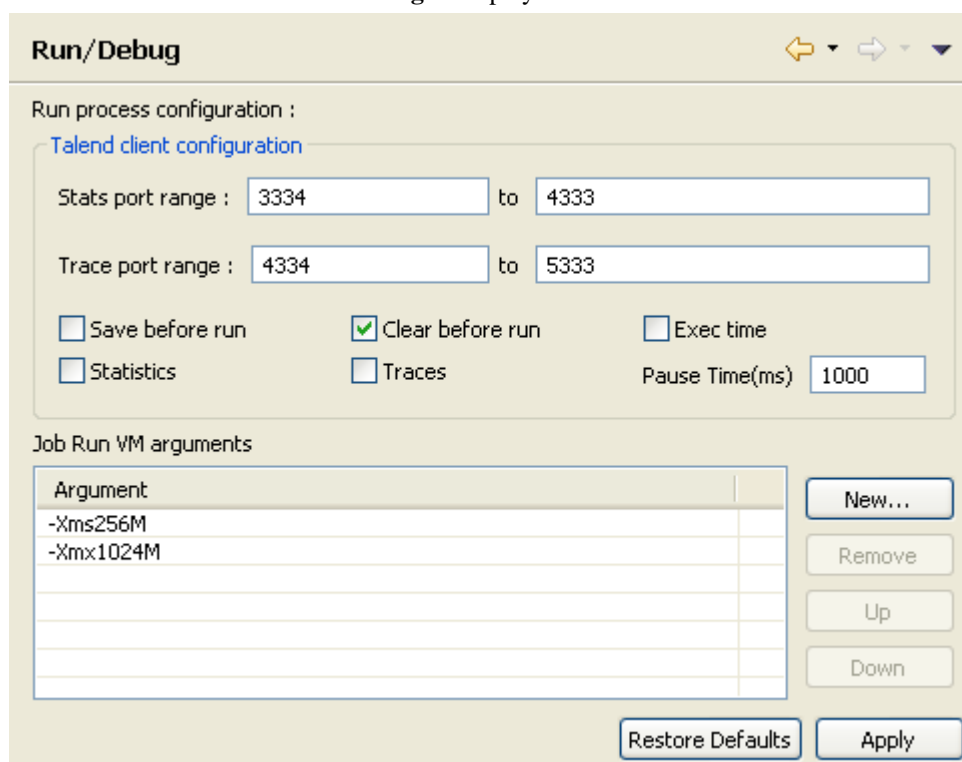


3. From the **Local Language** list, select the language you want to use for *Talend Open Studio for ESB* graphical interface.
4. Click **Apply** and then **OK** to validate your change and close the **[Preferences]** dialog box.
5. Restart *Talend Open Studio for ESB* to display the graphical interface in the selected language.

2.5.5. Debug and Job execution preferences

You can set your preferences for debug and job executions in *Talend Open Studio for ESB*. To do so:

1. From the menu bar, click **Window > Preferences** to display the **[Preferences]** dialog box.
2. Expand the **Talend** node and click **Run/Debug** to display the relevant view.



- In the **Talend client configuration** area, you can define the execution options to be used by default:

Stats port range	Specify a range for the ports used for generating statistics, in particular, if the ports defined by default are used by other applications.
Trace port range	Specify a range for the ports used for generating traces, in particular, if the ports defined by default are used by other applications.
Save before run	Select this check box to save your Job automatically before its execution.
Clear before run	Select this check box to delete the results of a previous execution before re-executing the Job.
Exec time	Select this check box to show Job execution duration.
Statistics	Select this check box to show the statistics measurement of data flow during Job execution.
Traces	Select this check box to show data processing during job execution.
Pause time	Enter the time you want to set before each data line in the traces table.

- In the **Job Run VM arguments** list, you can define the parameter of your current JVM according to your needs. The by-default parameters **-Xms256M** and **-Xmx1024M** correspond respectively to the minimal and maximal memory capacities reserved for your Job executions.

If you want to use some JVM parameters for only a specific Job execution, for example if you want to display the execution result for this specific Job in Japanese, you need open this Job's **Run** view and then in the **Run** view, configure the advanced execution settings to define the corresponding parameters.

For further information about the advanced execution settings of a specific Job, see [Section 4.2.7.4, "How to set advanced execution settings"](#).

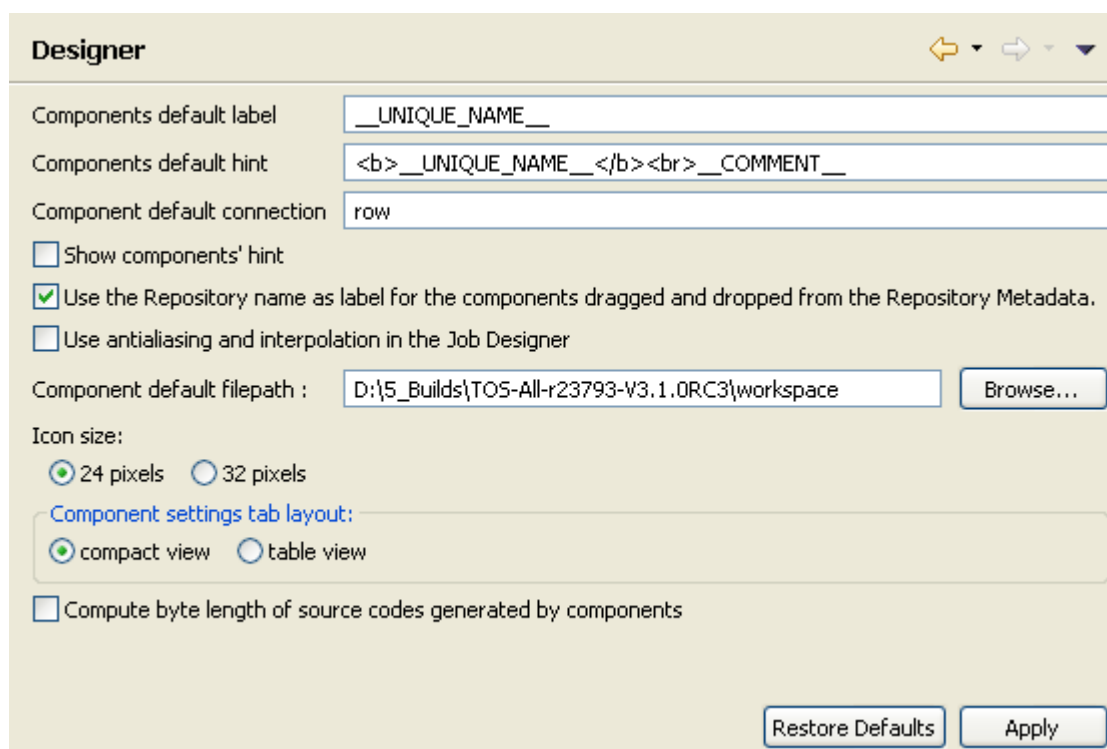
For more information about possible parameters, check the site <http://www.oracle.com/technetwork/java/javase/tech/vmoptions-jsp-140102.html>.

2.5.6. Designer preferences

You can set component and Job design preferences to let your settings be permanent in the Studio.

1. From the menu bar, click **Window > Preferences** to open the **[Preferences]** dialog box.
2. Expand the **Talend > Appearance** node.
3. Click **Designer** to display the corresponding view.

On this view, you can define the way component names and hints will be displayed.

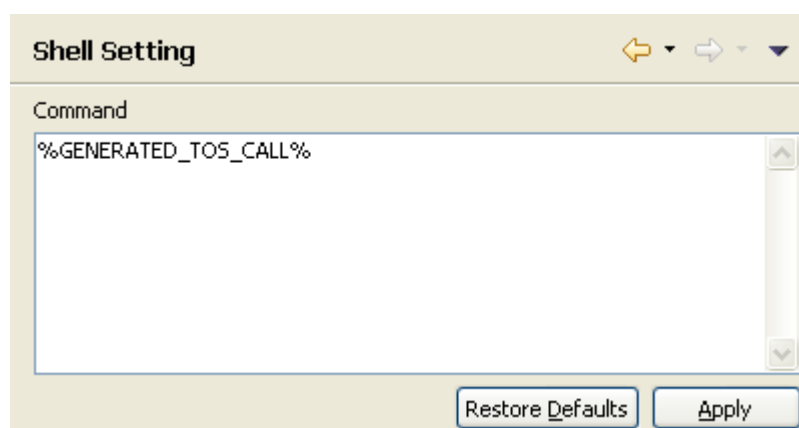


4. Select the relevant check boxes to customize your use of *Talend Open Studio for ESB* design workspace.

2.5.7. Adding code by default

You can add pieces of code by default at the beginning and at the end of the code of your Job.

1. From the menu bar, click **Window > Preferences** to open the [Preferences] dialog box.
2. Expand the **Talend** and **Import/Export** nodes in succession and then click **Shell Setting** to display the relevant view.



3. In the **Command** field, enter your piece/pieces of code before or after %GENERATED_TOS_CALL% to display it/them before or after the code of your Job.

2.5.8. Performance preferences

You can set the **Repository** tree view preferences according to your use of *Talend Open Studio for ESB*. To refresh the **Repository** view:

1. From the menu bar, click **Window > Preferences** to open the [Preferences] dialog box.
2. Expand the **Talend** node and click **Performance** to display the repository refresh preference.



You can improve your performance when you deactivate automatic refresh.

3. Set the performance preferences according to your use of *Talend Open Studio for ESB*:
 - Select the **Deactivate auto detect/update after a modification in the repository** check box to deactivate the automatic detection and update of the repository.
 - Select the **Check the property fields when generating code** check box to activate the audit of the property fields of the component. When one property field is not correctly filled in, the component is surrounded by red on the design workspace.



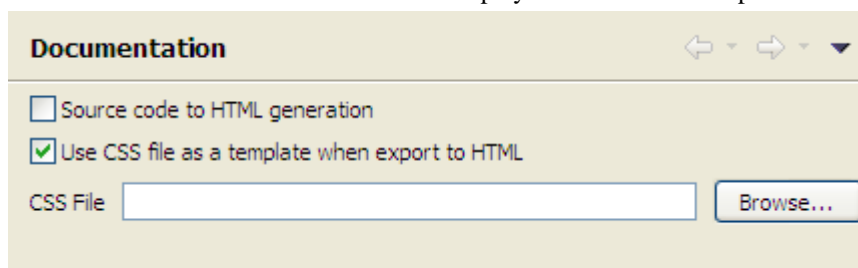
You can optimize performance if you disable property fields verification of components, i.e. if you clear the **Check the property fields when generating code** check box.

- Select the **Generate code when opening the job** check box to generate code when you open a Job.
- Select the **Check only the last version when updating jobs or joblets** check box to only check the latest version when you update a Job.
- Select the **Propagate add/delete variable changes in repository contexts** to propagate variable changes in the Repository Contexts.
- Select the **Activate the timeout for database connection** check box to establish database connection time out. Then set this time out in the **Connection timeout (seconds)** field.
- Select the **Add all user routines to job dependencies, when create new job** check box to add all user routines to Job dependencies upon the creation of new Jobs.
- Select the **Add all system routines to job dependencies, when create job** check box to add all system routines to Job dependencies upon the creation of new Jobs.

2.5.9. Documentation preferences

You can include the source code on the generated documentation.

1. From the menu bar, click **Window > Preferences** to open the [Preferences] dialog box.
2. Expand the **Talend** node and click **Documentation** to display the documentation preferences.



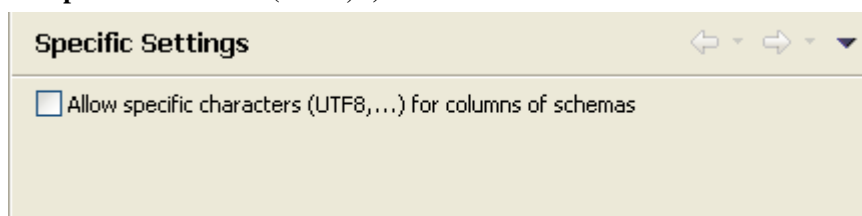
3. Customize the documentation preferences according to your needs:
 - Select the **Source code to HTML generation** check box to include the source code in the HTML documentation that you will generate.
 - Select the **Use CSS file as a template when export to HTML** check box to activate the **CSS File** field if you need to use a CSS file to customize the exported HTML files.

For more information on documentation, see [Section 7.6.1, “How to generate HTML documentation”](#) and [Section 4.2.6.5, “Documentation tab”](#).

2.5.10. Displaying special characters for schema columns

You may need to retrieve a table schema that contains columns written with special characters like Chinese, Japanese, Korean. In this case, you need to enable *Talend Open Studio for ESB* to read the special characters. To do so:

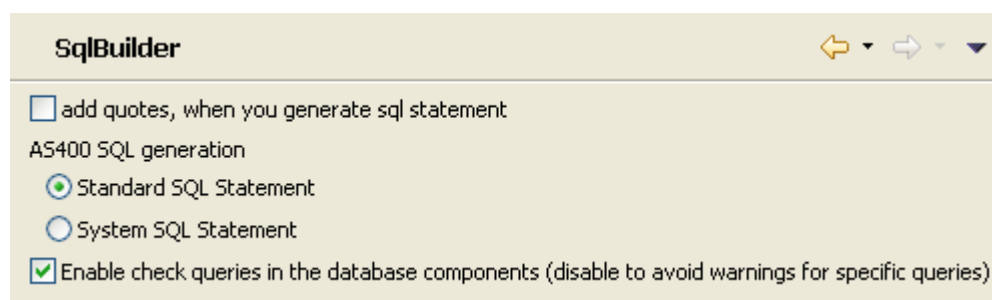
1. From the menu bar, click **Window > Preferences** to open the [Preferences] dialog box.
2. On the tree view of the opened dialog box, expand the **Talend** node.
3. Click the **Specific settings** node to display the corresponding view on the right of the dialog box.
4. Select the **Allow specific characters (UTF8,...) for columns of schemas** check box.



2.5.11. SQL Builder preferences

You can set your preferences for the SQL Builder. To do so:

1. From the menu bar, click **Window > Preferences** to open the **[Preferences]** dialog box.
2. Expand the **Talend** and **Specific Settings** nodes in succession and then click **Sql Builder** to display the relevant view.

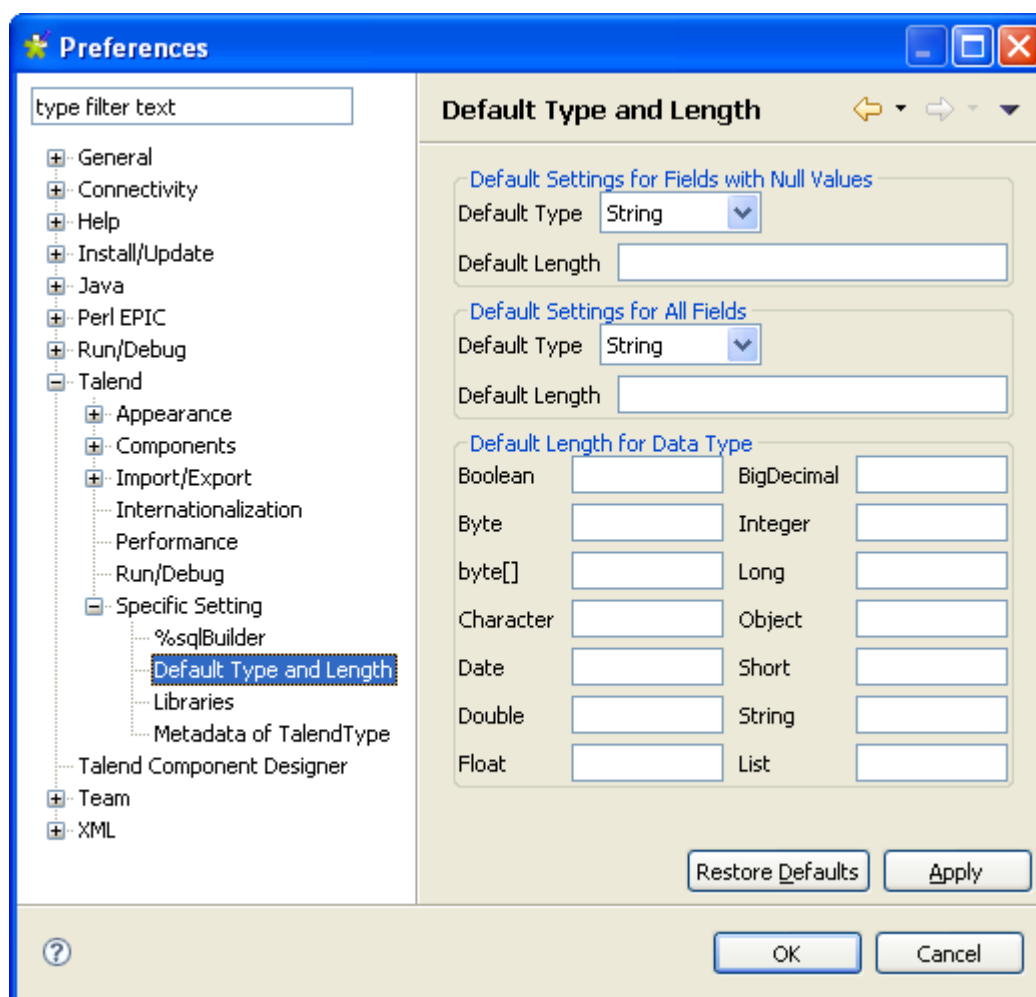


3. Customize the SQL Builder preferences according to your needs:
 - Select the **add quotes, when you generated sql statement** check box to precede and follow column and table names with inverted commas in your SQL queries.
 - In the **AS400 SQL generation** area, select the **Standard SQL Statement** or **System SQL Statement** check boxes to use standard or system SQL statements respectively when you use an AS400 database.
 - Clear the **Enable check queries in the database components (disable to avoid warnings for specific queries)** check box to deactivate the verification of queries in all database components.

2.5.12. Schema preferences

You can define the default data length and type of the schema fields of your components.

1. From the menu bar, click **Window > Preferences** to open the **[Preferences]** dialog box.
2. Expand the **Talend** node and click **Default Type and Length** to display the data length and type of your schema.



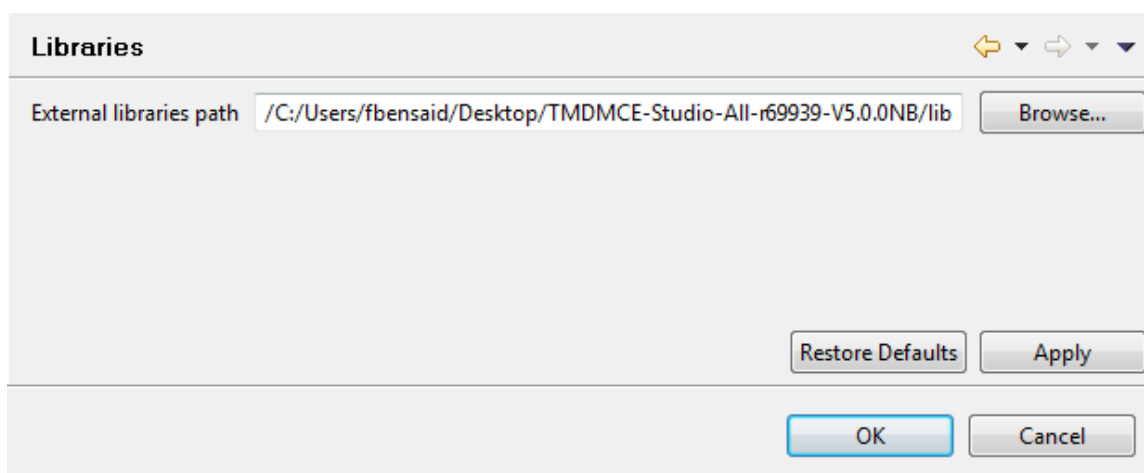
3. Set the parameters according to your needs:

- In the **Default Settings for Fields with Null Values** area, fill in the data type and the field length to apply to the null fields.
- In the **Default Settings for All Fields** area, fill in the data type and the field length to apply to all fields of the schema.
- In the **Default Length for Data Type** area, fill in the field length for each type of data.

2.5.13. Libraries preferences

You can define the folder where to store the different libraries used in *Talend Open Studio for ESB*. To do so:

1. From the menu bar, click **Window > Preferences** to display the **[Preferences]** dialog box.
2. Expand the **Talend** and **Specific Settings** nodes in succession and then click **Libraries** to display the relevant view.

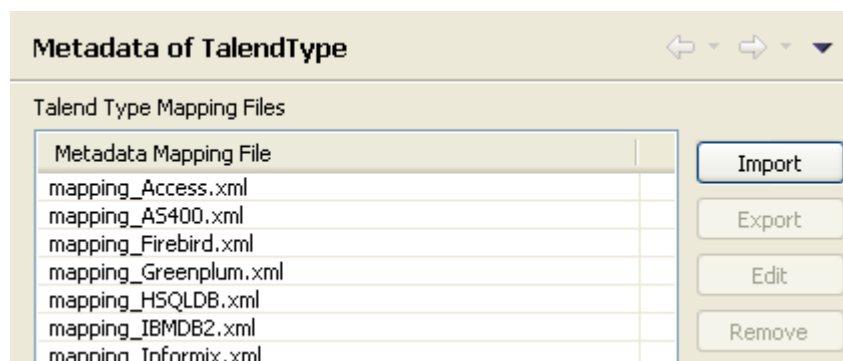


3. Set the access path in the **External libraries path** field through the **Browse...** button. The default path leads to the library of your current build.

2.5.14. Type conversion

You can set the parameters for type conversion in *Talend Open Studio for ESB*, from Java towards databases and vice versa.

1. From the menu bar, click **Window > Preferences** to display the **[Preferences]** dialog box.
2. Expand the **Talend** and **Specific Settings** nodes in succession and then click **Metadata of Talend Type** to display the relevant view.



The **Metadata Mapping File** area lists the XML files that hold the conversion parameters for each database type used in *Talend Open Studio for ESB*.

- You can import, export, or delete any of the conversion files by clicking **Import**, **Export** or **Remove** respectively.
- You can modify any of the conversion files according to your needs by clicking the **Edit** button to open the **[Edit mapping file]** dialog box and then modify the XML code directly in the open dialog box.

2.5.15. Usage Data Collector preferences

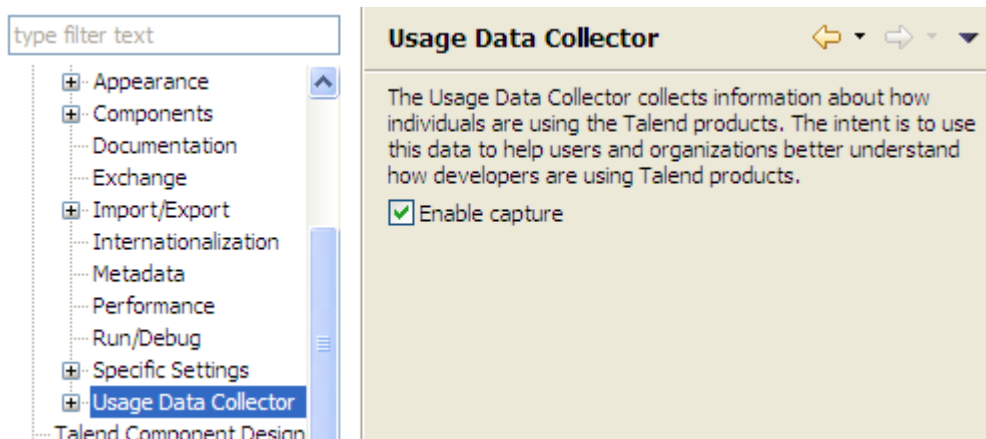
By allowing *Talend Open Studio for ESB* to collect your Studio usage statistics, you help users better understand **Talend** products and help **Talend** better learn how users are using the products, thus enabling **Talend** to improve product quality and performance to serve users better.

By default, *Talend Open Studio for ESB* automatically collects your Studio usage data and sends this data on a regular basis to servers hosted by **Talend**. You can view the usage data collection and upload information and customize the Usage Data Collector preferences according to your needs.

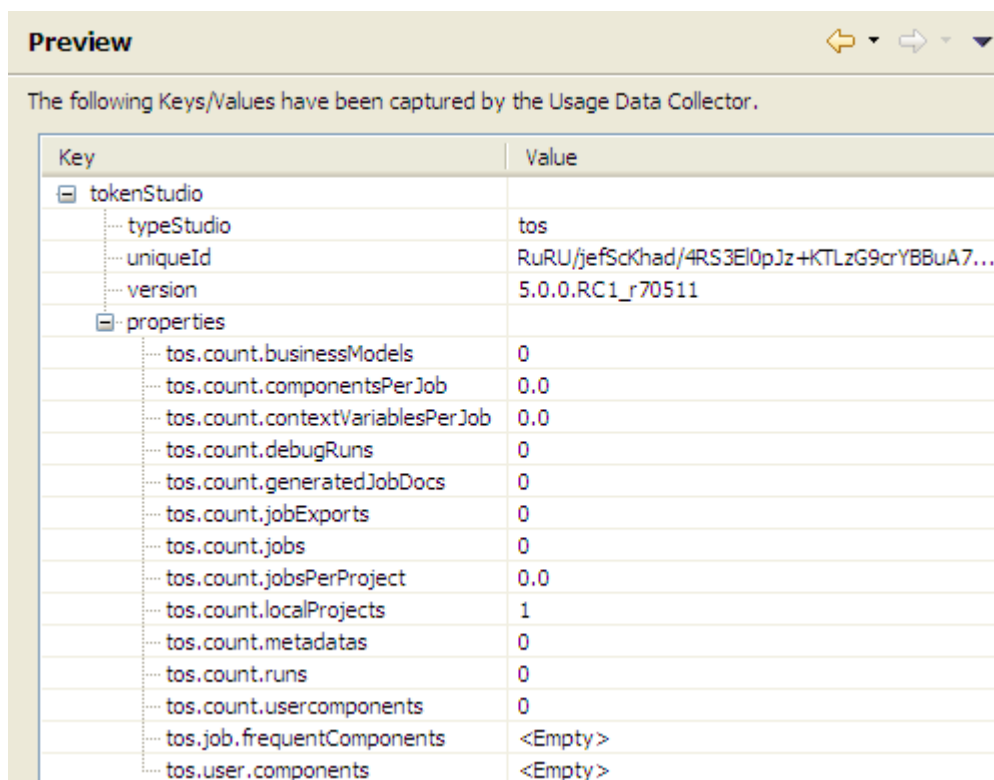


Be assured that only the Studio usage statistics data will be collected and none of your private information will be collected and transmitted to **Talend**.

1. From the menu bar, click **Window > Preferences** to display the **[Preferences]** dialog box.
2. Expand the **Talend** node and click **Usage Data Collector** to display the **Usage Data Collector** view.



3. Read the message about the Usage Data Collector, and, if you do not want the Usage Data Collector to collect and upload your Studio usage information, clear the **Enable capture** check box.
4. To have a preview of the usage data captured by the Usage Data Collector, expand the **Usage Data Collector** node and click **Preview**.



5. To customize the usage data upload interval and view the date of the last upload, click **Uploading** under the **Usage Data Collector** node.

- By default, if enabled, the Usage Data Collector collects the product usage data and sends it to **Talend** servers every 10 days. To change the data upload interval, enter a new integer value (in days) in the **Upload Period** field.
- The read-only **Last Upload** field displays the date and time the usage data was last sent to **Talend** servers.

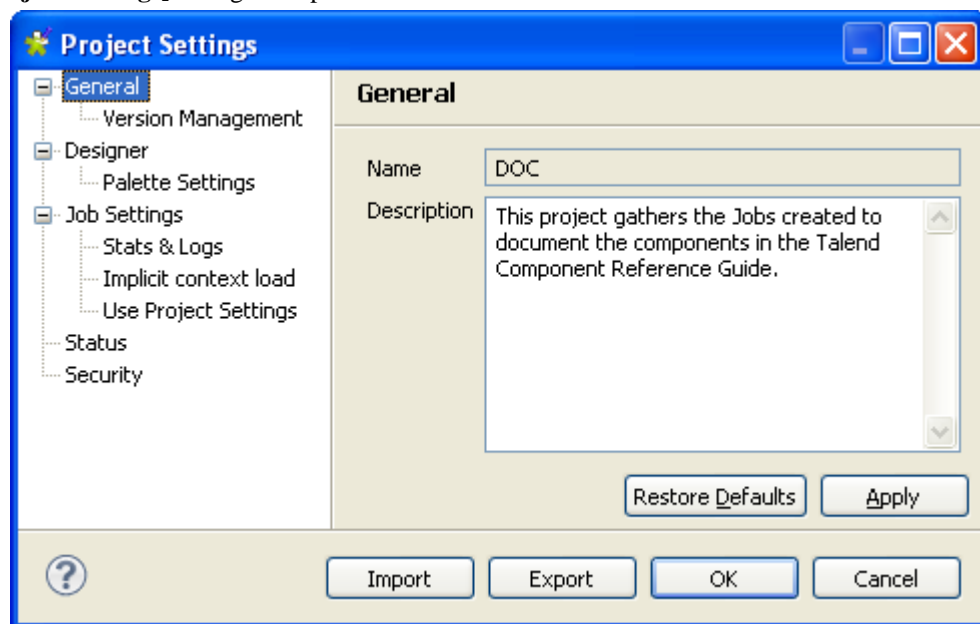
2.6. Customizing project settings

Talend Open Studio for ESB enables you to customize the information and settings of the project in progress, including the **Palette**, Job settings and Job version management, for example.

To customize project settings:

1. Click  on the Studio tool bar, or select **File > Edit Project Properties** from the menu bar.

The **[Project Settings]** dialog box opens.



2. In the tree diagram to the left of the dialog box, select the setting you wish to customize and then customize it, using the options that appear to the right of the box.

From the dialog box you can also export or import the full assemblage of settings that define a particular project:

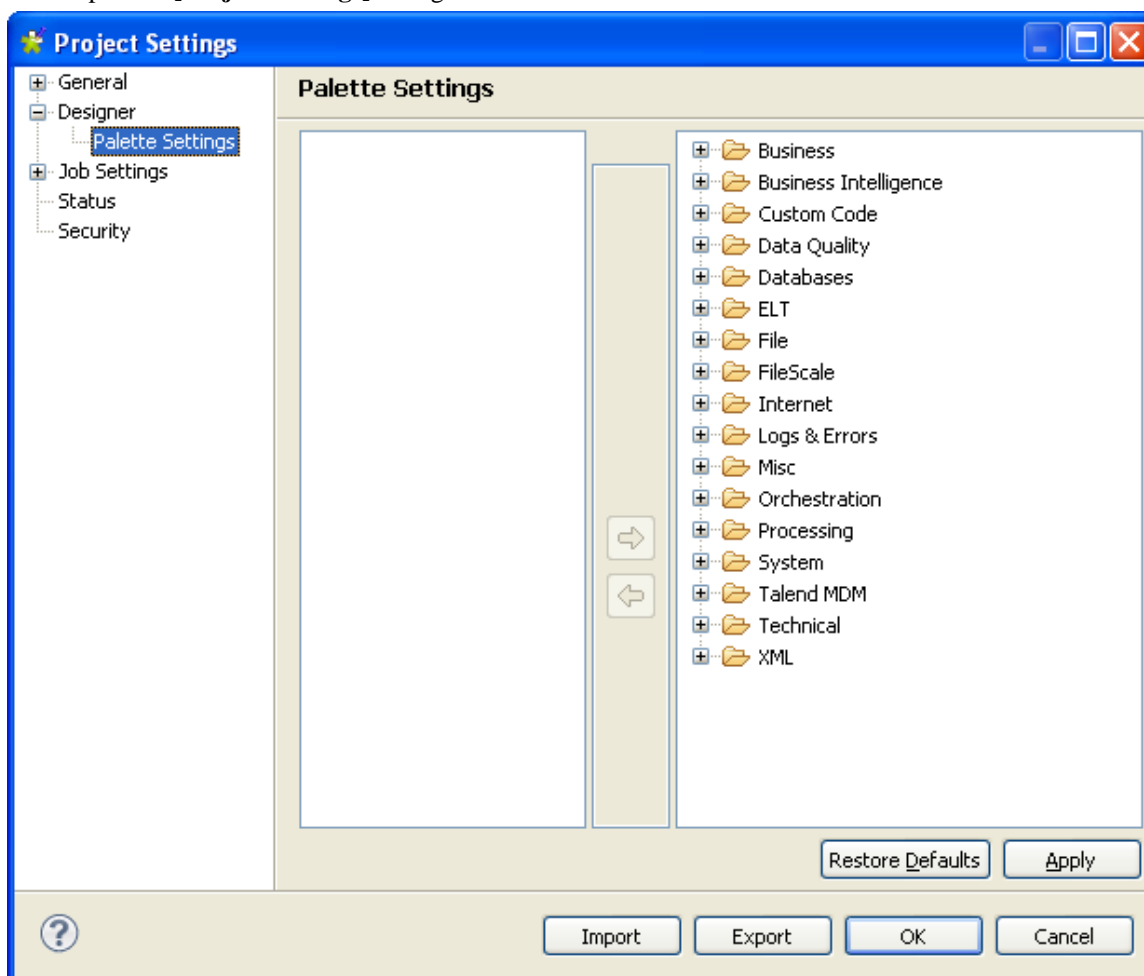
- To export the settings, click on the **Export** button. The export will generate an XML file containing all of your project settings.
- To import settings, click on the **Import** button and select the XML file containing the parameters of the project which you want to apply to the current project.

2.6.1. Palette Settings

You can customize the settings of the **Palette** display so that only the components used in the project are loaded. This will allow you to launch the Studio more quickly.

To customize the **Palette** display settings:

1. On the toolbar of the Studio's main window, click  or click **File > Edit Project Properties** on the menu bar to open the **[Project Settings]** dialog box.



In the **General** view of the **[Project Settings]** dialog box, you can add a project description, if you did not do so when creating the project.

2. In the tree view of the **[Project Settings]** dialog box, expand **Designer** and select **Palette Settings**. The settings of the current **Palette** are displayed in the panel to the right of the dialog box.
3. Select one or several components, or even set(s) of components you want to remove from the current project's **Palette**.
4. Use the left arrow button to move the selection onto the panel on the left. This will remove the selected components from the **Palette**.
5. To re-display hidden components, select them in the panel on the left and use the right arrow button to restore them to the **Palette**.
6. Click **Apply** to validate your changes and **OK** to close the dialog box.




To get back to the **Palette** default settings, click **Restore Defaults**.

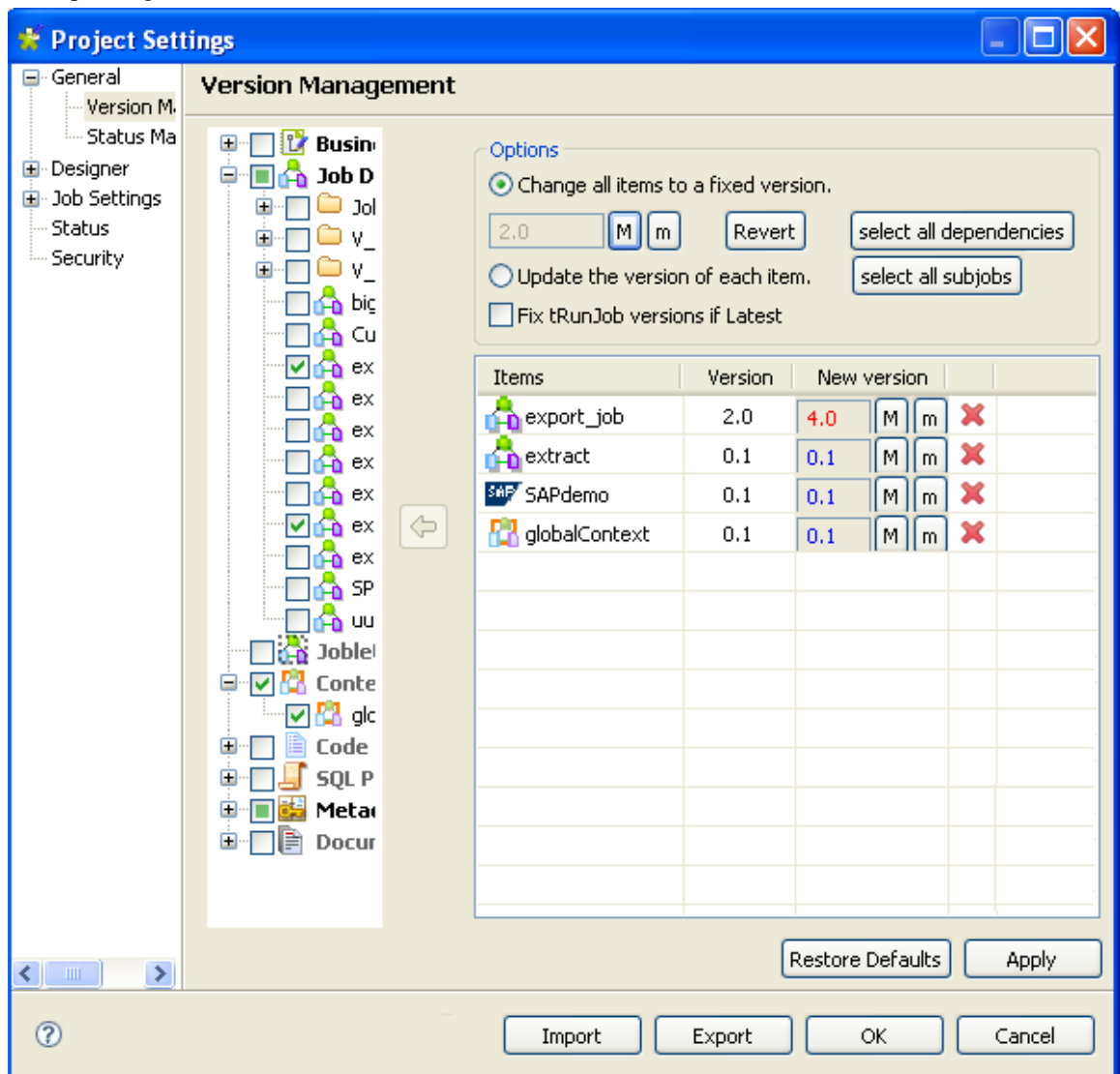
For more information on the **Palette**, see [Section 4.2.8.1, “How to change the Palette layout and settings”](#).

2.6.2. Version management

You can also manage the version of each item in the **Repository** tree view through **General > Version Management** of the **[Project Settings]** dialog box.

To do so:

1. On the toolbar of the Studio main window, click  or click **File** > **Edit Project Properties** from the menu bar to open the **[Project Settings]** dialog box.
2. In the tree view of the dialog box, expand **General** and select **Version Management** to open the corresponding view.



3. In the **Repository** tree view, expand the node holding the items you want to manage their versions and then select the check boxes of these items.

The selected items display in the **Items** list to the right along with their current version in the **Version** column and the new version set in the **New Version** column.

4. Make changes as required:

- In the **Options** area, select the **Change all items to a fixed version** check box to change the version of the selected items to the same fixed version.
- Click **Revert** if you want to undo the changes.
- Click **Select all dependencies** if you want to update all of the items dependent on the selected items at the same time.
- Click **Select all subjobs** if you want to update all of the subjobs dependent on the selected items at the same time.
- To increment each version of the items, select the **Update the version of each item** check box and change them manually.
- Select the **Fix tRunjob versions if Latest** check box, if you want the father job of current version to keep using the child Job(s) of current version in the **tRunjob** to be versioned, , regardless of how their versions will update. For example, a **tRunjob** will update from the current version *1.0* to *1.1* at both father and child levels. Once this check box is selected, the father Job *1.0* will continue to use the child Job *1.0* rather than the latest one as usual, say, version *1.1* when the update is done.



*To use this check box, the father Job must be using child Job(s) of the latest version as current version in the tRunjob to be versioned, by having selected the Latest option from the drop-down version list in the Component view of the child Job(s). For more information on **tRunJob**, see Talend Open Studio Components Reference Guide.*

5. Click **Apply** to apply your changes and then **OK** to close the dialog box.




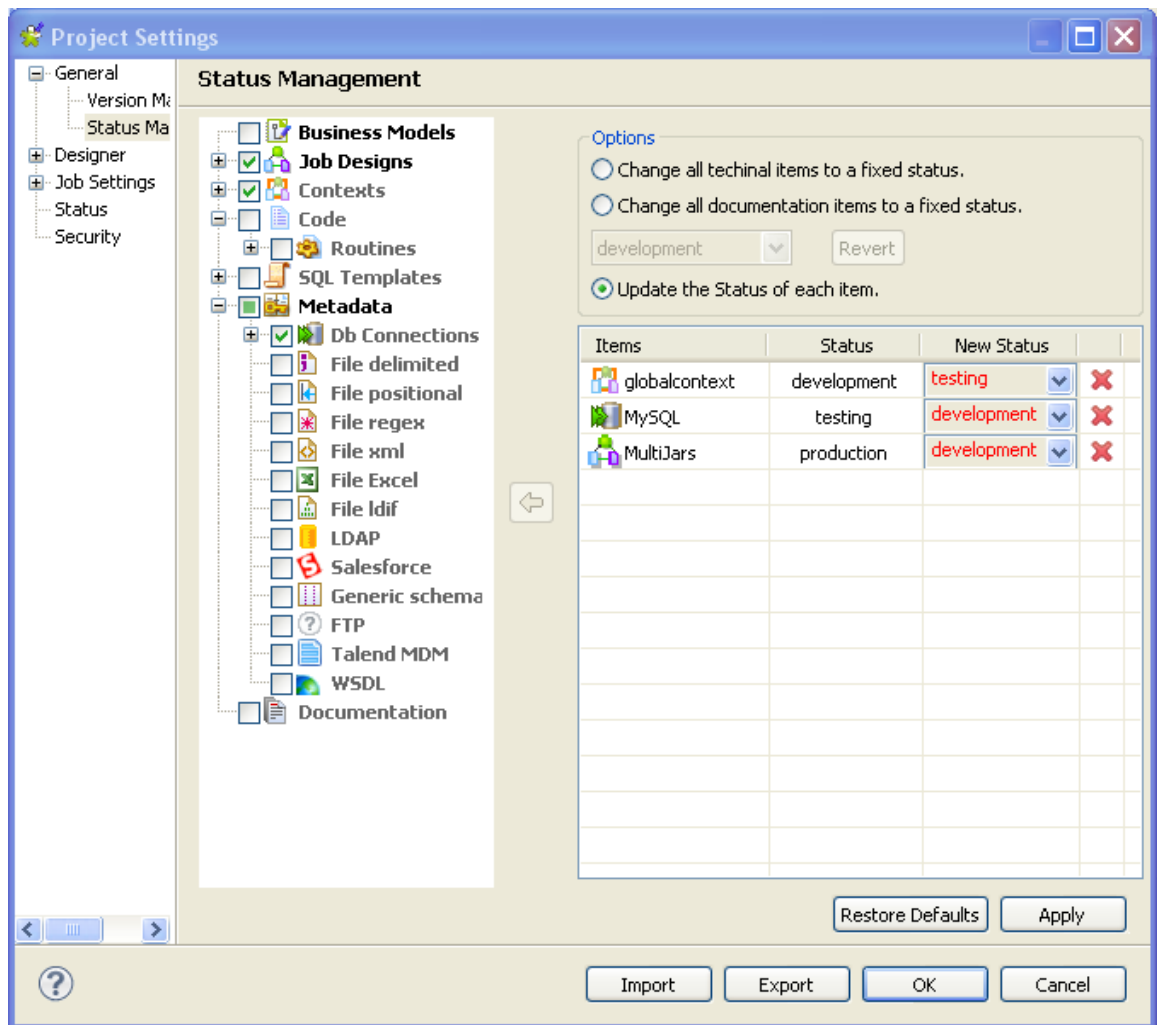
For more information on version management, see [Section 7.5, “Managing Job and Route versions”](#).

2.6.3. Status management

You can also manage the status of each item in the **Repository** tree view through **General > Status Management** of the **[Project Settings]** dialog box.

To do so:

1. On the toolbar of the Studio main window, click  or click **File > Edit Project Properties** from the menu bar to open the **[Project Settings]** dialog box.
2. In the tree view of the dialog box, expand **General** and select **Status Management** to open the corresponding view.



3. In the **Repository** tree view, expand the node holding the items you want to manage their status and then select the check boxes of these items.

The selected items display in the **Items** list to the right along with their current status in the **Status** column and the new status set in the **New Status** column.

4. In the **Options** area, select the **Change all technical items to a fixed status** check box to change the status of the selected items to the same fixed status.
5. Click **Revert** if you want to undo the changes.
6. To increment each status of the items, select the **Update the version of each item** check box and change them manually.
7. Click **Apply** to apply your changes and then **OK** to close the dialog box.




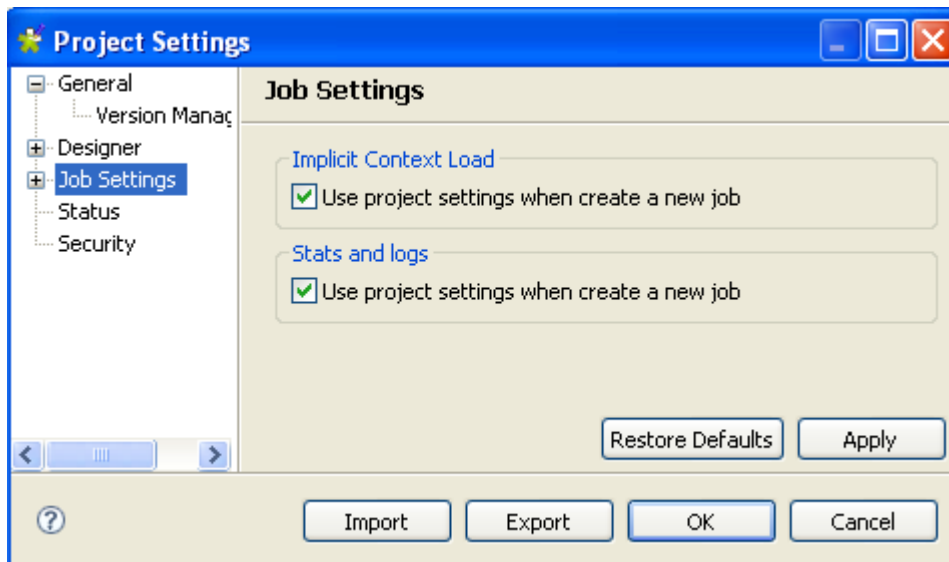
For further information about Job status, see [Section 2.6.8, “Status settings”](#).

2.6.4. Job Settings

You can automatically use **Implicit Context Load** and **Stats and Logs** settings you defined in the **[Project Settings]** dialog box of the actual project when you create a new Job.

To do so:

1. On the toolbar of the Studio main window, click  or click **File > Edit Project Properties** from the menu bar to open the **[Project Settings]** dialog box.
2. In the tree view of the dialog box, click the **Job Settings** node to open the corresponding view.
3. Select the **Use project settings when create a new job** check boxes of the **Implicit Context Load** and **Stats and Logs** areas.




4. Click **Apply** to validate your changes and then **OK** to close the dialog box.

2.6.5. Stats & Logs

When you execute a Job, you can monitor the execution through the **tStatCatcher Statistics** option or through using a log component. This will enable you to store the collected log data in .csv files or in a database.

You can then set up the path to the log file and/or database once for good in the **[Project Settings]** dialog box so that the log data get always stored in this location.

To do so:

1. On the toolbar of the Studio main window, click  or click **File > Edit Project Properties** from the menu bar to open the **[Project Settings]** dialog box.
2. In the tree view of the dialog box, expand the **Job Settings** node and then click **Stats & Logs** to display the corresponding view.



If you know that the preferences for Stats & Logs will not change depending upon the context of execution, then simply set permanent preferences. If you want to apply the Stats & Logs settings individually, then it is better to set these parameters directly onto the Stats & Logs view. For more information about this view, see [Section 4.6.7.1, “How to automate the use of statistics & logs”](#).

3. Select the **Use Statistics**, **Use Logs** and **Use Volumetrics** check boxes where relevant, to select the type of log information you want to set the path for.
4. Select a format for the storage of the log data: select either the **On Files** or **On Database** check box. Or select the **On Console** check box to display the data in the console.

The relevant fields are enabled or disabled according to these settings. Fill out the **File Name** between quotes or the **DB name** where relevant according to the type of log information you selected.

You can now store the database connection information in the **Repository**. Set the **Property Type** to **Repository** and browse to retrieve the relevant connection metadata. The fields get automatically completed.




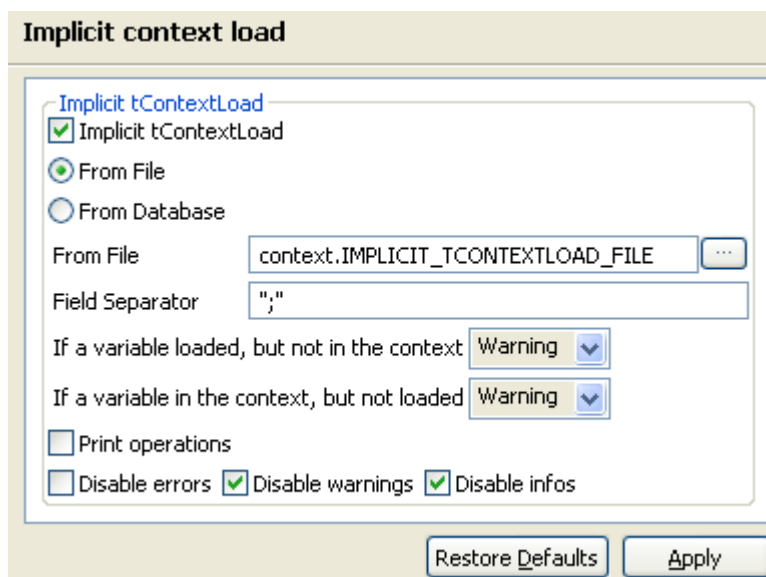
Alternatively, if you save your connection information in a Context, you can also access them through **Ctrl+Space**.

2.6.6. Context settings

You can define default context parameters you want to use in your Jobs.

To do so:

1. On the toolbar of the Studio main window, click  or click **File > Edit Project Properties** from the menu bar to open the **[Project Settings]** dialog box.
2. In the tree view of the dialog box, expand the **Job Settings** node and then select the **Implicit Context Load** check box to display the configuration parameters of the Implicit **tContextLoad** feature.




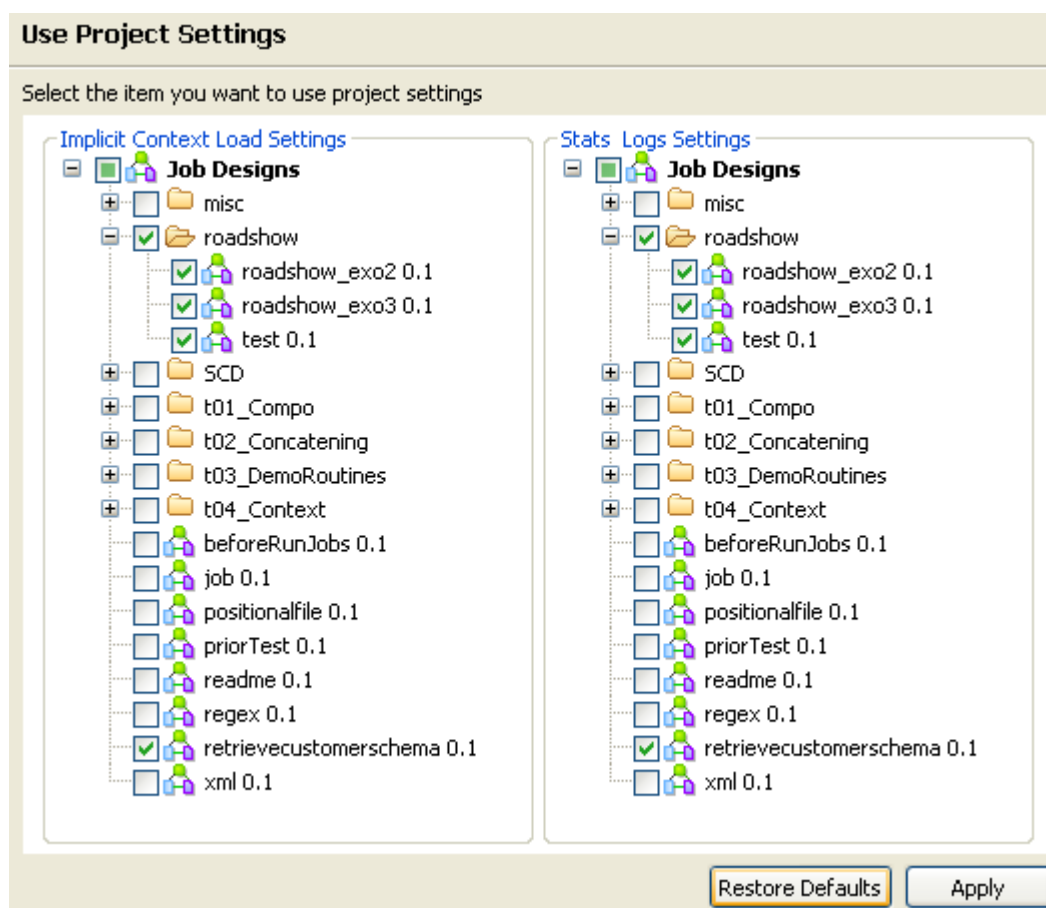
3. Select the **From File** or **From Database** check boxes according to the type of file you want to store your contexts in.
4. For files, fill in the file path in the **From File** field and the field separator in the **Field Separator** field.
5. For databases, select the **Built-in** or **Repository** mode in the **Property Type** list and fill in the next fields.
6. Fill in the **Table Name** and **Query Condition** fields.
7. Select the type of system message you want to have (warning, error, or info) in case a variable is loaded but is not in the context or vice versa.
8. Click **Apply** to validate your changes and then **OK** to close the dialog box.

2.6.7. Project Settings use

From the [Project Settings] dialog box, you can choose to which Job in the **Repository** tree view you want to apply the **Implicit Context Load** and **Stats and Logs** settings.

To do so:

1. On the toolbar of the Studio main window, click  or click **File > Edit Project Properties** from the menu bar to open the [Project Settings] dialog box.
2. In the tree view of the dialog box, expand the **Job Settings** node and then click **Use Project Settings** to display the use of **Implicit Context Load** and **Stats and Logs** option in the Jobs.




3. In the **Implicit Context Load Settings** area, select the check boxes corresponding to the Jobs in which you want to use the implicit context load option.
4. In the **Stats Logs Settings** area, select the check boxes corresponding to the Jobs in which you want to use the stats and logs option.
5. Click **Apply** to validate your changes and then **OK** to close the dialog box.

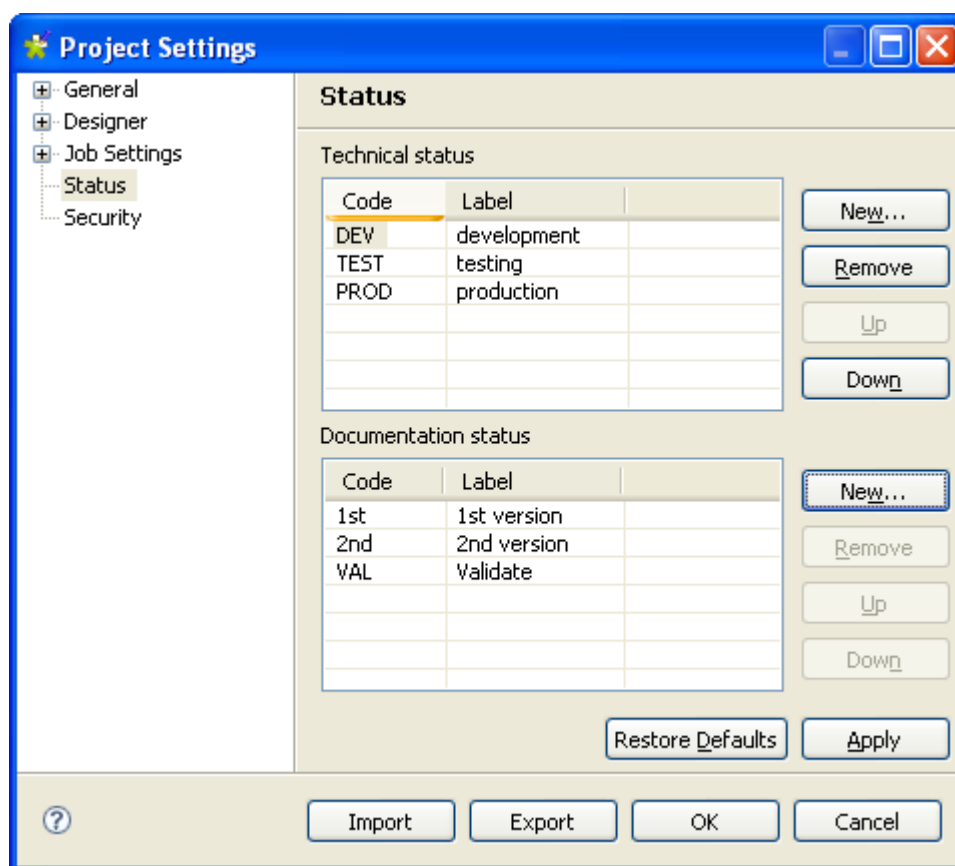
2.6.8. Status settings

In the [Project Settings] dialog box, you can also define the Status.

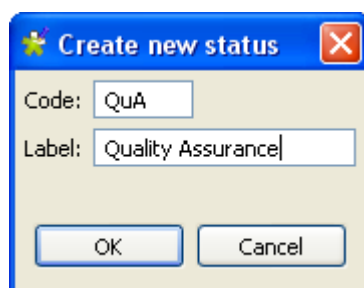
To do so:

1. On the toolbar of the Studio main window, click  or click **File > Edit Project Properties** from the menu bar to open the [Project Settings] dialog box.
2. In the tree view of the dialog box, click the **Status** node to define the main properties of your **Repository** tree view elements.

The main properties of a repository item gathers information data such as **Name**, **Purpose**, **Description**, **Author**, **Version** and **Status** of the selected item. Most properties are free text fields, but the **Status** field is a drop-down list.



- Click the **New...** button to display a dialog box and populate the **Status** list with the most relevant values, according to your needs. Note that the **Code** cannot be more than 3-character long and the **Label** is required.



Talend makes a difference between two status types: **Technical status** and **Documentation status**.

The **Technical status** list displays classification codes for elements which are to be running on stations, such as Jobs, metadata or routines.

The **Documentation status** list helps classifying the elements of the repository which can be used to document processes (Business Models or documentation).

- Once you completed the status setting, click **OK** to save


The **Status** list will offer the status levels you defined here when defining the main properties of your Job designs and business models.

- In the **[Project Settings]** dialog box, click **Apply** to validate your changes and then **OK** to close the dialog box.

2.6.9. Security settings

You can hide or show your passwords on your documentations, metadata, contexts, and so on when they are stored in the **Repository** tree view.

To hide your password:

1. On the toolbar of the Studio main window, click  or click **File > Edit Project Properties** from the menu bar to open the **[Project Settings]** dialog box.
2. In the tree view of the dialog box, click the **Security** node to open the corresponding view.
3. Select the **Hide passwords** check box to hide your password.



If you select the **Hide passwords** check box, your password will be hidden for all your documentations, contexts, and so on, as well as for your component properties when you select **Repository** in the **Property Type** field of the component **Basic settings** view, i.e. the screen capture below. However, if you select **Built-in**, the password will not be hidden.

Property Type: **Repository** (dropdown) | DB (MYSQL):demoMysql (text field) [...]

☐ Use an existing connection

Host: "localhost" [lightbulb icon] | Port: "3306" [lightbulb icon]

Database: "test" [lightbulb icon]

Username: "root" [lightbulb icon] | Password: "*****" [lightbulb icon]

4. In the **[Project Settings]** dialog box, click **Apply** to validate your changes and then **OK** to close the dialog box.


2.7. Filtering entries listed in the Repository tree view

Talend Open Studio for ESB provides the possibility to choose what nodes, Jobs or items you want to list in the **Repository** tree view.

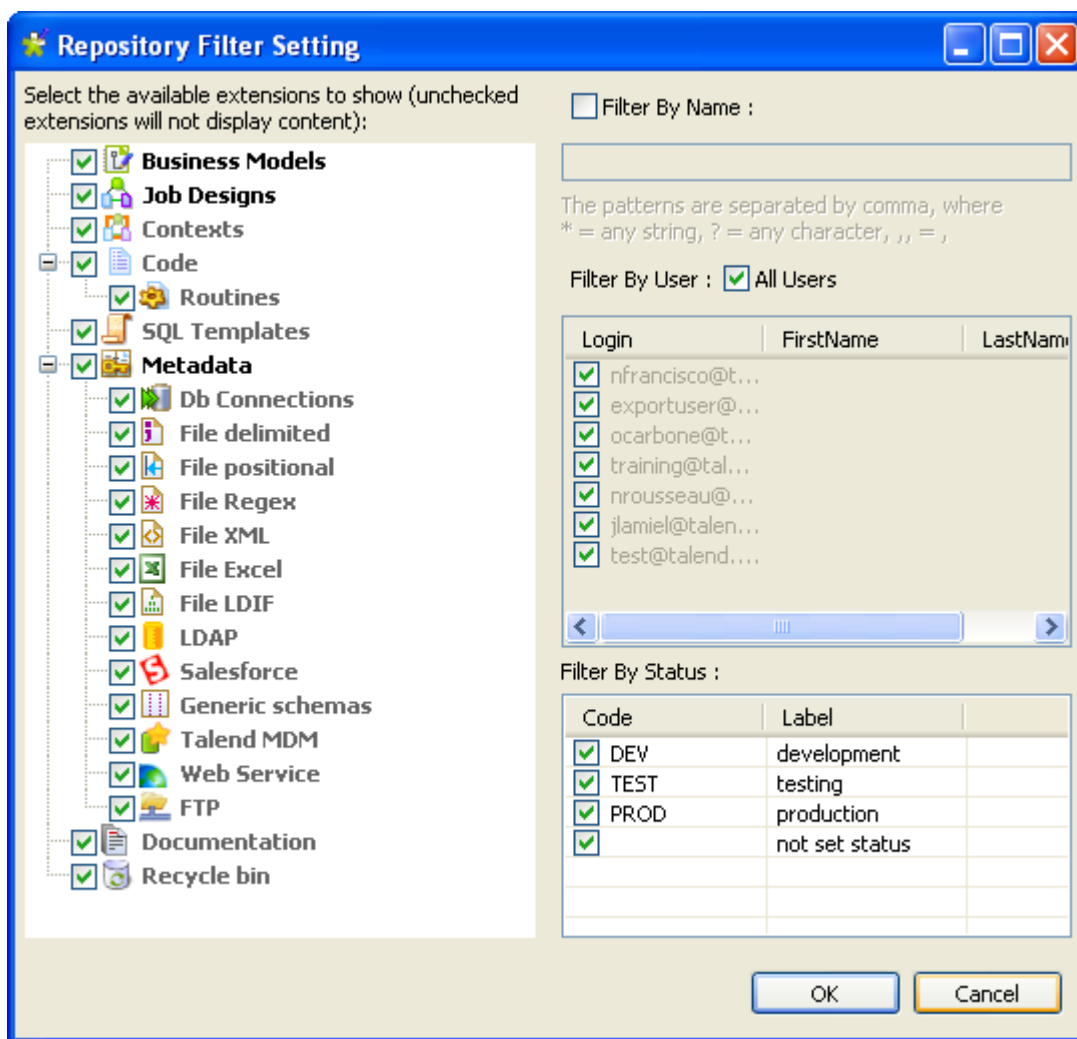
You can filter the **Repository** tree view by job name, Job status, the user who created the Job/items or simply by selecting/clearing the check box next to the node/ item you want to display/hide in the view. You can also set several filters simultaneously.

2.7.1. How to filter by Job name

To filter Jobs listed in the **Repository** tree view by Job name, complete the following:

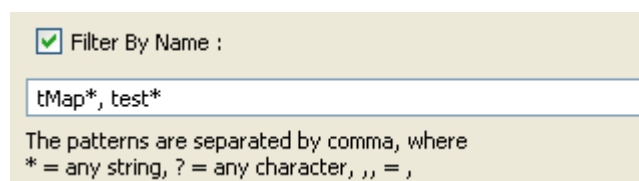
1. In the Studio, click the  icon in the upper right corner of the **Repository** tree view and select **Filter settings** from the contextual menu.

The **[Repository Filter]** dialog box displays.



2. Select the **Filter By Name** check box.

The corresponding field becomes available.

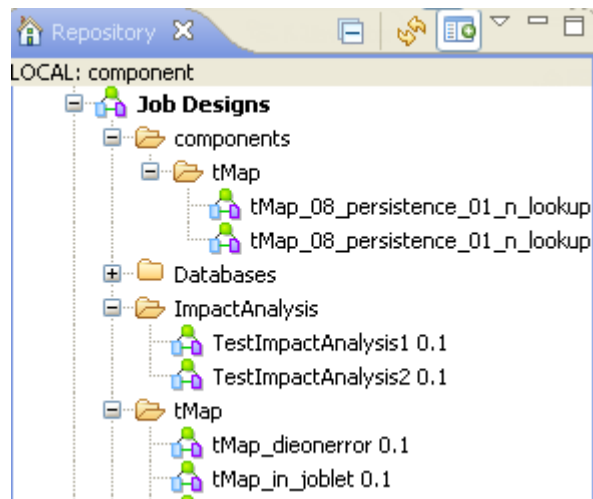




3. Follow the rules set below the field when writing the patterns you want to use to filter the Jobs.

In this example, we want to list in the tree view all Jobs that start with *tMap* or *test*.

4. In the **[Repository Filter]** dialog box, click **OK** to validate your changes and close the dialog box.


Only the Jobs that correspond to the filter you set are displayed in the tree view, those that start with *tMap* and *test* in this example



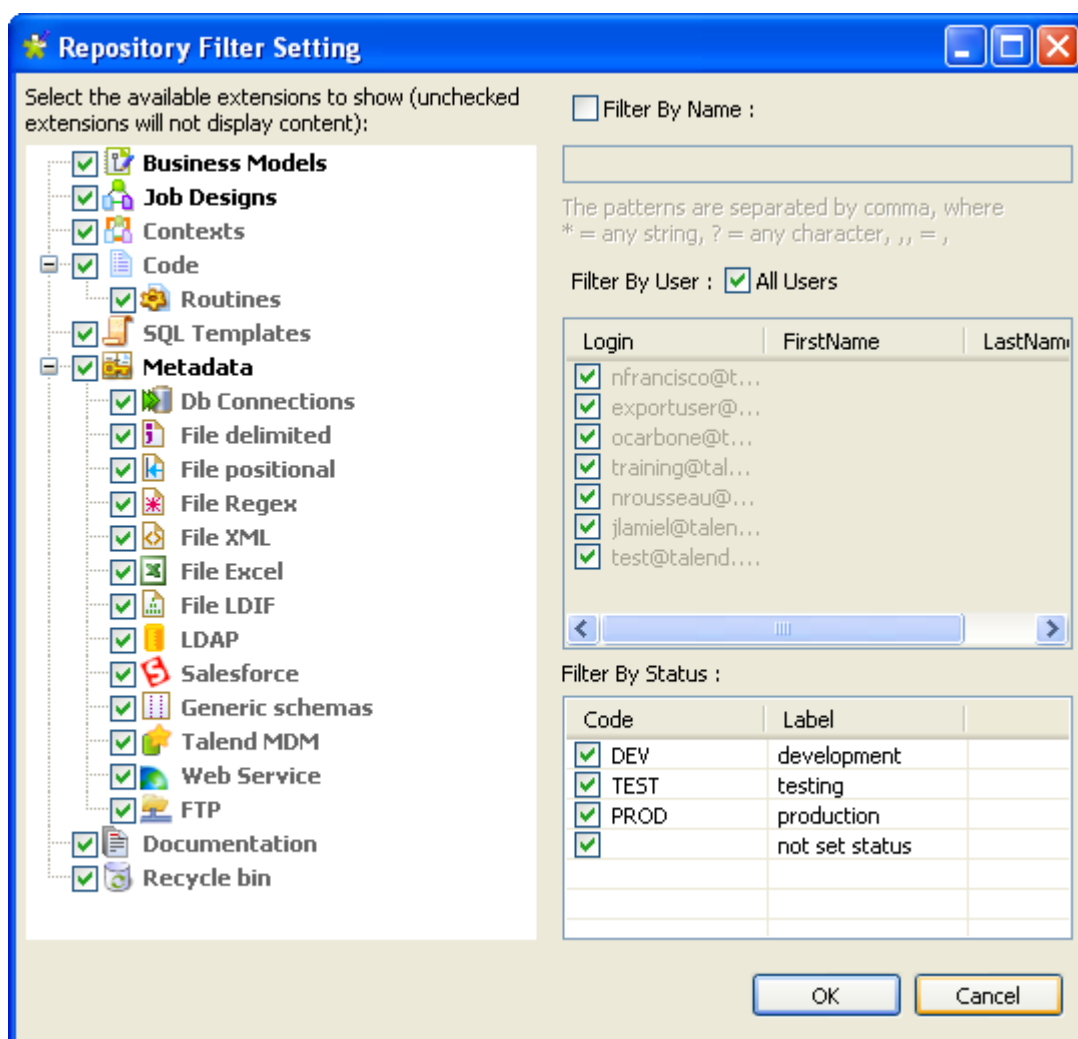
You can switch back to the by-default tree view, which lists all nodes, Jobs and items, by simply clicking the icon . This will cause the green plus sign appended on the icon to turn to a minus red sign (.

2.7.2. How to filter by user

To filter entries in the **Repository** tree view by the user who created the Jobs/items, complete the following:

1. In the Studio, click the  icon in the upper right corner of the **Repository** tree view and select **Filter settings** from the contextual menu.

The **[Repository Filter]** dialog box displays.



2. Clear the **All Users** check box.

The corresponding fields in the table that follows become available.

Filter By User : ☐ All Users

Login	FirstName	LastName
<input checked="" type="checkbox"/> admin@company.com	Administrator	Administrator
<input checked="" type="checkbox"/> nha@talend.com	talend	nha
<input checked="" type="checkbox"/> Copy_of_Copy_of...	talend	nha

This table lists the authentication information of all the users who have logged in to *Talend Open Studio for ESB* and created a Job or an item.

3. Clear the check box next to a user if you want to hide all the Jobs/items created by him/her in the **Repository** tree view.
4. Click **OK** to validate your changes and close the dialog box.


All Jobs/items created by the specified user will disappear from the tree view.



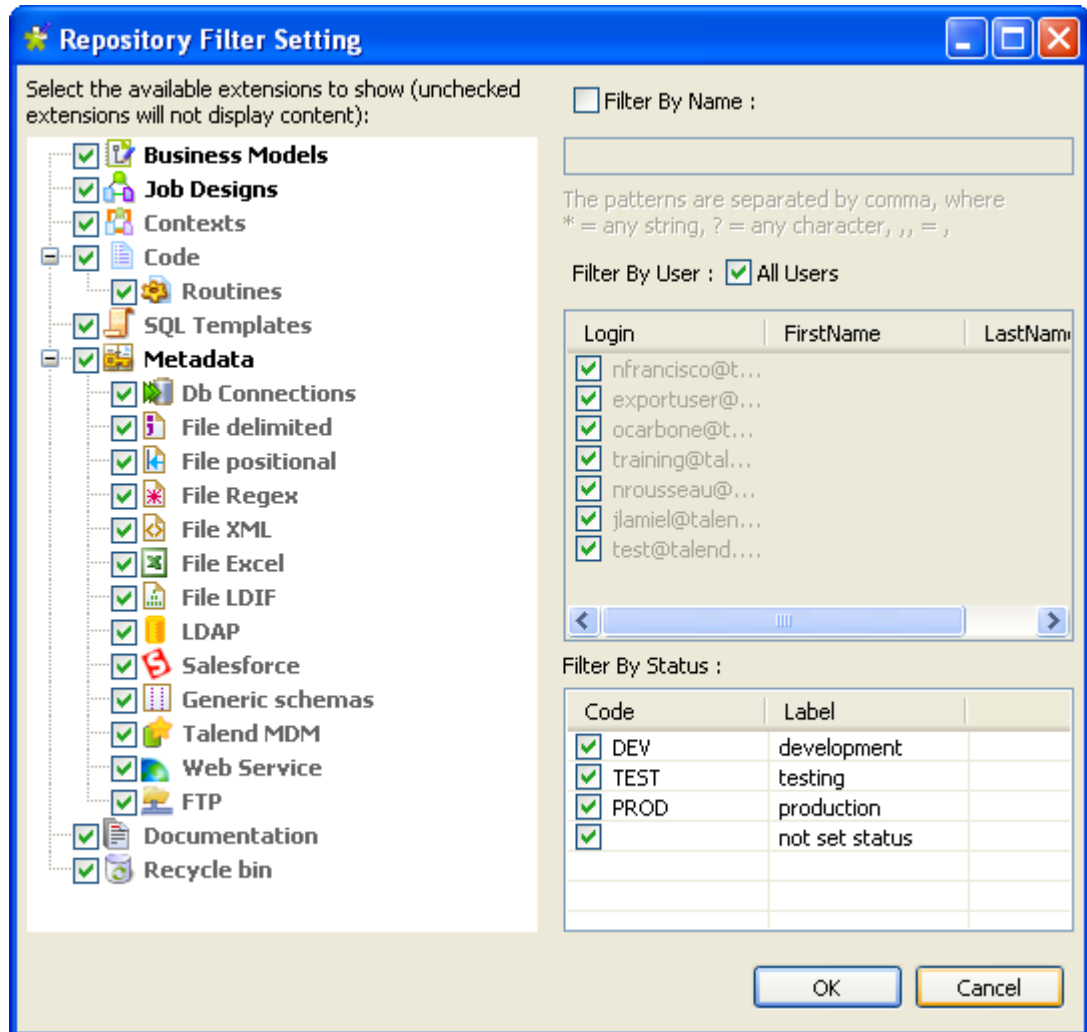
You can switch back to the by-default tree view, which lists all nodes, Jobs and items, by simply clicking the icon . This will cause the green plus sign appended on the icon to turn to a minus red sign ().

2.7.3. How to filter by job status

To filter Jobs in the **Repository** tree view by the job status, complete the following:

1. In the Studio, click the  icon in the upper right corner of the **Repository** tree view and select **Filter settings** from the contextual menu.



The **[Repository Filter]** dialog box displays.



2. In the **Filter By Status** area, clear the check boxes next to the status type if you want to hide all the Jobs that have the selected status.
3. Click **OK** to validate your changes and close the dialog box.

All Jobs that have the specified status will disappear from the tree view.




You can switch back to the by-default tree view, which lists all nodes, Jobs and items, by simply clicking the icon . This will cause the green plus sign appended on the icon to turn to a minus red sign (.

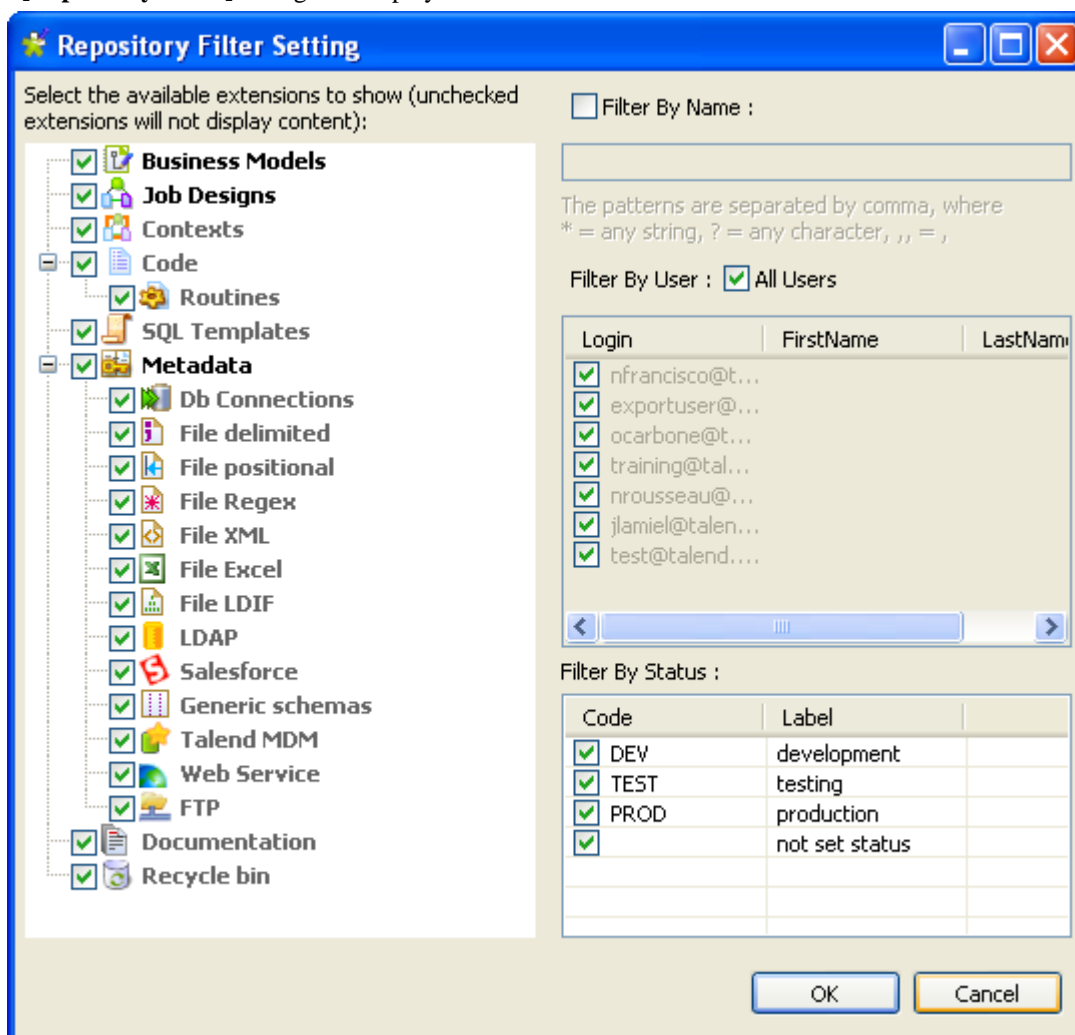
2.7.4. How to choose what repository nodes to display

To filter repository nodes, complete the following:

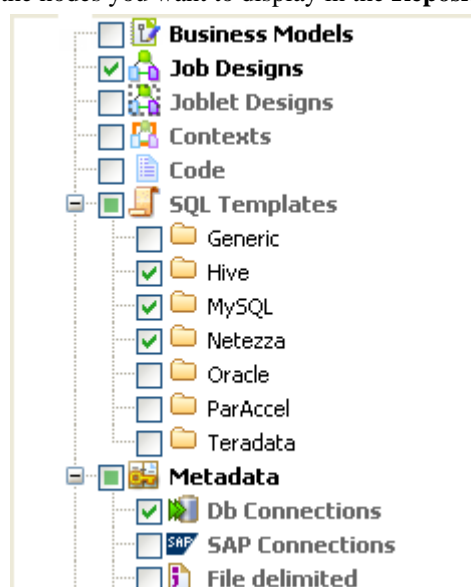
1.

In the Studio, click the  icon in the upper right corner of the **Repository** tree view and select **Filter settings** from the contextual menu.

The [**Repository Filter**] dialog box displays.



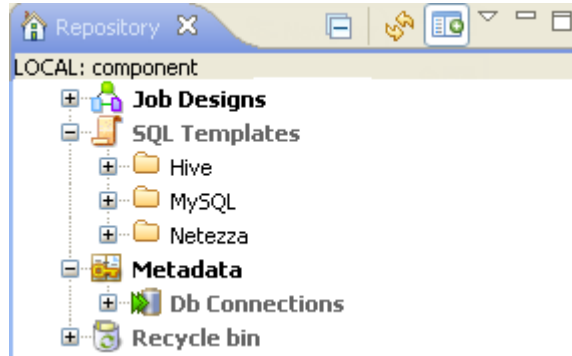
2. Select the check boxes next to the nodes you want to display in the **Repository** tree view.



Consider, for example, that you want to show in the tree view all the Jobs listed under the **Job Designs** node, three of the folders listed under the **SQL Templates** node and one of the metadata items listed under the **Metadata** node.

3. Click **OK** to validate your changes and close the dialog box.

Only the nodes/folders for which you selected the corresponding check boxes are displayed in the tree view.



If you do not want to show all the Jobs listed under the **Job Designs** node, you can filter the Jobs using the **Filter By Name** check box. For more information on filtering Jobs, see [Section 2.7.1, “How to filter by Job name”](#).



Chapter 3. Designing a Business Model

Talend Open Studio for ESB offers the best tool to formalize business descriptions into building blocks and their relationships. *Talend Open Studio for ESB* allows to design systems, connections, processes and requirements using standardized workflow notation through an intuitive graphical library of shapes and links.

This chapter aims at business managers, decision makers or developers who want to model their flow management needs at a macro level.

Before starting any business processes, you need to be familiar with *Talend Open Studio for ESB* Graphical User Interface (GUI). For more information, see [Appendix A, GUI](#).

3.1. What is a Business Model

Talend's Business Models allow data integration project stakeholders to graphically represent their needs regardless of the technical implementation requirements. Business Models help the IT operation staff understand these expressed needs and translate them into technical processes (Jobs). They typically include both the systems and processes already operating in the enterprise, as well as the ones that will be needed in the future.

Designing Business Models is part of the enterprises' best practices that organizations should adopt at a very early stage of a data integration project in order to ensure its success. Because Business Models usually help detect and resolve quickly project bottlenecks and weak points, they help limit the budget overspendings and/or reduce the upfront investment. Then during and after the project implementation, Business Models can be reviewed and corrected to reflect any required change.

A Business Model is a non technical view of a business workflow need.

Generally, a typical Business Model will include the strategic systems or processes already up and running in your company as well as new needs. You can symbolize these systems, processes and needs using multiple shapes and create the connections among them. Likely, all of them can be easily described using repository attributes and formatting tools.

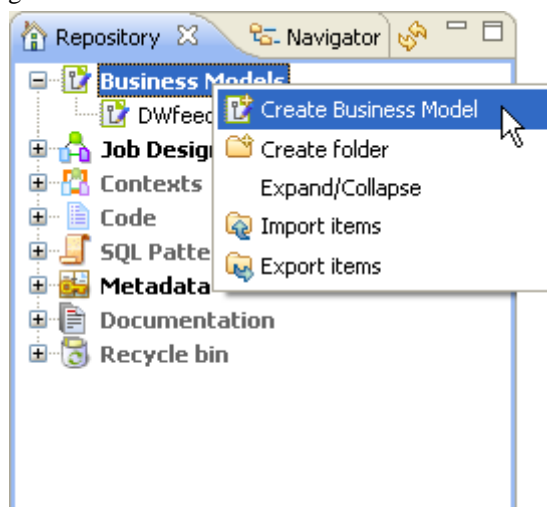
In the design workspace of *Talend Open Studio for ESB*, you can use multiple tools in order to:

- draw your business needs,
- create and assign numerous repository items to your model objects,
- define the business model properties of your model objects.

3.2. Opening or creating a Business Model

Open *Talend Open Studio for ESB* following the procedure as detailed in [Section 2.2, "Launching Talend Open Studio for ESB"](#).

In the **Repository** tree view, right-click the **Business Models** node.



Select **Expand/Collapse** to display all existing Business Models (if any).

3.2.1. How to open a Business Model

Double-click the name of the Business Model to be opened.

The selected Business Model opens up on the design workspace.

3.2.2. How to create a Business Model

Right-click the **Business Models** node and select **Create Business Model**.

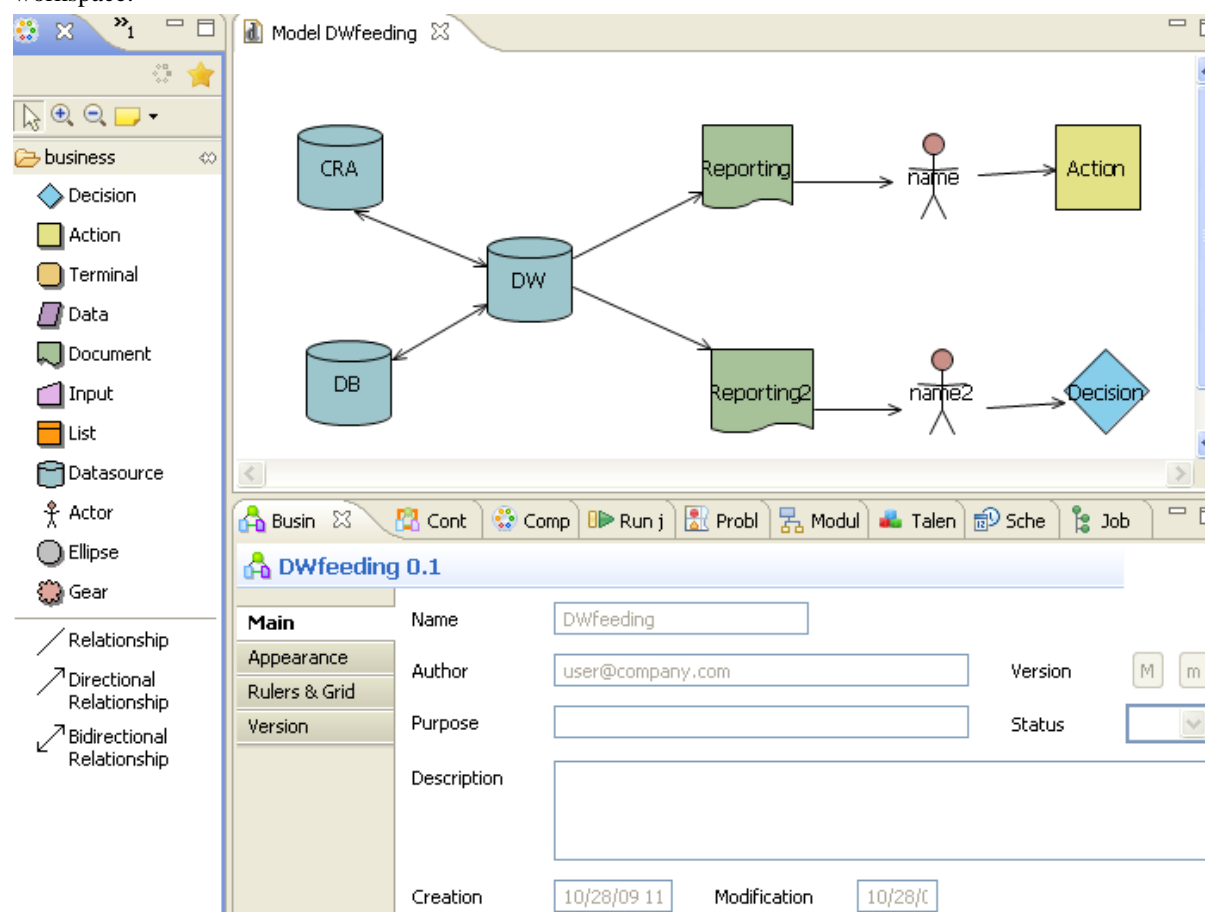
The creation wizard guides through the steps to create a new Business Model.

Select the **Location** folder where you want the new model to be stored.

And fill in a **Name** for it. The name you allocate to the file shows as a label on a tab at the top of the design workspace and the same name displays under the **Business Models** node in the **Repository** tree view.

The **Modeler** opens up on the empty design workspace.

You can create as many models as you want and open them all, they will display in a tab system on your design workspace.



The **Modeler** is made of the following panels:

- *Talend Open Studio for ESB's* design workspace
- a **Palette** of shapes and lines specific to the business modeling

- the **Business Model** panel showing specific information about all or part of the model.

3.3. Modeling a Business Model

If you have multiple tabs opened on your design workspace, click the relevant tab in order to show the appropriate model information.

In the **Business Model** view, you can see information relative to the active model.

Use the **Palette** to drop the relevant shapes on the design workspace and connect them together with branches and arrange or improve the model visual aspect by zooming in or out.



This **Palette** offers graphical representations for *objects* interacting within a Business Model.

The *objects* can be of different types, from strategic system to output document or decision step. Each one having a specific role in your Business Model according to the description, definition and assignment you give to it.

All objects are represented in the **Palette** as *shapes*, and can be included in the model.

Note that you must click the **business** folder to display the library of shapes on the **Palette**.

3.3.1. Shapes

Select the shape corresponding to the relevant *object* you want to include in your Business Model. Double-click it or click the shape in the **Palette** and drop it in the modeling area.

Alternatively, for a quick access to the shape library, keep your cursor still on the modeling area for a couple of seconds to display the quick access toolbar:



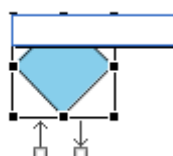
For instance, if your business process includes a decision step, select the diamond shape in the **Palette** to add this decision step to your model.



When you move the pointer over the quick access toolbar, a tooltip helps you to identify the shapes.

Then a simple click will do to make it show on the modeling area.

The shape is placed in a dotted black frame. Pull the corner dots to resize it as necessary.



Also, a blue-edged input box allows you to add a label to the shape. Give an expressive name in order to be able to identify at a glance the role of this shape in the model.

Two arrows below the added shape allow you to create connections with other shapes. You can hence quickly define sequence order or dependencies between shapes.

Related topic: [Section 3.3.2, “Connecting shapes”](#).

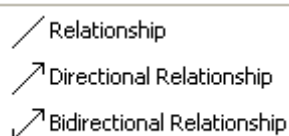
The available shapes include:

Callout	Details
Decision	The diamond shape generally represents an if condition in the model. Allows to take context-sensitive actions.
Action	The square shape can be used to symbolize actions of any nature, such as transformation, translation or formatting.
Terminal	The rounded corner square can illustrate any type of output terminal.
Data	A parallelogram shape symbolize data of any type.
Document	Inserts a Document object which can be any type of document and can be used as input or output for the data processed.
Input	Inserts an input object allowing the user to type in or manually provide data to be processed.
List	forms a list with the extracted data. The list can be defined to hold a certain nature of data.
Database	Inserts a database object which can hold the input or output data to be processed.
Actor	This schematic character symbolizes players in the decision-support as well technical processes.
Ellipse	Inserts an ellipse shape.
Gear	This gearing piece can be used to illustrate pieces of code programmed manually that should be replaced by a Talend Job for example.

3.3.2. Connecting shapes

When designing your Business Model, you want to implement relations between a source shape and a target shape.

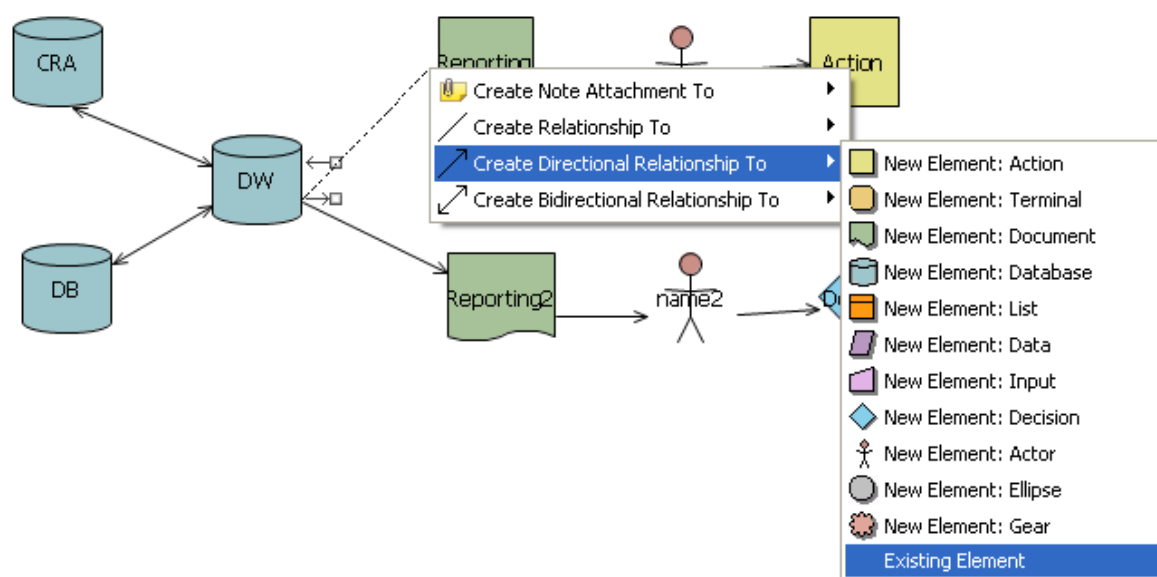
There are two possible ways to connect shapes in your design workspace:



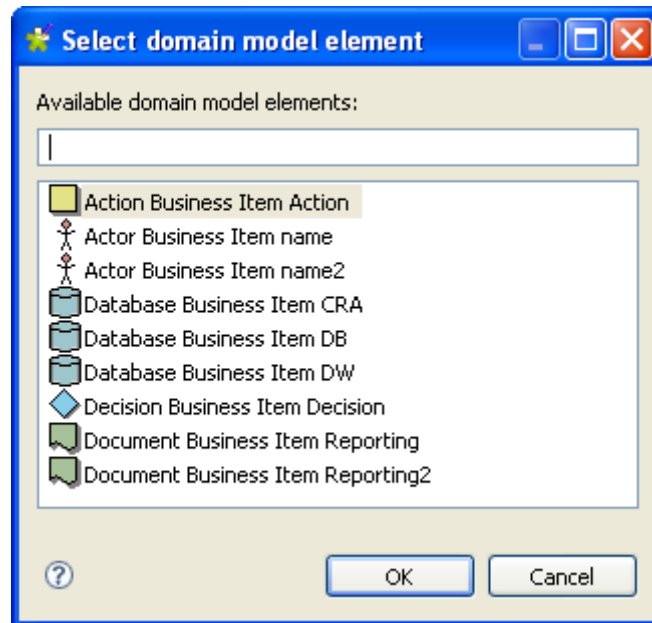
Either select the relevant **Relationship** tool in the **Palette**. Then, in the design workspace, pull a link from one shape to the other to draw a connection between them.

Or, you can implement both the relationship and the element to be related to or from, in a few clicks.

1. Simply move the mouse pointer over a shape that you already dropped on your design workspace, in order to display the double connection arrows.
2. Select the relevant arrow to implement the correct directional connection if need be.
3. Drag a link towards an empty area of the design workspace and release to display the connections popup menu.
4. Select the appropriate connection from the list. You can choose among **Create Relationship To**, **Create Directional Relationship To** or **Create Bidirectional Relationship To**.
5. Then, select the appropriate element to connect to, among the items listed.



You can create a connection to an existing element of the model. Select **Existing Element** in the popup menu and choose the existing element you want to connect to in the displaying list box.



The connection is automatically created with the selected shape.

The nature of this connection can be defined using **Repository** elements, and can be formatted and labelled in the **Properties** panel, see [Section 3.3.4, “Business Models”](#).

When creating a connection, an input box allows you to add a label to the connection you have created. Choose a meaningful name to help you identify the type of relationship you created.



You can also add notes and comments to your model to help you identify elements or connections at a later date.

Related topic: [Section 3.3.3, “How to comment and arrange a model”](#).

3.3.3. How to comment and arrange a model

The tools of the **Palette** allow you to customize your model:

Callout	Details
Select	Select and move the shapes and lines around in the design workspace’s modeling area.
Zoom	Zoom in to a part of the model. To watch more accurately part of the model. To zoom out, press <i>Shift</i> and click the modeling area.
Note/Text/Note attachment	Allows comments and notes to be added in order to store any useful information regarding the model or part of it.

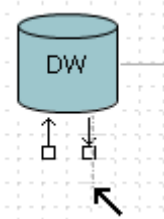
3.3.3.1. How to add a note or free text

To add a note, select the **Note** icon in the **Palette**, docked to the right of the design workspace.

Alternatively right-click the model or the shape you want to link the note to, and select *Add Note*. Or select the Note tool in the quick access toolbar.

A sticky note displays on the modeling area. If the note is linked to a particular shape, a line is automatically drawn to the shape.

Type in the text in the input box or, if the latter does not show, type in directly on the sticky note.

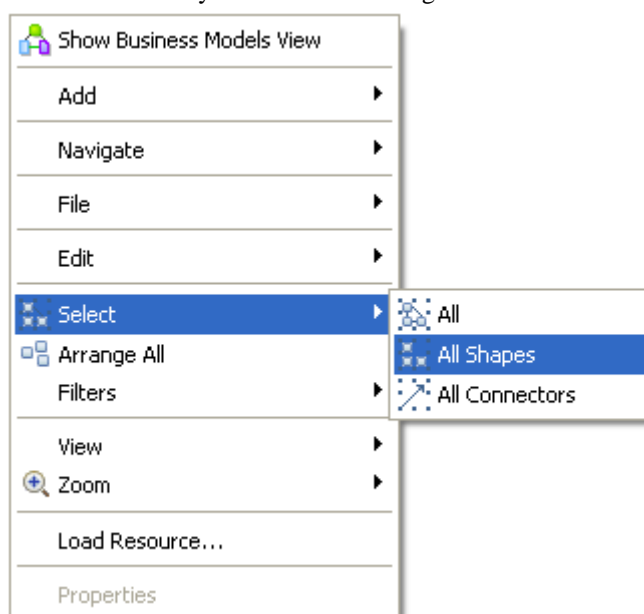


If you want to link your notes and specific shapes of your model, click the down arrow next to the **Note** tool on the **Palette** and select **Note attachment**. Pull the black arrow towards an empty area of the design workspace, and release. The popup menu offers you to attach a new Note to the selected shape.

You can also select the *Add Text* feature to type in free text directly in the modeling area. You can access this feature in the **Note** drop-down menu of the **Palette** or via a shortcut located next to the *Add Note* feature on the quick access toolbar.

3.3.3.2. How to arrange the model view

You can also rearrange the look and feel of your model via the right-click menu.



Place your cursor in the design area, right-click to display the menu and select *Arrange all*. The shapes automatically move around to give the best possible reading of the model.

Alternatively, you can select manually the whole model or part of it.

To do so, right-click any part of the modeling area, and click *Select*.

You can select:

- **All** shapes and connectors of the model,
- **All shapes** used in the design workspace,
- **All connectors** branching together the shapes.

From this menu you can also zoom in and out to part of the model and change the view of the model.

3.3.4. Business Models

The information in the **Business Models** view corresponds to the current selection, if any. This can be the whole model if you selected all shapes of it or more specifically one of the shapes it is made of. If nothing is selected, the **Business Models** tab gives general information about the model.

The **Business Models** view contains different types of information grouped in the **Main**, **Appearance**, **Rules & Grid**, and **Assignment** tabs.

The **Main** tab displays basic information about the selected item in the design workspace, being a Business Model or a Job. For more information about the **Main** tab, see [Section 4.2.8.3, “How to display Job configuration tabs/views”](#).

3.3.4.1. Appearance tab

From the **Appearance** tab you can apply filling or border colors, change the appearance of shapes and lines in order to customize your Business Model or make it easier to read.

The **Business Model** view includes the following formats:

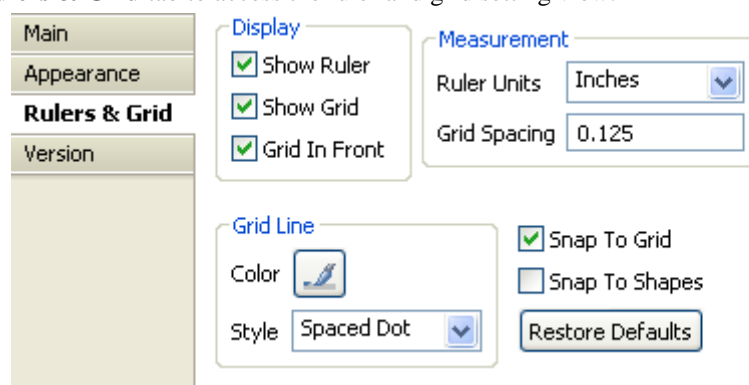
- fill the shape with selected color.
- color the shape border
- insert text above the shape
- insert gradient colors to the shape
- insert shadow to the shape

You can also move and manage shapes of your model using the edition tools. Right-click the relevant shape to access these editing tools.

3.3.4.2. Rulers and Grid tab

To display the **Rulers & Grid** tab, click  on the **Palette**, then click any empty area of the design workspace to deselect any current selection.

Click the **Rulers & Grid** tab to access the ruler and grid setting view.



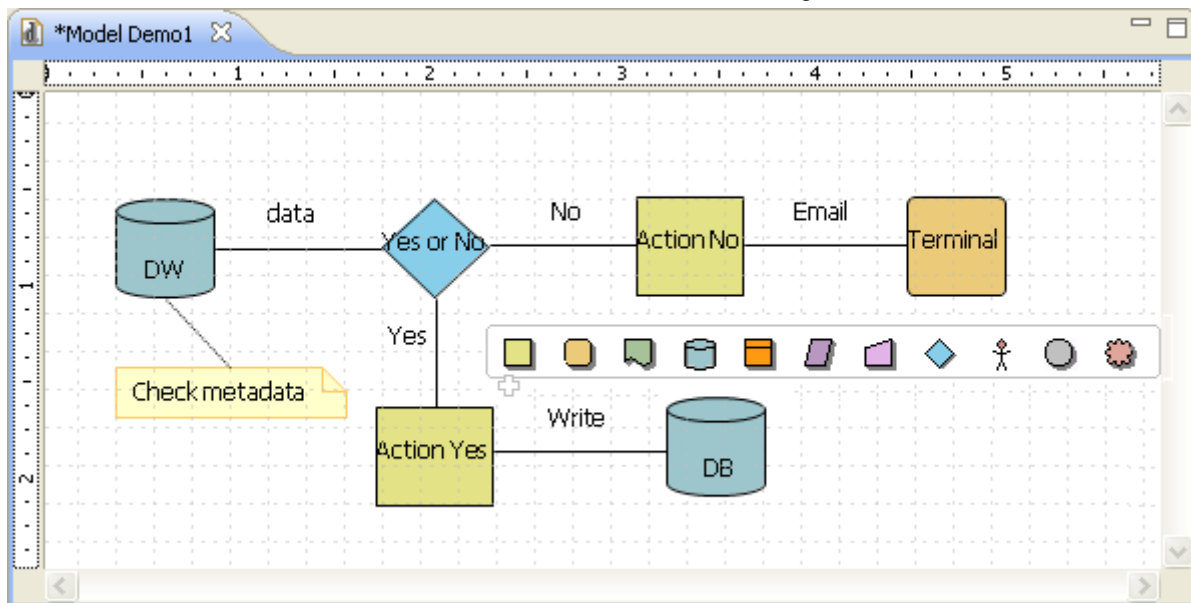
In the **Display** area, select the **Show Ruler** check box to show the **Ruler**, the **Show Grid** check box to show the **Grid**, or both check boxes. **Grid in front** sends the grid to the front of the model.

In the **Measurement** area, select the ruling unit among **Centimeters**, **Inches** or **Pixels**.

In the **Grid Line** area, click the **Color** button to set the color of the grid lines and select their style from the **Style** list.

Select the **Snap To Grid** check box to bring the shapes into line with the grid or the **Snap To Shapes** check box to bring the shapes into line with the shapes already dropped in the Business Model.

You can also click the **Restore Defaults** button to restore the default settings.



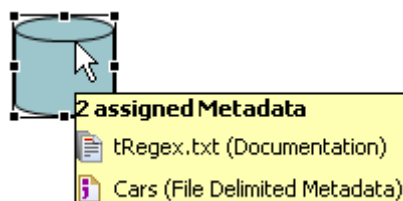
3.3.4.3. Assignment tab

The **Assignment** tab displays in a tabular form details of the **Repository** attributes you allocated to a shape or a connection.

To display any assignment information in the table, select a shape or a connection in the active model, then click the **Assignment** tab in the **Business Model** view.

Type	Name	Comment
Documentation	tRegex.txt	
File Delimited Metadata	Cars	

You can also display the assignment list placing the mouse over the shape you assigned information to.

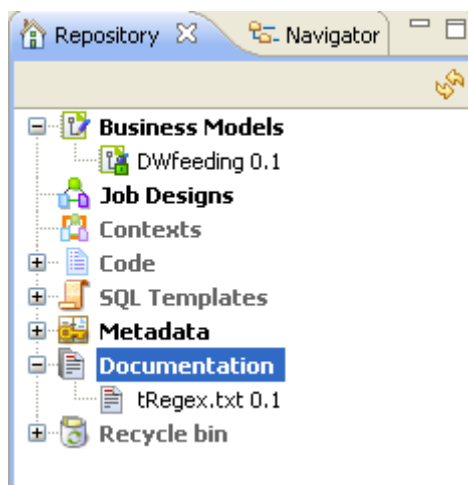


You can modify some information or attach a comment. Also, if you update data from the **Repository** tree view, assignment information gets automatically updated.

For further information about how to assign elements to a Business Model, see [Section 3.4, “Assigning repository elements to a Business Model”](#).

3.4. Assigning repository elements to a Business Model

The **Assignment** tab in the **Business Models** view lists the elements from the **Repository** tree view which have been assigned to a shape in the Business Model.



You can define or describe a particular object in your Business Model by simply associating it with various types of information, i.e. by adding metadata items.

You can set the nature of the metadata to be assigned or processed, thus facilitating the Job design phase.

To assign a metadata item, simply drop it from the **Repository** tree view to the relevant shape in the design workspace.

The **Assignment** table, located underneath the design workspace, gets automatically updated accordingly with the assigned information of the selected object.

The types of items that you can assign are:

Element	Details
Job designs	If any Job Designs developed for other projects in the same repository are available, you can reuse them as metadata in the active Business Model.
Metadata	You can assign any descriptive data stored in the repository to any of the objects used in the model. It can be connection information to a database for example.
Business Models	You can use in the active model all other Business Models stored in the repository of the same project.
Documentation	You can assign any type of documentation in any format. It can be a technical documentation, some guidelines in text format or a simple description of your databases.
Routines (Code)	If you have developed some routines in a previous project, to automate tasks for example, you can assign them to your Business Model. Routines are stored in the Code folder of the Repository tree view.

For more information about the **Repository** elements, see [Chapter 4, Designing a Job](#).

3.5. Editing a Business Model

Follow the relevant procedure according to your needs:

3.5.1. How to rename a Business Model

Right-click the relevant Business Model label on the **Repository** tree view then select **Edit properties** to display the corresponding **Main** properties information in the **[Edit properties]** dialog box.

Edit the model name in the **Name** field, then click **Finish** to close the dialog box. The model label changes automatically in the **Repository** tree view and will be reflected on the model tab of the design workspace, the next time you open the Business Model.



If the Business Model is open, the information in the **[Edit properties]** dialog box will be read-only so you will not be able to edit them.

3.5.2. How to copy and paste a Business Model

In **Repository > Business model**, right-click the Business Model name to be copied and select **Copy** in the popup menu, or press **Ctrl+C**.

Then right-click where you want to paste your Business Model, and select **Paste**.

3.5.3. How to move a Business Model

To move a Business Model from a location to another in your business models project folder, select a Business Model in the **Repository > Business Models** tree.

Then simply drop it to the new location.

3.5.4. How to delete a Business Model

Right-click the name of the model to be deleted and select **Delete** in the popup menu.

Alternatively, simply select the relevant Business Model, then drop it into the **Recycle bin** of the **Repository** tree view.

3.6. Saving a Business Model

To save a Business Model, click **File > Save** or press **Ctrl+S**. The model is saved under the name you gave during the creation process.

An asterisk displays in front of the Business Model name on the tab to indicate that changes have been made to the model but not yet saved.



To save a Business Model and increment its version at the same time:

1. click **File>Save as...**The **[Save as]** dialog box displays.
2. Next to the **Version** field, click the **M** button to increment the major version and the **m** button to increment the minor version.
3. Click **Finish** to validate the modification



By default, when you open a Business Model, you open its last version. Any previous version of the Business Model is read-only and thus cannot be modified.

You can access a list of the different versions of a Business Model and perform certain operations. To do that:

1. In the **Repository** tree view, select the Business Model you want to consult the versions of.
2. Click the **Business Models>Version** in succession to display the version list of the selected Job.
3. Right-click the Business Model version you want to consult.
4. Do one of the followings:

Select	To...
Edit properties	edit Business Model properties. Note: The Business Model should not be open on the design workspace, otherwise it will be in read-only mode.
Read Business Model	consult the Business Model in read-only mode.



You can open and modify the last version of a Business Model from the **Version** view if you select **Edit Business Model** from the drop-down list.



Chapter 4. Designing a Job

Talend Open Studio for ESB is the tool with the capabilities that treat all of the different sources and targets required in data integration and data service processes and all other associated operations.

Talend Open Studio for ESB helps you to design data integration and data service Jobs that allow you to put in place up and run dataflow management processes.

This chapter addresses the needs of programmers or IT managers who are ready to implement the technical aspects of a Business Model (regardless of whether it was designed in *Talend Open Studio for ESB*'s Business Modeler).

Before starting any data integration and data service processes, you need to be familiar with the *Talend Open Studio for ESB* Graphical User Interface (GUI). For more information, see [Appendix A, GUI](#).

4.1. What is a Job design



Data service Jobs will simply be referred to as Jobs in the following documentation.

A Job Design is the runnable layer of a business model. It is a graphical design, of one or more components connected together, that allows you to set up and run dataflow management processes. A Job Design translates business needs into code, routines and programs, in other words it technically implements your data flow.

The Jobs you design can address all of the different sources and targets that you need for data integration and data service processes and any other related process.

When you design a Job in *Talend Open Studio for ESB*, you can:

- put in place data integration and data service actions using a library of technical components.
- change the default setting of components or create new components or family of components to match your exact needs.
- set connections and relationships between components in order to define the sequence and the nature of actions.
- access code at any time to edit or document the components in the designed Job.
- create and add items to the repository for reuse and sharing purposes (in other projects or Jobs or with other users).



In order to be able to execute the Jobs you design in Talend Open Studio for ESB, you need to install an Oracle JVM 1.6 or later (IBM JVM is not supported). You can download it from <http://www.oracle.com/technetwork/java/javase/downloads/index.html>.

4.2. Getting started with a basic Job design



Until a Job is created, the design workspace is unavailable and the Palette does not display.

A Job design is made of one or several subjobs, which are themselves defined by one or, most likely, several components linked together. The properties of each component require to be configured individually, in order to function properly.

For more information, see [Section 4.2.4, “How to connect components together”](#) and [Section 4.2.6, “How to define component properties”](#).

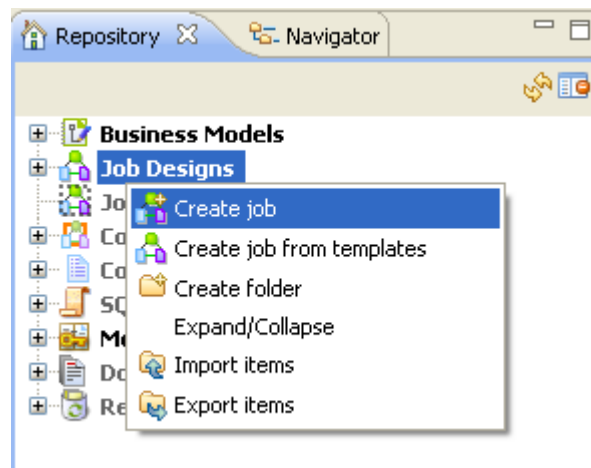
4.2.1. How to create a Job

Talend Open Studio for ESB enables you to create a Job Design by dropping different technical components from the **Palette** onto the design workspace and then connecting these components together.

You can also create different folders to better classify these Jobs.

To create a Job, complete the following:

1. Open *Talend Open Studio for ESB* following the procedure detailed in [Section 2.2, “Launching Talend Open Studio for ESB”](#).
2. In the **Repository** tree view, right-click the **Job Designs** node and select **Create job** from the contextual menu.



The [New job] wizard opens to help you define the main properties of the new Job.

3. Enter the Job properties according to the following table:

Field	Description
Name	the name of the new Job. A message comes up if you enter prohibited characters.
Purpose	Job purpose or any useful information regarding the Job use.
Description	Job description.
Author	a read-only field that shows by default the current user login.
Locker	a read-only field that shows by default the login of the user who owns the lock on the current Job. This field is empty when you are creating a Job and has data only when you are editing the properties of an existing Job.

Field	Description
Version	a read-only field. You can manually increment the version using the M and m buttons. For more information, see Section 7.5, “Managing Job and Route versions” .
Status	a list to select from the status of the Job you are creating.
Path	a list to select from the folder in which the Job will be created.

An empty design workspace opens up showing the name of the Job as a tab label.

- Drop the components you want to use in your Job design from the **Palette** onto the design workspace and connect them together. For more information, see [Section 4.2.2, “How to drop components to the workspace”](#) and [Section 4.2.4, “How to connect components together”](#).
- Define the properties of each of the components used in the Job. For more information, see [Section 4.2.6, “How to define component properties”](#).
- Save your Job and then press **F6** to execute it. For more information, see [Section 4.2.7, “How to run a Job”](#).

The Job you created is now listed under the **Job Designs** node in the **Repository** tree view.

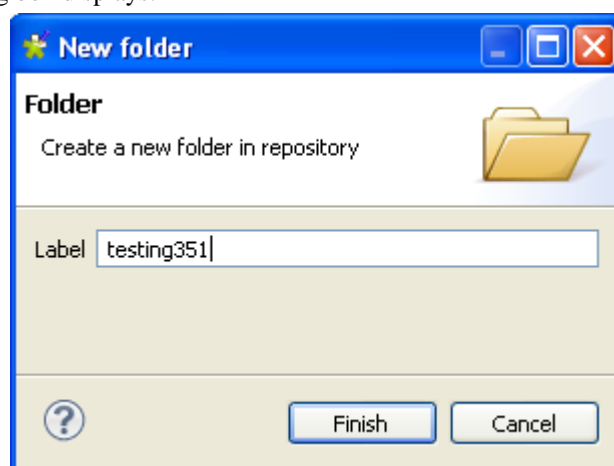


You can open one or more of the created Jobs by simply double-clicking the Job label in the **Repository** tree view.

To create different folders for your Jobs, complete the following:

- In the **Repository** tree view, right-click **Job Designs** and select **Create folder** from the contextual menu.

The **[New folder]** dialog box displays.



- In the **Label** field, enter a name for the folder and then click **Finish** to confirm your changes and close the dialog box.

The created folder is listed under the **Job Designs** node in the **Repository** tree view.



If you have already created Jobs that you want to move into this new folder, simply drop them into the folder.

For a scenario showing how to create a real-life data service Job, see [Appendix B, Theory into practice: Data service and routing examples](#).

4.2.2. How to drop components to the workspace

4.2.2.1. How to drop components from the Palette

To actually start building a Job, click a component on the **Palette**. Then click again on the design workspace to drop it there and add it to your Job Design.

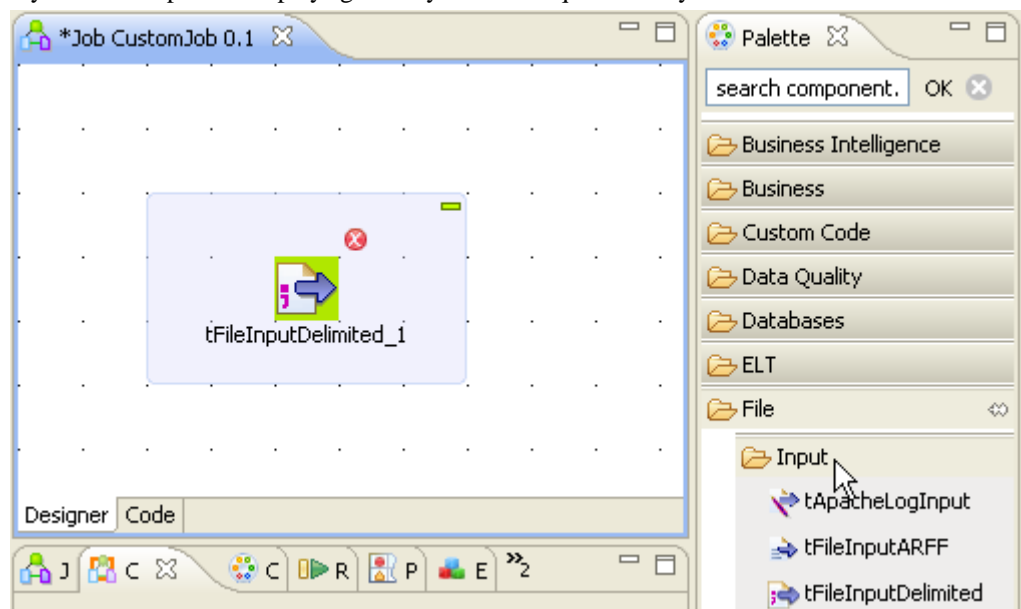


If the **Palette** does not show in the Studio, see [the section called “How to show, hide the Palette and change its position”](#).



You can drop a note to your Job the same way you drop components. For more information, see [Section 4.6.4, “How to add notes to a Job design”](#).

Each newly added component displays generally in a blue square that symbolizes it as an individual Subjob.



Connect components together in a logical order using the connections offered, in order to build a full Job or subjob. For more information about component connection types, see [Section 4.3.1, “Connection types”](#).

The Job or subjob gets highlighted in one single blue rectangle. For more information about Job and subjob background color, see [Section 4.6.6, “How to manage the subjob display”](#).

Multiple information or warnings may show next to the component. Browse over the component icon to display the information tooltip. This will display until you fully completed your Job design and defined all basic (and sometimes advanced) component properties of the **Component** view.



You will be required to use Java code for your project.

Related topics:

- [Section 4.2.4, “How to connect components together”](#).
- [Section 4.6.3.1, “Warnings and error icons on components”](#).
- [Section 4.2.6, “How to define component properties”](#).

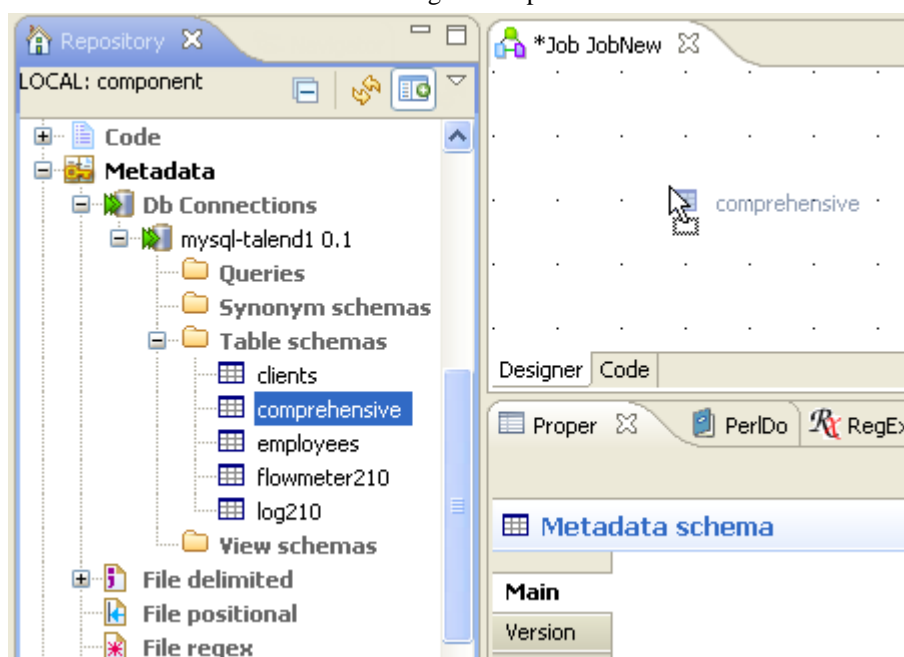
4.2.2.2. How to drop components from the Metadata node

For recurrent use of files and DB connections in various Jobs, we recommend you to store the connection and schema metadata in the **Repository** tree view under the **Metadata** node. Different folders under the **Metadata** node will group the established connections including those to databases, files and systems.

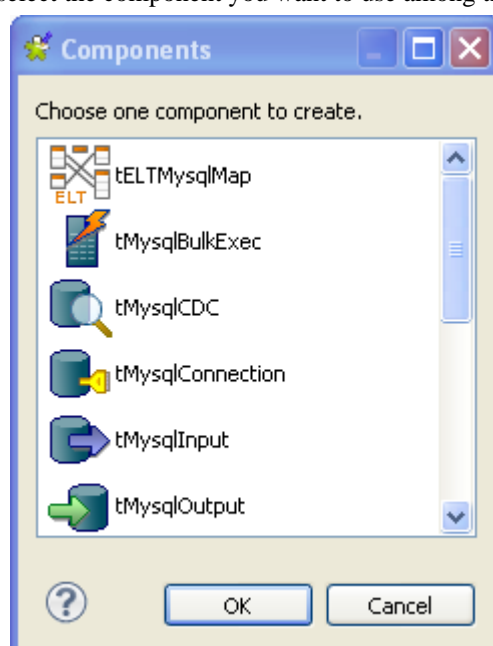
Different wizards will help you centralize connection and schema metadata in the **Repository** tree view. For more information about the **Metadata Manager** wizards, see [Section 4.4.1, “How to centralize the Metadata items”](#).

Once the relevant metadata is stored under the **Metadata** node, you will be able to drop the corresponding components directly onto the design workspace.

1. In the **Repository** tree view, expand **Metadata** and the folder holding the connection you want to use in your Job.
2. Drop the relevant connection/schema onto the design workspace.



A dialog box prompts you to select the component you want to use among those offered.



3. Select the component and then click **OK**. The selected component displays on the design workspace.

Alternatively, according to the type of component (Input or Output) that you want to use, perform one of the following operations:

- **Output:** Press **Ctrl** on your keyboard while you are dropping the component onto the design workspace to directly include it in the active Job.
- **Input:** Press **Alt** on your keyboard while you drop the component onto the design workspace to directly include it in the active Job.

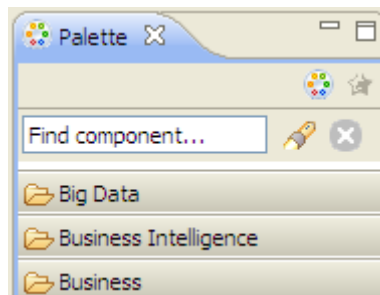
If you double-click the component, the **Component** view shows the selected connection details as well as the selected schema information.




If you select the connection without selecting a schema, then the properties will be filled with the first encountered schema.

4.2.3. How to search components in the Palette

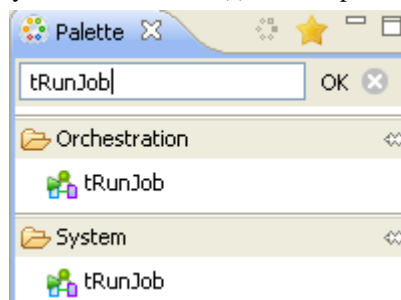
If you do not want to browse the components families in the **Palette** to find the components you want to use in your Job, you can search the desired component directly in the search field at the top of the **Palette**.



To search for a component, do the following:

1. Click  to clear the search field of any text.
2. Enter the name of the component you want to look for and click **OK**.

The **Palette** displays only the family/families that hold(s) the component.



To go back to the default **Palette** settings, click .

4.2.4. How to connect components together

A Job or a subjob is defined by a group of components interconnected in a logical way. The Job itself can be built with several subjobs carrying out various processings.

The component forming a subjob, as well as the subjobs are connected to each other using various types of connections.

Also, a Job (made of one or more subjobs) can be preceded by a pre-job and followed by a post-job components, in order to ensure that some specific tasks (often not related to the actual data processing) are performed first or last in the process. For more information, see [Section 4.5.6, “How to use the tPrejob and tPostjob components”](#).

To connect two components, right-click the source component on your design workspace, select your type of connection from the contextual menu, and click the target component.

When dragging the link from your source component towards the target component, a graphical plug indicates if the destination component is valid or not. The black crossed circle disappears only when you reach a valid target component.

Only the connections authorized for the selected component are listed on the right-click contextual menu.

The types of connections proposed are different for each component according to its nature and role within the Job, i.e. if the connection is meant to transfer data (from a defined schema) or if no data is handled.

The types of connections available also depend if data comes from one or multiple input files and gets transferred towards one or multiple outputs.

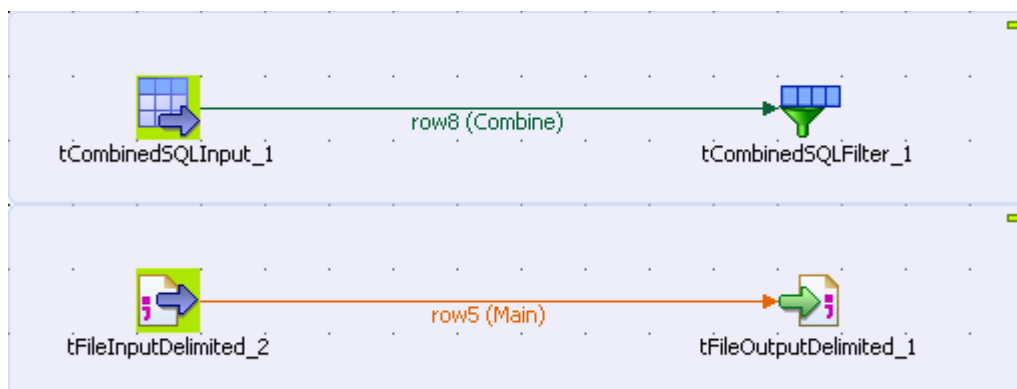
For more information about the various types of connections and their specific settings, see [Section 4.3, “Using connections”](#).

4.2.5. How to drop components in the middle of a Row link

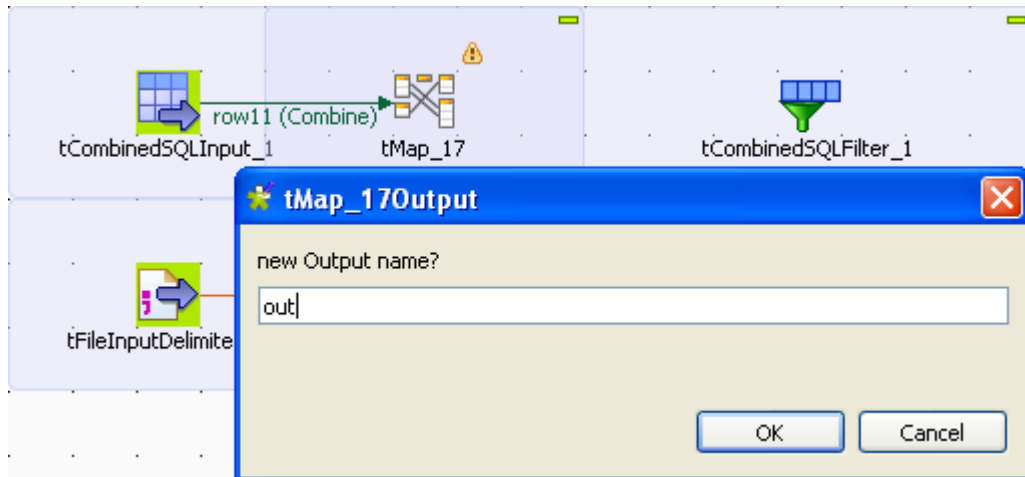
When creating a Job, *Talend Open Studio for ESB* enables you to insert a component in the middle of a **Row > Main**, **Row > Reject** or **Row > Combine** connection.

To do so, complete the following:

1. Drop two combine and two file components from the **Palette** onto the design workspace.
2. Connect the component pairs using a **Row > Main** (or a **Row > Reject**) connection and a **Row > Combine** one.



3. Drop the component you want to insert in the middle of the row. The link gets bold and then a dialog box displays, prompting you to type in a name for the output link.

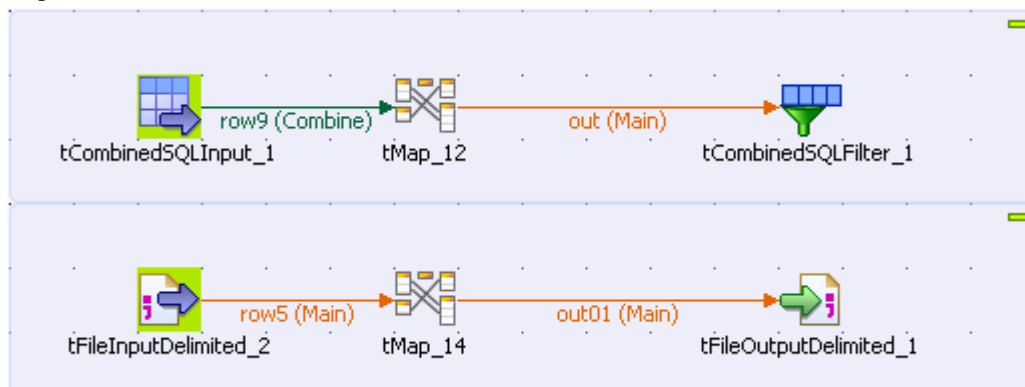


4. Type in a name and click **OK** to close the dialog box.



You may be asked to retrieve the schema of the target component. In that case, click **OK** to accept or click **No** to deny.

The component is inserted in the middle of the link, which is now divided in two links.



4.2.6. How to define component properties

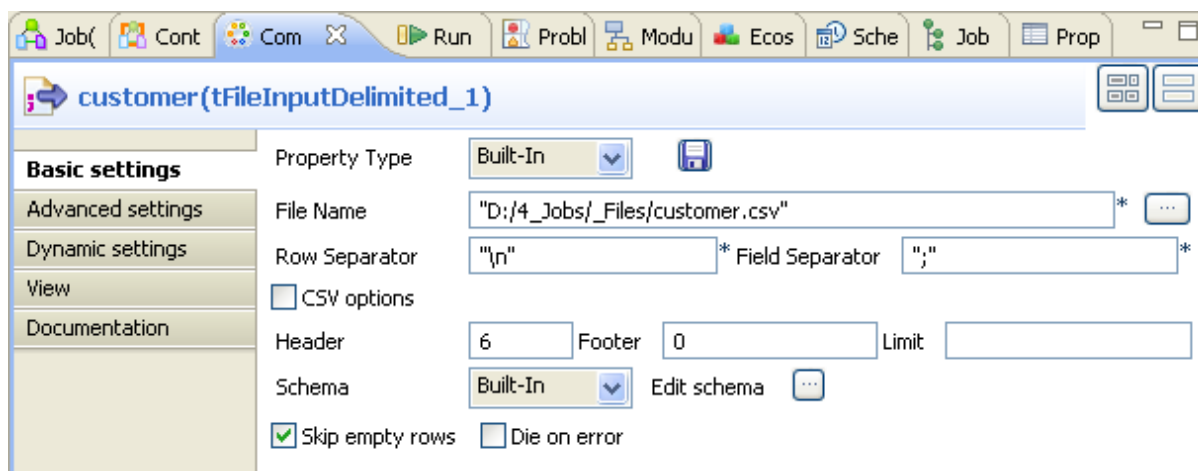
The properties information for each component forming a Job or a subjob allows to set the actual technical implementation of the active Job.

Each component is defined by basic and advanced properties shown respectively on the **Basic Settings** tab and the **Advanced Settings** tab of the **Component** view of the selected component in the design workspace. The **Component** view gathers also other collateral information related to the component in use, including **View** and **Documentation** tabs.

For detailed configuration for each component displaying in the **Palette**, check out the *Talend Open Studio Components Reference Guide*.

4.2.6.1. Basic Settings tab

The **Basic Settings** tab is part of the **Component** view, which is located on the lower part of the designing editor of *Talend Open Studio for ESB*.



Each component has specific basic settings according to its function requirements within the Job. For a detailed description of each component properties and use, see the *Talend Open Studio Components Reference Guide*.



Some components require code to be input or functions to be set. Make sure you use Java code in properties.

For **File** and **Database** components, you can centralize properties in metadata files located in the **Metadata** directory of the **Repository** tree view. This means that on the **Basic Settings** tab you can set properties on the spot, using the **Built-In Property Type** or use the properties you stored in the **Metadata Manager** using the **Repository Property Type**. The latter option helps you save time.

Select **Repository** as **Property Type** and choose the metadata file holding the relevant information. Related topic: [Section 4.4.1, “How to centralize the Metadata items”](#).

Alternatively, you can drop the **Metadata** item from the **Repository** tree view directly to the component already dropped on the design workspace, for its properties to be filled in automatically.

If you selected the **Built-in** mode and set manually the properties of a component, you can also save those properties as metadata in the **Repository**. To do so:

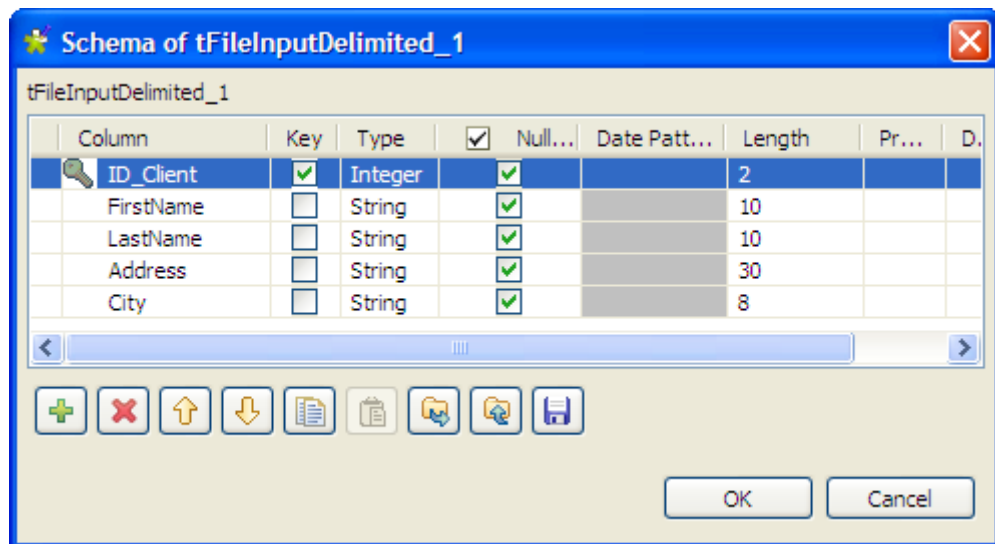
1. Click the floppy disk icon. The metadata creation wizard corresponding to the component opens.
2. Follow the steps in the wizard. For more information about the creation of metadata items, see [Chapter 9, Managing Metadata](#).
3. The metadata displays under the **Metadata** node of the **Repository**.

For all components that handle a data flow (most components), you can define a **Talend** schema in order to describe and possibly select the data to be processed. Like the **Properties** data, this schema is either **Built-in** or stored remotely in the **Repository** in a metadata file that you created. A detailed description of the Schema setting is provided in the next sections.

How to set a built-in schema

A schema created as **Built-in** is meant for a single use in a Job, hence cannot be reused in another Job.

Select **Built-in** in the **Property Type** list of the **Basic settings** view, and click the **Edit Schema** button to create your built-in schema by adding columns and describing their content, according to the input file definition.



In all output properties, you also have to define the schema of the output. To retrieve the schema defined in the input schema, click the **Sync columns** tab in the **Basic settings** view.

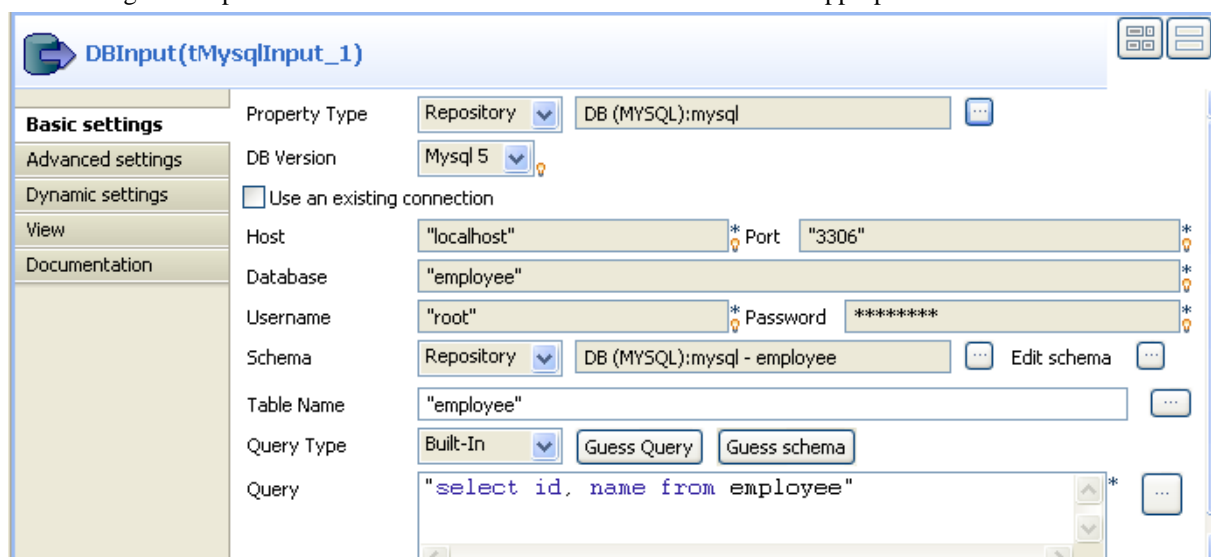


Some extra information is required. For more information about Date pattern for example, check out: <http://docs.oracle.com/javase/6/docs/api/index.html>.

How to set a repository schema

If you often use certain database connections or specific files when building your data integration Jobs, you can avoid defining the same properties over and over again by creating metadata files and storing them in the **Metadata** node in the **Repository** tree view

To recall a metadata file into your current Job, select **Repository** in the **Schema** list and then select the relevant metadata file. Or, drop the metadata item from the **Repository** tree view directly to the component already dropped on the design workspace. Then click **Edit Schema** to check that the data is appropriate.



You can edit a repository schema used in a Job from the **Basic settings** view. However, note that the schema hence becomes **Built-in** in the current Job.



You cannot change the schema stored in the repository from this window. To edit the schema stored remotely, right-click it under the **Metadata** node and select the corresponding edit option (**Edit connection** or **Edit file**) from the contextual menu.

Related topics: [Section 4.4.1, “How to centralize the Metadata items”](#).

How to set a field dynamically (Ctrl+Space bar)

On any field of your Job/component **Properties** view, you can use the **Ctrl+Space** bar to access the global and context variable list and set the relevant field value dynamically.

1. Place the cursor on any field of the **Component** view.
2. Press **Ctrl+Space bar** to access the proposal list.
3. Select on the list the relevant parameters you need. Appended to the variable list, a information panel provides details about the selected parameter.

<p>Description: Error Message</p> <p>Global variable, property of component tMap [tMap_1].</p> <p>Type: String</p> <p>Availability: After</p> <p>Variable Name: ((String)globalMap.get("tMap_1_ERROR_MESSAGE"))</p>	<p>tFileInputDelimited_2.ERROR_MESSAGE</p> <p>tFileInputDelimited_2.NB_LINE</p> <p>tMap_1.ERROR_MESSAGE</p> <p>tFileOutputDelimited_1.ERROR_MESSAGE</p> <p>tFileOutputDelimited_1.NB_LINE</p> <p>tFileOutputDelimited_2.ERROR_MESSAGE</p> <p>tFileOutputDelimited_2.NB_LINE</p> <p>tFileInputDelimited_3.ERROR_MESSAGE</p> <p>tFileInputDelimited_3.NB_LINE</p> <p>tFlowMeter_1.ERROR_MESSAGE</p> <p>tFlowMeter_2.ERROR_MESSAGE</p>
---	--

This can be any parameter including: error messages, number of lines processed, or else... The list varies according to the component in selection or the context you are working in.

Related topic: [Section 4.4.2, “How to centralize contexts and variables”](#).

4.2.6.2. Advanced settings tab

Some components, especially **File** and **Databases** components, provides numerous advanced use possibilities.

Column	Trim
key	<input type="checkbox"/>
value	<input type="checkbox"/>

The content of the **Advanced settings** tab changes according to the selected component.

Generally you will find on this tab the parameters that are not required for a basic or usual use of the component but may be required for a use out of the standard scope.

How to measure data flows

You can also find in the **Advanced settings** view the option **tStatCatcher Statistics** that allows you, if selected, to display logs and statistics about the current Job without using dedicated components. For more information regarding the stats & log features, see [Section 4.6.7.1, “How to automate the use of statistics & logs”](#).

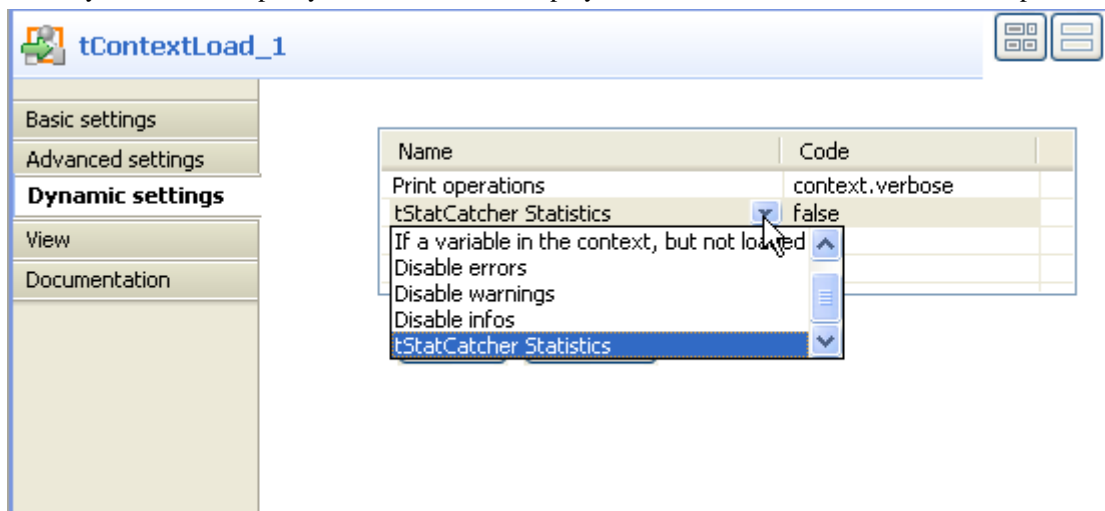
4.2.6.3. Dynamic settings tab

The **Basic settings** and **Advanced settings** tabs of all all components display various check boxes and drop-down lists for component parameters. Usually, available values for these types of parameters are either *true* or *false* and can only be edited when designing your Job.

The **Dynamic settings** tab, on the **Component** view, allows you to customize these parameters into code or variable.

This feature allows you, for example, to define these parameters as variables and thus let them become context-dependent, whereas they are not meant to be by default.

Another benefit of this feature is that you can now change the context setting at execution time. This makes full sense when you intend to export your Job in order to deploy it onto a Job execution server for example.



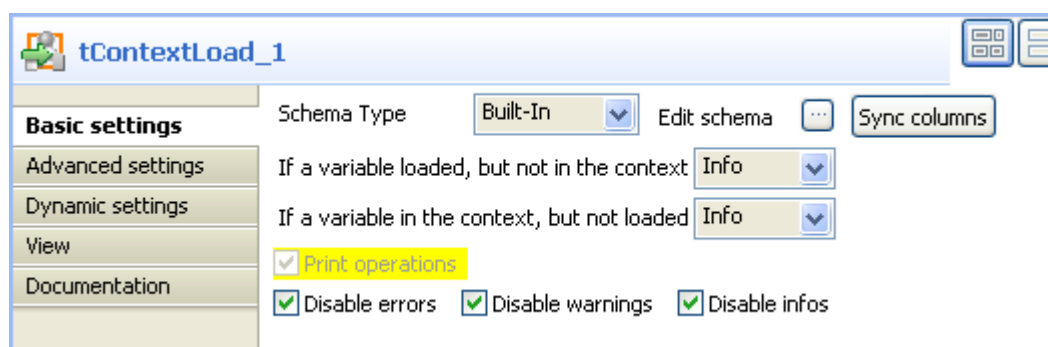
To customize these types of parameters, as context variables for example, follow the following steps:

1. Select the relevant component basic settings or advanced settings view that contains the parameter you want to define as a variable.
2. Click the **Dynamic settings** tab.
3. Click the **plus** button to display a new parameter line in the table.
4. Click the **Name** of the parameter displaying to show the list of available parameters. For example: *Print operations*
5. Then click in the facing **Code** column cell and set the code to be used. For example: *context.verbose* if you create the corresponding context variable, called *verbose*.



As code, you can input a context variable or a piece of Java code.

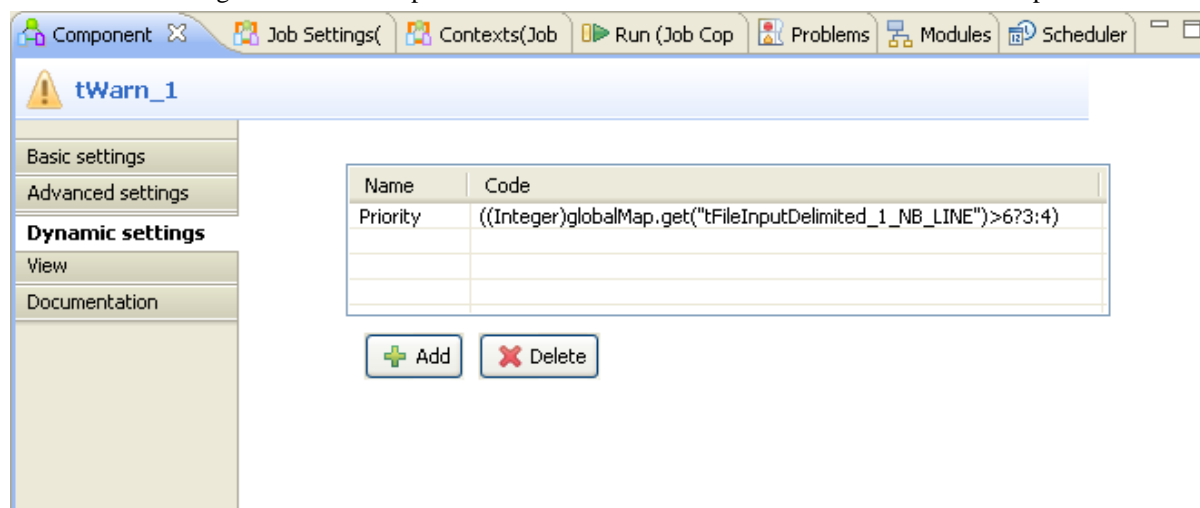
The corresponding lists or check boxes thus become unavailable and are highlighted in yellow in the **Basic settings** or **Advanced settings** tab.



If you want to set a parameter as context variable, make sure you create the corresponding variable in the **Context** view.

For more information regarding the context variable definition, see [Section 4.4.2.2, “How to use variables in the Contexts view”](#).

You can also use a global variable or pieces of Java code to store the values to be used for each parameter.



For example, use some global variable available through the **Ctrl+Space** bar keys, and adapt it to your context.

4.2.6.4. View tab

The **View** tab of the **Component** view allows you to change the default display format of components on the design workspace.

Field	Description
Label format	Free text label showing on the design workspace. Variables can be set to retrieve and display values from other fields. The field tooltip usually shows the corresponding variable where the field value is stored.
Hint format	Hidden tooltip, showing only when you mouse over the component.
Connection format	Indicates the type of connection accepted by the component.

You can graphically highlight both **Label** and **Hint** text with HTML formatting tags:

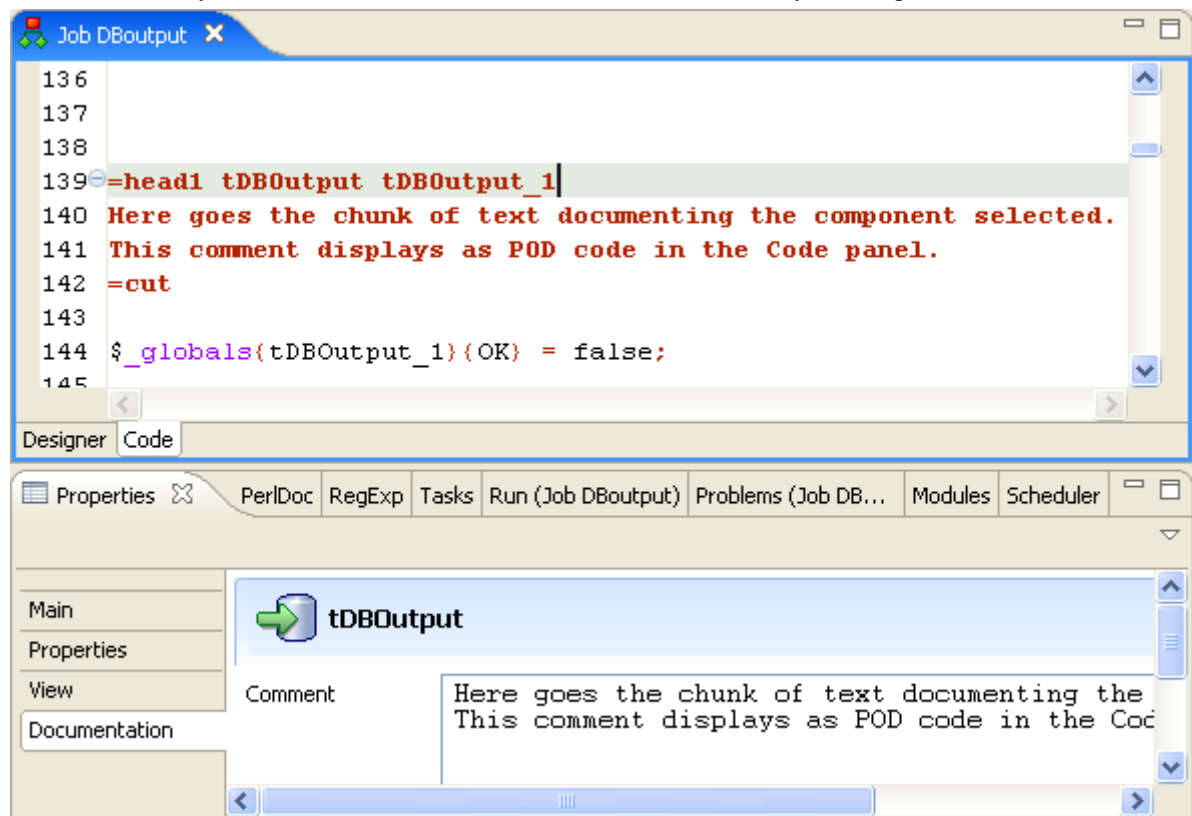
- Bold: ` YourLabelOrHint `
- Italic: `<i> YourLabelOrHint </i>`

- Return carriage: YourLabelOrHint
 ContdOnNextLine
- Color: YourLabelOrHint

To change your preferences of this **View** panel, click **Window>Preferences>Talend>Designer**.

4.2.6.5. Documentation tab

Feel free to add any useful comment or chunk of text or documentation to your component.



In the **Documentation** tab, you can add your text in the **Comment** field. Then, select the **Show Information** check box and an information icon display next to the corresponding component in the design workspace.

You can show the Documentation in your hint tooltip using the associated variable `_COMMENT_`, so that when you place your mouse on this icon, the text written in the **Comment** field displays in a tooltip box.

For advanced use of Documentations, you can use the **Documentation** view in order to store and reuse any type of documentation.

4.2.7. How to run a Job

You can execute a Job in several ways. This mainly depends on the purpose of your Job execution and on your user level.

If you are an advanced Java user and want to execute your Job step by step to check and possibly modify it on the run, see [Section 4.2.7.2, “How to run a Job in Java Debug mode”](#).

If you do not have advanced Java knowledge and want to execute and monitor your Job in normal mode, see [Section 4.2.7.1, “How to run a Job in normal mode”](#).

4.2.7.1. How to run a Job in normal mode



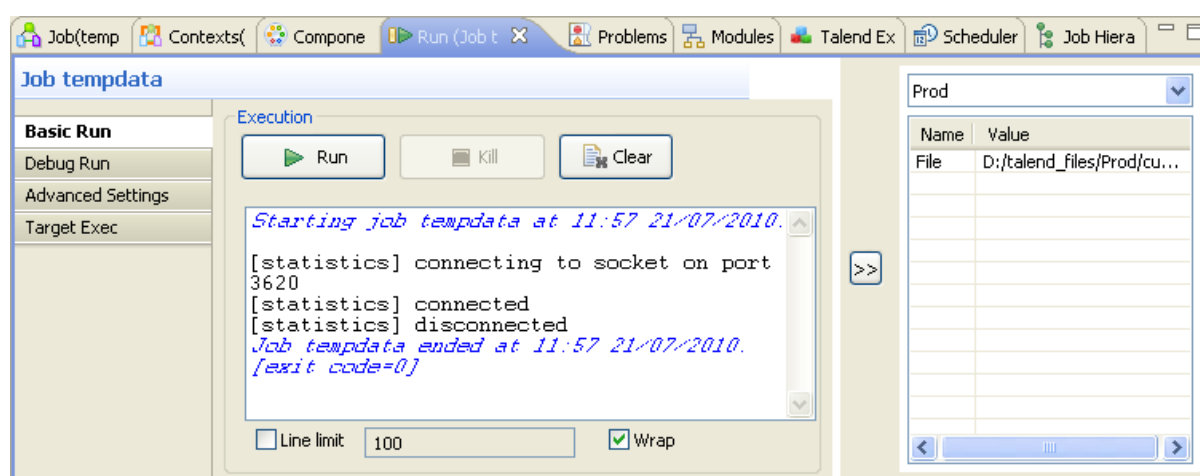
Make sure you saved your **Job** before running it in order for all properties to be taken into account.

To run your Job in a normal mode, complete the following:

1. Click the **Run** view to access it.
2. Click the **Basic Run** tab to access the normal execution mode.
3. In the **Context** area to the right of the view, select in the list the proper context for the Job to be executed in. You can also check the variable values.

If you have not defined any particular execution context, the context parameter table is empty and the context is the default one. Related topic: [Section 4.4.2, “How to centralize contexts and variables”](#).

1. Click **Run** to start the execution.
2. On the same view, the console displays the progress of the execution. The log includes any error message as well as start and end messages. It also shows the Job output in case of a **tLogRow** component is used in the Job design.
3. To define the lines of the execution progress to be displayed in the console, select the **Line limit** check box and type in a value in the field.
4. Select the **Wrap** check box to wrap the text to fit the console width. This check box is selected by default. When it is cleared, a horizontal scrollbar appears, allowing you to view the end of the lines.



Before running again a Job, you might want to remove the execution statistics and traces from the designing workspace. To do so, click the **Clear** button.

If for any reason, you want to stop the Job in progress, simply click the **Kill** button. You will need to click the **Run** button again, to start again the Job.

Talend Open Studio for ESB offers various informative features displayed during execution, such as statistics and traces, facilitating the Job monitoring and debugging work. For more information, see the following sections.

4.2.7.2. How to run a Job in Java Debug mode

To follow step by step the execution of a Job to identify possible bugs, you can run it in Debug mode.

To access the Debug mode:

1. Click the **Run** view to access it.
2. Click the **Debug Run** tab to access the debug execution modes.



In order to be able to run a Job in Debug mode, you need the EPIC module to be installed.

Before running your Job in Debug mode, add breakpoints to the major steps of your Job flow.



This will allow you to get the Job to automatically stop at each breakpoint. This way, components and their respective variables can be verified individually and debugged if required.

To add breakpoints to a component, right-click it on the design workspace, and select **Add breakpoint** on the contextual menu.

A pause icon displays next to the component where the break is added.

To switch to debug mode, click the **Java Debug** button on the **Debug Run** tab of the **Run** panel. *Talend Open Studio for ESB's* main window gets reorganized for debugging.

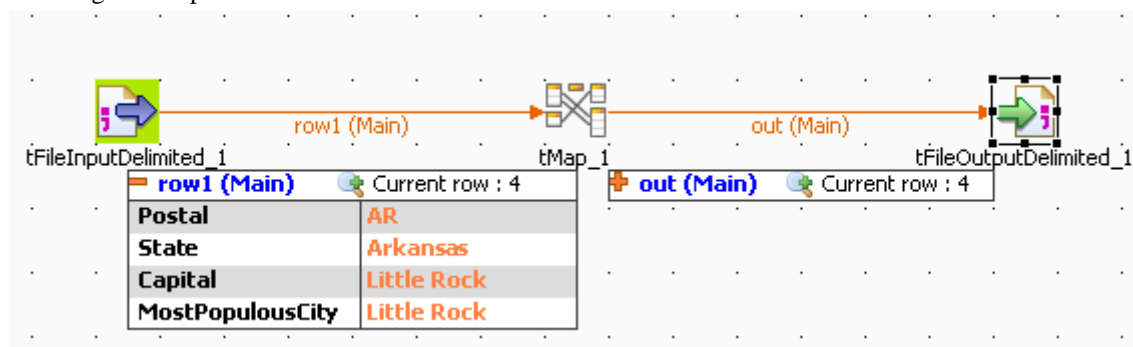
You can then run the Job step by step and check each breakpoint component for the expected behavior and variable values.

To switch back to *Talend Open Studio for ESB* designer mode, click **Window**, then **Perspective** and select *Talend Open Studio for ESB*.

4.2.7.3. How to run a Job in Traces Debug mode

The traces feature allows you to monitor data processing when running a Job in *Talend Open Studio for ESB*.

It provides a row by row view of the component behavior and displays the dynamic result next to the **Row** link on the design workspace.



This feature allows you to monitor all the components of a Job, without switching to the debug mode, hence without requiring advanced Java knowledge.

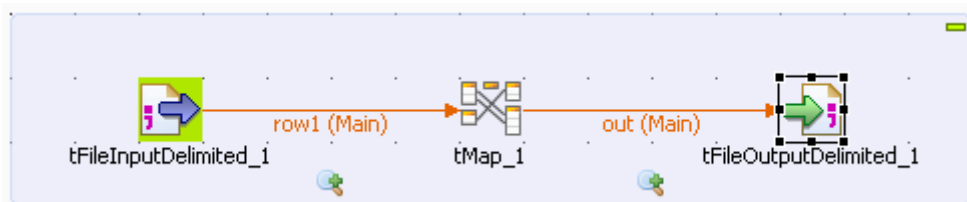
The **Traces** function displays the content of processed rows in a table.



Exception is made for external components which cannot offer this feature if their design does not include it.

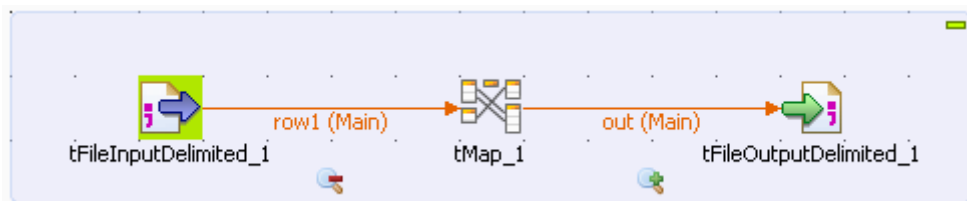
You can activate or deactivate **Traces** or decide what processed columns to display in the traces table that displays on the design workspace when launching the current Job.

To activate the **Traces** mode in a Job:



1. Click the **Run** view.
2. Click the **Debug Run** tab to access the debug and traces execution modes.
3. Click the down arrow of the **Java Debug** button and select the **Traces Debug** option. An icon displays under every flow of your Job to indicate that process monitoring is activated.
4. Click the **Traces Debug** to execute the Job in Traces mode.

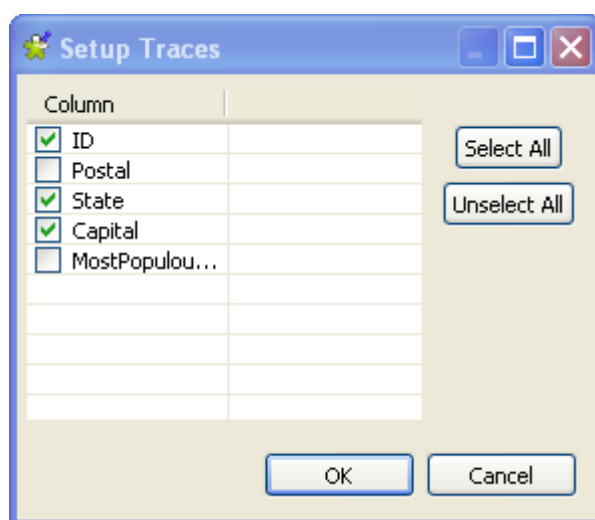
To deactivate the **Traces** on one of the flows in your Job:



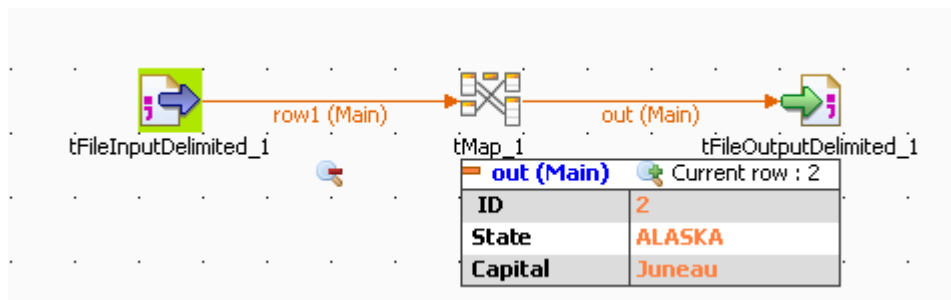
1. Right-click the **Traces** icon under the relevant flow.
2. Select **Disable Traces** from the list. A red minus sign replaces the green plus sign on the icon to indicate that the **Traces** mode has been deactivated for this flow.

To choose which columns of the processed data to display in the traces table, do the following:

1. Right-click the **Traces** icon for the relevant flow, then select **Setup Traces** from the list. The **[Setup Traces]** dialog box appears.



2. In the dialog box, clear the check boxes corresponding to the columns you do not want to display in the Traces table.
3. Click **OK** to close the dialog box.



Monitoring data processing starts when you execute the Job and stops at the end of the execution.

To remove the displayed monitoring information, click the **Clear** button in the **Debug Run** tab.

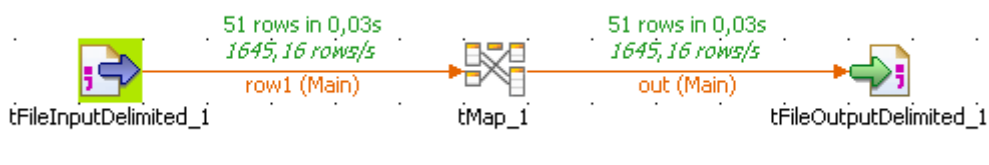
4.2.7.4. How to set advanced execution settings

Several advanced execution settings are available to make the execution of the Jobs handler:

- **Statistics**, this feature displays processing performance rate. For more information, see [the section called “How to display Statistics”](#).
- **Exec time**, this feature displays the execution time in the console at the end of the execution. For more information, see [the section called “How to display the execution time and other options”](#).
- **Save Job before execution**, this feature allows to automatically save the Job before its execution.
- **Clear before run**, this feature clears all the results of a previous execution before re-executing the Job.
- **JVM Setting**, this feature allows you to define the parameters of your JVM according to your needs, for example the parameters used to display special characters.

How to display Statistics

The **Statistics** feature displays each component performance rate, under the flow links on the design workspace.



It shows the number of rows processed and the processing time in row per second, allowing you to spot straight away any bottleneck in the data processing flow.

For trigger links like **OnComponentOK**, **OnComponentError**, **OnSubjobOK**, **OnSubjobError** and **If**, the **Statistics** option displays the state of this trigger during the execution time of your Job: Ok or Error and True or False.



Exception is made for external components which cannot offer this feature if their design does not include it.

In the **Run** view, click the **Advanced settings** tab and select the **Statistics** check box to activate the Stats feature and clear the box to disable it.

The calculation only starts when the Job execution is launched, and stops at the end of it.

Click the **Clear** button from the **Basic** or **Debug Run** views to remove the calculated stats displayed. Select the **Clear before Run** check box to reset the Stats feature before each execution.



The statistics thread slows down Job execution as the Job must send these stats data to the design workspace in order to be displayed.

You can also save your Job before the execution starts. Select the relevant option check box.

How to display the execution time and other options

To display the Job total execution time after Job execution, select in the **Advanced settings** tab of the **Run** view the **Exec time** check box before running the Job.

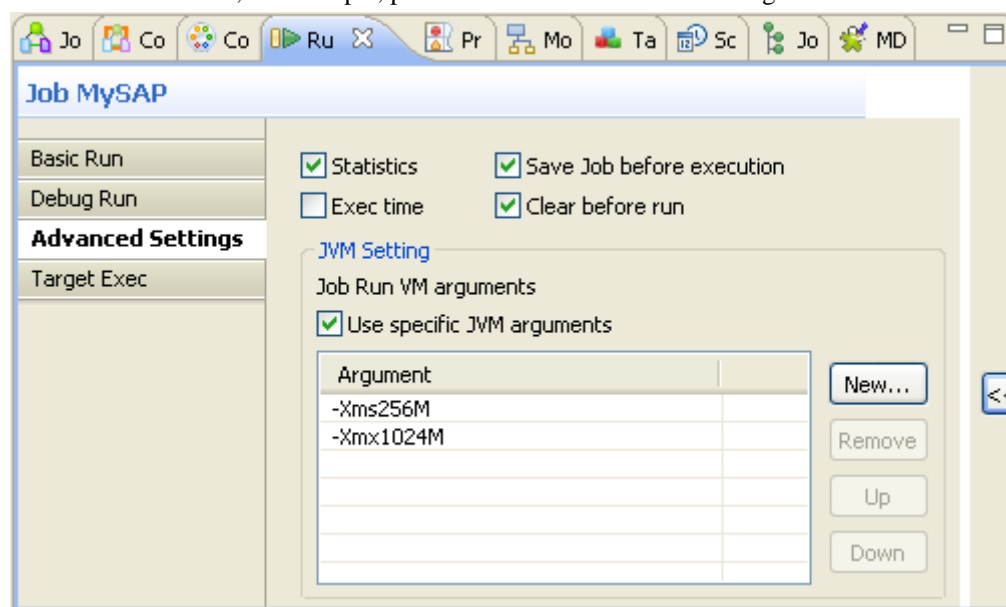
This way you can test your Job before going to production.

You can also clear the design workspace before each Job execution by selecting the check box **Clear before Run**.

You can also save your Job before the execution starts. Select the relevant option check box.

How to display special characters in the console

Talend Open Studio for ESB can display special characters in the console. To enable the display of Chinese, Japanese or Korean characters, for example, proceed as follows before executing the Job:



1. Select the **Advanced settings** tab.
2. In the **JVM settings** area of the tab view, select the **Use specific JVM arguments** checkbox to activate the **Argument** table.
3. Next to the **Argument** table, click the **New...** button to pop up the **[Set the VM argument]** dialog box.
4. In the dialog box, type in `-Dfile.encoding=UTF-8`.
5. Click **OK** to close the dialog box.

This argument can be applied for all of your Job executions in *Talend Open Studio for ESB*. For further information about how to apply this JVM argument for all of the Job executions, see [Section 2.5.5, “Debug and Job execution preferences”](#).

4.2.8. How to customize your workspace

When using *Talend Open Studio for ESB* to design a data integration Job, you can customize the **Palette** layout and setting according to your needs. You can as well change the position of any of the panels that exist in the Studio to meet your requirements.



All the panels, tabs, and views described in this documentation are specific to *Talend Open Studio for ESB*. Some views listed in the [Show View] dialog box are Eclipse specific and are not subjects of this documentation. For information on such views, check Eclipse online documentation at <http://www.eclipse.org/documentation/>.

4.2.8.1. How to change the Palette layout and settings

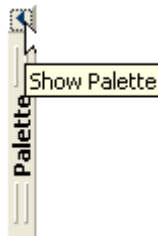
The **Palette** contains all basic technical components necessary to create the most complex Jobs in the design workspace. These components are grouped in families and sub-families.

For specific component configuration, check out the *Talend Open Studio Components Reference Guide*.

Talend Open Studio for ESB enables you to change the layout and position of your **Palette** according to your requirements. the below sections explain all management options you can carry out on the **Palette**.

How to show, hide the Palette and change its position

By default, the **Palette** might be hidden on the right hand side of your design workspace.



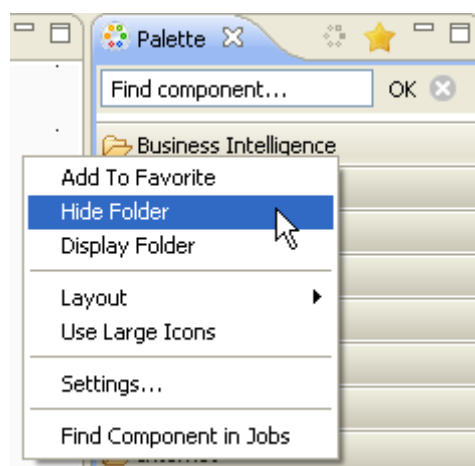
If you want the **Palette** to show permanently, click the left arrow, at the upper right corner of the design workspace, to make it visible at all times.

You can also move around the **Palette** outside the design workspace within *Talend Open Studio for ESB*'s main window. To enable the standalone **Palette** view, click the **Window** menu > **Show View...** > **General** > **Palette**.

If you want to set the Palette apart in a panel, right-click the **Palette** head bar and select **Detached** from the contextual menu. The **Palette** opens in a separate view that you can move around wherever you like within *Talend Open Studio for ESB*'s main window.

How to display/hide components families

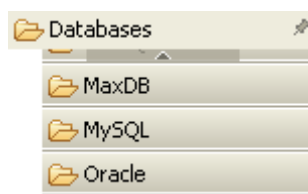
You can display/hide components families according to your needs in case of visibility problems, for example. To do so, right-click the **Palette** and select **Display folder** to display components families and **Hide folder** to display components without their families.



This display/hide option can be very useful when you are in the **Favorite** view of the **Palette**. In this view, you usually have a limited number of components that if you display without their families, you will have them in an alphabetical list and thus facilitate their usage. for more information about the **Palette** favorite, see [the section called “How to set the Palette favorite”](#).

How to maintain a component family open

If you often use one or many component families, you can add a pin on their names to stop them from collapsing when you select components from other families.



To add a pin, click the pin icon on the top right-hand corner of the family name.

How to filter the Palette

You can select the components to be shown or hidden on your **Palette**. You can also add to the **Palette** the components that you developed yourself.

For more information about filtering the **Palette**, see [Section 2.6.1, “Palette Settings”](#).

For more information about adding components to the **Palette**, either from **Talend Exchange** or from your own development, see [Section 4.5.3, “How to download/upload Talend Community components”](#) and/or [Section 2.5.2, “External or User components”](#).

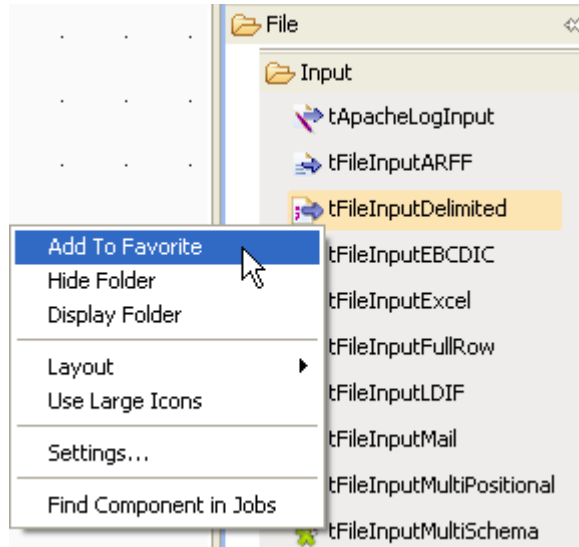
How to set the Palette favorite


The **Palette** offers you search and favorite possibilities that by turn facilitate its usage.

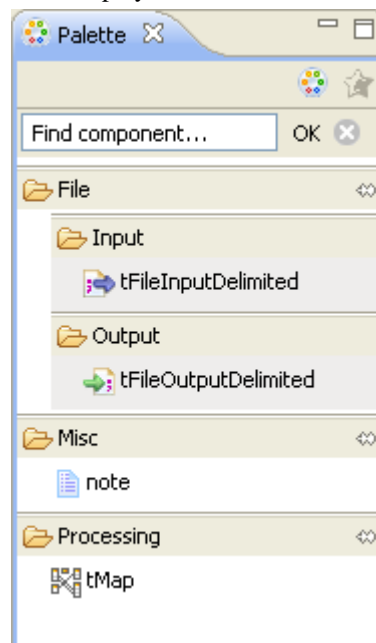
You can add/remove components to/from the **Palette** favorite view of *Talend Open Studio for ESB* in order to have a quick access to all the components that you mostly use.

To do so:

1. From the **Palette**, right-click the component you want to add to **Palette** favorite and select **Add To Favorite**.



2. Do the same for all the components you want to add to the **Palette** favorite then click the **Favorite**  button in the upper right corner of the **Palette** to display the **Palette** favorite.



Only the components added to the favorite are displayed.

To delete a component from the **Palette** favorite, right-click the component you want to remove from the favorite and select **Remove From Favorite**.

To restore the **Palette** standard view, click the **Standard**  button in the upper right corner of the **Palette**.

How to change components layout in the Palette

You can change the layout of the component list in the **Palette** to display them in columns or in lists, as icons only or as icons with short description.

You can also enlarge the component icons for better readability of the component list.

To do so, right-click any component family in the **Palette** and select the desired option in the contextual menu or click **Settings** to open the **[Palette Settings]** window and fine-tune the layout.

How to add external components to the Palette

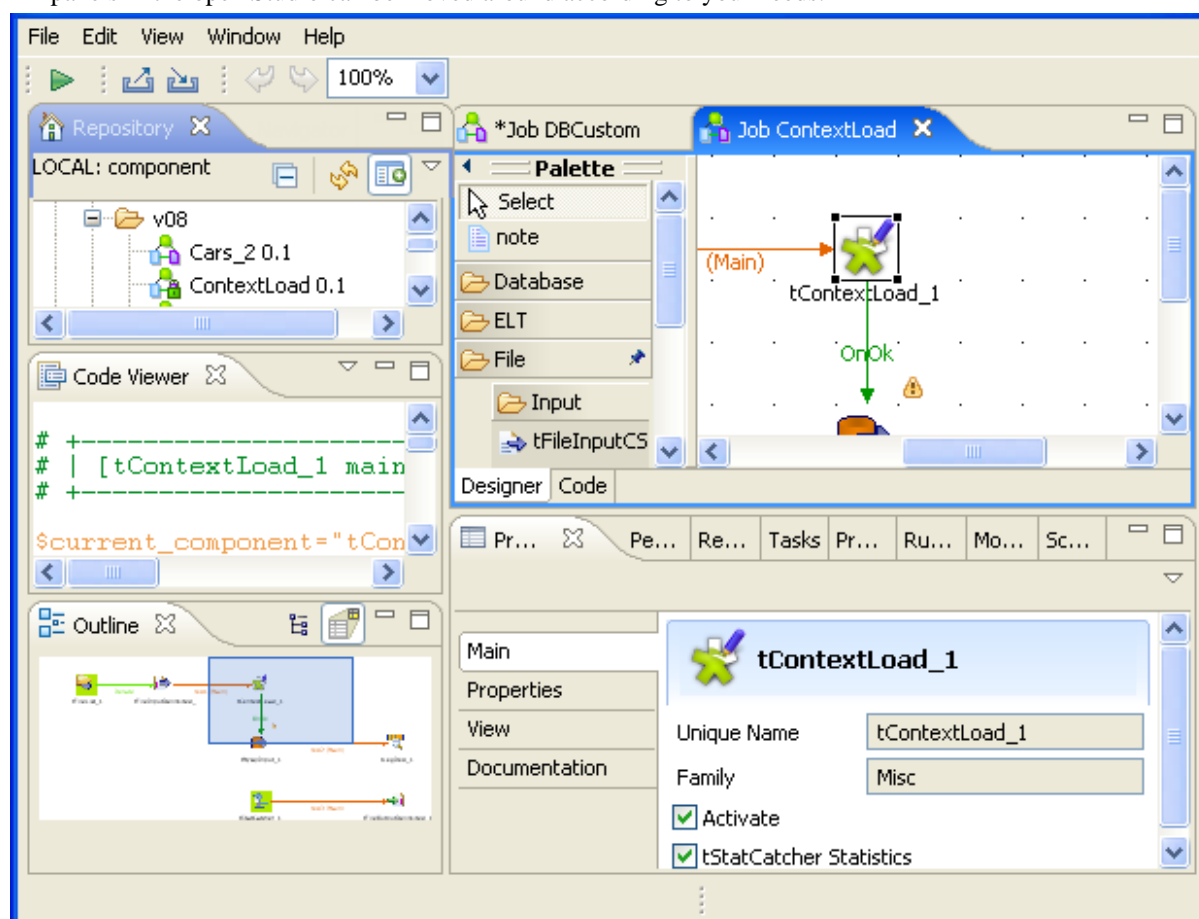
Talend Open Studio for ESB enables you to add external components to the **Palette** of your Studio and use them in your Job designs.

For more information about the creation and development of user components, refer to the component creation tutorial on our wiki at http://www.talendforge.org/wiki/doku.php?id=component_creation.

For more information about how to download user components in your Studio, see [Section 4.5.3, “How to download/upload Talend Community components”](#).

4.2.8.2. How to change panels positions

All panels in the open Studio can be moved around according to your needs.



All you need to do is to click the head border of a panel or to click a tab, hold down the mouse button and drag the panel to the target destination. Release to change the panel position.

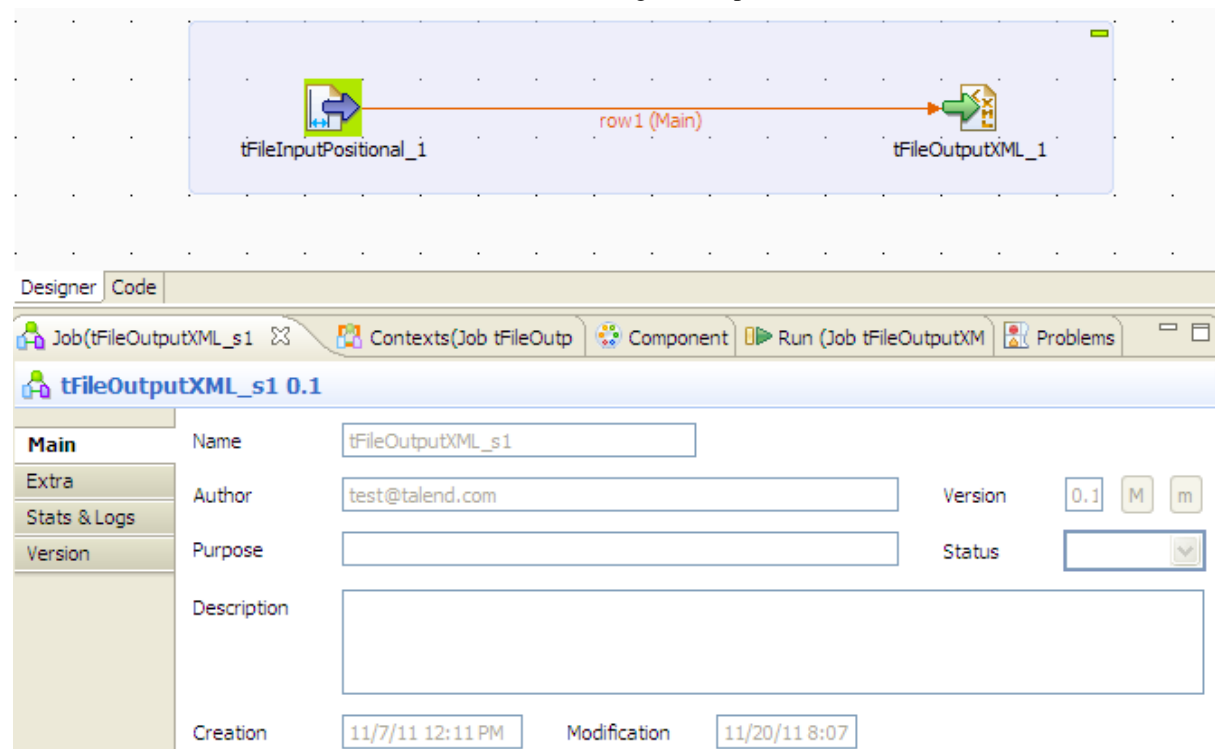
Click the minimize/maximize icons (☐/☐) to minimize the corresponding panel or maximize it. For more information on how to display or hide a panel/view, see [Section 4.2.8.3, “How to display Job configuration tabs/views”](#).

Click the close icon (✕) to close a tab/view. To reopen a view, click **Window > Show View > Talend**, then click the name of the panel you want to add to your current view or see [Section A.8, “Shortcuts and aliases”](#).

If the **Palette** does not show or if you want to set it apart in a panel, go to **Window > Show view...> General > Palette**. The **Palette** opens in a separate view that you can move around wherever you like within *Talend Open Studio for ESB*’s main window.

4.2.8.3. How to display Job configuration tabs/views

The configuration tabs are located in the lower half of the design workspace. Each tab opens a view that displays detailed information about the selected element in the design workspace.



The **Component**, **Run Job**, and **Contexts** views gather all information relative to the graphical elements selected in the design workspace or the actual execution of the open Job.



By default, when you launch *Talend Open Studio for ESB* for the first time, the **Problems** tab will not be displayed until the first Job is created. After that, **Problems** tab will be displayed in the tab system automatically.

The **Modules** and **Scheduler[deprecated]** tabs are located in the same tab system as the **Component**, **Logs** and **Run Job** tabs. Both views are independent from the active or inactive Jobs open on the design workspace.

Some of the configuration tabs are hidden by default such as the **Error Log**, **Navigator**, **Job Hierarchy**, **Problems**, **Modules** and **Scheduler[deprecated]** tabs. You can show hidden tabs in this tab system and directly open the corresponding view if you select **Window > Show view** and then, in the open dialog box, expand the corresponding node and select the element you want to display.

For detailed description about these tabs, see [Section 4.2.8.3, “How to display Job configuration tabs/views”](#).

4.3. Using connections

In *Talend Open Studio for ESB*, a Job or a subjob is composed of a group of components logically linked to one another via connections. This section will describe the types of connections and their related settings.

4.3.1. Connection types

There are various types of connections which define either the data to be processed, the data output, or the Job logical sequence.

Right-click a component on the design workspace to display a contextual menu that lists all available links for the selected component.

The sections below describe all available connection types.

4.3.1.1. Row connection

A **Row** connection handles the actual data. The **Row** connections can be **main**, **lookup**, **reject** or **output** according to the nature of the flow processed.

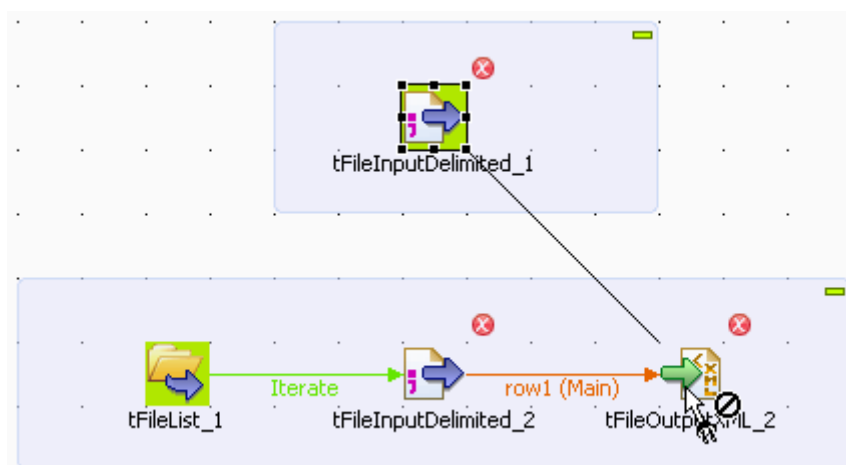
Main

This type of row connection is the most commonly used connection. It passes on data flows from one component to the other, iterating on each row and reading input data according to the component properties setting (schema).

Data transferred through main rows are characterized by a schema definition which describes the data structure in the input file.



You cannot connect two Input components together using a **main Row** connection. Only *one* incoming **Row** connection is possible per component. You will not be able to link twice the same target component using a main **Row** connection. The second row linking a component will be called **Lookup**.



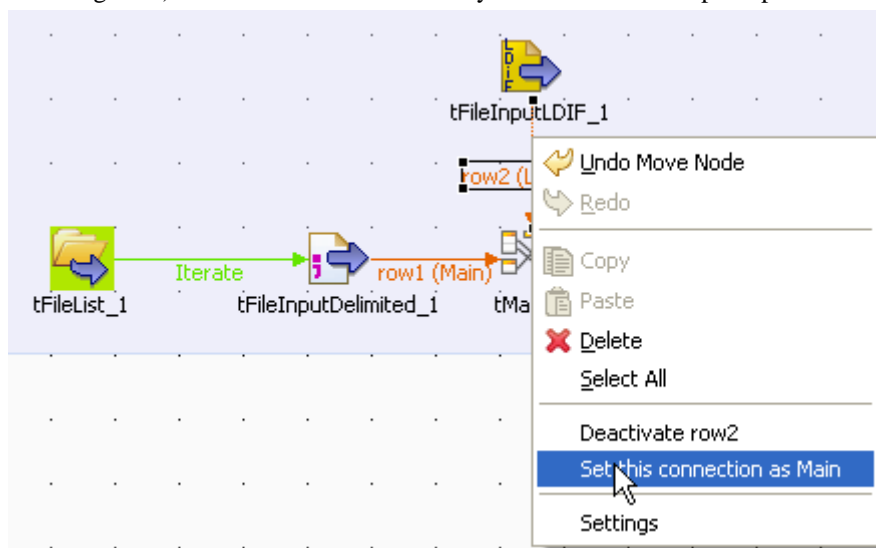
To connect two components using a Main connection, right-click the input component and select **Row > Main** on the connection list.

Alternatively, you can click the component to highlight it, then right-click it and drag the cursor towards the destination component. This will automatically create a **Row > Main** type of connection.

For information on using multiple **Row** connections, see [the section called “Multiple Input/Output”](#).

Lookup

This row link connects a sub-flow component to a main flow component (which should be allowed to receive more than one incoming flow). This connection is used only in the case of multiple input flows.



A **Lookup** row can be changed into a main row at any time (and reversely, a main row can be changed to a lookup row). To do so, right-click the row to be changed, and on the pop-up menu, click **Set this connection as Main**.

Related topic: [the section called “Multiple Input/Output”](#).

Filter

This row link connects specifically a **tFilterRow** component to an output component. This row link gathers the data matching the filtering criteria. This particular component offers also a **Reject** link to fetch the non-matching data flow.

Rejects

This row link connects a processing component to an output component. This row link gathers the data that does NOT match the filter or are not valid for the expected output. This link allows you to track the data that could not be processed for any reason (wrong type, undefined null value, etc.). On some components, this link is enabled when the **Die on error** option is deactivated. For more information, refer to the relevant component properties available in the *Talend Open Studio for ESB Reference Guide*.

ErrorReject

This row link connects a **tMap** component to an output component. This link is enabled when you clear the **Die on error** check box in the **tMap editor** and it gathers data that could not be processed (wrong type, undefined null value, unparseable dates, etc.).

Related topic: [Section 8.2.4.6, “Handling errors”](#).

Output

This row link connects a **tMap** component to one or several output components. As the Job output can be multiple, you get prompted to give a name for each output row created.



The system also remembers deleted output link names (and properties if they were defined). This way, you do not have to fill in again property data in case you want to reuse them.

Related topic: [the section called “Multiple Input/Output”](#).

Uniques/Duplicates

These row links connect a **tUniqRow** to output components.

The **Uniques** link gathers the rows that are found first in the incoming flow. This flow of unique data is directed to the relevant output component or else to another processing subjob.

The **Duplicates** link gathers the possible duplicates of the first encountered rows. This reject flow is directed to the relevant output component, for analysis for example.

Multiple Input/Output

Some components help handle data through multiple inputs and/or multiple outputs. These are often processing-type components such as the **tMap**.

If this requires a join or some transformation in one flow, you want to use the **tMap** component, which is dedicated to this use.

For further information regarding data mapping, see [Chapter 4, *Designing a Job*](#).

For properties regarding the **tMap** component as well as use case scenarios, see Talend Open Studio Components Reference Guide.

4.3.1.2. Iterate connection

The **Iterate** connection can be used to loop on files contained in a directory, on rows contained in a file or on DB entries.

A component can be the target of only one **Iterate** link. The **Iterate** link is mainly to be connected to the start component of a flow (in a subjob).

Some components such as the **tFileList** component are meant to be connected through an iterate link with the next component. For how to set an **Iterate** connection, see [Section 4.3.2.2, “Iterate connection settings”](#).

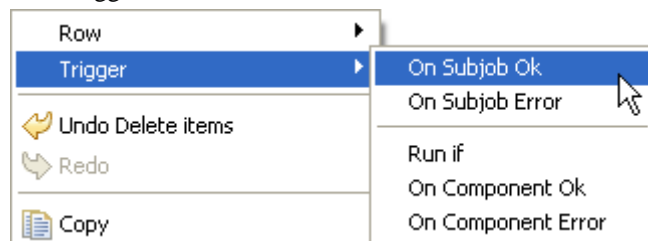


The name of the **Iterate** link is read-only unlike other types of connections.

4.3.1.3. Trigger connections

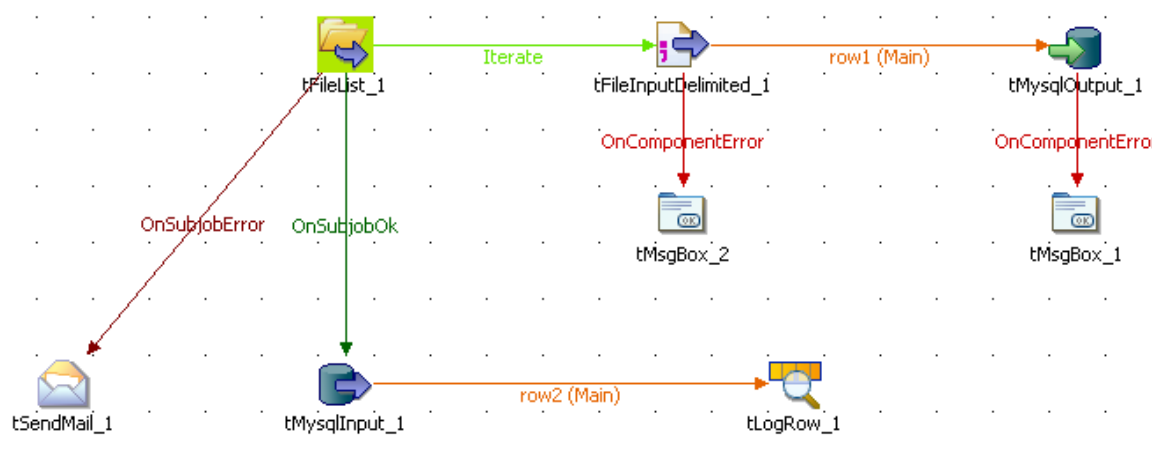
Trigger connections define the processing sequence, i.e. no data is handled through these connections.

The connection in use will create a dependency between Jobs or subjobs which therefore will be triggered one after the other according to the trigger nature.



Trigger connections fall into two categories:

- subjob triggers: **On Subjob Ok**, **On Subjob Error** and **Run if**,
- component triggers: **On Component Ok**, **On Component Error** and **Run if**.



OnSubjobOK (previously **Then Run**): This link is used to trigger the next subjob on the condition that the main subjob completed without error. This connection is to be used only from the start component of the Job.

These connections are used to orchestrate the subjobs forming the Job or to easily troubleshoot and handle unexpected errors.

OnSubjobError: This link is used to trigger the next subjob in case the first (main) subjob do not complete correctly. This “on error” subjob helps flagging the bottleneck or handle the error if possible.

Related topic: [Section 4.6.2, “How to define the Start component”](#).

OnComponentOK and **OnComponentError** are component triggers. They can be used with any source component on the subjob.

OnComponentOK will only trigger the target component once the execution of the source component is complete without error. Its main use could be to trigger a notification subjob for example.

OnComponentError will trigger the sub-job or component as soon as an error is encountered in the primary Job.

Run if triggers a subjob or component in case the condition defined is met. For how to set a trigger condition, see [the section called “Run if connection settings”](#).

4.3.1.4. Link connection

The **Link** connection can only be used with ETL components. These links transfer table schema information to the ETL mapper component in order to be used in specific DB query statements.

Related topics: ETL components in the *Talend Open Studio Components* Reference Guide.

The **Link** connection therefore does not handle actual data but only the metadata regarding the table to be operated on.

When right-clicking the ETL component to be connected, select **Link > New Output**.



Be aware that the name you provide to the link MUST reflect the actual table name.

In fact, the link name will be used in the SQL statement generated through the ETL Mapper, therefore the same name should never be used twice.

4.3.2. How to define connection settings

You can display the properties of a connection by selecting it and clicking the **Component** view tab, or by right-clicking the connection and selecting **Settings** from the contextual menu. This section summarizes connection property settings.

4.3.2.1. Row connection settings

The **Basic settings** vertical tab of the **Component** view of the connection displays the schema of the data flow handled by the connection. You can change the schema by clicking the **Edit schema** button. Once you change the schema of the data flow, the schema type of the two components across the connection will become **Built-In**. For more information, see [the section called “How to set a built-in schema”](#).

Column	Key	T...	✓	N..	Date ...	L...	P...	D..	C...
id	<input checked="" type="checkbox"/>	In...	<input checked="" type="checkbox"/>			2	0		
Custome...	<input type="checkbox"/>	St...	<input checked="" type="checkbox"/>			255	0		
Custome...	<input type="checkbox"/>	St...	<input checked="" type="checkbox"/>			255	0		
idState	<input type="checkbox"/>	In...	<input checked="" type="checkbox"/>			2	0		
id2	<input type="checkbox"/>	In...	<input checked="" type="checkbox"/>			2	0		
RegTime	<input type="checkbox"/>	St...	<input checked="" type="checkbox"/>			50	0		
Register...	<input type="checkbox"/>	St...	<input checked="" type="checkbox"/>			50	0		

The **Advanced settings** vertical tab lets you monitor the data flow over the connection in a Job without using a separate **tFlowMeter** component. The measured information will be interpreted and displayed in a supervising tool such as *Talend Activity Monitoring Console*. For information about *Talend Activity Monitoring Console*, see *Talend Activity Monitoring Console User Guide*.

row1

Basic settings

Advanced settings

☐ Use input connection name as label

Label: "MyLabel"

Mode: Absolute *

Thresholds:

Label	Low end	Top end	Color

☒ Monitor this connection

To monitor the data over the connection, perform the following settings in the **Advanced settings** vertical tab:

1. Select the **Monitor this connection** check box.
2. Select **Use input connection name as label** to use the name of the input flow to label your data to be logged, or enter a label in the **Label** field.
3. From the **Mode** list, select **Absolute** to log the actual number of rows passes over the connection, or **Relative** to log the ratio (%) of the number of rows passed over this connection against a reference connection. If you select **Relative**, you need to select a reference connection from the **Connections List** list.
4. Click the plus button to add a line in the **Thresholds** table and define a range of the number of rows to be logged.

For more information about flow metrics, see tFlowMeterCatcher component in the *Talend Open Studio Components Reference Guide* and see *Talend Activity Monitoring Console User Guide*.

4.3.2.2. Iterate connection settings

You can set an **Iterate** link to run parallel iterations:

1. Simply select the **Iterate** link of your subjob to display the related **Basic settings** view of the **Components** tab.
2. Select the **Enable parallel execution** check box and set the number of executions to be carried out in parallel.

iterate1

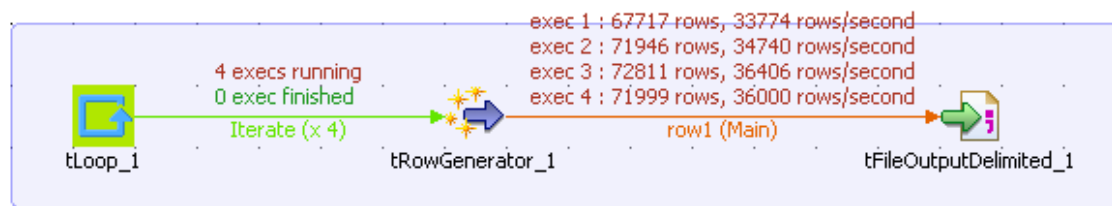
Basic settings

Advanced settings

☒ Enable parallel execution

Number of parallel execution: 4

When executing your Job, the number of parallel iterations will be distributed onto the available processors.

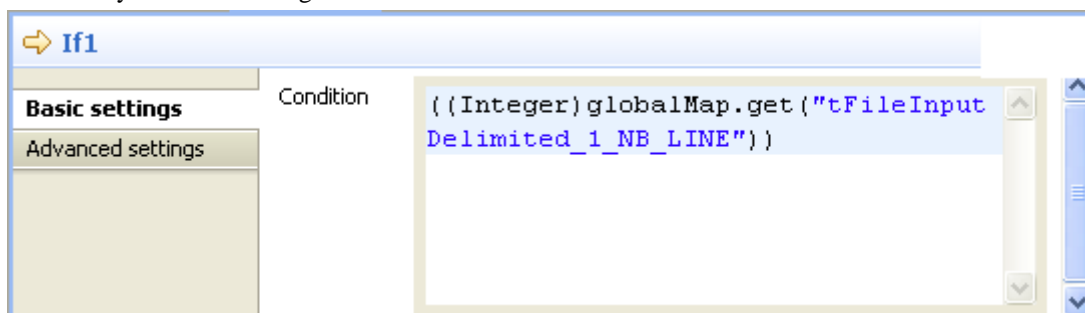


3. Select the **Statistics** check box of the **Run** view to show the real time parallel executions on the design workspace.

4.3.2.3. Trigger connection settings

Run if connection settings

In the **Basic settings** view of a **Run if** connection, you can set the condition to the Subjob in Java. Pressing **Ctrl** + **Space** allows you to access all global and context variables.



4.4. Using the Metadata Manager

Talend Open Studio for ESB is a metadata-driven solution, and can therefore help you ensure the whole Job consistency and quality through a centralized Metadata Manager in the repository.

Talend Open Studio for ESB provides a Metadata Manager that consolidates all project information in a centralized repository shared by all users in the integration processes. This shared repository facilitates collaboration between team members by allowing them to store and share their business models, integration jobs and metadata.

4.4.1. How to centralize the Metadata items

The **Metadata** node in the **Repository** tree view gathers various wizards to help you set up connections to the files, databases, and/or systems you may connect to.

This way you can store centrally the connection details you need to connect to your usual utilities and you can easily reuse them over and over again in all your Job designs without filling them manually every time when configuring the Job components.

From the metadata wizards, you can collect and centralize connection details to various utilities including:

- DB Connection: connection details and table data description (schema) of any type of database and JDBC connections
- File Delimited/Positional/Regex/XML/Excel/Ldif: file access details and description of data from the types of file listed.
- LDAP: access details and data description of an LDAP directory.
- Salesforce: access details and data description of a Salesforce table.
- WSDL: access details and data description of a webservice.
- Generic schema: access details and data description of any sort of sources.

For more information about these Metadata creation procedures, see [Chapter 4, Designing a Job](#).

4.4.2. How to centralize contexts and variables

Depending on the circumstances the Job is being used in, you might want to manage it differently for various execution types (Prod and Test in the example given below). For instance, there might be various testing stages you want to perform and validate before a Job is ready to go live for production use.

Talend Open Studio for ESB offers you the possibility to create multiple context data sets. Furthermore you can either create context data sets on a one-shot basis, from the context tab of a Job or you can centralize the context data sets in the **Contexts** node of the **Repository** tree view in order to reuse them in different Jobs.

A context is characterized by parameters. These parameters are mostly context-sensitive variables which will be added to the list of variables for reuse in the component-specific properties on the **Component** view through the **Ctrl+Space bar** keystrokes.

4.4.2.1. How to use variables in a Job

Variables represent values which change throughout the execution of a program. A global variable is a system variable which can be accessed by any module or function. It retains its value after the function or program using it has completed execution. A context variable is a variable which is defined by the user for a particular context.

You can use an existing global variable or context variable in any component properties field. Press **Ctrl+Space bar** to display the full list of global and context variables used in various predefined Java functions.

<p>Description: Error Message</p> <p>Global variable, property of component tMap [tMap_1].</p> <p>Type: String</p> <p>Availability: After</p> <p>Variable Name: ((String)globalMap.get("tMap_1_ERROR_MESSAGE"))</p>	<p>tFileInputDelimited_2.ERROR_MESSAGE</p> <p>tFileInputDelimited_2.NB_LINE</p> <p>tMap_1.ERROR_MESSAGE</p> <p>tFileOutputDelimited_1.ERROR_MESSAGE</p> <p>tFileOutputDelimited_1.NB_LINE</p> <p>tFileOutputDelimited_2.ERROR_MESSAGE</p> <p>tFileOutputDelimited_2.NB_LINE</p> <p>tFileInputDelimited_3.ERROR_MESSAGE</p> <p>tFileInputDelimited_3.NB_LINE</p> <p>tFlowMeter_1.ERROR_MESSAGE</p> <p>tFlowMeter_2.ERROR_MESSAGE</p>
---	--

The list grows along with new user-defined variables (context variables).

Related topics:

- [Section 4.4.2.4, “How to define variables from the Component view”](#)
- [Section 4.4.2.2, “How to use variables in the Contexts view”](#)

4.4.2.2. How to use variables in the Contexts view

Various ways are at your disposal to create and define variables. You can manage your variables through the **Contexts** view or directly on the **Component** view.

For more information regarding the variable definition directly on the **Component** view, see [Section 4.4.2.4, “How to define variables from the Component view”](#).

The **Contexts** view is positioned on the lower part of the design workspace and is made of three tabs: **Variables**, **Values as tree** and **Values as table**.



If you cannot find the **Contexts** view on the tab system of *Talend Open Studio for ESB*, go to **Window > Show view > Talend**, and select **Contexts**.

Variables tab



The **Variables** tab is part of the **Contexts** tab and shows all of the variables that have been defined for each component in the current Job.

Name	Source	Type	Script code
TST_DBNAME	built-in	String	context.TST_DBNAME
TST_USER	built-in	String	context.TST_USER
TST_PASS	built-in	String	context.TST_PASS
context4Prod	context4Prod	-	-
File	context4Prod	String	context.File
new1	context4Prod	String	context.new1
Newvariable	built-in	String	context.Newvariable

Below the table are navigation buttons: Add (+), Delete (X), Up arrow, Down arrow, and Repository variables (box with 'R'). There is also an 'Original order' checkbox.

From this panel, you can manage your built-in variables:

- Add a parameter line to the table by clicking on [+]
- Edit the **Name** of the new variable and type in the *<Newvariable>* name.
- Delete built-in variables. (Reminder: repository variables are read-only.)
- Import variables from a repository context source, using the **Repository variables** button.
- Display the context variables in their original order. They are sorted automatically by the studio upon creation in the tab view or when imported from the **Repository**. To do this, select the **Original order** check box.

- Reorganize the context variables by selecting the variable of interest and then using the  and  buttons. To do so, you need select the **Original order** check box to activate the two arrow buttons.

To define the actual value of a newly created variable, click the **Value as tree** tab.

You can add as many entries as you need on the **Variables** tab. By default the variable created is of built-in type.

Fields	Description
Name	Name of the variable. You can edit this field, on the condition that the variable is of Built-in type. Repository variables are read-only.
Source	Built-in: The variable is created in this Job and will be used in this Job only. <Repository entry name>: The variable has been defined in a context stored in the repository. The source is thus the actual context group you created in the repository.
Type	Select the type of data being handled. This is required in Java.
Script code	Code corresponding to the variable value. Displayed code will be: <code>context.YourParameterName</code> . This Script code is automatically generated when you define the variable in the Component view.
Comment	Add any useful comment.



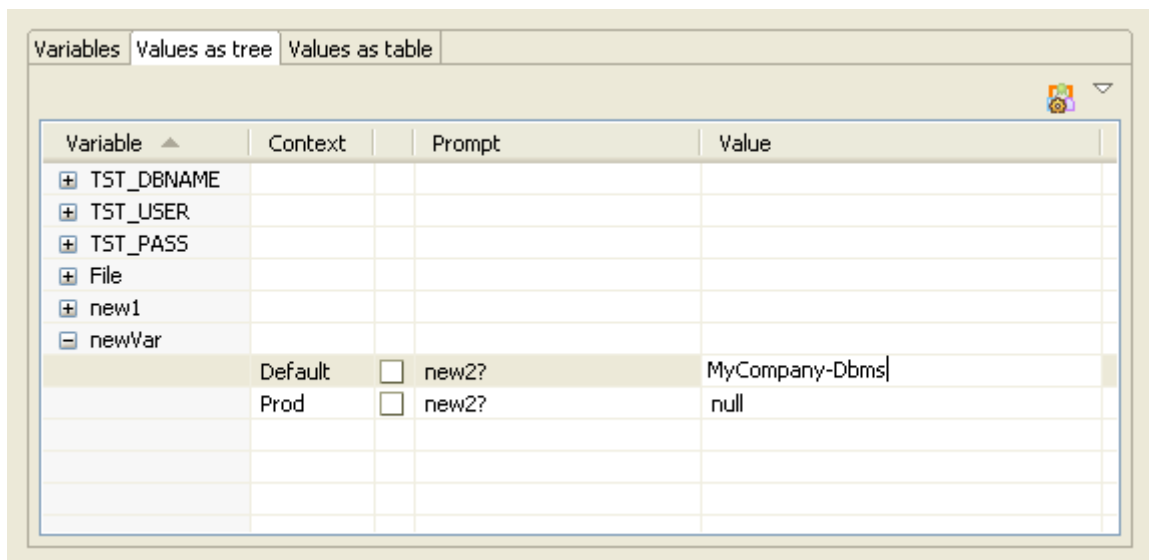
You cannot create contexts from the **Variables** view, but only from the **Values as table** or **as tree** views.

For further information regarding variable definition on the component view, see [Section 4.4.2.4, “How to define variables from the Component view”](#).

For more information about the repository variables, see [Section 4.4.2.5, “How to store contexts in the repository”](#).

Values as tree tab

This tab shows the variables as well as their values in a tree view.




From this view, you can:

- Define the value of a built-in variable directly in the **Value** field. Note that repository variables values are read-only and can only be edited in the relevant repository context.

- Define a question to prompt the user for variable value confirmation at execution time.
- Create or Edit a context name through the top right dedicated button.
- Rearrange the variable/context groupby display.

Fields	Description
Variable	Name of the variables.
Context	Name of the contexts.
Prompt	Select this check box, if you want the variable to be editable in the confirmation dialog box at execution time.
	If you asked for a prompt to popup, fill in this field to define the message to show on the dialog box.
Value	Value for the corresponding variable. Define the value of your built-in variables. Note that repository variables are read-only.

You can manage your contexts from this tab, through the dedicated button  placed on the top right hand side of the **Contexts** view. See [Section 4.4.2.3, “How to configure contexts”](#) for further information regarding the context management.

On the **Values as tree** tab, you can display the values based on the *contexts* or on the *variables* for more clarity.

To change the way the values are displayed on the tree, click the small down arrow button, then click the **group by** option you want.

For more information regarding variable definition, see [Section 4.4.2.4, “How to define variables from the Component view”](#) and [Section 4.4.2.5, “How to store contexts in the repository”](#).

Values as table tab


This **Values as table** tab shows the context and variable settings in the form of a table.

Fields	Description
Name	Name of the variable.
<YourContextName>	Corresponding value for the variable.

You can manage your contexts from this tab, through the **Configure contexts** button placed on the top right hand side of the **Contexts** panel. See [Section 4.4.2.3, “How to configure contexts”](#) for further information regarding the context management.

For more information regarding variable definition, see [Section 4.4.2.4, “How to define variables from the Component view”](#) and [Section 4.4.2.5, “How to store contexts in the repository”](#).

4.4.2.3. How to configure contexts

You can only manage your contexts from the **Values as table** or **Values as tree** tabs. A dedicated button  shows up on the top right hand side of the **Contexts** view.

Click the **Configure Contexts...** icon to open the management dialog box.



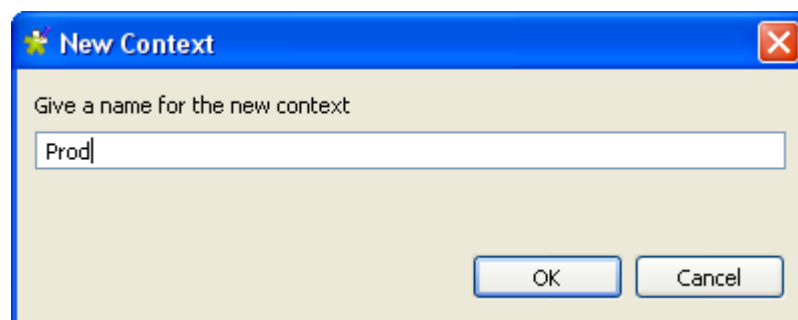
The default context cannot be removed, therefore the **Remove** button is unavailable. To make it editable, select another context on the list.

Creating a context

Based on the default context you set, you can create as many contexts as you need.

To create a new context:

1. Click **New** in the [Configure Contexts] dialog box.
2. Type in a name for the new context.



3. Click **OK** to validate the creation.

When you create a new context, the entire default context legacy is copied over to the new context. You hence only need to edit the relevant fields on the **Value as tree** tab to customize the context according to your use.

The drop-down list **Default Context** shows all the contexts you created.

You can switch default context by simply selecting the new default context on the **Default Context** list on the **Variables** tab of the **Contexts** view.

Note that the Default (or last) context can never be removed. There should always be a context to run the Job, be this context called Default or any other name.

Renaming or editing a context

To change the name of an existing context:

1. Click **Edit** on the **[Configure contexts]** dialog box and enter the new context name in the dialog box showing up.
2. Click **OK** to validate the change.

To carry out changes on the actual values of the context variables, go to the **Values as tree** or **Values as table** tabs. For more information about these tabs, see [Section 4.4.2.2, “How to use variables in the Contexts view”](#).

4.4.2.4. How to define variables from the Component view

Various ways are at your disposal to create and define context variables. You can mostly manage your variables from the **Contexts** view, but you can also create them directly on the **Component** view.

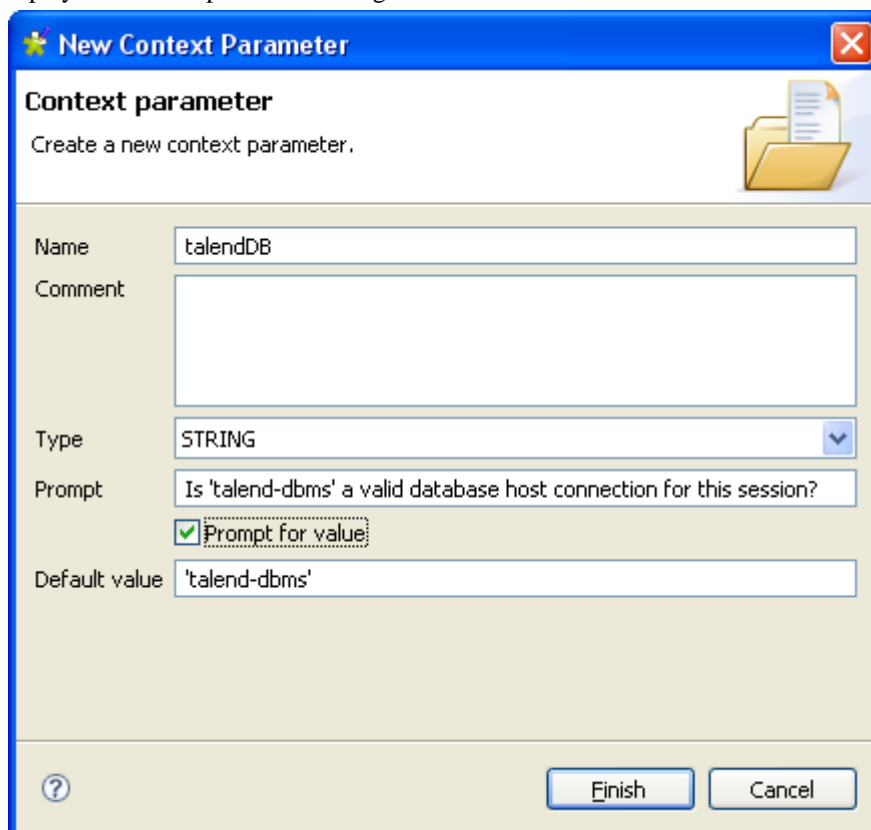
For more information related to the variable definition through the **Contexts** view, see [Section 4.4.2.2, “How to use variables in the Contexts view”](#).

For more information regarding the variable definition in the repository, see [Section 4.4.2.5, “How to store contexts in the repository”](#).

Context variables creation

The quickest way to create context variables on the spot is to use the **F5** key:

1. On the relevant **Component** view, place your cursor on the field that you want to parameterize.
2. Press **F5** to display the context parameter dialog box:



3. Give a **Name** to this new variable, fill in the **Comment** area and choose the **Type**.
4. Enter a **Prompt** to be displayed to confirm the use of this variable in the current Job execution (generally used for test purpose only). And select the **Prompt for value** check box to display the field as editable value.
5. If you filled in a value already in the corresponding properties field, this value is displayed in the **Default value** field. Else, type in the default value you want to use for one context.
6. Click **Finish** to validate.
7. Go to the **Contexts** view tab. Notice that the context variables tab lists the newly created variables.

The newly created variables are listed in the **Contexts** view.



The variable name should follow some typing rules and should not contain any forbidden characters, such as space character.

The variable created this way is automatically stored in all existing contexts, but you can subsequently change the value independently in each context.

For more information on how to create or edit a context, see [Section 4.4.2.3, “How to configure contexts”](#).

StoreSQLQuery

StoreSQLQuery is a user-defined variable and is mainly dedicated to debugging.

StoreSQLQuery is different from other context variables in the fact that its main purpose is to be used as parameter of the specific global variable called **Query**. It allows you to dynamically feed the global query variable.

The global variable **Query**, is available on the proposals list (**Ctrl+Space bar**) for some DB input components.

For further details on **StoreSQLQuery** settings, see the *Talend Open Studio Components Reference Guide*, and in particular the scenarios of the **tDBInput** component.

4.4.2.5. How to store contexts in the repository

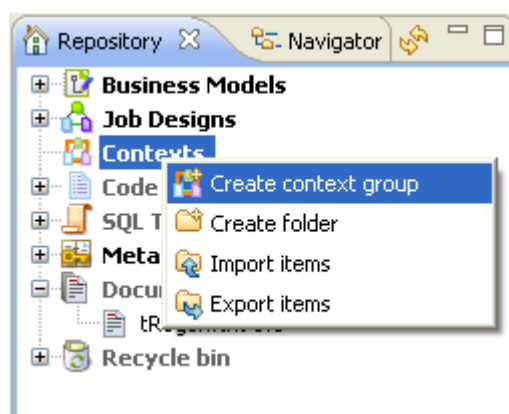
You can store centrally all contexts if you need to reuse them across various Jobs.

How to create a context group

To create a context group, proceed as follows:

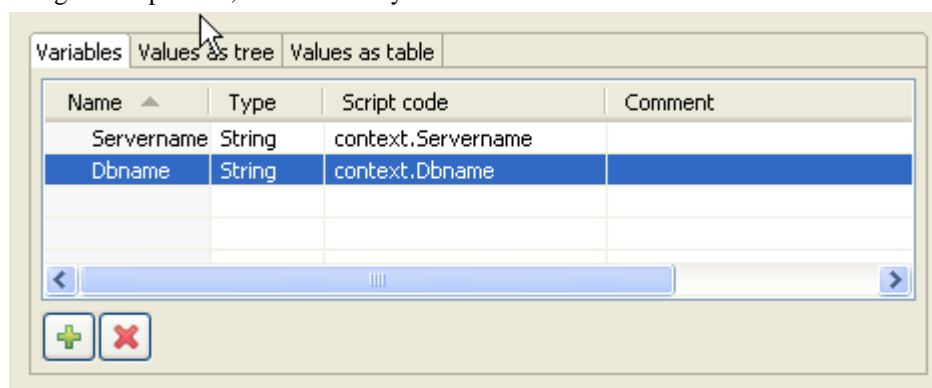
Procedure 4.1. Create the context group and add required information

1. Right-click the **Contexts** node in the **Repository** tree view and select **Create new context group** from the contextual menu.



A 2-step wizard appears to help you define the various contexts and context parameters, which you will be able to select in the **Contexts** view of the design workspace.

2. In Step 1 of 2, type in a name for the context group to be created, and add any general information such as a description if required.
3. Click **Next** to go to Step 2 of 2, which allows you to define the various contexts and variables that you need.



Procedure 4.2. Define the default context's variable set to be used as basis for other contexts

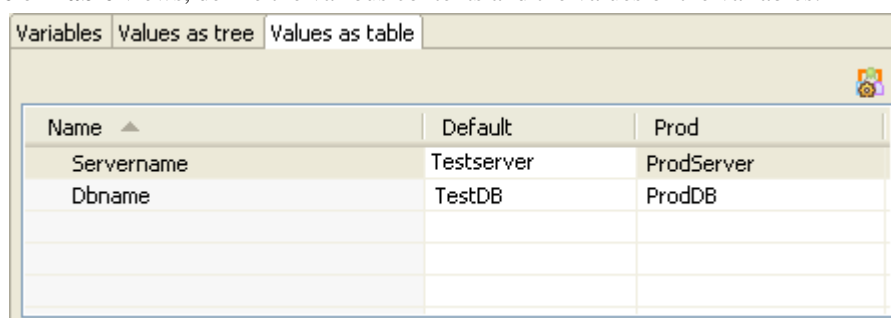
1. On the **Variables** tab, click the [+] button to add as many new variable lines as needed and define the name of the variables.

In this example, we define the variables that can be used in the **Name** field of the **Component** view.

2. Select the type of the variable from the **Type** list.

The **Script code** varies according to the type of variable you selected, and will be used in the generated code. The screen shot above shows the Java code produced.

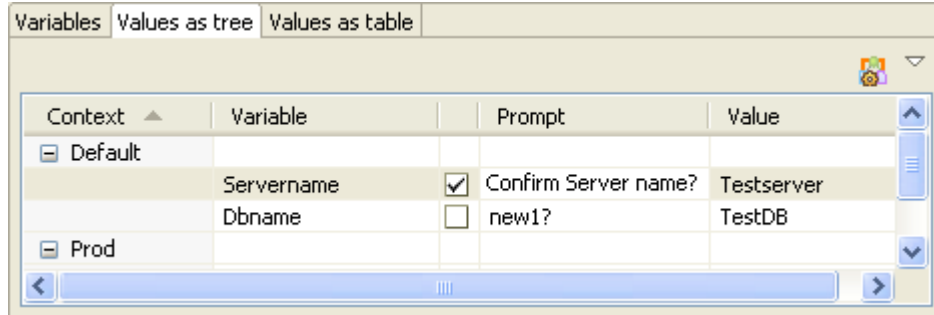
3. On the **Tree** or **Table** views, define the various contexts and the values of the variables.



First, define the values for the default (first) context variables, then create a new context that will be based on the variables values that you just set.

For more information about how to create a new context, see [Section 4.4.2.3, “How to configure contexts”](#).

4. On the **Values as tree** tab, add a prompt if you want the variable to be editable in a confirmation dialog box at execution time.



To add a prompt message, select the facing check box, then type in the message you want to display at execution time.

Once you created and adapted as many context sets as you want, click **Finish** to validate. The group of contexts thus displays under the **Contexts** node in the **Repository** tree view.


How to create a context from a Metadata

When creating a Metadata (through the File or DB metadata wizards), you have the possibility to Export the metadata connection information as a Context.

For more information about this feature, see [Section 9.17, “Exporting Metadata as context”](#).

4.4.2.6. How to apply context variables to a Job from the repository

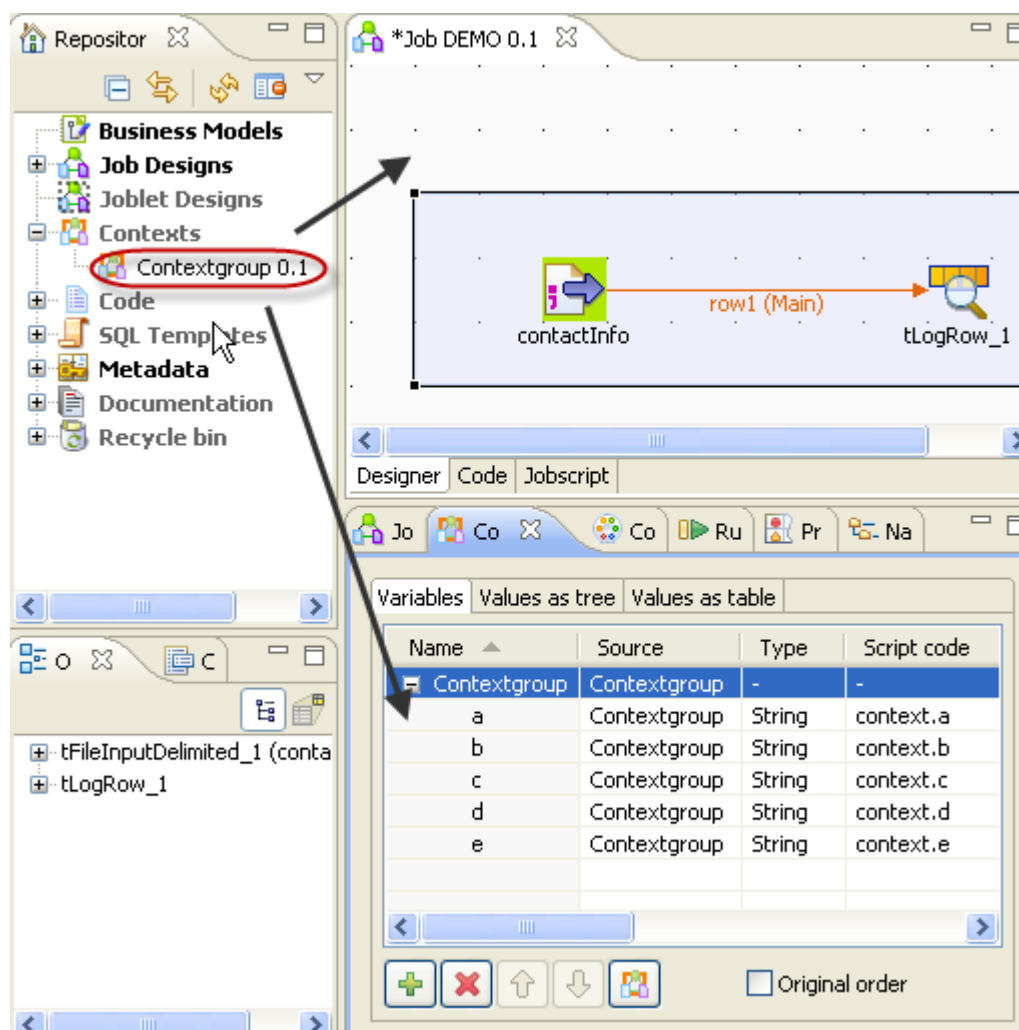
Once a context group is created and stored in the **Repository**, there are two ways of applying it to a Job:

1. Drop a context group. This way, the group is applied as a whole.
2. Use the context icon button . This way, the variables of a context group can be applied separately.

How to drop a context group onto a Job

To drop a context group onto a Job, proceed as follows:

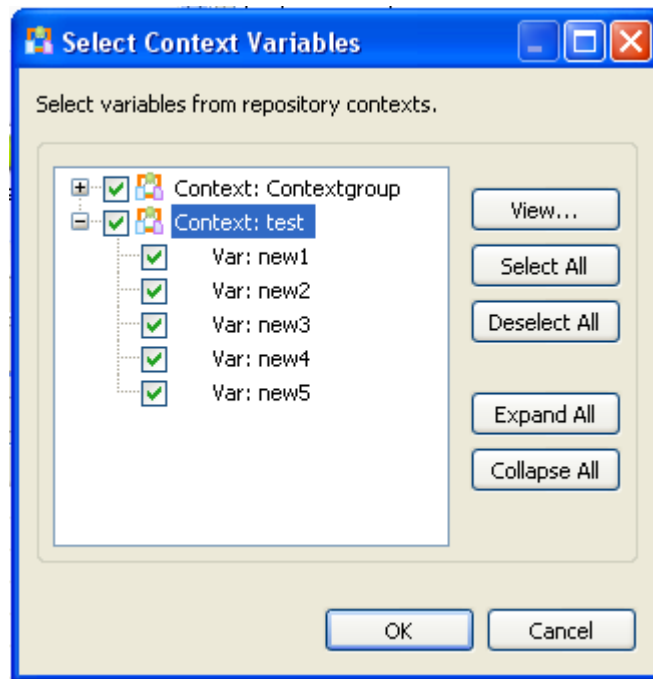
1. Double-click the Job to which a context group is to be added.
2. Once the Job is opened, drop the context group of your choice either onto the Job workspace or onto the **Contexts** view beneath the workspace.



How to use the context icon button

To use the context icon button to apply context variables to a Job, proceed as follows:

1. Double-click the Job to which a context variable is to be added.
2. Once the Job is opened in the workspace, click the **Contexts** view beneath the workspace to open it.
3. At the bottom of the **Contexts** view, click the button to open the wizard to select the context variables to be applied.



4. In the wizard, select the context variables you need to apply or clear those you do not need to.

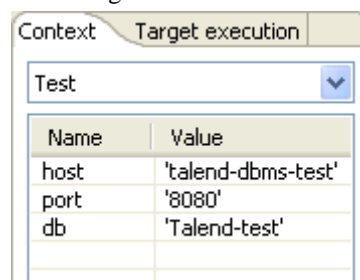


The context variables that have been applied are automatically selected and cannot be cleared.

5. Click **OK** to apply the selected context variables to the Job.

4.4.2.7. How to run a Job in a selected context

You can select the context you want the Job design to be executed in.



Click the **Run Job** tab, and in the **Context** area, select the relevant context among the various ones you created.

If you did not create any context, only the **Default** context shows on the list.

All the context variables you created for the selected context display, along with their respective value, in a table underneath. If you clear the **Prompt** check box next to some variables, you will get a dialog box allowing you to change the variable value for this Job execution only.

To make a change permanent in a variable value, you need to change it on the Context view if your variable is of type built-in or in the Context group of the repository.

Related topics:

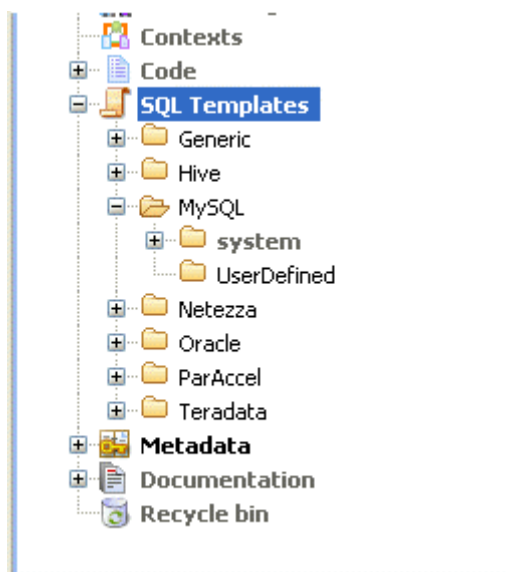
- [Section 4.4.2.2, “How to use variables in the Contexts view”](#)

- [Section 4.4.2.5, “How to store contexts in the repository”](#)

4.4.3. How to use the SQL Templates

Talend Open Studio for ESB allows you to benefit from using some system SQL templates since many query structures are standardized with common approaches.

Talend Open Studio for ESB lists system SQL templates under the **SQL Templates** node in the **Repository** tree view. There, you can find several standardized SQL templates including Generic, Hive, MySQL, Oracle, and Teradata.



In each of the above categories, you can create your own user-defined SQL templates using the SQL templates wizard and thus centralize them in the repository for reuse.

For more information about the use of SQL templates in *Talend Open Studio for ESB*, see [Chapter 4, Designing a Job](#).

For more information about how to create a user-defined SQL template and use it in a Job context, see the scenario of **tMysqlTableList** component in the *Talend Open Studio Components* Reference Guide.

4.5. Handling Jobs: advanced subjects

The sections below give detail information about various advanced configuration situations of a data integration Job including handling multiple input and output flows, using SQL queries, using external components in the Job, scheduling a task to run your Job.

4.5.1. How to map data flows

The most common way to handle multiple input and output flows in your Job including transformations and data re-routing is to use the **tMap** component.

For more information about the principles of using this component, see [Chapter 4, Designing a Job](#).

For examples of Jobs using this component, see **tMap** in the *Talend Open Studio Components* Reference Guide.

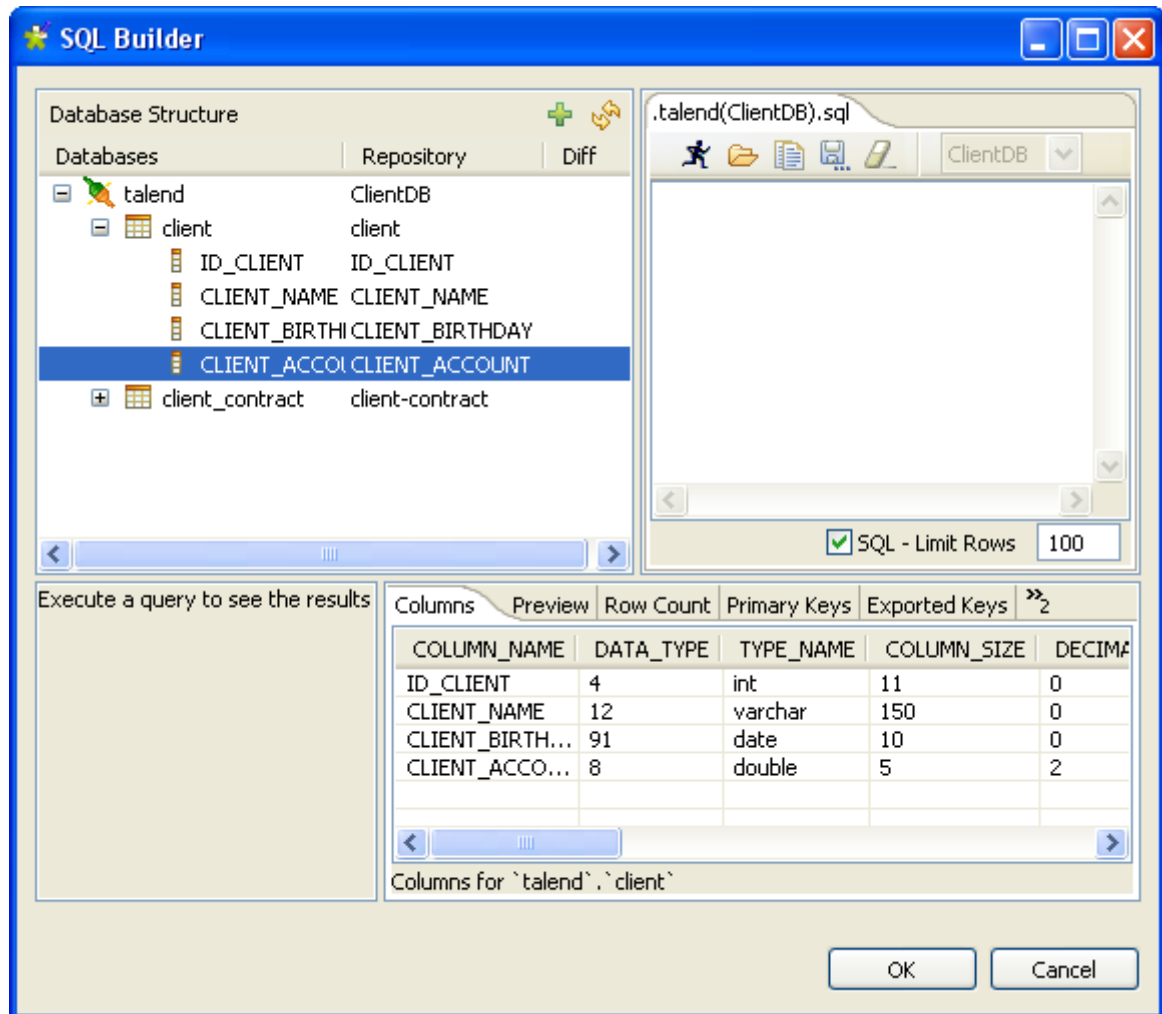
4.5.2. How to create queries using the SQLBuilder

SQLBuilder helps you build your SQL queries and monitor the changes between DB tables and metadata tables. This editor is available in all DBInput and DBSQLRow components (specific or generic).

You can build a query using the SQLbuilder whether your database table schema is stored in the **Repository** tree view or built-in directly in the Job.

Fill in the DB connection details and select the appropriate repository entry if you defined it.

Remove the default query statement in the **Query** field of the **Basic settings** view of the **Component** panel. Then click the [...] button to open the **[SQL Builder]** editor.



The **[SQL Builder]** editor is made of the following panels:

- Database structure,
- Query editor made of editor and designer tabs,
- Query execution view,
- Schema view.

The Database structure shows the tables for which a schema was defined either in the repository database entry or in your built-in connection.

The schema view, in the bottom right corner of the editor, shows the column description.

4.5.2.1. How to compare database structures

On the **Database Structure** panel, you can see all tables stored in the DB connection metadata entry in the **Repository** tree view, or in case of built-in schema, the tables of the database itself.



The connection to the database, in case of built-in schema or in case of a refreshing operation of a repository schema might take quite some time.

Click the refresh icon to display the differences between the DB metadata tables and the actual DB tables.

Databases	Repository	Diff
talend	ClientDB	
client	client	
CLIENT_ACCOUNT	CLIENT_ACCOUNT	
CLIENT_BIRTHDAY	CLIENT_BIRTHDAY	
ID_CLIENT	ID_CLIENT	
CLIENT_NAME	CLIENT_NAME	
client_contract	client-contract	
CONTRACT_TYPE	CONTRACT_TYPE	
CONTRACT_VALUE	CONTRACT_VALUE	
ID_CLIENT	ID_CLIENT	
ID_CONTRACT	ID_CONTRACT	
	FOO	
axeltable1	axeltable1	
ID_MONTH	ID_MONTH	
ID_TYPE	ID_TYPE	
MONTH	MONTH	
ID_USER		
sales		

The **Diff** icons point out that the table contains differences or gaps. Expand the table node to show the exact column containing the differences.

The red highlight shows that the content of the column contains differences or that the column is missing from the actual database table.

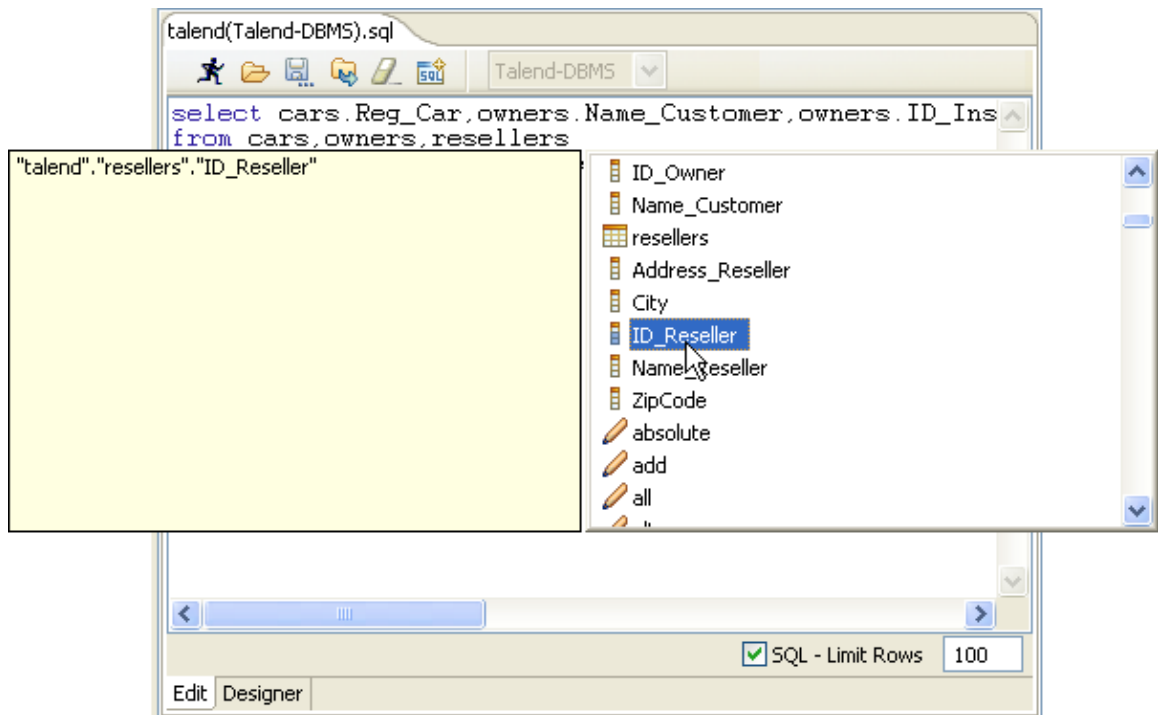
The blue highlight shows that the column is missing from the table stored in **Repository > Metadata**.

4.5.2.2. How to build a query

The **[SQL Builder]** editor is a multiple-tab editor that allows you to write or graphically design as many queries as you want.

To create a new query, complete the following:

1. Right-click the table or on the table column and select **Generate Select Statement** on the pop-up list.
2. Click the empty tab showing by default and type in your SQL query or press **Ctrl+Space** to access the autocompletion list. The tooltip bubble shows the whole path to the table or table section you want to search in.



Alternatively, the graphical query **Designer** allows you to handle tables easily and have real-time generation of the corresponding query in the **Edit** tab.

3. Click the **Designer** tab to switch from the manual **Edit** mode to the graphical mode.

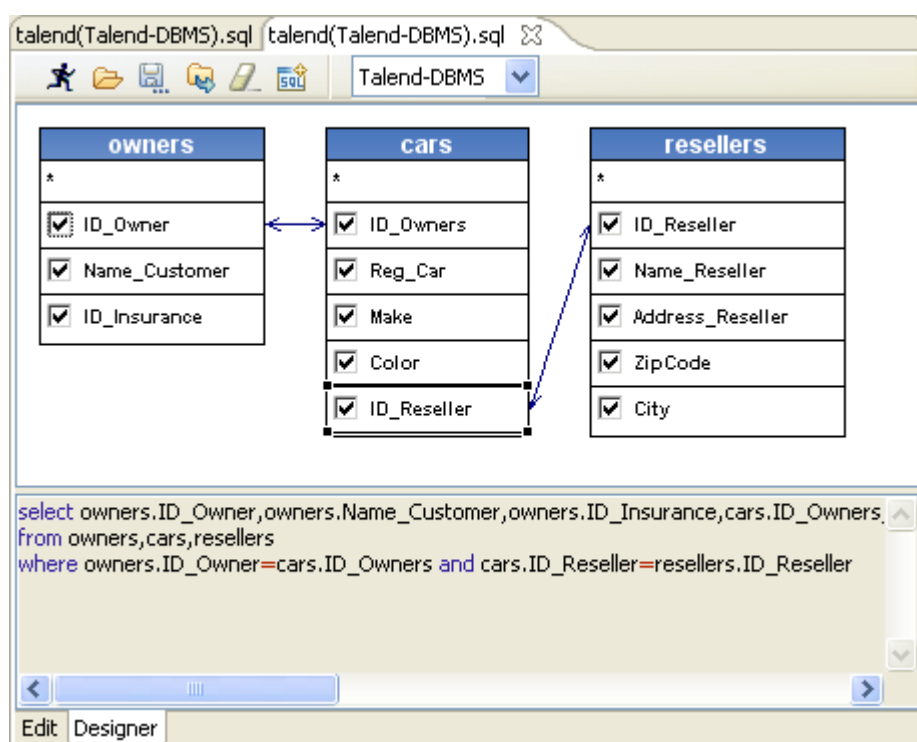


You may get a message while switching from one view to the other as some SQL statements cannot be interpreted graphically.

4. If you selected a table, all columns are selected by default. Clear the check box facing the relevant columns to exclude them from the selection.
5. Add more tables in a simple right-click. On the **Designer** view, right-click and select **Add tables** in the pop-up list then select the relevant table to be added.

If joins between these tables already exist, these joins are automatically set up graphically in the editor.


You can also create a join between tables very easily. Right-click the first table columns to be linked and select **Equal** on the pop-up list, to join it with the relevant field of the second table.



The SQL statement corresponding to your graphical handlings is also displayed on the viewer part of the editor or click the **Edit** tab to switch back to the manual **Edit** mode.



In the **Designer** view, you cannot include graphically filter criteria. You need to add these in the **Edit** view.

6. Once your query is complete, execute it by clicking the  icon on the toolbar.

The toolbar of the query editor allows you to access quickly usual commands such as: execute, open, save and clear.

The results of the active query are displayed on the **Results** view in the lower left corner.

4.5.2.3. How to store a query in the repository

To be able to retrieve and reuse queries, we recommend you to store them in the repository.

In the [SQL Builder] editor, click the icon on the toolbar to bind the query with the DB connection and schema in case these are also stored in the repository.

The query can then be accessed from the **Database structure** view, on the left-hand side of the editor.

4.5.3. How to download/upload Talend Community components

Talend Open Studio for ESB enables you to access a list of all community components in **Talend Exchange** that are compatible with your current version of *Talend Open Studio for ESB*. You can then download and install these

components to use them later in the Job designs you carry out in the Studio. From *Talend Open Studio for ESB*, you can also upload components you have created to **Talend Exchange** to share with other community users.

A click on the **Exchange** link on the toolbar of *Talend Open Studio for ESB* opens the **Exchange** tab view on the design workspace, where you can find lists of:

- components available in **Talend Exchange** for you to download and install,
- components you downloaded and installed in previous versions of *Talend Open Studio for ESB* but not installed yet in your current Studio, and
- components you have created and uploaded to **Talend Exchange** to share with other **Talend** Community users.




- Before you can download community components or upload your own components to the community, you need to sign in to **Talend Exchange** from your Studio first. If you did not sign in to **Talend Exchange** when launching the Studio, you still have a chance to sign in from the **Talend Exchange** preferences settings page. For more information, see [Section 2.5.3, “Exchange preferences”](#).
- The community components available for download are not validated by **Talend**. This explains why you may encounter component loading errors sometimes when trying to install certain community components, why an installed community component may have a different name in the **Palette** than in the **Exchange** tab view, and why you may not be able to find a component in the **Palette** after it is seemingly installed successfully.

4.5.3.1. How to install community components from Talend Exchange

To install community components from **Talend Exchange** to the **Palette** of your current *Talend Open Studio for ESB*:

1. Click the **Exchange** link on the toolbar of *Talend Open Studio for ESB* to open the **Exchange** tab view on the design workspace.

Available Extensions	<input type="text"/> 				
Downloaded Extensions					
My Extensions	Extension Name	Version	Rating	Author	View
	Facebook Application Insights Component	0.1	★★★★★	saburo	view/download
	tFileOutputDelimitedEx	1.0	★★★★★	Alezis	view/download
	tScriptRules	1.0	★★★★★	walkerca	view/download
	tWorldBank components Demo	0.2	★★★★★	saburo	view/download
	pUpdateMailOffer	V02	★★★★★	PayZen	view/download
	pCreateMailOffer	V02	★★★★★	PayZen	view/download
	tDBFOutput	1.1	★★★★★	BiSi	view/download
	tDBFInput	1.1	★★★★★	BiSi	view/download
	pCreatePayment	V06	★★★★★	PayZen	view/download

2. In the **Available Extensions** view, if needed, enter a full component name or part of it in the text field and click the fresh button to find quickly the component you are interested in.
3. Click the **view/download** link for the component of interest to display the component download page.



tPDFToText

Version 1.1
2011-05-17

Convert a PDF to text file. It's possible to extract a delimited area.



★★★★☆

Install

User Reviews

[write a review](#)

- ★★★★☆ bien it's good
- ★★★★★ It works on 4.2.2 You can use it, it works on 4.2.2!
- ★☆☆☆☆ tPDFToText This does not seem compatible with TOS 4.2.2r63143. I have installed it on TOS and it does not generate an output file.

4. View the information about the component, including component description and review comments from community users, or write your own review comments and/or rate the component if you want. For more information on reviewing and rating a community component, see [Section 4.5.3.3, “How to review and rate a community component”](#).

If needed, click the left arrow button to return to the component list page.

5. Click the **Install** button in the right part of the component download page to start the download and installation process.

A progress indicator appears to show the completion percentage of the download and installation process. Upon successful installation of the component, the **Downloaded Extensions** view opens and displays the status of the component, which is **Installed**.

Available Extensions	Extension Name	Downloaded Version	Download Date	Install/Update
Downloaded Extensions				
My Extensions	tDBFInput	1.1	2011-11-17	Install
	tDBFOutput	1.1	2011-10-31	Install
	bcLogbackConfig	1.2	2011-10-31	Install
	bcLogbackCatch	1.3	2011-10-31	Install
	tLog4j	1.4	2011-11-17	Install
	tFileOutputDelimitedEx	1.0	2011-11-17	Install
	null	null	2011-11-17	Install
	tScriptRules	1.0	2011-11-17	Install
	BRules	1.1	2011-11-17	Install
	tPDFToText	1.1	2011-11-18	Installed

4.5.3.2. How to reinstall or update community components

From the **Exchange** tab view, you can reinstall components you already downloaded and installed in your previous version of *Talend Open Studio for ESB* or install the updated version of *Talend Open Studio for ESB* or components in your current Studio.



By default, while you are connected to **Talend Exchange**, a dialog box appears to notify you whenever an update to an installed community component is available. If you often check for community component updates and you do not want that dialog box to appear again, you can turn it off in **Talend Exchange** preferences settings. For more information, see [Section 2.5.3, “Exchange preferences”](#).

To reinstall a community component you already downloaded or update an installed one, complete the following:

1. From the **Exchange** tab view, click **Downloaded Extensions** to display the list of components you have already downloaded from **Talend Exchange**.

In the **Downloaded Extensions** view, the components you have installed in your previous version of *Talend Open Studio for ESB* but not in your current Studio have an **Install** link in the **Install/Update** column, and those with updates available in **Talend Exchange** have an **Update** link.

2. Click the **Install** or **Update** link for the component of interest to start the installation process.

A progress indicator appears to show the completion percentage of the installation process. Upon successful installation, the **Downloaded Extensions** view displays the status of the component, which is **Installed**.

4.5.3.3. How to review and rate a community component

To review and rate a community component:

1. From the **Available Extensions** view, click the **view/download** link for the component you want to review or rate to open the community component download page.
2. On the component download page, click the **write a review** link to open the **[Review the component]** dialog box.

- Fill in the required information, including a title and a review comment, click one of the five stars to rate the component, and click **Submit Review** to submit your review to the **Talend Exchange** server.

Upon validation by the **Talend Exchange** moderator, your review is published on **Talend Exchange** and displayed in the **User Review** area of the component download page.

4.5.3.4. How to upload a component you created to Talend Exchange

You can create your own components for use in your Jobs in *Talend Open Studio for ESB* and upload them to **Talend Exchange** to share with other **Talend** Community users. For information on how to create your own components and deploy them in *Talend Open Studio for ESB*, see [Section 2.5.2, “External or User components”](#).

To upload a component you created to **Talend Exchange**, complete the following:

- From the **Exchange** tab view, click **My Extensions** to open the **My Extensions** view.

Available Extensions	Add New Extension			
Downloaded Extensions				
My Extensions	Extension Name	Version	Upload Date	Operation

- Click the **Add New Extension** link in the upper right part of the view to open the component upload page.

Extension Title: Initial Version:

Compatibility: ☐ All versions ☐ Version and older: ☒ Versions and newer: ☐ All versions except: ☐ Only these versions:

Description:

File:

- Complete the required information, including the component title, initial version, Studio compatibility information, and component description, fill in or browse to the path to the source package in the **File** field, and click the **Upload Extension** button.

Upon successful upload, the component is listed in the **My Extensions** view, where you can update, modify and delete any component you have uploaded to **Talend Exchange**.

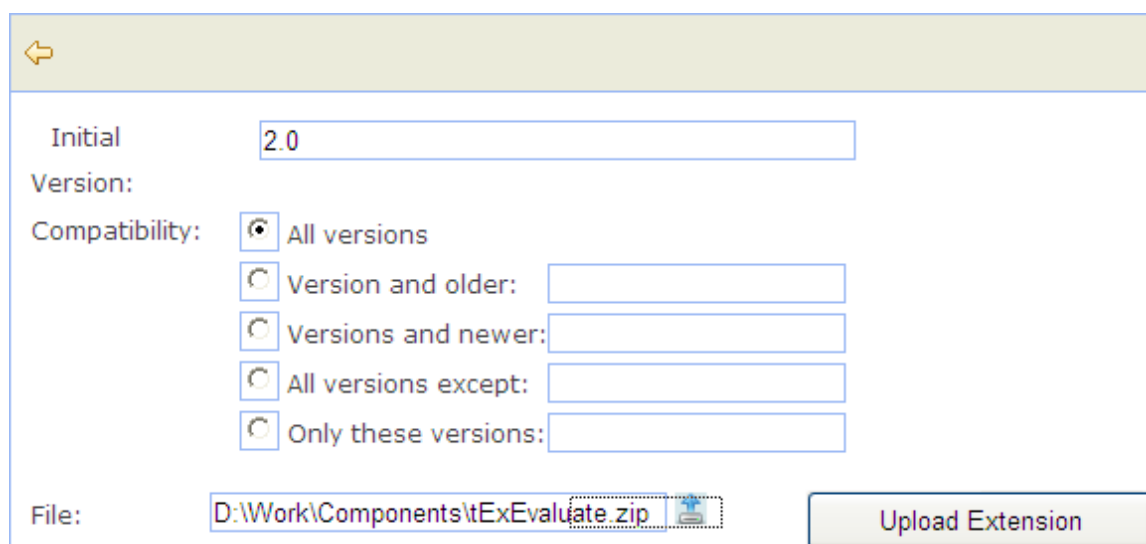
Available Extensions	Add New Extension			
Downloaded Extensions				
My Extensions	Extension Name	Version	Upload Date	Operation
	tExEvaluate	1.0	2011-11-18	

4.5.3.5. How to manage components you uploaded to Talend Exchange

From the **Exchange** tab view, you can manage components you have uploaded to **Talend Exchange**, including updating component version, modifying component information, and deleting components from **Talend Exchange**.

To update the version of a component, complete the following:

- From the **My Extensions** view, click the icon in the **Operation** column for the component your want to update to open the component update page.



Initial Version: 2.0

Version:

Compatibility:

- ☒ All versions
- ☐ Version and older:
- ☐ Versions and newer:
- ☐ All versions except:
- ☐ Only these versions:


File: D:\Work\Components\tExEvaluate.zip

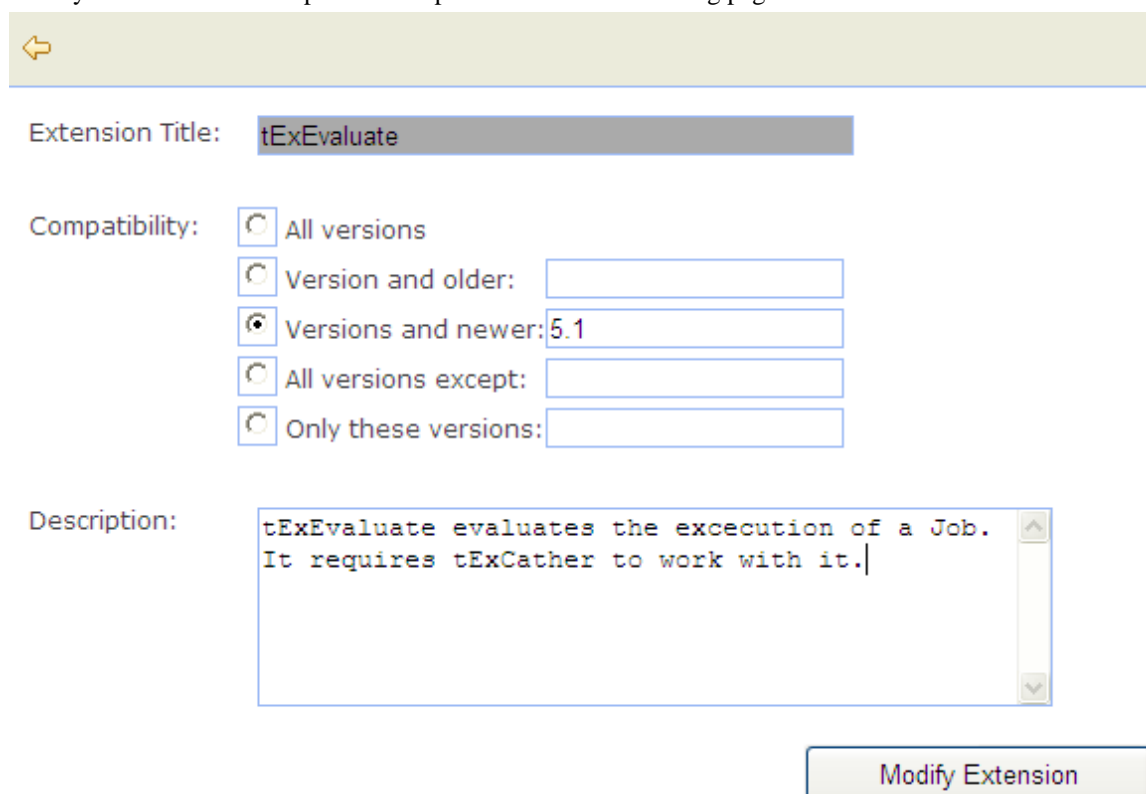
Upload Extension

2. Fill in the initial version and Studio compatibility information, fill in or browse to the path to the source package in the **File** field, and click the **Update Extension** button.

Upon successful upload of the updated component, the component is replaced with the new version on **Talend Exchange** and the **My Extension** view displays the component's new version and update date.

To modify the information of a component uploaded to **Talend Exchange**, complete the following:

1. From the **My Extensions** view, click the  icon in the **Operation** column for the component your want to modify information for to open the component information editing page.



Extension Title: tExEvaluate

Compatibility:

- ☐ All versions
- ☐ Version and older:
- ☒ Versions and newer: 5.1
- ☐ All versions except:
- ☐ Only these versions:

Description: tExEvaluate evaluates the execution of a Job. It requires tExCather to work with it.

Modify Extension

2. Complete the Studio compatibility information and component description, and click the **Modify Extension** button to update the component information to **Talend Exchange**.

To delete a component you have uploaded to **Talend Exchange**, click  icon for the component from the **My Extensions** view. The component is then removed from **Talend Exchange** and is no longer displayed on the component list in the **My Extensions** view.

4.5.4. How to install external modules

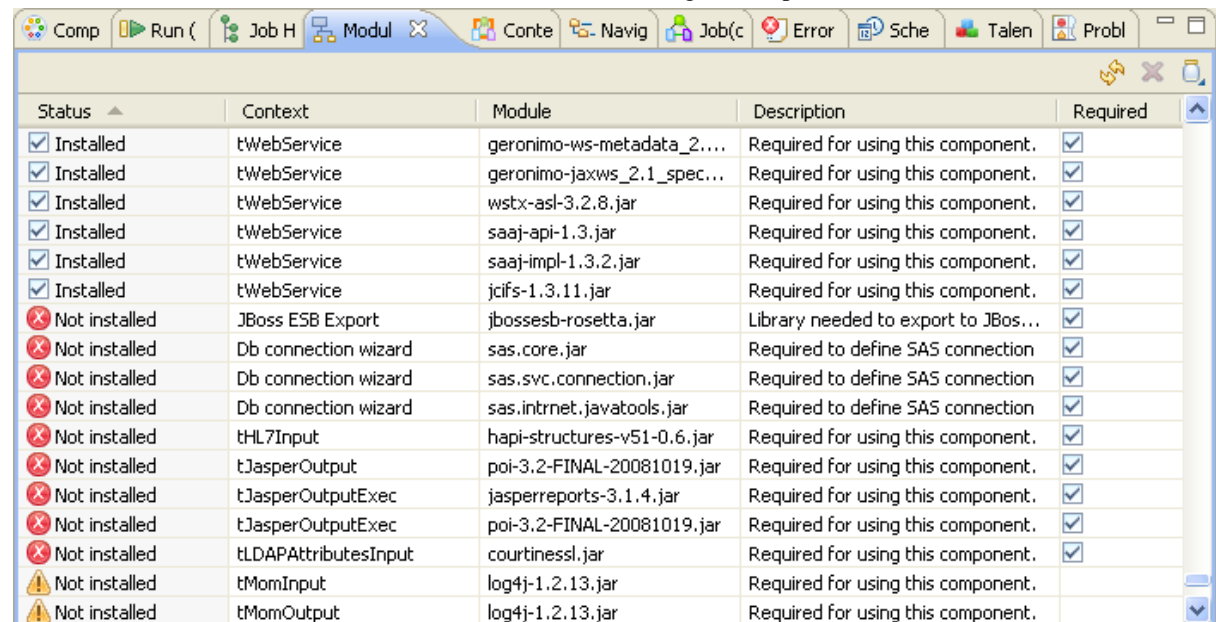
The use of some components in *Talend Open Studio for ESB* requires specific modules to be installed.

The **Modules** view lists all modules necessary to use the components embedded in the Studio. Few of these modules do not exist by default and thus you must install them in order to be able to run smoothly the Jobs that use such components.






If the **Modules** tab does not show on the tab area of your design workspace, go to **Window > Show View... > Talend** and then select **Modules** from the list.

To access the **Modules** view, click the **Modules** tab in the design workspace.




Status	Context	Module	Description	Required
Installed	tWebService	geronimo-ws-metadata_2...	Required for using this component.	<input checked="" type="checkbox"/>
Installed	tWebService	geronimo-jaxws_2.1_spec...	Required for using this component.	<input checked="" type="checkbox"/>
Installed	tWebService	wstx-asl-3.2.8.jar	Required for using this component.	<input checked="" type="checkbox"/>
Installed	tWebService	saaj-api-1.3.jar	Required for using this component.	<input checked="" type="checkbox"/>
Installed	tWebService	saaj-impl-1.3.2.jar	Required for using this component.	<input checked="" type="checkbox"/>
Installed	tWebService	jcifs-1.3.11.jar	Required for using this component.	<input checked="" type="checkbox"/>
Not installed	JBoss ESB Export	jbosseb-rosetta.jar	Library needed to export to JBoss...	<input checked="" type="checkbox"/>
Not installed	Db connection wizard	sas.core.jar	Required to define SAS connection	<input checked="" type="checkbox"/>
Not installed	Db connection wizard	sas.svc.connection.jar	Required to define SAS connection	<input checked="" type="checkbox"/>
Not installed	Db connection wizard	sas.intrnet.javatools.jar	Required to define SAS connection	<input checked="" type="checkbox"/>
Not installed	tHL7Input	hapi-structures-v51-0.6.jar	Required for using this component.	<input checked="" type="checkbox"/>
Not installed	tJasperOutput	poi-3.2-FINAL-20081019.jar	Required for using this component.	<input checked="" type="checkbox"/>
Not installed	tJasperOutputExec	jasperreports-3.1.4.jar	Required for using this component.	<input checked="" type="checkbox"/>
Not installed	tJasperOutputExec	poi-3.2-FINAL-20081019.jar	Required for using this component.	<input checked="" type="checkbox"/>
Not installed	tLDAPAttributesInput	courtinessl.jar	Required for using this component.	<input checked="" type="checkbox"/>
Not installed	tMomInput	log4j-1.2.13.jar	Required for using this component.	<input type="checkbox"/>
Not installed	tMomOutput	log4j-1.2.13.jar	Required for using this component.	<input type="checkbox"/>

The table below describes the information presented in the **Modules** view.

Column	Description
Status	points out if a module is installed or not installed on your system. The  icon indicates that the module is not necessarily required for the corresponding component listed in the Context column. The  icon indicates that the module is absolutely required for the corresponding component.
Context	lists the name of Talend component using the module. If this column is empty, the module is then required for the general use of <i>Talend Open Studio for ESB</i> .  This column lists any external libraries added to the routines you create and save in the Studio library folder. For more information, see Section 10.4.3, "How to edit user routine libraries" .
Module	lists the module exact name.
Description	explains why the module/library is required.

Column	Description
Required	the selected check box indicates that the module is required.

To install any missing module, complete the following:

1. In the **Modules** view, click the  icon in the upper right corner of the view.
The **[Open]** dialog box of your operating system appears.
2. Browse to the module you want to install, select it and then click **Open** on the dialog box.
The dialog box closes and the selected module is installed in the library folder of the current Studio.
You can now use the component dependent on this module in any of your Job designs.

4.5.5. How to launch a Job periodically

The **Scheduler** view in *Talend Open Studio for ESB* helps you to schedule a task that will launch periodically a Job via a task scheduling (crontab) program.

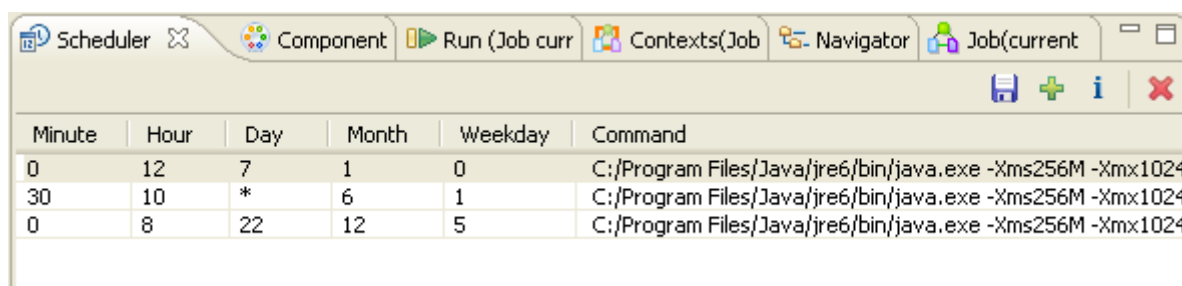
Through the **Scheduler** view, you can generate a crontab file that holds cron-compatible entries (the data required to launch the Job). These entries will allow you to launch periodically a Job via the crontab program.

This Job launching feature is based on the *crontab* command, found in Unix and Unix-like operating systems. It can be also installed on any Windows system.

To access the **Scheduler** view, click the **Scheduler** tab in the design workspace.




If the **Scheduler** tab does not display on the tab system of your design workspace, go to **Window > Show View... > Talend**, and then select **Scheduler** from the list.



Minute	Hour	Day	Month	Weekday	Command
0	12	7	1	0	C:/Program Files/Java/jre6/bin/java.exe -Xms256M -Xmx1024
30	10	*	6	1	C:/Program Files/Java/jre6/bin/java.exe -Xms256M -Xmx1024
0	8	22	12	5	C:/Program Files/Java/jre6/bin/java.exe -Xms256M -Xmx1024

This view is empty if you have not scheduled any task to run a Job. Otherwise, it lists the parameters of all the scheduled tasks.

The procedure below explains how to schedule a task in the **Scheduler** view to run a specific Job periodically and then generate the crontab file that will hold all the data required to launch the selected Job. It also points out how to use the generated file with the *crontab* command in Unix or a task scheduling program in Windows.

1. Click the  icon in the upper right corner of the **Scheduler** view.

The **[Open Scheduler]** dialog box displays.

2. From the **Project** list, select the project that holds the Job you want to launch periodically.
3. Click the three-dot button next to the **Job** field and select the Job you want to launch periodically.
4. From the **Context** list, if more than one exists, select the desired context in which to run the Job.
5. Set the time and date details necessary to schedule the task.

The command that will be used to launch the selected Job is generated automatically and attached to the defined task.

6. Click **Add this entry** to validate your task and close the dialog box.

The parameters of the scheduled task are listed in the **Scheduler** view.



7. Click the  icon in the upper right corner of the **Scheduler** view to generate a crontab file that will hold all the data required to start the selected Job.

The **[Save as]** dialog box displays.

8. Browse to set the path to the crontab file you are generating, enter a name for the crontab file in the **File name** field, and then click **Save** to close the dialog box.

The crontab file corresponding to the selected task is generated and stored locally in the defined path.

9. In Unix, paste the content of the crontab file into the crontab configuration of your Unix system; in Windows, install a task scheduling program that will use the generated crontab file to launch the selected Job.

You can use the  icon to delete any of the listed tasks and the  icon to edit the parameters of any of the listed tasks.

4.5.6. How to use the tPrejob and tPostjob components

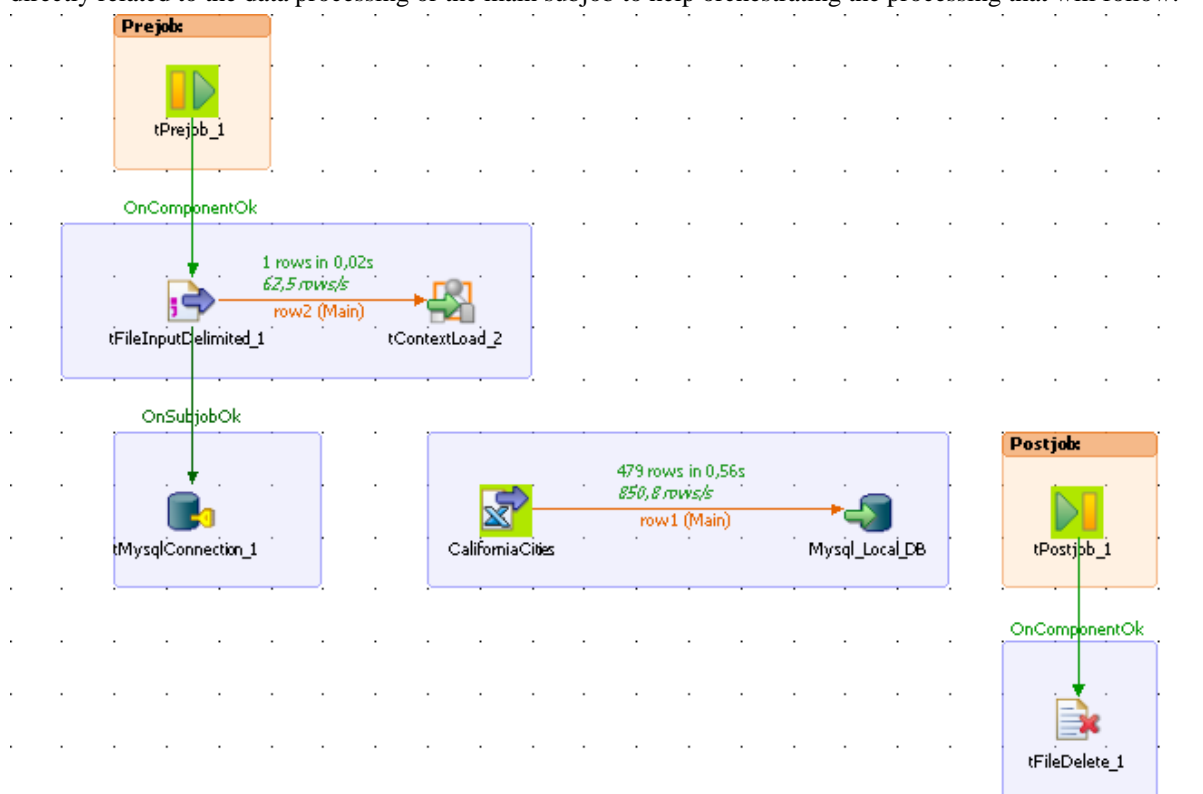
The prejob and postjob parts display as components on the design workspace, and are thus available in the **Palette** of components. To use these **tPrejob** and **tPostjob** components, simply drop them to the design workspace as you would do with any other components. An orange square shows the pre and post-job parts which are different types of subjobs.

However note that their use slightly differs from typical components, in the way that these two components do not actually process data nor flows but are meant to help you make your Job design clearer.



As **tPrejob** and **tPostjob** are not meant to take part in any data processing, they can not be part of multi thread execution. The tasks included in the **tPrejob** and **tPostjob** are done once for all following subjobs, whether the subjobs are executed in sequence or in parallel.

Connect to these **tPrejob** and **tPostjob** components, all the components that perform organizing tasks that are not directly related to the data processing or the main subjob to help orchestrating the processing that will follow.



Tasks that require a **tPrejob** component to be used include for example:

- loading context information required for the subjob execution,
- opening a database connection,
- making sure that a file exists.

Many more tasks that are collateral to your Job and might be damaging the overall readability of your Job may as well need a prejob component.

Tasks that require a **tPostjob** component to be used include, for example:

- clearing a folder or deleting a file,
- any tasks to be carried out even though the preceding subjob(s) failed.

4.6. Handling Jobs: miscellaneous subjects

The sections below give detail information about various subjects related to the management of a data integration Job including defining the start component, handling errors, using the **tPrejob** and **tPostjob** components and searching for jobs that use specific components.

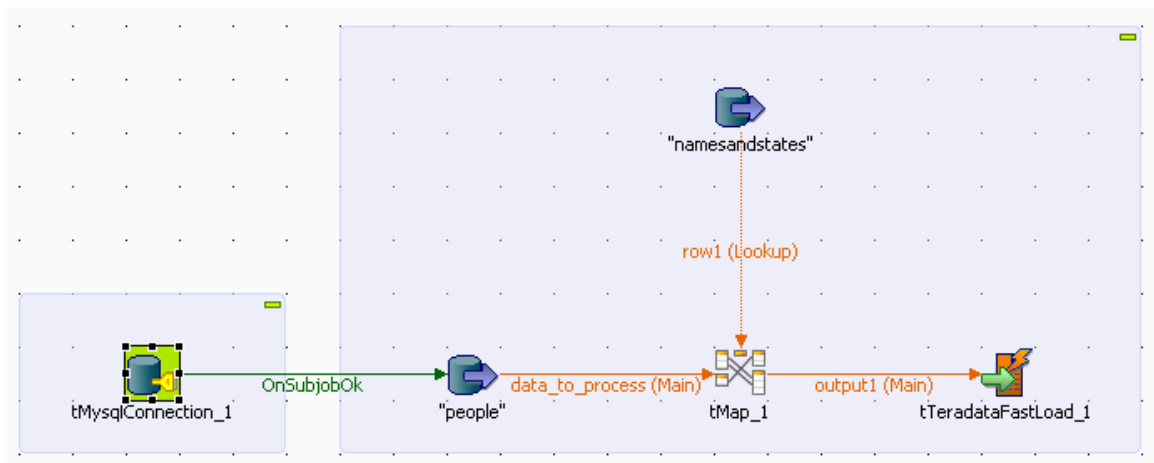
4.6.1. How to share a database connection

If you have various Jobs using the same database connection, you can now factorize the connection by using the **Use or Register a shared connection** option.

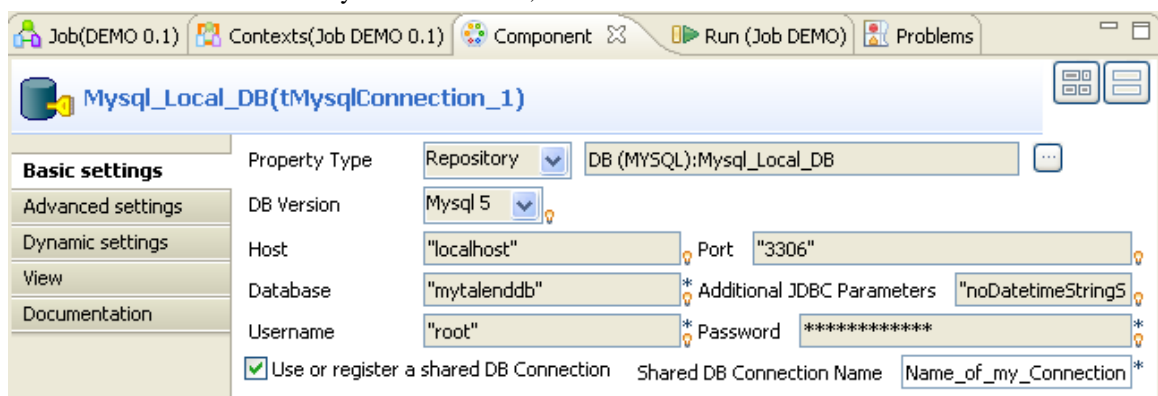
This option box has been added to all Connection components in order to reduce the number of connection opening and closing.

Assuming that you have two related Jobs (a parent Job and a child Job) that both need to connect to your remote MySQL database, then perform the operations below:

1. Drag and drop a **tMySQLConnection** (assuming that you work with a MySQL database)
2. Connect it to the first component of your parent Job



3. On the Connection Component view, tick the box **Use or Register a shared connection**.
4. Give a name to the connection you want to share, in the **Shared DB Connection Name** field.



You are now able to re-use the connection in your Child Job (and any other Job that requires a connection to the same database).

5. Simply follow the same steps again and make sure you use the same name in the **Shared DB Connection Name** field.

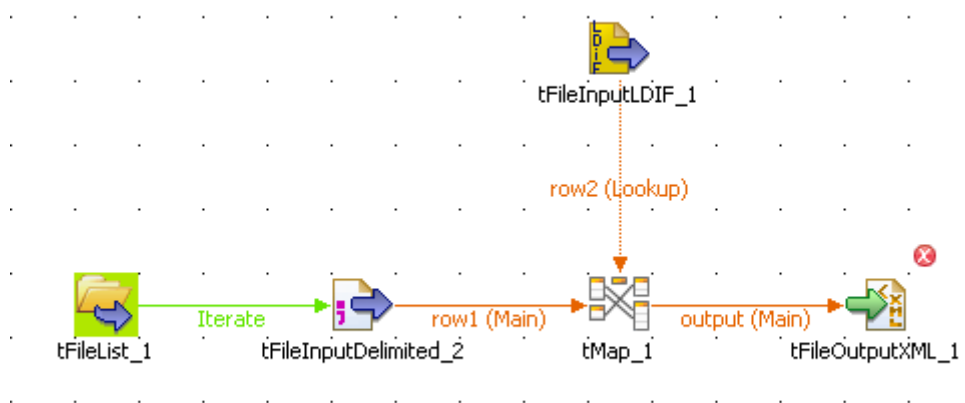
For more information about how to use the Connection components, see *Talend Open Studio Components Reference Guide*.

4.6.2. How to define the Start component

The **Start** component is the trigger of a Job. There can be several Start components per Job design if there are several flows running in parallel. But for one flow and its *connected* subflows, only one component can be the Start component.

Drop a component to the design workspace, all possible start components take a distinctive bright green background color. Notice that most of the components, can be **Start** components.

Only components which do not make sense to trigger a flow, will not be proposed as Start components, such as the **tMap** component for example.



To distinguish which component is to be the **Start** component of your Job, identify the main flow and the secondary flows of your Job.

- The main flow should be the one connecting a component to the next component using a Row type link. The Start component is then automatically set on the first component of the main flow (icon with green background).
- The secondary flows are also connected using a Row-type link which is then called Lookup row on the design workspace to distinguish it from the main flow. This Lookup flow is used to enrich the main flow with more data.

Be aware that you can change the Start component hence the main flow by changing a main Row into a Lookup Row, simply through a right-click the row to be changed.

Related topics:

- [Section 4.2.4, “How to connect components together”](#)
- [Section 7.1, “Activating/Deactivating a Job or a sub-job or a Route”](#)

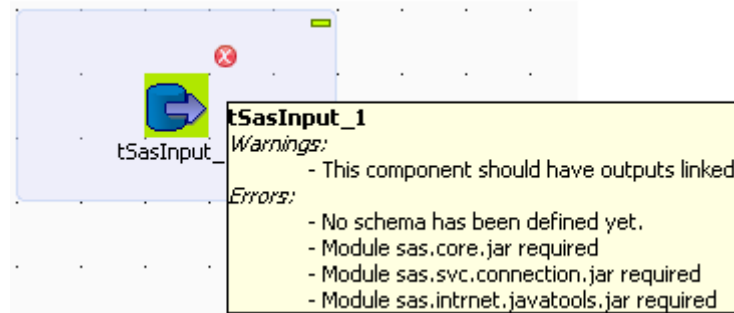
4.6.3. How to handle error icons on components or Jobs

When the properties of a component are not properly defined and contain one or several errors that can prevent the Job code to compile properly, error icons will automatically show next to the component icon on the design workspace and the Job name in the **Repository** tree view.

4.6.3.1. Warnings and error icons on components

When a component is not properly defined or if the link to the next component does not exist yet, a red checked circle or a warning sign is docked at the component icon.

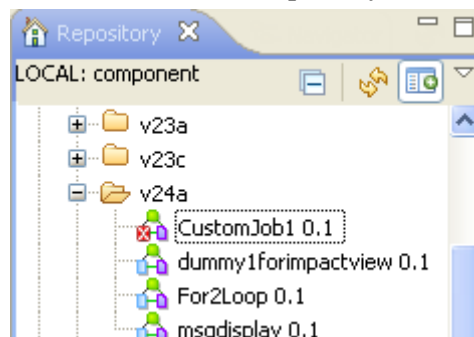
Mouse over the component, to display the tooltip messages or warnings along with the label. This context-sensitive help informs you about any missing data or component status.



When the tooltip messages of a component indicate that a module is required, you must install this module for this component using the **Module** view. This view is hidden by default. For further information about how to install external modules using this view, see [Section 4.5.4, “How to install external modules”](#).

4.6.3.2. Error icons on Jobs

When the component settings contain one or several errors that can prevent the Job code to compile properly, an icon will automatically show next to the Job name in the **Repository** tree view.



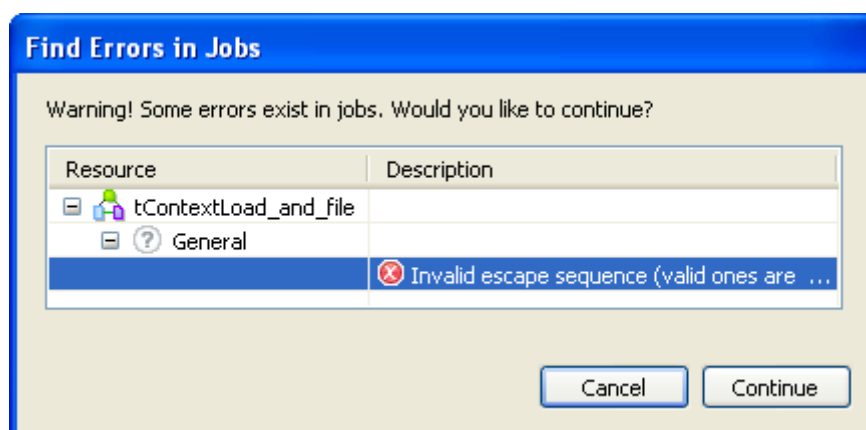
The error icon displays as well on the tab next to the Job name when you open the Job on the design workspace.

The compilation or code generation does only take place when carrying out one of the following operations:

- opening a Job,
- clicking on the **Code Viewer** tab,
- executing a Job (clicking on **Run Job**),
- saving the Job.

Hence, the red error icon will only show then.

When you execute the Job, a warning dialog box opens to list the source and description of any error in the current Job.

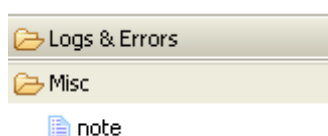


Click **Cancel** to stop your Job execution or click **Continue** to continue it.

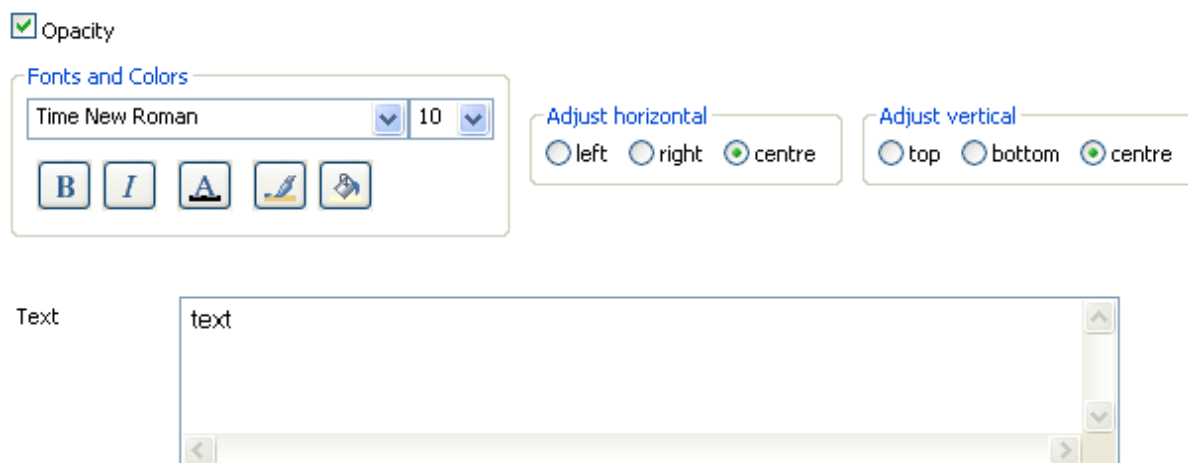
For information on errors on components, see [Section 4.6.3.1, “Warnings and error icons on components”](#).

4.6.4. How to add notes to a Job design

In the **Palette**, click the **Misc** family and then drop the **Note** element to the design workspace to add a text comment to a particular component or to the whole Job.



You can change the note format. To do so, select the note you want to format and click the **Basic setting** tab of the **Component** view.



Select the **Opacity** check box to display the background color. By default, this box is selected when you drop a note on the design workspace. If you clear this box, the background becomes transparent.

You can select options from the **Fonts and Colors** list to change the font style, size, color, and so on as well as the background and border color of your note.

You can select the **Adjust horizontal** and **Adjust vertical** boxes to define the vertical and horizontal alignment of the text of your note.

The content of the **Text** field is the text displayed on your note.

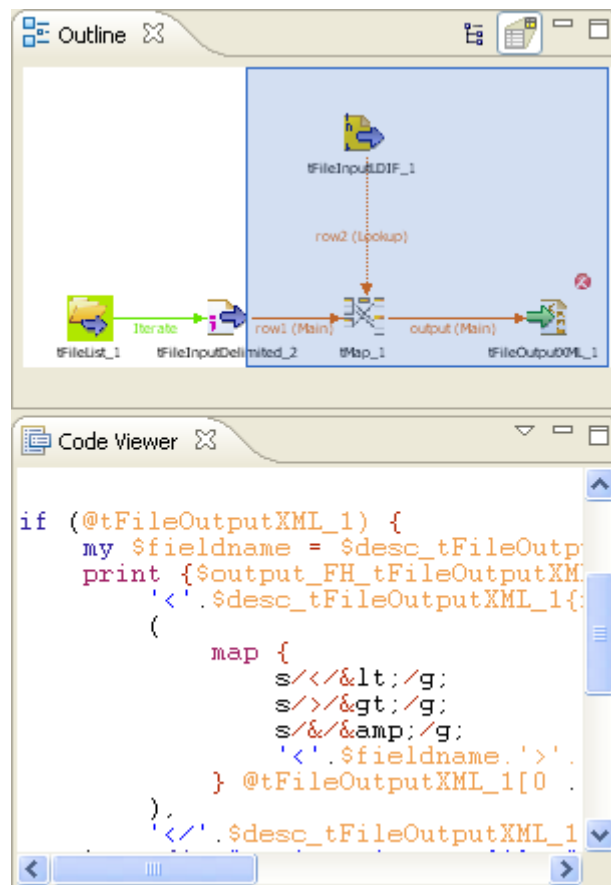
4.6.5. How to display the code or the outline of your Job

This panel is located below the **Repository** tree view. It displays detailed information about the open Job or Business Model in the design workspace.

The Information panel is composed of two tabs, **Outline** and **Code Viewer**, which provide information regarding the displayed diagram (either Job or Business Model).

4.6.5.1. Outline

The **Outline** tab offers a quick view of the business model or the open Job on the design workspace and also a tree view of all used elements in the Job or Business Model. As the design workspace, like any other window area can be resized upon your needs, the **Outline** view is convenient to check out where about on your design workspace, you are located.



This graphical representation of the diagram highlights in a blue rectangle the diagram part showing in the design workspace.

Click the blue-highlighted view and hold down the mouse button. Then, move the rectangle over the Job.

The view in the design workspace moves accordingly.

The **Outline** view can also be displaying a folder tree view of components in use in the current diagram. Expand the node of a component, to show the list of variables available for this component.

To switch from the graphical outline view to the tree view, click either icon docked at the top right of the panel.

4.6.5.2. Code viewer

The **Code viewer** tab provides lines of code generated for the selected component, behind the active Job design view, as well the run menu including Start, Body and End elements.

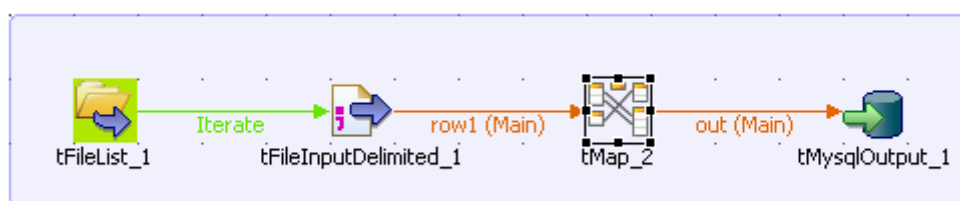


This view only concerns the Job design code, as no code is generated from Business Models.

Using a graphical colored code view, the tab shows the code of the component selected in the design workspace. This is a partial view of the primary Code tab docked at the bottom of the design workspace, which shows the code generated for the whole Job.

4.6.6. How to manage the subjob display

A subjob is graphically defined by a blue square gathering all connected components that belong to this subjob. Each individual component can be considered as a subjob when they are not yet connected to one another.



This blue highlight helps you easily distinguish one subjob from another.

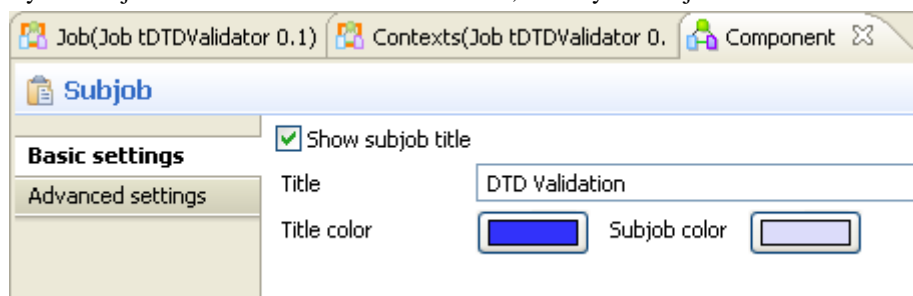


A Job can be made of one single subjob. An orange square shows the prejob and postjob parts which are different types of subjobs.

For more information about prejob and postjob, see [Section 4.5.6, “How to use the tPrejob and tPostjob components”](#).

4.6.6.1. How to format subjobs

You can modify the subjob color and its title color. To do so, select your subjob and click the **Component** view.



In the **Basic setting** view, select the **Show subjob title** check box if you want to add a title to your subjob, then fill in a title.

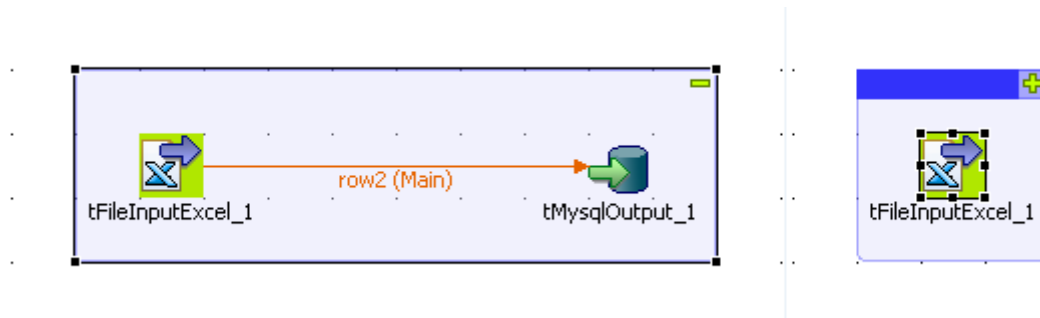
To modify the title color and the subjob color:

1. In the **Basic settings** view, click the **Title color/Subjob color** button to display the **[Colors]** dialog box.

- Set your colors as desired. By default, the title color is blue and the subjob color is transparent blue.

4.6.6.2. How to collapse the subjobs

If your Job is made of numerous subjobs, you can collapse them to improve the readability of the whole Job. The minus (-) and plus (+) signs on the top right-hand corner of the subjob allow you to collapse and restore the complete subjob.



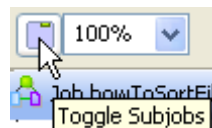
Click the minus sign (-) to collapse the subjob. When reduced, only the first component of the subjob is displayed.

Click the plus sign (+) to restore your subjob.

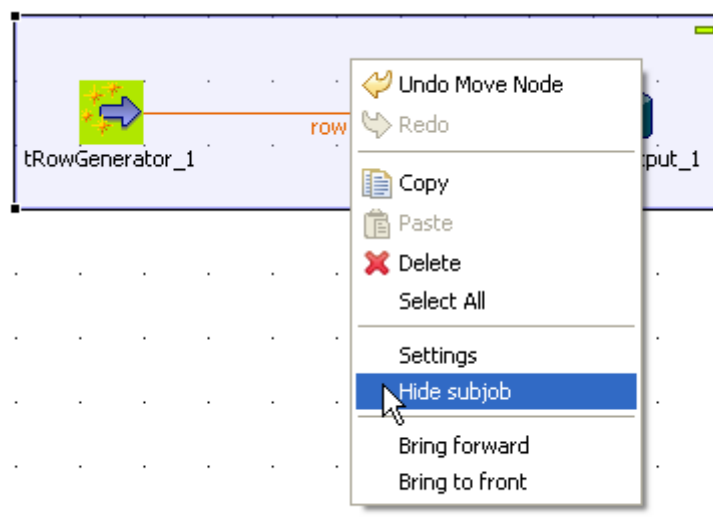
4.6.6.3. How to remove the subjob background color

If you do not want your subjobs to be highlighted, you can remove the background color on all or specific subjobs.

To remove the background color of all your subjobs, click the **Toggle Subjobs** icon on the toolbar of *Talend Open Studio for ESB*.



To remove the background color of a specific subjob, right-click the subjob and select the **Hide subjob** option on the pop-up menu.



4.6.7. How to define options on the Job view

On the **Job** view located on the bottom part of the design workspace, you can define Job's optional functions. This view is made of two tabs: **Stats & Logs** tab and **Extra** tab.

The **Stats & Logs** tab allows you to automate the use of **Stats & Logs** features and the Context loading feature. For more information, see [Section 4.6.7.1, "How to automate the use of statistics & logs"](#).

The **Extra** tab lists various options you can set to automate some features such as the context parameters use, in the **Implicit Context Loading** area. For more information, see [Section 4.6.7.2, "How to use the features in the Extra tab"](#).

4.6.7.1. How to automate the use of statistics & logs

If you have a great need of log, statistics and other measurement of your data flows, you are facing the issue of having too many log-related components loading your Job Designs. You can automate the use of **tFlowMeterCatcher**, **tStatCatcher**, **tLogCatcher** component functionalities without using the components in your Job via the **Stats & Logs** tab.

For more information regarding the Log component, see the *Talend Open Studio Components* Reference Guide.

The **Stats & Logs** panel is located on the **Job** tab underneath the design workspace and prevents your Jobs Designs to be overloaded by components.




This setting supersedes the log-related components with a general log configuration.

To set the **Stats & Logs** properties:

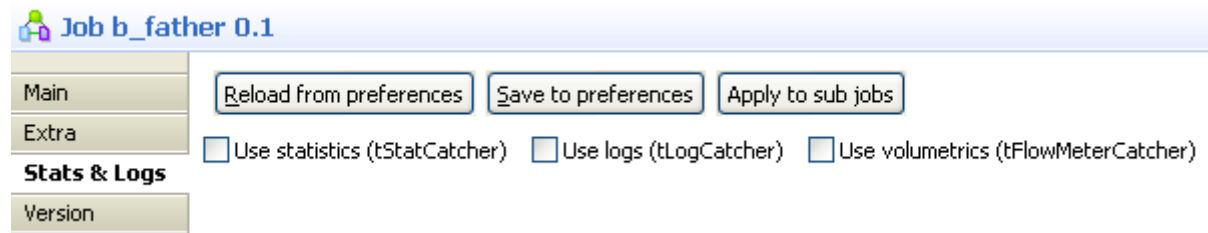
1. Click the **Job** tab.
2. Select the **Stats & Logs** panel to display the configuration view.

3. Set the relevant details depending on the output you prefer (console, file or database).
4. Select the relevant **Catch** check box according to your needs.



You can save the settings into your Project Settings by clicking the **Save to project settings** button. This way, you can access such settings via **File > Edit project settings > Job settings > Stats & Logs** or via the  button on the toolbar.

When you use **Stats & Logs** functions in your Job, you can apply them to all its subjobs.



To do so, click the **Apply to subjobs** button in the **Stats & Logs** panel of the **Job** view and the selected stats & logs functions of the main Job will be selected for all of its subjobs.

4.6.7.2. How to use the features in the Extra tab

The **Extra** tab offers some optional function parameters.

- Select the **Multithread execution** check box to allow two Job executions to start at the same time.
- Set the **Implicit tContextLoad** option parameters to avoid using the **tContextLoad** component on your Job and automate the use of context parameters.

Choose between **File** and **Database** as source of your context parameters and set manually the file or database access.

Set notifications (error/warning/info) for unexpected behaviors linked to context parameter setting.

- When you fill in **Implicit tContextLoad** manually, you can store these parameters in your project by clicking the **Save to project settings** button, and thus reuse these parameters for other components in different Jobs.
- Select the **Use Project Settings** check box to recuperate the context parameters you have already defined in the **Project Settings** view.

The **Implicit tContextLoad** option becomes available and all fields are filled in automatically.

For more information about context parameters, see [Section 2.6.6, “Context settings”](#).

- Click **Reload from project settings** to update the context parameters list with the latest context parameters from the project settings.

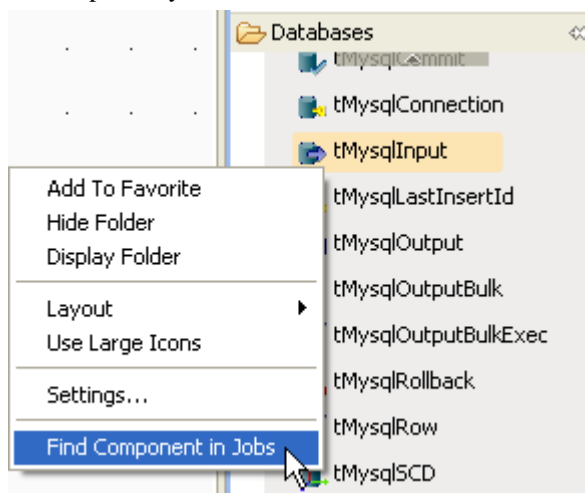
4.6.8. How to find components in Jobs



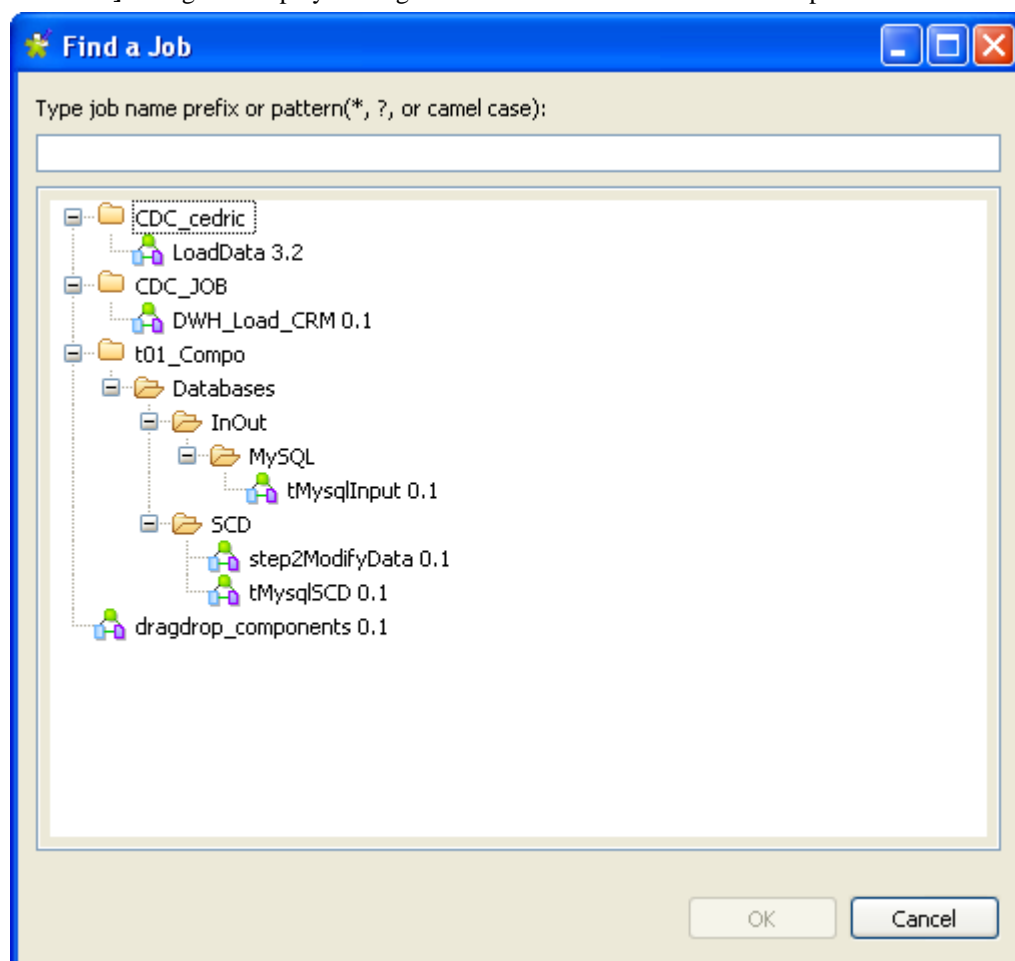
You should open one Job at least in the Studio to display the **Palette** to the right of the design workspace and thus start the search.

From the **Palette**, you can search for all the Jobs that use the selected component. To do so:

1. In the **Palette**, right-click the component you want to look for and select **Find Component in Jobs**.



A progress indicator displays to show the percentage of the search operation that has been completed then the **[Find a Job]** dialog box displays listing all the Jobs that use the selected component.



2. From the list of Jobs, click the desired Job and then click **OK** to open it on the design workspace.

4.6.9. How to set default values in the schema of an component

You can set default values in the schema of certain components to replace null values retrieved from the data source.



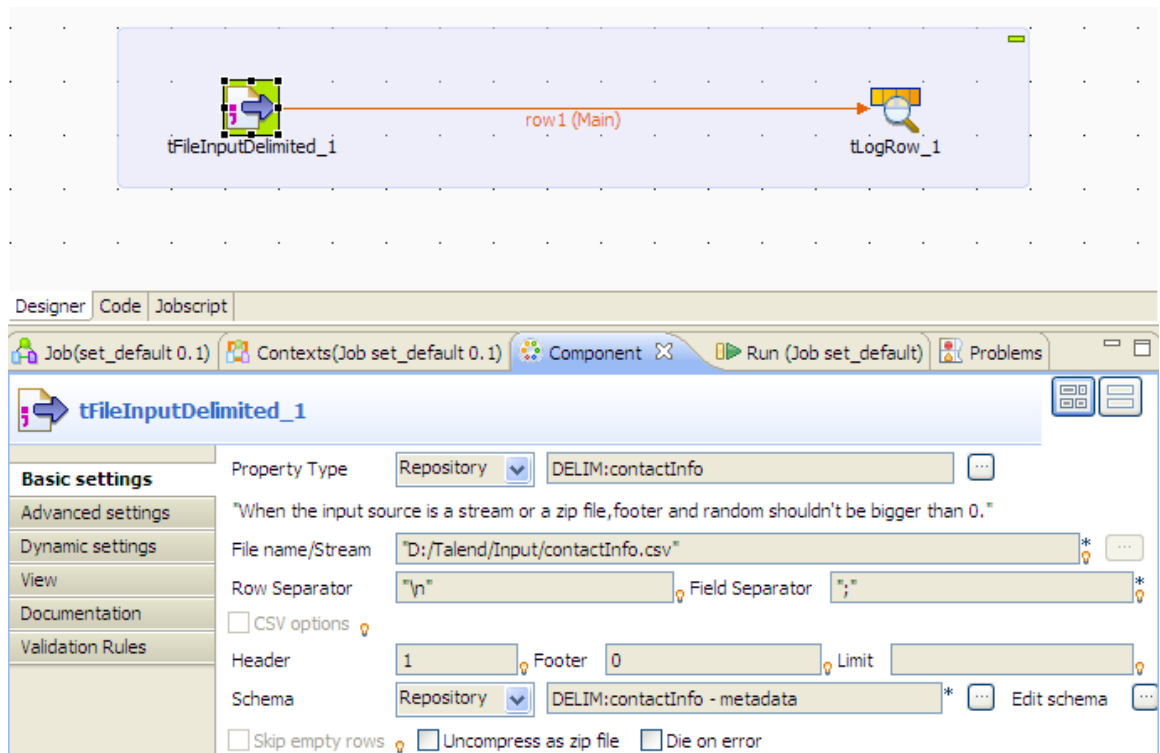
At present, only **tFileInputDelimited**, **tFileInputExcel**, and **tFixedFlowInput** support default values in the schema.

In the following example, the *company* and *city* fields of some records of the source CSV file are left blank, as shown below. The input component reads data from the source file and completes the missing information using the default values set in the schema, *Talend* and *Paris* respectively.

```
id;firstName;lastName;company;city;phone
1;Michael;Jackson;IBM;Roma;2323
2;Elisa;Black;Microsoft;London;4499
3;Michael;Dujardin;;8872
4;Marie;Dolvina;;6655
5;Jean;Perfide;;3344
6;Emilie;Taldor;Oracle;Madrid;2266
7;Anne-Laure;Paldufier;Apple;;4422
```

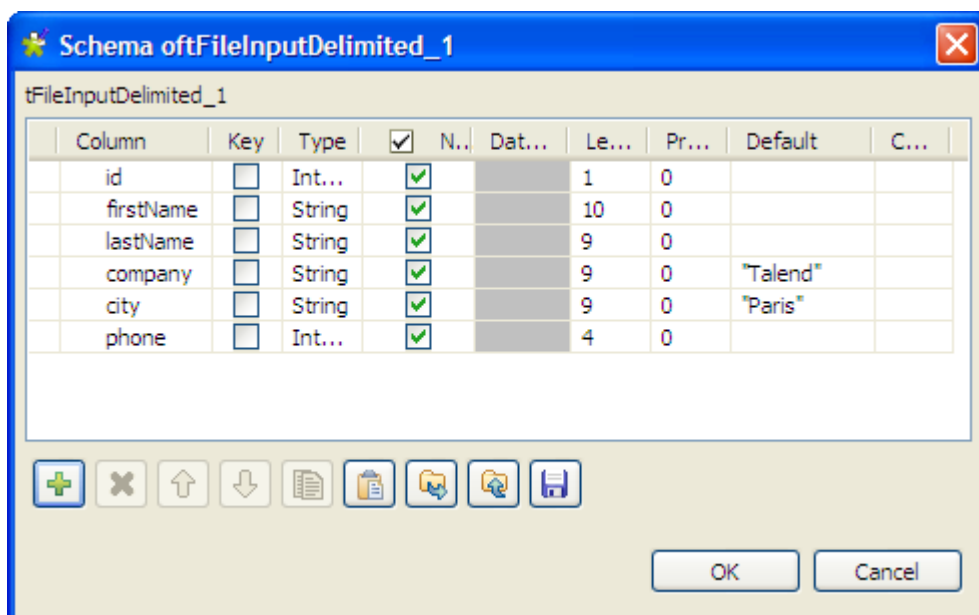
To set default values:

1. Double-click the input component **tFileInputDelimited** to show its **Basic settings** view.

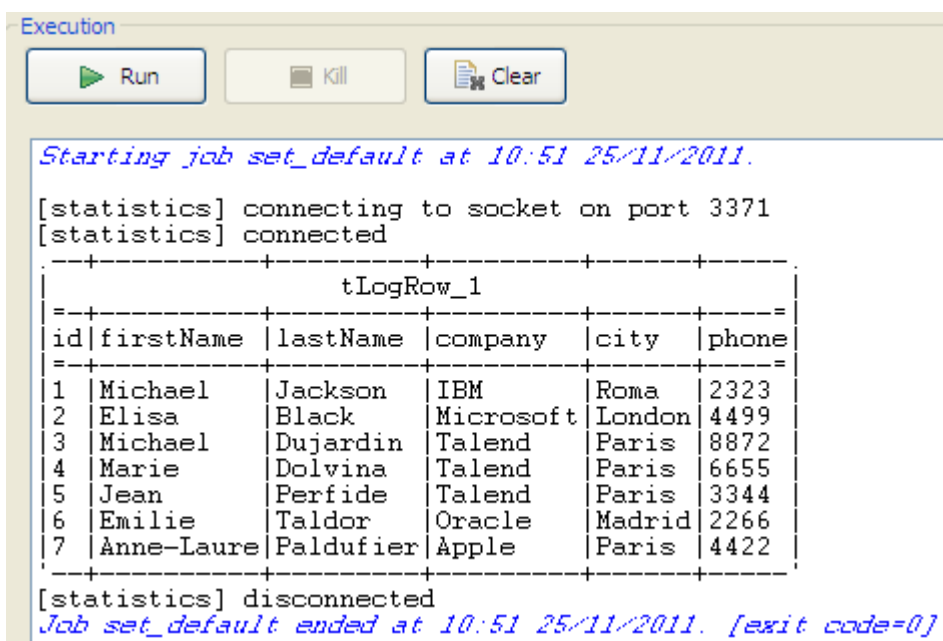


In this example, the metadata for the input component is stored in the Repository. For information about metadata creation in the Repository, see [Section 4.4.1, “How to centralize the Metadata items”](#).

2. Click the [...] button next to **Edit schema**, and select the **Change to built-in property** option from the pop-up dialog box to open the schema editor.
3. Enter *Talend* between quotation marks in the **Default** field for the *company* column, enter *Paris* between quotation marks in the **Default** field for the *city* column, and click **OK** to close the schema editor.



- Configure the output component **tLogRow** to display the execution result the way you want, and then run the Job.



In the output data flow, the missing information is completed according to the set default values.

Chapter 5. Designing a Route

Integrated business operations involve many complex relationships among various systems and applications. The decision on what to process, when and where, can change rapidly. *Talend Open Studio for ESB* provides dynamic mediation and routing capabilities based on enterprise integration patterns (EIPs) to meet the demands of these everchanging systems. Mediation and routing decisions are based on architecture, business rules, and deployed enterprise integration patterns. *Talend Open Studio for ESB* enables routing decisions at the endpoint. This eliminates bottlenecks and results in a scalable, high performance solution.

This chapter describes the process of using *Talend Open Studio for ESB* to create and configure mediation and routing rules by building graphical Routes.

Before starting any Route design processes, you need to be familiar with the *Talend Open Studio for ESB* Graphical User Interface (GUI). For more information, see [Appendix A, GUI](#).

5.1. What is a Route

A Route is a rule defining how messages will be moved from one service (or endpoint) to another. It is a graphical design, of two or more components connected together, that allows you to easily set up and test routing and mediation rules.

The Routes you design can address all of the different sources and targets that you need for your routing processes.

When you design a Route in *Talend Open Studio for ESB*, you can:

- put in place routing or mediation rules using a library of technical components based on the standard Enterprise Integration Patterns.
- change the default setting of components or create new components or family of components to match your exact needs.
- create and add items to the repository for reuse and sharing purposes (in other projects or Routes or with other users).



In order to be able to execute the Jobs you design in Talend Open Studio for ESB, you need to install an Oracle JVM 1.6 or later (IBM JVM is not supported). You can download it from <http://www.oracle.com/technetwork/java/javase/downloads/index.html>.

5.2. Accessing the Mediation perspective

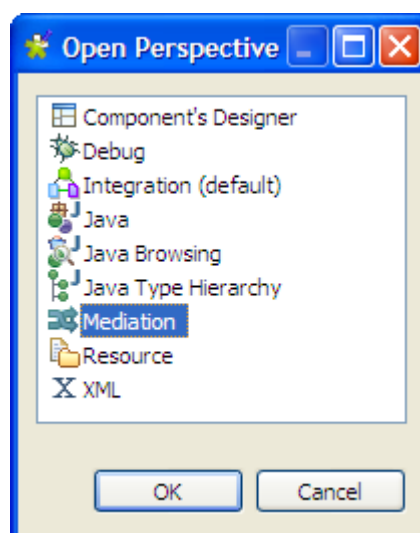
Talend Open Studio for ESB offers a comprehensive set of tools and functions for all its key capabilities accessible from one studio but in different perspectives. Thus, from *Talend Open Studio for ESB*, you will be able to design Business models, data services and data integration Jobs and so on via the **Integration** perspective and you will be able to design service Routes via the **Mediation** perspective.

5.2.1. Switching to the Mediation perspective

There are different ways to switch between all available perspectives including those of data services and service Routes.

To switch between different perspectives:

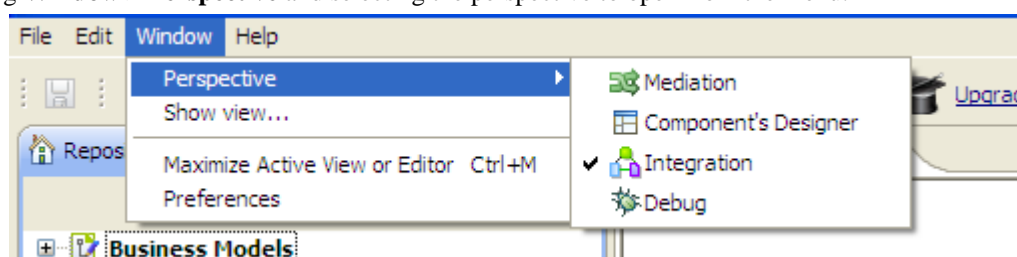
1. In the upper right corner of the *Talend Open Studio for ESB* main window, click the  icon and **Other...** to display the **[Open Perspective]** dialog box.



2. Select the perspective you want to access, **Mediation** for example, and click **OK** to close the dialog box.

Alternatively, you can switch between different perspectives by:

- clicking **Window > Perspective** and selecting the perspective to open from the menu.



- Directly clicking the corresponding icon, when available, on the upper right corner of the main window.



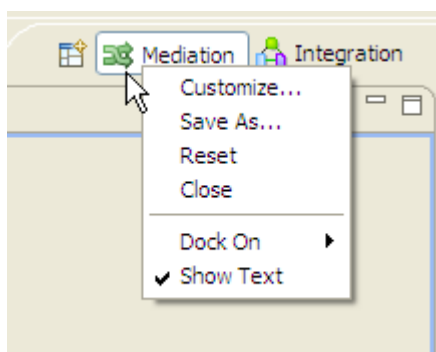
For more information on how to manage the display of these icons, see [Section 5.2.2, “Managing quick access icons for different perspectives”](#).

5.2.2. Managing quick access icons for different perspectives

Every time you open a perspective in *Talend Open Studio for ESB*, a corresponding icon is docked on the upper right corner of the main window.

To manage the display of these quick access icons, do the following:

1. Right-click the icon of the open perspective to display a contextual menu.



2. Select the needed management option from the list.

The quick access icon is changed accordingly.

The table below lists all available management options and their indications.

Option	Description
Customize	Opens a dialog box where you can -customize shortcuts in the current perspective -add command groups to the current perspective
Save As...	Changes the text that displays next to the icon as a title for the current perspective
Reset	Puts back the current perspective to its default
Close	Closes the current perspective
Dock on	Places the icons of the open perspectives: Top right: in the upper right corner of the Studio Top left: in the upper left corner of the Studio Left: to the left of the Studio
Show text	Displays/hides text next to the icon

5.3. Getting started with a basic Route



Until a Route is created, the design workspace is unavailable and the Palette does not display.

A Route is made of two or more components linked together. The properties of each component have to be configured individually, when necessary, in order to function properly.

The way components and connection between them is handled when building a Route is almost the same as when designing a Job. So, for more information, see [Section 4.2.3, “How to search components in the Palette”](#) and [Section 4.2.4, “How to connect components together”](#).

For a real-life use case of Routes, refer to the Route creation tutorial at <http://www.talendforge.org/tutorials/tutorial.php?language=english&idTuto=103>.

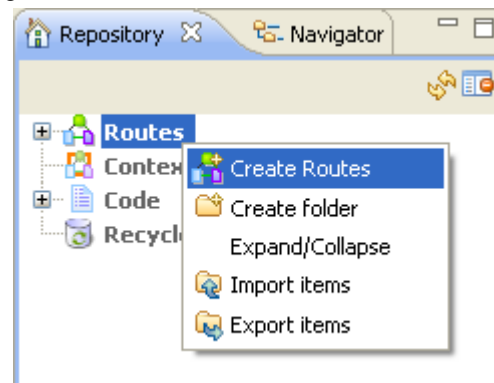
5.3.1. How to create a Route

Talend Open Studio for ESB enables you to create Routes by dropping different EIP components from the **Palette** onto the design workspace and then connecting these components together.

These created Routes are all stored in a central repository. You can create different folders to better classify them.

To create a Route, complete the following:

1. Open *Talend Open Studio for ESB* following the procedure as detailed in [Section 2.2, “Launching Talend Open Studio for ESB”](#).
2. In the **Repository** tree view, right-click the **Routes** node and select **Create Routes** from the contextual menu.



The **[New Route]** wizard opens to help you define the main properties of the new Route.

3. Enter the Route properties according to the description in the following table:

Field	Description
Name	the name of the new Route. A message comes up if you enter prohibited characters.

Field	Description
Purpose	Route purpose or any useful information regarding the Route use.
Description	Route description.
Author	a read-only field that shows by default the current user login.
Locker	a read-only field that shows by default the login of the user who owns the lock on the current Route. This field is empty when you are creating a Route and has data only when you are editing the properties of an existing Route.
Version	a read-only field. You can manually increment the version using the M and m buttons. For more information, see Section 7.5, “Managing Job and Route versions” .
Status	a list to select from the status of the Route you are creating.
Path	a list to select from the folder in which the Route will be created.

An empty design workspace opens up showing the name of the Route as a tab label and the Route is now listed under the **Routes** node in the **Repository** tree view.

- Drop the components you want to use in your Route from the **Palette** onto the design workspace and connect them together. For more information, see [Section 4.2.2, “How to drop components to the workspace”](#) and [Section 4.2.4, “How to connect components together”](#).
- Define the properties of each of the components used in the Route, when necessary. For more information, see [Section 4.2.6, “How to define component properties”](#).
- Save your Route and then press **F6** to execute it. For more information, see [Section 4.2.7, “How to run a Job”](#).

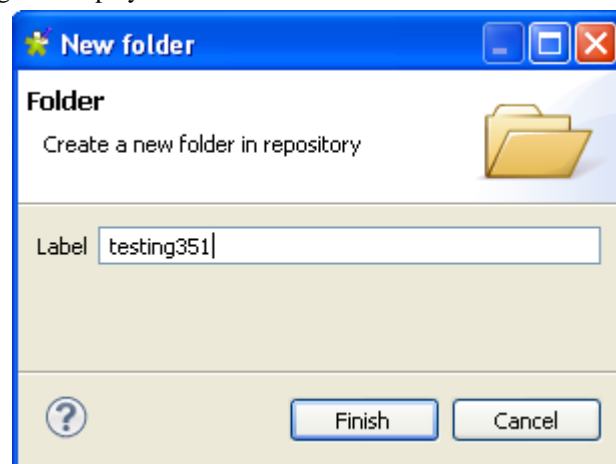


You can open one or more of the created Routes by simply double-clicking their label in the **Repository** tree view.

To create different folders for your Routes, complete the following:

- In the **Repository** tree view, right-click **Routes** and select **Create folder** from the contextual menu.

The **[New folder]** dialog box displays.



- In the **Label** field, enter a name for the folder and then click **Finish** to confirm your changes and close the dialog box.

The created folder is listed under the **Routes** node in the **Repository** tree view.



If you have already created Routes that you want to move into this new folder, simply drop them into the folder.

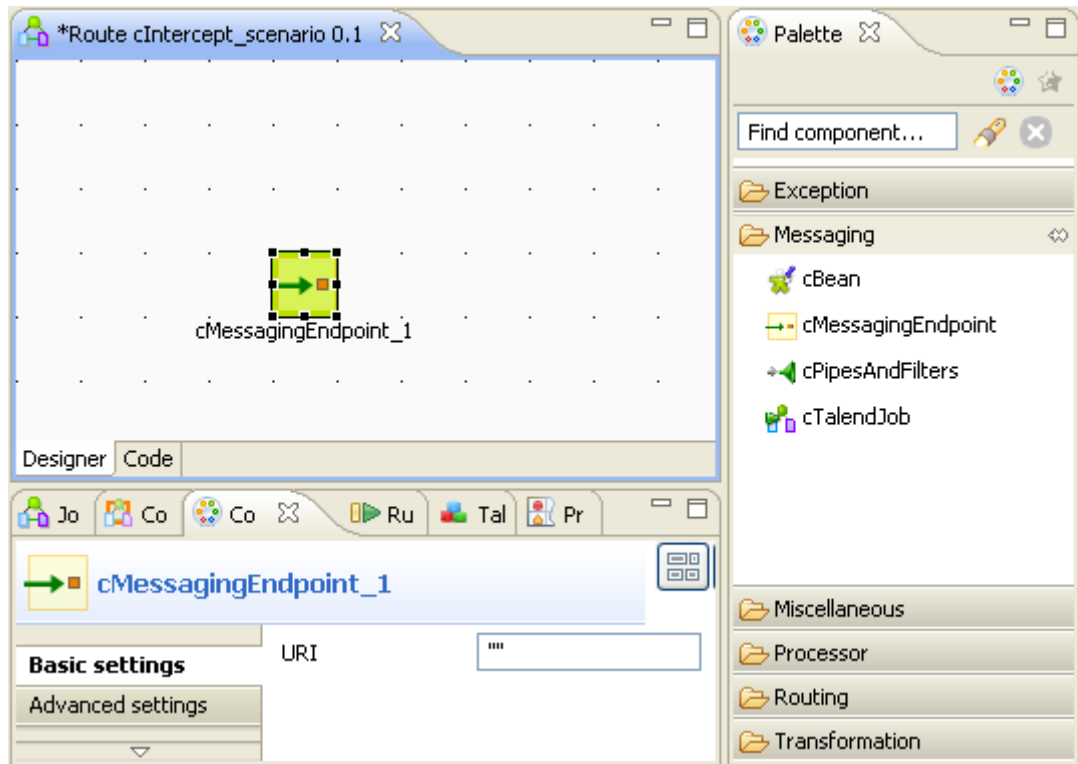
For a scenario showing how to create a real-life Route, see *Talend Open Studio for ESB Mediation Components Reference Guide*.

5.3.2. How to drop components to the workspace

To actually start building a Route, click a Component on the **Palette**. Then click again on the design workspace to drop it there and add it to your Route.



If the **Palette** does not show in the Studio, see [the section called “How to show, hide the Palette and change its position”](#).



Connect components together in a logical order using the connections offered, in order to build a full routing rule. For more information about component connection types, see [Section 4.3.1, “Connection types”](#).

Multiple information or warnings may show next to the component. Browse over the component icon to display the information tooltip. This will display until you fully completed your Route and defined all basic (and sometimes advanced) component properties of the **Component** view.

Related topics:

- [Section 4.2.4, “How to connect components together”](#).
- [Section 4.6.3.1, “Warnings and error icons on components”](#).
- [Section 4.2.6, “How to define component properties”](#).

5.3.3. How to define component properties

The properties information for each component forming a Route allows to set the actual technical implementation of the active Route.

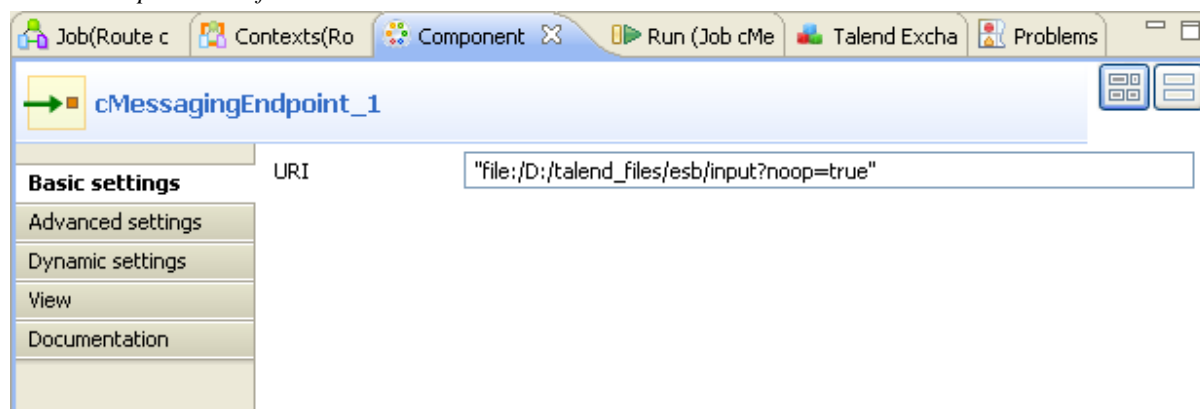
Each component is defined by basic and advanced properties shown respectively on the **Basic Settings** tab and the **Advanced Settings** tab of the **Component** view of the selected component in the design workspace. The

Component view gathers also other collateral information related to the component in use, including **View** and **Documentation** tabs.

For detailed configuration for each component displaying in the **Palette** of the **Mediation** perspective, see *Talend Open Studio for ESB Mediation Components Reference Guide*.

5.3.3.1. Basic Settings tab

The **Basic Settings** tab is part of the **Component** view, which is located on the lower part of the designing editor of *Talend Open Studio for ESB*.



Each component has specific basic settings according to its function requirements within the Route. For a detailed description of each component properties and use, see *Talend Open Studio for ESB Mediation Components Reference Guide*.

How to set a field dynamically (Ctrl+Space bar)

On any field of your Route/component settings view, you can use the **Ctrl+Space** bar to access the global and context variable list and set the relevant field value dynamically.

1. Place the cursor on any field of the **Component** view.
2. Press **Ctrl+Space** bar to access the proposal list.
3. Select on the list the relevant parameters you need. Appended to the variable list, a information panel provides details about the selected parameter.

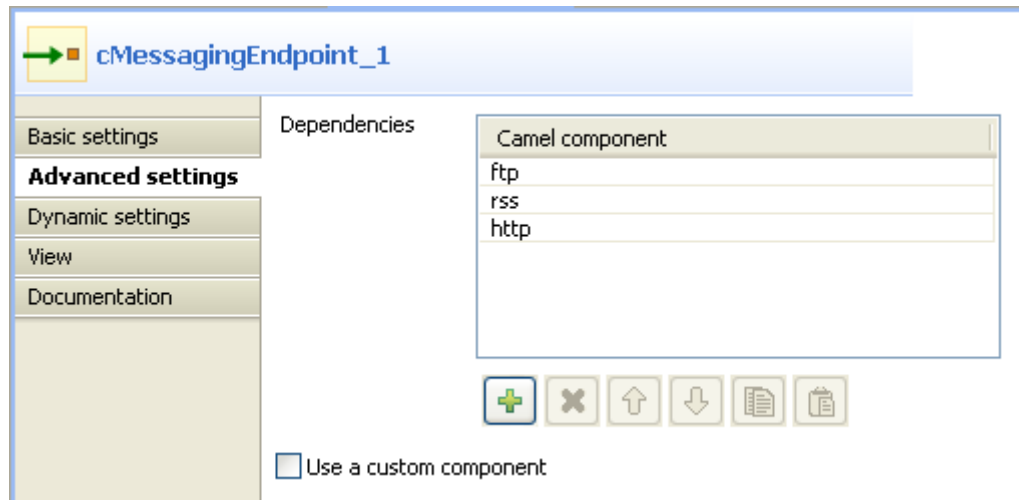
<p>Description: Error Message</p> <p>Global variable, property of component tMap [tMap_1].</p> <p>Type: String</p> <p>Availability: After</p> <p>Variable Name: ((String)globalMap.get("tMap_1_ERROR_MESSAGE"))</p>	<p>tFileInputDelimited_2.ERROR_MESSAGE</p> <p>tFileInputDelimited_2.NB_LINE</p> <p>tMap_1.ERROR_MESSAGE</p> <p>tFileOutputDelimited_1.ERROR_MESSAGE</p> <p>tFileOutputDelimited_1.NB_LINE</p> <p>tFileOutputDelimited_2.ERROR_MESSAGE</p> <p>tFileOutputDelimited_2.NB_LINE</p> <p>tFileInputDelimited_3.ERROR_MESSAGE</p> <p>tFileInputDelimited_3.NB_LINE</p> <p>tFlowMeter_1.ERROR_MESSAGE</p> <p>tFlowMeter_2.ERROR_MESSAGE</p>
---	--

This can be any parameter including: error messages, system variables, etc. The list varies according to the component in selection or the context you're working in.

Related topic: [Section 4.4.2, "How to centralize contexts and variables"](#).

5.3.3.2. Advanced settings tab

Some components, for example the **cMessagingEndpoint** component, can require advanced settings in some cases to function properly.




The content of the **Advanced settings** tab changes according to the selected component.

Generally you will find on this tab the parameters that are not required for a basic or usual use of the component but may be required for a use out of the standard scope.

5.3.3.3. View tab

The **View** tab of the **Component** view allows you to change the default display format of components on the design workspace.

Field	Description
Label format	Free text label showing on the design workspace. Variables can be set to retrieve and display values from other fields. The field tooltip usually shows the corresponding variable where the field value is stored.  It is recommended to label each component with a unique name to better identify its role in the Route. This is especially useful for the code generation of some components, for example, the cCXF component. For more information, see <i>Talend Open Studio for ESB Mediation Components Reference Guide</i> .
Hint format	Hidden tooltip, showing only when you mouse over the component.
Connection format	Indicates the type of connection accepted by the component.

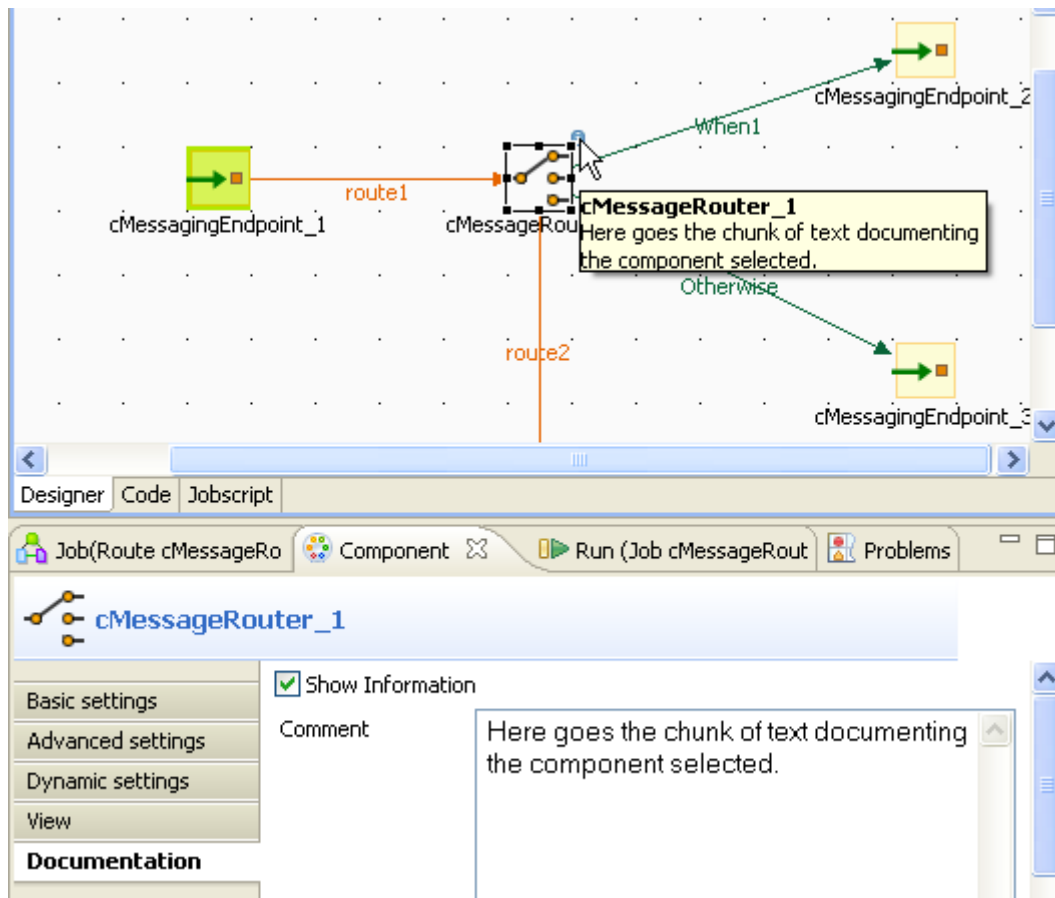
You can graphically highlight both **Label** and **Hint** text with HTML formatting tags:

- Bold: ` YourLabelOrHint `
- Italic: `<i> YourLabelOrHint </i>`
- Return carriage: `YourLabelOrHint
 ContdOnNextLine`
- Color: ` YourLabelOrHint `

To change your preferences of this View panel, click **Window>Preferences>Talend>Designer**.

5.3.3.4. Documentation tab

Feel free to add any useful comment or chunk of text or documentation to your component.



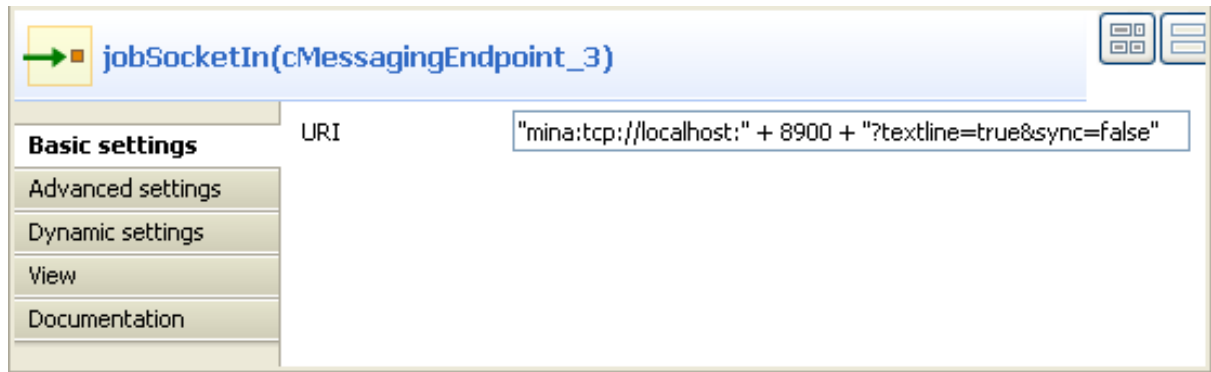
In the **Documentation** tab, you can add your text in the **Comment** field. Then, select the **Show Information** check box and an information icon display next to the corresponding component in the design workspace.

You can show the Documentation in your hint tooltip using the associated variable `_COMMENT_`, so that when you place your mouse on this icon, the text written in the **Comment** field displays in a tooltip box.

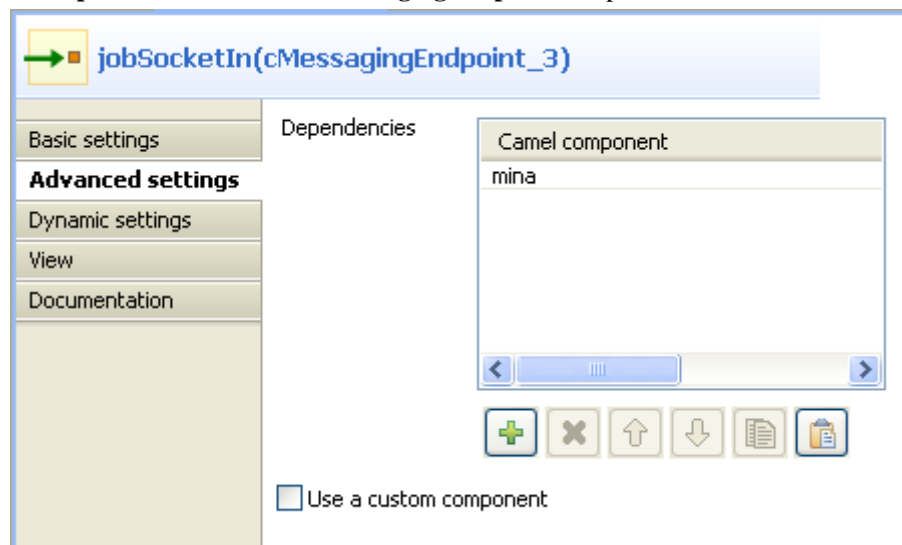
5.3.4. How to use Camel components in a Route

There are many Camel components such as camel-mina, camel-http4 and others which are supported, but not directly included in the **Palette**. These Camel components require the generic `cMessagingEndpoint` to use them, and need to be added to the dependencies list of the `cMessagingEndpoint` component.

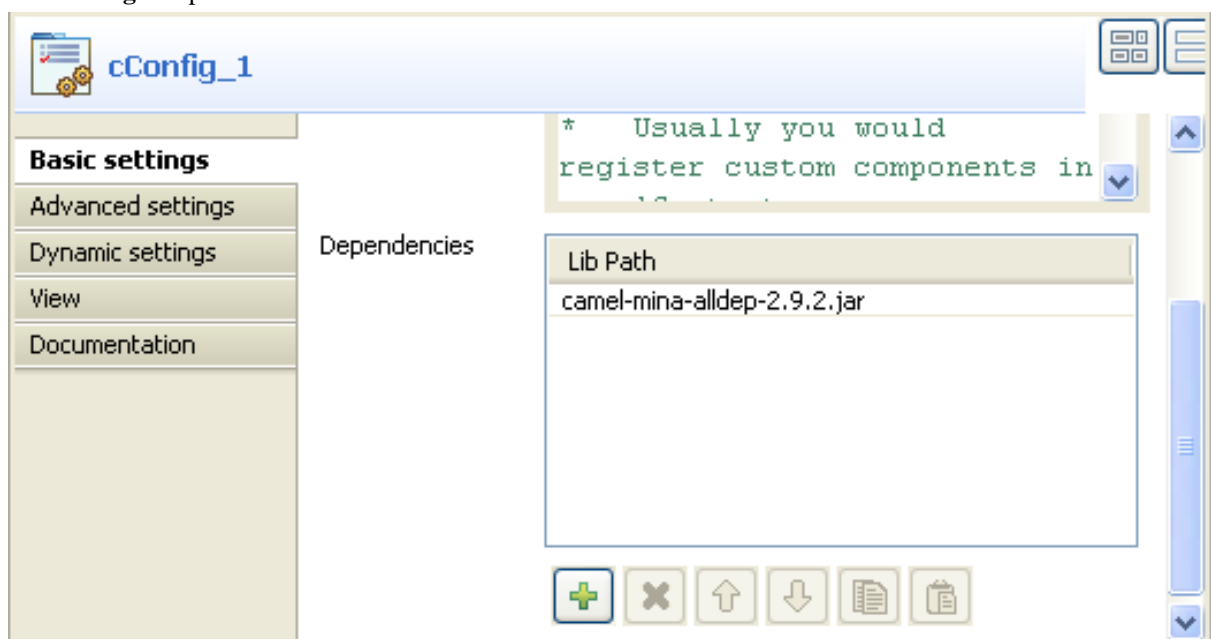
The following screenshot shows an example of the camel-mina component being used.



It is added to the **Dependencies** list of the **cMessagingEndpoint** component in its **Advanced settings** view.



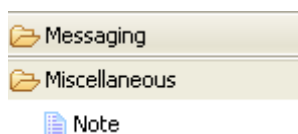
Alternatively, you can use a **cConfig** and add the library of the Camel component to the **Dependencies** list of the **cConfig** component.



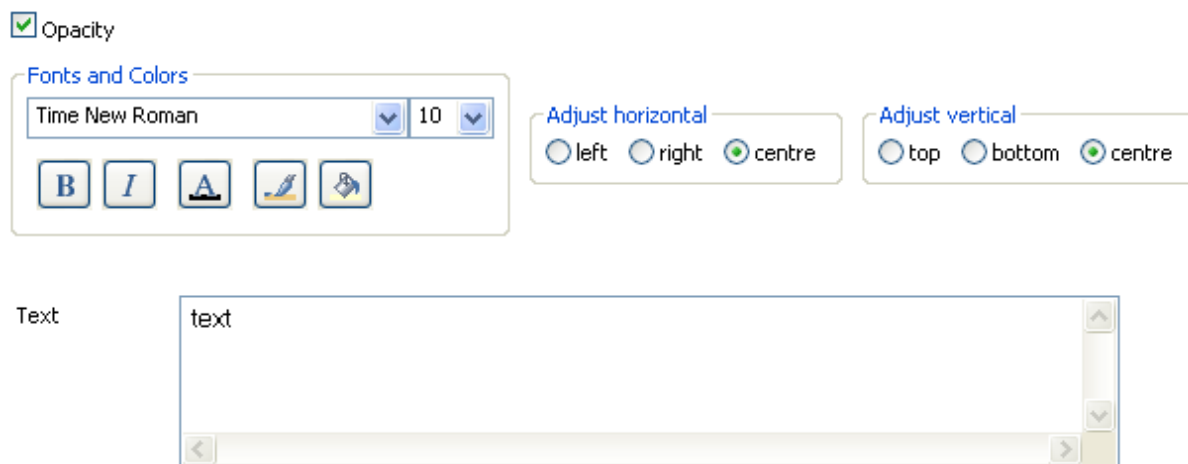
For a real-life use case of the Camel components, see [Section B.2, "Job and Route example"](#).

5.3.5. How to add notes to a Route

In the **Palette**, click the **Miscellaneous** family and then drop the **Note** element to the design workspace to add a text comment to a particular component or to the whole Route.



You can change the note format. To do so, select the note you want to format and click the **Basic setting** tab from the **Component** view.



Select the **Opacity** check box to display the background color. By default, this box is selected when you drop a note on the design workspace. If you clear this box, the background becomes transparent.

You can select options from the **Fonts and Colors** list to change the font style, size, color, and so on as well as the background and border color of your note.

You can select the **Adjust horizontal** and **Adjust vertical** boxes to define the vertical and horizontal alignment of the text of your note.

The content of the **Text** field is the text displayed on your note.

5.3.6. How to run a Route

You can execute a Route in several ways. This mainly depends on the purpose of your Route execution and on your user level.

If you are an advanced Java user and want to execute your Route step by step to check and possibly modify it on the run, see [Section 4.2.7.2, “How to run a Job in Java Debug mode”](#).

If you do not have advanced Java knowledge and want to execute and monitor your Job in normal mode, see [Section 4.2.7.1, “How to run a Job in normal mode”](#).

5.3.6.1. How to run a Route in normal mode



Make sure you saved your **Route** before running it in order for all properties to be taken into account.

To run your Route in a normal mode, complete the following:

1. Click the **Run** view to access it.
2. Click the **Basic Run** tab to access the normal execution mode.
3. In the **Context** area to the right of the view, select in the list the proper context for the Route to be executed in. You can also check the variable values.

If you have not defined any particular execution context, the context parameter table is empty and the context is the default one. Related topic: [Section 4.4.2, “How to centralize contexts and variables”](#).

1. Click **Run** to start the execution.
2. On the same view, the console displays the progress of the execution. The log is provided by the Apache logging utility log4j shipped with the Route builder. By default the message level is set to INFO in the log4j.properties file, as shown below. It includes any error message as well as start and end messages. It also shows the Route output in case you used a System.out.println Java code in a **cProcessor** component, for example. For more information about the logging utility, see the site <http://logging.apache.org/log4j/1.2/>.

```
#
# The logging properties used
#
log4j.rootLogger=INFO, out

# uncomment the following line to turn on Camel debugging
#log4j.logger.org.apache.camel=DEBUG

# uncomment the following line to turn on ActiveMQ debugging
#log4j.logger.org.apache.activemq=DEBUG

log4j.logger.org.springframework=WARN

# CONSOLE appender not used by default
log4j.appender.out=org.apache.log4j.ConsoleAppender
log4j.appender.out.layout=org.apache.log4j.PatternLayout
log4j.appender.out.layout.ConversionPattern=[%-15.15t] %-30.30c{1} %-5p %m%n
#log4j.appender.out.layout.ConversionPattern=%d [%-15.15t] %-5p %-30.30c{1} - %m%n

log4j.throwableRenderer=org.apache.log4j.EnhancedThrowableRenderer
```

This INFO logging level is fixed and can not be changed in *Talend Open Studio for ESB*. However, when deploying your route in the **Talend Runtime**, you can decide the level used by the **Talend Runtime** to log information. For more information about the logging system of **Talend Runtime**, see the *Talend ESB Container Administration Guide*.

An example of the log without errors is shown below.

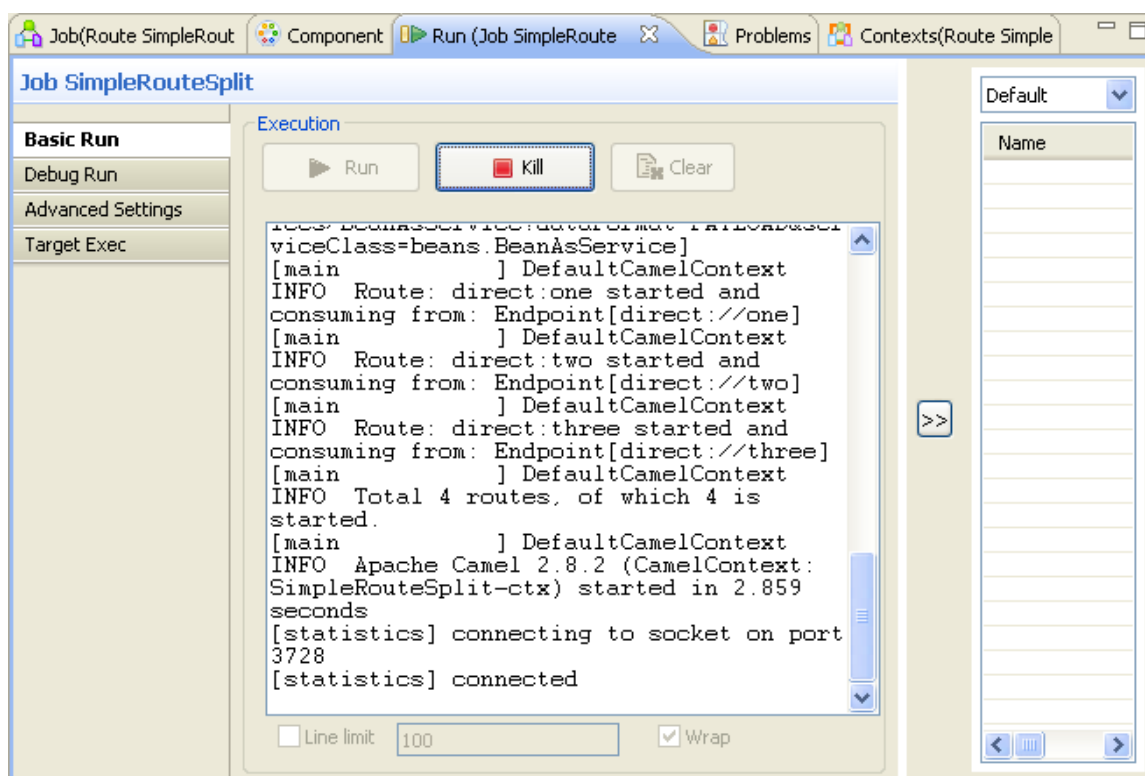
```
Starting job cMessageRouter at 15:47 06/04/2012.

[main      ] AnnotationTypeConverterLoader  INFO  Found 3 packages with
15 @Converter classes to load
[main      ] DefaultTypeConverter              INFO  Loaded 169 core type
converters (total 169 type converters)
[main      ] DefaultTypeConverter              INFO  Loaded additional 0
type converters (total 169 type converters) in 0.016 seconds
[main      ] MainSupport                          INFO  Apache Camel
2.9.2-SNAPSHOT starting
[main      ] ManagementStrategyFactory           INFO  JMX enabled. Using
ManagedManagementStrategy.
[main      ] DefaultCamelContext                INFO  Apache Camel
2.9.2-SNAPSHOT (CamelContext: cMessageRouter-ctx) is starting
[main      ] ultManagementLifecycleStrategy INFO  StatisticsLevel at All
so enabling load performance statistics
[main      ] FileEndpoint                       INFO  Endpoint is configured
with noop=true so forcing endpoint to be idempotent as well
[main      ] FileEndpoint                       INFO  Using default memory
based idempotent repository with cache max size: 1000
[main      ] XPathBuilder                       INFO  Created default
XPathFactory com.sun.org.apache.xpath.internal.jaxp.XPathFactoryImpl@10e35d5
[main      ] DefaultCamelContext                INFO  Route: Sender started
and consuming from:
Endpoint[file:///F:/data/input/messagerouter?bufferSize=128&noop=true]
[main      ] DefaultCamelContext                INFO  Route:
directParisRoute started and consuming from: Endpoint[direct://Paris]
[main      ] DefaultCamelContext                INFO  Route:
directOthersRoute started and consuming from: Endpoint[direct://Others]
[main      ] DefaultCamelContext                INFO  Total 3 routes, of
which 3 is started.
[main      ] DefaultCamelContext                INFO  Apache Camel
2.9.2-SNAPSHOT (CamelContext: cMessageRouter-ctx) started in 0.672 seconds
[statistics] connecting to socket on port 4009
[statistics] connected
[t/messagerouter] Monitor_Paris      INFO  Message sent to folder
Paris_only: Message_1.xml
[t/messagerouter] Monitor_Others     INFO
Exchange[ExchangePattern: InOnly,
BodyType: org.apache.camel.component.file.GenericFile, Body:[Body is file
based: GenericFile[F:\data\input\messagerouter\Message_2.xml]]]
```

The following screenshot shows another example of the log with error messages.

```
[main      ] AnnotationTypeConverterLoader  INFO  Found 3 packages with 15 @Converter classes to load
[main      ] DefaultTypeConverter              INFO  Loaded 169 core type converters (total 169 type converters)
[main      ] DefaultTypeConverter              INFO  Loaded additional 0 type converters (total 169 type converters) in 0.000 seconds
[main      ] MainSupport                          INFO  Apache Camel 2.9.2-SNAPSHOT starting
[main      ] ManagementStrategyFactory           INFO  JMX enabled. Using ManagedManagementStrategy
[main      ] DefaultCamelContext                INFO  Apache Camel 2.9.2-SNAPSHOT (CamelContext: cMessageRouter-ctx) is starting
[main      ] ultManagementLifecycleStrategy INFO  StatisticsLevel at All so enabling load performance statistics
[main      ] DefaultCamelContext                INFO  Apache Camel 2.9.2-SNAPSHOT (CamelContext: cMessageRouter-ctx) is shutting down
[main      ] DefaultShutdownStrategy            INFO  Starting to graceful shutdown 0 routes (timeout 300 seconds)
[main      ] DefaultShutdownStrategy            INFO  Graceful shutdown of 0 routes completed in 0 seconds
[main      ] DefaultInflightRepository           INFO  Shutting down with no inflight exchanges.
[main      ] DefaultCamelContext                INFO  Uptime: 0.156 seconds
Failed to create route directParisRoute: Route[From[diect:Paris]] -> [To[file:///F:/data/output/Paris... because of Failed to resolve
endpoint: diect://Paris due to: No component found with scheme: diect
org.apache.camel.FailedToCreateRouteException: Failed to create route directParisRoute: Route[From[diect:Paris]] ->
[To[file:///F:/data/output/Paris... because of Failed to resolve endpoint: diect://Paris due to: No component found with scheme: diect
at org.apache.camel.model.RouteDefinition.addRoutes(RouteDefinition.java:173)
at org.apache.camel.impl.DefaultCamelContext.startRoute(DefaultCamelContext.java:710)
at org.apache.camel.impl.DefaultCamelContext.startRouteDefinitions(DefaultCamelContext.java:1734)
at org.apache.camel.impl.DefaultCamelContext.doStart(DefaultCamelContext.java:1526)
at org.apache.camel.impl.DefaultCamelContext.doStart(DefaultCamelContext.java:1421)
at org.apache.camel.support.ServiceSupport.start(ServiceSupport.java:60)
at org.apache.camel.impl.DefaultCamelContext.start(DefaultCamelContext.java:1389)
[main      ] DefaultCamelContext                INFO  Apache Camel 2.9.2-SNAPSHOT (CamelContext: cMessageRouter-ctx) is shutdown in 0.000
seconds
[main      ] MainSupport                          INFO  Apache Camel 2.9.2-SNAPSHOT stopping
at talenddemojava.cmessagerouter_0_1.cMessageRouter$1CamelImpl.doStart(cMessageRouter.java:223)
at org.apache.camel.support.ServiceSupport.start(ServiceSupport.java:60)
at org.apache.camel.main.MainSupport.run(MainSupport.java:139)
at talenddemojava.cmessagerouter_0_1.cMessageRouter.Route(cMessageRouter.java:326)
at talenddemojava.cmessagerouter_0_1.cMessageRouter.runJobInTOS(cMessageRouter.java:447)
at talenddemojava.cmessagerouter_0_1.cMessageRouter.main(cMessageRouter.java:360)
Caused by: org.apache.camel.ResolveEndpointFailedException: Failed to resolve endpoint: diect://Paris due to: No component found with scheme:
diect
at org.apache.camel.impl.DefaultCamelContext.getEndpoint(DefaultCamelContext.java:462)
at org.apache.camel.util.CamelContextHelper.getMandatoryEndpoint(CamelContextHelper.java:48)
at org.apache.camel.model.RouteDefinition.resolveEndpoint(RouteDefinition.java:193)
at org.apache.camel.impl.DefaultRouteContext.resolveEndpoint(DefaultRouteContext.java:106)
at org.apache.camel.impl.DefaultRouteContext.resolveEndpoint(DefaultRouteContext.java:112)
at org.apache.camel.model.FromDefinition.resolveEndpoint(FromDefinition.java:72)
at org.apache.camel.impl.DefaultRouteContext.getEndpoint(DefaultRouteContext.java:88)
at org.apache.camel.model.RouteDefinition.addRoutes(RouteDefinition.java:83)
at org.apache.camel.model.RouteDefinition.addRoutes(RouteDefinition.java:168)
... 12 more
[Thread-1 ] MainSupport$HangupInterceptor INFO  Received hang up - stopping the main instance.
```

- To define the lines of the execution progress to be displayed in the console, select the **Line limit** check box and type in a value in the field.
- Select the **Wrap** check box to wrap the text to fit the console width. This check box is selected by default. When it is cleared, a horizontal scroll bar appears, allowing you to view the end of the lines.



Before running again a Route, you might want to remove the execution statistics from the designing workspace. To do so, click the **Clear** button.

Routes are running continuously, so if you want to stop a Route, you will have to click the **Kill** button. You will need to click the **Run** button again, to start again the Route.

Talend Open Studio for ESB offers to display execution statistics during the execution of your Route, which facilitate the Route monitoring, as well as a debugging execution mode. Java Debug execution mode works identically for Jobs and Routes, so for more information, see [Section 4.2.7.2, “How to run a Job in Java Debug mode”](#) and for more information about statistics and other execution features, see [Section 4.2.7.4, “How to set advanced execution settings”](#).

5.4. Using connections

In *Talend Open Studio for ESB*, a Route is composed of a group of components logically linked to one another via connections. This section will describe the types of connections available for a Route and their related settings.

5.4.1. Connection types

There are various types of connections which define how to route your messages within a camel Route.

Right-click a component on the design workspace to display a contextual menu that lists all available links for the selected component.

The sections below describe all available connection types.

5.4.1.1. Row connection

A **Row** connection handles the messages to be routed. The **Row** connections can be **try**, **catch**, **finally** or **route** according to the component selected.

route

This type of connection is the most commonly used connection. It passes on messages from one endpoint to the other, or from one endpoint to a processor and from a processor to another endpoint.

try

This link connects specifically a **cTry** component to a receiving component to be able to handle error in part of your Route.

To isolate the part of your Route likely to generate an error, you can put it in a Try block via the **cTry** component and its **try** link. Once isolated, if the Route generates an error, the error will be sent to the error handler unless a Catch block is found right after the Try block. For more information, see the **catch** section below.

catch

The **catch** link can only be used with a **cTry** component and if a **try** link has already been used in order to isolate a part of a Route likely to generate an error.

So, the **catch** link catches the errors generated by the Route put in a Try block and enable you to handle it, if necessary, and continue the Route, if possible.

finally

The **finally** link can only be used with a **cTry** component and if a **try** link has already been used to isolate part of a Route likely to generate an error.

The **finally** link will enable you to execute final instructions regardless of any problem that may occur in the Try and/or Catch block, to close a connection to a database, for example.

5.4.1.2. Trigger connections

Trigger connections define specific channels to route messages in according to specific conditions.

Messages will be filtered and routed to specific Routes according to defined conditions.

when

The **when** link connects specifically a **cMessageRouter** component to receiving components to filter and route messages in one or several output Routes according to specified conditions.

These conditions can be defined in the **Connection** settings of each **when** link you create in your Route. Messages not matching the condition set can be retrieved with an **otherwise** link. For more information, see the **otherwise** section below.

otherwise

The **otherwise** link can only be used with a **cMessageRouter** component and if at least one **when** link has already been used to filter and route messages.

So, the **otherwise** link retrieves all messages that are not matching the conditions defined in the several **when** links.



It is recommended not to put any message handling after the **when** or the **otherwise** link. Always use a Mock/Direct endpoint to replace them and make a new Route to handle the messages.

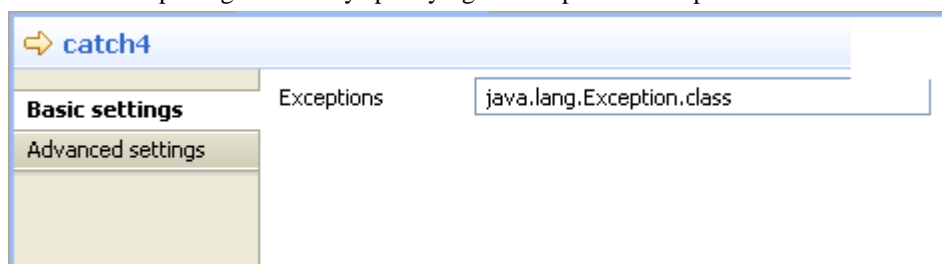
5.4.2. How to define connection settings

You can display the properties of a connection by selecting it and clicking the **Component** view tab, or by right-clicking the connection and selecting **Settings** from the contextual menu. However, not all connections have properties displayed in this tab. This section summarizes connection property settings.

5.4.2.1. Catch connection settings

You can set a **Catch** link to catch the exception thrown by the **try** link of the **cTry** component.

1. Simply select the **Catch** link of your Route to display the related **Basic settings** view of the **Components** tab.
2. In the **Exceptions** field, type in the name of the exception class that is likely to occur in the **try** routing. If you type in *java.lang.Exception.class*, mother of all exceptions, any exception will be caught. But you can try to fine-tune the exception generated by specifying a more precise exception class.



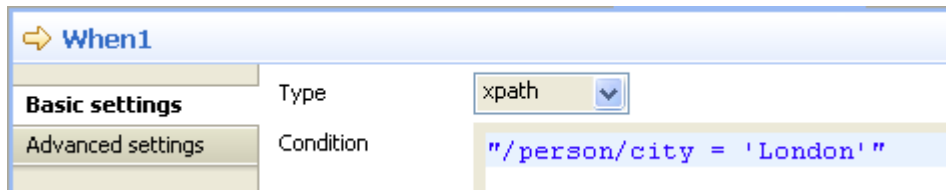
When executing your Route, if an exception occurs in routing following a **try** link, it will be caught by the **catch** link corresponding better to the exception. For more information, see [Section 4.3.1.1, “Row connection”](#).

5.4.2.2. When connection settings

You can set a **When** link to filter messages according to a condition and route those filtered messages:

1. Simply select the **When** link of your routing to display the related **Basic settings** view of the **Components** tab.

2. In the **Type** list, select the type of condition you want to use in the routing.
3. In the **Condition** field, type in the condition you want to use to filter your messages.



When executing your Route, messages matching the condition defined in the **when** link will be passed to the following component and messages not matching it can either be lost or retrieved via an **otherwise** link. For more information, see [Section 5.4.2.1, “Catch connection settings”](#) and for a scenario using those links, see *Talend Open Studio for ESB Mediation Components Reference Guide*.

5.5. Using Beans

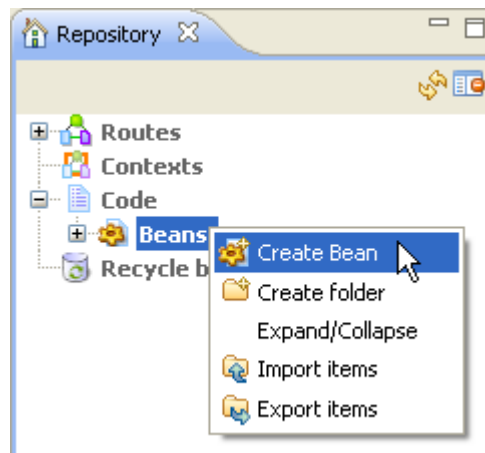
Talend Open Studio for ESB offers you the possibility to create Java Beans that can easily be called by the Mediation components in Routes. This way, you will be able to easily reuse libraries, components, applets, or other classes.

Beans are stored under the **Code** node in the **Repository**.

5.5.1. How to create a Bean

To create a new bean:

1. In the **Repository**, expand the **Code** folder.
2. Right-click the **Beans** node.
3. Select **Create Bean** from the contextual menu.



The **[New Bean]** wizard opens to help you define the main properties of the new Bean.

4. Enter the Bean properties as the following:

Field	Description
Name	the name of the new Bean. A message comes up if you enter prohibited characters.
Purpose	the purpose of the Bean or any useful information regarding the Bean use.
Description	Bean description.
Author	a read-only field that shows by default the current user login.
Locker	a read-only field that shows by default the login of the user who owns the lock on the current Bean. This field is empty when you are creating a Bean and has data only when you are editing the properties of an existing Bean.
Version	a read-only field. You can manually increment the version using the M and m buttons. For more information, see Section 7.5, “Managing Job and Route versions” .
Status	a list to select from the status of the Bean you are creating.
Path	a list to select from the folder in which the Bean will be created.

5. Click **Finish** to create the Bean.

An example bean opens in the editor.

6. Type in the code of your Bean.

5.5.2. How to use a Bean

Once created, you can use the Beans in the **Mediation** components. To do so, go to the **Component** view of the relevant component to define its properties:

For some components (for example **cBean**, **cDynamicRouter**, etc.), you can directly call Beans:

1. In the **Beans class** field of those components, type in *beans.BEAN_NAME.class* and the corresponding bean will be called.
2. If the Bean called has more than one method, you can specify which method you want to call by selecting the **Specify the method** check box and typing its name in the field displaying.

For some components (for example **cAggregate**, **cMutlicast**, etc.), you can call Beans as an aggregation strategy:

1. Select the **Use aggregation strategy** check box.
2. In the **Strategy** field, type in the name of the Bean without any prefix or extention.

For more information on how to create Routes, see [Section 5.3.1, “How to create a Route ”](#).

For more information on the properties and usage of the Mediation components, see *Talend Open Studio for ESB Mediation Components Reference Guide*.

Chapter 6. Designing a Service

Talend Open Studio for ESB is the tool with the capabilities that treat all of the different sources and targets required in data integration and data service processes and all other associated operations.

Talend Open Studio for ESB helps you to design data services that allow you to build Web services combined with data service Jobs.

This chapter addresses the needs of programmers or IT managers who are ready to implement the technical aspects of a Business Model (regardless of whether it was designed in *Talend Open Studio for ESB*'s Business Modeler).

Before starting any data integration and data service processes, you need to be familiar with the *Talend Open Studio for ESB* Graphical User Interface (GUI). For more information, see [Appendix A, GUI](#).

6.1. What is a Service

Talend Open Studio for ESB combines data integration with Web services and enables the graphical design of a Service which includes a WSDL file and one or more data service Jobs that addresses all of the different sources and targets required to publish the Web service. The WSDL editor makes it possible to create and edit WSDL files graphically, automating most of the tasks involved with these processes.

When you design a Service in *Talend Open Studio for ESB*, you can:

- Create new WSDL files or import existing WSDL files for structured viewing.
- Create, set, and delete WSDL objects.
- Access the structured view at any time to edit or document the WSDL objects in the Service.
- Associate Services with data service Jobs.
- Create and add items to the repository for reuse and sharing purposes (in other projects or Services or with other users).



*In order to be able to execute the data service Jobs you design in *Talend Open Studio for ESB*, you need to install an Oracle JVM 1.6 or later (IBM JVM is not supported). You can download it from <http://www.oracle.com/technetwork/java/javase/downloads/index.html>.*

6.2. Getting started with a basic Service



Until a Service is created, the design workspace is unavailable.

A Service consists of a WSDL file and one or more data service Jobs. Data service Jobs address all of the different sources and targets that you need for data integration processes and combine them with Web services. The properties of each WSDL objects and data service Job components require to be configured individually, in order to function properly.

For more information, see [Section 6.2.2, “How to edit a WSDL file”](#) and [Section 6.2.3, “How to associate data service Jobs with a Service”](#).

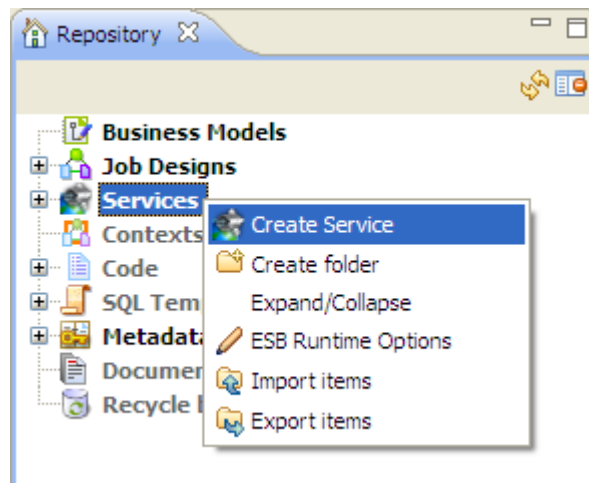
6.2.1. How to create a Service

Talend Open Studio for ESB enables you to create a Service from an existing WSDL file or to create a new WSDL file from scratch using the WSDL editor.

You can also create different folders to better classify these Services.

To create a Service:

1. Open *Talend Open Studio for ESB* following the procedure as detailed in [Section 2.2, “Launching Talend Open Studio for ESB”](#).
2. In the **Repository** tree view, right-click the **Services** node and select **Create Service** from the contextual menu.



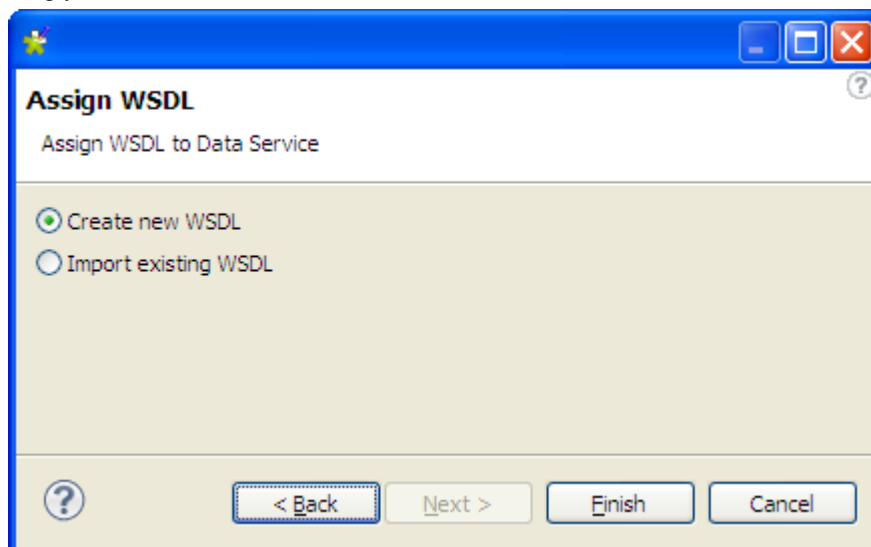
3. The dialog box displays to help you define the main properties of the new Service.

Enter the Service properties as the following:

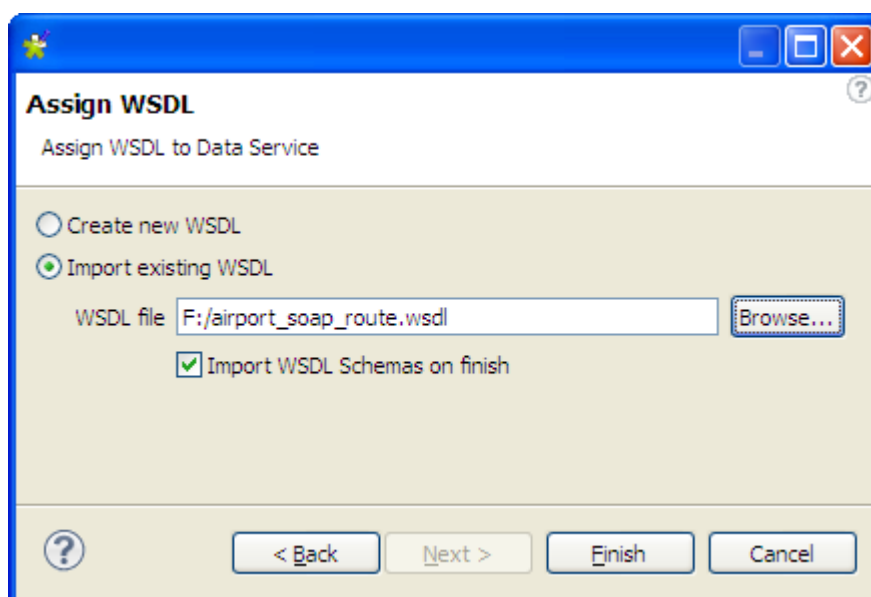
Field	Description
Name	the name of the new Service. A message comes up if you enter prohibited characters.
Purpose	Service purpose or any useful information regarding the Service use.
Description	Service description.
Author	a read-only field that shows by default the current user login.

Field	Description
Locker	a read-only field that shows by default the login of the user who owns the lock on the current Service. This field is empty when you are creating a Service and has data only when you are editing the properties of an existing Service.
Version	a read-only field. You can manually increment the version using the M and m buttons. For more information, see Section 7.5, “Managing Job and Route versions” .
Status	a list to select from the status of the Services you are creating.
Path	a list to select from the folder in which the Service will be created.

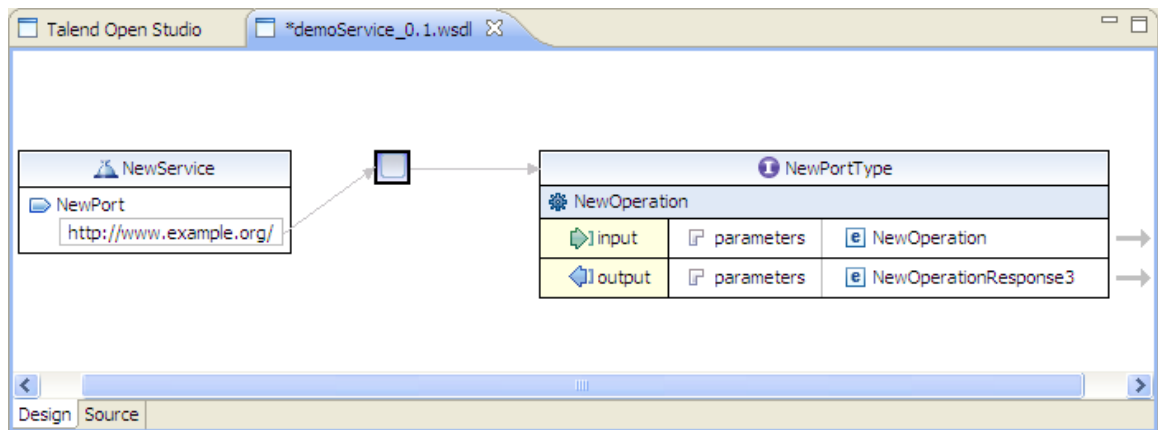
- Click **Next** to open the **Assign WSDL** view on the wizard. By default, the **Create new WSDL** option is selected allowing you to create a new WSDL file.



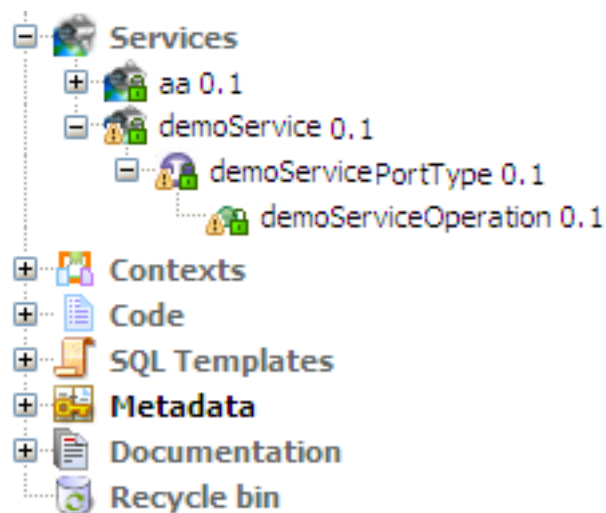
- To import an existing WSDL file, click the **Import existing WSDL** radio button. Click **Browse...** next to the **WSDL file** field to navigate to an existing WSDL file. Select the **Import WSDL Schemas on finish** check box if you want to retrieve and store the schema from the WSDL file in the **Metadata** folder of the repository tree view.



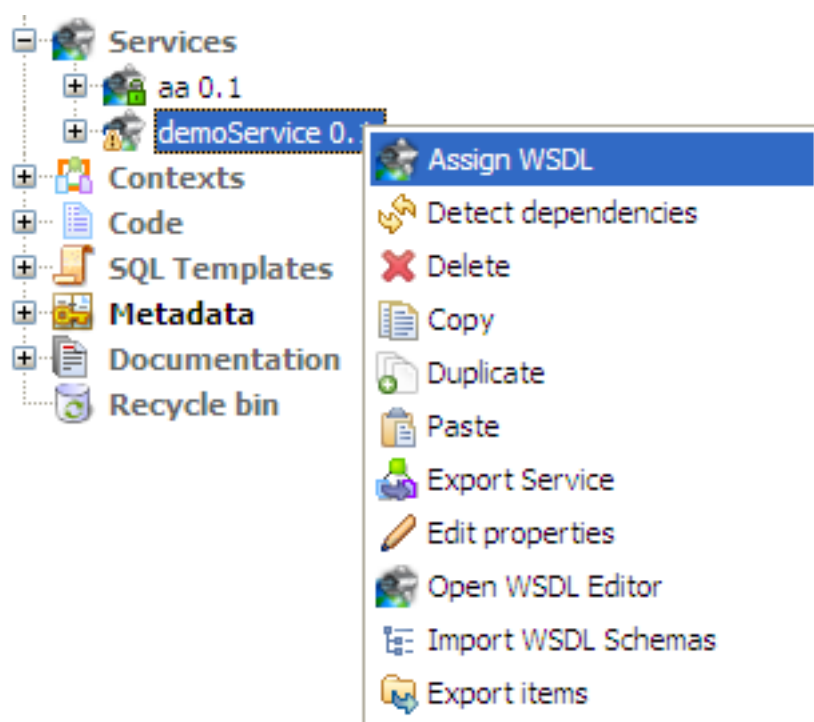
- Click **Finish** to close the wizard and the Service opens in the design workspace with a basic WSDL skeleton.



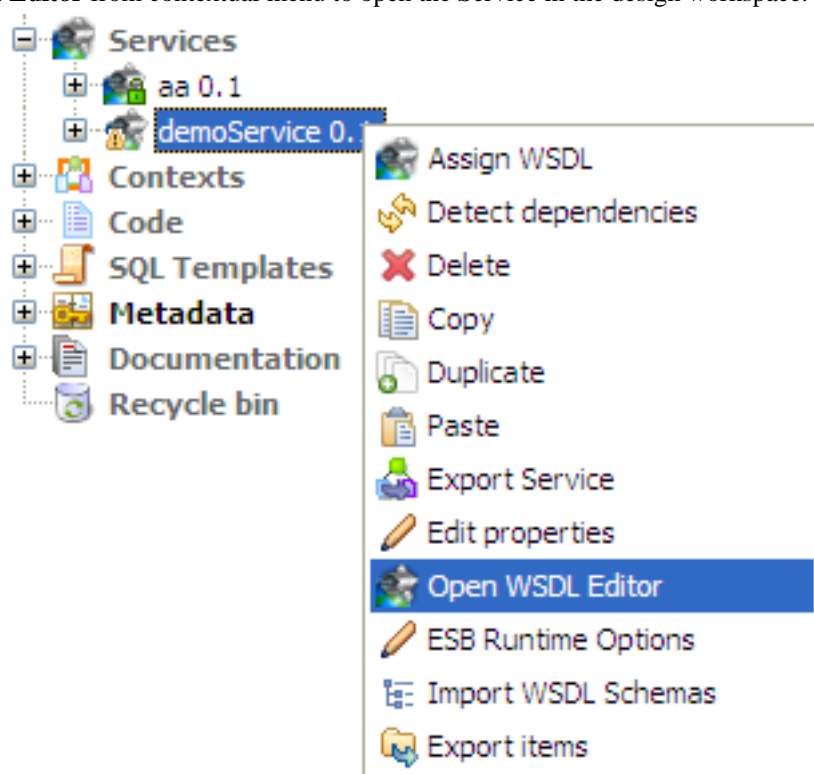
The Service you just created is listed under the **Services** node in the **Repository** tree view. The port type and the operation objects are displayed on the tree. The exclamation icon means that this defined Web service is not yet used.



7. You can also assign an existing WSDL file to a Service after the creation by right clicking on the Service from the **Repository** tree view and select **Assign WSDL** from contextual menu to show the **Assign WSDL** wizard.



8. Double click the Service you just created from the **Repository** tree view, or right click the Service and select **Open WSDL Editor** from contextual menu to open the Service in the design workspace.



9. Edit the WSDL file in the WSDL editor. For more information, see [Section 6.2.2, “How to edit a WSDL file”](#).
10. Create or assign one or more data service Jobs to the Service to implement the Web service. For more information, see [Section 6.2.3, “How to associate data service Jobs with a Service”](#).

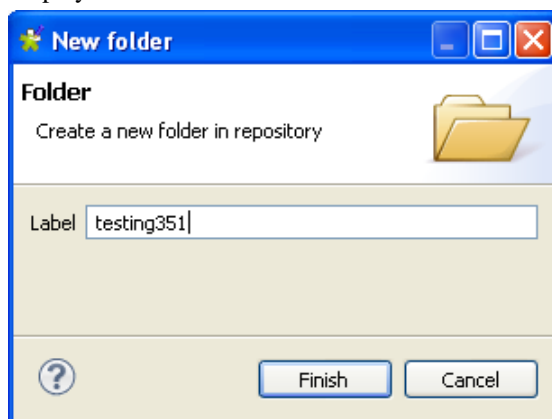


You can open one or more of the created Services by simply double-clicking the Service label in the **Repository** tree view.

To create different folders for your Services, complete the following:

1. In the **Repository** tree view, right-click **Services** and select **Create folder** from the contextual menu.

The **New folder** dialog box displays.



2. In the **Label** field, enter a name for the folder and then click **Finish** to confirm your changes and close the dialog box.

The created folder is listed under the **Services** node in the **Repository** tree view.

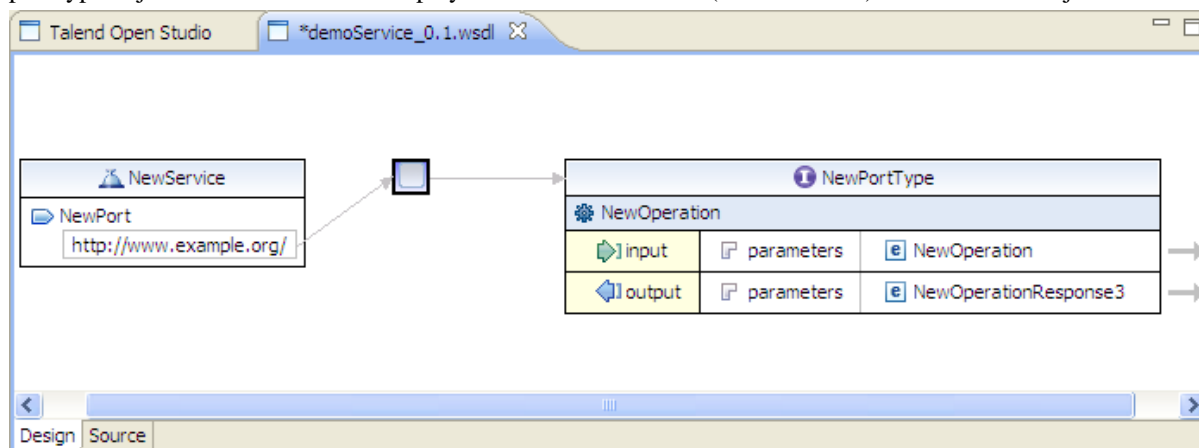


If you have already created Services that you want to move into this new folder, simply drag and drop them into the folder.

For a scenario showing how to create a real-life data service, see [Appendix B, Theory into practice: Data service and routing examples](#).

6.2.2. How to edit a WSDL file

Talend Open Studio for ESB provides a graphical way to browse and edit your WSDL file in the WSDL editor. Each type of top level WSDL object is shown within a tabular view (for example, service, binding, port type). Each tabular view contains one or more rows that represent the structure of the object. The service, binding, and port type objects are linked. A line displays to denote a reference (or association) between these objects.



The above screenshot gives a basic WSDL skeleton which contains:

- a **service**, used to aggregate a set of related ports which specify addresses for bindings, thus defining a single communication endpoint.
- a **binding**, specifies concrete protocol and data format specifications for the operations and messages defined by a particular port type.
- a **port type**, a set of abstract operations that each refer to an input message and output messages.

The WSDL editor allows you to edit a WSDL file in the **Properties** view.

The **Properties** view is located on the lower part of the designing editor of *Talend Open Studio for ESB* and displays a list of attributes and editable attribute values of a selected WSDL object and contains the following tabs to edit:

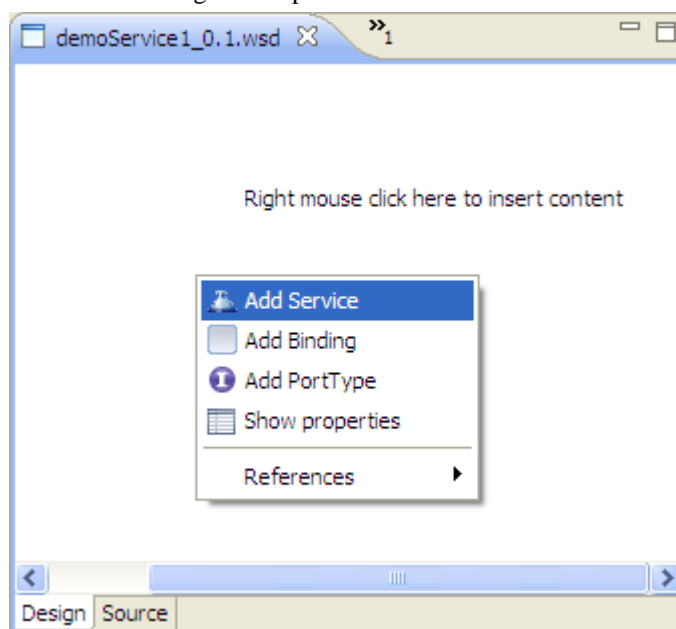
- **General** tab, displays a list of object attributes.
- **Documentation** tab, specifies the information you want the user to read.
- **Extensions** tab, used to add extension components.

6.2.2.1. How to add a service

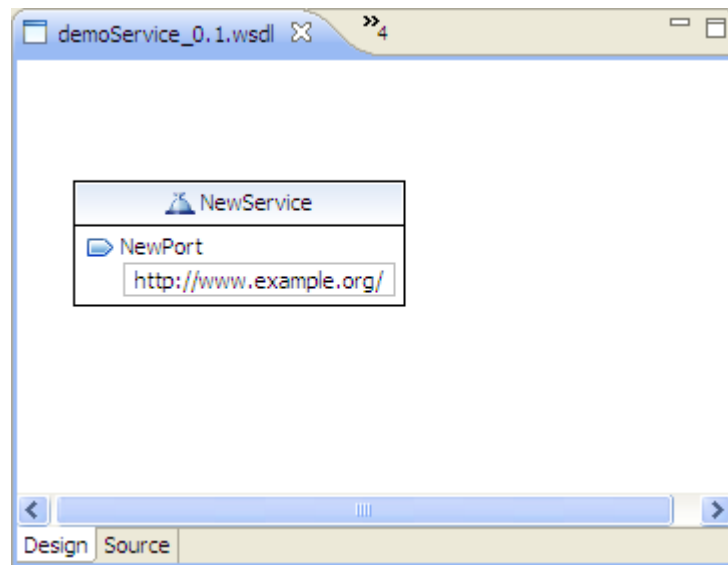
The service object is a collection of related ports and specifies the location of the service.

To create a new service, complete the following steps:

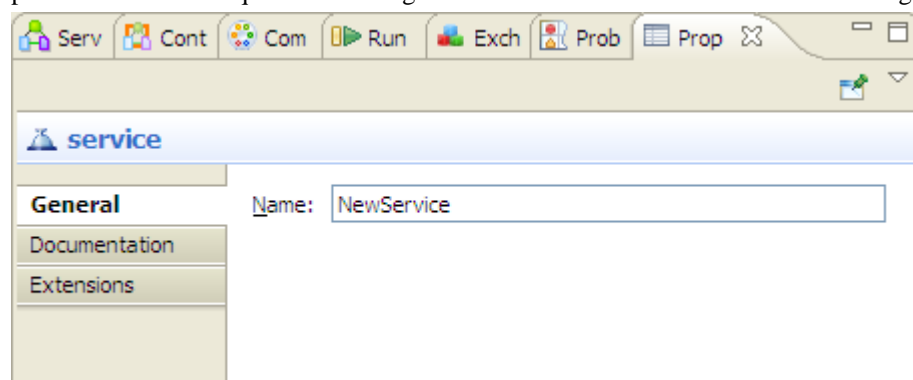
1. Right click in the blank area of the design workspace to show the contextual menu and select **Add Service**.



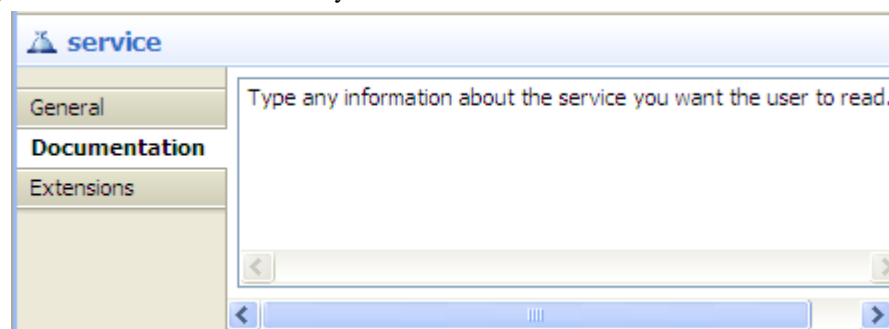
A new service object is added in the design workspace with a port.



2. Click the service object to show its **Properties** view.
3. In the **Properties** view, click the **General** tab. In the **Name** field, type in the name of the service. The name of the service provides it with a unique name among all the services defined within in the enclosing WSDL file.



4. Type in any information about the service you want the user to read in the **Documentation** tab.



5. To manage extensions, click the **Extensions** tab. You can either add, sort, or remove extensions.

You can add ports to your service. A port defines an individual endpoint by specifying a single address for a binding. For more information, see [Section 6.2.2.2, “How to add a port to a service”](#).

6.2.2.2. How to add a port to a service

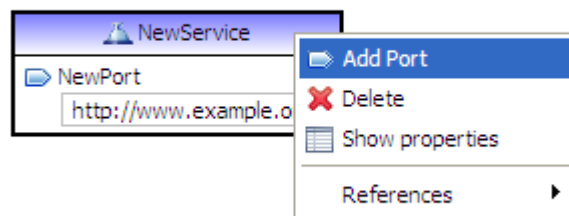
A port defines an individual endpoint by specifying a single address for a binding. The port contains a **Binding** attribute that references a binding and an address element that provides a specification for the endpoint.

Services are used to group sets of related ports together. Ports within a service have the following relationship:

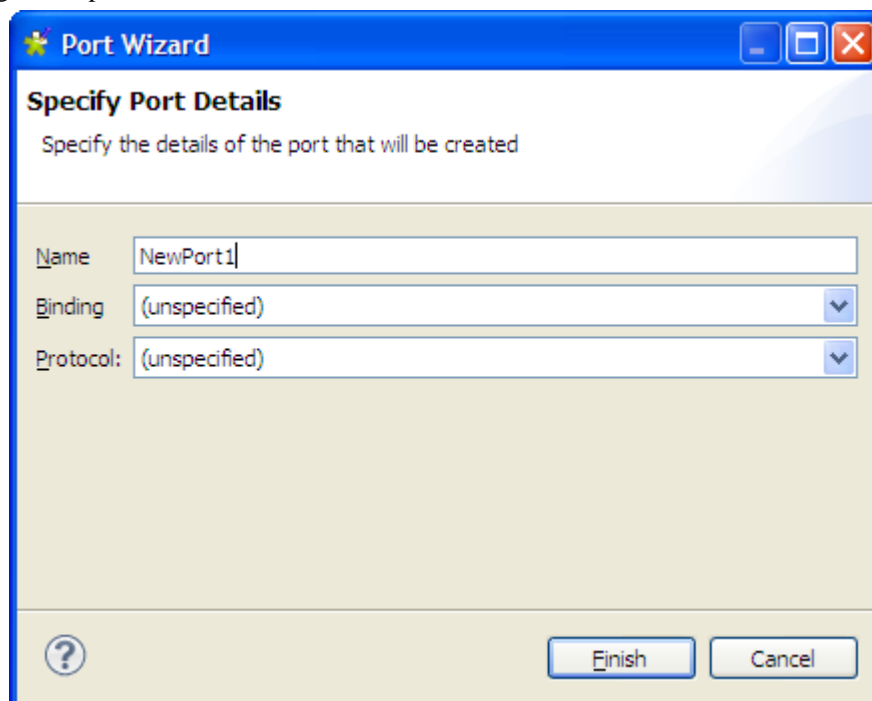
- None of the ports communicate with each other (for example, the output of one port is not the input of another).
- If a service has several ports that share a port type, but employ different bindings or addresses, the ports are alternatives. Each port provides semantically equivalent behavior (within the transport and message format limitations imposed by each binding).
- You can determine a service's port types by examining its ports. Using this information, a user can determine if a given machine supports all the operations needed to complete a given task.

To add a port to a service, complete the following steps:

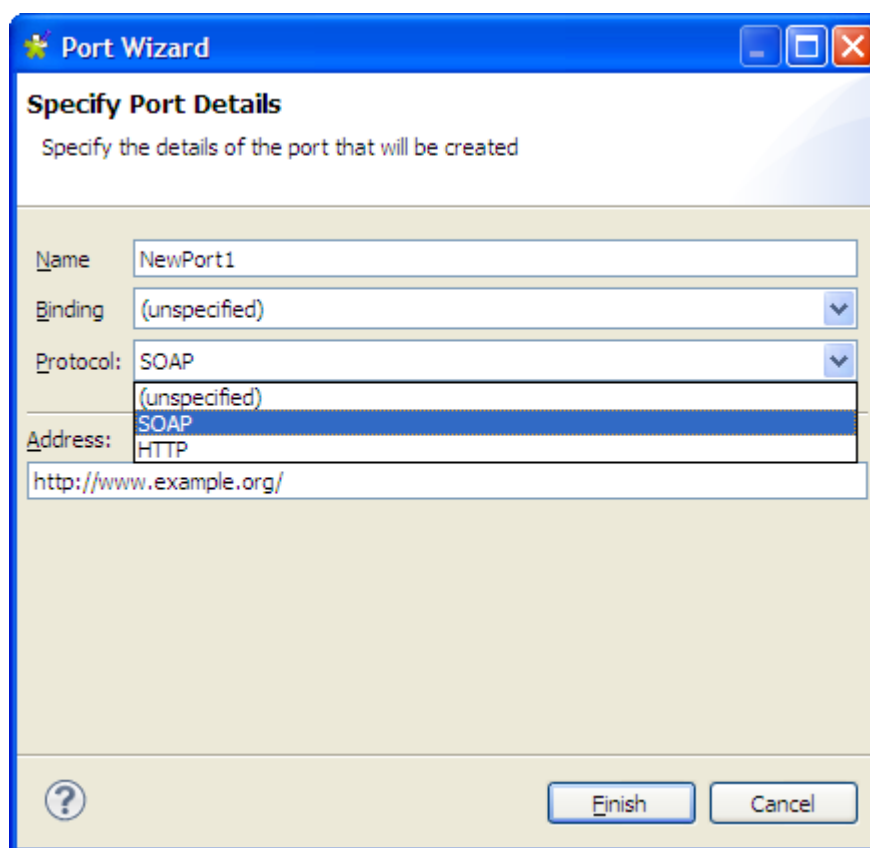
1. In the design workspace, right click the service you want to add a port to and select **Add Port** in the contextual menu.



2. The **Port Wizard** displays. Fill in the **Name** for the port. The name of the port should provide it with a unique name among all the ports defined within the service.

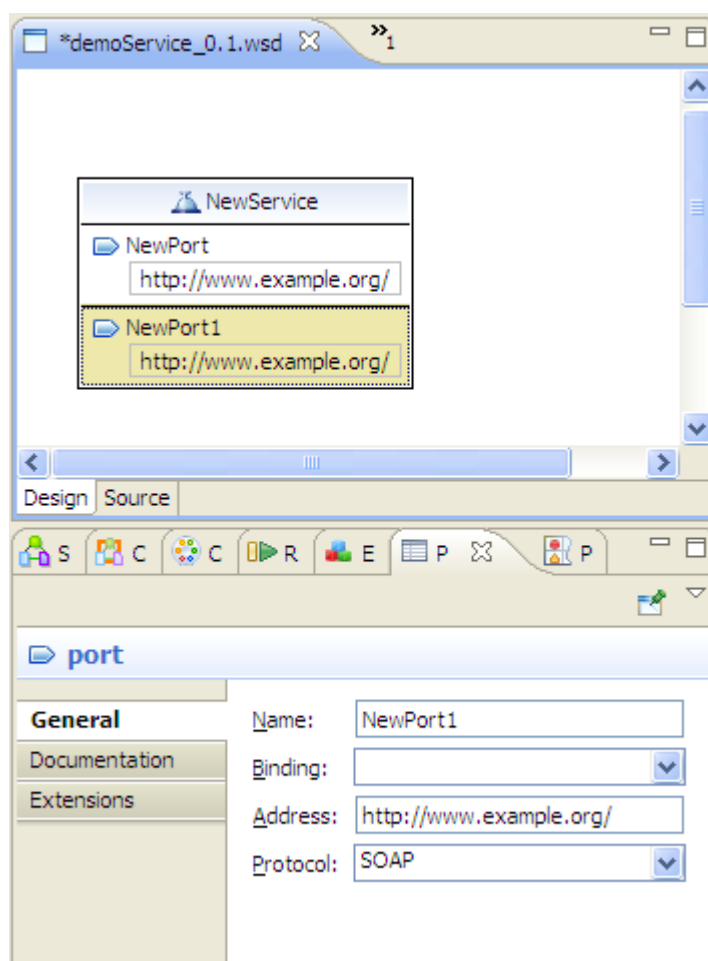


3. Select a **Binding** for the port. For more information on how to set a binding, see [Section 6.2.2.3, “How to set a binding”](#).
4. Select a Protocol for the port from the **Protocol** list and enter the address of the port into the **Address** field that appears.

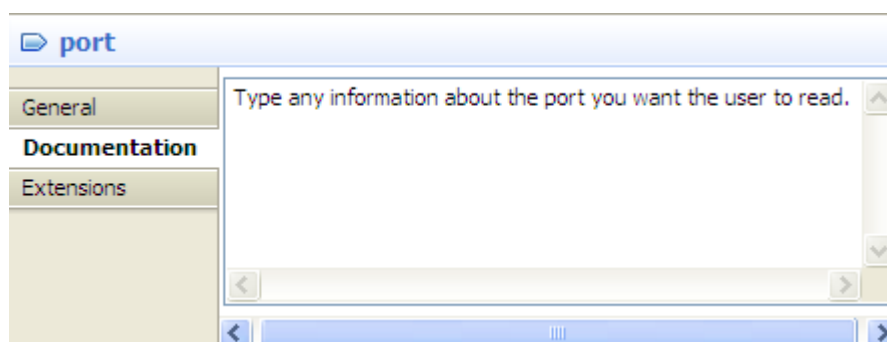


The image shows a 'Port Wizard' dialog box with the title 'Specify Port Details'. The instructions inside say 'Specify the details of the port that will be created'. There are four input fields: 'Name' with the text 'NewPort1', 'Binding' with a dropdown menu showing '(unspecified)', 'Protocol' with a dropdown menu showing 'SOAP' and a list of options including '(unspecified)', 'SOAP', and 'HTTP' (which is currently selected), and 'Address' with the text 'http://www.example.org/'. At the bottom right are 'Finish' and 'Cancel' buttons, and at the bottom left is a help icon (a question mark in a circle).

5. Click **Finish** to validate the creation. The port is added to the service and its **Properties** view displays.



6. To select the new port, click the port in the service object. You can change the information you entered about this port any time in the **General** tab of its **Properties** view.
7. Type any information about the port you want the user to read in the **Documentation** tab of the **Properties** view.



8. To manage extensions, click the **Extensions** tab. You can either add, sort, or remove extensions.

You can create a new binding for your port or re-use an existing one. A binding defines the message format and protocol details for operations and messages defined by a particular port type. For more information, see [Section 6.2.2.3, “How to set a binding”](#).

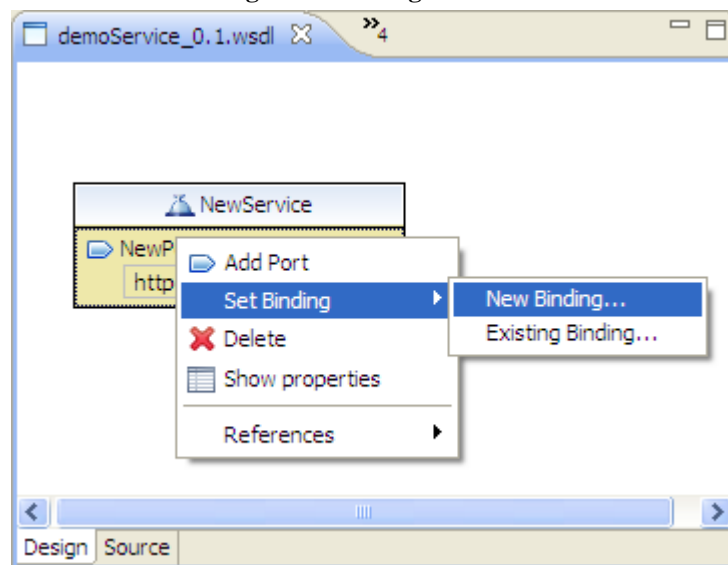
6.2.2.3. How to set a binding

A binding is a top level WSDL object that provides a concrete specification regarding the transmission of messages to and from a Web service. A binding references exactly one port type. The structure of a binding corresponds very closely to that of the port type. The binding contains extensibility elements (for example, SOAP, HTTP, and MIME) that specify protocol specific details. Each port within a service references exactly one binding.

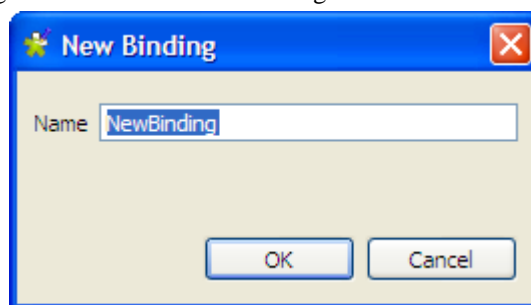
You can create a binding by right clicking in any blank area in the design workspace and selecting **Add Binding** in the contextual menu. You can create a new binding or reuse an existing one.

How to create a new binding

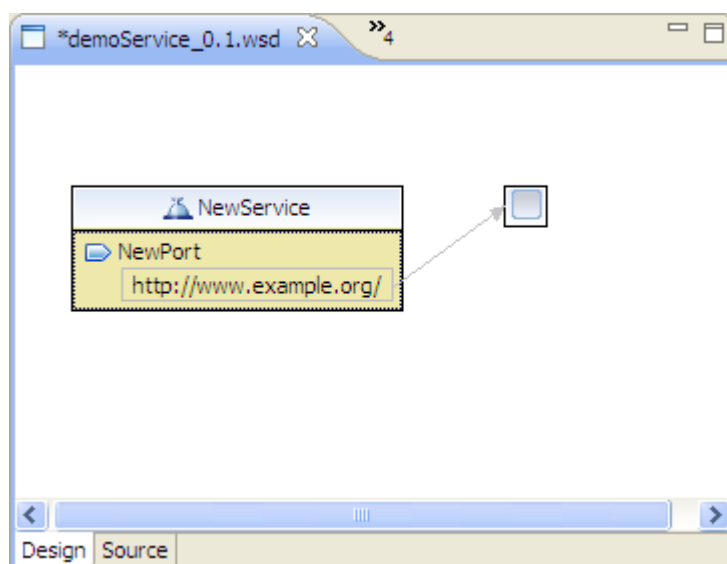
1. Right click the port and select **Set Binding>New Binding**.



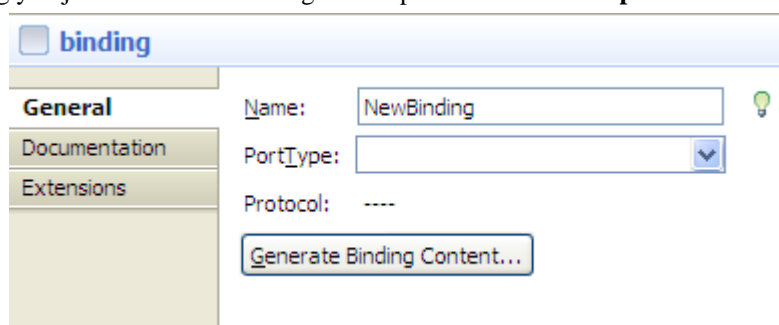
2. The **New Binding** dialog box displays. Type in the name of the binding in the **Name** field . The name should be unique among all bindings defined within the enclosing WSDL file. Click **OK**.



The new binding is created in the design workspace and connected to the port.

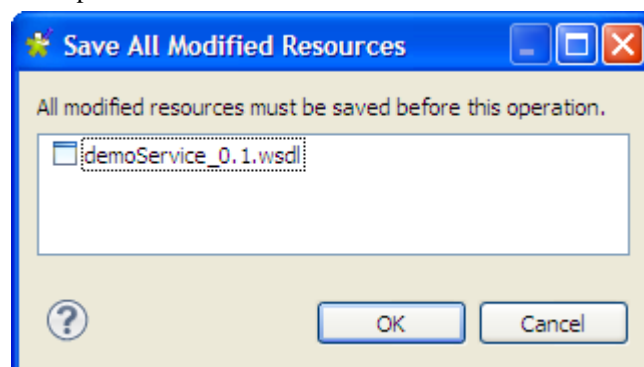


3. Click the binding you just created in the design workspace to show its **Properties** view.

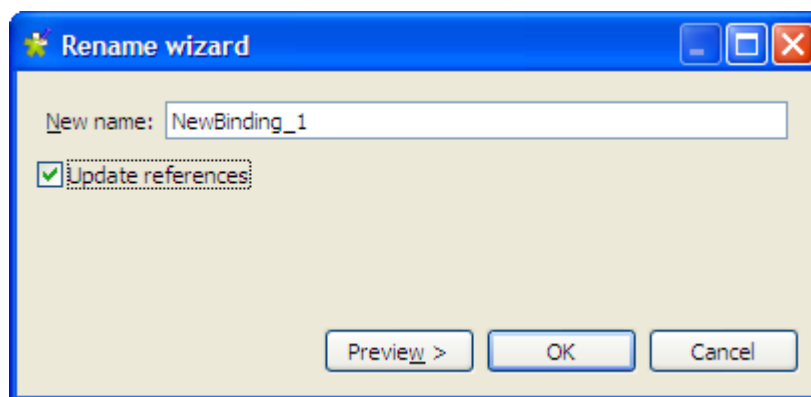


4. To specify the name of the binding, type it in the **Name** field or click the bulb icon beside the **Name** field to invoke the rename refactoring.

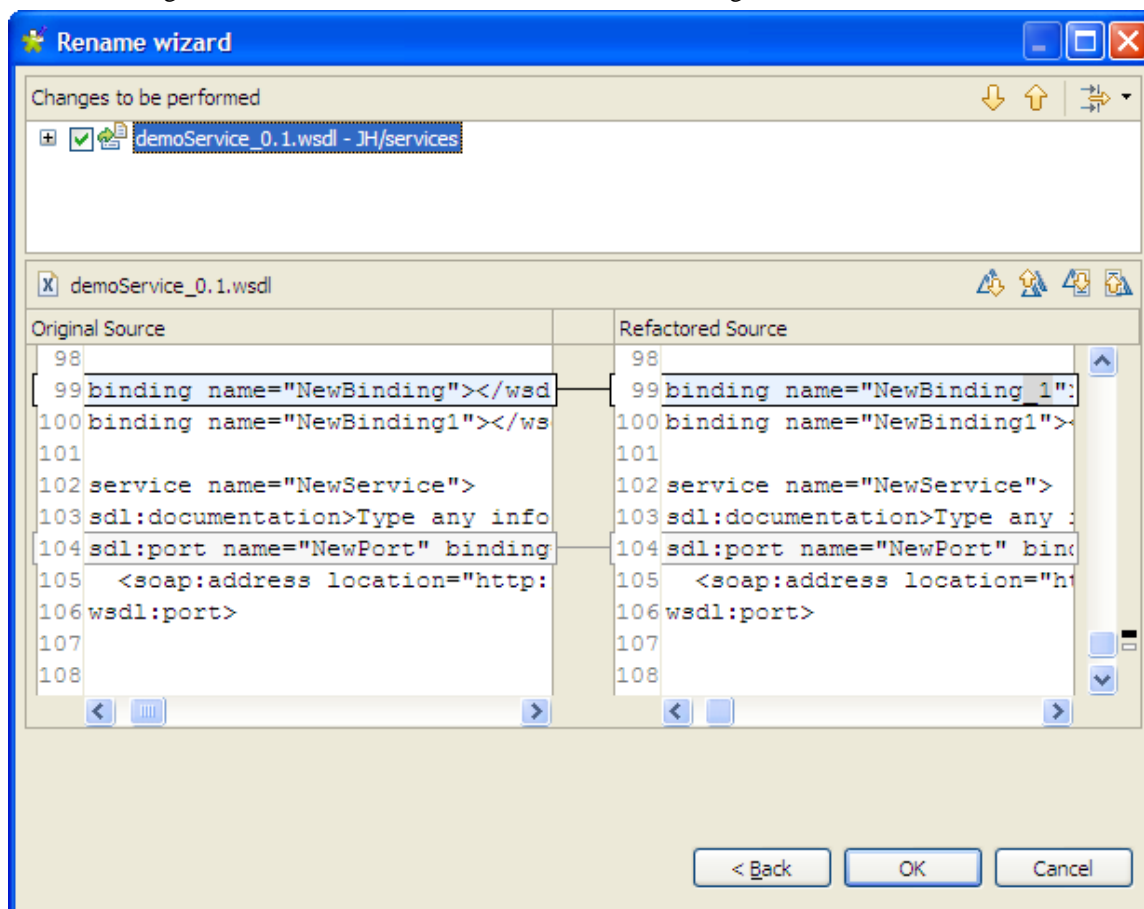
When clicking on the bulb icon, a **Save All Modified Resources** dialog box displays prompting you to save all modified resources before proceed.



Click **OK** and the **Rename wizard** appears.

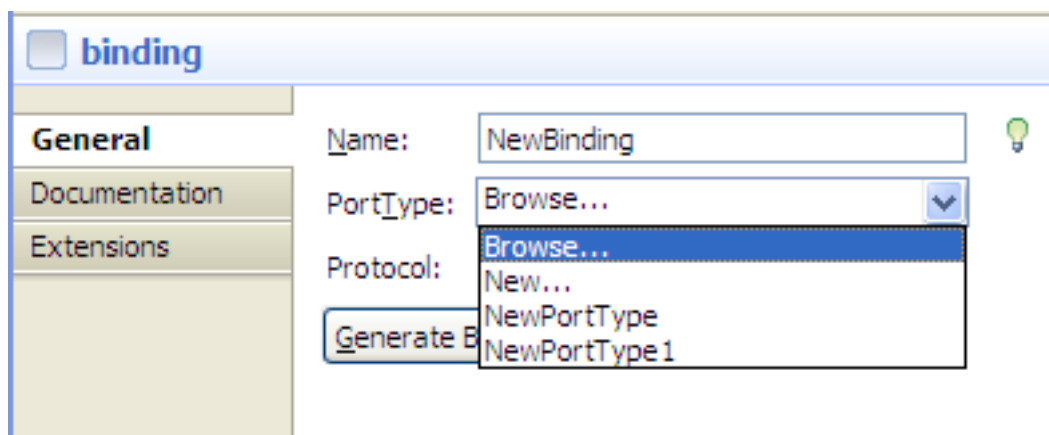


Type in a new name in the **New name** field. Select the **Update references** check box to propagate this change to the enclosing WSDL file. Click **Preview** to have a look at the original source and the refactored source.

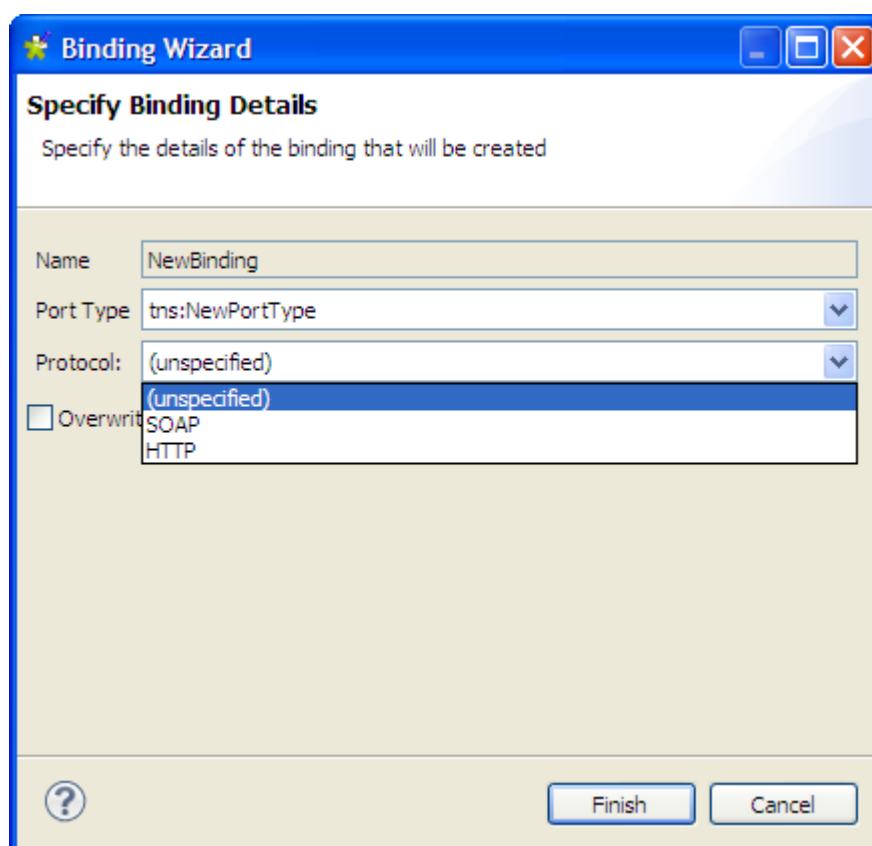


Click **OK** to validate the change and close the dialog box.

5. Click the **PortType** field to set the port type of the binding. You can select the port type you want from the list which contains the port types in your current file, select **New...** to open the **New PortType** wizard and create a new port type, or select **Browse...** to open the **Specify Port Type** wizard. For more information, see [Section 6.2.2.4, “How to set a port type”](#).

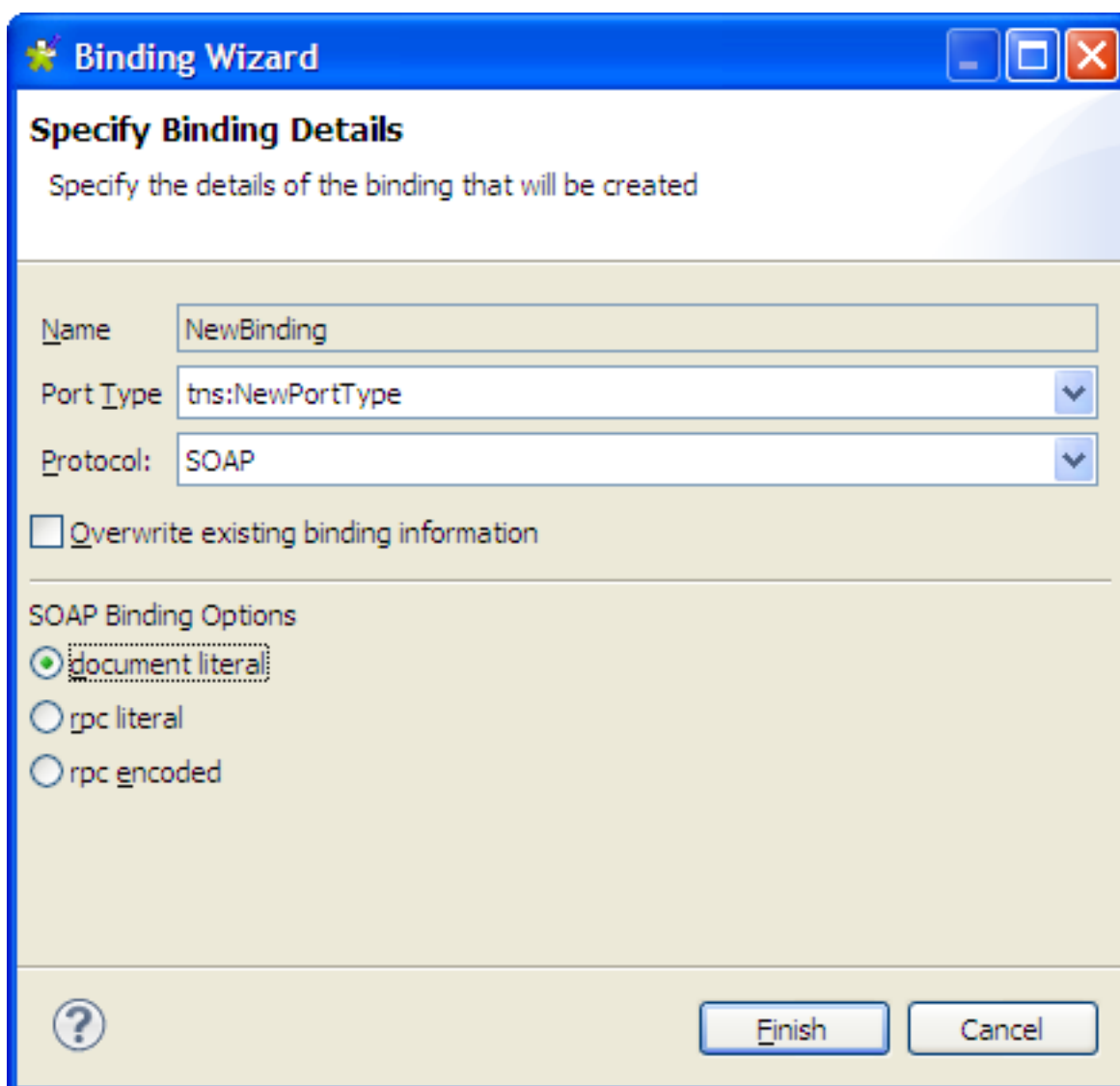


- Click the **Generate Binding Content...** button to show the **Binding Wizard** and specify the details of the binding.



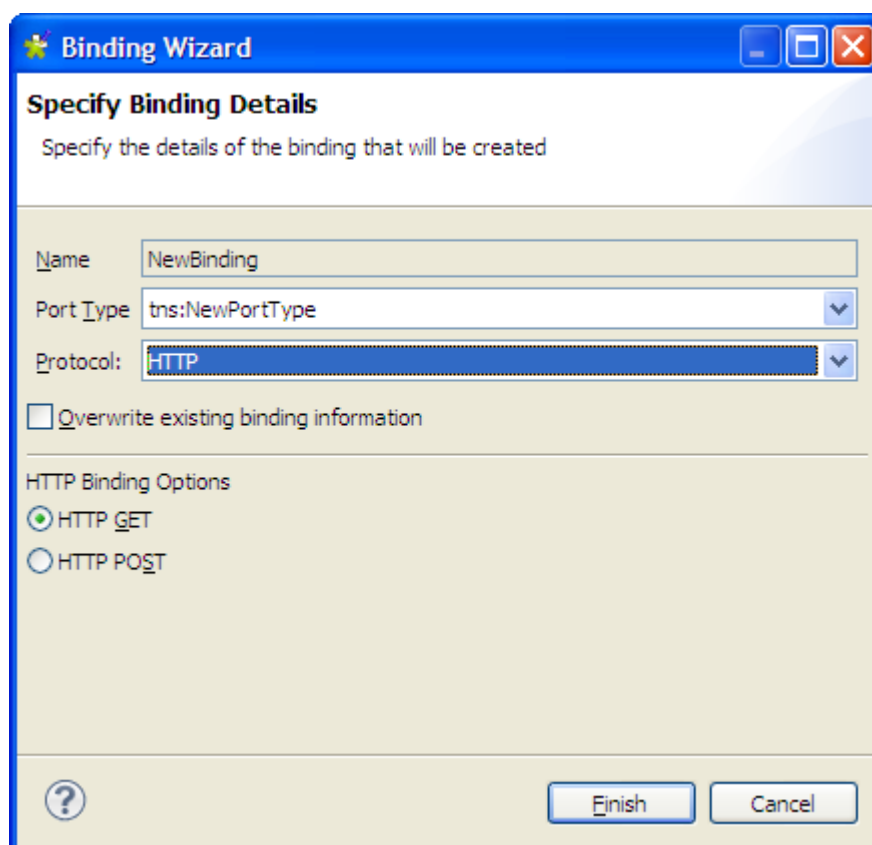
The **Name** field shows the name of the binding to be specified. The **PortType** field shows the port type that the binding references. You can also select the port type you want from the list which contains the port types in your current file. Select the binding options you want to use in the **Protocol** list. The options are **SOAP** and **HTTP**. Use the **SOAP** protocol when you want to exchange structured and typed information. Use the **HTTP** protocol when you want your application client to just request or update information.

If you select **SOAP** you can then select the encoding style you want to use:



- **document literal.** Document style messages, literal encoding. Use this style of binding when you want to send SOAP messages that can be validated by an XML validator. All the data types in the SOAP message body are defined in a schema, so the WSDL parts must point to schema elements.
- **rpc literal.** RPC style messages, literal encoding. Use this style of binding when you want to specify the operation method names in your SOAP messages so a server can dispatch the specified methods. Data types must be defined, so the WSDL parts must point to XSD types.
- **rpc encoded.** RPC style messages and SOAP encoding. Use this style of binding when you want to encode data graphs in your SOAP messages so a server can deserialize the object data. Data types must be defined, so the WSDL parts must point to XSD types.

If you select **HTTP** you can select whether to create an HTTP getter or setter.



- **HTTP GET.** A GET request fetches data from a Web server based on an URL value and a set of HTTP headers. Use this method when you want to retrieve information specified in the request.
- **HTTP POST.** A POST request sends additional data to the server, specified after the URL and the headers. Use this method when you want to send data enclosed in the body of the request.

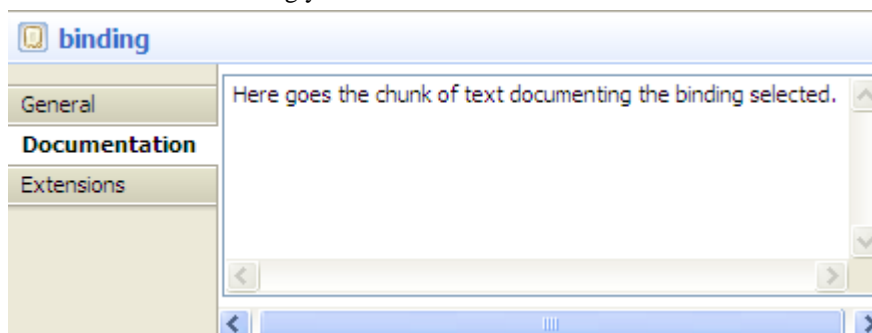
Select the **Overwrite existing binding information** check box if you are modifying an existing binding.



Changes of service elements are not automatically propagated to bindings. You need to regenerate the binding content and select the **Overwrite existing binding information** check box in the binding wizard to reflect changes in the WSDL.

Click **Finish** to validate your specification and close the wizard.

7. Type any information about the binding you want the user to read in the **Documentation** tab.

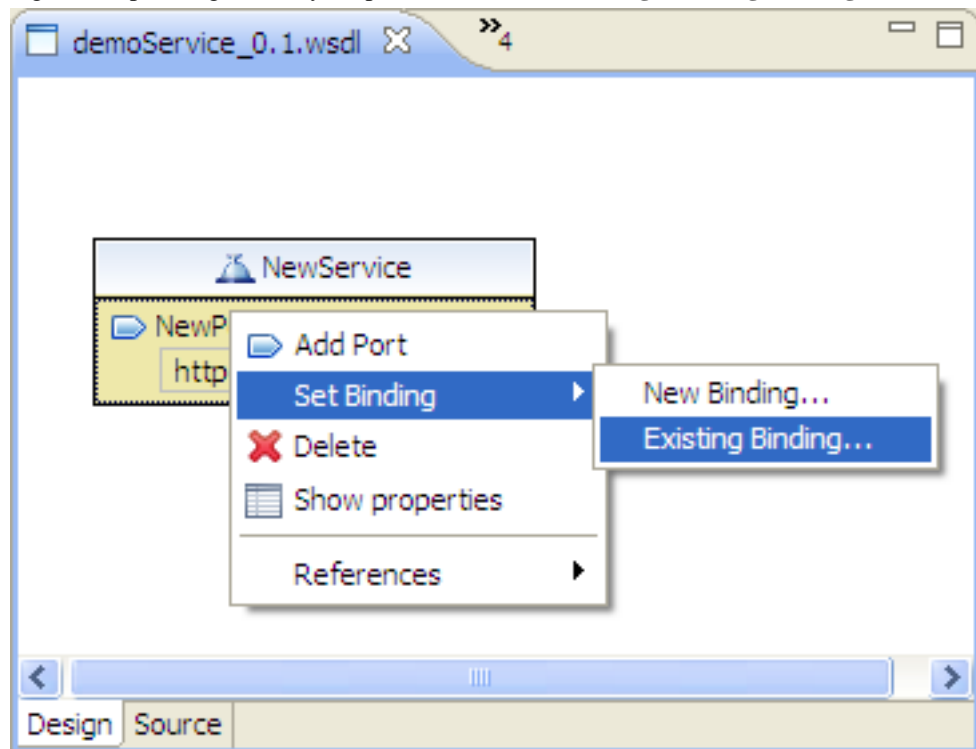


8. To manage extensions, click the **Extensions** tab. You can either add, sort, or remove extensions.

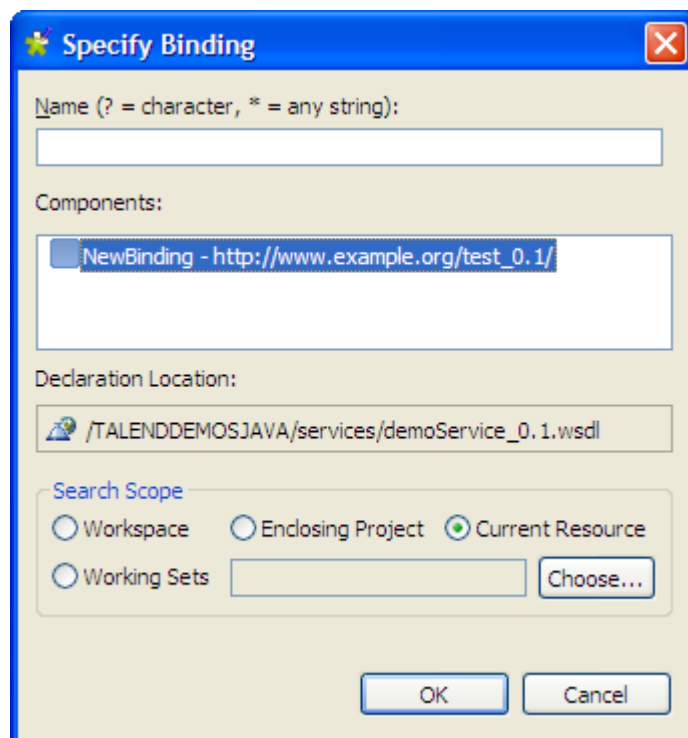
How to reuse an existing binding

To reuse an existing binding for your port, complete the following steps:

1. In the design workspace, right-click your port and click **Set Binding>Existing Binding**.



2. The **Specify Binding** wizard displays. In the **Name** field, type a search string to filter the list or leave the field blank to view all.



3. You can change the scope of the list by selecting one of the following options:
 - **Workspace:** lists the bindings available in your workspace.

- **Enclosing Project:** lists the bindings available in the project that contains your file.
 - **Current Resource:** lists the binding types available in your current file.
 - **Working Sets:** lists the bindings available in a specified set of files. To specify a working set, click **Choose**. You can select files from an existing working set or create a new working set. Once you have selected the files, click **OK**.
4. The bindings will be listed in the **Components** area. The **Declaration Location** field shows where the selected binding is located. Select the binding you want and click **OK**. The binding is connected to the port in the design workspace.
 5. For more information on how to modify the binding in the **Properties** view, see [the section called “How to create a new binding”](#).

6.2.2.4. How to set a port type

A port type is a named set of abstract operations and the abstract messages involved. Each operation refers to an input message and output messages. A port type is referenced by a binding object. Each binding references exactly one port type. Since each port refers to exactly one binding, each port has exactly one port type.

You can create a port type by right clicking in any blank area in the design workspace and selecting **Add PortType**. As a convenience you can also add a port type via a binding. This allows you to create a port type and modify the binding to reference the new port type in one action.

You can either create a new port type for your binding, or reuse an existing one.

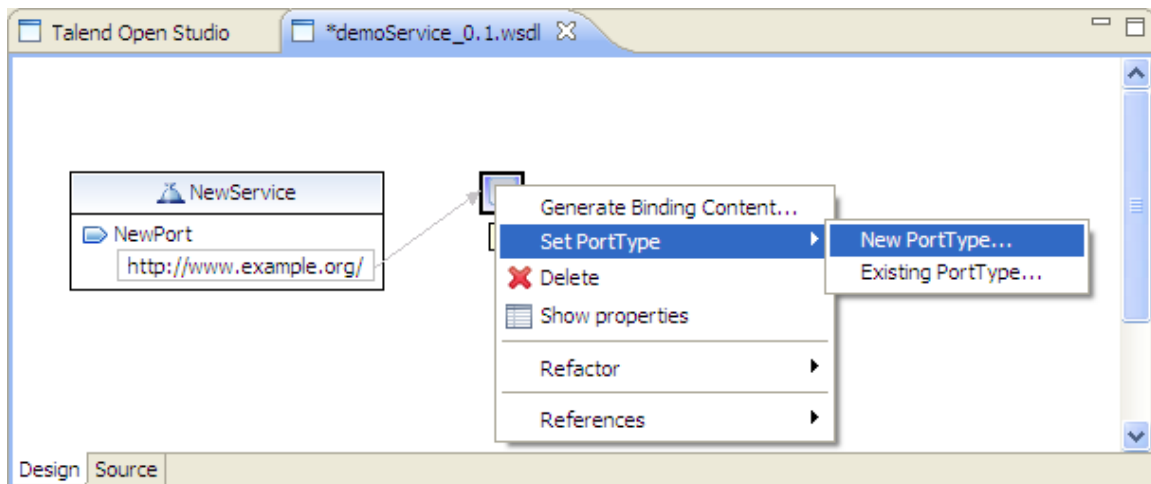


Changes of service elements are not automatically propagated to bindings. You need to regenerate the binding content and select the **Overwrite existing binding information** check box in the binding wizard to reflect changes in the WSDL. For more information, see [Section 6.2.2.3, “How to set a binding”](#).

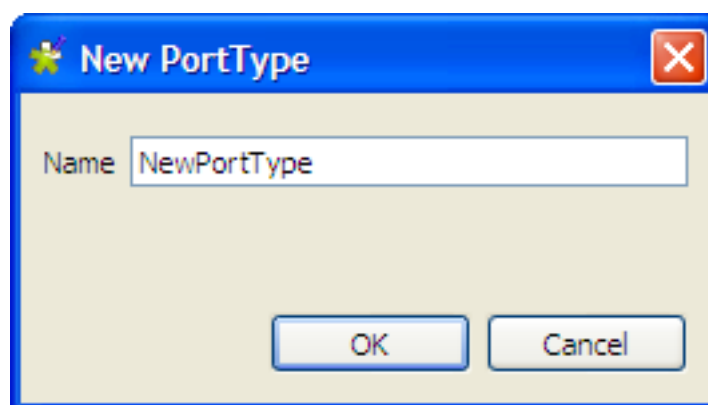
How to create a new port type for a binding

If you want to create a new port type for your binding, complete the following steps:

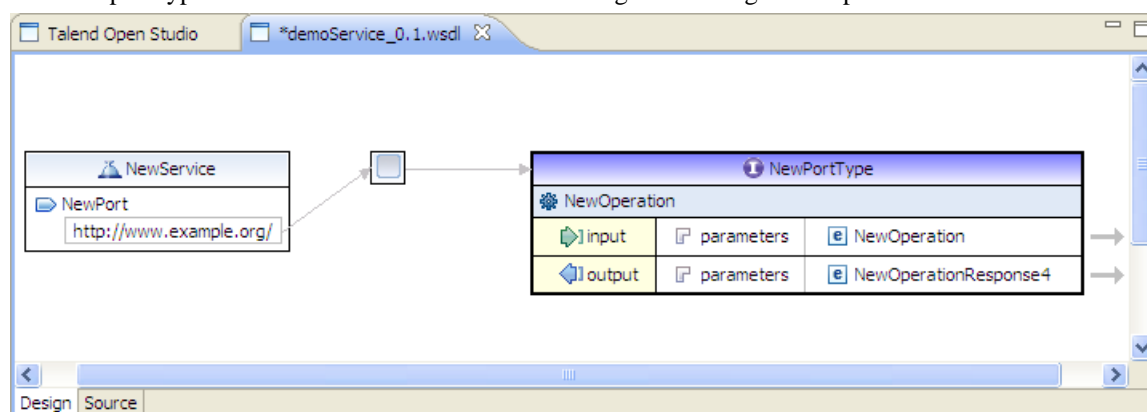
1. In the design workspace, right click on a binding object and click **Set PortType>New PortType**.



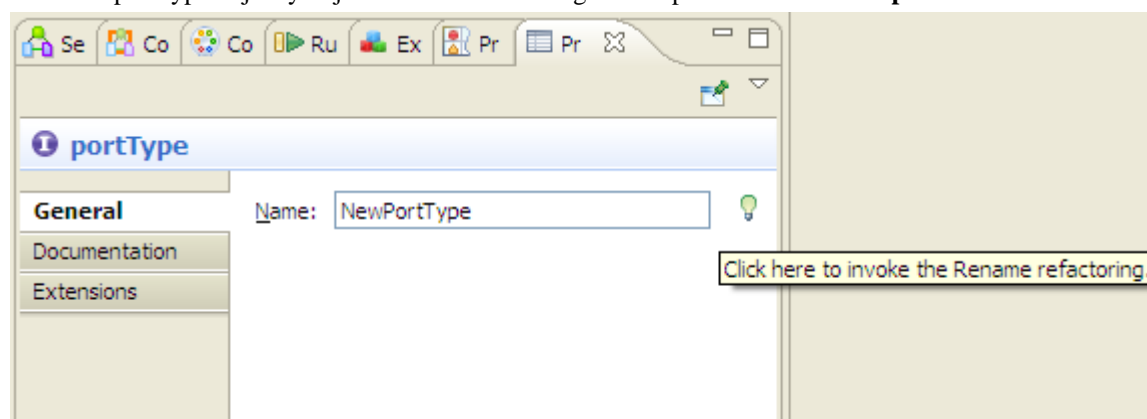
2. The **New PortType** wizard displays. Type the name of the port type in the **Name** field. The name should be unique name among all port types defined within the enclosing WSDL file. Click **OK**.



The new port type is created and connected to the binding in the design workspace.

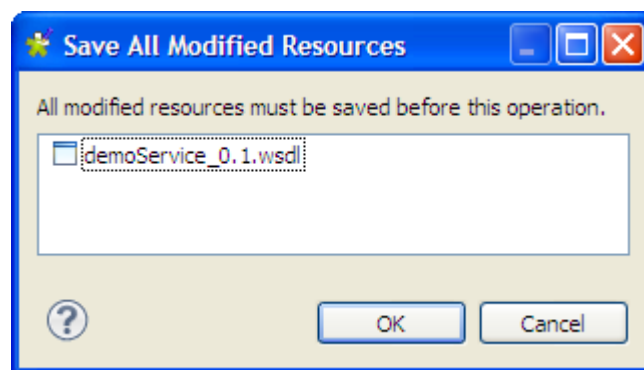


- Click the port type object you just created in the design workspace to show its **Properties** view.

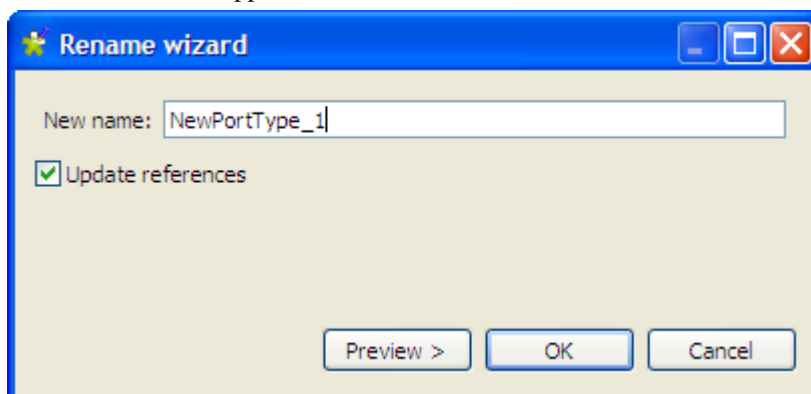


- The **General** tab allows you to define the name of the port type. To specify the name of the port type, type it in the **Name** field or click the bulb icon beside the **Name** field to open the **Rename** wizard.

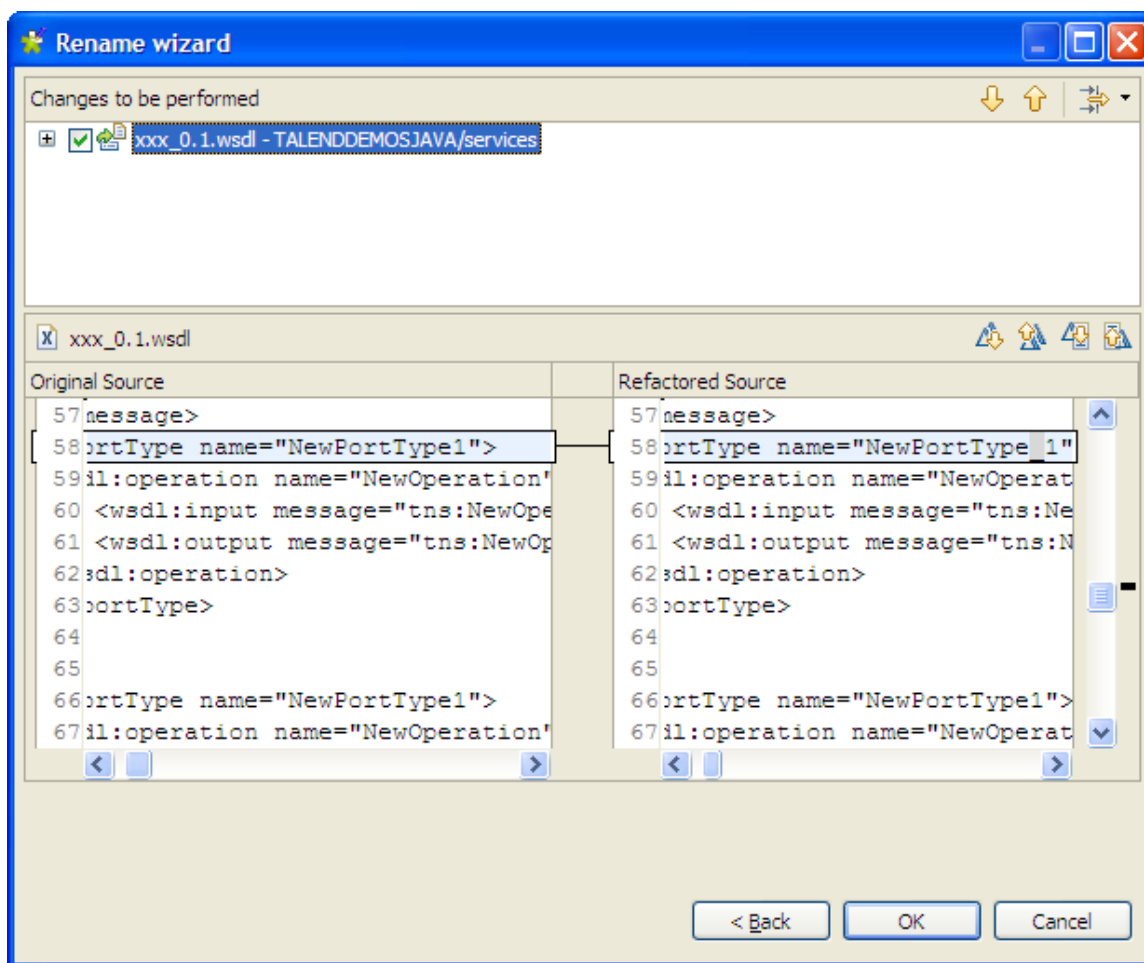
When clicking on the bulb icon, a **Save All Modified Resources** dialog box displays prompting you to save all modified resources before proceed.



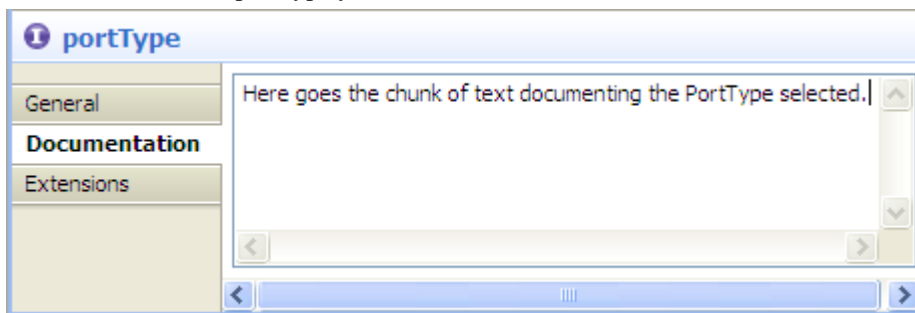
Click **OK** and the **Rename wizard** appears.



Type in a new name in the **New name** field. Select the **Update references** check box to propagate this change to the enclosing WSDL file. Click **Preview** to have a look at the original source and the refactored source. Click **OK** to validate the renaming and close the wizard.



5. Type any information about the port type you wish the user to read in the **Documentation** tab.



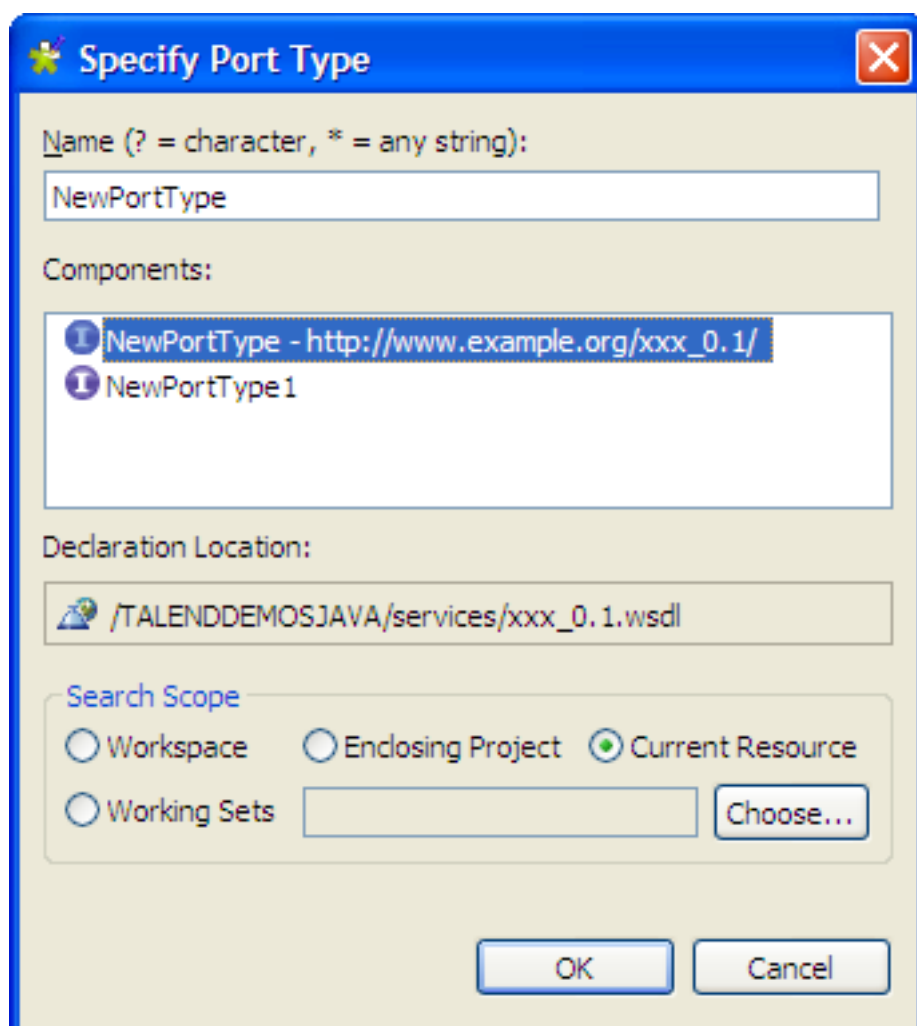
6. To manage extensions, click the **Extensions** tab. You can either add, sort, or remove extensions.

Once you have created a port type, you can add operations to it to can send and receive messages. For more information, see [the section called “How to create a new binding”](#).

How to reuse an existing port type

To reuse an existing port type for your binding, complete the following steps:

1. In the design view, right click on a binding object and click **Set PortType>Existing PortType**.
2. The **Specify Port Type** wizard displays. In the **Name** field, type a search string to filter the list or leave the field blank to view all.



3. You can change the scope of the list by selecting one of the following options:
 - **Workspace:** lists the port types available in your workspace.
 - **Enclosing Project:** lists the port types available in the project that contains your file.
 - **Current Resource:** lists the port types available in your current file.
 - **Working Sets:** lists the port types available in a specified set of files. To specify a working set, click **Choose**. You can select files from an existing working set or create a new working set. Once you have selected the files, click **OK**.
4. The existing port types will be listed in the **Components** area. The **Declaration Location** field shows where the selected port type is located. Select the port type you want and click **OK**.
5. For more information on how to modify the port type in the **Properties** view, see [the section called “How to create a new port type for a binding”](#).

6.2.2.5. How to add an operation

An operation names the operation and lists the expected inputs and outputs. The operation element may also contain a fault sub-element that describes any error data that the operation may return.

Using an operation, you can declare four transmission primitives that an endpoint can support:

- **One-way**: the endpoint receives a message.
- **Request-response**: the endpoint receives a message, and sends a correlated message.
- **Solicit-response**: the endpoint sends a message, and receives a correlated message.
- **Notification**: the endpoint sends a message.

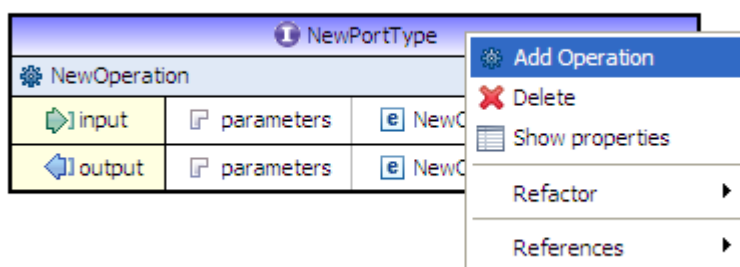
You can add an operation to a port type or a port binding.



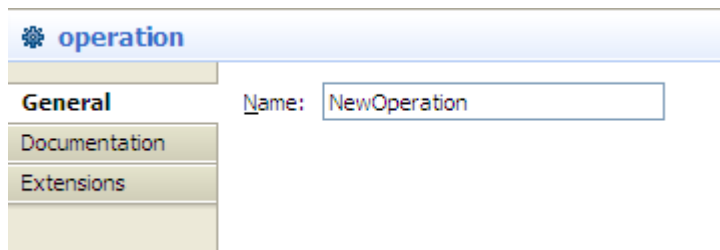
Changes of service elements are not automatically propagated to bindings. You need to regenerate the binding content and select the **Overwrite existing binding information** check box in the binding wizard to reflect changes in the WSDL. For more information, see [Section 6.2.2.3, “How to set a binding”](#).

To add an operation to a port type, complete the following steps:

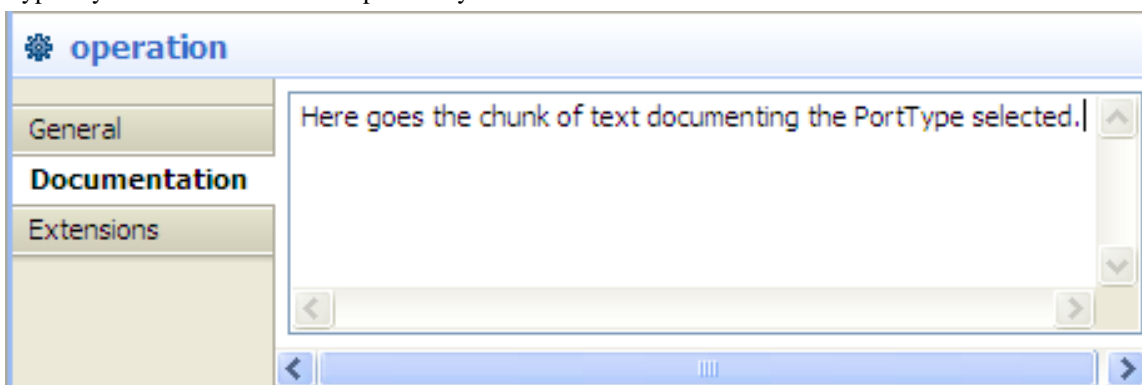
1. In the design workspace, right-click your port type. Click **Add Operation**.



2. The **Properties** view of the new operation opens. In the **General** tab, type the name of the operation in the **Name** field.



3. Type any information about the operation you wish the user to read in the **Documentation** tab.



4. To manage extensions, click the **Extensions** tab. You can either add, sort, or remove extensions.
5. Right click on the operation to add input, output or fault objects. Depending on an operation's inputs and outputs it can be classified as follows:
 - **One way operation**: input

- **Request response operation:** input, output
- **Solicit response operation:** output, input
- **Notification operation:** output

6.2.2.6. How to add a message

Messages represent an abstract definition of the data being transmitted. A message consists of logical parts, each of which is associated with a definition within some type system. WSDL messages are top level objects that can be referenced by an operation's input, output and fault elements (within a port type).

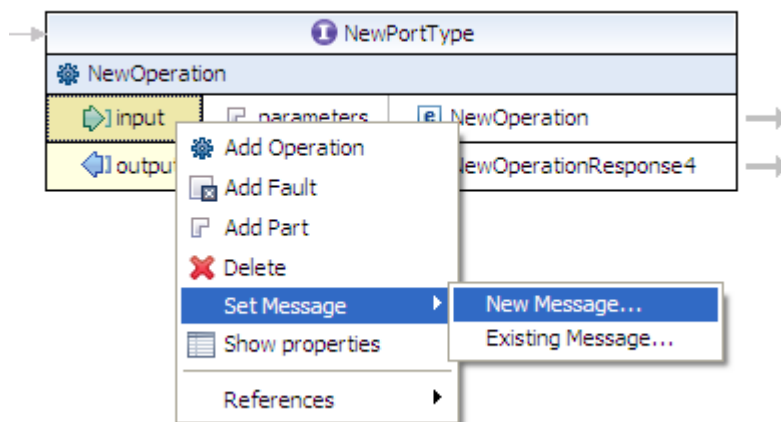


Changes of service elements are not automatically propagated to bindings. You need to regenerate the binding content and select the **Overwrite existing binding information** check box in the binding wizard to reflect changes in the WSDL. For more information, see [Section 6.2.2.3, “How to set a binding”](#).

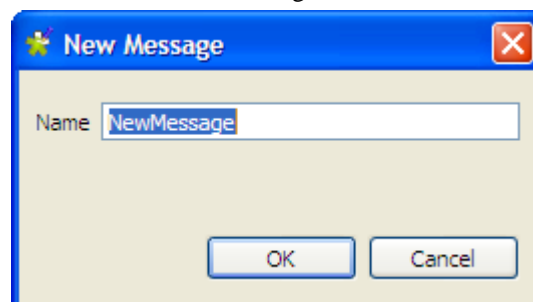
How to create a new message

To create a new message for your input, output, or fault element, complete the following steps:

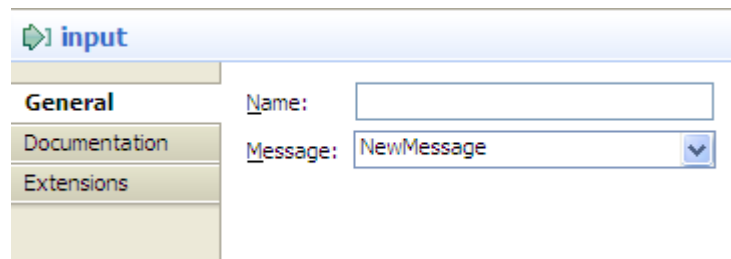
1. In the design workspace, right-click your input, output, or fault element and click **Set Message>New Message**.



2. The **New Message** wizard displays. Type the name of the message in the **Name** field. This name should be unique among all messages defined within the enclosing WSDL file. Click **OK**.



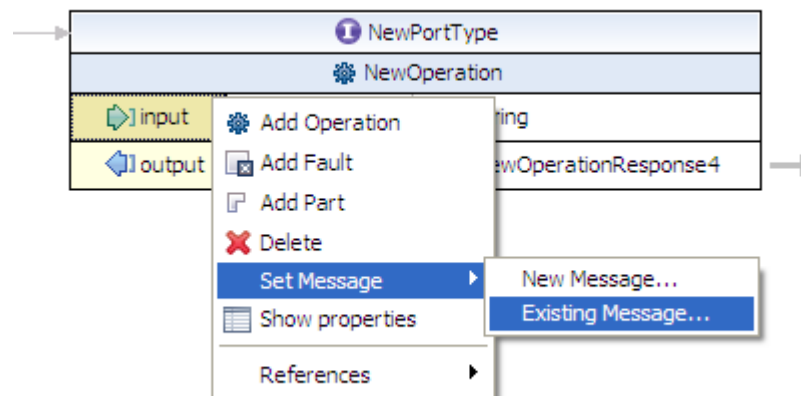
3. The new message will appear in the design workspace within the port type.



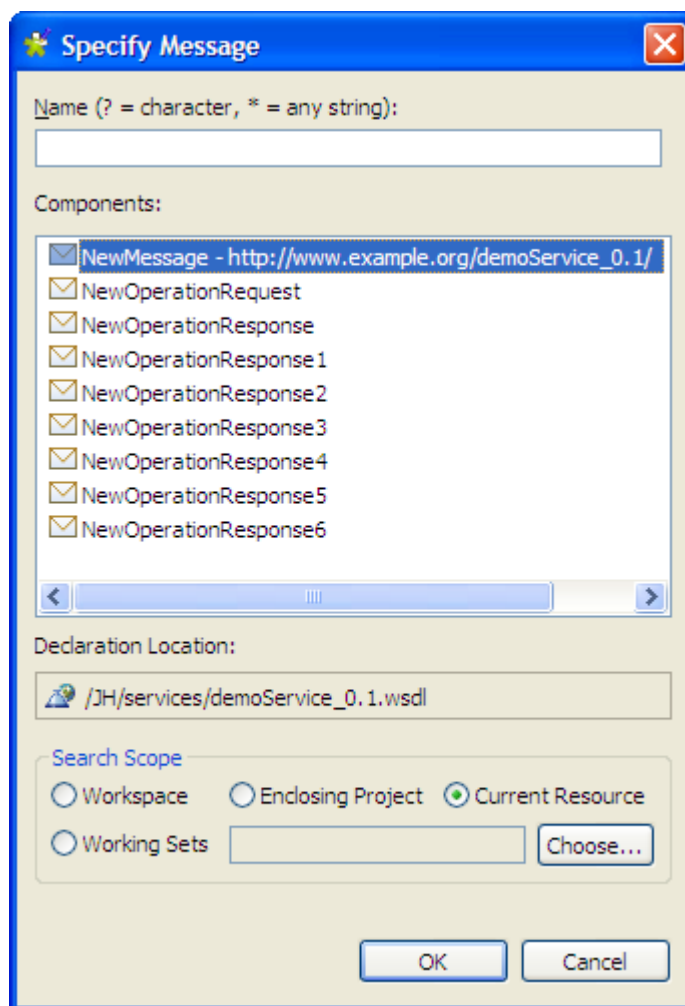
How to reuse an existing message

If you want to reuse an existing message for your input, output, or fault element, follow these steps:

1. In the design workspace, right-click your input, output, or fault element and click **Set Message>Existing Message**.



2. The **Specify Message** wizard displays. In the **Name** field, type a search string to filter the list or leave the field blank to view all the available messages in the **Components** area.



3. You can change the scope of the list by selecting one of the following options:
 - **Workspace:** lists the messages available in your workspace.
 - **Enclosing Project:** lists the messages available in the project that contains your file.
 - **Current Resource:** lists the messages available in your current file.
 - **Working Sets:** lists the messages available in a specified set of files. To specify a working set, click **Choose**. You can select files from an existing working set or create a new working set. Once you have selected the files, click **OK**.
4. The existing messages will be listed in the **Components** area. The **Declaration Location** field shows where the selected message is located. Select the message you want and click **OK**.

You can now add parts to your message. Parts are a flexible mechanism for describing the logical abstract content of a message. For more information, see [Section 6.2.2.7, “How to add a part to a message”](#).

6.2.2.7. How to add a part to a message

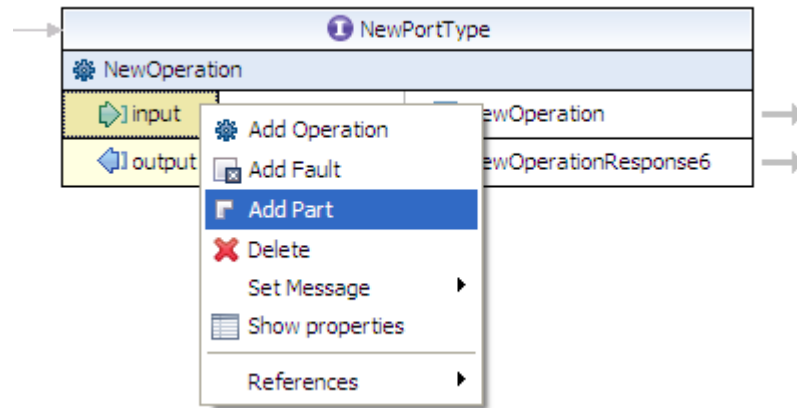
All messages contain one or more parts. Parts are a flexible mechanism for describing the logical abstract content of a message. The message definition associates each part with a type using a message-typing attribute.



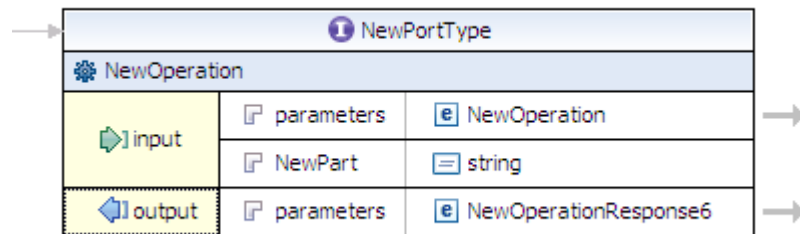
Changes of service elements are not automatically propagated to bindings. You need to regenerate the binding content and select the **Overwrite existing binding information** check box in the binding wizard to reflect changes in the WSDL. For more information, see [Section 6.2.2.3, “How to set a binding”](#).

To add a part to a message, complete the following steps:

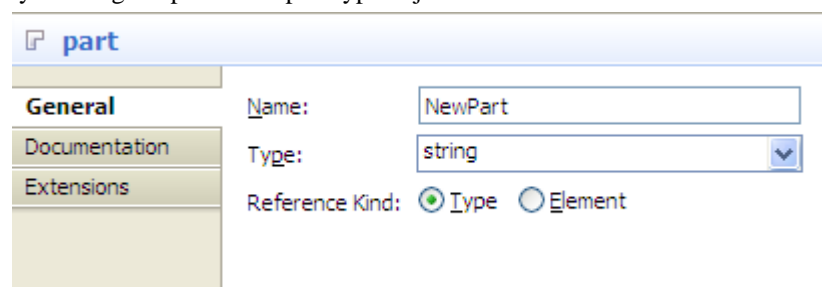
1. In design workspace, right click an input, output, or fault object and click **Add Part**.



The part is added to the input, output, or fault's message reference.



2. Select the part by clicking the part in the port type object and click the **General** tab in the Properties view.



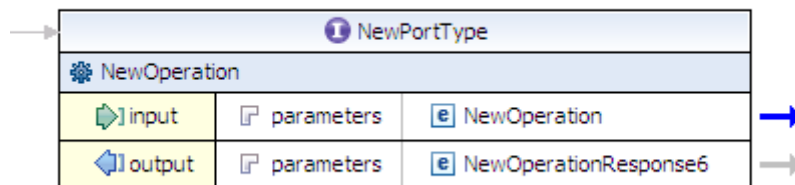
3. In the **Name** field, type the name of the part.
4. Your part can refer to either a **Type** or an **Element**. Select the appropriate option as the **Reference Kind**.
5. If you selected **Type**, you now have the option of selecting an XML schema data type from the **Type** list. The list of types comes from any available referenced XML schemas.
6. If you selected **Element**, you now have the option of selecting an XML schema element from the **Element** list. The list of elements comes from any available referenced XML schemas.

6.2.2.8. How to create a new type for your WSDL file

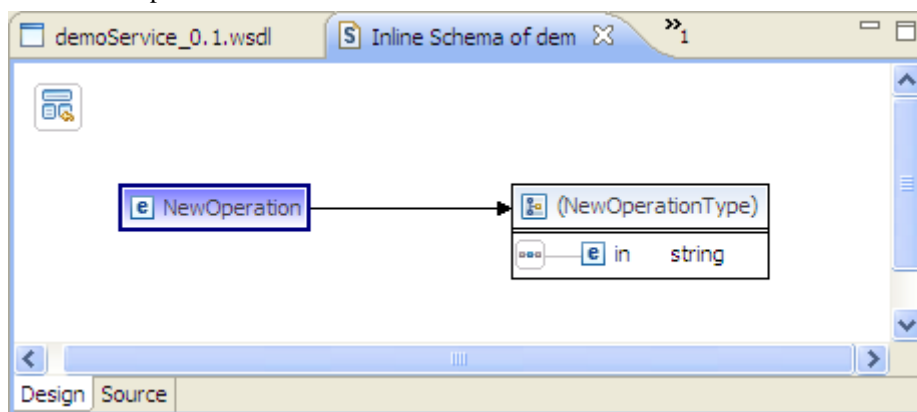
Types describe all the data types used between the client and server. WSDL is not tied exclusively to a specific typing system, but it uses the W3C XML Schema specification as its default choice.

WSDL allows type systems to be added via extensibility elements. An extensibility element may appear under the types element to identify the type definition system being used and to provide an XML container element for the type definitions.

To add an XSD type or element to your WSDL file, in the design workspace, click the arrow icon to the right of the port type object.



The XML schema editor opens.



For more information about XML schemas, see the site <http://help.eclipse.org/indigo/topic/org.eclipse.wst.xseditor.doc.user/topics/cxmlsced.html>.



Changes of service elements are not automatically propagated to bindings. You need to regenerate the binding content and select the **Overwrite existing binding information** check box in the binding wizard to reflect changes in the WSDL. For more information, see [Section 6.2.2.3, “How to set a binding”](#).

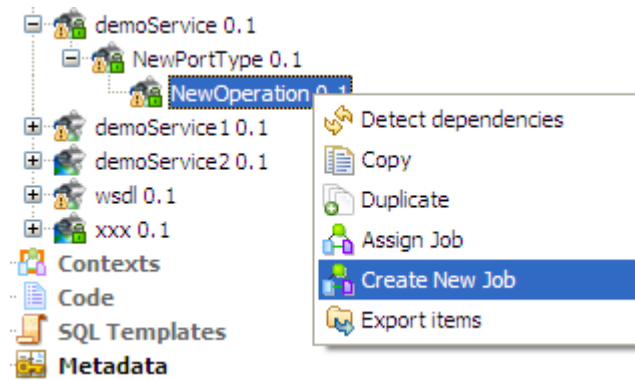
6.2.3. How to associate data service Jobs with a Service

After the WSDL file is created, you need to associate each of the operations in the WSDL file with a data service provider Job to implement the Web service. You can either create a new data service Job or assign an existing one to it.

6.2.3.1. How to create a new data service Job for an operation

To create a new data service Job for an operation:

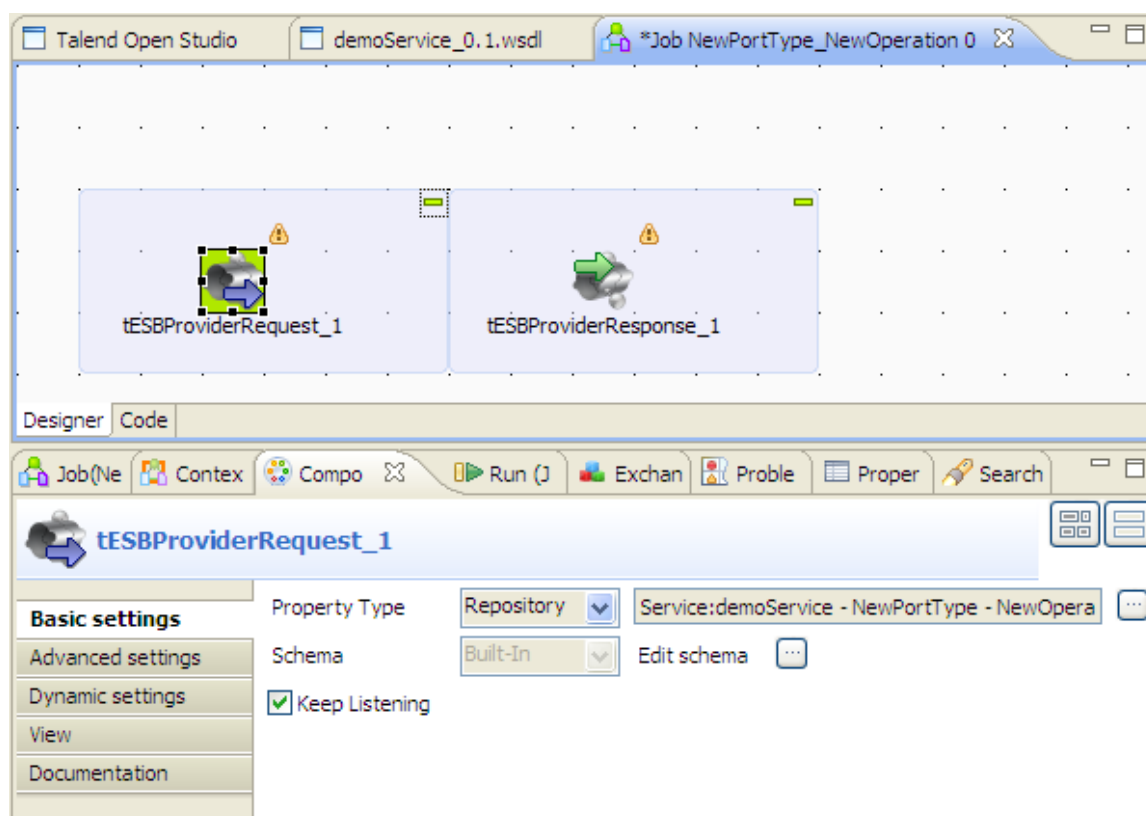
1. Right click the operation node of a Service in the Repository tree view and select **Create New Job** in the contextual menu.



2. The **New Job** wizard opens. By default, a name of the Job is already set in the **Name** field, which is the operation name. Change it and enter the Job properties as you want. For more information on how to define the Job properties, see [Section 4.2.1, “How to create a Job”](#).

3. Click **Finish** to validate the creation.

A draft Job is opened on the design workspace with the **tESBProviderRequest** and the **tESBProviderResponse** components already selected and configured. The **Property Type** of **tESBProviderResponse** is set as **Repository** and the name of the operation is retrieved by default. **tESBProviderRequest** will listen to all requests sent to the specified Web service and **tESBProviderResponse** will send back the response corresponding to the request. These two components can be found in the **Web Services** group of the **ESB** family on the **Palette**. You can design your data service provider Job according to your own need for the data integration process and execute the Job to publish the Web service.



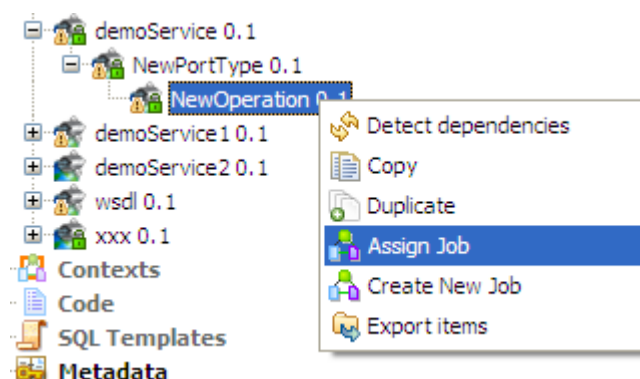
For more information on how to build a Job, see [Chapter 4, Designing a Job](#). For details about the **tESBProviderRequest**, **tESBProviderResponse**, and the other components you can find on the **Palette**, see *Talend Open Studio Components Reference Guide*.

For a scenario showing how to create a real-life data service Job, see [Appendix B, Theory into practice: Data service and routing examples](#).

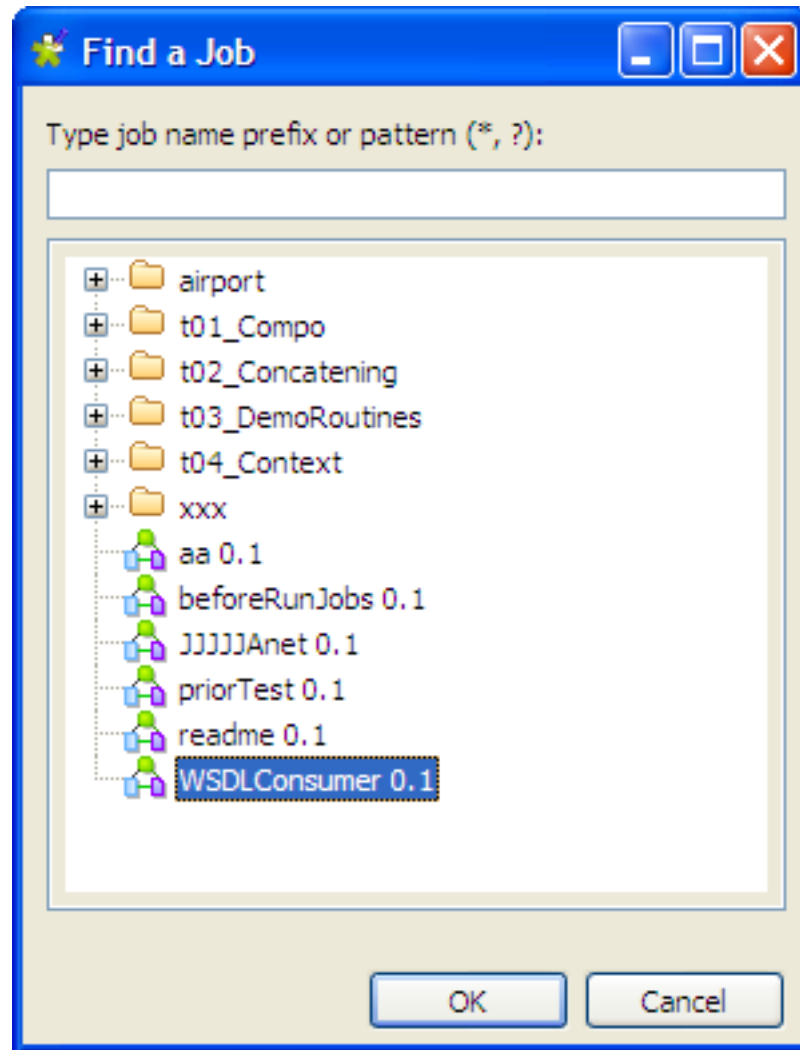
6.2.3.2. How to assign an existing data service Job to an operation

To assign an existing data service Job to an operation:

1. Right click the operation node of a Service in the Repository tree view and select **Assign Job** in the contextual menu.



2. The **Find a Job** wizard appears. You can type a search string to filter the list of Jobs or leave the field blank to view all the available Jobs. Select the Job you want from the tree view. Click **OK** to assign the selected Job to the operation and close the wizard.

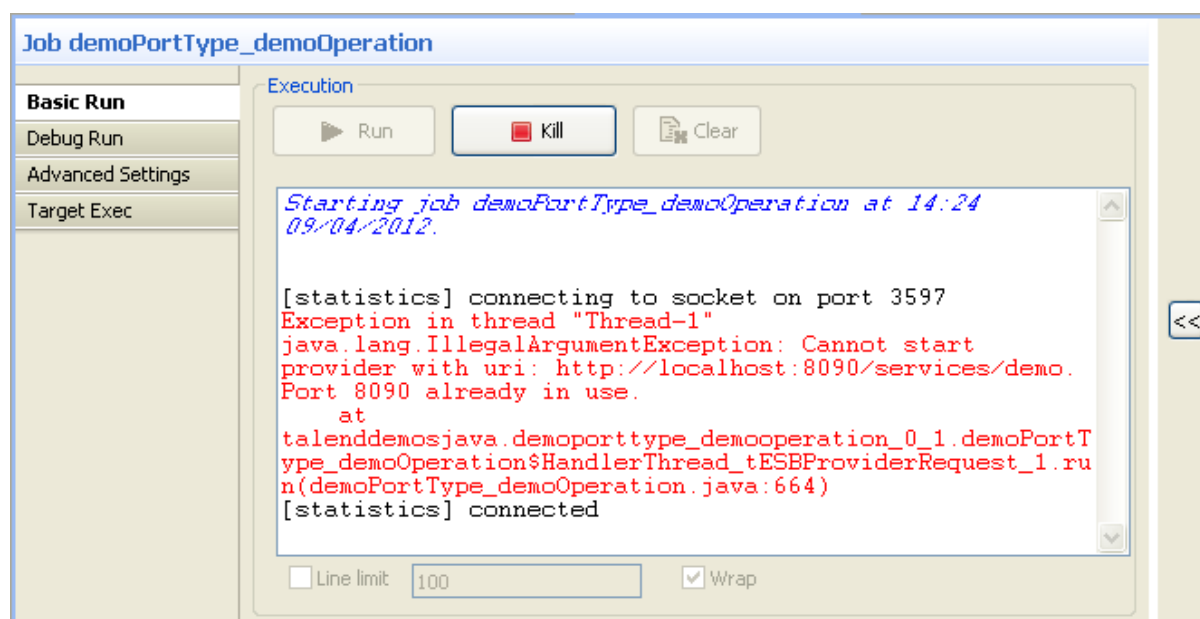


6.2.3.3. How to monitor the log messages of a data service Job

When you run a data service Job in the Studio, the progress of the execution is displayed in the console on the **Basic Run** or **Debug Run** tab of the **Run** view. The log is provided by the Apache CXF logging utility shipped with the Service builder. By default the message level is set to **INFO**. It includes any error message as well as start and end messages. It also shows the Job output in case you used a **tLogRow** component in the Job design.

This **INFO** logging level is fixed and can not be changed in *Talend Open Studio for ESB* when using the **tLogRow** component. However, the **tLogCatcher** component can be used and offers more logging capabilities. For more information on **tLogCatcher**, see the *Talend Open Studio Components*. Otherwise, when deploying your service in the **Talend Runtime**, you can decide the level used by the **Talend Runtime** to log information. For more information about the logging system of **Talend Runtime**, see the *Talend ESB Container Administration Guide*.

The following screenshot shows an example of the log with error messages.

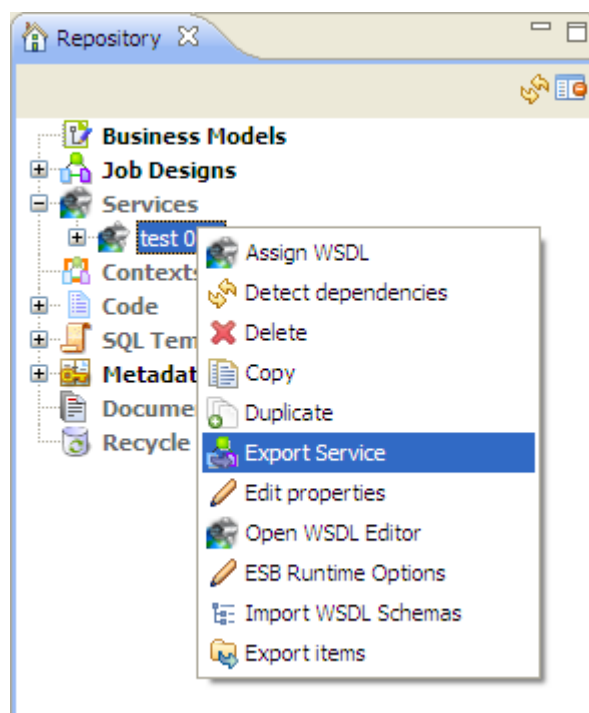


For more information about how to run a Job, see [Section 4.2.7, “How to run a Job”](#).

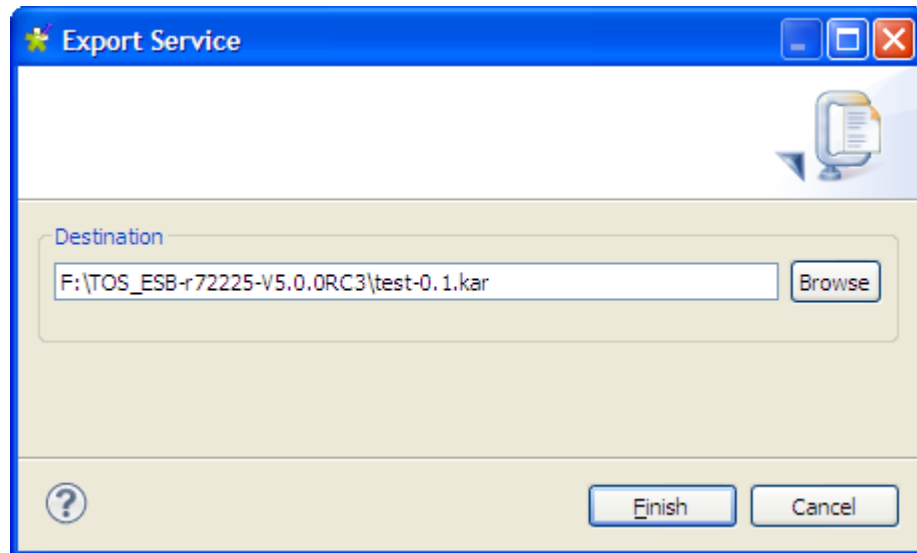
6.2.4. How to export a Service for deployment

Talend Open Studio for ESB enables you to export a Service to a .kar file that can be deployed at runtime. To do so:

1. In the **Repository** tree view, right-click the Service you want to export and select **Export Service** from the contextual menu.



2. The **Export service** wizard displays. Click **Browse** to browse to where you want to store the exported Service.



3. Click **Finish** to complete the export operation and close the wizard.

A .kar file for the Service is created in the defined place that can be deployed on your Talend Runtime.

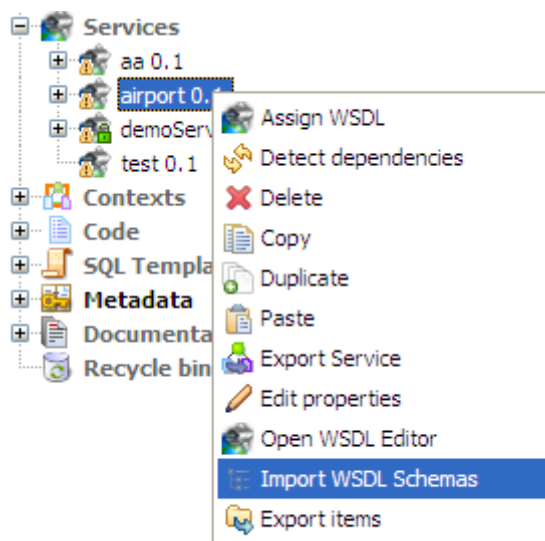
6.3. Handling Services: miscellaneous subjects

The sections below give detailed information about various subjects related to the management of a Service.

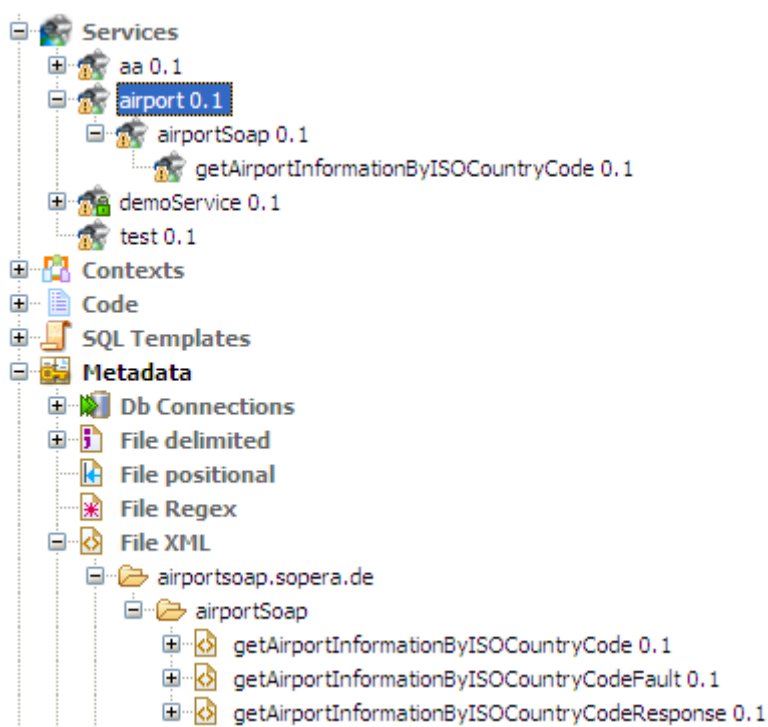
6.3.1. How to import WSDL schemas

The **Import WSDL Schemas** option of the Service allows you to retrieve and store the schemas from the WSDL file in the **Metadata** folder of the repository tree view.

To do so, right click the Service in the repository tree view and select the **Import WSDL Schemas** option from the contextual menu.



The schemas of the WSDL file in then imported to the **Metadata** folder in the repository tree view under the **File XML** node.

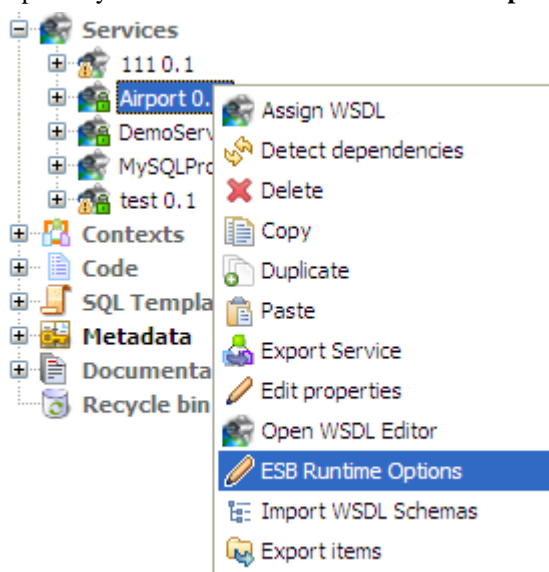


For more information about how to use the Metadata manager, see [Chapter 4, Designing a Job](#).

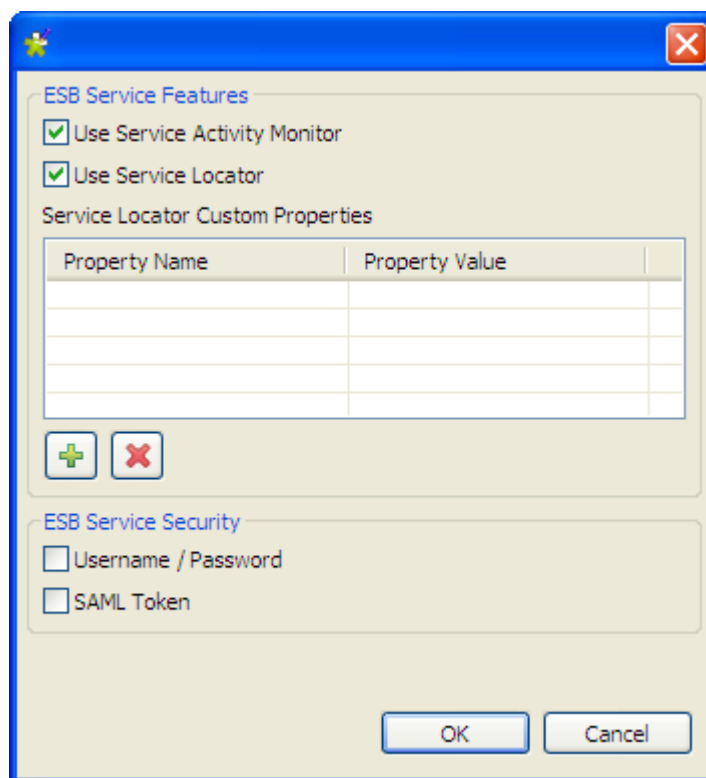
6.3.2. How to set the Runtime options

Talend Open Studio for ESB enables you to set the Runtime options of a Service. To do so:



1. Right-click a Service in the repository tree view and select **ESB Runtime Options** from the contextual menu.




The wizard appears allowing you to set the Runtime options.



- Set the Runtime options as required.

Options	Description
Use Service Activity Monitor	Select this check box to capture events and store this information to facilitate in-depth analysis of service activity and track-and-trace of messages throughout a business transaction. This can be used to analyze service response times, identify traffic patterns, perform root cause analysis and more. For more information, see <i>Talend ESB Runtime Configuration Guide</i> for Service Activity Monitoring.
Use Service Locator	Select this check box to maintain the availability of the service to help meet demands and service level agreements (SLAs).
Service Locator Custom Properties	Click  to add as many properties as needed to the Service Locator Custom Properties table. Enter the name and the value of each property in the Property Name field and the Property Value field respectively to identify the service. The table is disabled when the Use Service Locator check box is cleared. For more information, see <i>Talend ESB Runtime Configuration Guide</i> for how to install and configure the Service Locator.
Username / Password	<p>Select this check box to enable the Username token. When using the Username token, the web service provider requires the consumer to send the credentials as a part of the request and authentication is performed on the provider side.</p> <p> Note that the authentication is the same for all services running in the same Talend ESB container.</p> <p>For more information on how to configure the authentication, see <i>Talend ESB Runtime Configuration Guide</i> for how to use STS with Talend Runtime.</p>
SAML Token	Select this check box to use the SAML token. When using the SAML token, the web service provider requires the consumer to make a SAML

Options	Description
	<p>token issue request to the Security Token Service (STS) passing its credentials. On successful authentication the STS issues a SAML token. This SAML token is sent as a part of the request to the provider and the provider verifies the validity of the SAML token.</p> <p> Note that the authentication is the same for all services running in the same Talend ESB container.</p> <p>For more information on how to configure the authentication, see <i>Talend ESB Runtime Configuration Guide</i> for how to use STS with Talend Runtime.</p>

3. Click **OK** to validate your settings and close the wizard.



Chapter 7. Managing Jobs, Routes and Services

This chapter describes the management procedures you can carry out on the Jobs, Routes and Services you design in *Talend Open Studio for ESB* or you can carry out on any of the items included in a project, for example routines or metadata.

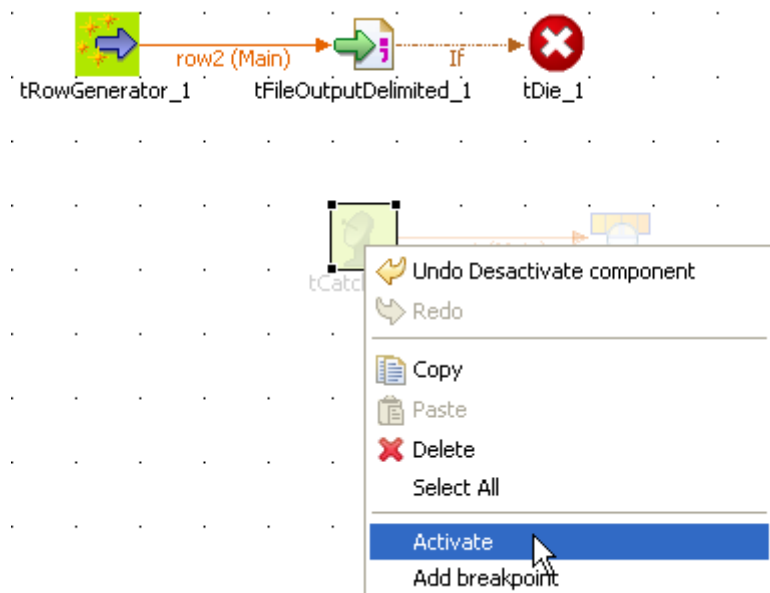
These management procedures include importing and exporting Jobs, Routes, Services and items between different projects or machines, scheduling Job execution, etc.

Before starting any processes, you need to be familiar with *Talend Open Studio for ESB* Graphical User Interface (GUI). For more information, see [Appendix A, GUI](#).

7.1. Activating/Deactivating a Job or a sub-job or a Route

You can enable or disable the whole Job, sub-job or Route directly connected to the selected component. By default, a component is activated.

In the **Main** properties of the selected component, select or clear the **Activate** check box.



Alternatively, right-click the component and select the relevant **Activate/Deactivate** command according to the current component status.

If you disable a component, no code will be generated, you will not be able to add or modify links from the disabled component to active or new components.

Related topic: [Section 4.6.2, “How to define the Start component”](#).

7.1.1. How to disable a Start component

In the case the component you deactivated is a **Start** component, components of all types and links of all nature connected directly and indirectly to it will get disabled too.

7.1.2. How to disable a non-Start component

When you clear the **Activate** check box of a regular (non Start) component, are deactivated only the selected component itself along with all direct links.

If a direct link to the disabled component is a **Main Row** connection to a sub-job, all components of this subjob will also get disabled.

7.2. Importing/exporting items, Routes, Services or Jobs

Talend Open Studio for ESB enables you to import/export your Jobs, Routes, Services, or items in your Jobs, Routes and Services from/to various projects or various versions of the Studio. It enables you as well to export Jobs, Routes, Services and thus deploy and execute those created in the Studio on any server.

7.2.1. How to import items

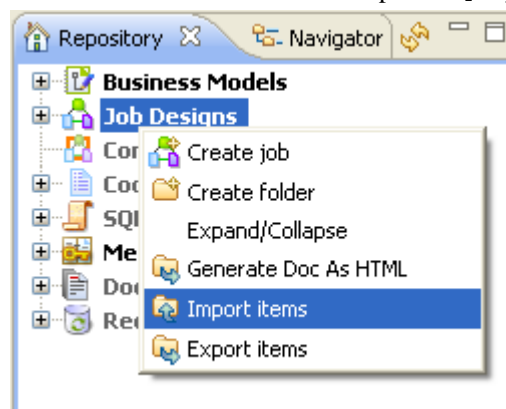
You can import items from previous versions of *Talend Open Studio for ESB* or from a different project of your current version.


The items you can possibly import are multiple:

- Business Models
- Jobs Designs
- Routes
- Services
- Routines
- Documentation
- Metadata

Follow the steps below to import them to the repository:

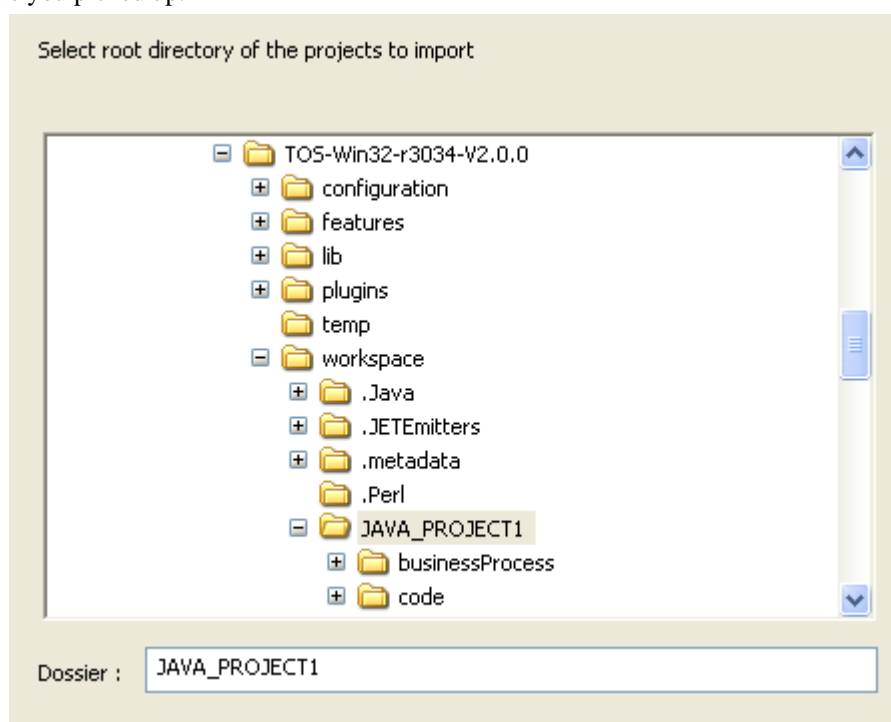
1. In the **Repository** tree view, right-click any entry such as **Job Designs**, **Routes**, **Services**, or **Business Models**, and select **Import Items** from the contextual menu to open the **[Import items]** dialog box.



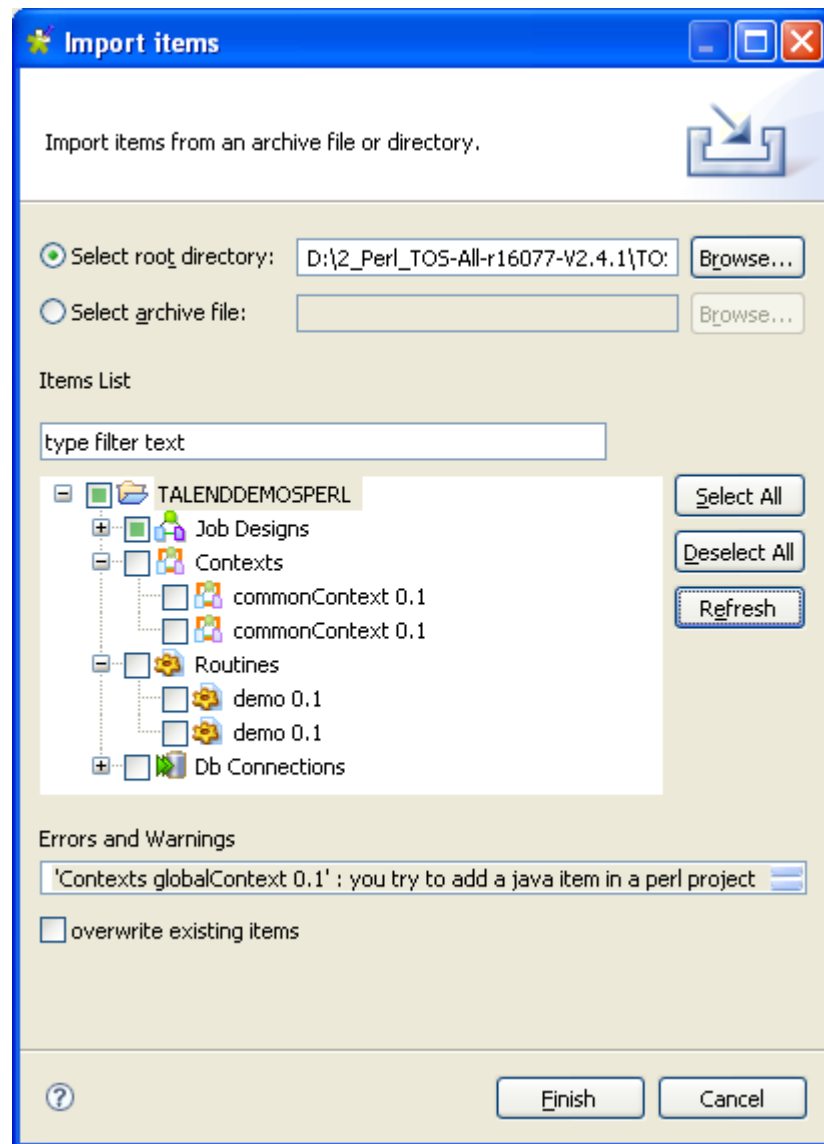
Alternatively, you can directly click the  icon on the toolbar.

2. In the dialog box that appears, select the root directory or the archive file to import the items from.
 - If the items to import are still stored on a local repository, use the **Select root directory** option and browse to the relevant project directory on your system, and then proceed to the next step.

- If you exported the items from your local repository into an archive file (including source files and scripts), use the **Select archive file** option, browse to the file and then click **Open** to go to [Step 6](#).
3. Browse down to the relevant **Project** folder within the workspace directory. It should correspond to the project name you picked up.



4. If you only want to import very specific items such as some **Job Designs**, you can select the specific folder, such as Process where all the Job Designs for the project are stored. If you only have **Business Models** to import, select the specific folder: **BusinessProcess**.
5. But if your project gather various types of items (Business Models, Jobs Designs, Routes, Services, Metadata, Routines...), we recommend you to select the **Project** folder to import all items in one go, and click **OK** to continue.



6. Select the **overwirte existing items** check box if you want to overwrite existing items with those having the same names to be imported. This will refresh the **Items List**.
7. From the **Items List** which displays all valid items that can be imported, select the items that you want to import by selecting the corresponding check boxes.
8. Click **Finish** to validate the import.

The imported items are displayed in the repository in the relevant folder respective to their nature.



If there are several versions of the same items, they will all be imported into the Project you are running, unless you already have identical items.

7.2.2. How to export Jobs

The **Export Job** feature allows you to deploy and execute a Job on any server, independent of *Talend Open Studio for ESB*.

The export Job feature adds all of the files required to execute the Job to an archive, including the .bat and .sh along with any context-parameter files or other related files.

To export Jobs, complete the following:



1. In the **Repository** tree view, right-click the Job you want to export, and select **Export Job** to open the **[Export Jobs]** dialog box.



You can show/hide a tree view of all created Jobs in *Talend Open Studio for ESB* directly from the **[Export Jobs]** dialog box by clicking the and the buttons respectively. The Jobs you earlier selected in the Studio tree view display with selected check boxes. This accessibility helps to modify the selected items to be exported directly from the dialog box without having to close it and go back to the **Repository** tree view in *Talend Open Studio for ESB* to do that.

2. On the **To archive file** field, browse to the directory where you want to save your exported Job.
3. On the **Job Version** area, select the version number of the Job you want to export if you have created more than one version of the Job.
4. Select the **Export Type** in the list between **Autonomous Job**, **Axis Webservice (WAR)**, **Axis Webservice (Zip)** and **OSGI Bundle For ESB**.
5. Select the **Extract the zip file** check box to automatically extract the archive file in the target directory.
6. In the **Options** area, select the file type(s) you want to add to the archive file. The check boxes corresponding to the file types necessary for the execution of the Job are selected by default. You can clear these check boxes depending on what you want to export.

Option	Description
Shell launcher	Select this check box to export the .bat and/or .sh files necessary to launch the exported Job. <ul style="list-style-type: none"> • All: exports the .bat and .sh files. • Unix exports the .sh file. • Windows exports the .bat file.
System routines	Select this check box to export system routines.

Option	Description
User routines	Select this check box to export user routines.
Required Talend modules	Select this check box to export export Talend modules.
Java classes	Select this check bo to export export the .java file holding Java classes generated by the Job when designing it.
Source files	<p>Select this check bo to export the sources used by the Job during its execution including the .item and .properties files, Java and Talend sources.</p> <p> If you select the Source files check box, you can reuse the exported Job in a <i>Talend Open Studio for ESB</i> installed on another machine. These source files are only used in <i>Talend Open Studio for ESB</i>.</p>
Export Dependencies	Select this check box if you want to export the dependencies of your Job, i.e. contexts, routines, connections, etc.
Context script	<p>Select this check box to export ALL context parameters files and not just those you select in the corresponding list.</p> <p> To export only one context, select the context that fits your needs from the Context script list, including the .bat or .sh files holding the appropriate context parameters. Then you can, if you wish, edit the .bat and .sh files to manually modify the context type.</p>
Apply to children	Select this check box if you want to apply the context selected from the list to all child Jobs.

- Click the **Override parameters' values** button, if necessary.

In the window which opens you can update, add or remove context parameters and values of the Job context you selected in the list.

- Click **Finish** to validate your changes, complete the export operation and close the dialog box.

A zipped file for the Jobs is created in the defined place.



If the Job to be exported calls a user routine that contains one or more extra Java classes in parallel with the public class named the same as the user routine, the extra class or classes will not be included in the exported file. To export such classes, you need to include them within the class with the routine name as inner classes. For more information about user routines, see [Section 10.4, "Managing user routines"](#). For more information about classes and inner classes, see relevant Java manuals.

7.2.2.1. How to export Jobs as Autonomous Job

In the case of a Plain Old Java Object export, if you want to reuse the Job in *Talend Open Studio for ESB* installed on another machine, make sure you selected the **Source files** check box. These source files (.item and .properties) are only needed within *Talend Open Studio for ESB*.

Select a context in the list when offered. Then once you click the **Override parameters' values** button below the **Context script** checkbox, the opened window will list all of the parameters of the selected context. In this window, you can configure the selected context as needs.

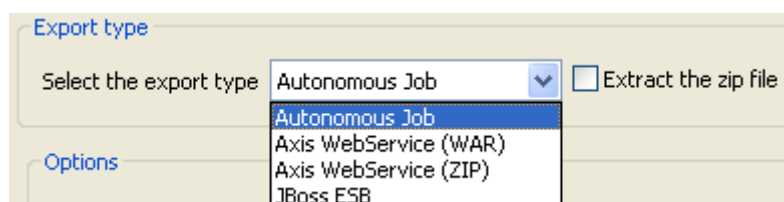
All contexts parameter files are exported along in addition to the one selected in the list.



After being exported, the context selection information is stored in the .bat/.sh file and the context settings are stored in the context **.properties** file.

7.2.2.2. How to export Jobs as Webservice

In the **[Export Jobs]** dialog box, you can change the type of export in order to export the Job selection as Webservice archive.



Select the type of archive you want to use in your Web application.

Archive type	Description
WAR	The options are read-only. Indeed, the WAR archive generated includes all configuration files necessary for the execution or deployment from the Web application.
ZIP	All options are available. In the case the files of your Web application config are all set, you have the possibility to only set the Context parameters if relevant and export only the Classes into the archive.

Once the archive is produced, place the WAR or the relevant Class from the ZIP (or unzipped files) into the relevant location, of your Web application server.

The URL to be used to deploy the Job, typically reads as follow:

```
http://localhost:8080/Webappname/services/JobName?method=runJob&args=null
```

where the parameters stand as follow:

URL parameters	Description
http://localhost:8080/	Type in the Webapp host and port.
/Webappname/	Type in the actual name of your web application.
/services/	Type in “services” as the standard call term for web services.
/JobName	Type in the exact name of the Job you want to execute.
?method=runJob&args=null	The method is RunJob to execute the Job.

The call return from the Web application is 0 when there is no error and different from 0 in case of error. For a real-life example of creating and exporting a Job as a Webservice and calling the exported Job from a browser, see [Section 7.2.2.3, “An example of exporting a Job as a Web service”](#).

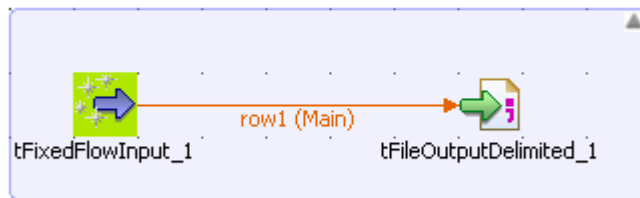
The **tBufferOutput** component was especially designed for this type of deployment. For more information regarding this component, see *Talend Open Studio Components Reference Guide*.

7.2.2.3. An example of exporting a Job as a Web service

This scenario describes first a simple Job that creates a .txt file and writes in it the current date along with first and last names. Second, it shows how to export this Job as a Webservice. And finally, it calls the Job exported as a Webservice from a browser. The exported Job as a Webservice will simply return the “return code” given by the operating system.

Procedure 7.1. Creating the Job:

1. Drop the following components from the **Palette** onto the design workspace: **tFixedFlowInput** and **tFileOutputDelimited**.
2. Connect **tFixedFlowInput** to **tFileOutputDelimited** using a **Row > Main** link.



3. In the design workspace, select **tFixedFlowInput**, and click the **Component** tab to define the basic settings for **tFixedFlowInput**.
4. Set the **Schema** to **Built-In** and click the [...] button next to **Edit Schema** to describe the data structure you want to create from internal variables. In this scenario, the schema is made of three columns, *now*, *firstname*, and *lastname*.

Column	Key	Type	Nullable	Date Pa...	Le...	Pr...	D...	Co...
now	<input checked="" type="checkbox"/>	Date	<input checked="" type="checkbox"/>	"dd-MM-...				
firstname	<input type="checkbox"/>	String	<input checked="" type="checkbox"/>					
lastname	<input type="checkbox"/>	String	<input checked="" type="checkbox"/>					

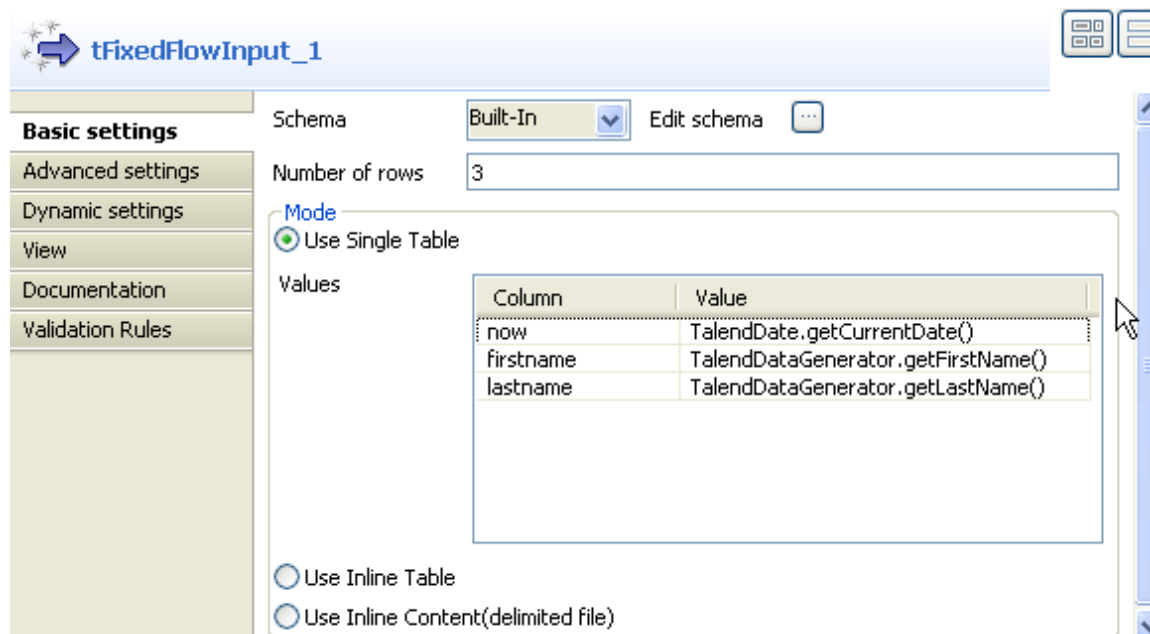
5. Click the [+] button to add the three parameter lines and define your variables, and then click **OK** to close the dialog box and accept propagating the changes when prompted by the system.

The three defined columns display in the **Values** table of the **Basic settings** view of **tFixedFlowInput**.

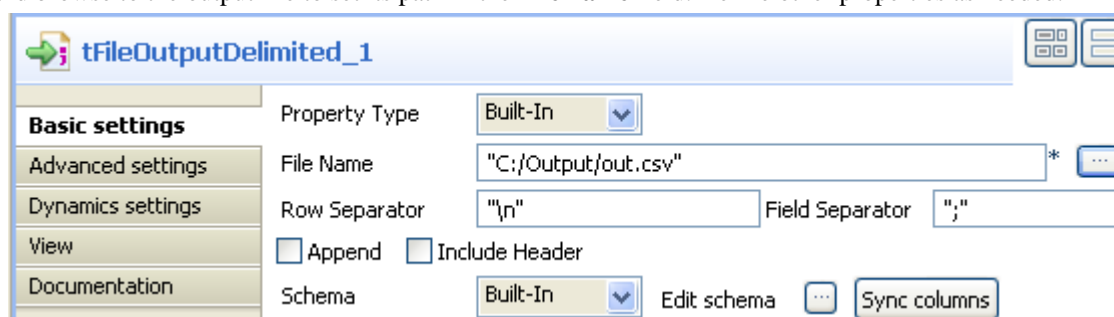
Values

Column	Value
now	
firstname	
lastname	

6. In the **Value** cell of each of the three defined columns, press **Ctrl+Space** to access the global variable list, and select *TalendDate.getCurrentDate()*, *talendDatagenerator.getFirstName*, and *talendDataGenerator.getLastName* for the *now*, *firstname*, and *lastname* columns respectively.
7. In the **Number of rows** field, enter the number of lines to be generated.



8. In the design workspace, select **tFileOutputDelimited**, click the **Component** tab for **tFileOutputDelimited**, and browse to the output file to set its path in the **File name** field. Define other properties as needed.



If you press **F6** to execute the Job, three rows holding the current date and first and last names will be written to the set output file.

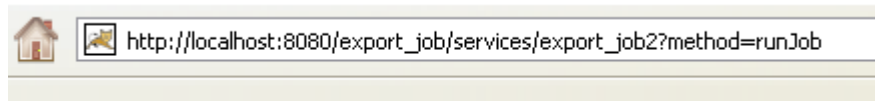
Procedure 7.2. Exporting the Job as a Webservice:

1. In the **Repository** tree view, right-click the above created Job and select **Export Job**. The [Export Jobs] dialog box appears.

2. Click the **Browse...** button to select a directory to archive your Job in.
3. In the **Job Version** area, select the version of the Job you want to export as a web service.
4. In the **Export type** area, select the export type you want to use in your Web application (WAR in this example) and click **Finish**. The **[Export Jobs]** dialog box disappears.
5. Copy the War folder and paste it in the Tomcat webapp directory.

Procedure 7.3. Calling the Job from a browser:

1. Type the following URL into your browser: *http://localhost:8080/export_job/services/export_job2?method=runJob* where “export_job” is the name of the webapp directory deployed in Tomcat and “export_job2” is the name of the Job.



2. Click **Enter** to execute the Job from your browser.

Ce fichier XML ne semble pas avoir d'information de style lui

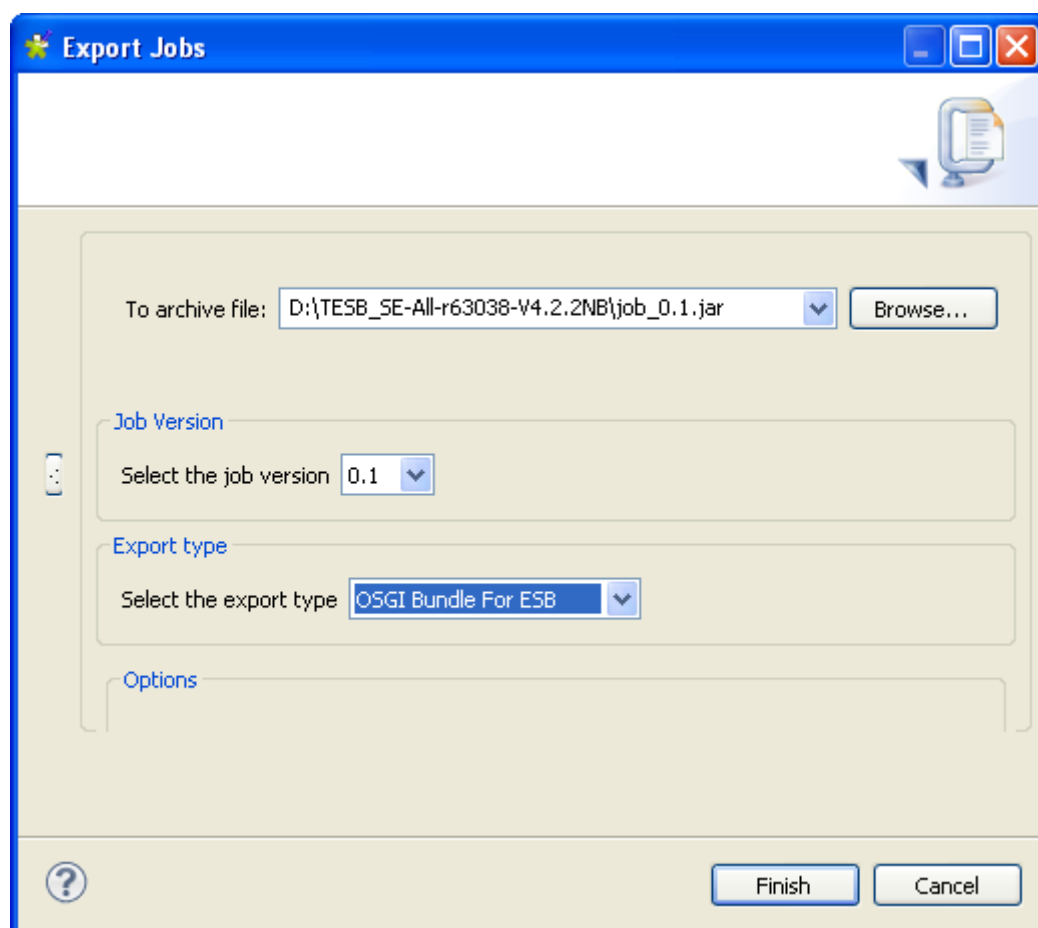
```
- <soapenv:Envelope>
  - <soapenv:Body>
    - <runJobReturn xsi:type="ns1:runJobReturn">
      - <ns1:item xsi:type="ns1:ArrayOf_xsd_string">
        <ns1:item xsi:type="xsd:string">0</ns1:item>
      </ns1:item>
    </runJobReturn>
  </soapenv:Body>
</soapenv:Envelope>
```

The return code from the Web application is 0 when there is no error and 1 if an error occurs.

For a real-life example of creating and exporting a Job as a Webservices using the **tBufferOutput** component, see the **tBufferOutput** component in *Talend Open Studio Components Reference Guide*.

7.2.2.4. How to export Jobs as OSGI Bundle For ESB

In the **[Export Jobs]** dialog box, you can change the type of export in order to export the Job selection as an OSGI Bundle in order to deploy your Job in **Talend ESB Container**.



1. In the **Job Version** area, select the version number of the Job you want to export if you have created more than one version of the Job.
2. In the **Export Type** area, select **OSGI Bundle For ESB** to export your Job as OSGI Bundle.
The extension of your export automatically change to *.jar* as it is what **Talend ESB Container** is expecting.
3. Click the **Browse...** button to specify the folder in which exporting your Job.
4. Click **Finish** to export it.

7.2.3. How to export Routes



The **Export Route** feature allows you to export a Route to an ESB Runtime KAR file in order to deploy it in Talend ESB Container.

The **Export Route** feature adds all of the files required to execute the Route to an archive, including the *.bat* and *.sh* along with any context-parameter files or other related files.

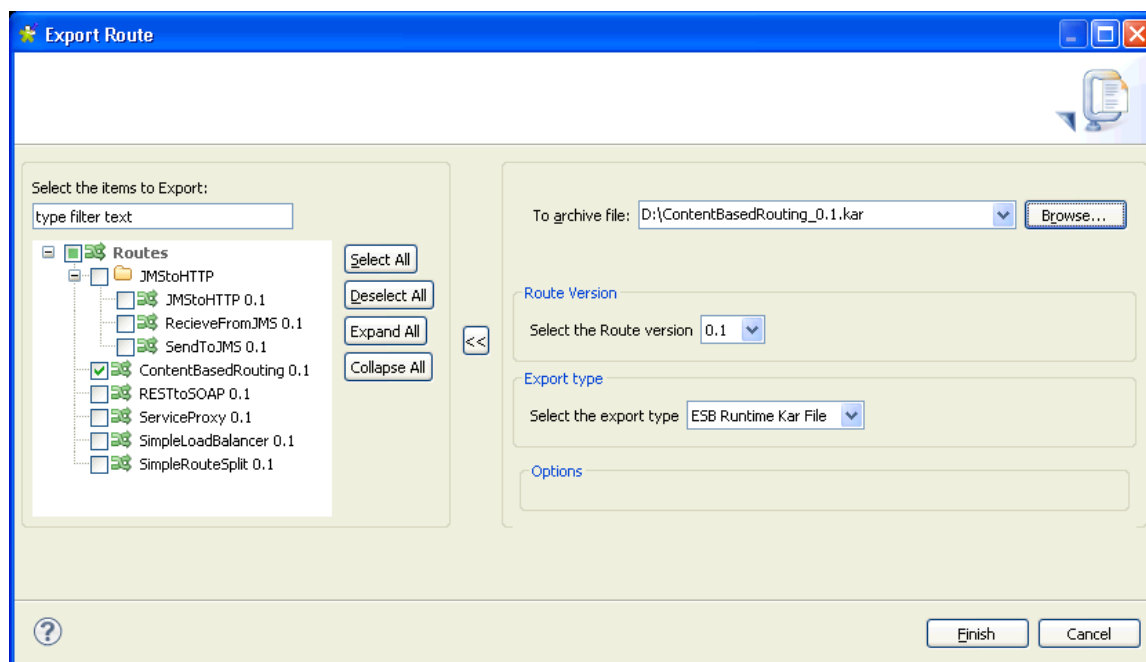
To export Routes, complete the following:

1. In the **Repository** tree view, right-click the Route you want to export, and select **Export Route** to open the **[Export Routes]** dialog box.



You can show/hide a tree view of all created Jobs in *Talend Open Studio for ESB* directly from the **[Export Routes]** dialog box by clicking the  and the  buttons respectively. The Routes you

earlier selected in the Studio tree view display with selected check boxes. This accessibility helps to modify the selected items to be exported directly from the dialog box without having to close it and go back to the **Repository** tree view in *Talend Open Studio for ESB* to do that.



2. In the **To archive file** field, browse to the directory where you want to save your exported Route.
3. In the **Route Version** area, select the version number of the Route you want to export if you have created more than one version of the Route.
4. In the **Export Type** area, **ESB Runtime Kar file** is selected by default.
5. Click **Finish** to validate your changes, complete the export operation and close the dialog box.

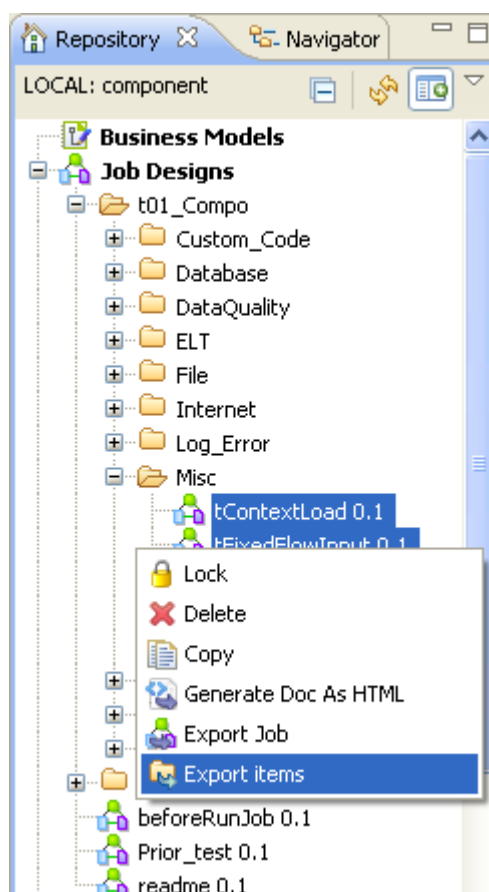
A KAR file for the Route is created in the defined place.

7.2.4. How to export items

You can export multiple items from the repository onto a directory or an archive file. Hence you have the possibility to export metadata information such as DB connection or Documentation along with your Job or your Business Model, or even your Routes and Services, for example.

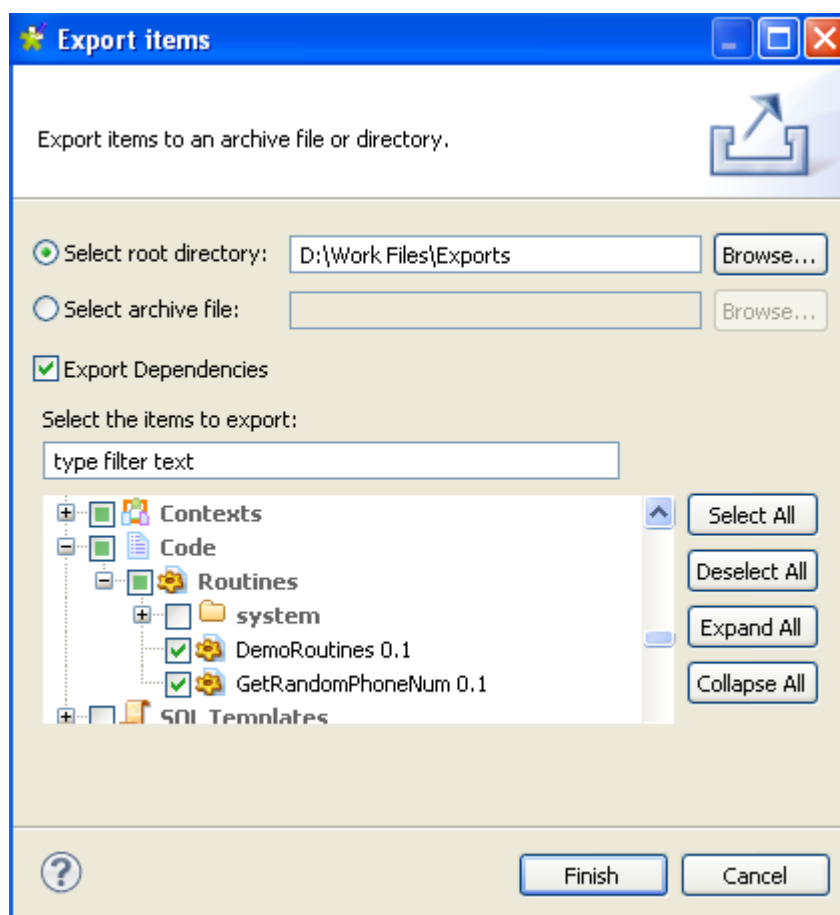
To do so:

1. In the **Repository** tree view, select the items you want to export.
2. To select several items at a time, press the **Ctrl** key and select the relevant items.



If you want to export a database table metadata entry, make sure you select the whole DB connection, and not only the relevant table as this will prevent the export process to complete correctly.

3. Right-click while maintaining the **Ctrl** key down and select **Export items** on the pop-up menu:



You can select additional items on the tree for exportation if required.

4. Click **Browse** to browse to where you want to store the exported items. Alternatively, define the archive file where to compress the files for all selected items.



If you have several versions of the same item, they will all be exported.



Select the **Export Dependencies** check box if you want to set and export routine dependencies along with Jobs you are exporting. By default, all of the user routines are selected. For further information about routines, see [Section 10.1, “What are routines”](#).

5. Click **Finish** to close the dialog box and export the items.

7.2.5. How to change context parameters in Jobs and Routes

As explained in [Section 7.2.2, “How to export Jobs”](#), you can edit the context parameters:

If you want to change the context selection, simply edit the .bat/.sh file and change the following setting: `--context=Prod` to the relevant context.

If you want to change individual parameters in the context selection, edit the .bat/.sh file and add the following setting according to your need:

Operation	Setting
To change <i>value1</i> for parameter <i>key1</i>	<code>--context_param key1=value1</code>
To change <i>value1</i> and <i>value2</i> for respective parameters <i>key1</i> and <i>key2</i>	<code>--context_param key1=value1 --context_param key2=value2</code>
To change a value containing space characters such as in a file path	<code>--context_param key1="path to file"</code>

7.3. Managing repository items

Talend Open Studio for ESB enables you to edit the items centralized in the repository and to update the Jobs that use these items accordingly.

7.3.1. How to handle updates in repository items

You can update the metadata, context parameters that are centralized in the **Repository** tree view any time in order to update the database connection or the context group details, for example.

When you modify any of the parameters of an entry in the **Repository** tree view, all Jobs using this repository entry will be impacted by the modification. This is why the system will prompt you to propagate these modifications to all the Jobs that use the repository entry.

The following sections explain how to modify the parameters of a repository entry and how to propagate the modifications to all or some of the Jobs that use the entry in question.

7.3.1.1. How to modify a repository item

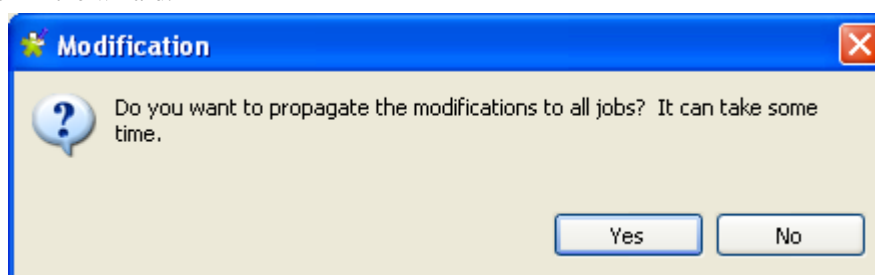
To update the parameters of a repository item, complete the following:

1. Expand the **Metadata**, or **Contexts** node in the **Repository** tree view and browse to the relevant entry that you need to update.
2. Right-click this entry and select the corresponding edit option in the contextual menu.

A respective wizard displays where you can edit each of the definition steps for the entry parameters.

When updating the entry parameters, you need to propagate the changes throughout numerous Jobs or all your Jobs that use this entry.

A prompt message pops up automatically at the end of your update/modification process when you click the **Finish** button in the wizard.



- Click **Yes** to close the message and implement the changes throughout all Jobs impacted by these changes. For more information about the first way of propagating all your changes, see [Section 7.3.1.2, “How to update impacted Jobs automatically”](#).

Click **No** if you want to close the message without propagating the changes. This will allow you to propagate your changes on the impacted Jobs manually on one by one basis. For more information on another way of propagating changes, see [Section 7.3.1.3, “How to update impacted Jobs manually”](#).

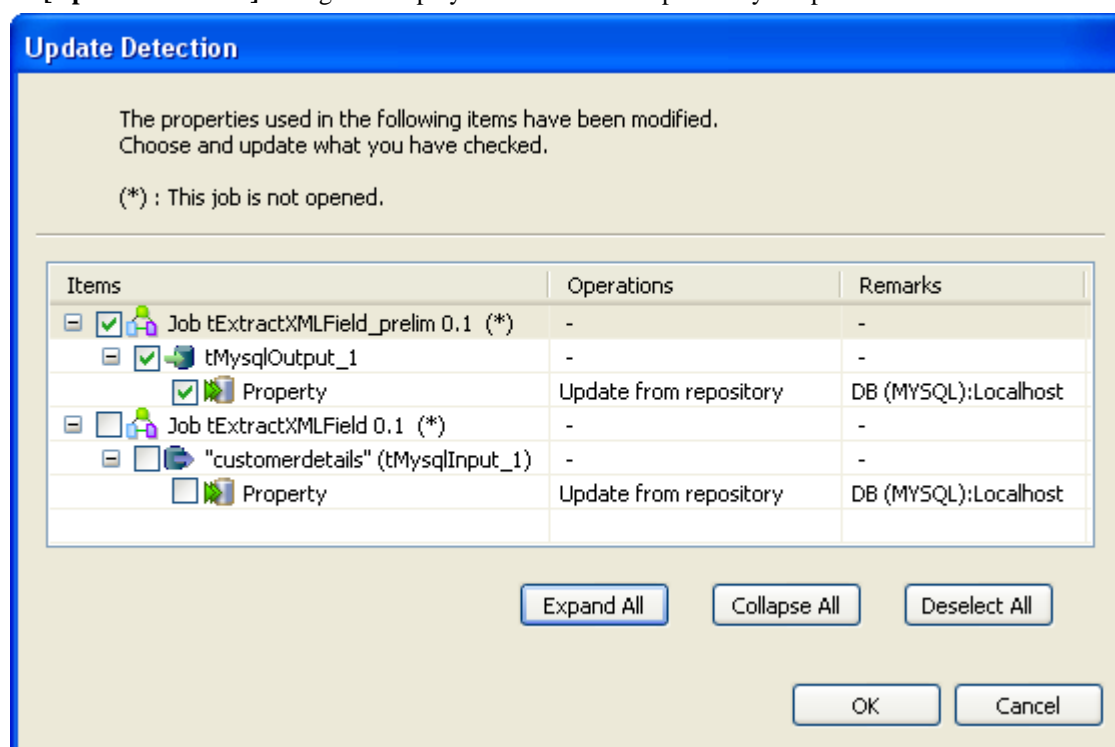
7.3.1.2. How to update impacted Jobs automatically

After you update the parameters of any item already centralized in the **Repository** tree view and used in different Jobs, a message will prompt you to propagate the modifications you did to all Jobs that use these parameters.

To update impacted Jobs, complete the following:

- In the **[Modification]** dialog box, click **Yes** to let the system scan your **Repository** tree view for the Jobs that get impacted by the changes you just made. This aims to automatically propagate the update throughout all your Jobs (open or not) in one click.

The **[Update Detection]** dialog box displays to list all Jobs impacted by the parameters that are modified.



You can open the **[Update Detection]** dialog box any time if you right-click the item centralized in the **Repository** tree view and select **Manage Dependencies** from the contextual menu. For more information, see [Section 7.3.1.3, “How to update impacted Jobs manually”](#).

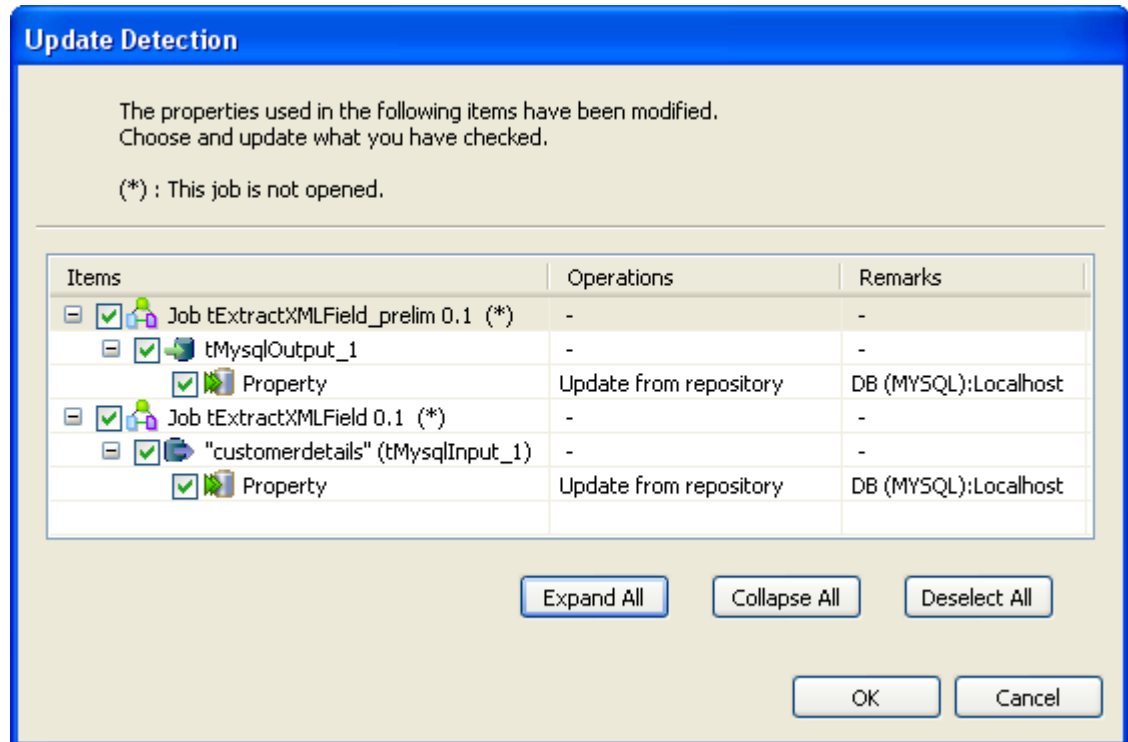
- If needed, clear the check boxes that correspond to the Jobs you do not wish to update. You can update them any time later through the **Detect Dependencies** menu. For more information, see [Section 7.3.1.3, “How to update impacted Jobs manually”](#).
- Click **OK** to close the dialog box and update all selected Jobs.

7.3.1.3. How to update impacted Jobs manually

Before propagating changes in the parameters of an item centralized in the tree view throughout the Jobs using this entry, you might want to view all Jobs that are impacted by the changes. To do that, complete the following:

1. In the **Repository** tree view, expand the node holding the entry you want to check what Jobs use it.
2. Right-click the entry and select **Detect Dependencies**.

A progress bar indicates the process of checking for all Jobs that use the modified metadata or context parameter. Then a dialog box displays to list all Jobs that use the modified item.




3. Select the check boxes corresponding to the Jobs you want to update with the modified metadata or context parameter and clear those corresponding to the Jobs you do not want to update.
4. Click **OK** to validate and close the dialog box.



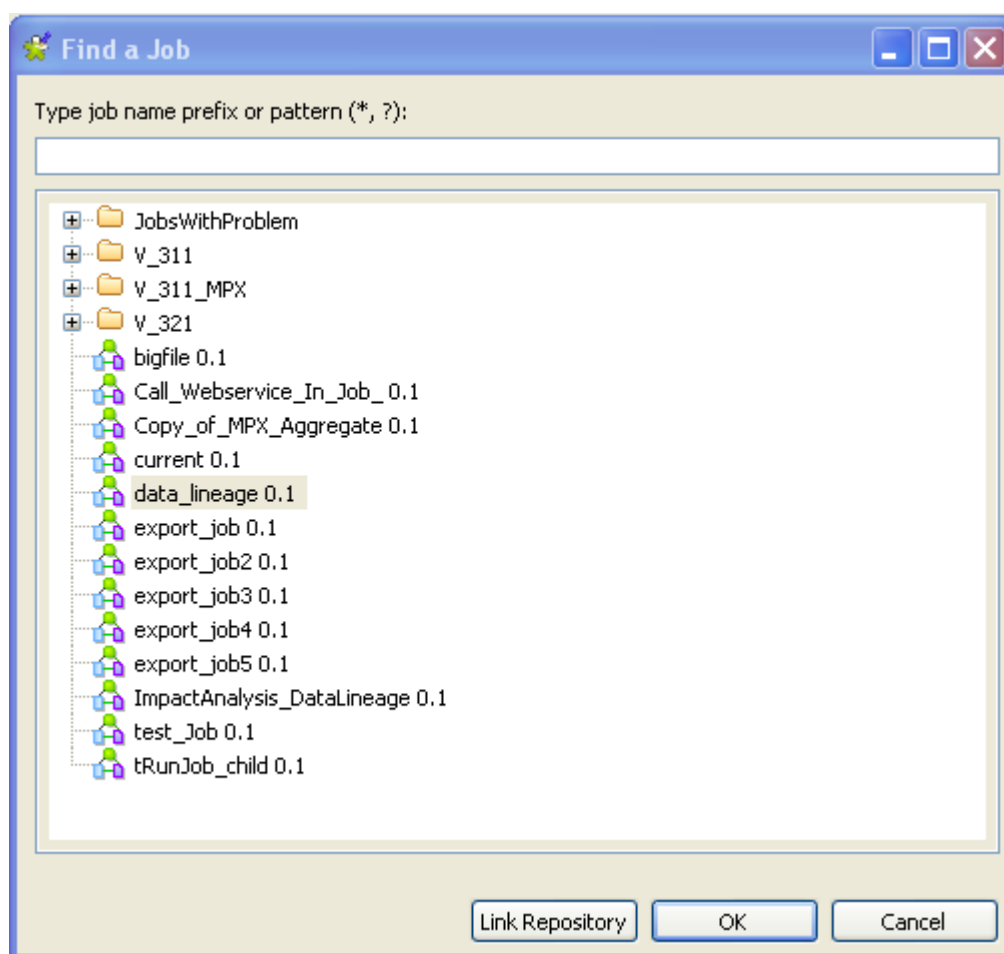
The Jobs that you choose not to update will be switched back to **Built-in**, as the link to the Repository cannot be maintained. It will thus keep their setting as it was before the change.

7.4. Searching a Job in the repository

If you want to open a specific Job in the **Repository** tree view of the current *Talend Open Studio for ESB* and you can not find it for one reason or another, you can simply click  on the quick access toolbar.

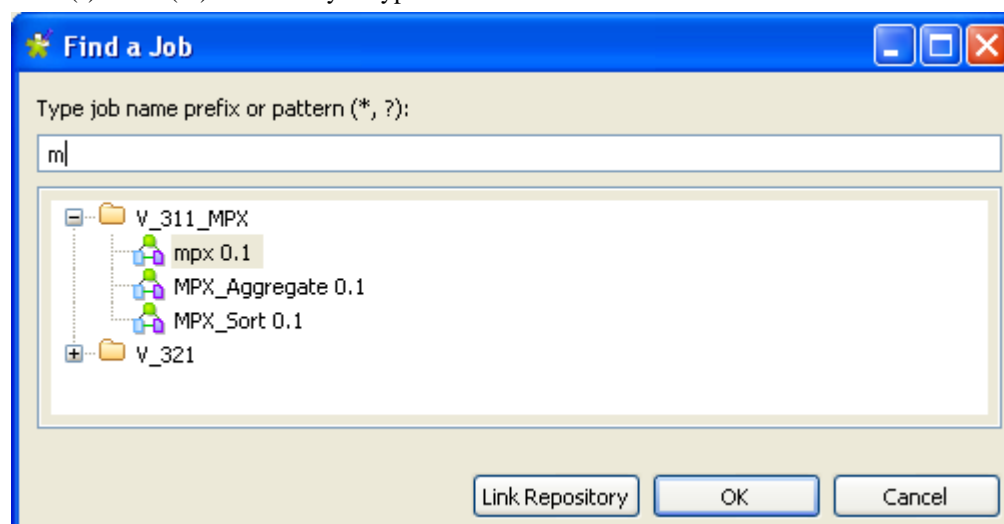
To find a Job in the **Repository** tree view, complete the following:

1. On *Talend Open Studio for ESB* toolbar, click  to open the **[Find a Job]** dialog box that lists automatically all the Jobs you created in the current Studio.



2. Enter the Job name or part of the Job name in the upper field.

When you start typing your text in the field, the Job list is updated automatically to display only the Job(s) which name(s) match(es) the letters you typed in.



3. Select the desired Job from the list and click **Link Repository** to automatically browse to the selected Job in the **Repository** tree view.
4. If needed, click **Cancel** to close the dialog box and then right-click the selected Job in the **Repository** tree view to perform any of the available operations in the contextual menu.

Otherwise, click **OK** to close the dialog box and open the selected Job on the design workspace.

7.5. Managing Job and Route versions

When you create a Job or a Route in *Talend Open Studio for ESB*, by default its version is 0.1 where 0 stands for the major version and 1 for the minor version.

You can create many versions of the same Job or Route. To do that:

1. Close your Job or Route if it is open on the design workspace, otherwise, its properties will be read-only and thus you can not modify them.
2. In the **Repository** tree view, right-click your Job or Route and select **Edit properties** in the drop-down list to open the **[Edit properties]** dialog box.
3. Next to the **Version** field, click the **M** button to increment the major version and the **m** button to increment the minor version.
4. Click **Finish** to validate the modification.



By default, when you open a Job or a Route, you open its last version.

Any previous version of the Job or the Route is read-only and thus can not be modified.

To change the version of your Job or your Route, you can also:

1. Close your Job or your Route if it is open on the design workspace, otherwise, its properties will be read-only and thus you can not modify them.
2. In the **Repository** tree view, right-click your Job or your Route and select **Open another version** in the drop-down list.
3. In the dialog box, select the **Create new version and open it** check box and click the **M** button to increment the major version and the **m** button to increment the minor version.
4. Click **Finish** to validate the modification and open this new version of your Job or your Route.

You can also save a Job or a Route and increment its version in the same time, by clicking **File > Save as...**





This option does not overwrite your current Job or Route, it saves your Job or Route as another new Job or Route and/or with another version.

You can access a list of the different versions of a Job and Routes and perform certain operations. To do that:

1. In the **Repository** tree view, select the Job or the Route you want to consult the versions of.
2. Click **Job > Version** or Routes in succession to display the version list of the selected Job or Route.
3. Right-click the Job or the Route version you want to consult.
4. Do one of the followings:

Select	To...
Edit Job/Route	open the last version of the Job or Route.

Select	To...
	 This option is only available when you select the last version of the Job or Route.
Read job/Route	consult the Job in read-only mode.
Open Job Hierarchy	consult the hierarchy of the Job.
Edit properties	<p>edit Job or Route properties.</p> <p>Note: The Job or Route should not be open on the design workspace, otherwise it will be in read-only mode.</p>  This option is only available when you select the last version of the Job or Route.
Run job/Route	execute the Job or Route.

You can also manage the version of several Jobs, Routes and/or metadata at the same time, as well as Jobs and their dependencies and/or child Jobs from the Project Settings. For more information, see [Section 2.6.2, “Version management”](#).

7.6. Documenting a Job

Talend Open Studio for ESB enables you to generate documentation that gives general information about your projects, Jobs or joblets. You can automate the generation of such documentation and edit any of the generated documents.

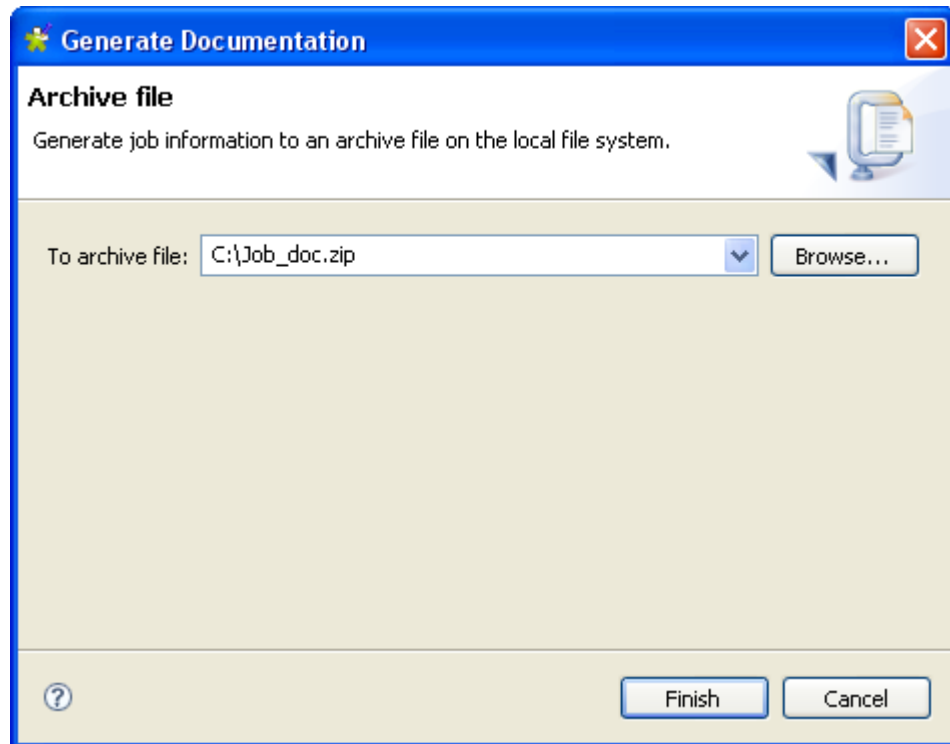
7.6.1. How to generate HTML documentation

Talend Open Studio for ESB allows you to generate detailed documentation in HTML of the Job(s) you select in the **Repository** tree view of your Studio. This auto-documentation offers the following:

- The properties of the project where the selected Jobs have been created,
- The properties and settings of the selected Jobs along with preview pictures of each of the Jobs,
- The list of all the components used in each of the selected Jobs and component parameters.

To generate an HTML document for a Job, complete the following:

1. In the **Repository** tree view, right-click a **Job** entry or select several **Job Designs** to produce multiple documentations.
2. Select **Generate Doc as HTML** on the contextual menu.



3. Browse to the location where the generated documentation archive should be stored.
4. In the same field, type in a name for the archive gathering all generated documents.
5. Click **Finish** to validate the generation operation.

The archive file is generated in the defined path. It contains all required files along with the Html output file. You can open the HTML file in your favorite browser.

7.6.2. How to update the documentation on the spot

You can choose to manually update your documentation on the spot.

To update a single document, right-click the relevant documentation entry and select **Update documentation**.

7.7. Handling Job execution

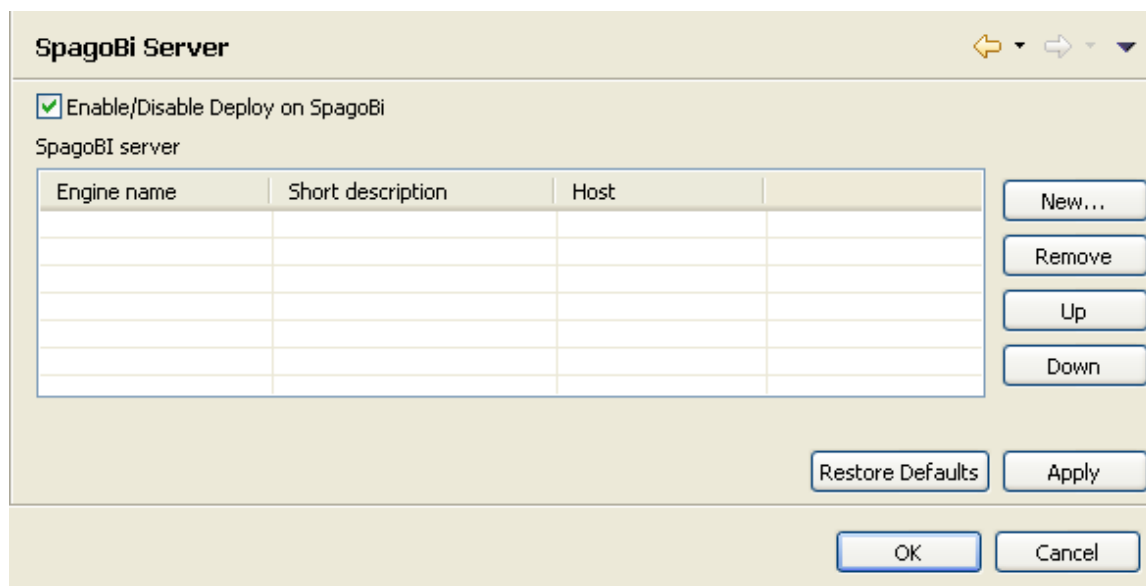
7.7.1. How to deploy a Job on SpagoBI server

From *Talend Open Studio for ESB* interface, you can deploy your Jobs easily on a SpagoBI server in order to execute them from your SpagoBI administrator.

7.7.1.1. How to create a SpagoBI server entry

Beforehand, you need to set up your single or multiple SpagoBI server details in *Talend Open Studio for ESB*.

1. On the menu bar, click **Window > Preferences** to open the **[Preferences]** dialog box.
2. Expand the **Talend > Import/Export** nodes in succession and select **SpagoBI Server** to display the relevant view.



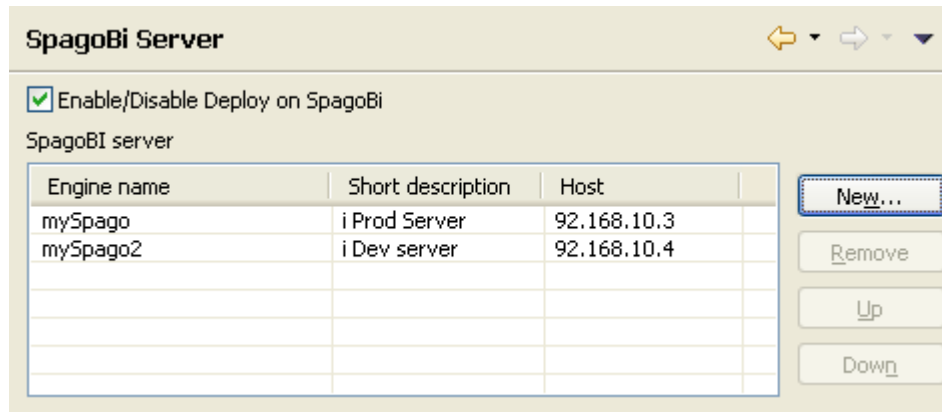
3. Select the **Enable/Disable Deploy on SpagoBI** check box to activate the deployment operation.
4. Click **New** to open the **[Create new SpagoBI server]** dialog box and add a new server to the list.



5. Enter your SpagoBI server details, as described below:

Field	Description
Engine Name	Internal engine name used in <i>Talend Open Studio for ESB</i> . This name is not used in the generated code.
Short description	Free text to describe the server entry you are recording.
Host	IP address or host name of the machine running the SpagoBI server.
Login	User name required to log on to the SpagoBI server.
Password	Password for SpagoBI server logon authentication.

6. Click **OK** to validate the details of the new server entry and close the dialog box.



The newly created entry is added to the table of available servers. You can add as many SpagoBI entries as you need.

7. Click **Apply** and then **OK** to close the **[Preferences]** dialog box.

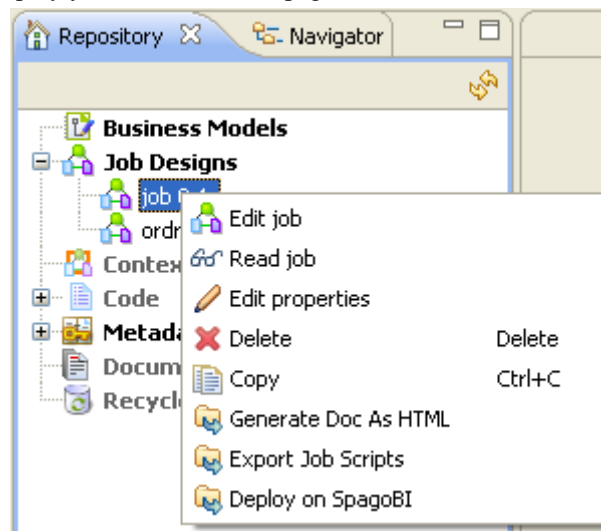
7.7.1.2. How to edit or remove a SpagoBI server entry

Select the relevant entry in the table, click the **Remove** button next to the table to first delete the outdated entry.

Then if required, simply create a new entry including the updated details.

7.7.1.3. How to deploy your Jobs on a SpagoBI server

Follow the steps below to deploy your Job(s) onto a SpagoBI server.



1. In the **Repository** tree view, expand **Job Designs** and right-click the Job to deploy.
2. In the drop-down list, select **Deploy on SpagoBI**.
3. As for any Job export, select a **Name** for the Job archive that will be created and fill it in the **To archive file** field.
4. Select the relevant **SpagoBI server** on the drop-down list.

5. The **Label**, **Name** and **Description** fields come from the Job main properties.
6. Select the relevant context in the list.
7. Click **OK** once you have completed the setting operation.

The Jobs are now deployed onto the relevant SpagoBI server. Open your SpagoBI administrator to execute your Jobs.



Chapter 8. Mapping data flows

The most common way to handle multiple input and output flows including transformations and data re-routing is to use the dedicated mapping components: **tMap** and **tXMLMap**.

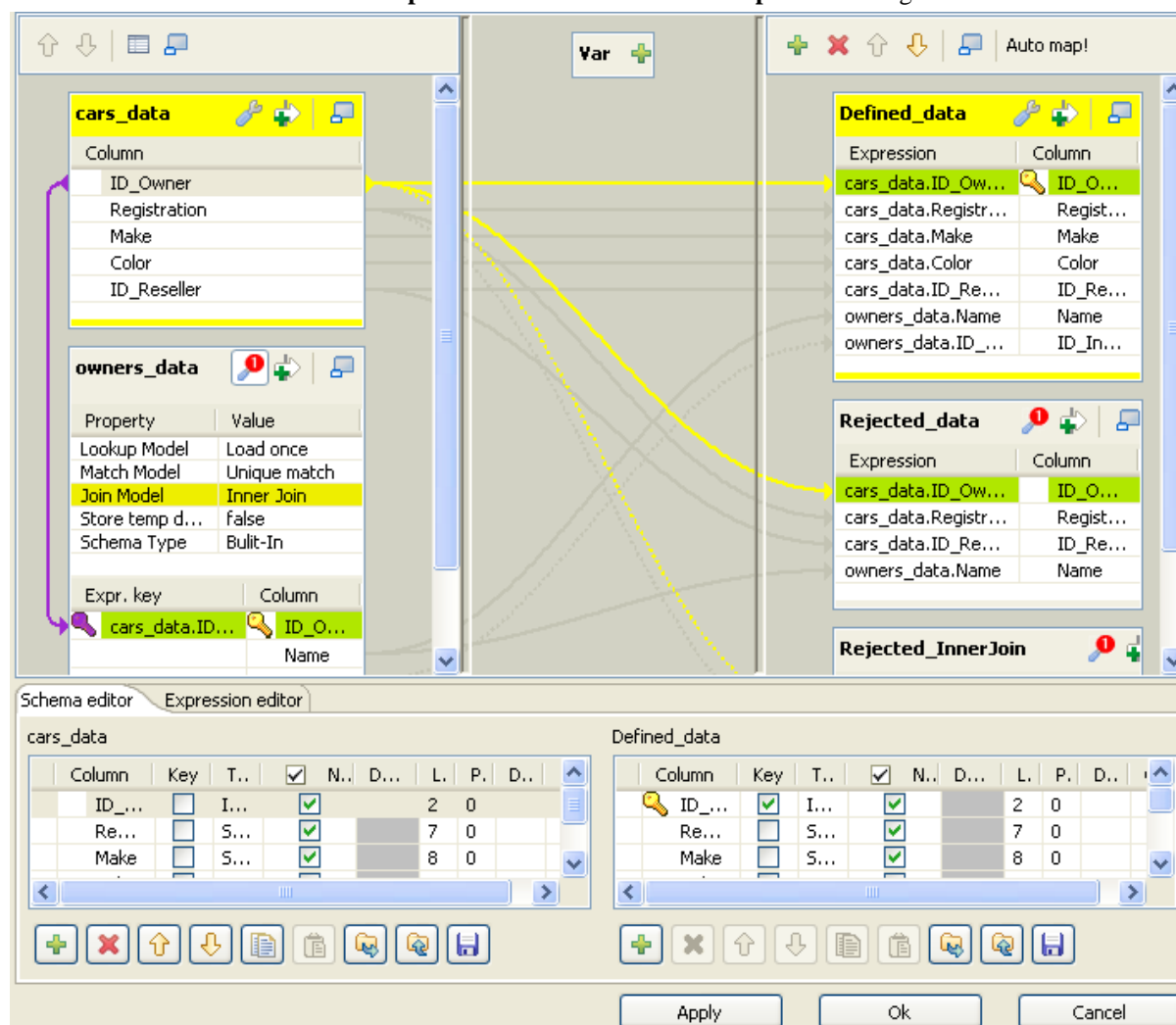
This chapter gives details separately about the usage principles of these two components, for further information or scenario and use cases, see *Talend Open Studio Components Reference Guide*.

Before starting any data integration processes, you need to be familiar with *Talend Open Studio for ESB* Graphical User Interface (GUI). For more information, see [Appendix A, GUI](#).

8.1. tMap and tXMLMap interfaces

tMap and **tXMLMap** are advanced components which require more detailed explanation than other **Talend** components. The **Map Editor** is an “all-in-one” tool allowing you to define all parameters needed to map, transform and route your data flows via a convenient graphical interface.

You can minimize and restore the **Map Editor** and all tables in the **Map Editor** using the window icons.



This figure presents the interface of **tMap**. That of **tXMLMap** differs slightly in appearance. For example, in addition to the **Schema editor** and the **Expression editor** tabs on the lower part of this interface, **tXMLMap** has a third tab called **Tree schema editor**. For further information about **tXMLMap**, see [Section 8.3, “tXMLMap operation”](#).

The **Map Editor** is made of several panels:

- The **Input panel** is the top left panel on the editor. It offers a graphical representation of all (main and lookup) incoming data flows. The data are gathered in various columns of input tables. Note that the table name reflects the main or lookup row from the Job design on the design workspace.
- The **Variable panel** is the central panel in the **Map Editor**. It allows the centralization of redundant information through the mapping to variable and allows you to carry out transformations.
- The **Output panel** is the top right panel on the editor. It allows mapping data and fields from Input tables and Variables to the appropriate Output rows.

- Both bottom panels are the Input and Output schemas description. The **Schema editor** tab offers a schema view of all columns of input and output tables in selection in their respective panel.
- **Expression editor** is the edition tool for all expression keys of Input/Output data, variable expressions or filtering conditions.

The name of input/output tables in the **Map Editor** reflects the name of the incoming and outgoing flows (row connections).

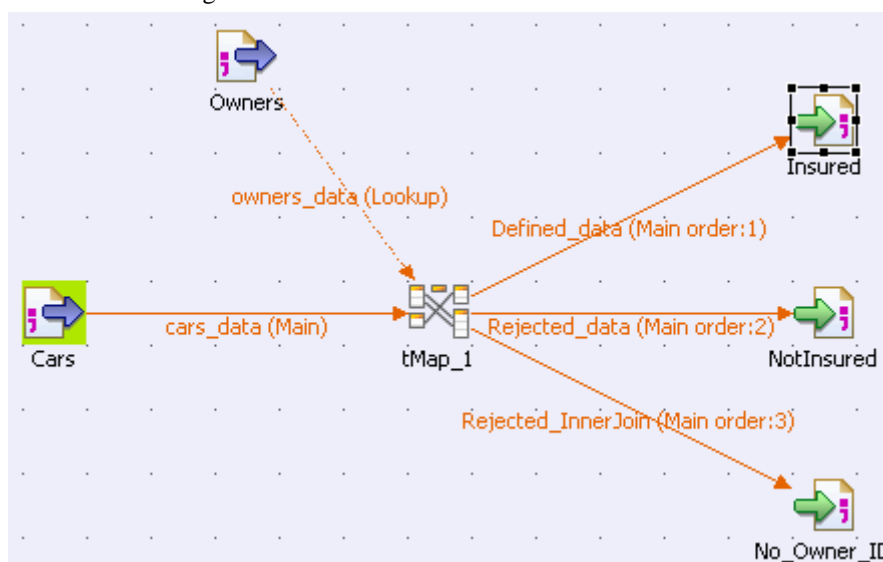
The following sections present separately **tMap** and **tXMLMap**.

8.2. tMap operation

tMap allows the following types of operations:

- data multiplexing and demultiplexing,
- data transformation on any type of fields,
- fields concatenation and interchange,
- field filtering using constraints,
- data rejecting.

As all these operations of transformation and/or routing are carried out by **tMap**, this component cannot be a start or end component in the Job design.



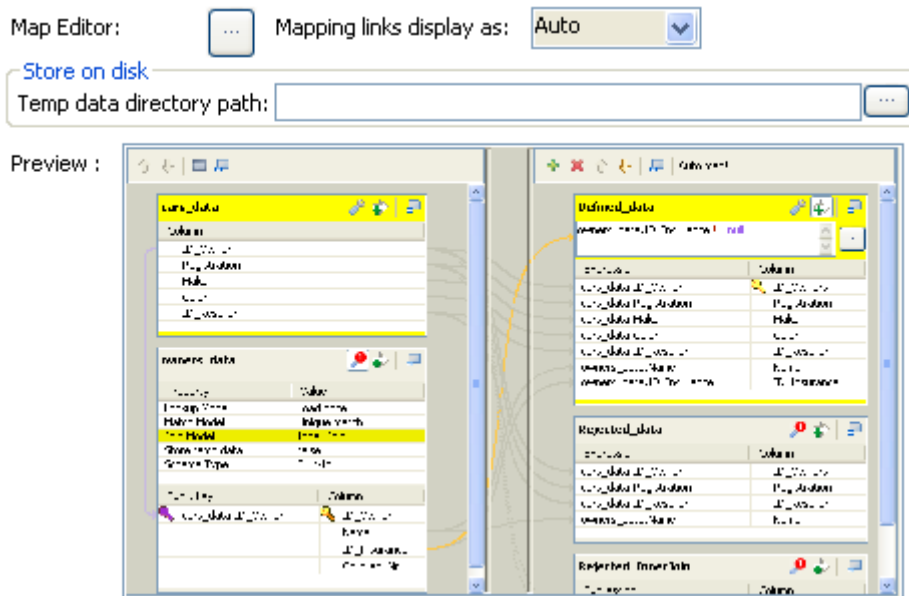
tMap uses incoming connections to pre-fill input schemas with data in the **Map Editor**. Therefore, you cannot create new input schemas directly in the **Map Editor**. Instead, you need to implement as many **Row** connections incoming to **tMap** component as required, in order to create as many input schemas as needed.

The same way, create as many output row connections as required. However, you can fill in the output with content directly in the **Map Editor** through a convenient graphical editor.

Note that there can be only one **Main** incoming rows. All other incoming rows are of **Lookup** type. Related topic: [Section 4.3.1.1, “Row connection”](#).

Lookup rows are incoming connections from secondary (or reference) flows of data. These reference data might depend directly or indirectly on the primary flow. This dependency relationship is translated with a graphical mapping and the creation of an expression key.

The **Map Editor** requires the connections to be implemented in your Job in order to be able to define the input and output flows in the **Map Editor**. You also need to create the actual mapping in your Job in order to display the **Map Editor** in the **Preview** area of the **Basic settings** view of the **tMap** component.



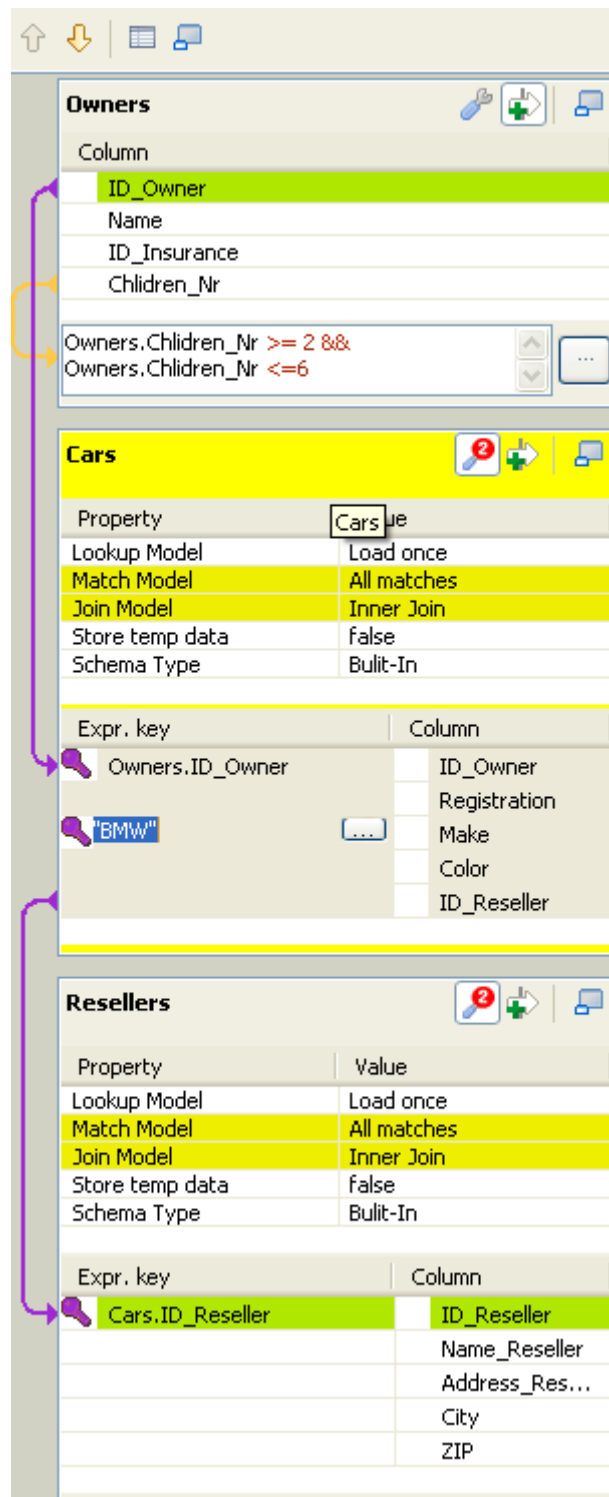
To open the **Map Editor** in a new window, double-click the **tMap** icon in the design workspace or click the three-dot button next to the **Map Editor** in the **Basic settings** view of the **tMap** component.

The following sections give the information necessary to use the **tMap** component in any of your Job designs.

8.2.1. Setting the input flow in the Map Editor

The order of the **Input** tables is essential. The top table reflects the **Main** flow connection, and for this reason, is given priority for reading and processing through the **tMap** component.

For this priority reason, you are not allowed to move up or down the **Main** flow table. This ensures that no Join can be lost.



Although you can use the up and down arrows to interchange **Lookup** tables order, be aware that the **Joins** between two lookup tables may then be lost.

Related topic: [Section 8.2.1.2, “How to use Explicit Join”](#).

8.2.1.1. How to fill in Input tables with a schema

To fill in the input tables, you need to define either the schemas of the input components connected to the **tMap** component on your design workspace, or the input schemas within the **Map Editor**.

For more information about setting a component schema, see [Section 4.2.6, “How to define component properties”](#).

For more information about setting an input schema in the **Map Editor**, see [Section 8.2.5, “Setting schemas in the Map Editor”](#).

Main and Lookup table content

The order of the **Input** tables is essential.

The **Main Row** connection determines the **Main** flow table content. This input flow is reflected in the first table of the **Map Editor**'s Input panel.

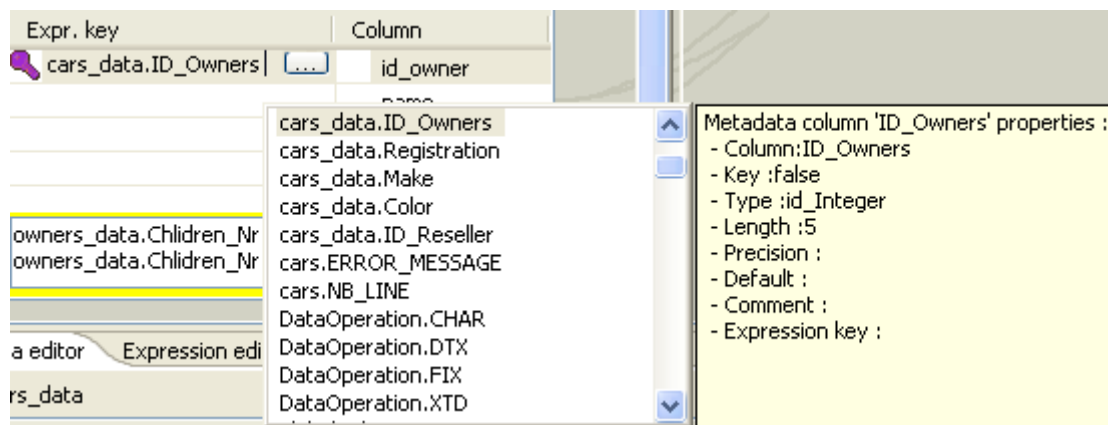
The **Lookup** connections' content fills in all other (secondary or subordinate) tables which displays below the **Main** flow table. If you have not define the schema of an input component yet, the input table displays as empty in the Input area.

The key is also retrieved from the schema defined in the Input component. This **Key** corresponds to the key defined in the input schema where relevant. It has to be distinguished from the hash key that is internally used in the **Map Editor**, which displays in a different color.

Variables

You can use global or context variables or reuse the variable defined in the **Variables** area. Press **Ctrl+Space bar** to access the list of variables. This list gathers together global, context and mapping variables.

The list of variables changes according to the context and grows along new variable creation. Only valid mappable variables in the context show on the list.



Docked at the **Variable** list, a metadata tip box display to provide information about the selected column.

Related topic: [Section 8.2.2, “Mapping variables”](#)

8.2.1.2. How to use Explicit Join

In fact, **Joins** let you select data from a table depending upon the data from another table. In the **Map Editor** context, the data of a **Main** table and of a **Lookup** table can be bound together on **expression keys**. In this case, the order of table does fully make sense.

Simply drop column names from one table to a subordinate one, to create a **Join** relationship between the two tables. This way, you can retrieve and process data from multiple inputs.

The join displays graphically as a purple link and creates automatically a key that will be used as a hash key to speed up the match search.

You can create direct joins between the main table and lookup tables. But you can also create indirect joins from the main table to a lookup table, via another lookup table. This requires a direct join between one of the **Lookup** table to the **Main** one.



You cannot create a **Join** from a subordinate table towards a superior table in the **Input** area.

The **Expression key** field which is filled in with the dragged and dropped data is editable in the input schema, whereas the column name can only be changed from the **Schema editor** panel.

You can either insert the dragged data into a new entry or replace the existing entries or else concatenate all selected data into one cell.

Cars

Property	Value
Lookup Model	Load once
Match Model	All matches
Join Model	Inner Join
Store temp data	false
Schema Type	Built-In

Expr. key	Column
Owners.ID_Owner	ID_Owner
	Registration
	Make
	Color
	ID_Reseller

Cars.Make.equals("BMW") ||
Cars.Make.equals("Mercedes")

Resellers

Property	Value
Lookup Model	Load once
Match Model	All matches
Join Model	Inner Join
Store temp data	false
Schema Type	Built-In

Expr. key	Column
Cars.ID_Reseller	ID_Reseller
	Name_Reseller
	Address_Reseller
	City
	ZIP

For further information about possible types of drag and drops, see [Section 8.2.4, "Mapping the Output setting"](#).



If you have a big number of input tables, you can use the minimize/maximize icon to reduce or restore the table size in the **Input** area. The Join binding two tables remains visible even though the table is minimized.

Creating a Join automatically assigns a hash key onto the joined field name. The key symbol displays in violet on the input table itself and is removed when the Join between the two tables is removed.

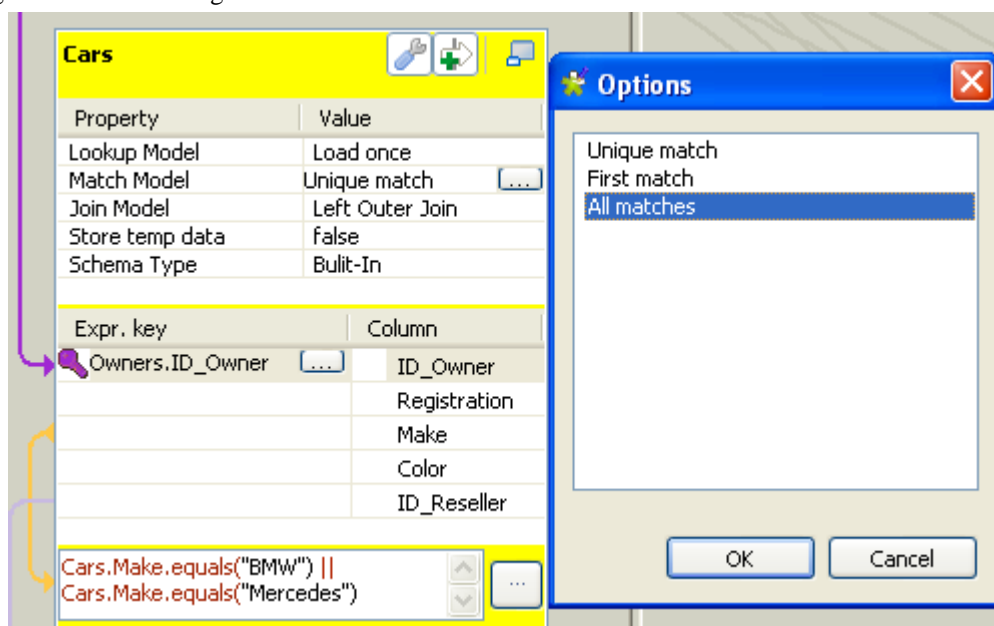
Related topics:

- [Section 8.2.5, “Setting schemas in the Map Editor”](#)
- [Section 8.2.1.3, “How to use Inner Join”](#)

Along with the explicit Join you can select whether you want to filter down to a unique match or if you allow several matches to be taken into account. In this last case, you can choose to consider only the first or the last match or all of them.

To define the match model for an explicit Join:

1. Click the **tMap settings** button at the top of the table to which the Join links to display the table properties.
2. Click in the **Value** field corresponding to **Match Model** and then click the three-dot button that appears to open the **[Options]** dialog box.
3. In the **[Options]** dialog box, double-click the wanted match model, or select it and click **OK** to validate the setting and close the dialog box.



Unique Match

This is the default selection when you implement an explicit Join. This means that only the last match from the Lookup flow will be taken into account and passed on to the output.

The other matches will be then ignored.

First Match

This selection implies that several matches can be expected in the lookup. The First Match selection means that in the lookup only the first encountered match will be taken into account and passed onto the main output flow.

The other matches will then be ignored.

All Matches

This selection implies that several matches can be expected in the lookup flow. In this case, all matches are taken into account and passed on to the main output flow.

8.2.1.3. How to use Inner Join

The **Inner join** is a particular type of Join that distinguishes itself by the way the rejection is performed.

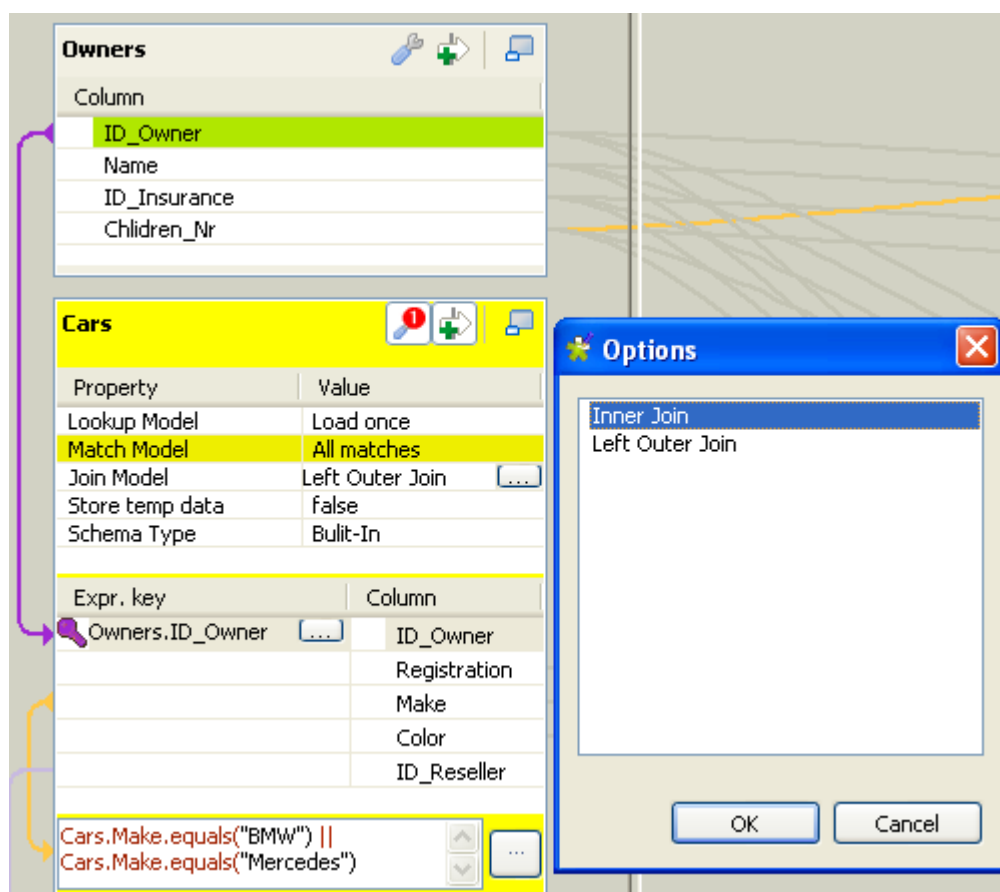
This option avoids that null values are passed on to the main output flow. It allows also to pass on the rejected data to a specific table called **Inner Join Reject** table.

If the data searched cannot be retrieved through the explicit Join or the filter Join, in other words, the Inner Join cannot be established for any reason, then the requested data will be rejected to the Output table defined as **Inner Join Reject** table if any.

Simply drop column names from one table to a subordinate one, to create a **Join** relationship between the two tables. The Join is displayed graphically as a purple link and creates automatically a key that will be used as a hash key to speed up the match search.

To define the type of an explicit Join:

1. Click the **tMap settings** button at the top of the table to which the Join links to display the table properties.
2. Click in the **Value** field corresponding to **Join Model** and then click the three-dot button that appears to open the **[Options]** dialog box.
3. In the **[Options]** dialog box, double-click the wanted Join type, or select it and click **OK** to validate the setting and close the dialog box.



An **Inner Join** table should always be coupled to an **Inner Join Reject** table. For how to define an output table as an **Inner Join Reject** table, see [Section 8.2.4.4, “Lookup Inner Join rejection”](#).

You can also use the filter button to decrease the number of rows to be searched and improve the performance (in Java).

Related topics:

- [Section 8.2.4.4, “Lookup Inner Join rejection”](#)
- [Section 8.2.1.5, “How to filter an input flow”](#)

8.2.1.4. How to use the All Rows option

By default, without a Join set up, in each input table of the input area of the **Map Editor**, the **All rows** match model option is selected. This **All rows** option means that all the rows are loaded from the **Lookup** flow and searched against the **Main** flow.

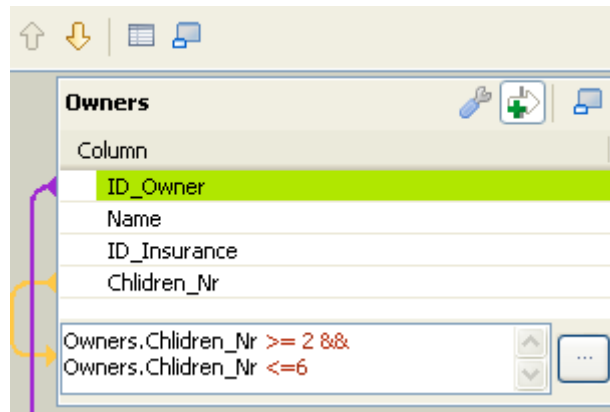
The output corresponds to the Cartesian product of both table (or more tables if need be).



If you create an explicit or an inner Join between two tables, the **All rows** option is no longer available. You then have to select **Unique match**, **First match** or **All matches**. For more information, see [Section 8.2.1.2, “How to use Explicit Join”](#) and [Section 8.2.1.3, “How to use Inner Join”](#).

8.2.1.5. How to filter an input flow

Click the **Filter** button next to the **tMap settings** button to add a **Filter** field.



In the **Filter** field, type in the condition to be applied. This allows to reduce the number of rows parsed against the main flow, enhancing the performance on long and heterogeneous flows.

You can use the Auto-completion tool via the **Ctrl+Space** keystrokes in order to reuse schema columns in the condition statement.

8.2.1.6. How to remove input entries from table

To remove input entries, click the red cross sign on the Schema Editor of the selected table. Press **Ctrl** or **Shift** and click fields for multiple selection to be removed.



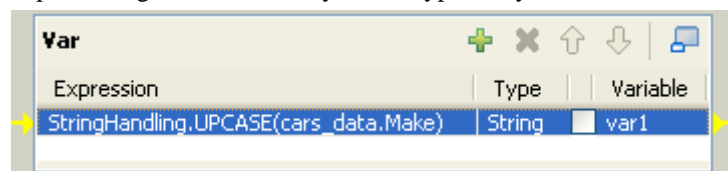
If you remove Input entries from the **Map Editor** schema, this removal also occurs in your component schema definition.

8.2.2. Mapping variables

The **Var** table (variable table) regroups all mapping variables which are used numerous times in various places.

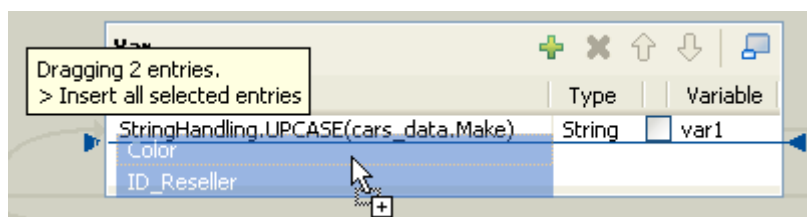
You can also use the **Expression** field of the **Var** table to carry out any transformation you want to, using Java Code.

Variables help you save processing time and avoid you to retype many times the same data.



There are various possibilities to create variables:

- Type in freely your variables in Java. Enter the strings between quotes or concatenate functions using the relevant operator.
- Add new lines using the plus sign and remove lines using the red cross sign. And press **Ctrl+Space** to retrieve existing global and context variables.
- Drop one or more **Input** entries to the **Var** table.



Select an entry on the Input area or press Shift key to select multiple entries of one Input table.

Press **Ctrl** to select either non-appended entries in the same input table or entries from various tables. When selecting entries in the second table, notice that the first selection displays in grey. Hold the **Ctrl** key down to drag all entries together. A tooltip shows you how many entries are in selection.

Then various types of drag-and-drops are possible depending on the action you want to carry out.

To...	You need to...
Insert all selected entries as separated variables.	Simply drag & drop to the Var table. Arrows show you where the new Var entry can be inserted. Each Input is inserted in a separate cell.
Concatenate all selected input entries together with an existing Var entry.	Drag & drop onto the Var entry which gets highlighted. All entries gets concatenated into one cell. Add the required operators using Java operations signs. The dot concatenates string variables.
Overwrite a Var entry with selected concatenated Input entries.	Drag & drop onto the relevant Var entry which gets highlighted then press Ctrl and release. All selected entries are concatenated and overwrite the highlighted Var.
Concatenate selected input entries with highlighted Var entries and create new Var lines if needed	Drag & drop onto an existing Var then press Shift when browsing over the chosen Var entries. First entries get concatenated with the highlighted Var entries. And if necessary new lines get created to hold remaining entries.

8.2.2.1. How to access global or context variables

Press **Ctrl+Space** to access the global and context variable list.

Appended to the variable list, a metadata list provides information about the selected column.

8.2.2.2. How to remove variables

To remove a selected **Var** entry, click the red cross sign. This removes the whole line as well as the link.

Press **Ctrl** or **Shift** and click fields for multiple selection then click the red cross sign.

8.2.3. Using the expression editor

All expressions (**Input**, **Var** or **Output**) and constraint statements can be viewed and edited from the expression editor. This editor provides visual comfort to write any function or transformation in a handy dedicated view.

8.2.3.1. How to access the expression editor

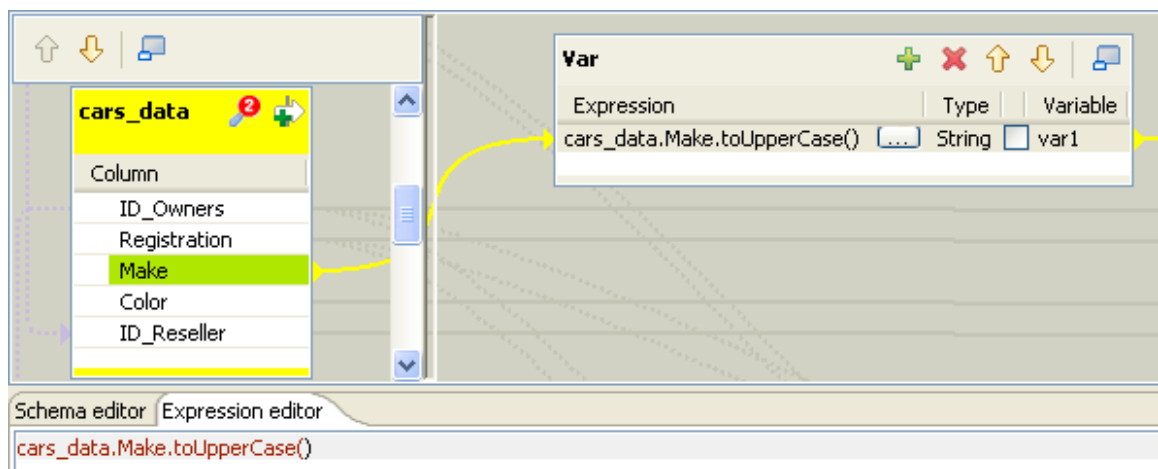
You can write the expressions necessary for the data transformation directly in the **Expression editor** view located in the lower half of the expression editor, or you can open the **[Expression Builder]** dialog box where you can write the data transformation expressions.

To open the **Expression editor** view, complete the following:

1. Double-click the **tMap** component in your job design to open the **Map Editor**.
2. In the lower half of the editor, click the **Expression editor** tab to open the corresponding view.



To edit an expression, select it in the **Input** panel and then click the **Expression editor** tab and modify the expression as required.

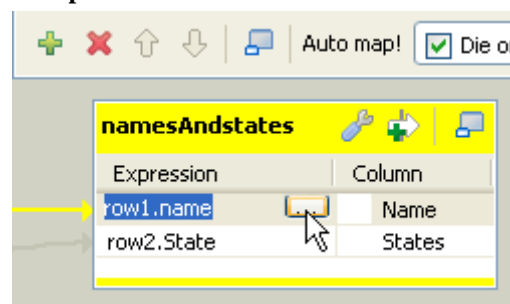


3. Enter the Java code according to your needs. The corresponding expression in the output panel is synchronized.

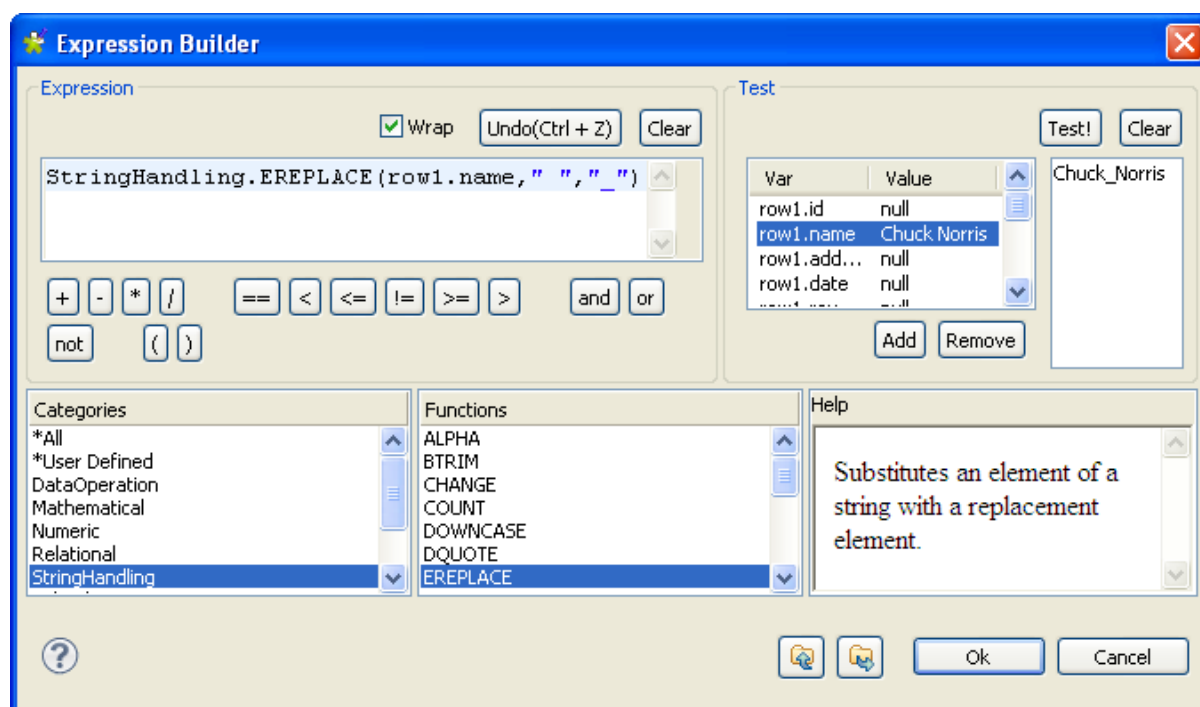


Refer to the Java documentation for more information regarding functions and operations.

To open the **[Expression Builder]** dialog box, click the three-dot button next to the expression you want to open in the **Var** or **Output** panel of the **Map Editor**.



The **[Expression Builder]** dialog box opens on the selected expression.

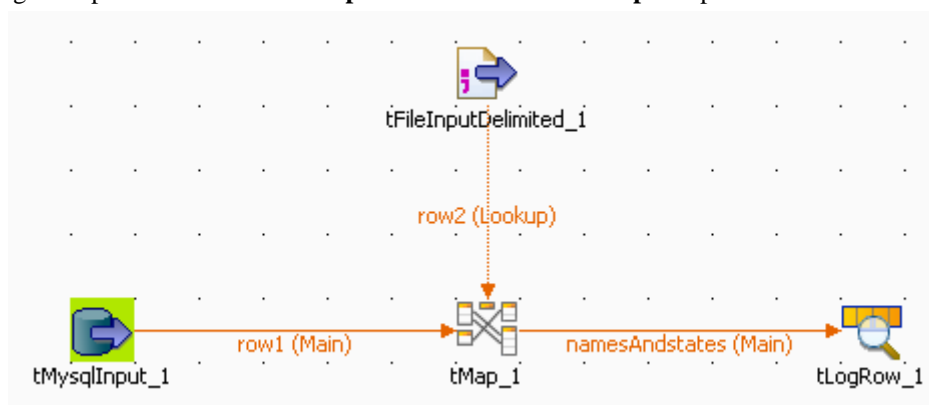


For a use case showing the usage of the expression editor, see the following section.

8.2.3.2. How to write code using the Expression Builder

Some Jobs require pieces of code to be written in order to provide components with parameters. In the **Component** view of some components, an **Expression Builder** interface can help you build these pieces of code (in Java).

The following example shows the use of **Expression Builder** in a **tMap** component.



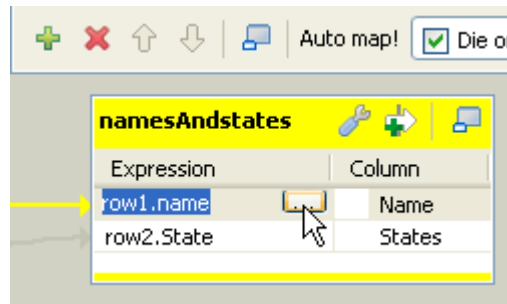
Two input flows are connected to the **tMap** component.

- From the DB input, comes a list of names made of a first name and a last name separated by a space char.
- From the File input, comes a list of US states, in lower case.

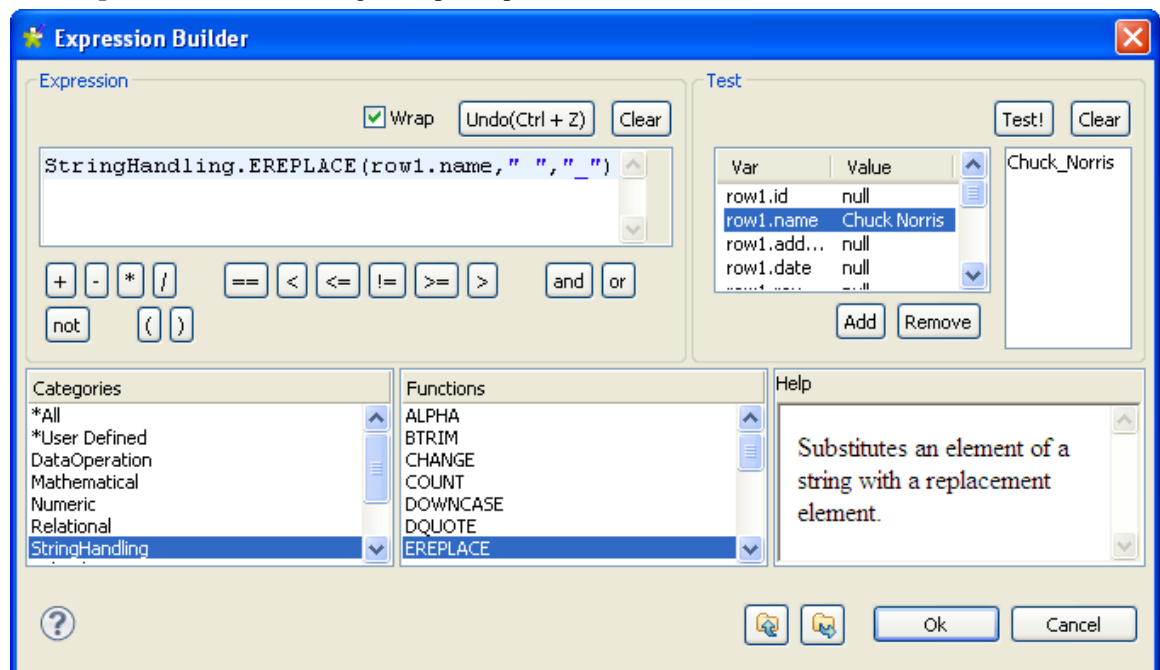
In the **tMap**, use the expression builder to: First, replace the blank char separating the first and last names with an underscore char, and second, change the states from lower case to upper case.

1. In the **tMap**, set the relevant inner join to set the reference mapping. For more information regarding **tMap**, see [Section 8.2, “tMap operation”](#) and [Section 8.1, “tMap and tXMLMap interfaces”](#).

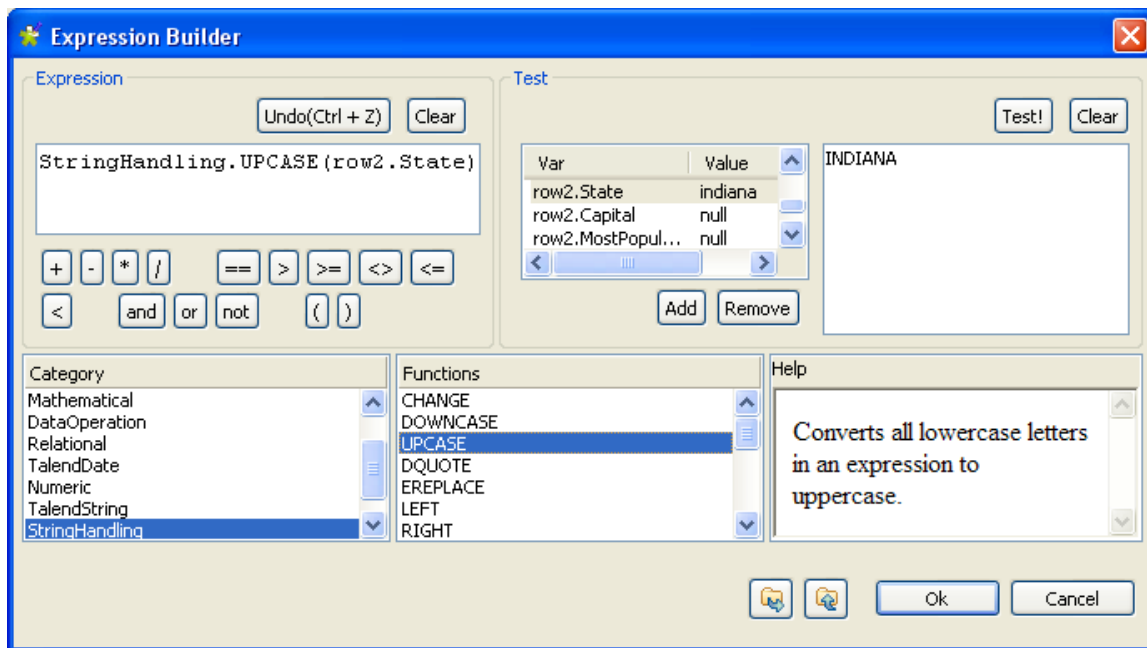
- From the main (*row1*) input, drop the *Names* column to the output area, and the *State* column from the lookup (*row2*) input towards the same output area.
- Then click in the first **Expression** field (*row1.Name*) to display the three-dot button.



The [Expression Builder] dialog box opens up.



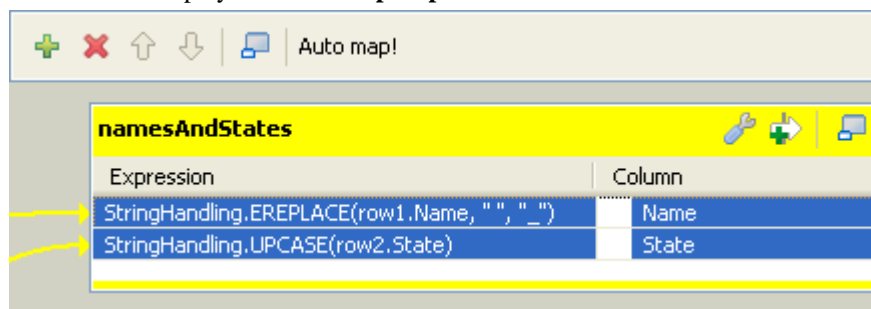
- In the **Category** area, select the relevant action you want to perform. In this example, select *StringHandling* and select the *EREPLACE* function.
- In the **Expression** area, paste *row1.Name* in place of the text expression, in order to get: `StringHandling.EREPLACE(row1.Name, " ", "_")`. This expression will replace the separating space char with an underscore char in the char string given.
- Now check that the output is correct, by typing in the relevant **Value** field of the **Test** area, a dummy value, e.g: *Chuck Norris* and clicking **Test!**. The correct change should be carried out, for example, *Chuck_Norris*.
- Click **OK** to validate the changes, and then proceed with the same operation for the second column (*State*).
- In the **tMap** output, select the *row2.State* Expression and click the [...] button to open the **Expression builder** again.



This time, the StringHandling function to be used is **UPCASE**. The complete expression says: `StringHandling.UPCASE(row2.State)`.

- Once again, check that the expression syntax is correct using a dummy **Value** in the **Test** area, for example *indiana*. The **Test!** result should display **INDIANA** for this example. Then, click **OK** to validate the changes.

Both expressions are now displayed in the **tMap Expression** field.



These changes will be carried out along the flow processing. The output of this example is as shown below.

Starting job NamesAndStates at 10:02 10/10/2007.


tLogRow_1	
Name	RandomStates
William_Grant	IOWA
William_Hoover	NEW YORK
Grover_Lincoln	NORTH DAKOTA
Lyndon_Jefferson	OHIO
Gerald_Hayes	WASHINGTON
Benjamin_Grant	MAINE
George_Pierce	CONNECTICUT
Jimmy_Reagan	ALASKA
Martin_Hayes	WASHINGTON
Franklin_Jefferson	IOWA
Andrew_Nixon	NEW HAMPSHIRE

8.2.4. Mapping the Output setting

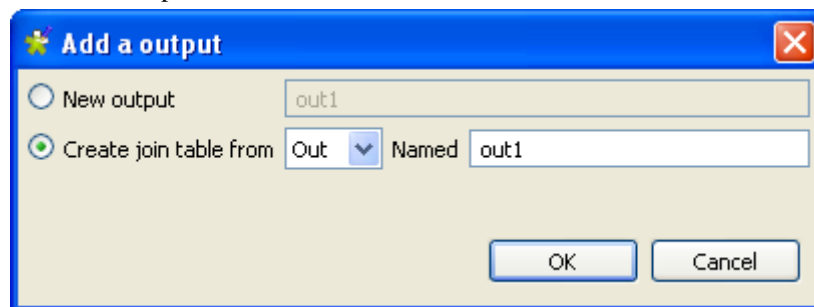
On the design workspace, the creation of a **Row** connection from the **tMap** component to the output components adds Output schema tables in the **Map Editor**.

You can also add an Output schema in your **Map Editor**, using the plus sign from the tool bar of the Output area.

You have as well the possibility to create a join between your output tables. The join on the tables enables you to process several flows separately and unite them in a single output. For more information about the output join tables feature, see *Talend Open Studio for ESB Reference Guide*.

 The join table retrieves the schema of the source table.

When you click the [+] button to add an output schema or to make a join between your output tables, a dialog box opens. You have then two options.



Select...	To...
New output	Add an independent table.
Create join table from	Create a join between output tables. In order to do so, select in the drop down list the table from which you want to create the join. In the Named field, type in the name of the table to be created.

Unlike the **Input** area, the order of output schema tables does not make such a difference, as there is no subordination relationship between outputs (of Join type).

Once all connections, hence output schema tables, are created, you can select and organize the output data via drag & drops.

You can drop one or several entries from the **Input** area straight to the relevant output table.

Press **Ctrl** or **Shift**, and click entries to carry out multiple selection.

Or you can drag expressions from the **Var** area and drop them to fill in the output schemas with the appropriate reusable data.

Note that if you make any change to the Input column in the Schema Editor, a dialog prompts you to decide to propagate the changes throughout all Input/Variable/Output table entries, where concerned.

Action	Result
Drag & Drop onto existing expressions.	Concatenates the selected expression with the existing expressions.
Drag & Drop to insertion line.	Inserts one or several new entries at start or end of table or between two existing lines.
Drag & Drop + Ctrl.	Replaces highlighted expression with selected expression.
Drag & Drop + Shift.	Adds the selected fields to all highlighted expressions. Inserts new lines if needed.

Action	Result
Drag & Drop + Ctrl + Shift.	Replaces all highlighted expressions with selected fields. Inserts new lines if needed.

You can add filters and rejections to customize your outputs.

8.2.4.1. Building complex expressions

If you have complex expressions to build, or advanced changes to be carried out on the output flow, then the Expression Builder interface can help in this task.

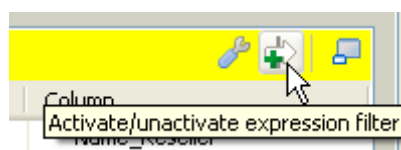
Click the **Expression** field of your input or output table to display the [...] button. Then click this three-dot button to open the **Expression Builder**.

For more information regarding the Expression Builder, see [Section 8.2.3.2, “How to write code using the Expression Builder”](#).

8.2.4.2. Filters

Filters allow you to make a selection among the input fields, and send only the selected fields to various outputs.

Click the [+] button at the top of the table to add a filter line.



You can enter freely your filter statements using Java operators and functions.

Drop expressions from the **Input** area or from the **Var** area to the Filter row entry of the relevant Output table.



An orange link is then created. Add the required Java operator to finalize your filter formula.

You can create various filters on different lines. The AND operator is the logical conjunction of all stated filters.

8.2.4.3. Output rejection

Reject options define the nature of an output table.

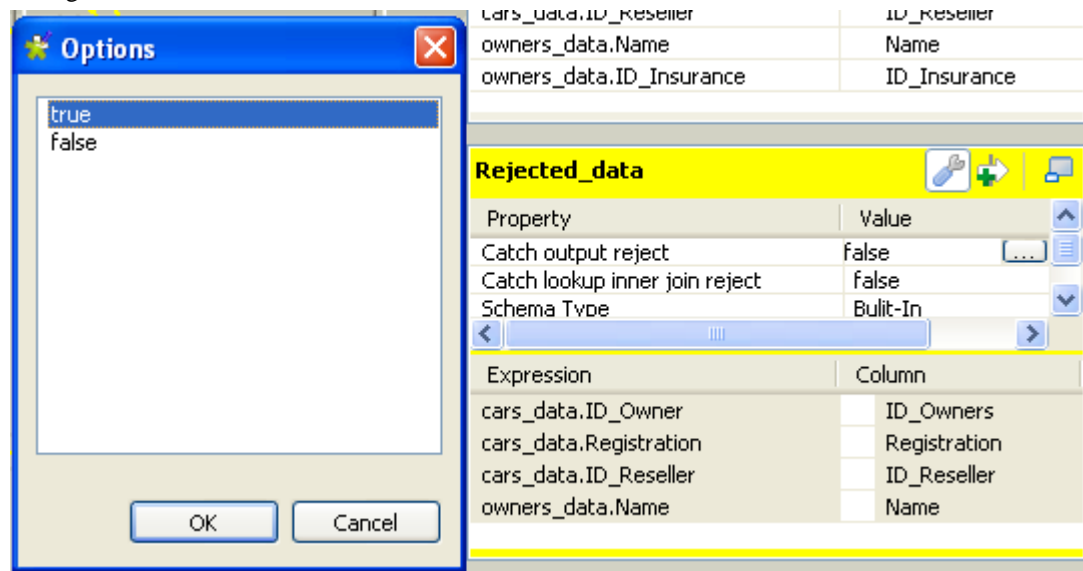
It groups data which do not satisfy one or more filters defined in the standard output tables. Note that as standard output tables, are meant all non-reject tables.

This way, data rejected from other output tables, are gathered in one or more dedicated tables, allowing you to spot any error or unpredicted case.

The Reject principle concatenates all non Reject tables filters and defines them as an ELSE statement.

To define an output table as the Else part of the regular tables:

1. Click the **tMap settings** button at the top of the output table to display the table properties.
2. Click in the **Value** field corresponding to **Catch output reject** and then click the [...] button that appears to display the **[Options]** dialog box.
3. In the **[Options]** dialog box, double-click **true**, or select it and click **OK** to validate the setting and close the dialog box.



You can define several Reject tables, to offer multiple refined outputs. To differentiate various Reject outputs, add filter lines, by clicking on the plus arrow button.

Once a table is defined as Reject, the verification process will be first enforced on regular tables before taking in consideration possible constraints of the Reject tables.

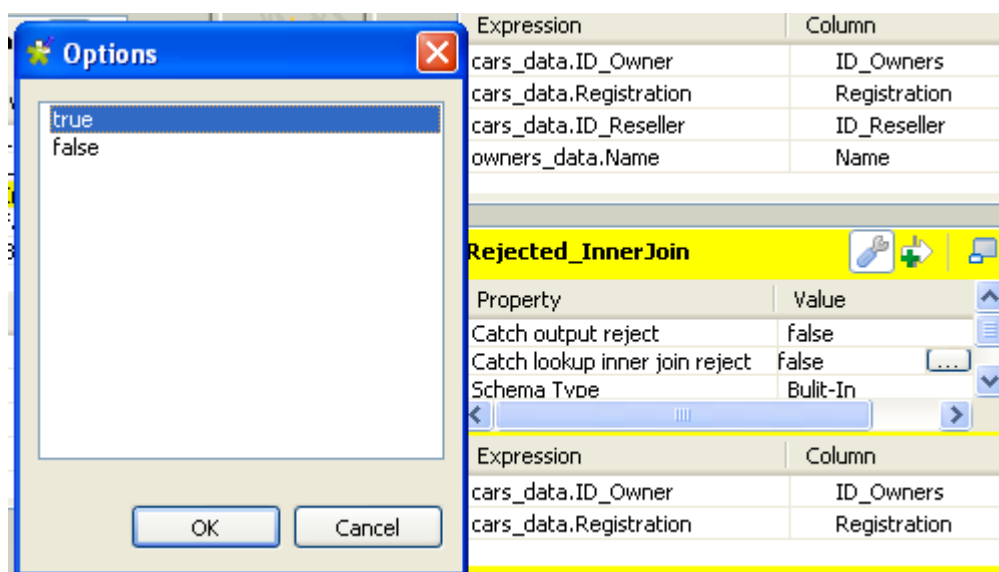
Note that data are not exclusively processed to one output. Although a data satisfied one constraint, hence is routed to the corresponding output, this data still gets checked against the other constraints and can be routed to other outputs.

8.2.4.4. Lookup Inner Join rejection

The Inner Join is a Lookup Join. The Inner Join Reject table is a particular type of Rejection output. It gathers rejected data from the main row table after an Inner Join could not be established.

To define an Output flow as container for rejected Inner Join data, create a new output component on your Job that you connect to the **Map Editor**. Then in the **Map Editor**, follow the steps below:

1. Click the **tMap settings** button at the top of the output table to display the table properties.
2. Click in the **Value** field corresponding to **Catch lookup inner join reject** and then click the [...] button that appears to display the **[Options]** dialog box.
3. In the **[Options]** dialog box, double-click **true**, or select it and click **OK** to validate the setting and close the dialog box.



8.2.4.5. Removing Output entries

To remove Output entries, click the cross sign on the Schema Editor of the selected table.

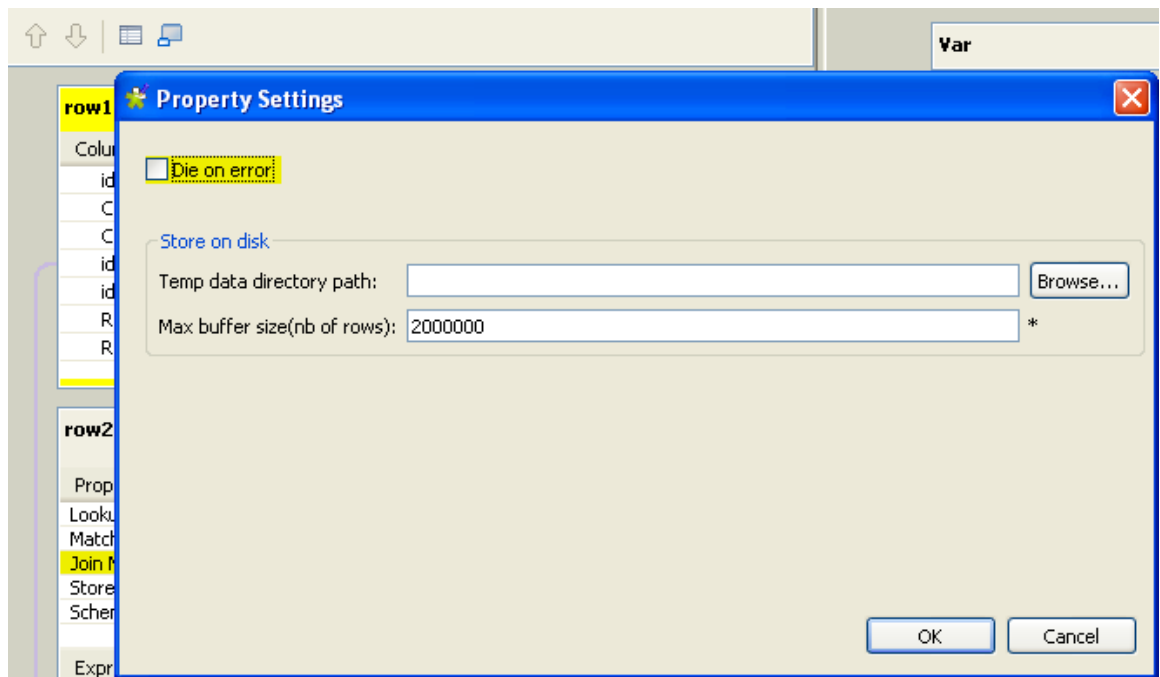
8.2.4.6. Handling errors

The **Die on error** option prevent error to be processed. To do so, it stops the Job execution as soon as an error is encountered. The **tMap** component provides this option to prevent processing erroneous data. The **Die on error** option is activated by default in **tMap**.

Deactivating the **Die on error** option will allow you to skip the rows on error and complete the process for error-free rows on one hand, and to retrieve the rows on error and manage them if needed.

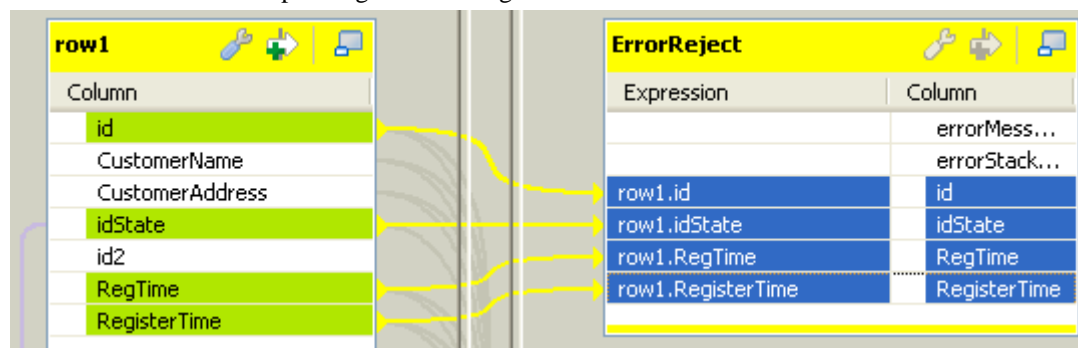
To deactivate the **Die on error** option:

1. Double-click the **tMap** component on the design workspace to open the **Map Editor**.
2. Click the **Property Settings** button at the top of the input area to display the **[Property Settings]** dialog box.
3. In **[Property Settings]** dialog box, clear the **Die on error** check box and click **OK**.

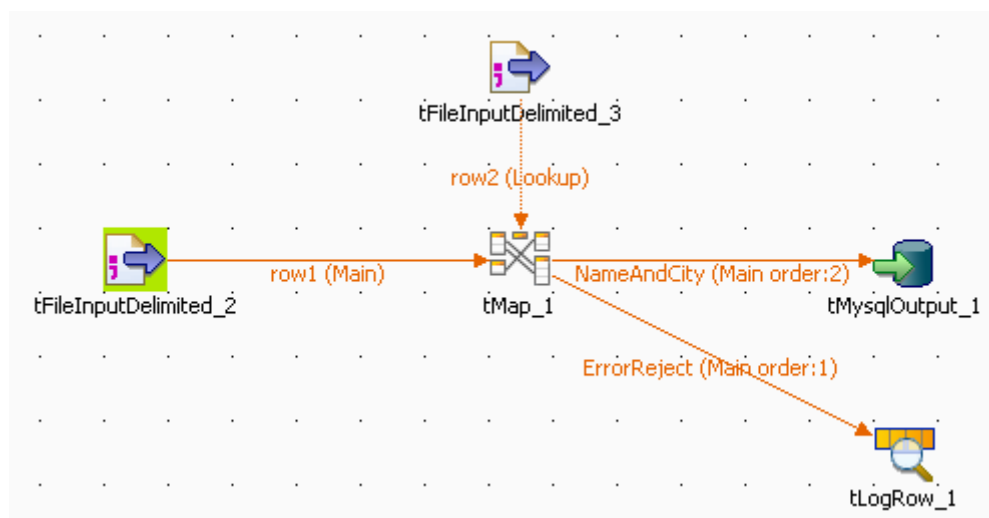


A new table called **ErrorReject** appears in the output area of the **Map Editor**. This output table automatically comprises two columns: **errorMessage** and **errorStackTrace**, retrieving the message and stack trace of the error encountered during the Job execution. Errors can be unparseable dates, null pointer exceptions, conversion issues, etc.

You can also drag and drop columns from the input tables to this error reject output table. Those erroneous data can be retrieved with the corresponding error messages and thus be corrected afterward.



Once the error reject table is set, its corresponding flow can be sent to an output component.



To do so, on the design workspace, right-click the **tMap** component, select **Row > ErrorReject** in the menu, and click the corresponding output component, here **tLogRow**.

When you execute the Job, errors are retrieved by the **ErrorReject** flow.

Starting job Die_on_error at 17:30 01/09/2010.

```
java.text.ParseException: Unparseable date: "08 01
1980"|java.lang.RuntimeException:
java.text.ParseException: Unparseable date: "08 01 1980"
    at routines.TalendDate.parseDate(TalendDate.java:503)
    at
doc.die_on_error_0_1.Die_on_error.tFileInputDelimited_2Pro
cess(Die_on_error.java:1409)
    at
doc.die_on_error_0_1.Die_on_error.runJobInTOS(Die_on_error.
java:2262)
    at
doc.die_on_error_0_1.Die_on_error.main(Die_on_error.java:2
160)
Caused by: java.text.ParseException: Unparseable date: "08
01 1980"
    at java.text.DateFormat.parse(Unknown Source)
    at routines.TalendDate.parseDate(TalendDate.java:501)
    ... 3 more
|1|08 01 1980
Job Die_on_error ended at 17:30 01/09/2010. [exit code=0]
```

The result contains the error message, its stack trace, and the two columns, *id* and *date*, dragged and dropped to the **ErrorReject** table, separated by a pipe “|”.

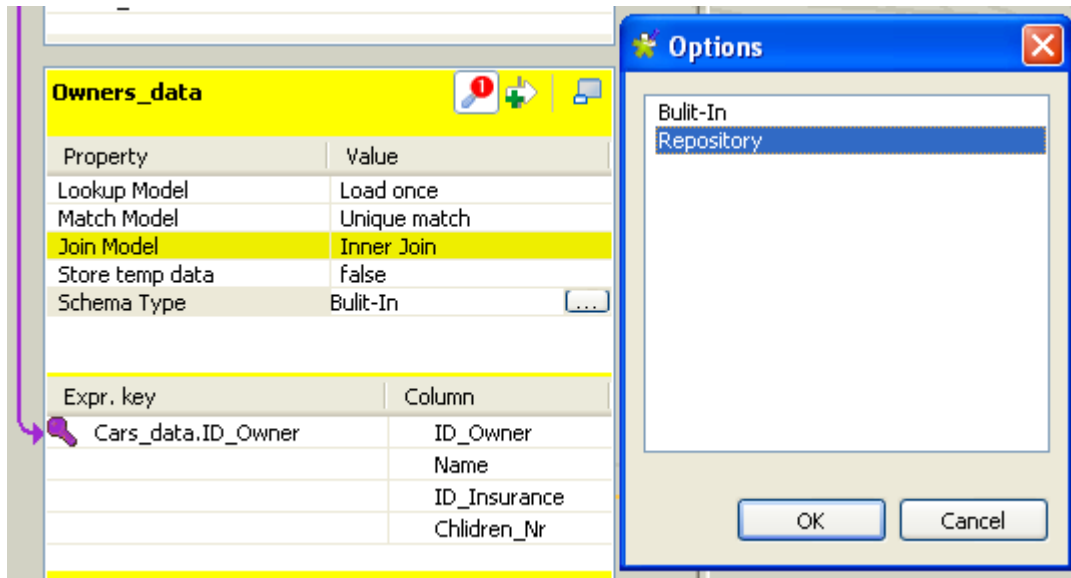
8.2.5. Setting schemas in the Map Editor

In the **Map Editor**, you can define the type of a table schema as **Built-In** so that you can modify the data structure in the **Schema editor** panel, or **Repository** and retrieve the data structure from the Repository. By default, the schema type is set to **Built-In** for all tables.

8.2.5.1. Retrieving the schema structure from the Repository

To retrieve the schema structure of the selected table from the Repository:

1. Click the **tMap Settings** button at the top of the table to display the table properties.
2. Click in the **Value** field of **Schema Type**, and then click the three-dot button that appears to open the **[Options]** dialog box.



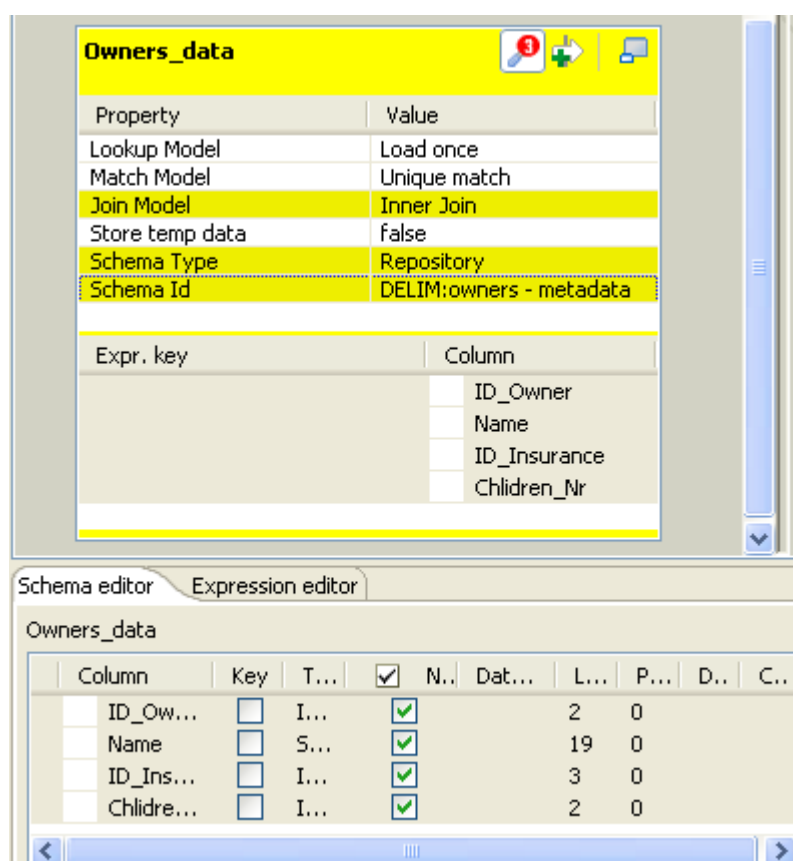
3. In the **[Options]** dialog box, double-click **Repository**, or select it and click **OK**, to close the dialog box and display the **Schema Id** property beneath **Schema Type**.



If you close the **Map Editor** now without specifying a Repository schema item, the schema type changes back to **Built-In**.

4. Click in the **Value** field of **Schema Id**, and then click the **[...]** button that appears to display the **[Repository Content]** dialog box.
5. In the **[Repository Content]** dialog box, select your schema as you define a centrally stored schema for any component, and then click **OK**.

The **Value** field of **Schema Id** is filled with the schema you just selected, and everything in the **Schema editor** panel for this table becomes read-only.



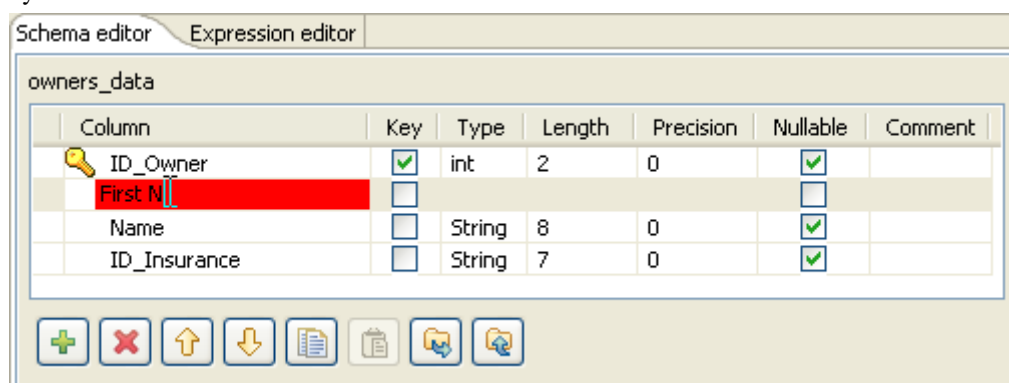
Changing the schema type of the subordinate table across a Join from **Built-In** to **Repository** causes the Join to get lost.



Changes to the schema of a table made in the **Map Editor** are automatically synchronized to the schema of the corresponding component connected with the **tMap** component.


8.2.5.2. Using the Schema Editor

The **Schema Editor** details all fields of the selected table. With the schema type of the table set to **Built-In**, you can modify the schema of the table.



Use the tool bar below the schema table, to add, move or remove columns from the schema.

You can also load a schema from the repository or export it into a file.

Metadata	Description
Column	Column name as defined on the Map Editor schemas and on the Input or Output component schemas.
Key	The Key shows if the expression key data should be used to retrieve data through the Join link. If unchecked, the Join relation is disabled.
Type	Type of data: String, Integer, Date, etc.  This column should always be defined in a Java version.
Length	-1 shows that no length value has been defined in the schema.
Precision	precise the length value if any is defined.
Nullable	Clear this check box if the field value should not be null.
Default	Shows any default value that may be defined for this field.
Comment	Free text field. Enter any useful comment.



Input metadata and output metadata are independent from each other. You can, for instance, change the label of a column on the output side without the column label of the input schema being changed.

However, any change made to the metadata are immediately reflected in the corresponding schema on the **tMap** relevant (Input or Output) area, but also on the schema defined for the component itself on the design workspace.

A Red colored background shows that an invalid character has been entered. Most special characters are prohibited in order for the Job to be able to interpret and use the text entered in the code. Authorized characters include lower-case, upper-case, figures except as start character.

8.2.6. Solving memory limitation issues in tMap use

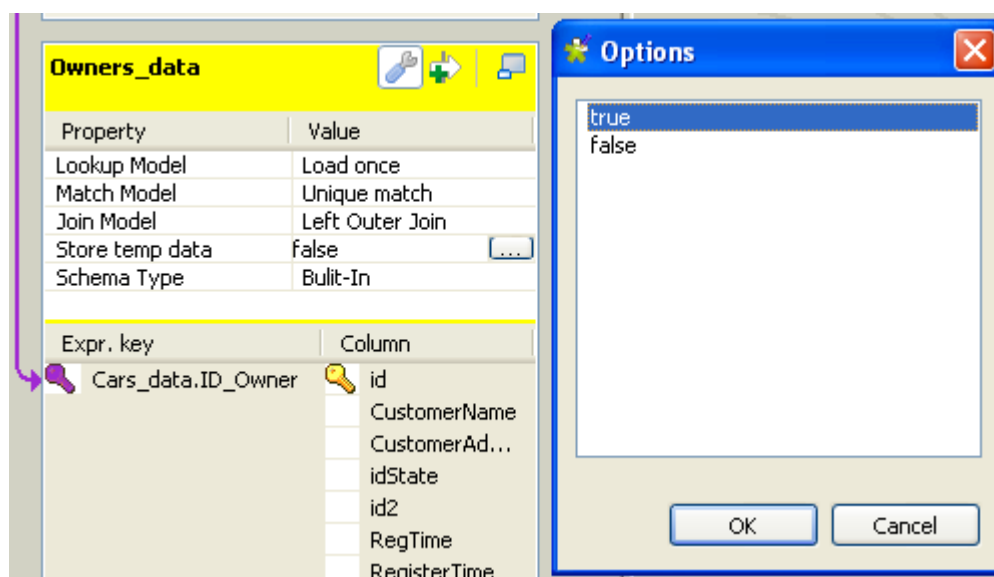
When handling large data sources, including for example, numerous columns, large number of lines or of column types, your system might encounter memory shortage issues that prevent your Job, to complete properly, in particular when using a **tMap** component for your transformation.

A feature has been added (in Java only for the time being) to the **tMap** component, in order to reduce the memory in use for lookup loading. In fact, rather than storing the temporary data in the system memory and thus possibly reaching the memory limitation, the **Store temp data** option allows you to choose to store the temporary data onto a directory of your disk instead.

This feature comes as an option to be selected in the Lookup table of the input data in the **Map Editor**.

To enable the **Store temp data** option:

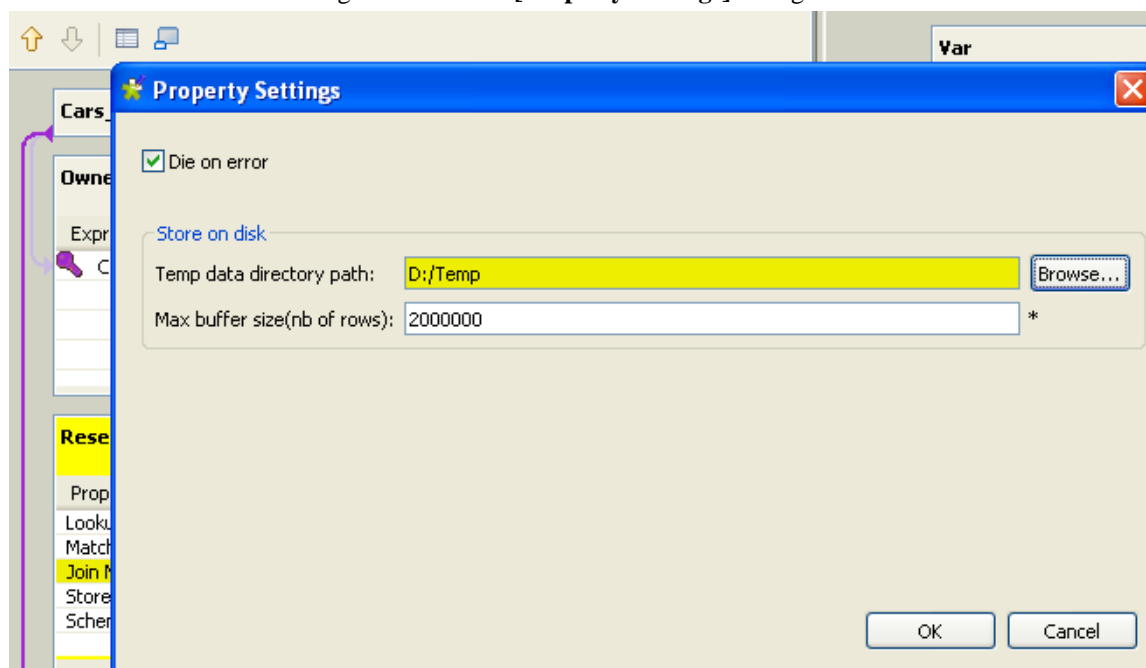
1. Double-click the **tMap** component in your Job to launch the **Map Editor**.
2. In input area, click the Lookup table describing the temporary data you want to be loaded onto the disk rather than in the memory.
3. Click the **tMap settings** button to display the table properties.
4. Click in the **Value** field corresponding to **Store temp data**, and then click the [...] button to display the **[Options]** dialog box.
5. In the **[Options]** dialog box, double-click **true**, or select it and click **OK**, to enable the option and close the dialog box.



For this option to be fully activated, you also need to specify the directory on the disk, where the data will be stored, and the buffer size, namely the number of rows of data each temporary file will contain. You can set the temporary storage directory and the buffer size either in the **Map Editor** or in the **tMap** component property settings.

To set the temporary storage directory and the buffer size in the **Map Editor**:

1. Click the **Property Settings** button at the top of the input area to display the **[Property Settings]** dialog box.
2. In **[Property Settings]** dialog box, fill the **Temp data directory path** field with the full path to the directory where the temporary data should be stored.
3. In the **Max buffer size (nr of rows)** field, specify the maximum number of rows each temporary file can contain. The default value is 2,000,000.
4. Click **OK** to validate the settings and close the **[Property Settings]** dialog box.

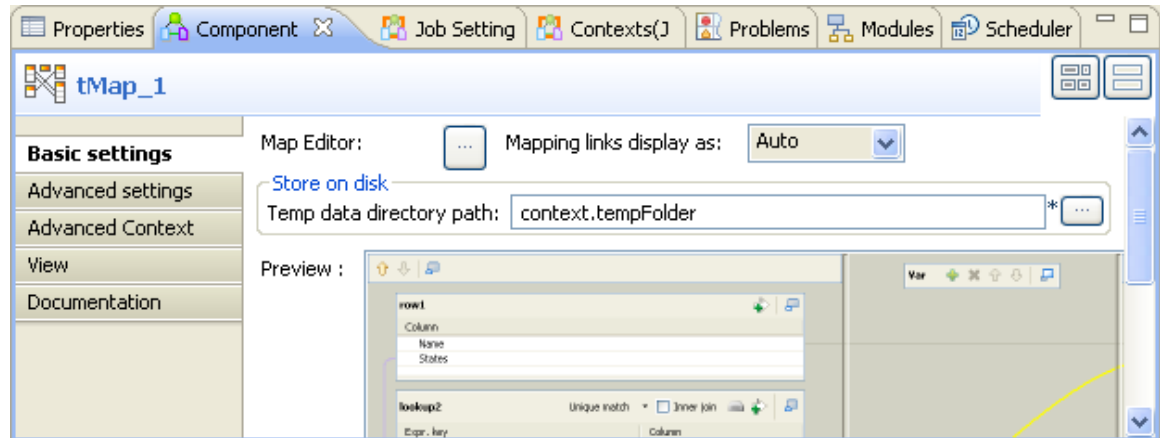


To set the temporary storage directory in the **tMap** component property settings without opening the **Map Editor**:

1. Click the **tMap** component to select it on the design workspace, and then select the **Component** tab to show the **Basic settings** view.

2. In the **Store on disk** area, fill the **Temp data directory path** field with the full path to the directory where the temporary data should be stored.

Alternatively, you can use a context variable through the **Ctrl+Space** bar if you have set the variable in a Context group in the repository. For more information about contexts, see [Section 4.4.2, “How to centralize contexts and variables”](#).



At the end of the subjob, the temporary files are cleared.

This way, you will limit the use of allocated memory per reference data to be written onto temporary files stored on the disk.



As writing the main flow onto the disk requires the data to be sorted, note that the order of the output rows cannot be guaranteed.

On the **Advanced settings** view, you can also set a buffer size if needed. Simply fill out the field **Max buffer size (nb of rows)** in order for the data stored on the disk to be split into as many files as needed.

8.2.7. Handling Lookups

In order to adapt to the multiple processing types as well as to address performance issues, the **tMap** component supports different lookup loading modes.

- **Load once:** Default setting. Select this option to load the entire lookup flow before processing the main flow. This is the preferred option if you have a great number of data from your main flow that needs to be requested in your lookup, or if your reference (or lookup) data comes from a file that can be easily loaded.
- **Reload at each row:** At each row, the lookup gets loaded again. This is mainly interesting in Jobs where the lookup volume is large, while the main flow is pretty small. Note that this option allows you to use dynamic variable settings such as where clause, to change/update the lookup flow on the fly as it gets loaded, before the main flow join is processed. This option could be considered as the counter-part of the **Store temp data** option that is available for file lookups.
- **Reload at each row (cache):** Expressions (in the Lookup table) are assessed and looked up in the cache first. The results of joins that have already been solved, are stored in the cache, in order to avoid loading the same results twice. This option optimizes the processing time and helps improve processing performance of the **tMap** component.

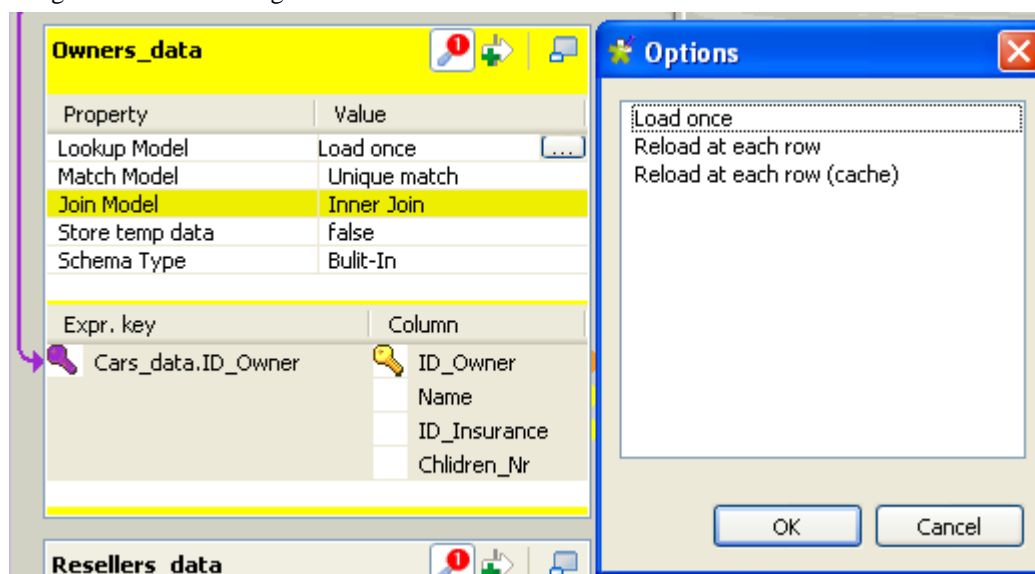


Note that for the time being, you cannot use **Reload at each row (cache)** and **Store temp data** at the same time.

To set the loading mode of a lookup flow:

1. Click the **tMap settings** button at the top of the lookup table to display the table properties.

- Click in the **Value** field corresponding to **Lookup Model**, and then click the [...] button to display the **[Options]** dialog box.
- In the **[Options]** dialog box, double-click the wanted loading mode, or select it and then click **OK**, to validate the setting and close the dialog box.



For use cases using these options, see the **tMap** section of the *Talend Open Studio Components Reference Guide*.



When your lookup is a database table, the best practise is to open the connection to the database in the beginning of your job design in order to optimize performance. For a use case using this option, see **tMap** in *Talend Open Studio Components Reference Guide*.

8.3. tXMLMap operation



Before starting this section, we recommend reading the previous **tMap** sections for the basic knowledge of a **Talend** mapping component.

tXMLMap is fine-tuned to leverage the **Document** data type for processing XML data, a case of transformation that often mixes hierarchical data (XML) and flat data together. This **Document** type carries a complete user-specific XML flow. In using **tXMLMap**, you are able to add as many input or output flows as required into a visual map editor to perform, on these flows, the operations as follows:

- data multiplexing and demultiplexing,
- data transformation on any type of fields, particularly on the **Document** type,
- data matching via different models, for example, the **Unique match** mode (related topic: [Section 8.2.1.2, “How to use Explicit Join”](#)),
- Automated XML tree construction on both of the input and the output sides,
- inner join and left outer join (related topic: [Section 8.2.1.3, “How to use Inner Join”](#))
- lookup between data sources whatever they are flat or XML data using models like **Load once** (related topic: [Section 8.2.7, “Handling Lookups”](#)),
- fields concatenation and interchange,
- field filtering using constraints,

- data rejecting.

Like **tMap**, a map editor is required to configure these operations. To open this map editor, you can double-click the **tXMLMap** icon in the design workspace, or alternatively, click the three-dot button next to the **Map Editor** in the **Basic settings** view of the **tXMLMap** component.

tXMLMap and **tMap** use the common approaches to accomplish most of these operations. Therefore, the following sections explain only the particular operations to which **tXMLMap** is dedicated for processing the hierarchical XML data.

The operations focusing on hierarchical data are:

- using the **Document** type to create the XML tree;
- managing the output XML data;
- editing the XML tree schema.

The following sections present more relevant details.



Different from **tMap**, **tXMLMap** does not provide the **Store temp data** option for storing temporary data onto the directory of your disk. For further information about this option of **tMap**, see [Section 8.2.6, “Solving memory limitation issues in tMap use”](#).

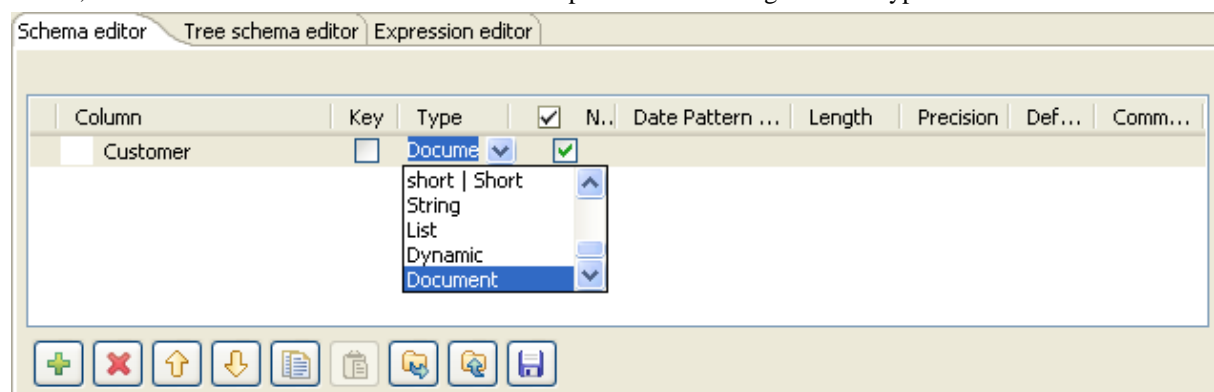
8.3.1. Using the document type to create the XML tree

The **Document** data type fits perfectly the conception of defining XML structure as easily as possible. When you need the XML tree structure to map the input or output flow or both, use this type. Then you can import the XML tree structure from various XML sources and edit the tree directly in the mapping editor, thus saving the manual efforts.

8.3.1.1. How to set up the Document type

The **Document** data type is one of the data types provided by **Talend**. This **Document** type is set up when you edit the schema for the corresponding data in the **Schema editor**. For further information about the schema editor, see [Section 8.2.5.2, “Using the Schema Editor”](#).

The following figure presents an example in which the input flow, *Customer*, is set up as the **Document** type. To replicate it, in the Map editor, you can simply click the **[+]** button to add one row on the input side of the **Schema editor**, rename it and select **Document** from the drop-down list of the given data types.



In practice for most cases, **tXMLMap** retrieves the schema of its preceding or succeeding components, for example, from a **tFileInputXML** component or in the ESB use case, from a **tESBProviderRequest** component.

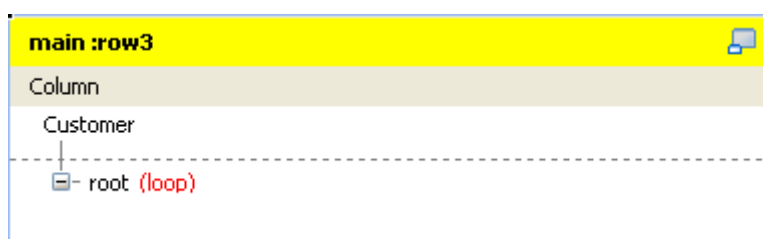
This avoids many manual efforts to set up the **Document** type for the XML flow to be processed. However, to continue to modify the XML structure as the content of a Document row, you need still to use the given Map editor.



Be aware that a **Document** flow carries a user-defined XML tree and is no more than one single field of a schema, which, same as the other schemas, may contain different data types between each field. For further information about how to set a schema, see [Section 4.2.6.1, “Basic Settings tab”](#).

Once the **Document** type is set up for a row of data, in the corresponding data flow table in the map editor, a basic XML tree structure is created automatically to reflect the details of this structure. This basic structure represents the minimum element required by a valid XML tree in using **tXMLMap**:

- The root element: it is the minimum element required by an XML tree to be processed and when needs be, the foundation to develop a sophisticated XML tree.
- The loop element: it determines the element over which the iteration takes place to read the hierarchical data of an XML tree. By default, the root element is set as loop element.



This figure gives an example with the input flow, *Customer*. Based on this generated XML root tagged as **root** by default, you can develop the XML tree structure of interest.

To do this, you need to:

1. Import the custom XML tree structure from one of the following types of sources:
 - XML or XSD files (related topic: [Section 8.3.1.2, “How to import the XML tree structure from XML and XSD files”](#))



When you import an XSD file, you will create the XML structure this XSD file describes.

- file XML connections created and stored in the **Repository** of your Studio (related topic: [Section 8.3.1.3, “How to import the XML tree structure from the Repository”](#)).





If needs be, you can develop the XML tree of interest manually using the options provided on the contextual menu.

2. Reset the loop element for the XML tree you are creating, if needs be. You can set as many loops as you need to. At this step, you may have to consider the following situations:
 - If you have to create several XML trees, you need to define the loop element for each of them.
 - If you import the XML tree from the **Repository**, the loop element will have been set depending on the set of the source structure. But you can still reset the loop element.

For further details, see [Section 8.3.1.4, “How to set or reset a loop element for an imported XML structure”](#)

If needed, you can continue to modify the imported XML tree using the options provided in the contextual menu. The following table presents the operations you can perform through the available options.

Options	Operations
Create Sub-element and Create Attribute	Add elements or attributes to develop an XML tree. Related topic: Section 8.3.1.5, “How to add a sub-element or an attribute to an XML tree structure”

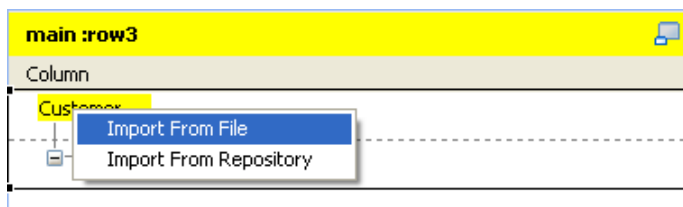
Options	Operations
Set a namespace	Add and manage given namespaces on the imported XML tree. Related topic: Section 8.3.1.7, “How to manage a namespace”
Delete	Delete an element or an attribute. Related topic: Section 8.3.1.6, “How to delete an element or an attribute from the XML tree structure”
Rename	Rename an element or an attribute.
As loop element	Set or reset an element as loop element. Multiple loop elements and optional loop element are supported.
As optional loop	<p>This option is not available unless to the loop element you have defined.</p> <p>When the corresponding element exists in the source file, an optional loop element works the same way as a normal loop element; otherwise, it resets automatically its parent element as loop element or in absence of parent element in the source file, it takes the element of the higher level until the root element. But in the real-world practice, with such differences between the XML tree and the source file structure, we recommend adapting the XML tree to the source file for better performance.</p>
As group element	On the XML tree of the output side, set an element as group element. Related topic: Section 8.3.1.8, “How to group the output data”
As aggregate element	On the XML tree of the output side, set an element as aggregate element. Related topic: Section 8.3.1.9, “How to aggregate the output data”
Add Choice	<p>Set the Choice element. Then all of its child elements developed underneath will be contained in this declaration. This Choice element originates from one of the XSD concepts. It enables tXMLMap to perform the function of the XSD Choice element to read or write a Document flow.</p> <p>When tXMLMap processes a choice element, the elements contained in its declaration will not be outputted unless their mapping expressions are appropriately defined.</p> <p> The tXMLMap component declares automatically any Choice element set in the XSD file it imports.</p>
Set as Substitution	<p>Set the Substitution element to specify the element substitutable for a given head element defined in the corresponding XSD. The Substitution element enables tXMLMap to perform the function of the XSD Substitution element to read or write a Document flow</p> <p>When tXMLMap processes a substitution element, the elements contained in its declaration will not be outputted unless their mapping expressions are appropriately defined.</p> <p> The tXMLMap component declares automatically any Substitution element set in the XSD file it imports.</p>

The following sections present more details about the process of creating the XML tree.

8.3.1.2. How to import the XML tree structure from XML and XSD files

To import the XML tree structure from an XML file, proceed as follows:

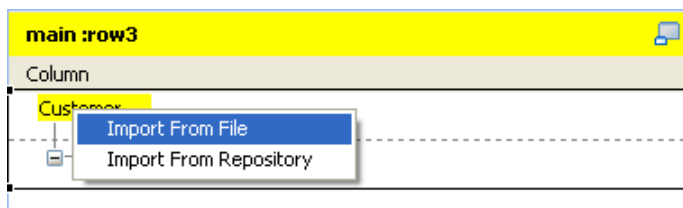
1. In the input flow table of interest, right-click the column name to open the contextual menu. In this example, it is *Customer*.



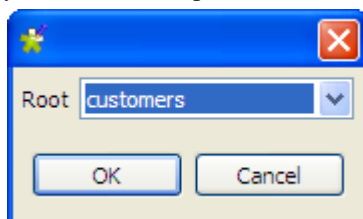
2. From this menu, select **Import From File**.
3. In the pop-up dialog box, browse to the XML file you need to use to provide the XML tree structure of interest and double-click the file.

To import the XML tree structure from an XSD file, proceed as follows:

1. In the input flow table of interest, right-click the column name to open the contextual menu. In this example, it is *Customer*.



2. From this menu, select **Import From File**.
3. In the pop-up dialog box, browse to the XSD file you need to use to provide the XML tree structure of interest and double-click the file.
4. In the dialog box that appears, select an element from the **Root** list as the root of your XML tree, and click **OK**. Then the XML tree described by the XSD file imported is established.



The root of the imported XML tree is adaptable:

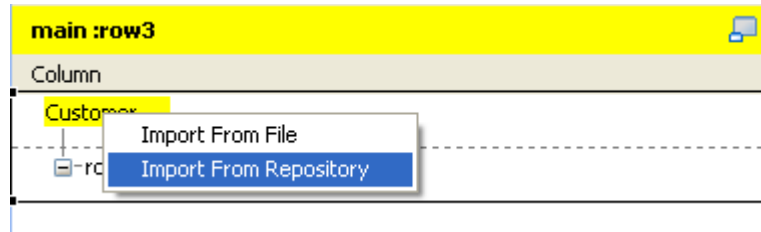
- When importing either an input or an output XML tree structure from an XSD file, you can choose an element as the root of your XML tree.
- Once an XML structure is imported, the **root** tag is renamed automatically with the name of the XML source. To change this root name manually, you need use the tree schema editor. For further information about this editor, see [Section 8.3.3, “Editing the XML tree schema”](#).

Then, you need to define the loop element in this XML tree structure. For further information about how to define a loop element, see [Section 8.3.1.4, “How to set or reset a loop element for an imported XML structure”](#).

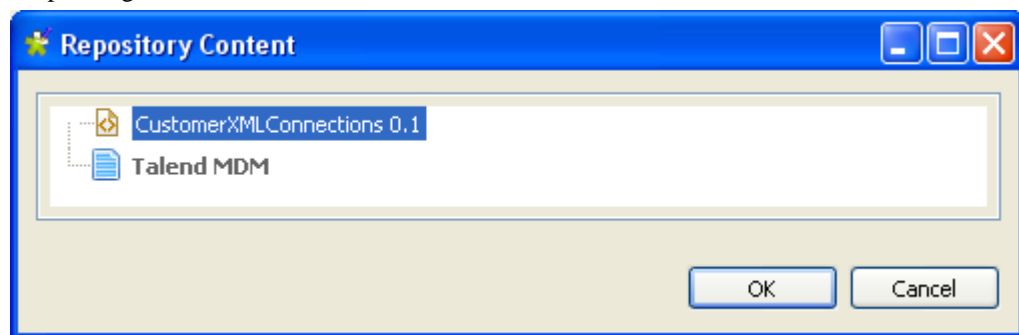
8.3.1.3. How to import the XML tree structure from the Repository

To do this, proceed as follows:

1. In any input flow table, right click the column name to open the contextual menu. In this example, it is *Customer*.



2. From this menu, select **Import From Repository**.
3. In the pop-up repository content list, select the XML connection or the MDM connection of interest to import the corresponding XML tree structure.



This figure presents an example of this **Repository**-stored XML connection.



To import an XML tree structure from the **Repository**, the corresponding XML connection should have been created. For further information about how to create a file XML connection in the Repository, see [Section 9.8, “Setting up an XML file schema”](#).

4. Click **OK** to validate this selection.

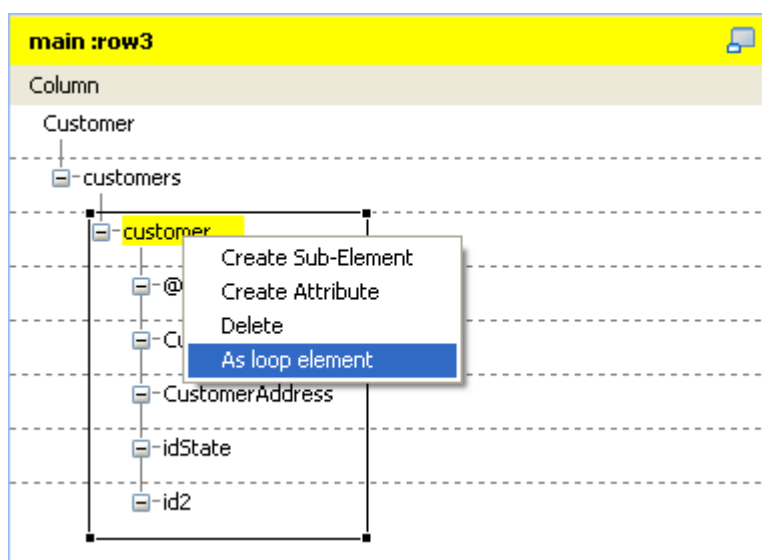
The XML tree structure is created and a loop is defined automatically as this loop was already defined during the creation of the current **Repository**-stored XML connection.

8.3.1.4. How to set or reset a loop element for an imported XML structure

You need to set at least one loop element for each XML tree if it does not have any. If it does, you may have to reset the existing loop element when needs be.

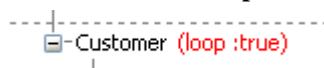
Whatever you need to set or reset a loop element, proceed as follows:

1. In the created XML tree structure, right-click the element you need to define as loop. For example, you need to define the *Customer* element as loop in the following figure.



- From the pop-up contextual menu, select **As loop element** to define the selected element as loop.

Once done, this selected element is marked with the text: **loop:true**.



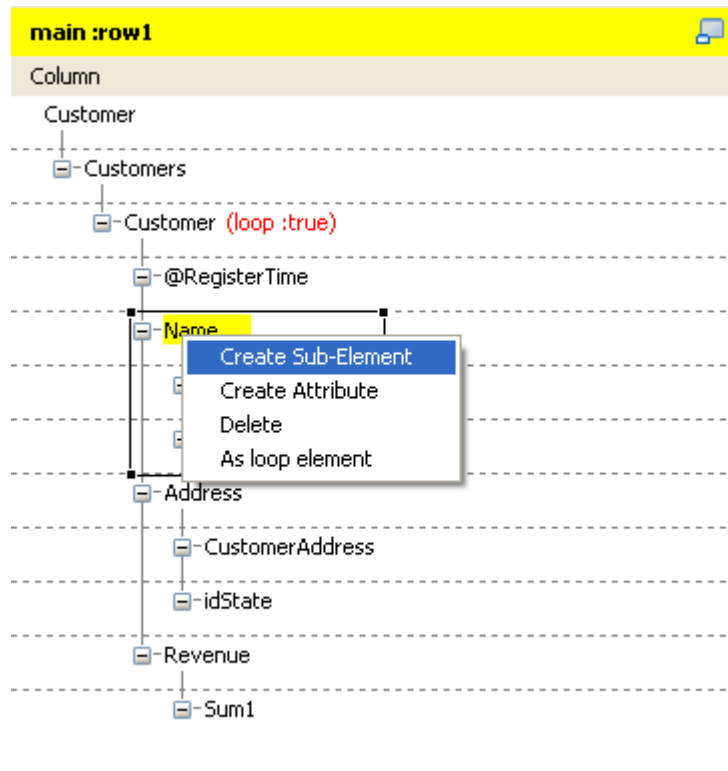
If you close the **Map Editor** without having set the required loop element for a given XML tree, its root element will be set automatically as loop element.

8.3.1.5. How to add a sub-element or an attribute to an XML tree structure

In the XML tree structure view, you are able to manually add a sub-element or an attribute to the root or to any of the existing elements when needs be.

To do either of these operations, proceed as follows:

- In the XML tree you need to edit, right-click the element to which you need to add a sub-element or an attribute underneath and select **Create Sub-Element** or **Create Attribute** according to your purpose.



2. In the pop-up [Create New Element] wizard, type in the name you need to use for the added sub-element or attribute.

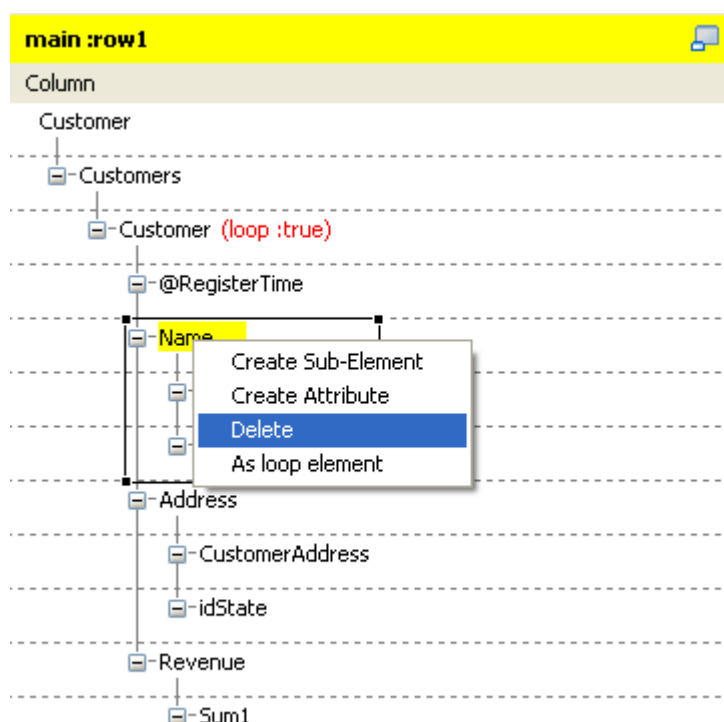


3. Click **OK** to validate this creation. The new sub-element or attribute displays in the XML tree structure you are editing.

8.3.1.6. How to delete an element or an attribute from the XML tree structure

From an established XML tree, you may need to delete an element or an attribute. To do this, proceed as follows:

1. In the XML tree you need to edit, right-click the element or the attribute you need to delete.



2. In the pop-up contextual menu, select **Delete**.

Then the selected element or attribute is deleted, including all of the sub-elements or the attributes attached to it underneath.

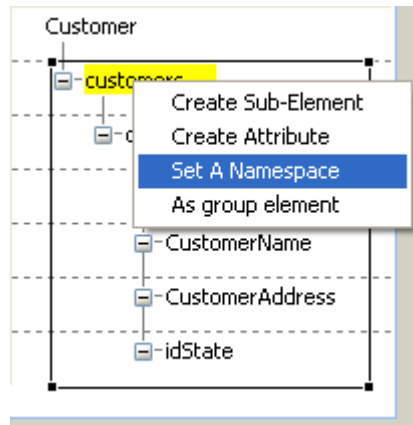
8.3.1.7. How to manage a namespace

When necessary, you are able to set and edit namespace for each of the element in the a created XML tree of the input or the output data flow.

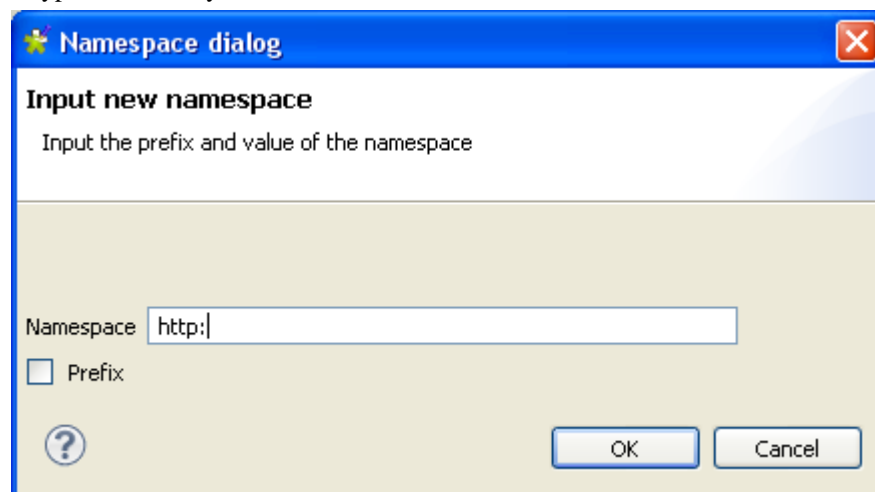
Defining a namespace

To do this, proceed as follows:

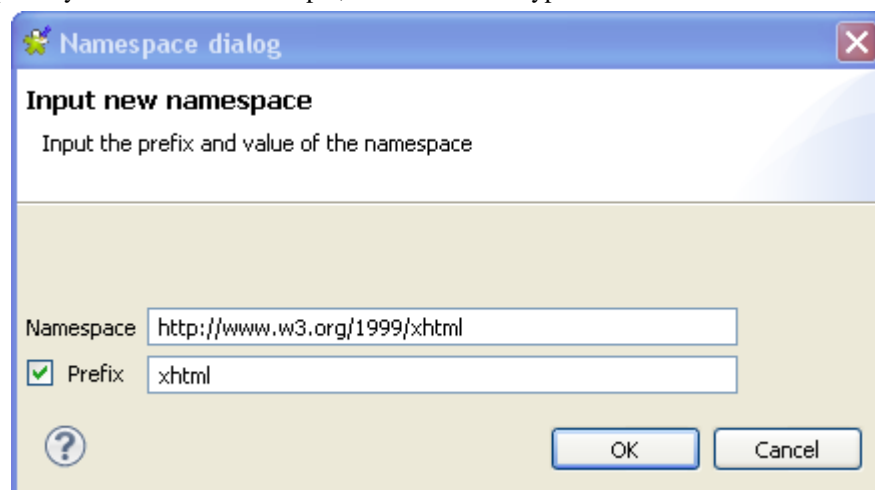
1. In the XML tree of the input or the output data flow you need to edit, right click the element for which you need to declare a namespace. For example, in a *Customer* XML tree of the output flow, you need to set a namespace for the root.



2. In the pop-up contextual menu, select **Set a namespace**. Then the [Namespace dialog] wizard displays.
3. In this wizard, type in the URI you need to use.



4. If you need to set a prefix for this namespace you are editing, select the **Prefix** check box in this wizard and type in the prefix you need. In this example, we select it and type in *xhtml*.



5. Click **OK** to validate this declaration.

Modifying the default value of a namespace

To do this, proceed as follows:

1. In the XML tree that the namespace you need to edit belongs to, right-click this namespace to open the contextual menu.



2. In this menu, select **Set A Fixed Prefix** to open the corresponding wizard.
3. Type in the new default value you need in this wizard.
4. Click **OK** to validate this modification.

Deleting a namespace

To do this, proceed as follows:

1. In the XML tree that the namespace you need to edit belongs to, right-click this namespace to open the contextual menu.



2. In this menu, click **Delete** to validate this deletion.

8.3.1.8. How to group the output data

The **tXMLMap** component uses a group element to group the output data according to a given grouping condition. This allows you to wrap elements matching the same condition with this group element.

To set a group element, two restrictions must be respected:

1. the root node cannot be set as group element;
2. the group element must be the parent of the loop element.



The option of setting group element is not visible until you have set the loop element; this option is also invisible if an element is not allowed to be set as group element.

Once the group element is set, all of its sub-elements except the loop one are used as conditions to group the output data.

You have to carefully design the XML tree view for the optimized usage of a given group element. For further information about how to use a group element, see **tXMLMap** in the *Talend Open Studio Components Reference Guide*.



tXMLMap provides group element and aggregate element to classify data in the XML tree structure. When handling a row of XML data flow, the behavioral difference between them is:

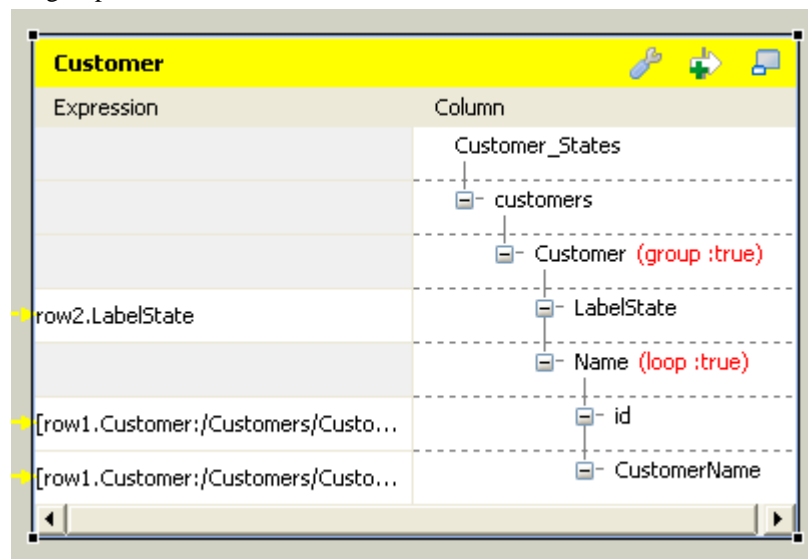
- The group element processes the data always within one single flow.
- The aggregate element splits this flow into separate and complete XML flows.

Setting a group element

To set a group element, proceed as follows:

1. In the XML tree view on the output side of the **Map editor**, right-click the element you need to set as group element.
2. From the opened contextual menu, select **As group element**.

Then this element of selection becomes the group element. The following figure presents an example of an XML tree with the group element.



Revoking a defined group element

To revoke a defined group element, proceed as follows:

1. In the XML tree view on the output side of the **Map editor**, right-click the element you have defined as **group element**.
2. From the opened contextual menu, select **Remove group element**.

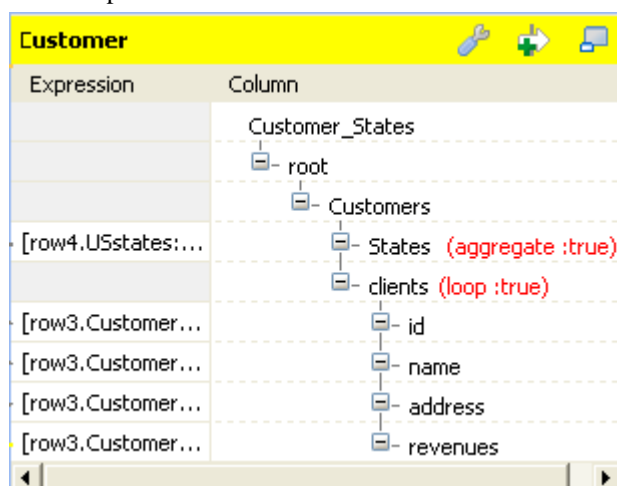
Then the defined group element is revoked.

8.3.1.9. How to aggregate the output data

With **tXMLMap**, you can define as many aggregate elements as required in the output XML tree to class the XML data accordingly. Then this component outputs these classes, each as one complete XML flow.

1. To define an element as aggregate element, simply right-click this element of interest in the XML tree view on the output side of the **Map editor** and from the contextual menu, select **As aggregate element**.

Then this element becomes the aggregate element. Texts in red are added to it, reading **aggregate : true**. The following figure presents an example.



- To revoke the definition of the aggregate element, simply right-click the defined aggregate element and from the contextual menu, select **Remove aggregate element**.



To define an element as aggregate element, ensure that this element has no child element and the **All in one** feature is being disabled. The **As aggregate element** option is not available in the contextual menu until both of the conditions are respected. For further information about the **All in one** feature, see [Section 8.3.2.1, “How to output elements into one document”](#).

For an example about how to use the aggregate element with **tXMLMap**, see *Talend Open Studio Components Reference Guide*



tXMLMap provides **group element** and **aggregate element** to classify data in the XML tree structure. When handling one row of data (one complete XML flow), the behavioral difference between them is:

- The **group element** processes the data always within one single flow.
- The **aggregate element** splits this flow into separate and complete XML flows.

8.3.2. Defining the output mode

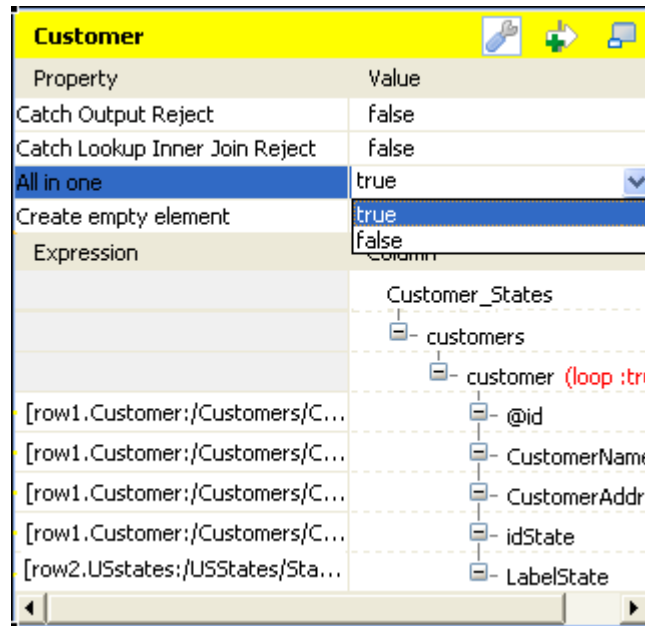
To define the output mode of the document-type data, you are defining whether to put all of the XML elements into one single XML flow and when empty element exist, whether to output them. By doing this, you do not change the structure of the XML tree you have created.

8.3.2.1. How to output elements into one document

Unless you are using the aggregate element which always classifies the output elements and splits an output XML flow, you are able to determine whether an XML flow is output as one single flow or as separate flows, using the **All in one** feature in the **tXMLMap** editor.

To do this, on the output side of the **Map editor**, proceed as follows:

- Click the pincer icon to open the map setting panel. The following figure presents an example.



2. Click the **All in one** field and from the drop-down list, select **true** or **false** to decide whether the output XML flow should be one single flow.
- If you select **true**, the XML data is output all in one single flow. In this example, the single flow reads as follows:

```
Starting job tXMLMap at 10:16 09/11/2011.

[statistics] connecting to socket on port 3643
[statistics] connected
<?xml version="1.0" encoding="UTF-8"?>
<customers><customer id="1"><CustomerName>Griffith Paving
and
Sealcoat</CustomerName><CustomerAddress>talend@apres91</C
ustomerAddress><idState>7</idState><LabelState>Connecticut<
/LabelState></customer><customer
id="56"><CustomerName>Glenn Oaks Office
Supplies</CustomerName><CustomerAddress>1859 Green Bay
Rd.</CustomerAddress><idState>7</idState><LabelState>Conne
cticut</LabelState></customer><customer
id="2"><CustomerName>Bill's Dive
Shop</CustomerName><CustomerAddress>511 Maple Ave. Apt.
1B</CustomerAddress><idState>35</idState><LabelState>Ohio</
LabelState></customer><customer id="61"><CustomerName>DBN
Bank</CustomerName><CustomerAddress>456 Grossman
Ln.</CustomerAddress><idState>35</idState><LabelState>Ohio<
/LabelState></customer><customer
id="63"><CustomerName>Pivot Point
College</CustomerName><CustomerAddress>1547 Knolwood
Rd.</CustomerAddress><idState>9</idState><LabelState>Flori
da</LabelState></customer></customers>
[statistics] disconnected
Job tXMLMap ended at 10:16 09/11/2011. [exit code=0]
```

The structure of this flow reads:

```
<?xml version="1.0" encoding="UTF-8"?>
<customers>
  <customer id="1">
    <CustomerName>Griffith Paving and Sealcoatin</CustomerName>
    <CustomerAddress>talend@apres91</CustomerAddress>
    <idState>7</idState>
    <LabelState>Connecticut</LabelState>
  </customer>
  <customer id="56">
    <CustomerName>Glenn Oaks Office Supplies</CustomerName>
    <CustomerAddress>1859 Green Bay Rd.</CustomerAddress>
    <idState>7</idState>
    <LabelState>Connecticut</LabelState>
  </customer>
  <customer id="2">
    <CustomerName>Bill's Dive Shop</CustomerName>
    <CustomerAddress>511 Maple Ave. Apt. 1B</CustomerAddress>
    <idState>35</idState>
    <LabelState>Ohio</LabelState>
  </customer>
  <customer id="61">
    <CustomerName>DBN Bank</CustomerName>
    <CustomerAddress>456 Grossman Ln.</CustomerAddress>
    <idState>35</idState>
    <LabelState>Ohio</LabelState>
  </customer>
  <customer id="63">
    <CustomerName>Pivot Point College</CustomerName>
    <CustomerAddress>1547 Knolwood Rd.</CustomerAddress>
    <idState>9</idState>
    <LabelState>Florida</LabelState>
  </customer>
</customers>
```

- If you select **false**, the XML data is output in separate flows, each loop being one flow, neither grouped nor aggregated. In this example, these flows read as follows:


```

Starting job tXMLMap at 10:25 09/11/2011.

[statistics] connecting to socket on port 4036
[statistics] connected
<?xml version="1.0" encoding="UTF-8"?>
<customers><customer id="1"><CustomerName>Griffith Paving
and
Sealcoatin</CustomerName><CustomerAddress>talend@apres91</C
ustomerAddress><idState>7</idState><LabelState>Connecticut<
/LabelState></customer></customers>
<?xml version="1.0" encoding="UTF-8"?>
<customers><customer id="56"><CustomerName>Glenn Oaks
Office Supplies</CustomerName><CustomerAddress>1859 Green
Bay
Rd.</CustomerAddress><idState>7</idState><LabelState>Conne
cticut</LabelState></customer></customers>
<?xml version="1.0" encoding="UTF-8"?>
<customers><customer id="2"><CustomerName>Bill's Dive
Shop</CustomerName><CustomerAddress>511 Maple Ave. Apt.
1B</CustomerAddress><idState>35</idState><LabelState>Ohio</
LabelState></customer></customers>
<?xml version="1.0" encoding="UTF-8"?>
<customers><customer id="61"><CustomerName>DBN
Bank</CustomerName><CustomerAddress>456 Grossman
Ln.</CustomerAddress><idState>35</idState><LabelState>Ohio<
/LabelState></customer></customers>
<?xml version="1.0" encoding="UTF-8"?>
<customers><customer id="63"><CustomerName>Pivot Point
College</CustomerName><CustomerAddress>1547 Knolwood
Rd.</CustomerAddress><idState>9</idState><LabelState>Flori
da</LabelState></customer></customers>

```

Each flow contains one complete XML structure. To take the first flow as example, its structure reads:

```

<?xml version="1.0" encoding="UTF-8"?>
<customers>
  <customer id="1">
    <CustomerName>Griffith Paving and Sealcoatin</CustomerName>
    <CustomerAddress>talend@apres91</CustomerAddress>
    <idState>7</idState>
    <LabelState>Connecticut</LabelState>
  </customer>
</customers>

```



The **All in one** feature is disabled if you are using the aggregate element. For further information about the aggregate element, see [Section 8.3.1.9, "How to aggregate the output data"](#)

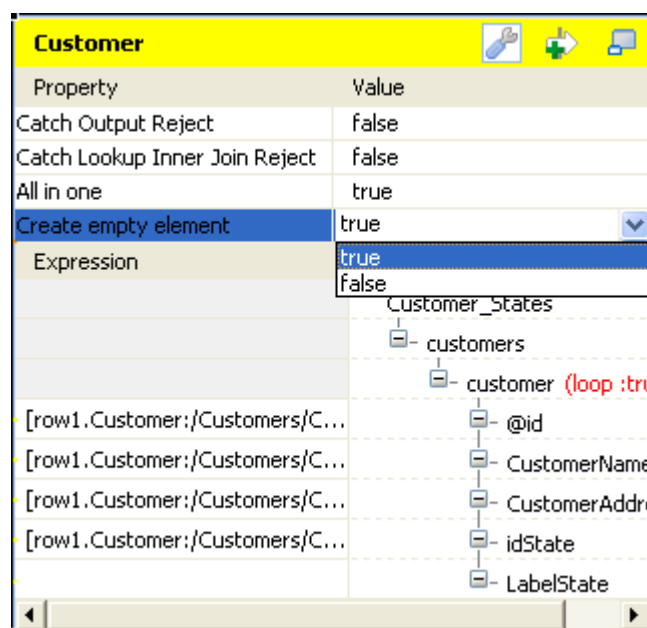
8.3.2.2. How to manage empty element in Map editor

It may be necessary to create and output empty elements during the process of transforming data into XML flow, such as, when **tXMLMap** works along with **tWriteXMLField** that creates empty elements or when there is no input column associated with certain XML node in the output XML data flow.

By contrast, in some scenarios, you do not need to output the empty element while you have to keep them in the output XML tree for some reasons.

tXMLMap allows you to set the boolean for the creation of empty element. To do this, on the output side of the **Map editor**, perform the following operations:

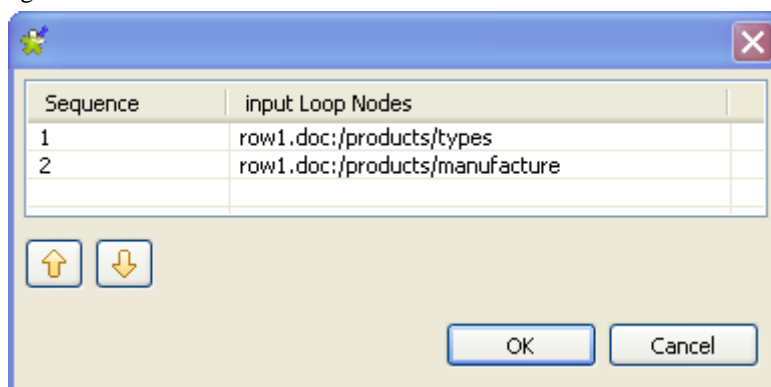
1. Click the pincer icon to open the map setting panel.



2. In the panel, click the **Create empty element** field and from the drop-down list, select **true** or **false** to decide whether to output the empty element.
 - If you select **true**, the empty element is created in the output XML flow and output, for example, `<customer><LabelState/></customer>`.
 - If you select **false**, the empty element is not output.

8.3.2.3. How to define the sequence of multiple input loops

If a loop element, or the flat data flow, receives mappings from more than one loop element of the input flow, you need to define the sequence of the input loops. The first loop element of this sequence will be the primary loop, so the transformation process related to this sequence will first loop over this element such that the data outputted will be sorted with regard to its element values.



For example, in this figure, the *types* element is the primary loop and the outputted data will be sorted by the values of this element.

```

<types>
  <type>DELL123</type>
  <manufacture_id>manu_1</manufacture_id>
</types>
<types>
  <type>DELL123</type>
  <manufacture_id>manu_2</manufacture_id>
</types>
<types>
  <type>DELL456</type>
  <manufacture_id>manu_1</manufacture_id>
</types>
<types>
  <type>DELL456</type>
  <manufacture_id>manu_2</manufacture_id>
</types>
<types>
  <type>HP123</type>
  <manufacture_id>manu_1</manufacture_id>
</types>
<types>
  <type>HP123</type>
  <manufacture_id>manu_2</manufacture_id>
</types>
<types>
  <type>HP456</type>
  <manufacture_id>manu_1</manufacture_id>
</types>
<types>
  <type>HP456</type>
  <manufacture_id>manu_2</manufacture_id>
</types>
</manufactures>

```

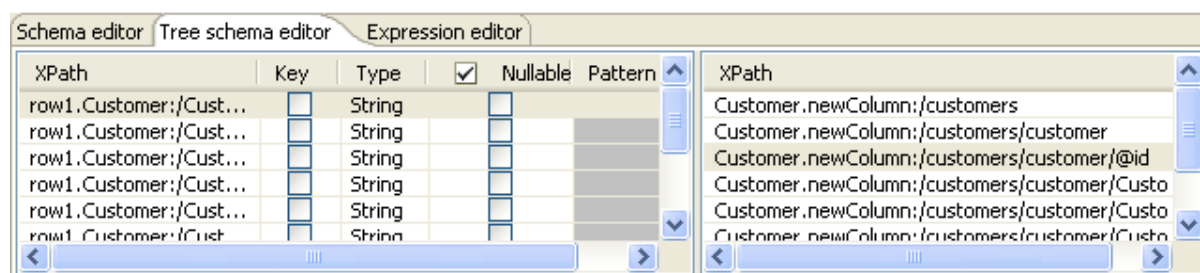
In this case in which one output loop element receives several input loop elements, a [...] button appears next to this receiving loop element or for the flat data, appears on the head of the table representing the flat data flow. To define the loop sequence, do the following:

1. Click this [...] button to open the sequence arrangement window as presented by the figure used earlier in this section.
2. Use the up or down flash button to arrange this sequence.

8.3.3. Editing the XML tree schema


In addition to the **Schema editor** and the **Expression editor** views that **tMap** is also equipped with, a **Tree schema editor** view is provided in the map editor of **tXMLMap** for you to edit the XML tree schema of an input or output data flow.

To access this schema editor, click the **Tree schema editor** tab on the lower part of the map editor.



The left half of this view is used to edit the tree schema of the input flow and the right half to edit the tree schema of the output flow.

The following table presents further information about this schema editor.

Metadata	Description
XPath	Use it to display the absolute paths pointing to each element or attribute in a XML tree and edit the name of the corresponding element or attribute.
Key	Select the corresponding check box if the expression key data should be used to retrieve data through the Join link. If unchecked, the Join relation is disabled.
Type	Type of data: String, Integer, Document, etc.  This column should always be defined in a Java version.
Nullable	Select this check box if the field value could be null.
Pattern	Define the pattern for the Date data type.



Input metadata and output metadata are independent from each other. You can, for instance, change the label of a column on the output side without the column label of the input schema being changed.

However, any change made to the metadata are immediately reflected in the corresponding schema on the **tXMLMap** relevant (Input or Output) area, but also on the schema defined for the component itself on the design workspace.

For detailed use cases about the multiple operations that you can perform using **tXMLMap**, see *Talend Open Studio Components Reference Guide*.



Chapter 9. Managing Metadata

Metadata in *Talend Open Studio for ESB* is definitional data that provides information about or documentation of other data managed within *Talend Open Studio for ESB*.

This chapter provides procedures to create and manage various metadata items that can be used in all your job designs.

Before starting any metadata management processes, you need to be familiar with *Talend Open Studio for ESB* Graphical User Interface (GUI). For more information, see [Appendix A, GUI](#).

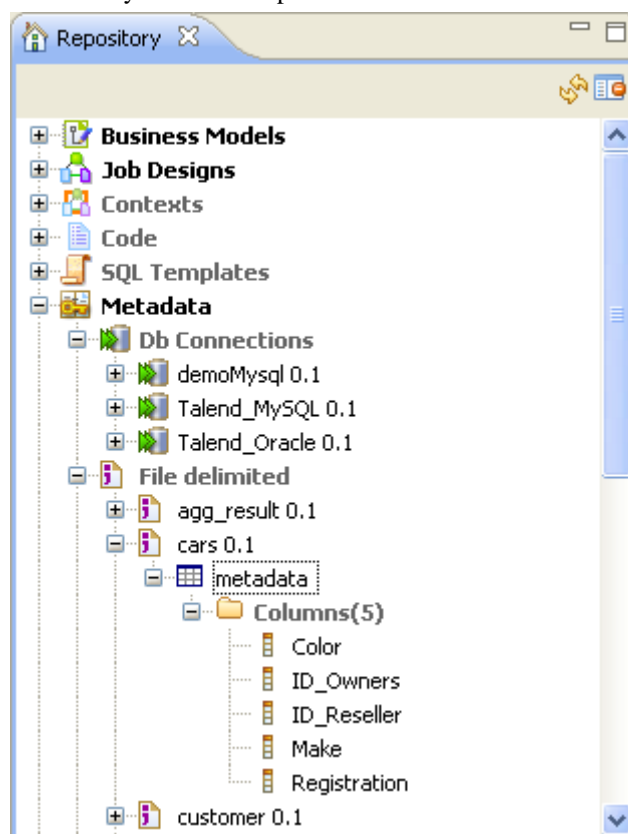
9.1. Objectives

The **Metadata** folder in the **Repository** tree view stores reusable information on files, databases, and/or systems that you need to build your Jobs.

Various corresponding wizards help you store these pieces of information and use them later to set the connection parameters of the relevant input or output components, but you can also store the data description called “schemas” in *Talend Open Studio for ESB*.

Wizards’ procedures slightly differ depending on the type of connection chosen.

Click **Metadata** in the **Repository** tree view to expand the folder tree. Each of the connection nodes will gather the various connections and schemas you have set up.



From *Talend Open Studio for ESB*, you can set up the following, amongst others:


- a DB connection,
- a JDBC schema,
- a SAS connection,
- a file schema,
- an LDAP schema,
- a salesforce schema,
- a generic schema,
- a MDM connection,
- a WSDL schema,
- a FTP connection,

The following sections explain in detail how to set up different connections and schemas.

9.2. Setting up a DB connection

If you often need to connect to database tables of any kind, then you may want to centralize the connection information details in the **Metadata** folder in the **Repository** tree view.



You can also set up a DB connection the same way by clicking the  icon in the **Basic settings** view of all input and output DB components.



A specific *.jar* file other than the file provided with your Studio may be required when you connect to an Oracle database of, for example, version 10. If so, you need to download this required *.jar* file from Oracle's website, copy-paste it into the directory: *your_studio_folder\lib\java*, and restart the Studio before creating the DB connection of interest.

9.2.1. Step 1: General properties

The creation procedure is made of two separate but closely related operations.

1. First expand **Metadata** in the **Repository** tree view and right-click **Db Connections** and select **Create connection** from the pop-up menu.
2. In the connection wizard that opens up, fill in the generic Schema properties such as Schema **Name** and **Description**. The **Status** field is a customized field you can define in **Window > Preferences**.

Database Connection

New Database Connection on repository - Step 1/2

Define the properties

Name: CustomersDBschemas

Purpose:

Description:

Author: user@doc.com

Locker:

Version: 0.1 [M] [m]

Status:

Path: [Select]

< Back Next > Finish Cancel

3. Click **Next** when completed. The second step requires you to fill in DB connection data.

9.2.2. Step 2: Connection

1. Select the type of the database to which you want to connect and some fields will be available/unavailable according to the DB connection detail requirements.

Database Connection

New Database Connection on repository - Step 2/2

You must press the Check Button to check the Database Setting

Database Settings

DB Type: MySQL

Db Version: MySQL 5

String of Connection: jdbc:mysql://talend-dbms:3306/talend?noDatetimeStringSync=true

Login: root

Password: ••••

Server: talend-dbms

Port: 3306

DataBase: talend

Schema:

Additional parameters: noDatetimeStringSync=true

Check

Database Properties

SQL Syntax: SQL 92

String Quote: "

Null Char: 000

Export as Context

Revert from Context

< Back Next > Finish Cancel



When you are creating the database connection of some databases like AS400, HSQDB, Informix, MsSQL, MySQL, Oracle, Sybase, or Teradata, you can specify additional connection properties through the **Additional parameters** field in the **Database Settings** area.

2. Fill in the connection details and click **Check** to check your connection.



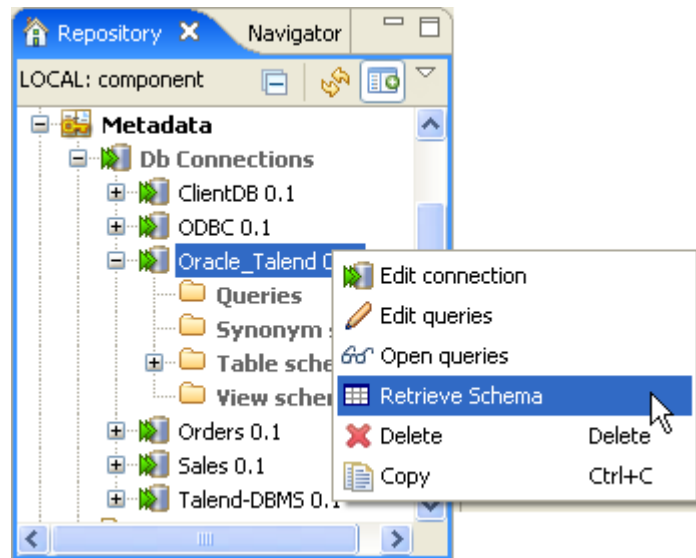
In order to be able to retrieve all table schemas in the database:

-enter *dbo* in the **Schema** field if you are connecting to MSSQL 2000,

-remove *dbo* from the **Schema** field if you are connecting to MSSQL 2005/2008.

3. Fill in, if need be, the database properties information. That is all for the first operation on DB connection setup, click **Finish** to validate.

The newly created DB connection is now available in the **Repository** tree view and displays four folders including **Queries** (for SQL queries you save) and **Table schemas** that will gather all schemas linked to this DB connection.



4. Right-click the newly created connection, and select **Retrieve schema** on the drop-down list in order to load the desired table schema from the established connection.



An error message will display if there are no tables to retrieve from the selected database or if you do not have the correct rights to access this database.

9.2.3. Step 3: Table upload


When you click **Retrieve Schema**, a new wizard opens up where you can filter and display different objects (tables, views and synonyms) in your database connection.



For the time being, synonyms option works for Oracle, IBM DB2 and MSSQL only.

In the **Select Filter Conditions** area, you can filter the database objects using either of the two options: **Set the Name Filter** or **Use the Sql Filter** through filtering on objects names or using SQL queries respectively.

To filter database objects using their names, do the following:

1. In the **Select Filter Conditions** area, select the **Use the Name Filter** option.
2. In the **Select Types** area, select the check box(es) of the database object(s) you want to filter or display.
 -  Available options can vary according to the selected database.
3. In the **Set the Name Filter** area, click **Edit...** to open the **[Edit Filter Name]** dialog box.
4. Enter the filter you want to use in the dialog box. For example, if you want to recuperate the database objects which names start with “A”, enter the filter “A%”, or if you want to recuperate all database objects which names end with “type”, enter “%type” as your filter.
5. Click **OK** to close the dialog box.
6. Click **Next** to open a new view on the wizard that lists the filtered database objects.

To filter database objects using an SQL query, do the following:

1. In the **Select Filter Conditions** area, select the **Use Sql Filter** option.
2. In the **Set the Sql Filter** field, enter the SQL query you want to use to filter database objects.
3. Click **Next** to open a new view that lists the filtered database objects.

Once you have the filtered list of the database objects (tables, views and synonyms), do the following to load the schemas of the desired objects onto your repository file system:

Select Schema to create

Name Filter:

Name	Type	Column number	Creation status
<input type="checkbox"/> executionvirtualserver_executionserver	TABLE		
<input type="checkbox"/> extract	TABLE		
<input checked="" type="checkbox"/> familiestype	TABLE	6	Success
<input checked="" type="checkbox"/> family	TABLE	3	Success
<input type="checkbox"/> feature516	TABLE		
<input type="checkbox"/> filetriggermask	TABLE		
<input checked="" type="checkbox"/> filter	TABLE	4	Success
<input type="checkbox"/> flowmeter210	TABLE		
<input type="checkbox"/> flowmeterlog	TABLE		
<input type="checkbox"/> formattype	TABLE		
<input type="checkbox"/> headertype	TABLE		
<input type="checkbox"/> helene	TABLE		
<input type="checkbox"/> implicitcontextsettings	TABLE		
<input type="checkbox"/> importstype	TABLE		
<input type="checkbox"/> importtype	TABLE		
<input type="checkbox"/> infobright_result	TABLE		
<input type="checkbox"/> information	TABLE		
<input type="checkbox"/> input	TABLE		
<input type="checkbox"/> installtype	TABLE		
<input type="checkbox"/> item	TABLE		

1. Select one or more database objects on the list and click **Next** to open a new view on the wizard where you can see the schemas of the selected object.



If no schema is visible on the list, click the **Check connection** button below the list to verify the database connection status.

Schema

familiestype
family
filter

Name: familiestype

Comment:

Type:

Based on table: familiestype

Retrieve Schema Guess Schema

Schema

Column	Db Column	Key	DB Type	Type	<input checked="" type="checkbox"/>	N..	Date P...
e_id	e_id	<input checked="" type="checkbox"/>	BIGINT	long	<input type="checkbox"/>		
dtype	dtype	<input type="checkbox"/>	VARCHAR	String	<input type="checkbox"/>		
e_version	e_version	<input type="checkbox"/>	INT	int	<input type="checkbox"/>		
econtainer...	econtainer_class	<input type="checkbox"/>	VARCHAR	String	<input checked="" type="checkbox"/>		
e_container	e_container	<input type="checkbox"/>	VARCHAR	String	<input checked="" type="checkbox"/>		
e_contain...	e_container_f...	<input type="checkbox"/>	INT	Integer	<input checked="" type="checkbox"/>		

Add Schema

< Back Next > Finish Cancel

2. Modify the schemas if needed and then click **Finish** to close the wizard.

The schemas based on the selected tables are listed under the **Table schemas** folder corresponding to the database connection you created.



In Java, make sure the data type in the Type column is correctly defined.

For more information regarding data types, including date pattern, check out <http://docs.oracle.com/javase/6/docs/api/index.html>.

9.2.4. Step 4: Schema definition

By default, the schema displayed on the Schema panel is based on the first table selected in the list of schemas loaded (left panel). You can change the name of the schema and according to your needs, you can also customize the schema structure in the schema panel.

Indeed, the tool bar allows you to add, remove or move columns in your schema. In addition, you can load an xml schema from a file or export the current schema as xml.

To retrieve a schema based on one of the loaded table schemas, select the DB table schema name in the drop-down list and click **Retrieve schema**. Note that the retrieved schema then overwrites any current schema and does not retain any custom edits.

Click **Finish** to complete the DB schema creation. All the retrieved schemas are displayed in the **Table schemas** sub-folder under the relevant DB connection node.

9.3. Setting up a JDBC schema

For DB table based schemas, the creation procedure is made of two separate but closely related operations. First right-click **Db Connections** and select **Create connection** on the pop-up menu.

9.3.1. Step 1: General properties

Fill in the schema generic information, such as Schema **Name** and **Description**.

For further information, see [Section 9.2.1, “Step 1: General properties”](#).

9.3.2. Step 2: Connection

1. Select the type of Database you want to connect to and fill in the DB connection details.

Database Connection

New Database Connection on repository - Step 2/2

You must press the Check Button to check the Database Setting

Database Settings

DB Type: General JDBC

JDBC URL: jdbc:mysql://talend-dbms:3306/talend

Driver jar: C:\MySQL\MySQL Tools for 5.0\java\lib\mysql-connector-java-5.1...

Class name: com.mysql.jdbc.Driver

User name: root

Password: *****

Mapping file: ...

Check

Database Properties

SQL Syntax: SQL 92

String Quote: "

Null Char: 000

☒ Standard SQL Statement ☐ System SQL Statement

< Back **Next >** **Finish** **Cancel**

2. Fill in the connection details below:

- Fill in the **JDBC URL** used to access the DB server.
- In the **Driver jar** field, select the jar driver validating your connection to the database.
- In the **Class name** field, fill in the main class of the driver allowing to communicate with the database.
- Fill in your **User name** and **Password**.
- In the **Mapping File** field, select the mapping allowing the DB Type to match the Java type of data on the schema. For example: the DB Type VARCHAR corresponds to the String Type for Java.

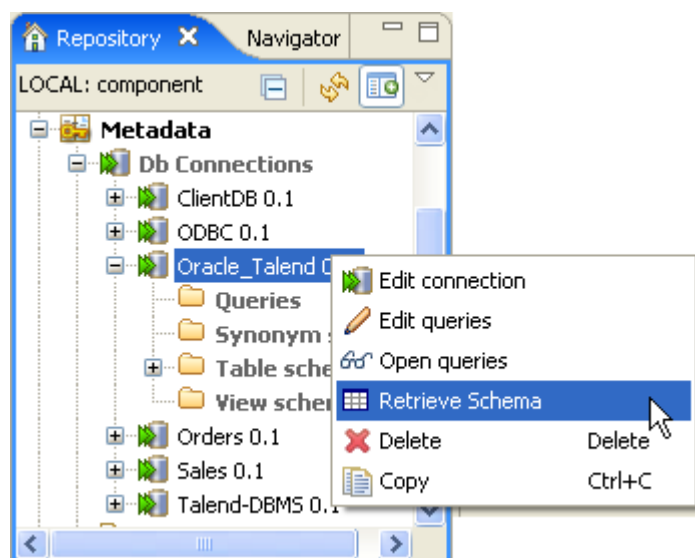


The Mapping files are XML files that you can access via **Window > Preferences > Talend > Metadata of TalendType**.

3. Click **Check** to check out your connection.

4. Fill in, if need be, the database properties information. That is all for the first operation on DB connection setup, click **Finish** to validate.

The newly created DB connection is now available in the **Repository** tree view and it displays four folders including **Queries** (for the SQL queries you save) and **Table schemas** that will gather all schemas linked to this DB connection.



5. Right-click the newly created connection, and select **Retrieve schema** on the drop-down list.

9.3.3. Step 3: Table upload

A new wizard opens up on the first step window. The List offers all tables present on the DB connection. It can be any type of databases.

Select one or more tables on the list, to load them onto your repository file system. You will base your repository schemas on these tables.

If no schema is visible on the list, click **Check connection**, to verify the DB connection status.

Click **Next**. On the next window, four setting panels help you define the schemas to create.

In Java, make sure the data type is correctly defined. For more information regarding data types, including date pattern, check out <http://docs.oracle.com/javase/6/docs/api/index.html>.

9.3.4. Step 4: Schema definition

By default, the schema displayed on the Schema panel is based on the first table selected in the list of schemas loaded (left panel). You can change the name of the schema and according to your needs, you can also customize the schema structure in the schema panel.

Indeed, the tool bar allows you to add, remove or move columns in your schema. In addition, you can load an xml schema from a file or export the current schema as xml.

To retrieve a schema based on one of the loaded table schemas, select the DB table schema name in the drop-down list and click **Retrieve schema**. Note that the retrieved schema then overwrites any current schema and does not retain any custom edits.

Click **Finish** to complete the DB schema creation. All the retrieved schemas are displayed in the **Table schemas** sub-folder under the relevant DB connection node.

9.4. Setting up a SAS connection

Talend Open Studio for ESB enables you to configure a connection to a remote SAS system.

9.4.1. Prerequisites

Before carrying on the below procedure to configure your SAS connection, make sure that you retrieve your metadata from the SAS server and export it in XML format.

9.4.2. Step 1: General properties

1. In the **Repository** tree view of *Talend Open Studio for ESB*, expand **Metadata** and then right-click **DB Connections**.
2. Select Create connection from the contextual menu to open the **[Database Connection]** wizard.
3. Fill in schema generic information, such as **Name** and **Description** and click **Next** to open a new view on the wizard.

For further information, see [Section 9.2.1, “Step 1: General properties”](#).

9.4.3. Step 2: Connection

1. In the **DB type** field in the **[Database Connection]** wizard, select **SAS** and fill in the fields that follow with SAS connection information.

Database Connection

New Database Connection on repository - Step 2/2

i You must press the Check Button to check the Database Setting

Database Settings

DB Type: SAS

String of Connection: jdbc:sasiom://localhost:7070/test

Login: root

Password: ••••

Server: localhost

Port: 7070

Database: test

Schema:

Additional parameters: noDatetimeStringSync=true

Check

Database Properties

SQL Syntax: SQL 92

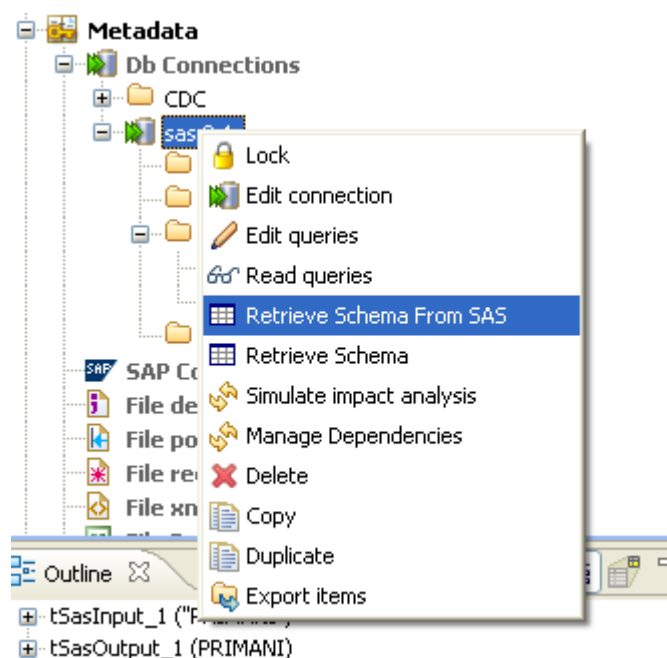
String Quote: " Null Char: 000

Export as Context Revert from Context

< Back Next > Finish Cancel

2. If needed, click the **Check** tab to verify if your connection is successful.
3. If needed, define the properties of the database in the corresponding fields in the **Database Properties** area.
4. Click **Finish** to validate your changes and close the wizard.

The newly set connection to the defined database displays under the **DB Connections** folder in the **Repository** tree view. This connection has four sub-folders among which **Table schemas** can group all schemas relative to this connection.



5. Right-click the SAS connection you created and then select **Retrieve Schema from SAS** to display all schemas in the defined database under the **Table schemas** sub-folder.

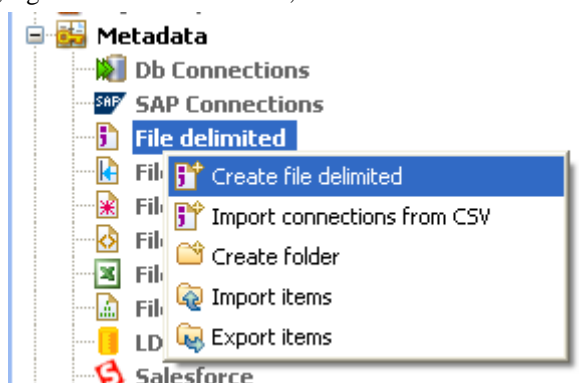
9.5. Setting up a File Delimited schema

File Delimited metadata can be used to define the properties of **tFileInputDelimited** and **tFileInputExcel** components.



The file schema creation is very similar for all types of file connections: Delimited, Positional, Regex, Xml, or Ldif.

In the **Repository** tree view, right-click **File Delimited**, and select **Create file delimited** on the pop-up menu.



Unlike the DB connection wizard, the **[New Delimited File]** wizard gathers both file connection and schema definitions in a four-step procedure.

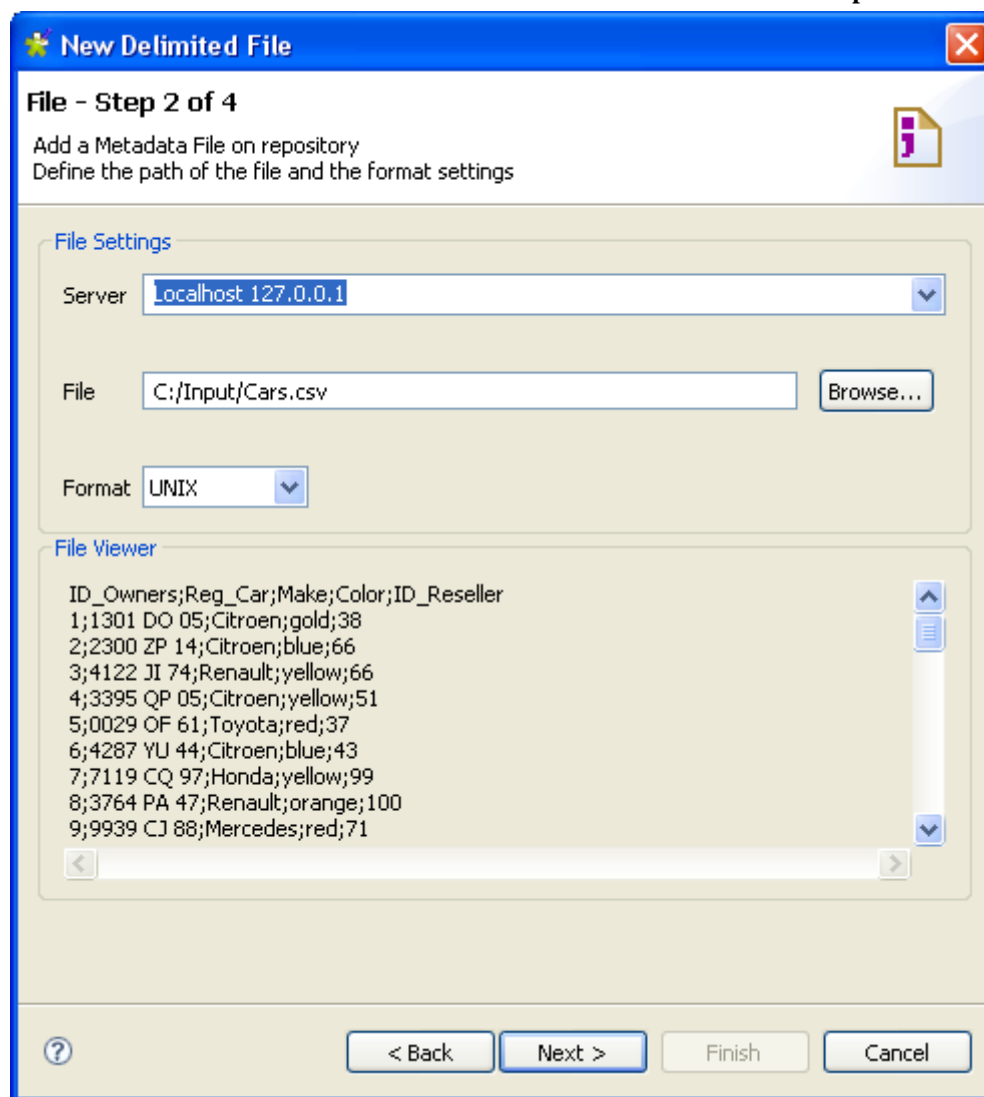
9.5.1. Step 1: General properties

In the first step, fill in the schema generic information, such as Schema **Name** and **Description**.

For further information, see [Section 9.2.1, “Step 1: General properties”](#).

9.5.2. Step 2: File upload

Define the **Server** IP address where the file is stored. And click **Browse...** to set the **File path**.



Select the OS **Format** the file was created in. This information is used to prefill subsequent step fields. If the list doesn't include the appropriate format, ignore it.

The **File viewer** gives an instant picture of the file loaded. It allows you to check the file consistency, the presence of header and more generally the file structure.

Click **Next** to proceed to Step3.

9.5.3. Step 3: Schema definition

On this view, you can refine your data description and file settings. Click the squares below, to zoom in and get more information.

New Delimited File

File - Step 3 of 4

Add a Metadata File on repository
Define the setting of the parse job

File Settings

Encoding: US-ASCII

Field length: Semicolon Corresponding Character: ;

Row Separator: Standard EOL Corresponding Character: \n

Rows To Skip

If any rows must be ignored, specify the following parameters

Header: ☒ 6

Footer: ☐

☐ Skip empty row

Escape Char Settings

☐ CSV ☒ Delimited

Escape Char: Empty

Text Enclosure: Empty

Limit Of Rows

If the number of lines must be limited, specify this number

Limit: ☐

Preview

☒ Set heading row as column names [Refresh Preview](#)

id	CustomerName	CustomerAddress	idReseller	Make	RegTime	Column 6	Column 7
1	Griffith Paving and Sealcoat	355 Golf Rd.	7	1	03/11/1991 09:20	1973-01-17 06:26:40.000	67852.
2	Bill's Dive Shop	511 Maple Ave. Apt. 1B	35	1	19/11/1984 15:48	2010-06-07 09:40:00.000	88792.
3	Childress Child Day Care	662 Lyons Circle	39	1	16/02/1981 08:27	1990-04-01 21:00:00.000	35340.
4	Facelift Kitchen and Bath	220 Vine Ave.	41	1	22/08/2010 09:55	1972-04-23 18:00:00.000	6097.8
5	Terrinni & Son Auto and Truck	770 Exmoor Rd.	5	1	28/06/1995 09:15	1982-04-19 10:26:40.000	5146.5
6	Kermitt the Pet Shop	1860 Parkside Ln.	28	1	17/08/2009 10:07	1991-05-27 17:00:00.000	16087.
7	Tub's Furniture Store	807 Old Trail Rd.	15	1	27/08/1976 03:13	1970-03-27 23:08:16.000	53216.
8	Toggle & Myerson Ltd	618 Sheriden rd.	9	1	24/03/2008 23:07	1981-08-02 01:26:40.000	74168.
9	Childress Child Day Care	788 Tennessee Ave	12	1	16/08/2001 06:22	1994-05-03 11:12:00.000	82172.

[? < Back](#) [Next >](#) [Finish](#) [Cancel](#)

Set the **Encoding**, as well as Field and Row separators in the Delimited File Settings.

File Settings

Encoding: US-ASCII

Field length: Semicolon Corresponding Character: ;

Row Separator: Standard EOL Corresponding Character: \n

Escape Char Settings

☒ CSV ☐ Delimited

Escape Char: Empty

Text Enclosure: Empty

Depending on your file type (csv or delimited), you can also set the Escape and Enclosure characters to be used.

If the file preview shows a header message, you can exclude the header from the parsing. Set the number of header rows to be skipped. Also, if you know that the file contains footer information, set the number of footer lines to be ignored.

Rows To Skip

If any rows must be ignored, specify the following parameters

Header ☒ 3

Footer ☐

☐ Skip empty row

Limit Of Rows

If the number of lines must be limited, specify this number

Limit ☐

The **Limit of rows** allows you to restrict the extend of the file being parsed.

In the **File Preview** panel, you can view the new settings impact.

Check the **Set heading row as column names** box to transform the first parsed row as labels for schema columns. Note that the number of header rows to be skipped is then incremented by 1.

Preview

☒ Set heading row as column names

ID	Registration	Make	Color	Reseller ID	Name	Insurance	
1	5776 ZQ 94	Volkswagen	gold	7	montmont	KVW2844	
3	2580 TT 77	Renault	orange	1	bouhnan	BNU9147	
4	1722 VE 11	Citroen	silver	10	hikla	TEU9320	

Click **Refresh** on the preview panel for the settings to take effect and view the result on the viewer.

9.5.4. Step 4: Final schema

The last step shows the Delimited File schema generated. You can customize the schema using the toolbar underneath the table.

New Delimited File

File - Step 4 of 4

Add a Schema on repository
Define the Schema

Name: metadata

Comment:

Schema

Click to update schema preview: **Guess**

Description of the Schema

Column	Key	Type	Nullable	Length	Preci...	D...	Comm...
ID_Owners	<input type="checkbox"/>	int	<input checked="" type="checkbox"/>	2			
Reg_Car	<input type="checkbox"/>	String	<input checked="" type="checkbox"/>	10			
Make	<input type="checkbox"/>	String	<input checked="" type="checkbox"/>	10			
Color	<input type="checkbox"/>	String	<input checked="" type="checkbox"/>	6			
ID_Reseller	<input type="checkbox"/>	int	<input checked="" type="checkbox"/>	3			

Buttons: +, -, Up, Down, Save, Print, Copy, Paste

Navigation: ? < Back Next > Finish Cancel

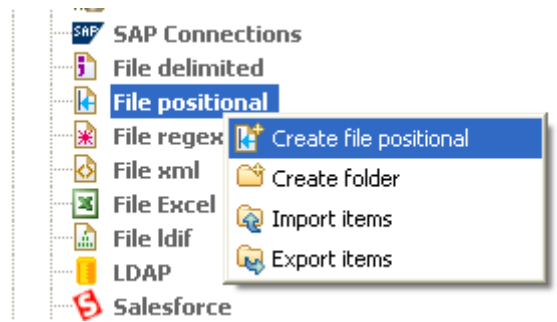
If the Delimited file which the schema is based on has been changed, use the **Guess** button to generate again the schema. Note that if you customized the schema, the **Guess** feature does not retain these changes.

Click **Finish**. The new schema is displayed under the relevant **File Delimited** connection node in the **Repository** tree view.

For further information about how to drop component metadata onto the workspace, see [Section 4.2.2.2, “How to drop components from the Metadata node”](#).

9.6. Setting up a File Positional schema

In the **Repository** tree view, right-click the **File Positional** node and select **Create file positional** on the pop-up menu.



Proceed the same way as for the file delimited connection. Right-click **Metadata** in the **Repository** tree view and select **Create file positional**.

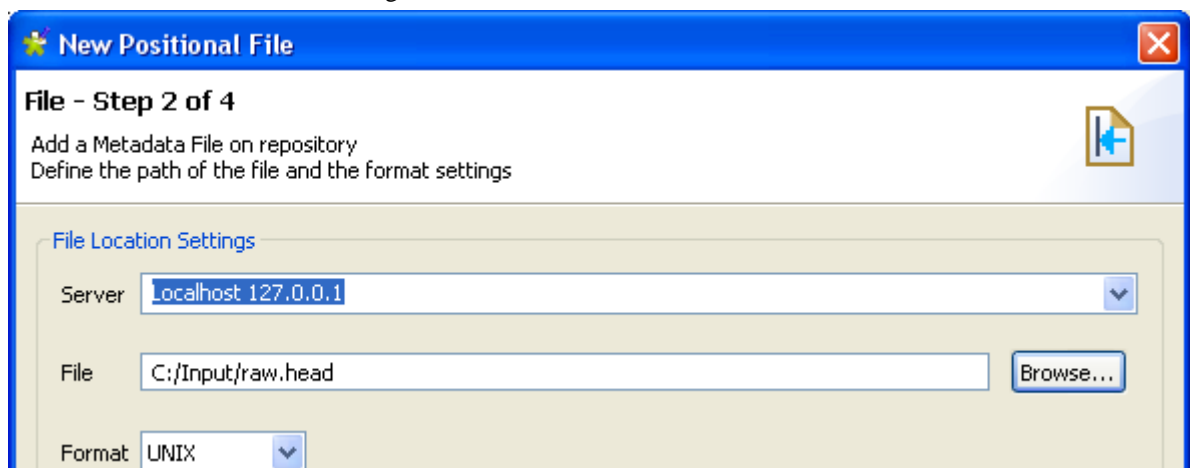
9.6.1. Step 1: General properties

Fill in the schema generic information, such as Schema **Name** and **Description**.

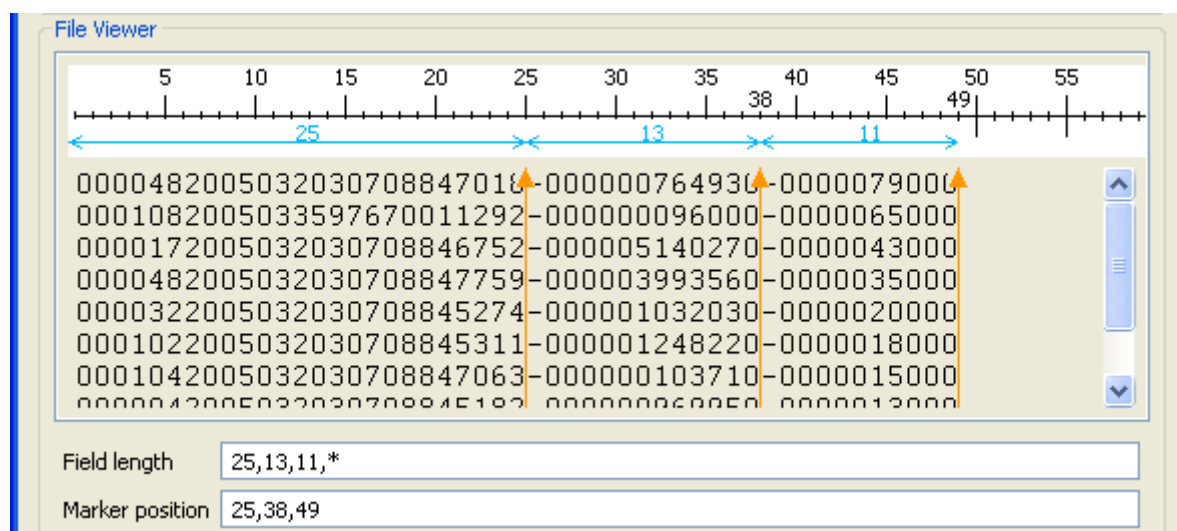
9.6.2. Step 2: Connection and file upload

Then define the positional file connection settings, by filling in the Server IP address and the File path fields.

Like for **Delimited File** schema creation, the format is requested to pre-fill the next step fields. If the file creation OS format is not offered in the list, ignore this field.



The file viewer shows a file preview and allows you to place your position markers.



Click the file preview and set the markers against the ruler. The orange arrow helps you refine the position.

The **Field length** field lists a series of figures separated by commas, these are the number of characters between the separators. The asterisk symbol means all remaining characters on the row, starting from the preceding marker position.

The **Marker Position** shows the exact position of the marker on the ruler. You can change it to specify the position precisely.

You can add as many markers as needed. To remove a marker, drag it towards the ruler.

Click **Next** to continue.

9.6.3. Step 3: Schema refining

The next step opens the schema setting window. As for the **Delimited File** schema, you can refine the schema definition by setting precisely the field and row separators, the header message lines and more...

At this stage, the preview shows the file delimited upon the markers' position. If the file contains column labels, select the check box **Set heading row as column names**.

9.6.4. Step 4: Finalizing the end schema

Step 4 shows the end schema generated. Note that any character which could be misinterpreted by the program is replaced by neutral characters. Underscores replace asterisks, for example.

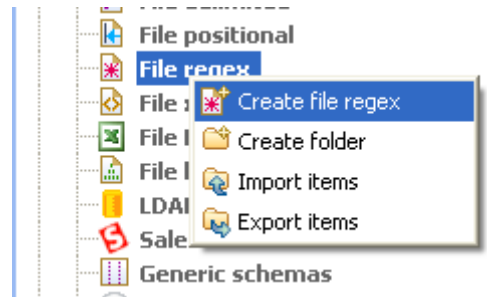
You can customize the metadata name (by default, metadata) and edit it using the tool bar.

You can also retrieve or update the Positional File schema by clicking on **Guess**. Note that, however, any edits to the schema might be lost after "guessing" the file-based schema.

The new schema is displayed under the relevant **File positional** connection node in the **Repository** tree view. You can drop the defined metadata from the **Repository** onto the design workspace. A dialog box then opens in which you can choose which component to use in your Job.

9.7. Setting up a File Regex schema

Regex file schemas are used for files which contain redundant information, such as log files.



Proceed the same way as for the file delimited or positional connection. Right-click **Metadata** in the **Repository** tree view and select **Create file regex**.

9.7.1. Step 1: General properties

Fill in the schema generic information, such as Schema **Name** and **Description**.

9.7.2. Step 2: File upload

Then define the Regex file connection settings, by filling in the Server IP address and the File path fields.

File Settings

Server: Localhost 127.0.0.1

File: C:/Users/fbensaid/Desktop/regex.txt Browse...

Format: WINDOWS

File Viewer

```

firstname=Michael lastname=Jackson age=42
firstname=Elisa lastname=Do Brasil age=28
firstname=Michel lastname=Dujardin age=26
firstname=Robert lastname=Partou age=56
firstname=Diana lastname=Melbourne age=34
firstname=Marie lastname=Dolvina age=46
firstname=Jean lastname=Perfide age=59
firstname=Emilie lastname=Taldor age=21
firstname=Anne-Laure lastname=Paldufier age=67
  
```

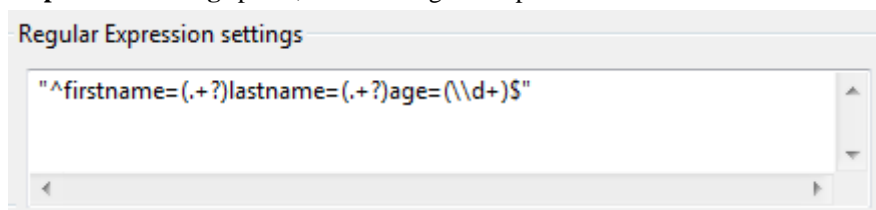
Like for **Delimited File** schema creation, the format is requested for pre-fill purpose of next step fields. If the file creation OS format is not offered in the list, ignore this field.

The file viewer gives an instant picture of the loaded file. Click **Next** to define the schema structure.

9.7.3. Step 3: Schema definition

This step opens the schema setting window. As for the other File schemas, you can refine the schema definition by setting precisely the field and row separators, the header message number of lines and else...

In the **Regular Expression settings** panel, enter the regular expression to be used to delimit the file.



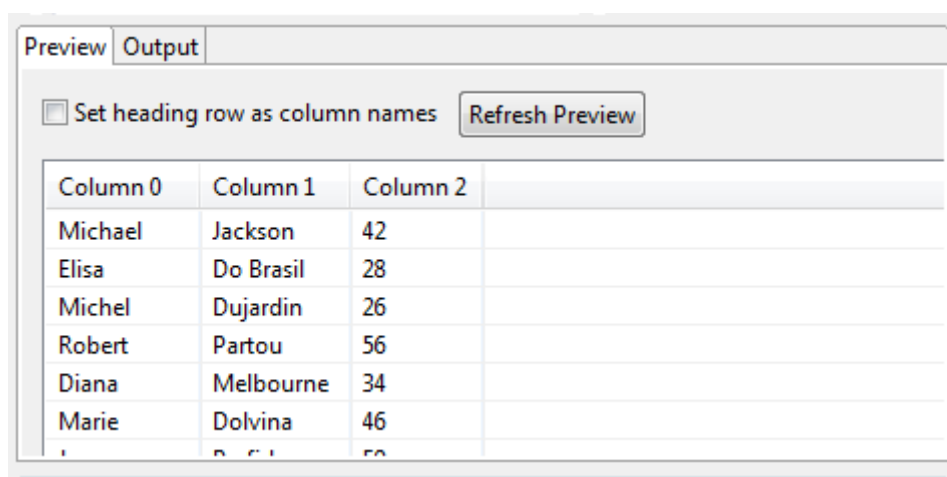
Regular Expression settings

`"^firstname=(.+?)lastname=(.+?)age=(\\d+)$"`



Make sure to include the Regex code in single or double quotes accordingly.

Then click **Refresh preview** to take the changes into account. The button changes to **Wait** until the preview is refreshed.



Preview Output

☐ Set heading row as column names **Refresh Preview**

Column 0	Column 1	Column 2
Michael	Jackson	42
Elisa	Do Brasil	28
Michel	Dujardin	26
Robert	Partou	56
Diana	Melbourne	34
Marie	Dolvina	46

Click **Next** when setting is complete. The last step generates the Regex File schema.

9.7.4. Step 4: Finalizing the end schema

You can add a customized name (by default, metadata) and edit it using the tool bar.

You can also retrieve or update the Regex File schema by clicking on **Guess**. Note however that any edits to the schema might be lost after guessing the file based schema.

Click **Finish**. The new schema is displayed under the relevant **File regex** connection node in the **Repository** tree view.

For further information about how to drop component metadata onto the workspace, see [Section 4.2.2.2, “How to drop components from the Metadata node”](#).

9.8. Setting up an XML file schema

Centralize your XPath query statements from a defined XML file and gather the values retrieved from it.

This is done in the same way as for the connection of delimited or positional files.

According to the option you select, the wizard helps you create either an input or an output schema. In a Job, the **tFileInputXML** component uses the input schema created to read XML files, whereas **tAdvancedFileOutputXML** uses the output schema created to either write an XML file, or to update an existing XML file.



Step 1, in which you enter the general properties of the schema to be created, precedes the step at which you set the type of schema as either input or output. It is therefore advisable to enter names which will help you to distinguish between your input and output schemas.

For further information about reading an XML file, see [Section 9.8.1, “Setting up an XML schema for an input file”](#).

For further information about writing an XML file, see [Section 9.8.2, “Setting up an XML schema for an output file”](#).

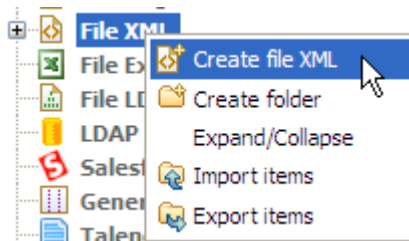
9.8.1. Setting up an XML schema for an input file

This section describes how to define and upload an XML schema for an input file. To define and upload an output file, see [Section 9.8.2, “Setting up an XML schema for an output file”](#).

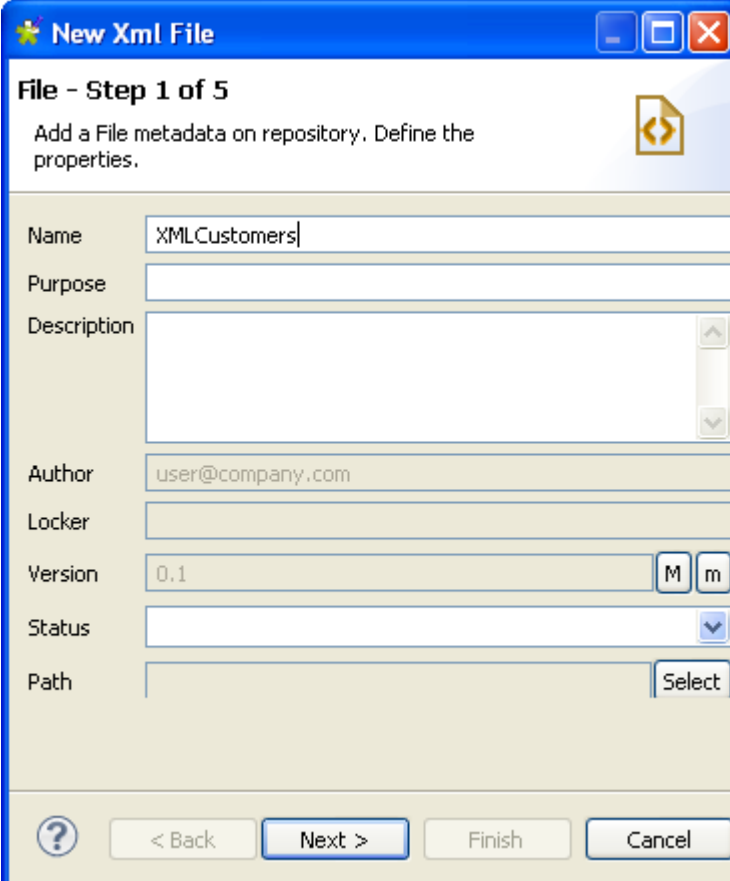
9.8.1.1. Step 1: General properties

In this step, the schema metadata such as the **Name**, **Purpose** and **Description** are set.

1. In the **Repository** view, expand the **Metadata** node.
2. Right click **File XML**, and select **Create file XML** from the pop-up menu.



3. Enter the generic schema information, such as its **Name** and **Description**.



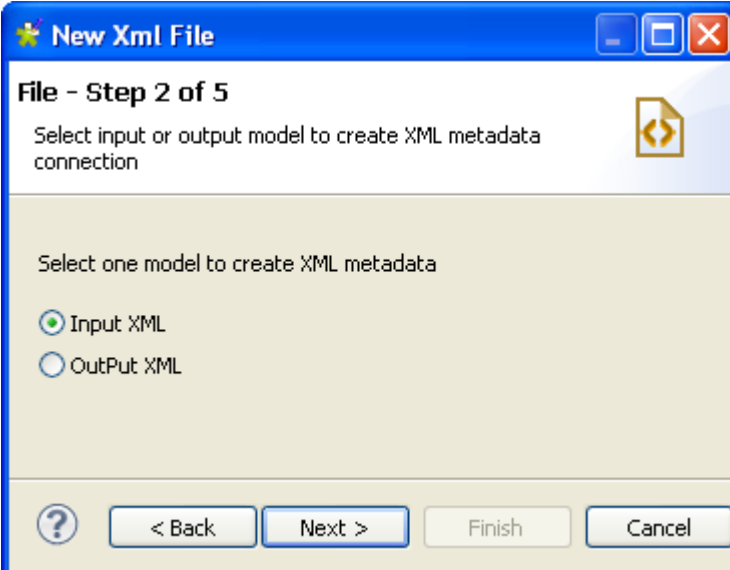
The dialog box is titled "New Xml File" and shows "File - Step 1 of 5". The instruction is "Add a File metadata on repository. Define the properties." The fields are: Name (XMLCustomers), Purpose (empty), Description (empty text area), Author (user@company.com), Locker (empty), Version (0.1) with M and m radio buttons, Status (empty dropdown), and Path (empty text field with a Select button). Navigation buttons at the bottom are Back, Next > (highlighted), Finish, and Cancel.

4. Click **Next** to select the type of schema.

9.8.1.2. Step 2: Setting the type of schema (input)

In this step, the type of schema is set as either input or output. For this procedure, the schema of interest is input.

1. In the dialog box, select **Input XML**.



The dialog box is titled "New Xml File" and shows "File - Step 2 of 5". The instruction is "Select input or output model to create XML metadata connection". The text "Select one model to create XML metadata" is followed by two radio buttons: "Input XML" (selected) and "OutPut XML". Navigation buttons at the bottom are Back, Next > (highlighted), Finish, and Cancel.

2. Click **Next** to upload the input file.

9.8.1.3. Step 3: Uploading the input file

This step involves selecting the input file, its encoding type and defining the number of columns on which the XPath query is to be run. You can also preview the structure of the selected input file, which can be an XML file or an XSD file.



If you load an XSD file,

- the data will be saved in the **Repository**, and therefore the metadata will not be affected by the deletion or displacement of the file.
- you can choose an element as the root of your XML tree.

To upload an XML file, do the following:

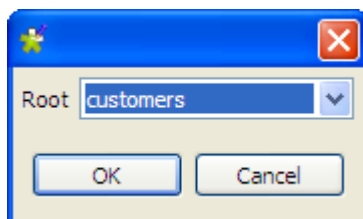
1. Click **Browse...** and browse your directory to the XML file to be uploaded. Alternatively, enter the access path to the file.

The **Schema Viewer** area displays a preview of the XML structure. You can expand and visualize every level of the file's XML tree structure.

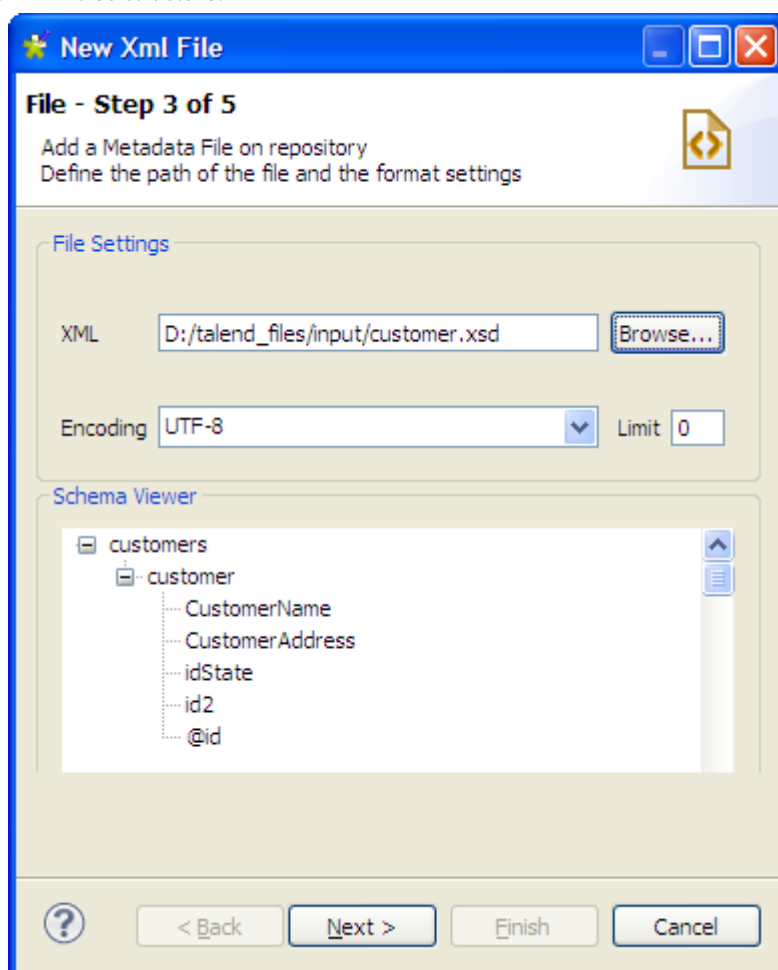
2. Enter the **Encoding** type in the corresponding field if the system does not detect it automatically.
3. In the **Limit** field, enter the number of columns on which the XPath query is to be executed, or 0 if you want to run it against all of the columns.
4. Click **Next** to define the schema parameters.

To upload an XSD file, do the following:

1. Click **Browse...** and browse your directory to the XSD file to be uploaded. Alternatively, enter the access path to the file.
2. In the dialog box the appears, select an element from the **Root** list as the root of your XML tree, and click **OK**.



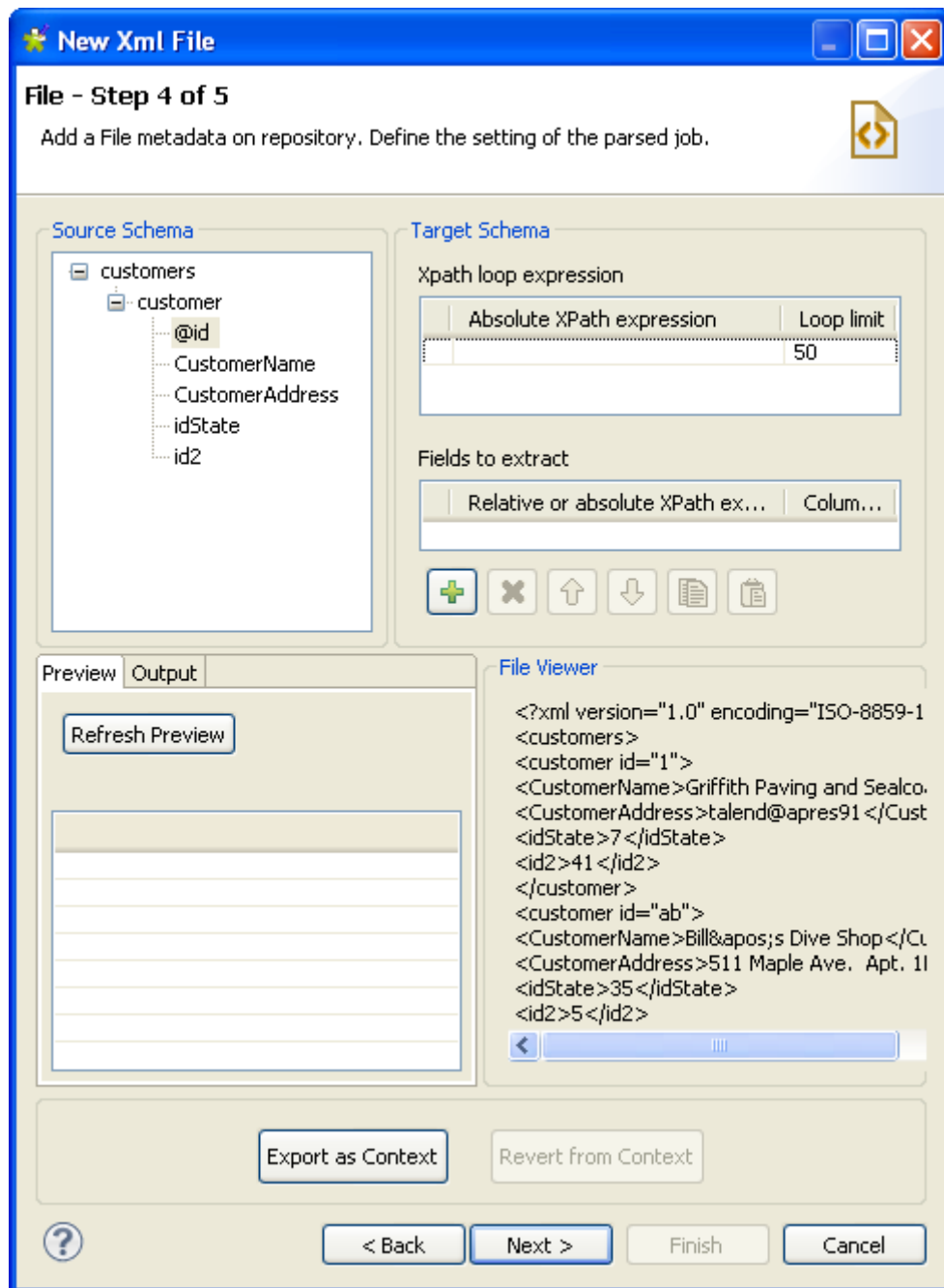
The **Schema Viewer** area displays a preview of the XML structure. You can expand and visualize every level of the file's XML tree structure.



3. Enter the **Encoding** type in the corresponding field if the system does not detect it automatically.
4. In the **Limit** field, enter the number of columns on which the XPath query is to be executed, or 0 if you want to run it against all of the columns.
5. Click **Next** to define the schema parameters.

9.8.1.4. Step 4: Defining the schema

In this step the schema parameters are set.



The schema definition window is composed of four views:

View	Description
Source Schema	Tree view of the XML file.
Target Schema	Extraction and iteration information.
Preview	Preview of the target schema, together with the input data of the selected columns displayed in the defined order. 💡 The preview functionality is not available if you loaded an XSD file.
File Viewer	Preview of the brute data.

First define an Xpath loop and the maximum number of times the loop can run. To do so:

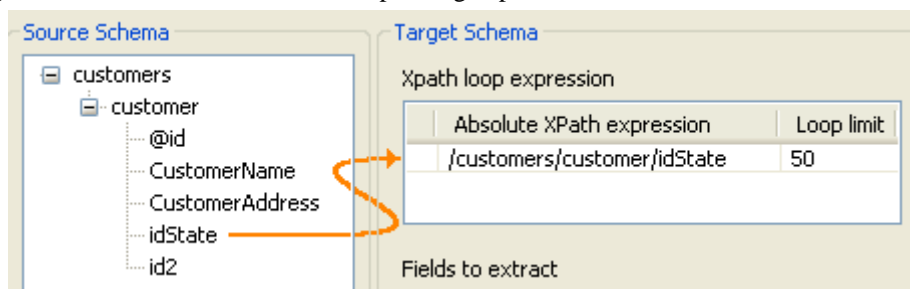
1. Populate the **XPath loop expression** field with the absolute XPath expression for the node to be iterated upon. There are two ways to do this, either:

- enter the absolute XPath expression for the node to be iterated upon (Enter the full expression or press **Ctrl+Space** to use the autocompletion list),

or

- drop a node from the tree view under **Source schema** onto the **Absolute XPath expression** field.

An orange arrow links the node to the corresponding expression.



The **Xpath loop expression** field is mandatory.

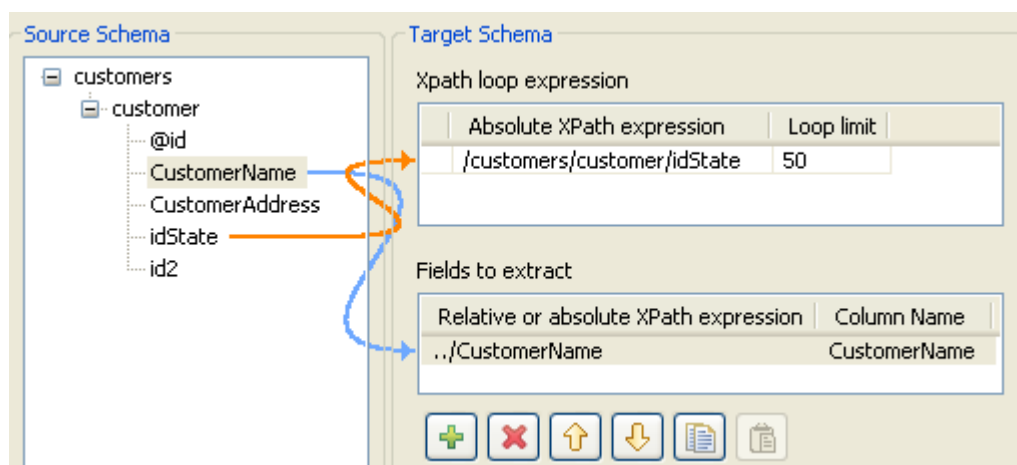
2. In the **Loop limit** field, specify the maximum number of times the selected node can be iterated.

Next, it is necessary to define the fields to be extracted. To do so:





1. Drop the node(s) of interest from the **Source Schema** tree onto the **Relative or absolute XPath expression**.

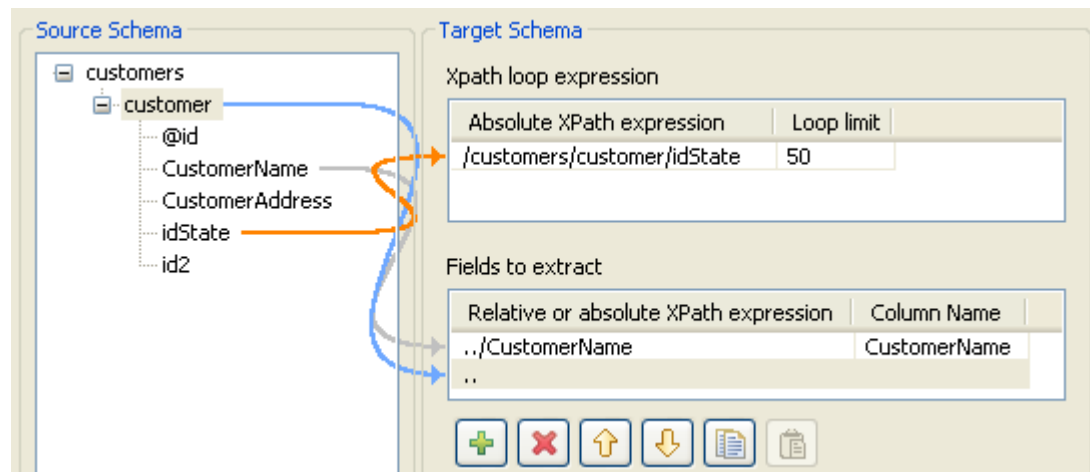


You can select several nodes to drop on the table by pressing **Ctrl** or **Shift** and clicking the nodes of interest. The arrow linking an individual node selected on the **Source Schema** to the **Fields to extract** table are blue in colour. The other ones are gray.



2. If needed, you can add as many columns to be extracted as necessary, delete columns or change the column order using the toolbar:

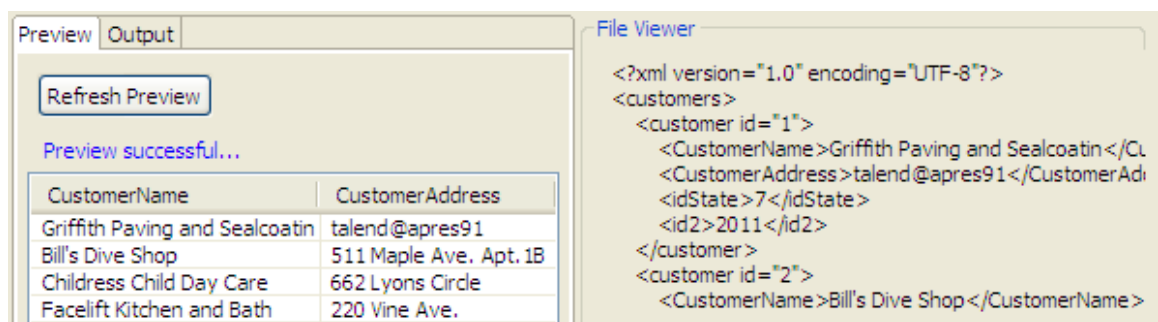
- Add or delete a column using the  and  buttons.
- Change the order of the columns using the  and  buttons.



3. In the **Column name** fields, enter labels for the columns to be displayed in the schema **Preview** area.
4. Click **Refresh Preview** to display a preview of the target schema. The fields are consequently displayed in the schema according to the defined order.



The preview functionality is not available if you loaded an XSD file.



5. Click **Next** to finalize the end schema.

9.8.1.5. Step 5: Finalizing the end schema

The schema generated displays the columns selected from the XML file. You can customize the schema according to your needs, or reload the original schema using the **Guess** button.

New Xml File

File - Step 5 of 5

Add a Schema on repository

Name: metadata

Comment:

Schema

Click to update schema preview:

Description of the Schema

Column	Key	Type	<input checked="" type="checkbox"/>	N..	Date Pattern ...	Length	Precision	Default	Comment
CustomerName	<input type="checkbox"/>	String	<input checked="" type="checkbox"/>				0		
Column1	<input type="checkbox"/>	String	<input checked="" type="checkbox"/>			64	0		

Buttons: +, -, Up, Down, [Icon], [Icon], [Icon], [Icon]

Bottom buttons: ? , < Back, Next >, Finish, Cancel

1. Click the and buttons to add or delete selected columns.
2. Click the and buttons to modify the order of the columns.
3. Click **Finish**.

The new schema, along with its columns, appears under the **File XML** node in the **Repository** tree view.

9.8.2. Setting up an XML schema for an output file

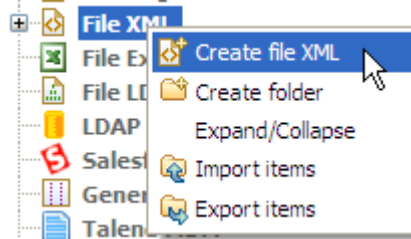
This section describes how to define and upload an XML schema for an output file. To define and upload an XML schema for an input file, see [Section 9.8.1, “Setting up an XML schema for an input file”](#).

9.8.2.1. Step 1: General properties

In this step, the schema metadata such as the **Name**, **Purpose** and **Description** are set.

1. In the **Repository** view, expand the **Metadata** node.

2. Right click **File XML**, and select **Create file XML** from the pop-up menu.



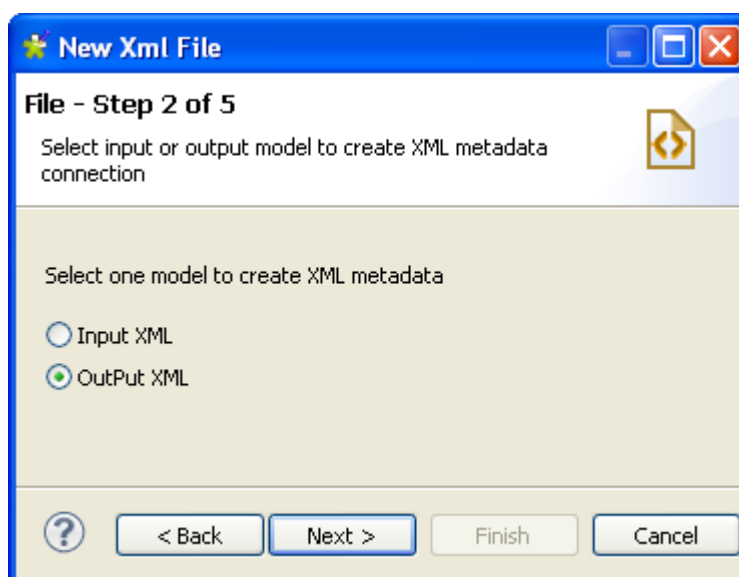
3. Enter the generic schema information, such as its **Name** and **Description**.

4. Click **Next** to set the type of schema.

9.8.2.2. Step 2: Setting the type of schema (output)

In this step, the type of schema is set as either input or output. For this procedure, the schema of interest is output.

1. From the dialog box, select **Output XML**.



2. Click **Next** to define the output file.

9.8.2.3. Step 3: Defining the output file

You can choose whether to create your file manually or from an existing file. If you choose the **Create manually** option you will have to configure your schema, source and target columns yourself. The file is created in a Job using a an XML output component such as **tAdvancedFileOutputXML**. In this example, we will create the output file by loading an existing XML or XSD file.



If you load an XSD file,

- the data will be saved in the **Repository**, and therefore the metadata will not be affected by the deletion or displacement of the file.
- you can choose an element as the root of your XML tree.

To create the output XML schema from an XML file, do the following:

1. Select the **Create from a file** option.
2. Click the **Browse...** button next to the **XML or XSD File** field, browse to the access path to the XML file the structure of which is to be applied to the output file, and double-click the file.

The **File Viewer** area displays a preview of the XML structure, and the **File Content** area displays a maximum of the first 50 rows of the file.

New Xml File

File - Step 3 of 5

Select create manually or from a file
Define the output file

Output Setting

☐ Create manually
☒ Create from a file

XML or XSD File

Encoding Limit

Output File Path

Output file

File Viewer

- customers
 - customer
 - @id
 - CustomerName
 - CustomerAddress
 - idState
 - id2

File Content

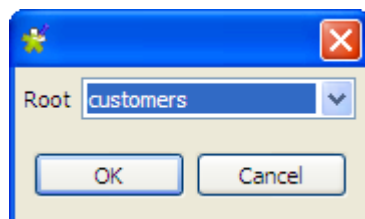
```
<?xml version="1.0" encoding="UTF-8"?>
<customers>
  <customer id="1">
    <CustomerName>Griffith Paving and Sealco
    <CustomerAddress>talend@apres91</Cus
    <idState>7</idState>
    <id2>2011</id2>
```

3. Enter the **Encoding** type in the corresponding field if the system does not detect it automatically.
4. In the **Limit** field, enter the number of columns on which the XPath query is to be executed, or enter 0 if you want it to be run against all of the columns.
5. In the **Output File** field, in the **Output File Path** zone, browse to or enter the path to the output file. If the file does not exist as yet, it will be created during the execution of a Job using a **tAdvancedFileOutputXML** component. If the file already exists, it will be overwritten.
6. Click **Next** to define the schema.

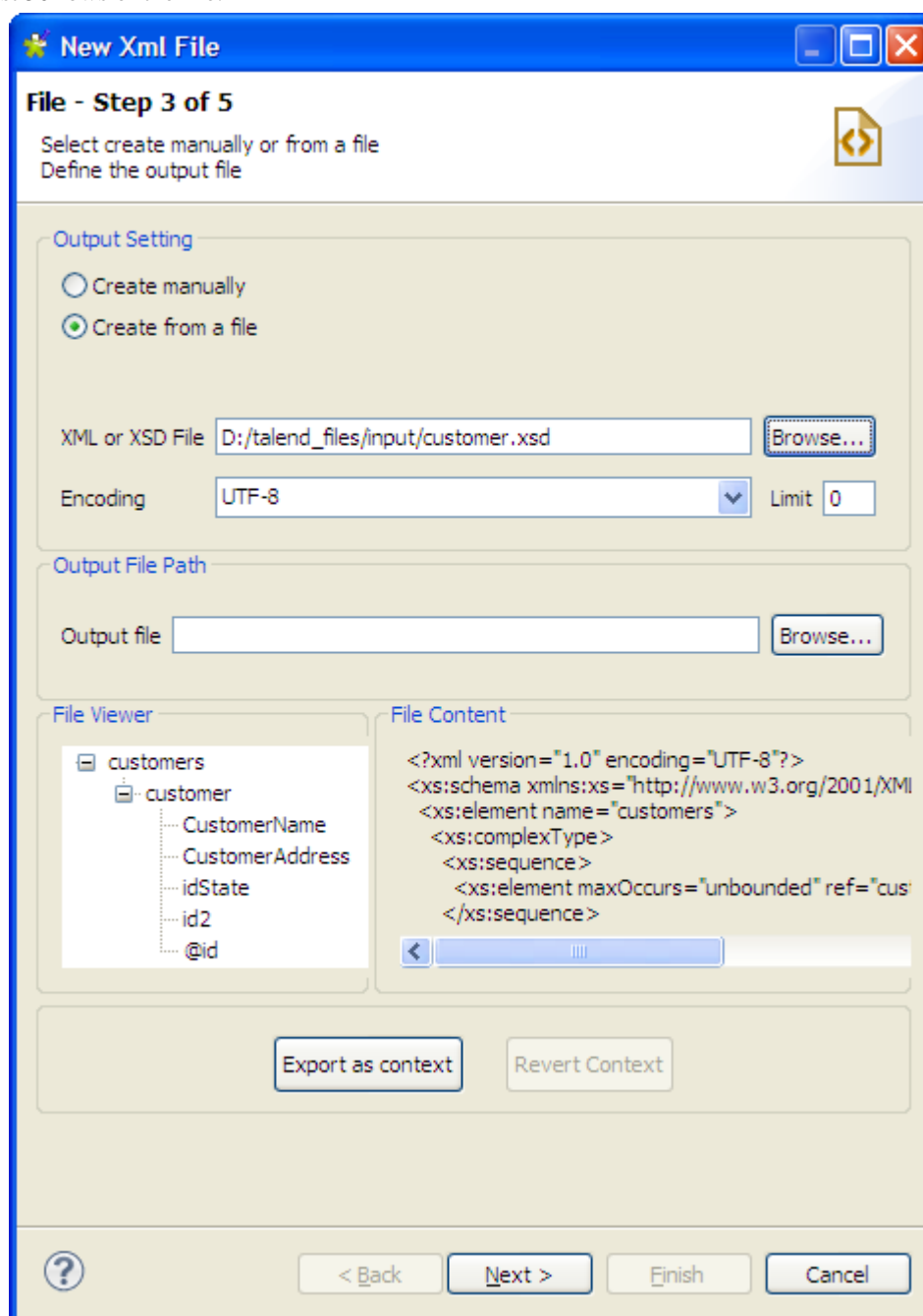
To create the output XML schema from an XSD file, do the following:

1. Select the **Create from a file** option.
2. Click the **Browse...** button next to the **XML or XSD File** field, browse to the access path to the XSD file the structure of which is to be applied to the output file, and double-click the file.

3. In the dialog box that appears, select an element from the **Root** list as the root of your XML tree, and click **OK**.



The **File Viewer** area displays a preview of the XML structure, and the **File Content** area displays a maximum of the first 50 rows of the file.








4. Enter the **Encoding** type in the corresponding field if the system does not detect it automatically.
5. In the **Limit** field, enter the number of columns on which the XPath query is to be executed, or enter 0 if you want it to be run against all of the columns.






6. In the **Output File** field, in the **Output File Path** zone, browse to or enter the path to the output file. If the file does not exist as yet, it will be created during the execution of a Job using a **tAdvancedFileOutputXML** component. If the file already exists, it will be overwritten.
7. Click **Next** to define the schema.

9.8.2.4. Step 4: Defining the schema

Upon completion of the previous operations, the columns in the **Linker Source** area are automatically mapped to the corresponding ones in the **Linker Target** area, as indicated by blue arrow links. You can customize the mappings while defining the schema.

In this step, you need to define the output schema. The following table describes how:

To...	Perform the following...
Define a loop element	<p>In the Linker Target area, right-click the element of interest and select Set As Loop Element from the contextual menu.</p> <p> It is a mandatory operation to define an element to run a loop on.</p>
Define a group element	<p>In the Linker Target area, right-click the element of interest and select Set As Group Element from the contextual menu.</p> <p> You can set a parent element of the loop element as a group element on the condition that the parent element is not the root of the XML tree.</p>
Create a child element for an element	<p>In the Linker Target area,</p> <ul style="list-style-type: none"> Right-click the element of interest and select Add Sub-element from the contextual menu, enter a name for the sub-element in the dialog box that appears, and click OK, or Select the element of interest, click the  button at the bottom, select Create as sub-element in the dialog box that appears, and click OK. Then, enter a name for the sub-element in the next dialog box and click OK.
Create an attribute for an element	<p>In the Linker Target area,</p> <ul style="list-style-type: none"> Right-click the element of interest and select Add Attribute from the contextual menu, enter a name for the attribute in the dialog box that appears, and click OK, or Select the element of interest, click the  button at the bottom, select Create as attribute in the dialog box that appears, and click OK. Then, enter a name for the attribute in the next dialog box and click OK.
Create a name space for an element	<p>In the Linker Target area,</p> <ul style="list-style-type: none"> Right-click the element of interest and select Add Name Space from the contextual menu, enter a name for the name space in the dialog box that appears, and click OK, or Select the element of interest, click the  button at the bottom, select Create as name space in the dialog box that appears, and click OK. Then, enter a name for the name space in the next dialog box and click OK.

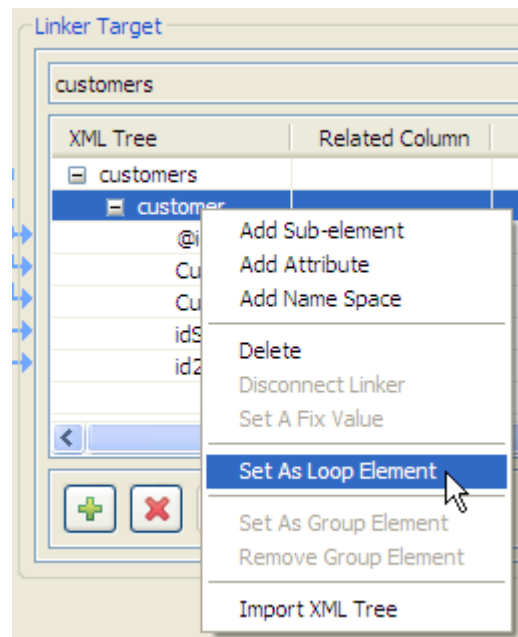
To...	Perform the following...
Delete one or more elements/attributes/name spaces	<p>In the Linker Target area,</p> <ul style="list-style-type: none"> Right-click the element(s)/attribute(s)/name space(s) of interest and select Delete from the contextual menu, or Select the element(s)/attribute(s)/name space(s) of interest and click the  button at the bottom, or Select the element(s)/attribute(s)/name space(s) of interest and press the Delete key. <p> Deleting an element will also delete its children, if any.</p>
Adjust the order of one or more elements	<p>In the Linker Target area, select the element(s) of interest and click the  and  buttons.</p>
Set a static value for an element/attribute/name space	<p>In the Linker Target area, right-click the element/attribute/name space of interest and select Set A Fix Value from the contextual menu.</p> <p> <ul style="list-style-type: none"> The value you set will replace any value retrieved for the corresponding column from the incoming data flow in your Job. You can set a static value for a child element of the loop element only, on the condition that the element does not have its own children and does not have a source-target mapping on it. </p>
Create a source-target mapping	<p>Select the column of interest in the Linker Source area, drop it onto the node of interest in the Linker Target area, and select Create as sub-element of target node, Create as attribute of target node, or Add linker to target node according to your need in the dialog box that appears, and click OK.</p> <p>If you choose an option that is not permitted for the target node, you will see a warning message and your operation will fail.</p>
Remove a source-target mapping	<p>In the Linker Target area, right-click the node of interest and select Disconnect Linker from the contextual menu.</p>
Create an XML tree from another XML or XSD file	<p>Right-click any schema item in the Linker Target area and select Import XML Tree from the contextual menu to load another XML or XSD file. Then, you need to create source-target mappings manually and define the output schema all again.</p>



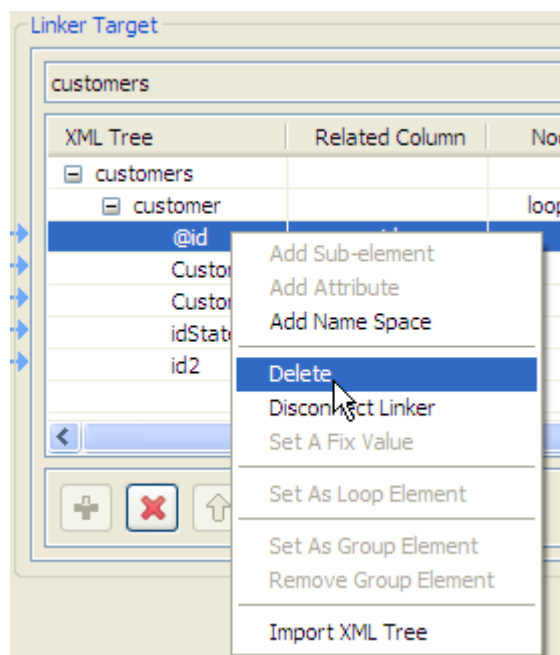
You can select and drop several fields at a time, using the **Ctrl + Shift** technique to make multiple selections, therefore making mapping faster. You can also make multiple selections for right-click operations.

In this example, we base the output schema on the loaded *customer.xml*, define a loop to run on the *customer* element, and define the *id* node as a child element, rather than an attribute as in the loaded file. To do so:

1. In the **Linker Target** area, right-click the *customer* element and select **Set As Loop Element** from the contextual menu.

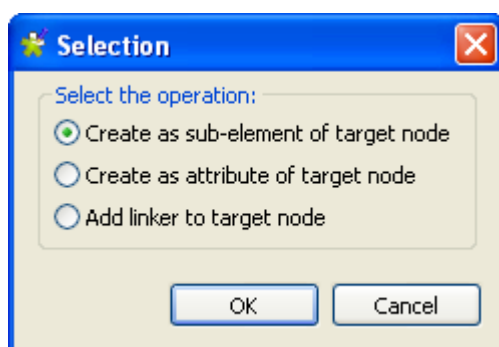


2. Right-click the **id** note and select **Delete** from the contextual menu.



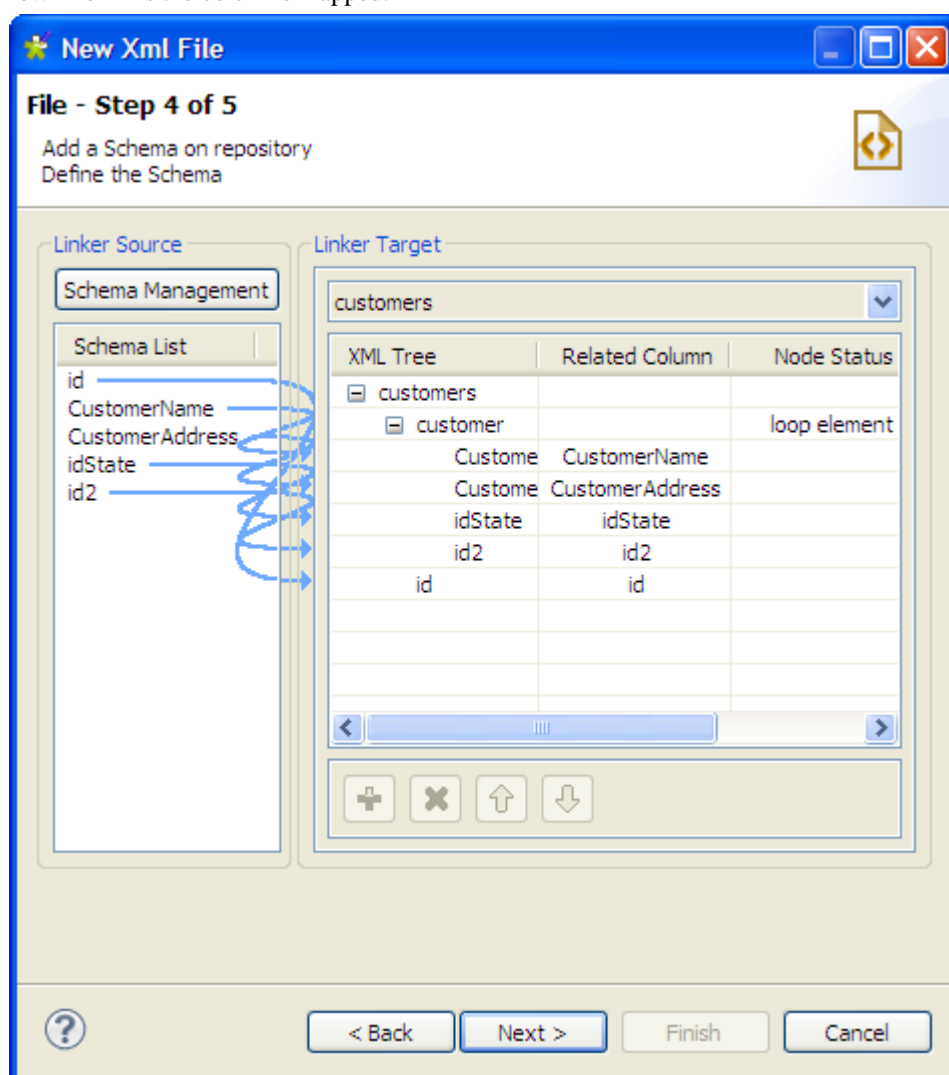
3. Select the *id* column in the **Linker Source** area, and drop it onto the *customer* element in the **Linker Target** area.

The **[Selection]** dialog box appears, asking you to define the source-target relationship.



4. Select **Create as sub-element of target node** and click **OK** to validate the choice.

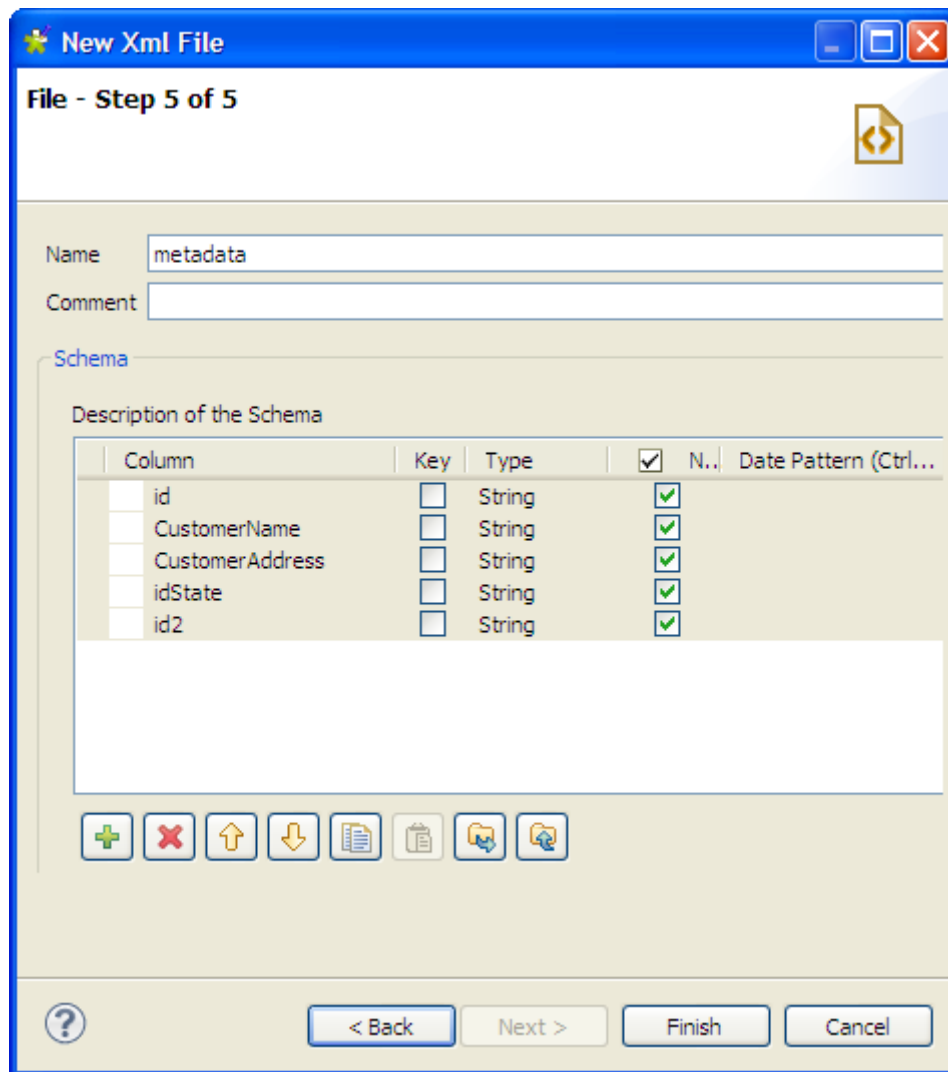
A blue arrow line links the columns mapped.



5. Click **Next** to finalize the end schema.

9.8.2.5. Step 5: Finalizing the end schema

Step 5 displays the end schema generated.



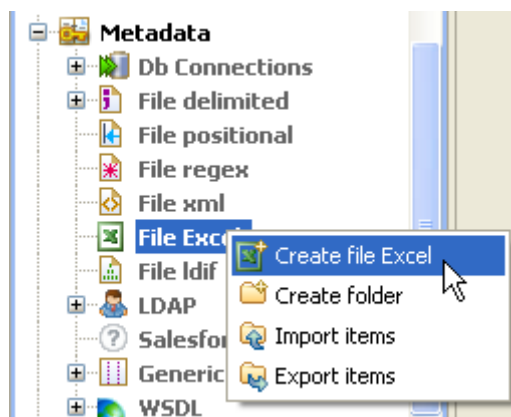
You can change the metadata in the **Name** field (*metadata*, by default), add a **Comment** in the corresponding field and make further modifications using the toolbar, for example:

- Add or delete a column using the and buttons.
- Change the order of the columns using the and buttons.
- Click **Finish** to finalize creation of the XML output file.

The new schema is displayed under the corresponding **File XML** node in the **Repository** tree view.

9.9. Setting up a File Excel schema

In the **Repository** tree view, right-click **File Excel** tree node, and select **Create file Excel** on the pop-up menu.



Then proceed the same way as for the file delimited connection.

9.9.1. Step 1: General properties

Fill in the schema general information, such as Schema **Name** and **Description**.

9.9.2. Step 2: File upload

Browse to the file to upload and fill out the **File** field.



You can upload all types of Excel files including *..xlsx* of Excel 2007.

New Excel File

File - Step 2 of 4

Add a Metadata File on repository
Define the path of the file and the format settings

File Settings

Server: localhost 127.0.0.1

File: D:/Input/video/Video_Collection.xlsx Browse...

File Viewer and Sheets setting

Set sheets parameters: ☒ All sheets/DSelect she...
 ☒ Sheet1
 ☐ Sheet2
 ☐ Sheet3

Please select sheet(Sheet structure as schema guide): Sheet1

A	B	C	D	E	F
Video co...					
id	Title	Category	Year	Language	Cast
1	Tootsie	Comedy	1982	English	Dustin ...
2	The 5t...	Science...	1997	French	Bruce ...
3	Leon, t...	Action ...	1994	French	Jean R...

< Back Next > Finish Cancel

The **File viewer and sheets setting** area shows a preview of the file.

In the **Set sheets parameters list**, select the dialog box next to the sheet you want to upload. By default the file preview displays the Sheet1.

Select another sheet on the drop-down list, if need be and check that the file is properly read on the preview table.

Click **Next** to continue.

9.9.3. Step 3: Schema refining

The next step opens the schema setting window.

New Excel File

File - Step 3 of 4

Add a Metadata File on repository
Define the setting of the parse job

File Settings

Encoding: UTF-8

☐ Advanced separator(for number)

Thousands separator: ,

Decimal separator: .

Rows To Skip

If any rows must be ignored, specify the following parameters

Header: ☒ 2

Footer: ☐

☐ Skip empty row

The same way as for the File Delimited schema procedure, you can set precisely the separator, the rows that should be skipped as they are header or footer.

Metadata column setting

First column: 1

Last column:

Limit Of Rows

If the number of lines must be limited, specify this number

Limit: ☐

You can fill out the **First column** and **Last column** fields, to set precisely the columns to be read in the file. You can need to skip column A for example as it may not contain proper data to be processed.

Preview

☒ Set heading row as column names Refresh Preview

A	B	C	D	E	F
Video collect...					
id	Title	Category	Year	Language	Cast
1	Tootsie	Comedy	1982	English	Dustin Hoffman, Jessica Lange, Sydney Pollack
2	The 5th Element	Science Fiction	1997	French	Bruce Willis, Gary Oldman, Milla Jovovich
3	Leon, the Professional	Action drama	1994	French	Jean Reno, Gary Oldman, Nathalie Portman

Also check the **Set heading row as column names** to take into account the heading names. Make sure you press **Refresh** to be able to view the change in the preview table.

Then click **Next** to continue.

9.9.4. Step 4: Finalizing the end schema

Step 4 shows the end schema generated. Note that any character which could be misinterpreted by the program is replaced by neutral characters, like underscores replace asterisks.

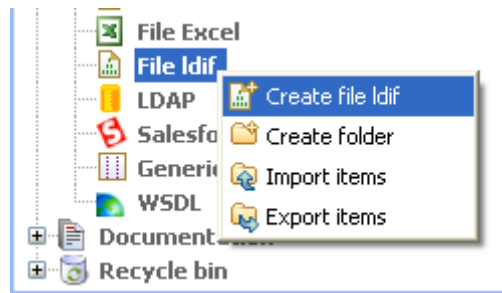
You can add a customized name (by default, metadata) and make edits to it using the tool bar.

You can also retrieve or update the Excel File schema by clicking on **Guess**. Note however that any custom edits to the schema might thus be lost using this feature.

Click **Finish**. The new schema is displayed under the relevant File Excel connection node in the **Repository** tree view.

9.10. Setting up a File LDIF schema

LDIF files are directory files described by attributes. File LDIF metadata centralize these LDIF-type files and their attribute description.



Proceed the same way as for other file connections. Right-click **Metadata** in the **Repository** tree view and select **Create file Ldif**.



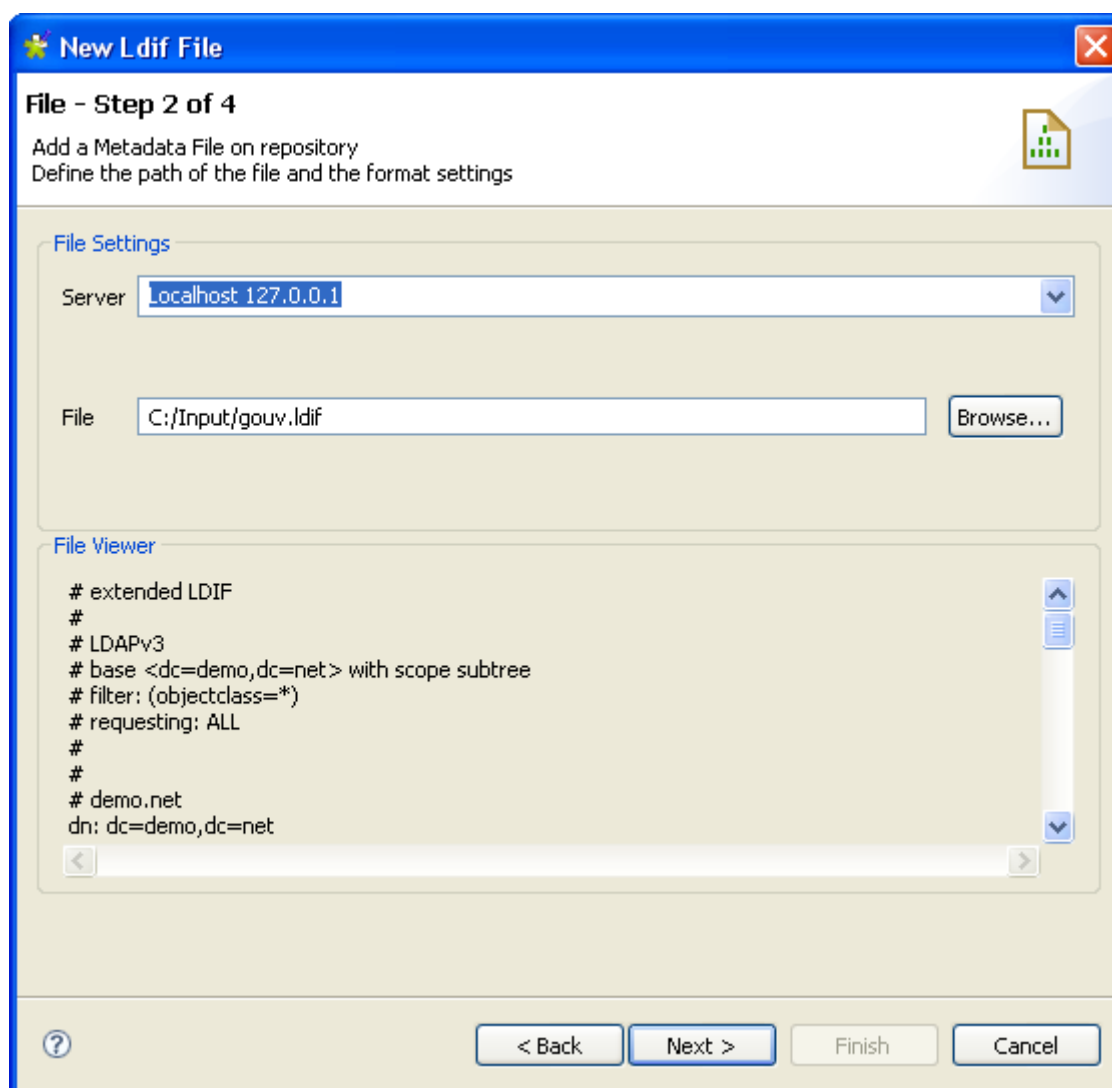
Make sure that you installed the relevant module as described in the Installation guide. For more information, check out <http://talendforge.org/wiki/doku.php>.

9.10.1. Step 1: General properties

On the first step, fill in the schema generic information, such as Schema name and description.

9.10.2. Step 2: File upload

Then define the Ldif file connection settings by filling in the **File path** field.



The connection functionality to a remote server is not operational yet for the LDIF file collection.

The File viewer provides a preview of the file's first 50 rows.

9.10.3. Step 3: Schema definition

The list of attributes of the description file displays on the top of the panel. Select the attributes you want to extract from the LDIF description file, by selecting the relevant check boxes.

New Ldif File

File - Step 3 of 4

Add a Metadata File on repository
Define the setting of the parse job

List Attributes of Ldif file

Attributes
<input type="checkbox"/> password
<input checked="" type="checkbox"/> name
<input type="checkbox"/> changetype
<input checked="" type="checkbox"/> lastname

Preview

Refresh Preview

dn	name	lastname
jbzy	jbzy	taus
fkcd	fkcd	kznj
beyy	beyy	yhwg
kskb	kskb	vuxx
fpfc	fpfc	ryzy
doqp	doqp	eumq
jpxi	jpxi	ocnb

< Back Next > Finish Cancel

Click **Refresh Preview** to include the selected attributes into the file preview.



DN is omitted in the list of attributes as this key attribute is automatically included in the file preview, hence in the generated schema.

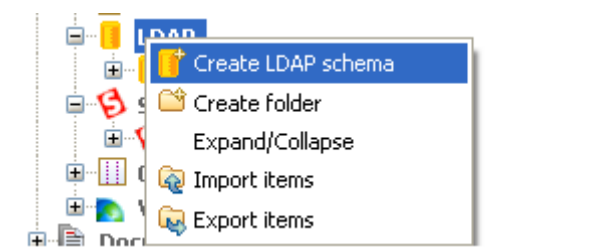
9.10.4. Step 4: Finalizing the end schema

The schema generated shows the columns of the description file. You can customize it upon your needs and reload the original schema using the **Guess** button.

Click **Finish**. The new schema is displayed under the relevant Ldif file connection node in the **Repository** tree view.

9.11. Setting up an LDAP schema

In the **Repository** tree view, right-click the **LDAP** tree node, and select **Create LDAP schema** on the pop-up menu.



Unlike the DB connection wizard, the LDAP wizard gathers both file connection and schema definition in a four-step procedure.

9.11.1. Step 1: General properties

On the first step, fill in the schema generic information, such as Schema **Name** and **Description**.

For further information, see [Section 9.2.1, “Step 1: General properties”](#).

9.11.2. Step 2: Server connection

Fill the connection details.

File – Step 2 of 5

Add a Metadata File on repository
Define the path of the file and the format settings

Network Parameter

Hostname:

Port:

Encryption method:

Click the button to check connection status.

Then check your connection using **Check Network Parameter** to verify the connection and activate the **Next** button.

Field	Description
Host	LDAP Server IP address
Port	Listening port to the LDAP directory
Encryption method	LDAP : no encryption is used LDAPS : secured LDAP TLS : certificate is used

Click **Next** to validate this step and continue.

9.11.3. Step 3: Authentication and DN fetching

In this view, set the authentication and data access mode.

Click **Check authentication** to verify your access rights.

Field	Description
Authentication method	Simple authentication: requires Authentication Parameters field to be filled in Anonymous authentication: does not require authentication parameters
Authentication Parameters	Bind DN or User: login as expected by the LDAP authentication method Bind password: expected password Save password: remembers the login details.
Get Base DN from Root DSE / Base DN	Path to user's authorized tree leaf Fetch Base DN's button retrieves the DN automatically from Root.
Alias Dereferencing	Never allows to improve search performance if you are sure that no aliases is to be dereferenced. By default, Always is to be used. Always: Always dereference aliases

Field	Description
	Never: Never dereferences aliases. Searching: Dereferences aliases only after name resolution. Finding: Dereferences aliases only during name resolution
Referral Handling	Redirection of user request: Ignore: does not handle request redirections Follow: does handle request redirections
Limit	Limited number of records to be read

Click **Fetch Base DN**s to retrieve the DN and click the **Next** button to continue.

9.11.4. Step 4: Schema definition

Select the attributes to be included in the schema structure.

Add a filter if you want selected data only.

Create new LDAP schema

File - Step 4 of 5

Add a Metadata File on repository
Define the setting of the parse job

List attributes of LDAP Schema

Attributes

- ☐ description
- ☐ userpassword
- ☒ uid
- ☐ sn
- ☐ cn

Filter

(objectClass=*)

Preview

Refresh Preview

uid	mail	givenname	telephonenumber
PIERRE DUPONT	Pierre.Dupont@talend.com	PIERRE	00149684750
PIERRE DUPON...	mhirt78@talend.com	PIERRE	
mhirt			
greg			
..			

< Back Next > Finish Cancel

Click **Refresh Preview** to display the selected column and a sample of the data.

Then click **Next** to continue.

9.11.5. Step 5: Finalizing the end schema

The last step shows the LDAP schema generated. You can customize the schema using the toolbar underneath the table.

Create new LDAP schema

File - Step 5 of 5

Add a Metadata File on repository
Define the setting of the parse job

Name:

Comment:

Schema

Click to update schema preview

Description of the Schema

Column	Key	Type	Nullable	Date ...	Length	Precision
uid	<input type="checkbox"/>	String	<input checked="" type="checkbox"/>		15	
mail	<input type="checkbox"/>	String	<input checked="" type="checkbox"/>		24	
givenname	<input type="checkbox"/>	String	<input checked="" type="checkbox"/>		6	
telephonenumber	<input type="checkbox"/>	String	<input checked="" type="checkbox"/>		11	

Toolbar:

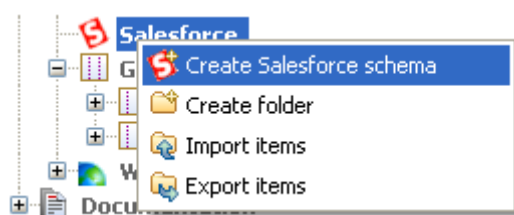
Buttons:

If the LDAP directory which the schema is based on has changed, use the **Guess** button to generate again the schema. Note that if you customized the schema, your changes will not be retained after the **Guess** operation.

Click **Finish**. The new schema is displayed under the relevant LDAP connection node in the **Repository** tree view.

9.12. Setting up a Salesforce connection

In the **Repository** tree view, right-click the **Salesforce** tree node, and select **Create Salesforce Connection** on the pop-up menu.



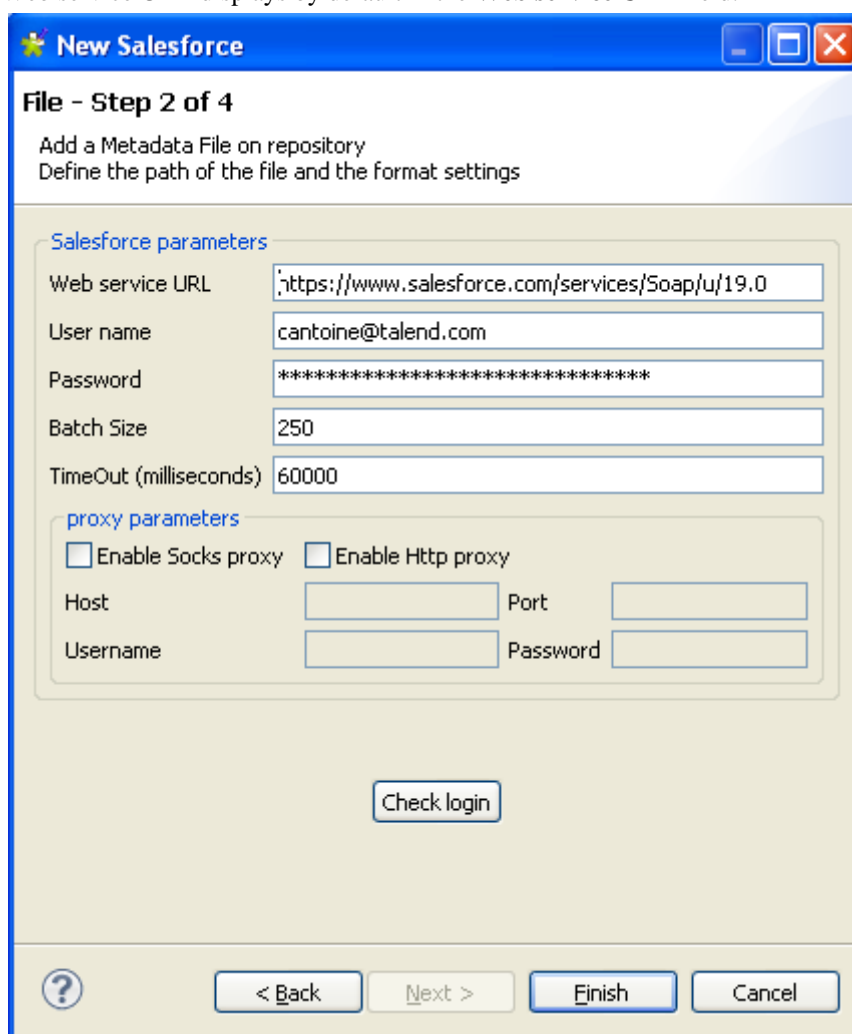
Then proceed the same way as for any other metadata connection.

9.12.1. Step 1: General properties

Fill in the schema general information, such as **Name** and **Description**.

9.12.2. Step 2: Connection to a Salesforce account

The Salesforce Web service URL displays by default in the **Web service URL** field.

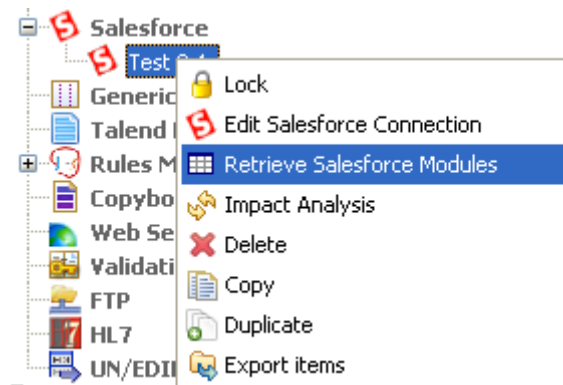


To define a Salesforce connection, do the following:

1. Enter your **User name** and **Password** in the corresponding fields to connect to your Salesforce account through the salesforce webservice.
2. Click **Check login** to verify that you can connect without issue.
3. Click **Finish** to close the wizard.

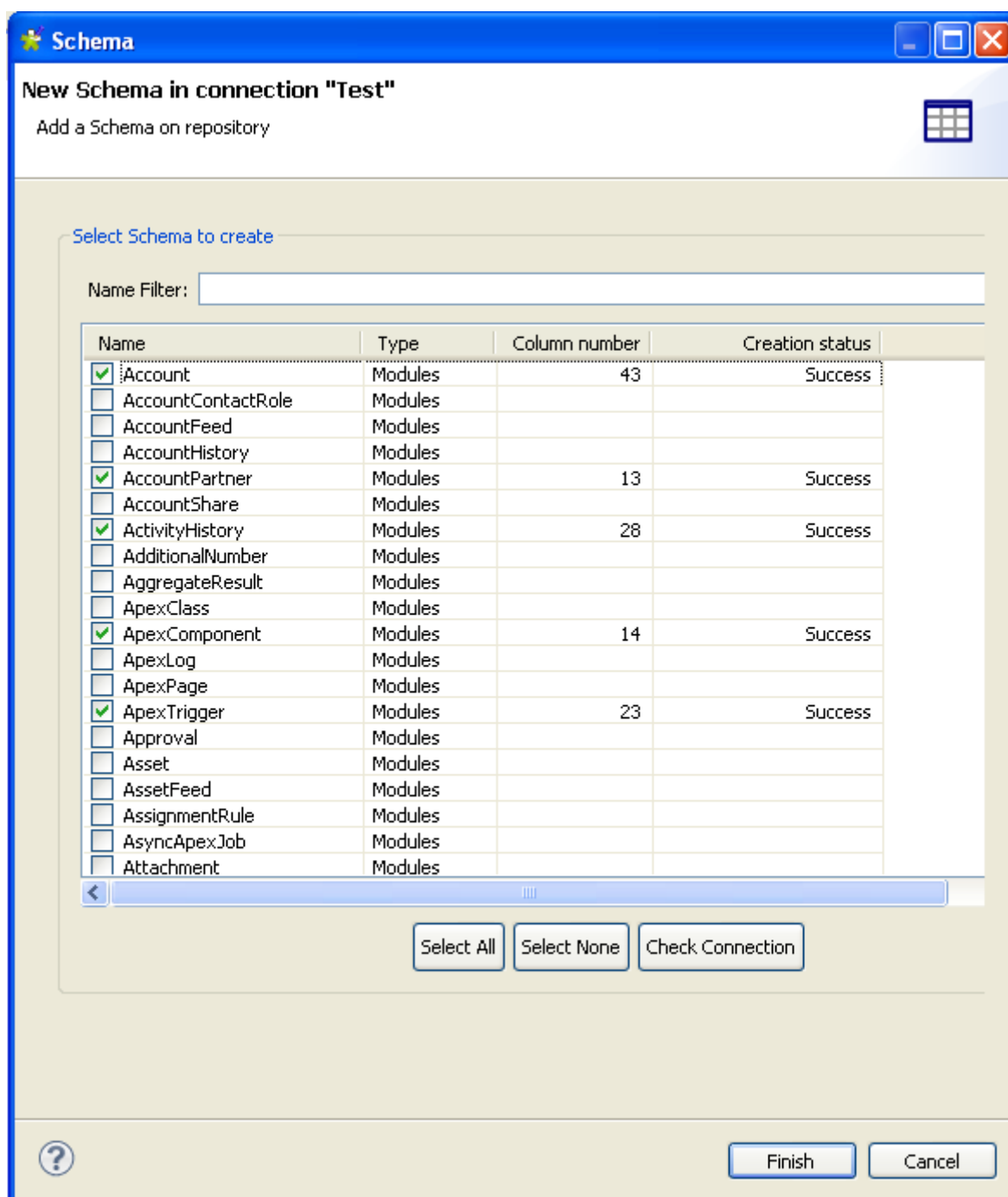
9.12.3. Step 3: Retrieving Salesforce modules

The next step retrieves Salesforce modules based on the connection you set up in Step 2.



In the **Repository** tree view, expand the **Connection** node, right-click the connection you set up in Step 2, and select **Retrieve Salesforce Modules** in the pop-up menu.

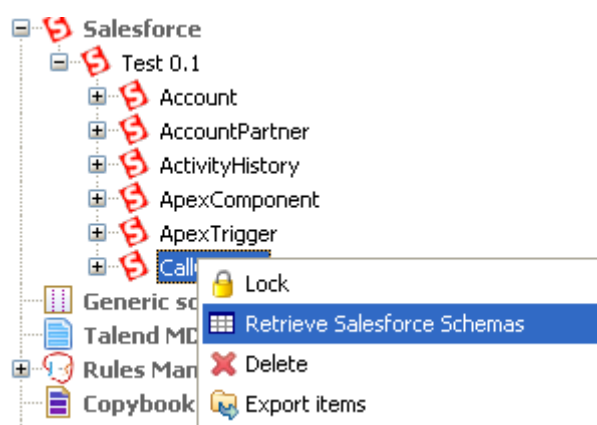
In the **Select Schema to create** area, you can narrow down the selection by filtering schemas displayed. Type in the schema you want to filter on in **Name Filter** field. To retrieve more modules, select the check boxes for respective schema.



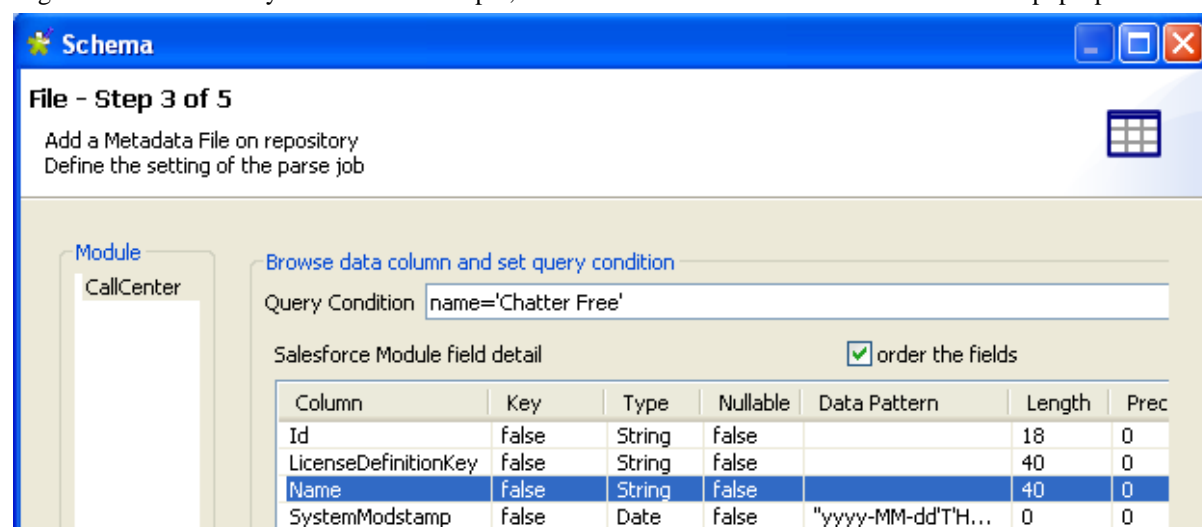
Click **Check Connection** to verify the creation status and click **Finish** to save the modules you retrieved.

9.12.4. Step 4: Retrieving Salesforce schemas

This step opens the schema setting window based on the module you retrieved in Step 3.

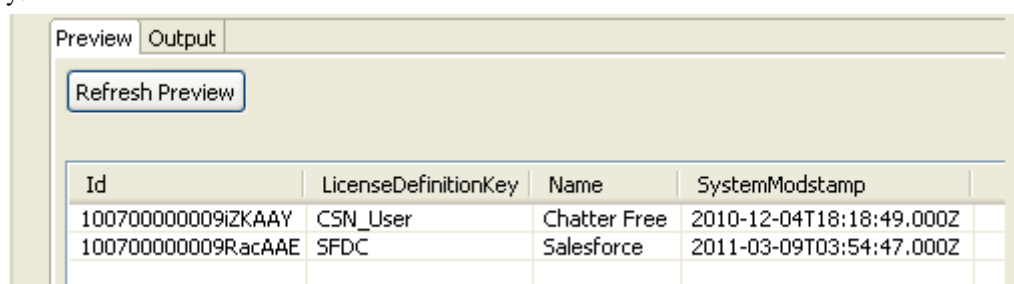


Right-click the module you retrieved in Step 3, and select **Retrieve Salesforce Schemas** in the pop-up menu.



In the **Browse data column and set query condition** area, you can narrow down the selection by filtering displayed data. To do that, type in the column you want to filter on and the value you want to drill down to in the **Query Condition** field.

The columns in the **Column** list are listed in alphabetical order. Clear the **order the fields** check box to list them randomly.



Click **Refresh Preview**, if you entered a query condition, so that the preview gets updated accordingly. By default, the preview shows all columns of the selected module.

Then click **Next** to continue.

9.12.5. Step 5: Finalizing the end schema

Step 5 shows the final generated schema.

You can add a customized name (by default, metadata) and edit the schema using the tool bar.

Schema

File - Step 4 of 5

Add a Schema on repository
Define the Schema

Schema

CallCenter

Name
CallCenter

Comment

Schema

Click "Guess" button to update the schema below according to your settings

Guess

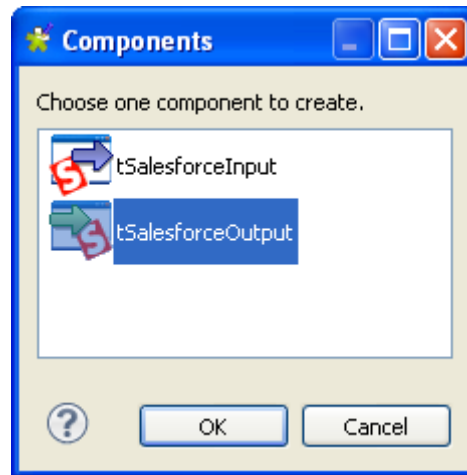
Description of the Schema

Column	Key	Type	<input checked="" type="checkbox"/>	N..	Date Pat...	Len...	Pre...
Id	<input type="checkbox"/>	String	<input type="checkbox"/>			18	0
Name	<input type="checkbox"/>	String	<input type="checkbox"/>			80	0
NamespacePr...	<input type="checkbox"/>	String	<input checked="" type="checkbox"/>			15	0
Description	<input type="checkbox"/>	String	<input checked="" type="checkbox"/>			255	0
TableEnumOrId	<input type="checkbox"/>	String	<input type="checkbox"/>			40	0
IsActive	<input type="checkbox"/>	bool...	<input type="checkbox"/>				0
CreatedById	<input type="checkbox"/>	String	<input type="checkbox"/>			18	0
CreatedDate	<input type="checkbox"/>	Date	<input type="checkbox"/>		"yyyy-M...		0
LastModifiedB...	<input type="checkbox"/>	String	<input type="checkbox"/>			18	0
LastModifiedD...	<input type="checkbox"/>	Date	<input type="checkbox"/>		"yyyy-M...		0

You can also retrieve or update the Salesforce schema by clicking **Guess**. Note however, that any changes or customization of the schema might be lost using this feature.

Click **Finish**. The new schema is displayed in the **Repository** tree view under the relevant File Excel connection node.

You can drop the metadata defined from the **Repository** onto the design workspace. A dialog box opens in which you can choose to use either a **tSalesforceInput** or **tSalesforceOutput** component in your Job.



For more information about dropping component metadata in the design workspace, see [Section 4.2.2.2, “How to drop components from the Metadata node”](#).

9.13. Setting up a Generic schema

Talend Open Studio for ESB allows you to create any schema from scratch if none of the specific metadata wizards matches your need or if you do not have any source file to take the schema from. The creation procedure is made of two steps.

First right-click **Generic Schema** in the **Repository** tree view and select **Create generic schema**.

9.13.1. Step 1: General properties

A connection wizard opens up. Fill in the generic Schema properties such as Schema **Name** and **Description**. The **Status** field is a customized field you can define in **Window > Preferences**.

Click **Next** when completed.

9.13.2. Step 2: Schema definition


There is no default schema displaying as there is no source to take it from.

- You can give a name to the schema or use the default name (metadata) and add a comment if you feel like.
- Customize the schema structure in the schema panel, based on your needs.
- The tool bar allows you to add, remove or move columns in your schema. Also, you can load an xml schema or export the current schema as xml.
- Click **Finish** to complete the generic schema creation. The created schema is displayed under the relevant **Generic Schemas** connection node.

9.14. Setting up an MDM connection

Talend Open Studio for ESB enables you to centralized the details of one or more MDM connections under the **Metadata** folder in the **Repository** tree view. You can then use any of these established connections to connect to the MDM server.



You can also set up an MDM connection the same way by clicking the  icon in the **Basic settings** view of the **tMDMInput** and **tMDMOutput** components. For more information, see the **Talend MDM** component chapter in the *Talend Open Studio Components Reference Guide*.

According to the option you select, the wizard helps you create an input XML, an output XML or a receive XML schema. Later, in a **Talend Job**, the **tMDMInput** component uses the defined input schema to read master data stored in XML documents, **tMDMOutput** uses the defined output schema to either write master data in an XML document on the MDM server, or to update existing XML documents and finally the **tMDMReceive** component uses the defined XML schema to receive an MDM record in XML from MDM triggers and processes.

9.14.1. Step 1: Setting up the connection

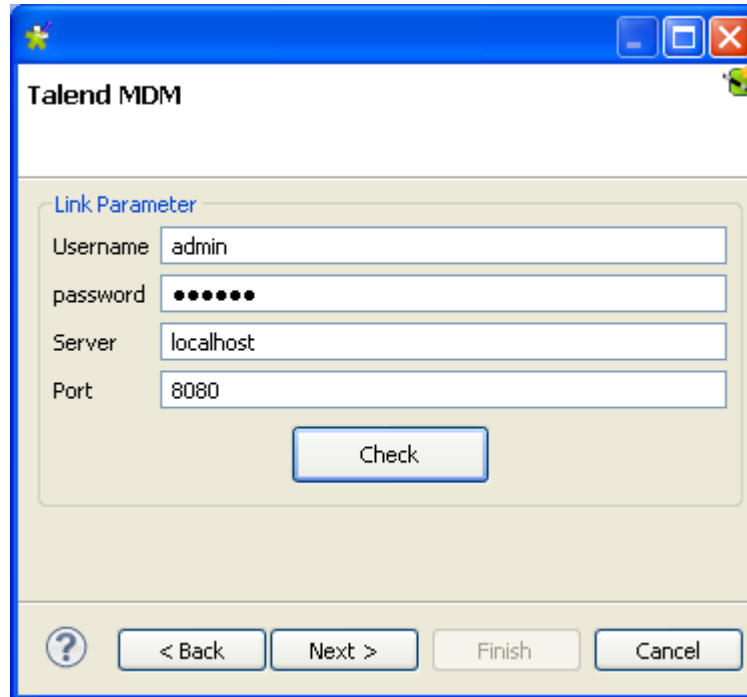
To establish an MDM connection, complete the following:

1. In the **Repository** tree view, expand **Metadata** and right-click **Talend MDM**.
2. Select **Create MDM** on the contextual menu.

The connection wizard displays.

3. Fill in the connection properties such as **Name**, **Purpose** and **Description**. The **Status** field is a customized field that can be defined. For more information, see [Section 2.6.8, “Status settings”](#).

4. Click **Next** to proceed to the next step.

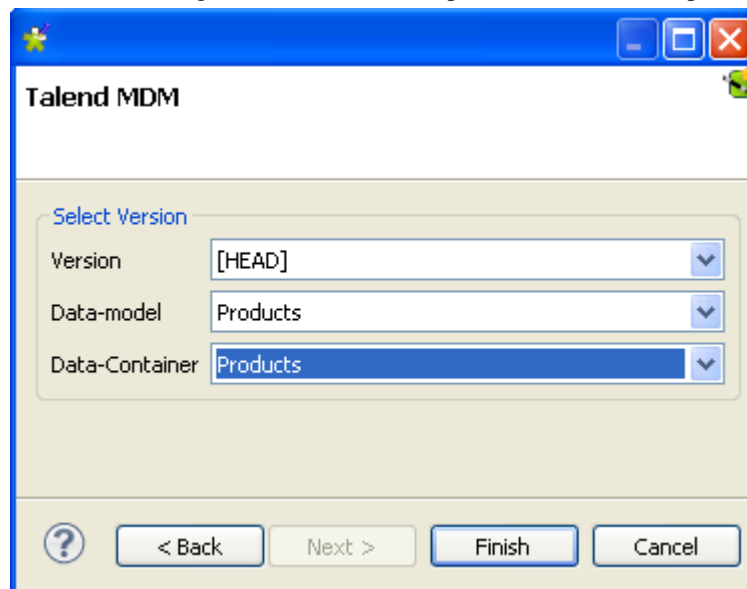


The image shows a 'Talend MDM' dialog box with a 'Link Parameter' section. It contains four text input fields: 'Username' with 'admin', 'password' with masked characters, 'Server' with 'localhost', and 'Port' with '8080'. Below these fields is a 'Check' button. At the bottom of the dialog are four buttons: a help icon (?), '< Back', 'Next >', and 'Finish'. The 'Next >' button is highlighted.

5. Fill in the connection details including the authentication information to the MDM server and then click **Check** to check the connection you have created.

A dialog box displays to confirm if your connection is successful.

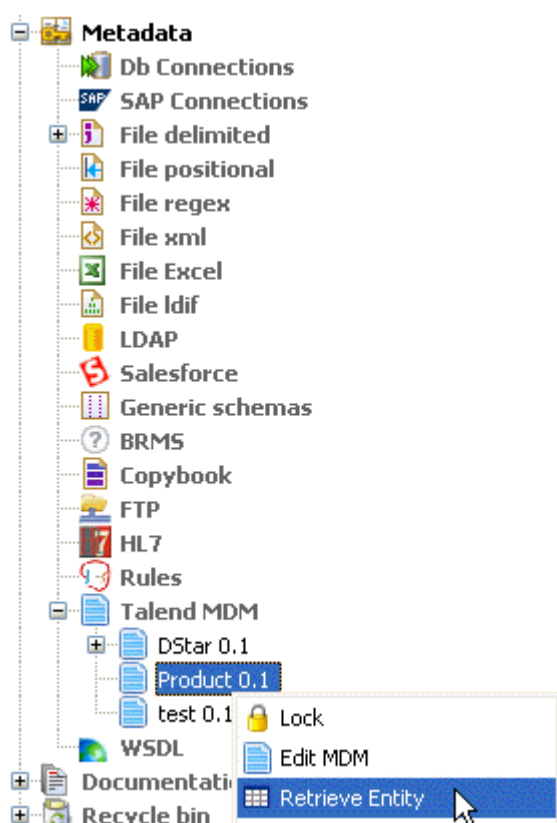
6. Click **OK** to close the confirm dialog box and then **Next** to proceed to the next step.



The image shows a 'Talend MDM' dialog box with a 'Select Version' section. It contains three dropdown menus: 'Version' with '[HEAD]', 'Data-model' with 'Products', and 'Data-Container' with 'Products'. At the bottom of the dialog are four buttons: a help icon (?), '< Back', 'Next >', and 'Finish'. The 'Finish' button is highlighted.

7. From the **Version** list, select the master data version on the MDM server to which you want to connect.
8. From the **Data-Model** list, select the data model against which the master data is validated.
9. From the **Data-Container** list, select the data container that holds the master data you want to access.
10. Click **Finish** to validate your changes and close the dialog box.

The newly created connection is listed under **Talend MDM** under the **Metadata** folder in the **Repository** tree view.



You need now to retrieve the XML schema of the business entities linked to this MDM connection.

9.14.2. Step 2: Defining MDM schema

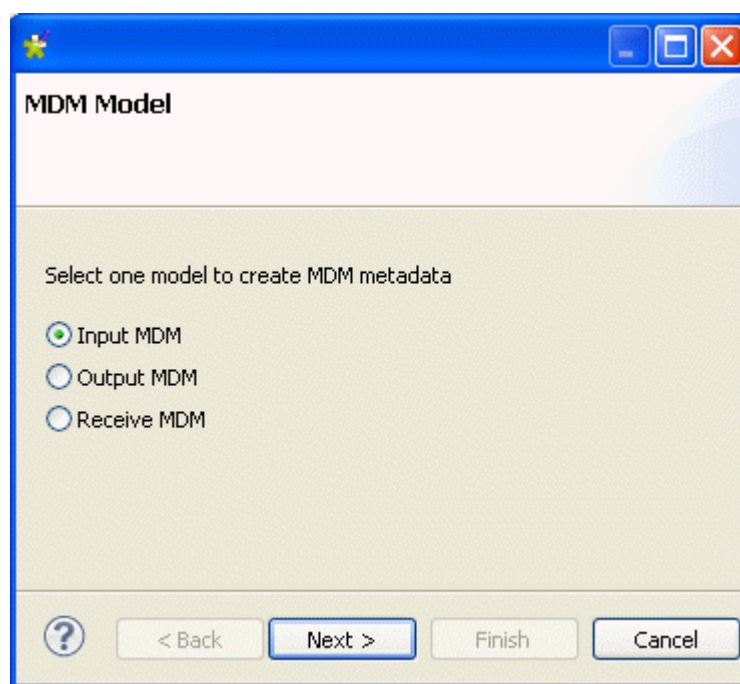
9.14.2.1. Defining Input MDM schema

This section describes how to define and download an input MDM XML schema. To define and download an output MDM XML schema, see [Section 9.14.2.2, “Defining output MDM schema”](#).

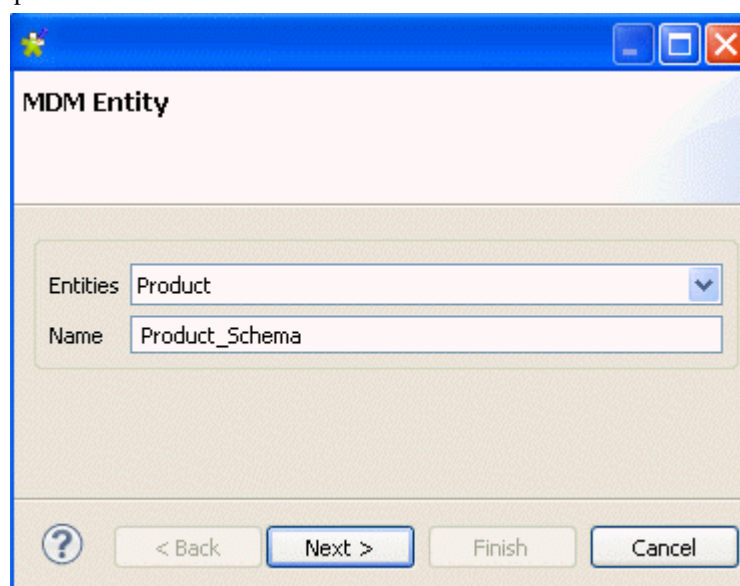
To set the values to be fetched from one or more entities linked to a specific MDM connection, complete the following:

1. In the **Repository** tree view, expand **Metadata** and right-click the MDM connection for which you want to retrieve the entity values.
2. Select **Retrieve Entity** from the contextual menu.

A dialog box displays.



3. Select the **Input MDM** option in order to download an input XML schema and then click **Next** to proceed to the following step.



4. From the **Entities** field, select the business entity (XML schema) from which you want to retrieve values.

The name displays automatically in the **Name** field.



You are free to enter any text in this field, although you would likely put the name of the entity from which you are retrieving the schema.

5. Click **Next** to proceed to the next step.



The schema of the entity you selected automatically display in the **Source Schema** panel.

Here, you can set the parameters to be taken into account for the XML schema definition.

The schema dialog box is divided into four different panels as the following:

Panel	Description
Source Schema	Tree view of the uploaded entity.
Target schema	Extraction and iteration information.
Preview	Target schema preview.
File viewer	Raw data viewer.

- In the **Xpath loop expression** area, enter the absolute XPath expression leading to the XML structure node on which to apply the iteration. Or, drop the node from the source schema to the target schema Xpath field. This link is orange in color.



The **Xpath loop expression** field is compulsory.

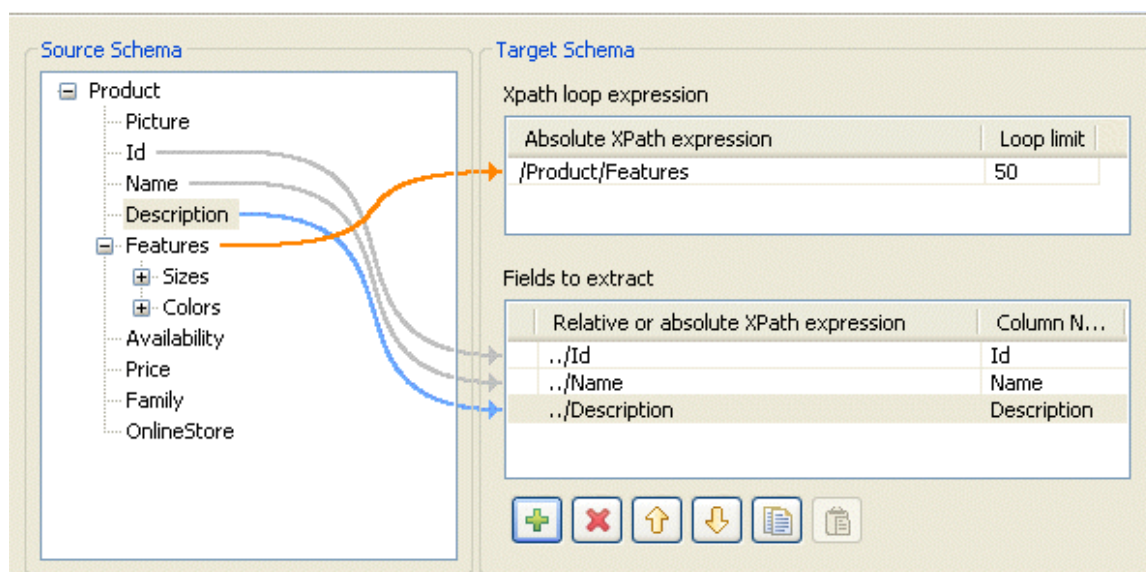
- If required, define a **Loop limit** to restrict the iteration to a number of nodes.

In the capture above, we use *Features* as the element to loop on because it is repeated within the *Product* entity as follows:

```
<Product>
  <Id>1</Id>
  <Name>Cup</Name>
  <Description/>
  <Features>
    <Feature>Color red</Feature>
    <Feature>Size maxi</Feature>
  </Features>
  ...
</Product>
<Product>
  <Id>2</Id>
  <Name>Cup</Name>
  <Description/>
  <Features>
    <Feature>Color blue</Feature>
    <Feature>Thermos</Feature>
  </Features>
  ...
</Product>
```

By doing so, the **tMDMInput** component that uses this MDM connection will create a new row for every item with different feature.

- To define the fields to extract, drop the relevant node from the source schema to the **Relative or absolute XPath expression** field.



Use the **[+]** sign to add rows to the table and select as many fields to extract as necessary. Press the **Ctrl** or the **Shift** keys for multiple selection of grouped or separate nodes and drop them to the table.

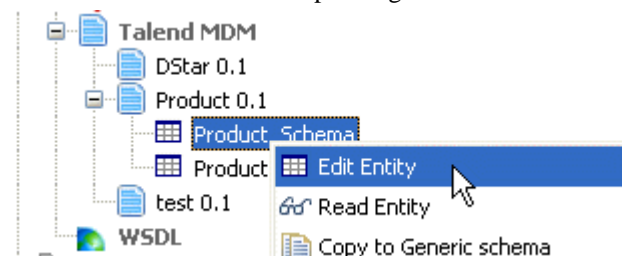
9. If required, enter a name to each of the retrieved columns in the **Column name** field.



You can prioritize the order of the fields to extract by selecting the field and using the up and down arrows. The link of the selected field is blue, and all other links are grey.

10. Click **Finish** to validate your modifications and close the dialog box.

The newly created schema is listed under the corresponding MDM connection in the **Repository** tree view.



To modify the created schema, complete the following:

1. In the **Repository** tree view, expand **Metadata** and **Talend MDM** and then browse to the schema you want to modify.
2. Right-click the schema name and select **Edit Entity** from the contextual menu.

A dialog box displays.

Name:

Comment:

Schema

Click Guess button to update the schema below according to your settings

Description of the Schema

Column	Key	Type	<input checked="" type="checkbox"/>	N..	Date Pattern (Ctrl...	Length	Precision	Default	Commer
Id	<input checked="" type="checkbox"/>	String	<input checked="" type="checkbox"/>				0		
Name	<input type="checkbox"/>	String	<input checked="" type="checkbox"/>				0		
Description	<input type="checkbox"/>	String	<input checked="" type="checkbox"/>				0		

Below the table are icons for adding (+), removing (X), moving up (↑), moving down (↓), and other schema management actions.

3. Modify the schema as needed.

You can change the name of the schema according to your needs, you can also customize the schema structure in the schema panel. The tool bar allows you to add, remove or move columns in your schema.

4. Click **Finish** to close the dialog box.

The MDM input connection (**tMDMInput**) is now ready to be dropped in any of your Jobs.

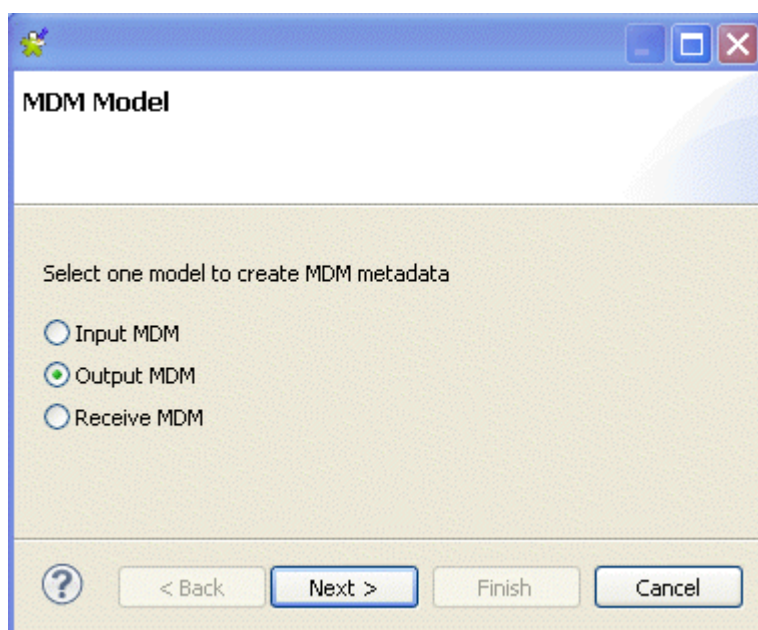
9.14.2.2. Defining output MDM schema

This section describes how to define and download an output MDM XML schema. To define and download an input MDM XML schema, see [Section 9.14.1, “Step 1: Setting up the connection”](#).

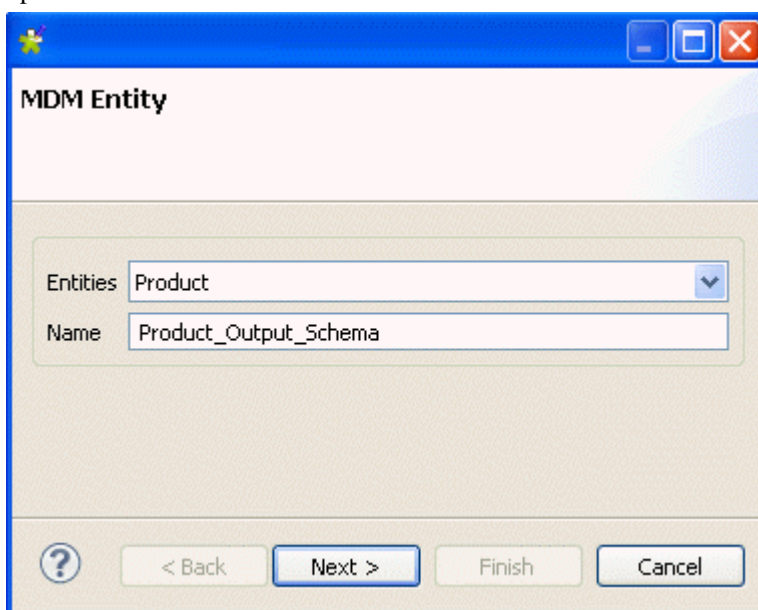
To set the values to be written in one or more entities linked to a specific MDM connection, complete the following:

1. In the **Repository** tree view, expand **Metadata** and right-click the MDM connection for which you want to write the entity values.
2. Select **Retrieve Entity** from the contextual menu.

A dialog box displays.



3. Select the **Output MDM** option in order to define an output XML schema and then click **Next** to proceed to the following step.



4. From the **Entities** field, select the business entity (XML schema) in which you want to write values.

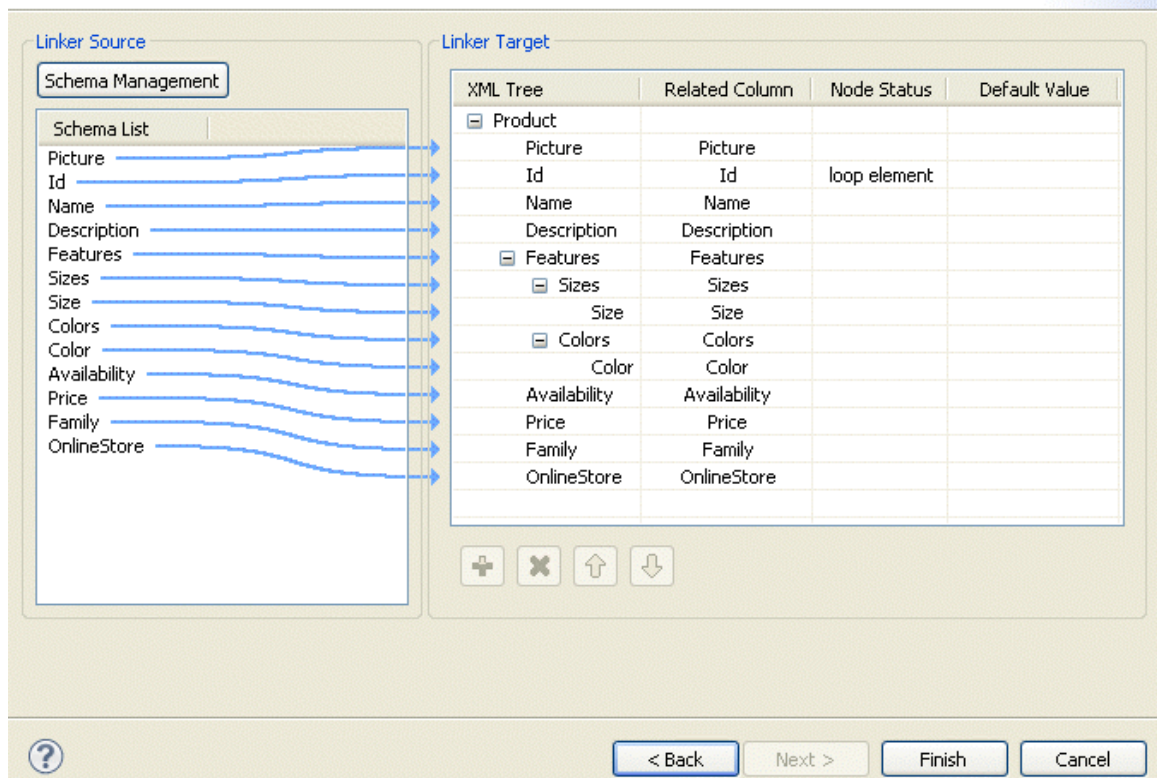
The name displays automatically in the **Name** field.



You are free to enter any text in this field, although you would likely put the name of the entity from which you are retrieving the schema.

5. Click **Next** to proceed to the next step.

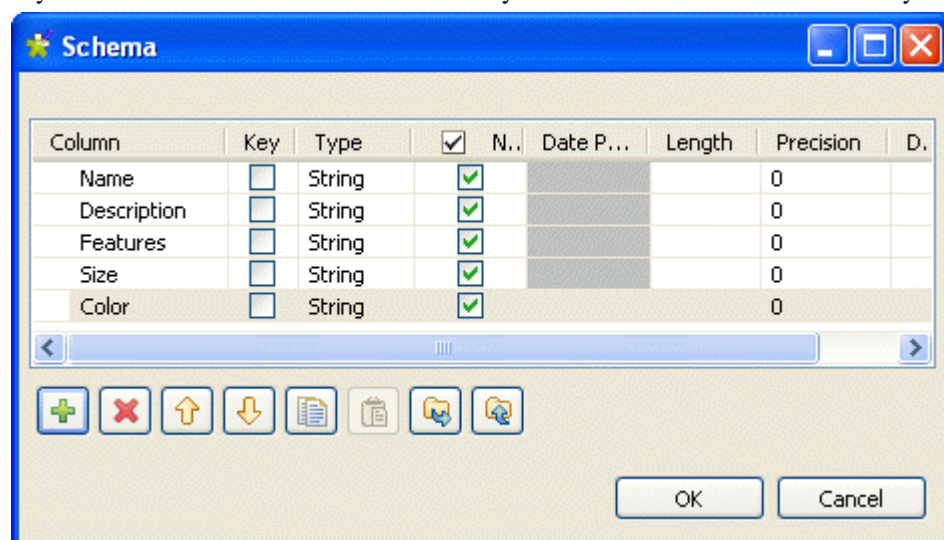
MDM Entity



Identical schema of the entity you selected is automatically created in the **Linker Target** panel, and columns are automatically mapped from the source to the target panels. The wizard automatically defines the item Id as the looping element. You can always select to loop on another element.

Here, you can set the parameters to be taken into account for the XML schema definition.

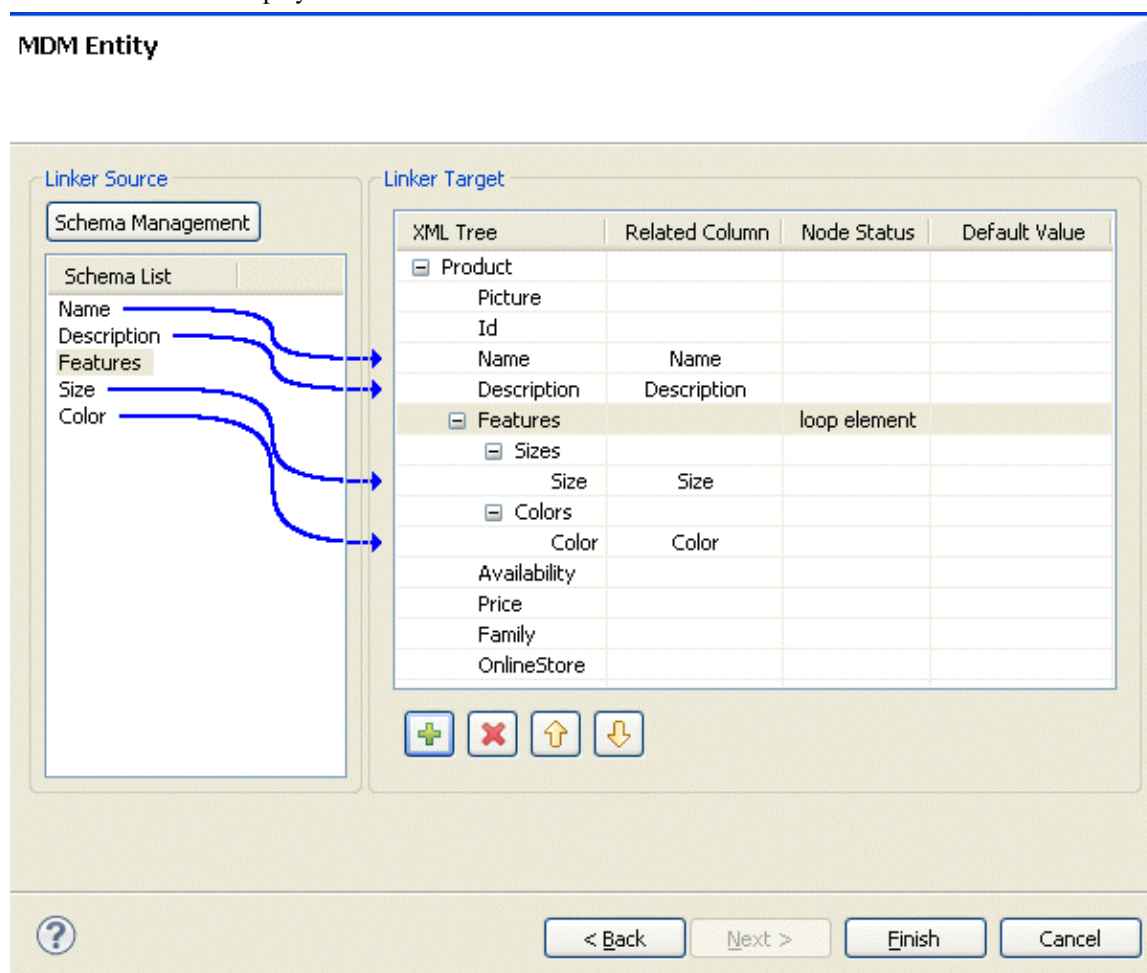
- Click **Schema Management** to display a dialog box.
- Do necessary modifications to define the XML schema you want to write in the selected entity.



Your **Linker Source** schema must correspond to the **Linker Target** schema, i.e. define the elements in which you want to write values.

- Click **OK** to close the dialog box.

The defined schema displays under **Schema list**.



- In the **Linker Target** panel, right-click the element you want to define as a loop element and select **Set as loop element**. This will restrict the iteration to one or more nodes.

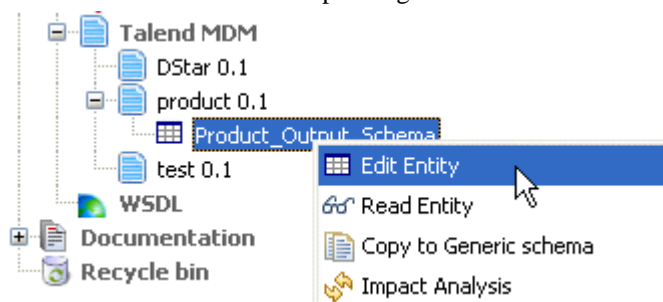
By doing so, the **tMDMOutput** component that uses this MDM connection will create a new row for every item with different feature.



You can prioritize the order of the fields to write by selecting the field and using the up and down arrows.

- Click **Finish** to validate your modifications and close the dialog box.

The newly created schema is listed under the corresponding MDM connection in the **Repository** tree view.



To modify the created schema, complete the following:

1. In the **Repository** tree view, expand **Metadata** and **Talend MDM** and then browse to the schema you want to modify.
2. Right-click the schema name and select **Edit Entity** from the contextual menu.

A dialog box displays.

Name:

Comment:

Schema

Click Guess button to update the schema below according to your settings

Description of the Schema

Column	Key	Type	Null...	Date Pattern (Ctr...	Length	Precision	Default	Comment
Name	<input checked="" type="checkbox"/>	String	<input checked="" type="checkbox"/>			0		
Description	<input type="checkbox"/>	String	<input checked="" type="checkbox"/>			0		
Features	<input type="checkbox"/>	String	<input checked="" type="checkbox"/>			0		
Size	<input type="checkbox"/>	String	<input checked="" type="checkbox"/>			0		
Color	<input type="checkbox"/>	String	<input checked="" type="checkbox"/>			0		

3. Modify the schema as needed.

You can change the name of the schema according to your needs, you can also customize the schema structure in the schema panel. The tool bar allows you to add, remove or move columns in your schema.

4. Click **Finish** to close the dialog box.

The MDM output connection (**tMDMOutput**) is now ready to be dropped in any of your Jobs.

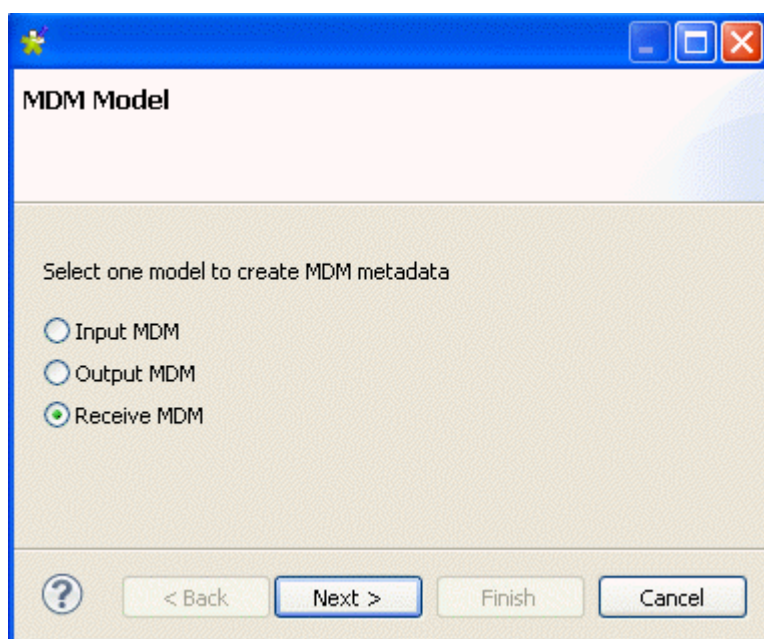
9.14.2.3. Defining Receive MDM schema

This section describes how to define a receive MDM XML schema based on the MDM connection.

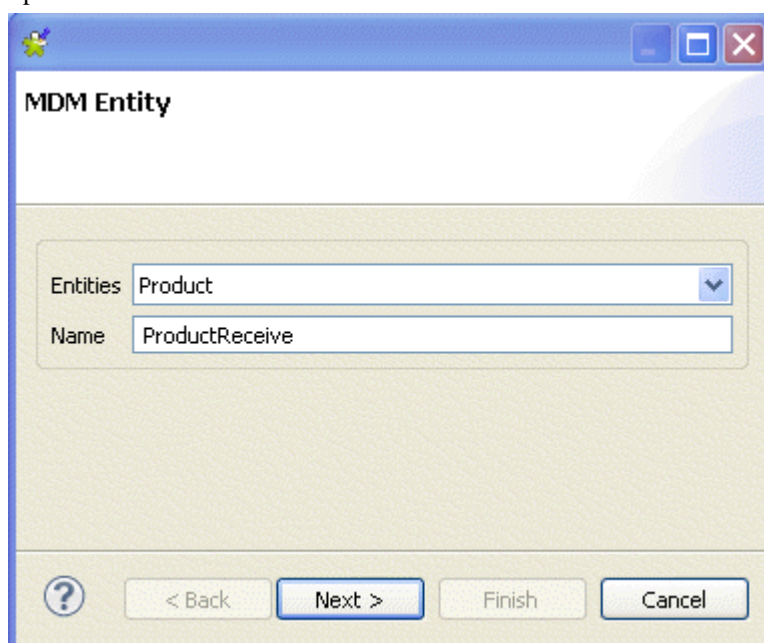
To set the XML schema you want to receive in accordance with a specific MDM connection, complete the following:

1. In the **Repository** tree view, expand **Metadata** and right-click the MDM connection for which you want to retrieve the entity values.
2. Select **Retrieve Entity** from the contextual menu.

A dialog box displays.



3. Select the **Receive MDM** option in order to define a receive XML schema and then click **Next** to proceed to the following step.



4. From the **Entities** field, select the business entity (XML schema) according to which you want to receive the XML schema.

The name displays automatically in the **Name** field.



You can enter any text in this field, although you would likely put the name of the entity according to which you want to receive the XML schema.

5. Click **Next** to proceed to the next step.

MDM Entity

Source Schema

- Product
 - Picture
 - Id
 - Name
 - Description
 - Features
 - Sizes
 - Colors
 - Availability
 - Price
 - Family
 - OnlineStore

Target Schema

Xpath loop expression

Absolute XPath expression Loop limit

Fields to extract

Relative or absolute XPath expression Column Name

Preview Output

No preview available for XSD file

File Viewer

```

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
<xsd:import namespace="http://www.w3.org/2001/XMLSchema"
<xsd:element name="Product">
  <xsd:annotation>
    <xsd:appinfo source="X_Label_EN">Product</xsd:appinfo>
    <xsd:appinfo source="X_Label_FR">Produit</xsd:appinfo>
  </xsd:annotation>
  <xsd:complexType>
    <xsd:all maxOccurs="1" minOccurs="1">
      <xsd:element maxOccurs="1" minOccurs="0" name="Picture">
        <xsd:annotation>
          <xsd:appinfo source="X_Label_EN">Picture</xsd:appinfo>
        </xsd:annotation>
      </xsd:element>
    </xsd:all>
  </xsd:complexType>
</xsd:element>

```

Export as Context Revert from Context

< Back Next > Finish Cancel



The schema of the entity you selected display in the **Source Schema** panel.

Here, you can set the parameters to be taken into account for the XML schema definition.

The schema dialog box is divided into four different panels as the following:

Panel	Description
Source Schema	Tree view of the uploaded entity.
Target schema	Extraction and iteration information.
Preview	Target schema preview.
File viewer	Raw data viewer.

- In the **Xpath loop expression** area, enter the absolute XPath expression leading to the XML structure node on which to apply the iteration. Or, drop the node from the source schema to the target schema Xpath field. This link is orange in color.



The **Xpath loop expression** field is compulsory.

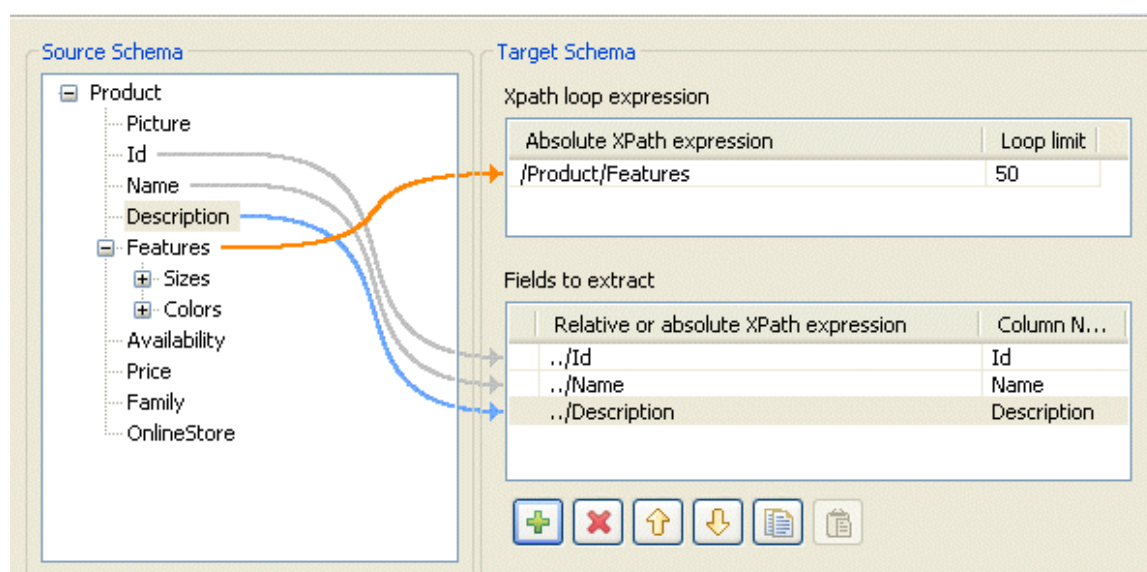
- If required, define a **Loop limit** to restrict the iteration to one or more nodes.

In the above capture, we use *Features* as the element to loop on because it is repeated within the *Product* entity as the following:

```
<Product>
  <Id>1</Id>
  <Name>Cup</Name>
  <Description/>
  <Features>
    <Feature>Color red</Feature>
    <Feature>Size maxi</Feature>
  </Features>
  ...
</Product>
<Product>
  <Id>2</Id>
  <Name>Cup</Name>
  <Description/>
  <Features>
    <Feature>Color blue</Feature>
    <Feature>Thermos</Feature>
  </Features>
  ...
</Product>
```

By doing so, the **tMDMReceive** component that uses this MDM connection will create a new row for every item with different feature.

- To define the fields to receive, drop the relevant node from the source schema to the **Relative or absolute XPath expression** field.



Use the plus sign to add rows to the table and select as many fields to extract as necessary. Press the **Ctrl** or the **Shift** keys for multiple selection of grouped or separate nodes and drop them to the table.

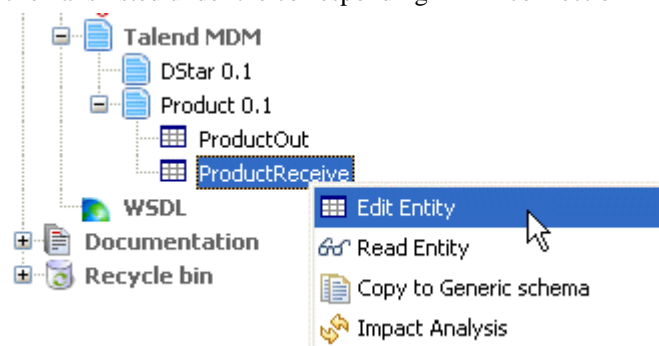
9. If required, enter a name to each of the received columns in the **Column name** field.



You can prioritize the order of the fields you want to receive by selecting the field and using the up and down arrows. The link of the selected field is blue, and all other links are grey.

10. Click **Finish** to validate your modifications and close the dialog box.

The newly created schema is listed under the corresponding MDM connection in the **Repository** tree view.



To modify the created schema, complete the following:

1. In the **Repository** tree view, expand **Metadata** and **Talend MDM** and then browse to the schema you want to modify.
2. Right-click the schema name and select **Edit Entity** from the contextual menu.

A dialog box displays.

Name: ProductReceive

Comment:

Schema

Click Guess button to update the schema below according to your settings

Guess

Description of the Schema

Column	Key	Type		N..	Date Pattern (Ctrl...	Length	Precision	Default	Comm
Id	<input checked="" type="checkbox"/>	String	<input checked="" type="checkbox"/>				0		
Name	<input type="checkbox"/>	String	<input checked="" type="checkbox"/>				0		
Description	<input type="checkbox"/>	String	<input checked="" type="checkbox"/>				0		

Toolbar: +, -, up, down, list, clipboard, link, unlink

Finish Cancel

3. Modify the schema as needed.

You can change the name of the schema according to your needs, you can also customize the schema structure in the schema panel. The tool bar allows you to add, remove or move columns in your schema.

4. Click **Finish** to close the dialog box.

The MDM receive connection (**tMDMReceive**) is now ready to be dropped in any of your Jobs.

9.15. Setting up a Web Service schema

In *Talend Open Studio for ESB* you can save your Web Service connections in the **Repository**.

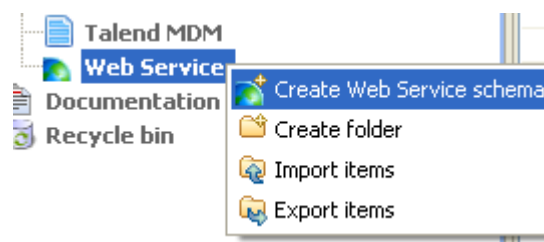
9.15.1. Setting up a simple schema

This section describes how to define a Web Service schema.

9.15.1.1. Step 1: General properties

This step illustrates how to define the file's general properties.

1. In the **Repository**, expand the **Metadata** node.
2. Right-click **Web Service** and select **Create Web Service schema** from the context menu list.



3. Enter the generic schema information such as its **Name** and **Description**.

 A screenshot of the 'Create new Web Service schema' dialog box, titled 'File - Step 1 of 4'. The dialog box contains the following fields and controls:

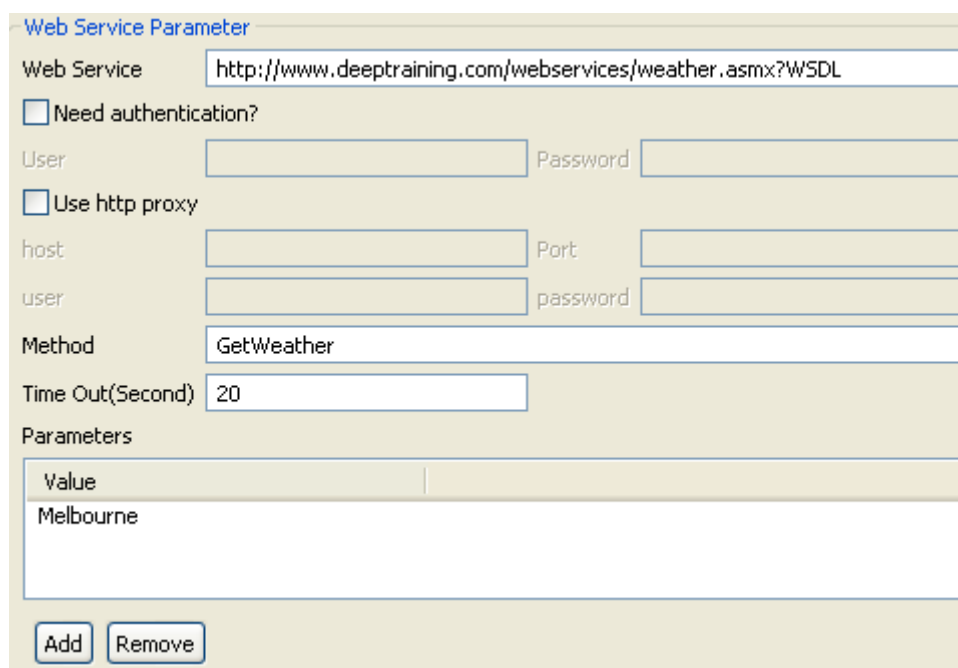
- Name:** A text field containing 'SimpleGetWeather'.
- Purpose:** An empty text field.
- Description:** A large text area with a vertical scrollbar.
- Author:** A text field containing 'user@company.com'.
- Locker:** An empty text field.
- Version:** A text field containing '0.1', followed by two buttons labeled 'M' and 'm'.
- Status:** A dropdown menu with a downward arrow.
- Path:** A text field followed by a 'Select' button.

 At the bottom of the dialog box, there are four buttons: a help button (question mark icon), '< Back', 'Next >', 'Finish', and 'Cancel'.

4. Click **Next** to select the schema type in step 2.

9.15.1.2. Step 2: URI and method definition.

This step involves the definition of the URI and other parameters required to obtain the desired values.



Web Service Parameter

Web Service:

☐ Need authentication?

User: Password:

☐ Use http proxy

host: Port:

user: password:

Method:

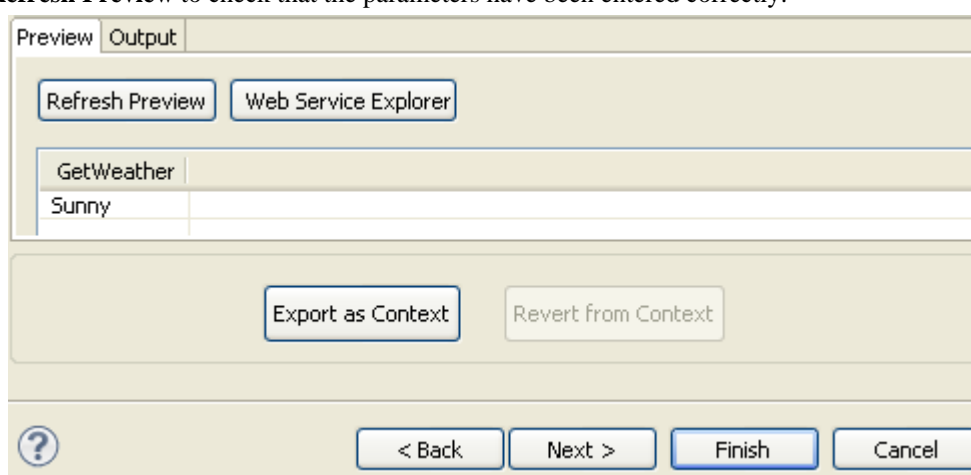
Time Out(Second):

Parameters

Value
Melbourne

In the **Web Service Parameter** zone:

1. Enter the URI which will transmit the desired values, in the **Web Service** field.
2. If necessary, select the **Need authentication?** check box and then enter your authentication information in the **User** and **Password** fields.
3. If you use an http proxy, select the **Use http proxy** check box and enter the information required in the **host**, **Port**, **user** and **password** fields.
4. Enter the **Method** name in the corresponding field.
5. In the **Value** table, **Add** or **Remove** values as desired, using the corresponding buttons.
6. Click **Refresh Preview** to check that the parameters have been entered correctly.



Preview **Output**

GetWeather
Sunny

In the **Preview** tab, the values to be transmitted by the Web Service method are displayed, based the parameters entered.

9.15.1.3. Step 3: Finalizing the end schema

You can modify the schema name (*metadata*, by default) and modify the schema itself using the tool bar.

Add a File metadata on repository. Define the setting of the parsed job.

Name: metadata

Comment:

Schema

Click to update schema preview

Description of the Schema

Column	Key	Type	Nullable	Date Pattern ...	Length	Precision	Def...	Comment
GetWeather	<input type="checkbox"/>	String	<input checked="" type="checkbox"/>		5	0		

Buttons:

Navigation:

1. Add or delete columns using the and buttons.
2. Modify the order of the columns using the and buttons.
3. Click **Finish**.

The new schema is added to the **Repository** under the **Web Service** node.

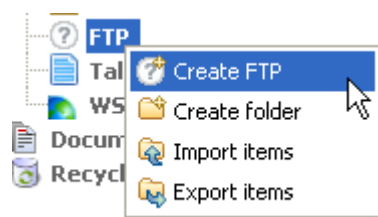
9.16. Setting up an FTP connection

If you need to connect to an FTP server regularly, you can centralize the connection information under the **Metadata** node in the **Repository** view.

9.16.1. Step 1: General properties

To create a connection to an FTP server, follow the below steps:

1. Expand the **Metadata** node in the **Repository** tree view.



2. Right-click **FTP** and select **Create FTP** from the context menu.

The connection wizard opens:

3. Enter the generic schema information such as its **Name** and **Description**.



The status field is a customized field which can be defined in the Preferences dialog box (**Window > Preferences**). For further information about setting preferences, see [Section 2.5, “Setting Talend Open Studio for ESB preferences”](#).

4. When you have finished, click **Next** to enter the FTP server connection information.

9.16.2. Step 2: Connection

In this step we shall define the connection information and parameters.

1. Enter your **Username** and **Password** in the corresponding fields.

2. In the **Host** field, enter the name of your FTP server host.
3. Enter the **Port** number in the corresponding field.
4. Select the **Encoding** type from the list.
5. From the **Connection Model** list, select the connection model you want to use:
 - Select **Passive** if you want the FTP server to choose the port connection to be used for data transfer.
 - Select **Active** if you want to choose the port yourself.
6. In the **Parameter** area, select a setting for FTP server usage. For standard usage, there is no need to select an option.
 - Select the **SFTP Support** check box to use the SSH security protocol to protect server communications.
An **Authentication method** appears. Select **Public key** or **Password** according to what you use.
 - Select the **FTPs Support** check box to protect server communication with the SSL security protocol.
 - Select the **Use Socks Proxy** check box if you want to use this option, then enter the proxy information (the host name, port number, username and password).
7. Click **Finish** to close the wizard.

All of the connections created appear under the FTP server connection node, in the **Repository** view.

You can drop the connection metadata from the **Repository** onto the design workspace. A dialog box opens in which you can choose the component to be used in your Job.

For further information about how to drop metadata onto the workspace, see [Section 4.2.2.2, “How to drop components from the Metadata node”](#).

9.17. Exporting Metadata as context

For every Metadata connection (either File or Database), you can export the connection details as a Context.

1. On the last step or second last step of the wizard, click **Export as Context**.
2. The Context creation 2-step wizard launches.
3. On Step 1, fill out a name for the Context. By default the name of the Metadata entry is proposed.
4. Click **Next**.
5. On Step 2, the automatically created context variables are displaying on the three tab table. Check out that the variable values showing on the Values tab are correctly set.
6. Click **Finish** to validate the creation.



Chapter 10. Managing routines

This chapter defines the routines, along with user scenarios, and explains how to create your own routines or how to customize the system routines. The chapter also gives an overview of the main routines and use cases of them. To have an overview of the most commonly used routines and other use cases, see [Appendix D, *System routines*](#).

Before starting any data integration processes, you need to be familiar with *Talend Open Studio for ESB* Graphical User Interface (GUI). For more information, see [Appendix A, *GUI*](#).

10.1. What are routines

Routines are fairly complex Java functions, generally used to factorize code. They therefore optimize data processing and improve Job capacities.

You can also use the **Repository** tree view to store frequently used parts of code or extract parts of existing company functions, by calling them via the routines. This factorization makes it easier to resolve any problems which may arise and allows you to update the code used in multiple Jobs quickly and easily.

On top of this, certain system routines adopt the most common Java methods, using the **Talend** syntax. This allows you to escalate Java errors in the studio directly, thereby facilitating the identification and resolution of problems which may arise as your integration processes evolve with **Talend**.

There are two types of routines:

- System routines: a number of system routines are provided. They are classed according to the type of data which they process: numerical, string, date...
- User routines: these are routines which you have created or adapted from existing routines.



You do not need any knowledge of the Java language to create and use **Talend** routines.

All of the routines are stored under **Code > Routines** in the **Repository** tree view.

For further information concerning the system routines, see [Section 10.2, “Accessing the System Routines”](#).

For further information about how to create user routines, see [Section 10.4.1, “How to create user routines”](#).



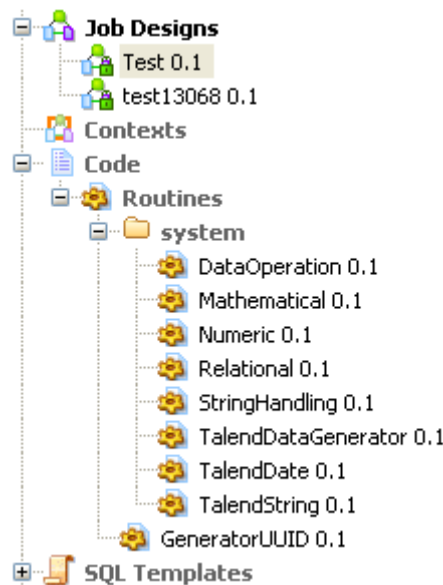
You can also set up routine dependencies on Jobs. To do so, simply right click a Job on the **Repository** tree view and select **Set up routine dependencies**. In the dialog box which opens, all routines are set by default. You can use the tool bar to remove routines if required.

10.2. Accessing the System Routines

To access the system routines, click **Code > Routines > system**. The routines or functions are classed according to their usage.



The **system** folder and its content are read only.



Each class or category in the system folder contains several routines or functions. Double-click the class that you want to open.

All of the routines or functions within a class are composed of some descriptive text, followed by the corresponding Java code. In the Routines view, you can use the scrollbar to browse the different routines. Or alternatively:

1. Press **Ctrl+O** in the routines view.

A dialog box displays a list of the different routines in the category.

2. Click the routine of interest.

The view jumps to the section comprising the routine's descriptive text and corresponding code.



The syntax of routine call statements is case sensitive.

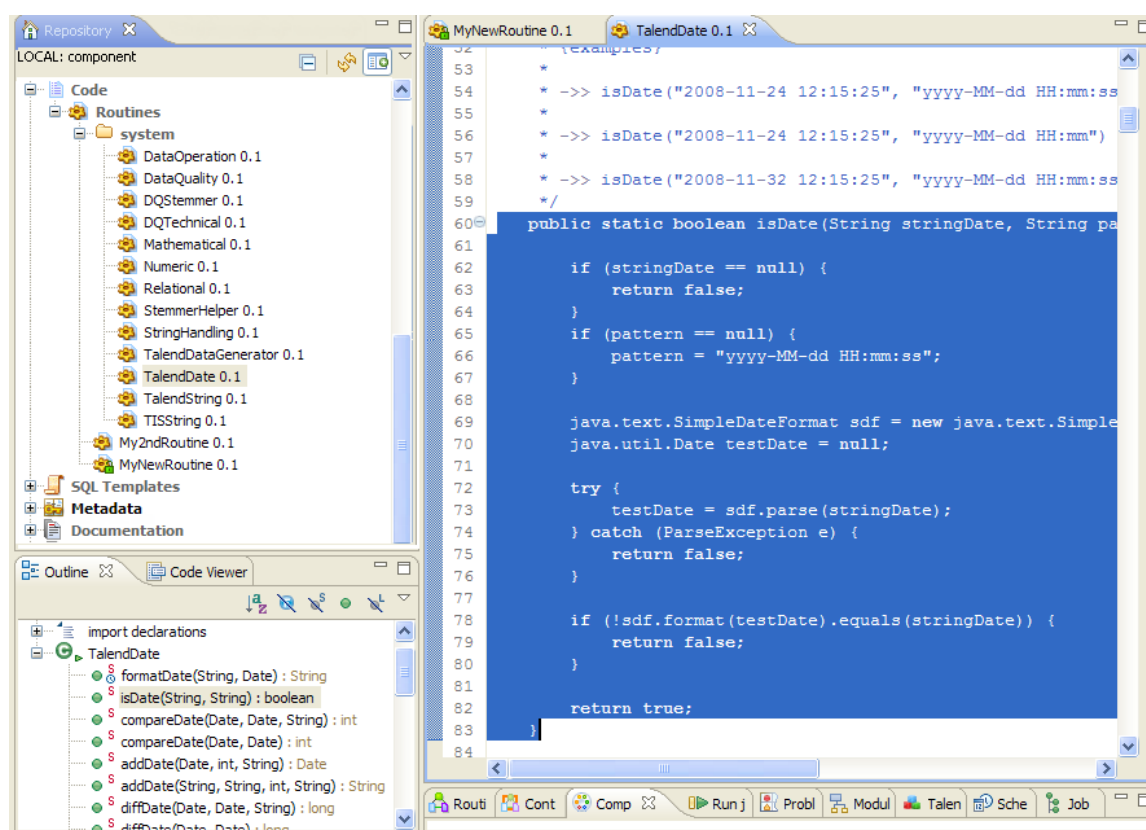
For more information about the most common Java routines, see [Appendix D, System routines](#).

10.3. Customizing the system routines

If the system routines are not adapted to your specific needs, you can customize them by copying and pasting the content in a user routine, then modify the content accordingly.

To customize a system routine:

1. First of all, create a user routine by following the steps outlined in the [Section 10.4.1, “How to create user routines”](#). The routine opens in the workspace, where you shall find a basic example of a routine.
2. Then, under **Code > Routines > system**, select the class of routines which contains the routine(s) you want to customize.
3. Double-click the class which contains the relevant routine to open it in the workspace.
4. Use the **Outline** panel on the bottom left of the studio to locate the routine from which you want to copy all or part of the content.



5. In the workspace, select all or part of the code and copy it using **Ctrl+C**.
6. Click the tab to access your user routine and paste the code by pressing **Ctrl+V**.
7. Modify the code as required and press **Ctrl+S** to save it.

We advise you to use the descriptive text (in blue) to detail the input and output parameters. This will make your routines easier to maintain and reuse.

10.4. Managing user routines

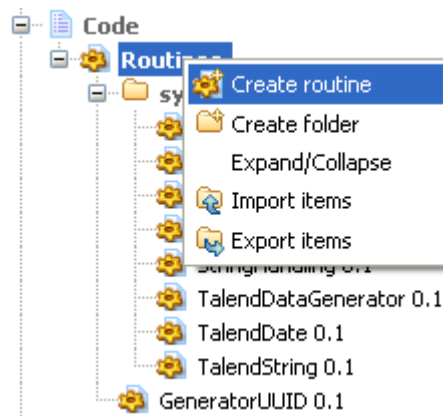
Talend Open Studio for ESB allows you to create user routines, to modify them or to modify system routines, in order to fill your specific needs.

10.4.1. How to create user routines

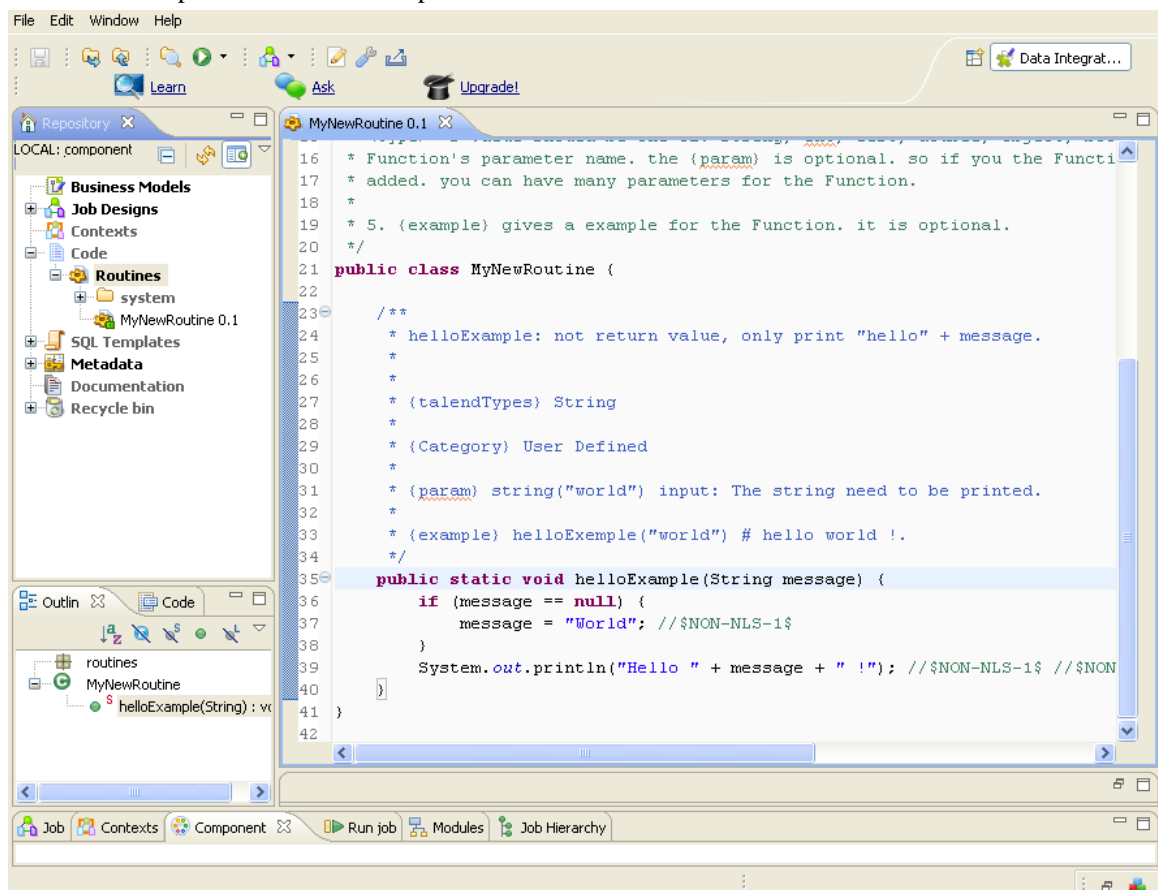
You can create your own routines according to your particular factorization needs. Like the system routines, the user routines are stored in the **Repository** tree view under **Code > Routines**. You can add folders to help organize your routines and call them easily in any of your Jobs.

To create a new user routine, complete the following:

1. In the **Repository** tree view, expand **Code** to display the **Routines** folder.



2. Right-click **Routines** and select **Create routine**.
3. The **[New routine]** dialog box opens. Enter the information required to create the routine, ie., its name, description...
4. Click **Finish** to proceed to the next step.



The newly created routine appears in the **Repository** tree view, directly below the **Routines** node. The routine editor opens to reveal a model routine which contains a simple example, by default, comprising descriptive text in blue, followed by the corresponding code.



We advise you to add a very detailed description of the routine. The description should generally include the input and output parameters you would expect to use in the routine, as well as the results returned along with an example. This information tends to be useful for collaborative work and the maintenance of the routines.

The following example of code is provided by default:

```
public static void helloExample(String message) {  
    if (message == null) {  
        message = "World"; //$NON-NLS-1$  
    }  
    System.out.println("Hello " + message + " !");  
}
```

5. Modify or replace the model with your own code and press **Ctrl+S** to save the routine. Otherwise, the routine is saved automatically when you close it.



You can copy all or part of a system routine or class and use it in a user routine by using the **Ctrl+C** and **Ctrl+V** commands, then adapt the code according to your needs. For further information about how to customize routines, see [Section 10.3, “Customizing the system routines”](#).

10.4.2. How to edit user routines

You can modify the user routines whenever you like.



The **system** folder and all of the routines held within are read only.

To edit your user routines:

1. Right click the routine you want to edit and select **Edit Routine**.
2. The routine opens in the workspace, where you can modify it.
3. Once you have adapted the routine to suit your needs, press **Ctrl+S** to save it.

If you want to reuse a system routine for your own specific needs, see [Section 10.3, “Customizing the system routines”](#).

10.4.3. How to edit user routine libraries

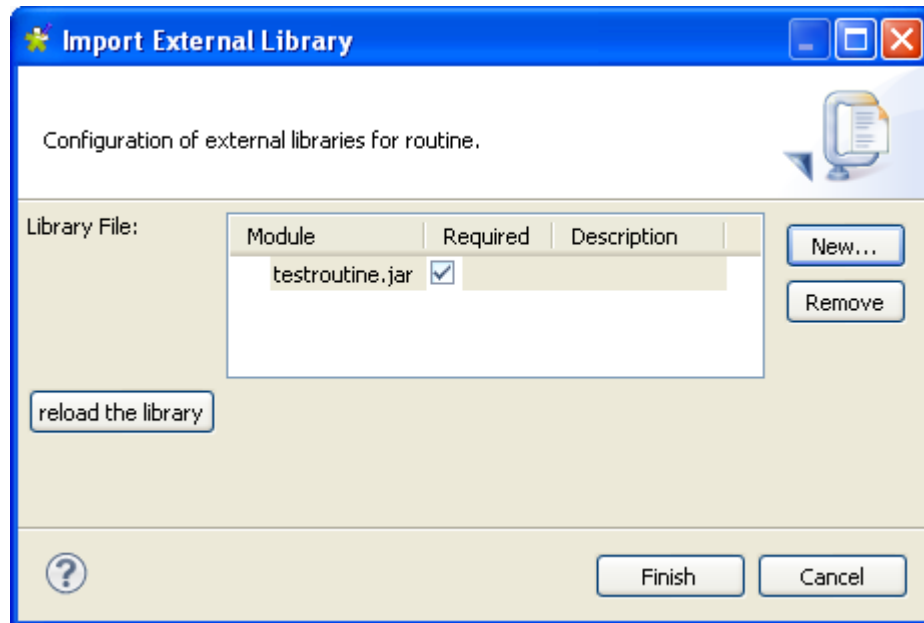
You can edit the library of any of the user routines by importing external .jar files for the selected routine. These external library files will be listed, like modules, in the **Modules** view in your current Studio. For more information on the **Modules** view, see [Section 4.5.4, “How to install external modules”](#).

The .jar file of the imported library will be also listed in the library file of your current Studio.

To edit a user routine library, complete the following:

1. In the **Repository** tree view, expand **Code > Routines**.
2. Right-click the user routine you want to edit its library and then select **Edit Routine Library**.

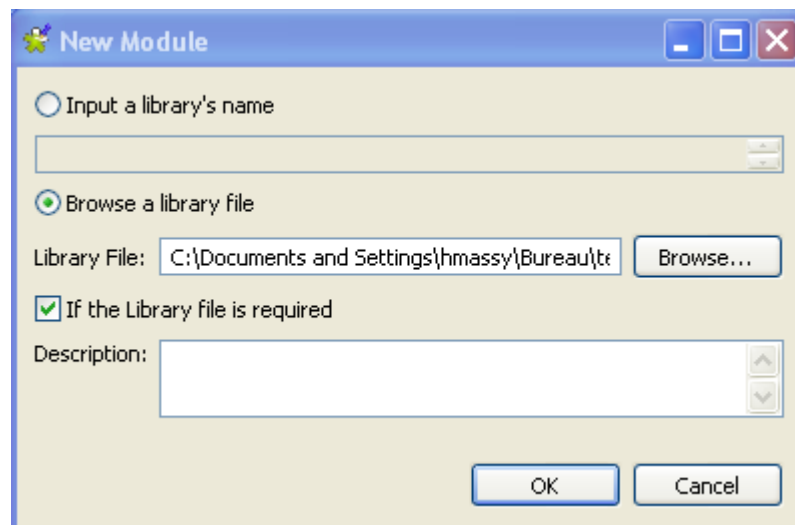
The [**Import External Library**] dialog box displays.



3. Click **New** to open a new dialog box where you can import the external library.



You can delete any of the already imported routine files if you select the file in the **Library File** list and click the **Remove** button.



4. Enter the name of the library file in the **Input a library's name** field followed by the file format (.jar), or
5. Select the **Browse a library file** option and click browse to set the file path in the corresponding field.
6. If required, enter a description in the **Description** field and then click **OK** to confirm your changes.

The imported library file is listed in the **Library File** list in the **[Import External Library]** dialog box.

7. Click **Finish** to close the dialog box.

The library file is imported into the library folder of your current Studio and also listed in the **Module** view of the same Studio.

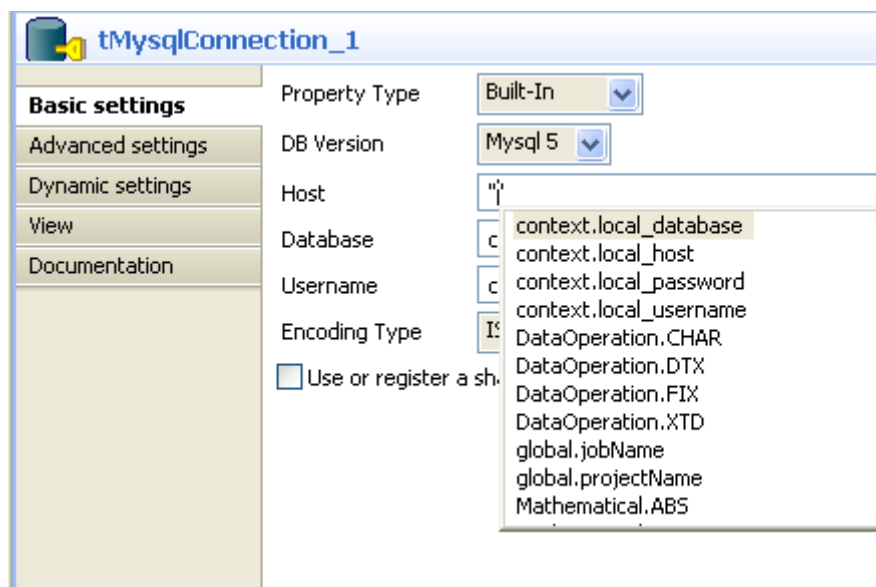
For more information about the **Modules** view, see [Section 4.5.4, "How to install external modules"](#).

10.5. Calling a routine from a Job

Pre-requisite: You must have at least one Job created, in order to run a routine. For further information regarding how to create a Job, see [Section 4.2.1, “How to create a Job”](#) of the *Talend Open Studio for ESB* User Guide.

You can call any of your user and system routines from your Job components in order to run them at the same time as your Job.

To access all the routines saved in the **Routines** folder in the **Repository** tree view, press **Ctrl+Space** in any of the fields in the **Basic settings** view of any of the **Talend** components used in your Job and select the one you want to run.



Alternatively, you can call any of these routines by indicating the relevant class name and the name of the routine, followed by the expected settings, in any of the **Basic settings** fields in the following way:

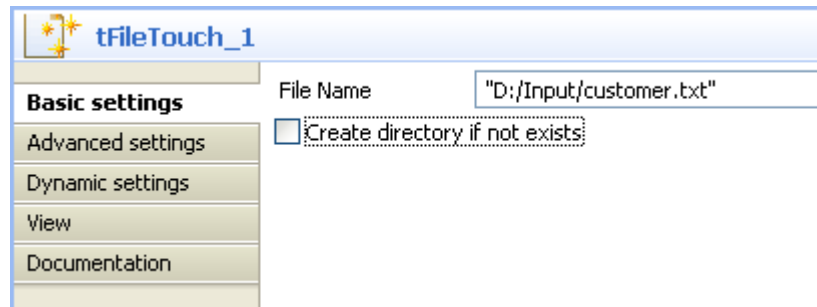
```
<ClassName>.<RoutineName>
```

10.6. Use case: Creating a file for the current date

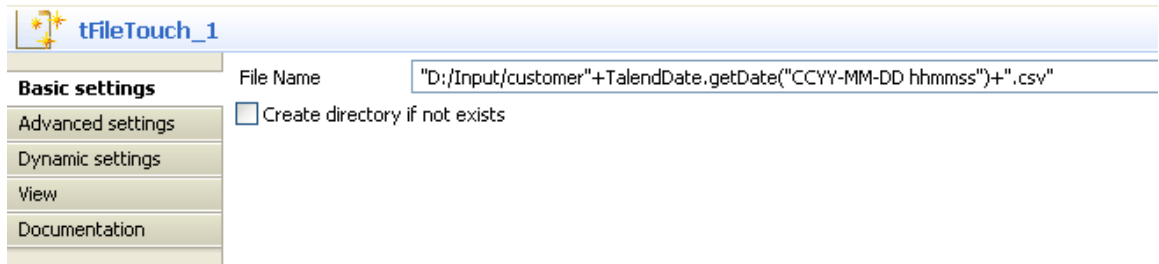
This scenario describes how to use a routine. The Job uses just one component, which calls a system routine.



1. In the **Palette**, click **File > Management**, then drop a **tFileTouch** component onto the workspace. This component allows you to create an empty file.
2. Double-click the component to open its **Basic settings** view in the **Component** tab.
3. In the **FileName** field, enter the path to access your file, or click [...] and browse the directory to locate the file.



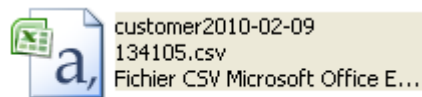
4. Close the double inverted commas around your file extension as follows: "D:/Input/customer".txt.
5. Add the plus symbol (+) between the closing inverted commas and the file extension.
6. Press **Ctrl+Space** to open a list of all of the routines, and in the auto-completion list which appears, select *TalendDate.getDate* to use the **Talend** routine which allows you to obtain the current date.
7. Modify the format of the date provided by default, if required.
8. Enter the plus symbol (+) next to the *getDate* variable to complete the routine call, and place double inverted commas around the file extension.



If you are working on windows, the ":" between the hours and minutes and between the minutes and seconds must be removed.

9. Press **F6** to run the Job.

The **tFileTouch** component creates an empty file with the days date, retrieved upon execution of the *GetDate* routine called.





Chapter 11. Using SQL templates

SQL templates are groups of pre-defined query arguments that run in the ELT mode. This chapter explains the ELT mode, defines the SQL templates and provides user scenarios to explain how to use the SQL templates or how to create your own ones.

Before starting any data integration processes, you need to be familiar with *Talend Open Studio for ESB* Graphical User Interface (GUI). For more information, see [Appendix A, GUI](#).

11.1. What is ELT

Extract, Load and Transform (ELT) is a data manipulation process in database usage, especially in data warehousing. Different from the traditional ETL (Extract, Transform, Load) mode, in ELT, data is extracted, loaded into the database and then is transformed where it sits in the database, prior to use. This data is migrated in bulk according to the data set and the transformation process occurs after the data has been loaded into the targeted DBMS in its raw format. This way, less stress is placed on the network and larger throughput is gained.

However, the ELT mode is certainly not optimal for all situations, for example,

- As SQL is less powerful than Java, the scope of available data transformations is limited.
- ELT requires users that have high proficiency in SQL tuning and DBMS tuning.
- Using ELT with *Talend Open Studio for ESB*, you cannot pass or reject one single row of data as you can do in ETL. For more information about row rejection, see [Section 4.3.1.1, “Row connection”](#).

Based on the advantages and disadvantages of ELT, the SQL templates are designed as the ELT facilitation requires.

11.2. Introducing Talend SQL templates

SQL is a standardized query language used to access and manage information in databases. Its scope includes data query and update, schema creation and modification, and data access control. *Talend Open Studio for ESB* provides a range of SQL templates to simplify the most common tasks. It also comprises a SQL editor which allows you to customize or design your own SQL templates to meet less common requirements.

These SQL templates are used with the components from the **Talend ELT** component family including **tSQLTemplate**, **tSQLTemplateFilterColumns**, **tSQLTemplateCommit**, **tSQLTemplateFilterRows**, **tSQLTemplateRollback**, **tSQLTemplateAggregate** and **tSQLTemplateMerge**. These components execute the selected SQL statements. Using the UNION, EXCEPT and INTERSECT operators, you can modify data directly on the DBMS without using the system memory.

Moreover, with the help of these SQL templates, you can optimize the efficiency of your database management system by storing and retrieving your data according to the structural requirements.

Talend Open Studio for ESB provides the following types of SQL templates under the **SQL templates** node in the **Repository** tree view:

- System SQL templates: They are classified according to the type of database for which they are tailored.
- User-defined SQL templates: these are templates which you have created or adapted from existing templates.

More detailed information about the SQL templates is presented in the below sections.

For further information concerning the components from the ELT component family, see *Talend Open Studio Components Reference Guide*.



As most of the SQL templates are tailored for specific databases, if you change database in your system, it is inevitable to switch to or develop new templates for the new database.

11.3. Managing Talend SQL templates

Talend Open Studio for ESB enables you via the **SQL Templates** folder in the **Repository** tree view to use system or user-defined SQL templates in the Jobs you create in the Studio using the ELT components.

The below sections show you how to manage these two types of SQL templates.

11.3.1. Types of system SQL templates

This section gives detail information related to the different types of the pre-defined SQL templates.

Even though the statements of each group of templates vary from database to database, according to the operations they are intended to accomplish, they are also grouped on the basis of their types in each folder.

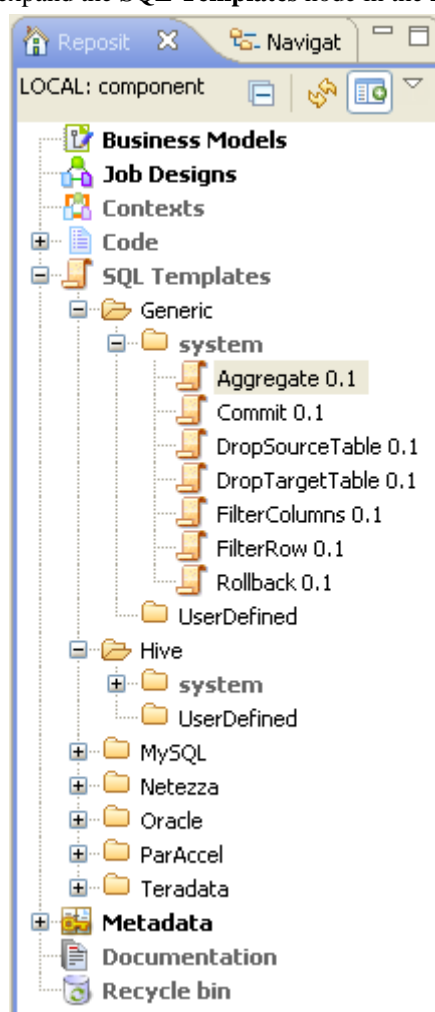
The below table provides these types and their related information.

Name	Function	Associated components	Required component parameters
<i>Aggregate</i>	Realizes aggregation (sum, average, count, etc.) over a set of data.	tSQLTemplateAggregate	Database name Source table name Target table name
<i>Commit</i>	Sends a Commit instruction to RDBMS	tSQLTemplate tSQLTemplateAggregate tSQLTemplateCommit tSQLTemplateFilterColumns tSQLTemplateFilterRows tSQLTemplateMerge tSQLTemplateRollback	Null
<i>Rollback</i>	Sends a Rollback instruction to RDBMS.	tSQLTemplate tSQLTemplateAggregate tSQLTemplateCommit tSQLTemplateFilterColumns tSQLTemplateFilterRows tSQLTemplateMerge tSQLTemplateRollback	Null
<i>DropSourceTable</i>	Removes a source table.	tSQLTemplate tSQLTemplateAggregate tSQLTemplateFilterColumns tSQLTemplateFilterRows	Table name (when use tSQLTemplate) Source table name
<i>DropTargetTable</i>	Removes a target table	tSQLTemplateAggregate tSQLTemplateFilterColumns tSQLTemplateFilterRows	Target table name
<i>FilterColumns</i>	Selects and extracts a set of data from given columns in RDBMS.	tSQLTemplateAggregate tSQLTemplateFilterColumns tSQLTemplateFilterRows	Target table name (and schema) Source table name (and schema)
<i>FilterRow</i>	Selects and extracts a set of data from given rows in RDBMS.	tSQLTemplateAggregate tSQLTemplateFilterColumns tSQLTemplateFilterRows	Target table name (and schema) Source table name (and schema) Conditions
<i>MergeInsert</i>	Inserts records from the source table to the target table.	tSQLTemplateMerge tSQLTemplateCommit	Target table name (and schema)

			Source table name (and schema)
			Conditions
<i>MergeUpdate</i>	Updates the target table with records from the source table.	tSQLTemplateMerge tSQLTemplateCommit	Target table name (and schema)
			Source table name (and schema)
			Conditions

11.3.2. How to access a system SQL template

To access a system SQL template, expand the **SQL Templates** node in the **Repository** tree view.



Each folder contains **system** sub-folders containing pre-defined SQL statements, as well as a **UserDefined** folder in which you can store SQL statements that you have created or customized.

Each system folder contains several types of SQL templates, each designed to accomplish a dedicated task.

Apart from the **Generic** folder, the SQL templates are grouped into different folders according to the type of database for which they are to be used. The templates in the **Generic** folder are standard, for use in any database. You can use these as a basis from which you can develop more specific SQL templates than those defined in *Talend Open Studio for ESB*.



The **system** folders and their content are read only.

From the **Repository** tree view, proceed as follows to open an SQL template:

1. In the **Repository** tree view, expand **SQL Templates** and browse to the template you want to open.
2. Double click the class that you want to open, for example, *aggregate* in the **Generic** folder.

The *aggregate* template view displays in the workspace.

```

1 <%=
2   EXTRACT(__GROUPBY__);
3   EXTRACT(__OPERATION__);
4   String operation = "";
5   boolean flag=false;
6   for(int i=0; i < __OPERATION_INPUT_COLUMN__.length; i++){
7     if(flag){
8       operation += ",";
9     }else{
10      flag=true;
11     }
12     operation += (__OPERATION_FUNCTION__[i] + "(" + __OPERATION_INPU
13   }
14 }
15 %>
16
17 INSERT INTO <%= __TABLE_NAME_TARGET__ %> (<%=StringUtils.list(__OPERATI
18 SELECT <%= operation %>, <%= StringUtils.list(__GROUPBY_INPUT_COLUMN_
19 GROUP BY <%=StringUtils.list(__GROUPBY_INPUT_COLUMN__, ",", "", "", "") %>
  
```

You can read the predefined *aggregate* statements in the template view. The parameters, such as `TABLE_NAME_TARGET`, `operation`, are to be defined when you design related Jobs. Then the parameters can be easily set in the associated components, as mentioned in the previous section.

Everytime you click or open an SQL template, its corresponding property view displays at the bottom of the studio. Click the *aggregate* template, for example, to view its properties as presented below:

Aggregate 0.1	
Main	Name: <input type="text" value="Aggregate"/>
Version	Author: <input type="text" value="ychen@talend.com"/> Version: <input type="text" value="0.1"/> <input type="button" value="M"/> <input type="button" value="m"/>
	Purpose: <input type="text"/> Status: <input type="text"/>
	Description: <input type="text"/>
	Creation: <input type="text" value="6/14/10 2:33 PM"/> Modification: <input type="text" value="6/14/10 2:33 PM"/>

For further information regarding the different types of SQL templates, see [Section 11.3.1, “Types of system SQL templates”](#)

For further information about how to use the SQL templates with the associated components, see [Section 4.4.3, “How to use the SQL Templates”](#).

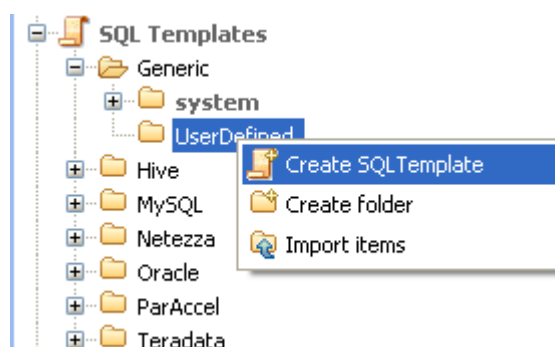
11.3.3. How to create user-defined SQL templates

As the transformation you need to accomplish in ELT may exceed the scope of what the given SQL templates can achieve, *Talend Open Studio for ESB* allows you to develop your own SQL templates according to some writing rules. These SQL templates are stored in the **User-defined** folders grouped according to the database type in which they will be used.

For more information on the SQL template writing rules, see [Appendix C, SQL template writing rules](#).

To create a user-defined SQL template:

1. In the **Repository** tree view, expand **SQL Templates** and then the category you want to create the SQL template in.



2. Right-click **UserDefined** and select **Create SQL Template** to open the [SQL Templates] wizard.

A screenshot of the 'New SQL Template' wizard dialog box. The title bar says 'New SQL Template'. The main title is 'New SQL Template' and the subtitle is 'Add a new SQL Template to repository'. The form contains the following fields: 'Name' (SQLTemplate), 'Purpose' (empty), 'Description' (empty text area), 'Author' (test@talend.com), 'Locker' (empty), 'Version' (0.1 with 'M' and 'm' buttons), 'Status' (dropdown menu), and 'Path' (Generic/UserDefined with a 'Select' button). At the bottom, there is a help icon, a 'Finish' button, and a 'Cancel' button.

3. Enter the information required to create the template and click **Finish** to close the wizard.

The name of the newly created template appears under **UserDefined** in the **Repository** tree view. Also, an SQL template editor opens on the design workspace, where you can enter the code for the newly created template.

For further information about how to create a user-defined SQL template and how to use it in a Job, see **tMysqlTableList** in *Talend Open Studio Components Reference Guide*.

11.3.4. A use case of system SQL templates

As there are many common, standardized SQL statements, *Talend Open Studio for ESB* allows you to benefit from various system SQL templates.

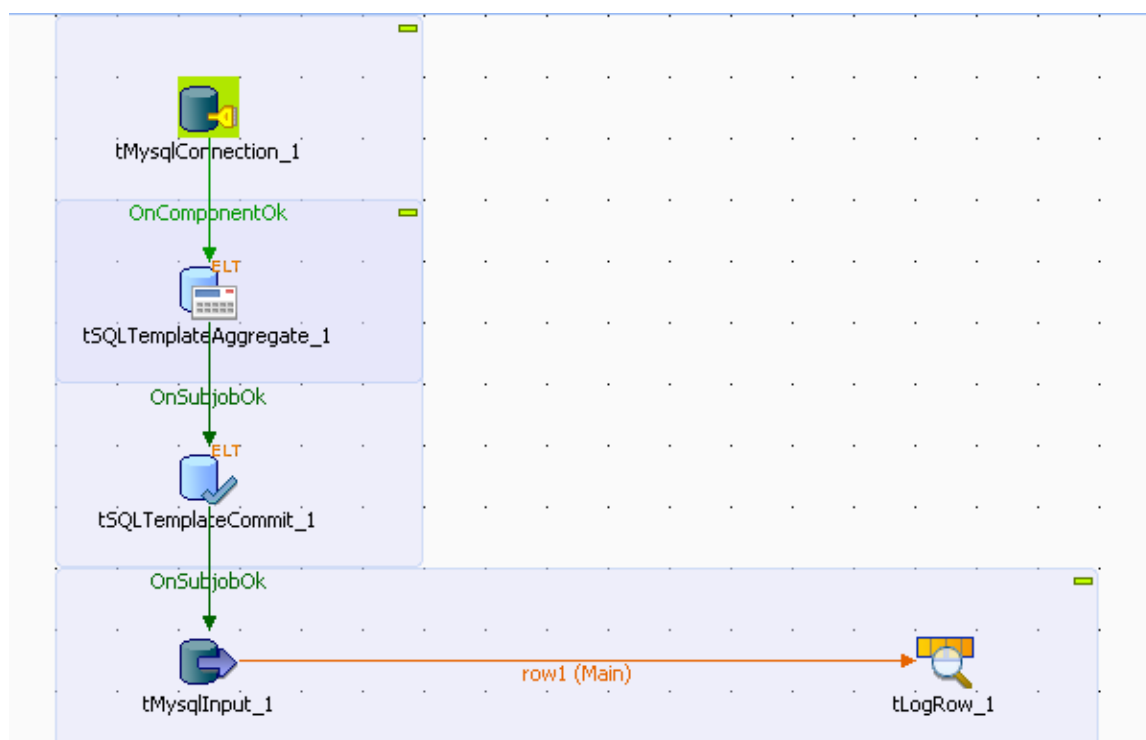
This section presents you with a use case that takes you through the steps of using MySQL system templates in a Job that:

- opens a connection to a Mysql database.
- collects data grouped by specific value(s) from a database table and writes aggregated data in a target database table.
- deletes the source table where the aggregated data comes from.
- reads the target database table and lists the Job execution result.

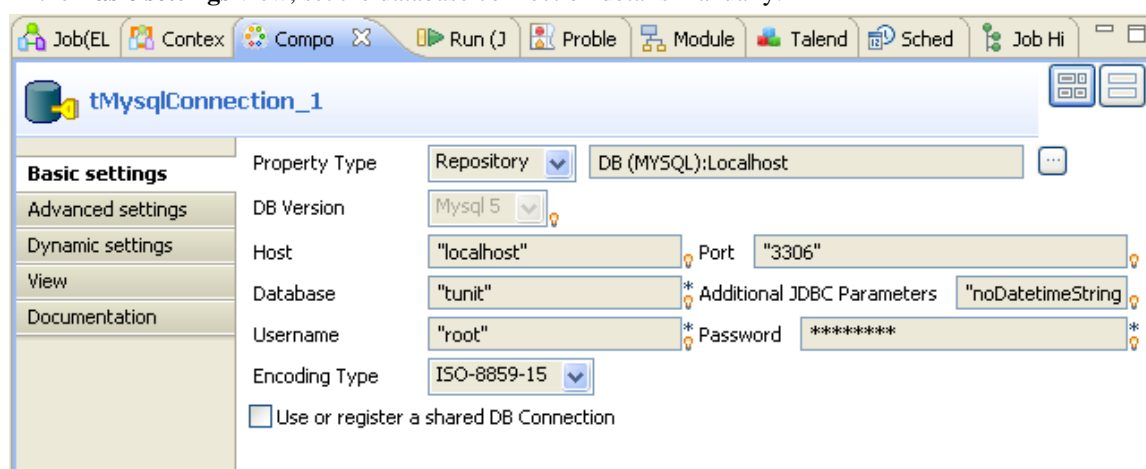
To connect to the database and aggregate the database table columns:

Procedure 11.1. Configuring a connection to a MySQL database

1. Drop the following components from the **Palette** onto the design workspace: **tMysqlConnection**, **tSQLTemplateAggregate**, **tSQLTemplateCommit**, and **tMysqlInput**, **tLogRow**.
2. Right-click **tMysqlconnection** and from the contextual menu, select **Trigger > OnComponentOk** to connect **tMysqlConnection** to **tSQLTemplateAggregate**.
3. Connect **tSQLTemplateAggregate**, **tSQLTemplateCommit** and **tMysqlInput** using **OnComponentOk** links.
4. Connect **tMysqlInput** to **tLogRow** using an **Main > Row** link.



5. In the design workspace, select **tMysqlConnection** and click the **Component** tab to define the component basic settings.
6. In the **Basic settings** view, set the database connection details manually.



7. In the design workspace, select **tSQLTemplateCommit** and click the **Component** tab to define its basic settings.
8. On the **Database type** list, select the relevant database type, and from the **Component list**, select the relevant database connection component if more than one connection is used.

Procedure 11.2. Grouping data, writing aggregated data and dropping the source table

1. In the design workspace, select **tSQLTemplateAggregate** and click the **Component** tab to define its basic settings.
2. On the **Database type** list, select the relevant database type, and from the **Component list**, select the relevant database connection component if more than one connection is used.
3. Enter the names for the database, source table, and target table in the corresponding fields and click the [...] button next to **Edit schema** to define the data structure in the source and target tables.

The source table schema consists of three columns: *First_Name*, *Last_Name* and *Country*. The target table schema consists of two columns: *country* and *total*. In this example, we want to group citizens by their nationalities and count citizen number in each country. To do that, we define the **Operations** and **Group by** parameters accordingly.

tSQLTemplateAggregate_1

Basic settings

Database Type: Mysql Component List: tMysqlConnection_1

Advanced settings

Database name: tunit

Source table name: CitizenWithLabel Schema: Repository DB (MYSQL):Localhost - citizen

Target table name: CitizenCount Schema: Repository DB (MYSQL):Localhost - citizen

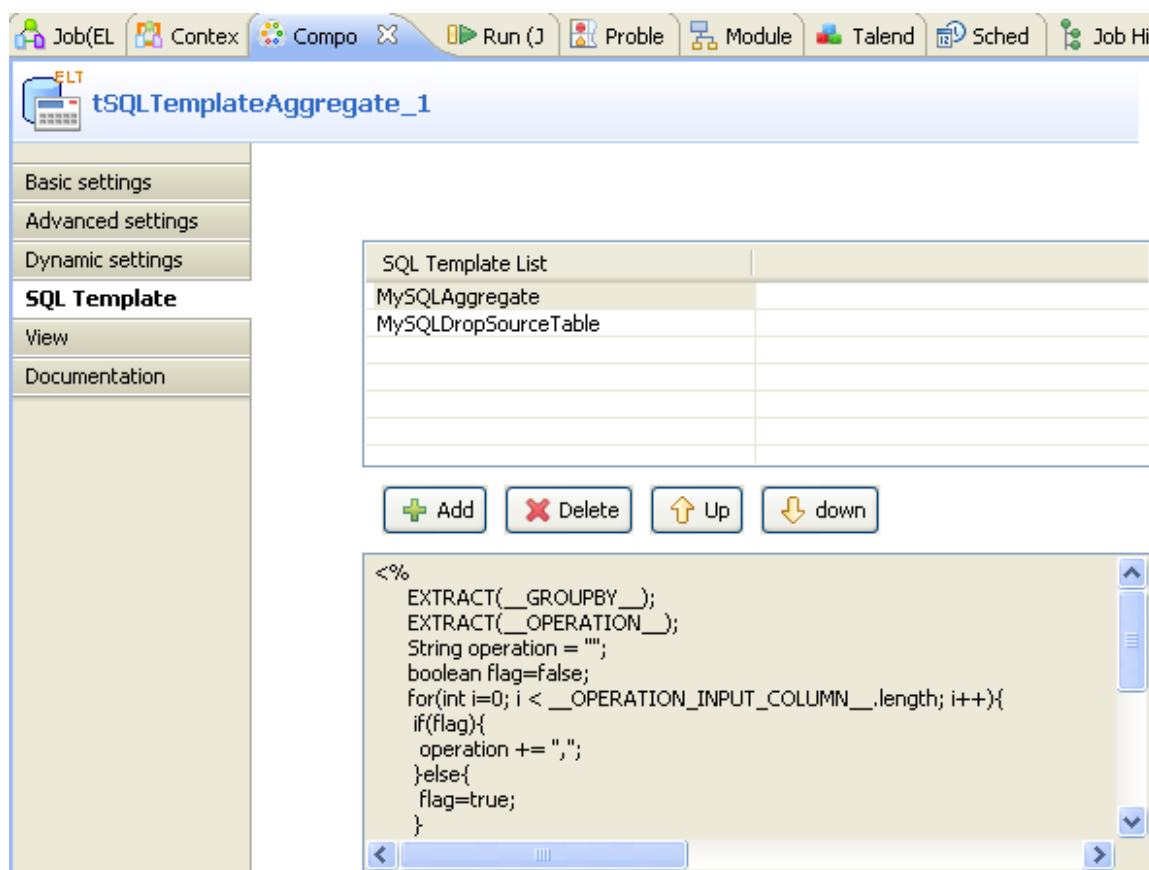
Operations

Output column	Function	Input column position
Total	count	Country

Group by

Output column	Input column position
Country	Country

4. In the **Operations** table, click the plus button to add one or more lines and then click in the **Output column** line to select the output column that will hold the counted data.
5. Click in the **Function** line and select the operation to be carried on.
6. In the **Group by** table, click the plus button to add one or more lines and then click in the **Output column** line to select the output column that will hold the aggregated data.
7. Click the **SQL template** tab to open the corresponding view.



8. Click the plus button twice under the **SQL template list** table to add two SQL templates.
9. Click on the first SQL template row and select the *MySQLAggregate* template from the drop-down list. This template generates codes to aggregate data according to the configuration in the **Basic settings** view.
10. Do the same to select the *MySQLDropSourceTable* template for the second SQL template row. This template generates codes to delete the source table where the data to be aggregated comes from.



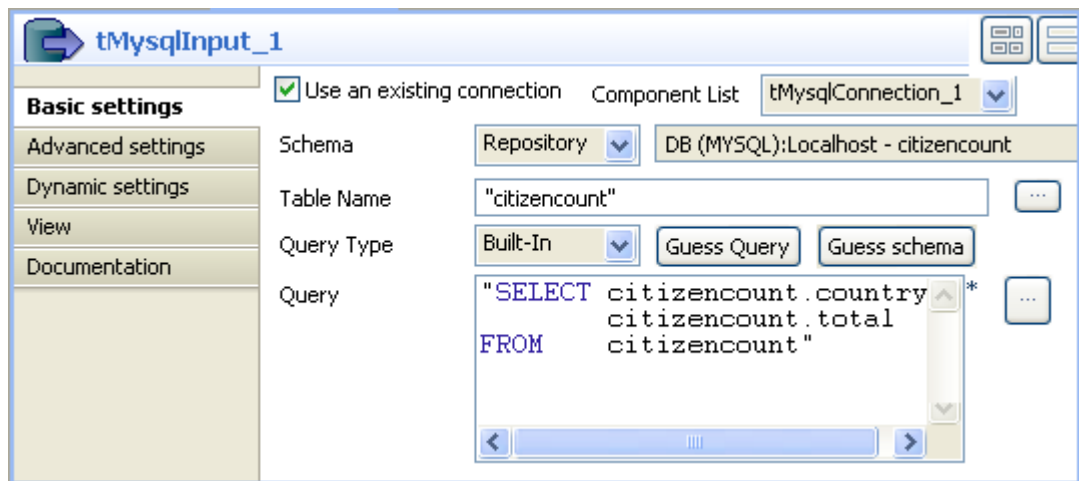
To add new SQL templates to an ELT component for execution, you can simply drop the templates of your choice either onto the component in the design workspace, or onto the component's **SQL template list** table.



The templates set up in the **SQL template list** table have priority over the parameters set in the **Basic settings** view and are executed in a top-down order. So in this use case, if you select **MySQLDropSourceTable** from the list, the source table will be deleted prior to aggregation, meaning that nothing will be aggregated.

Procedure 11.3. Reading the target database and listing the Job execution result

1. In the design workspace, select **tMysqlInput**, and click the **Component** tab to define its basic settings.



2. Select the **Use an existing connection** check box to use the database connection that you have defined on the **tMysqlConnection** component.
3. To define the schema, select **Repository** and then click the three-dot button to choose the database table whose schema is used. In this example, the target table holding the aggregated data is selected.
4. In the **Table Name** field, type in the name of the table you want to query. In this example, the table is the one holding the aggregated data.
5. In the **Query** area, enter the query statement to select the columns to be displayed.
6. Save your Job and press **F6** to execute it.

The source table is deleted.

```
Starting job ELTYudong at 02:43 24/05/2010.

[statistics] connecting to socket on port 3918
[statistics] connected

+-----+
| tLogRow_1 |
+-----+
| country | total |
+-----+
| Canada  | 2030  |
| China   | 2012  |
| France  | 2009  |
| Japan   | 1925  |
| USA     | 2024  |
+-----+

[statistics] disconnected
Job ELTYudong ended at 02:43 24/05/2010. [exit code=0]
```

A two-column table *citizencount* is created in the database. It groups citizens according to their nationalities and gives their total count in each country.



Appendix A. GUI

This appendix describes the Graphical User Interfaces (GUI) of *Talend Open Studio for ESB*.

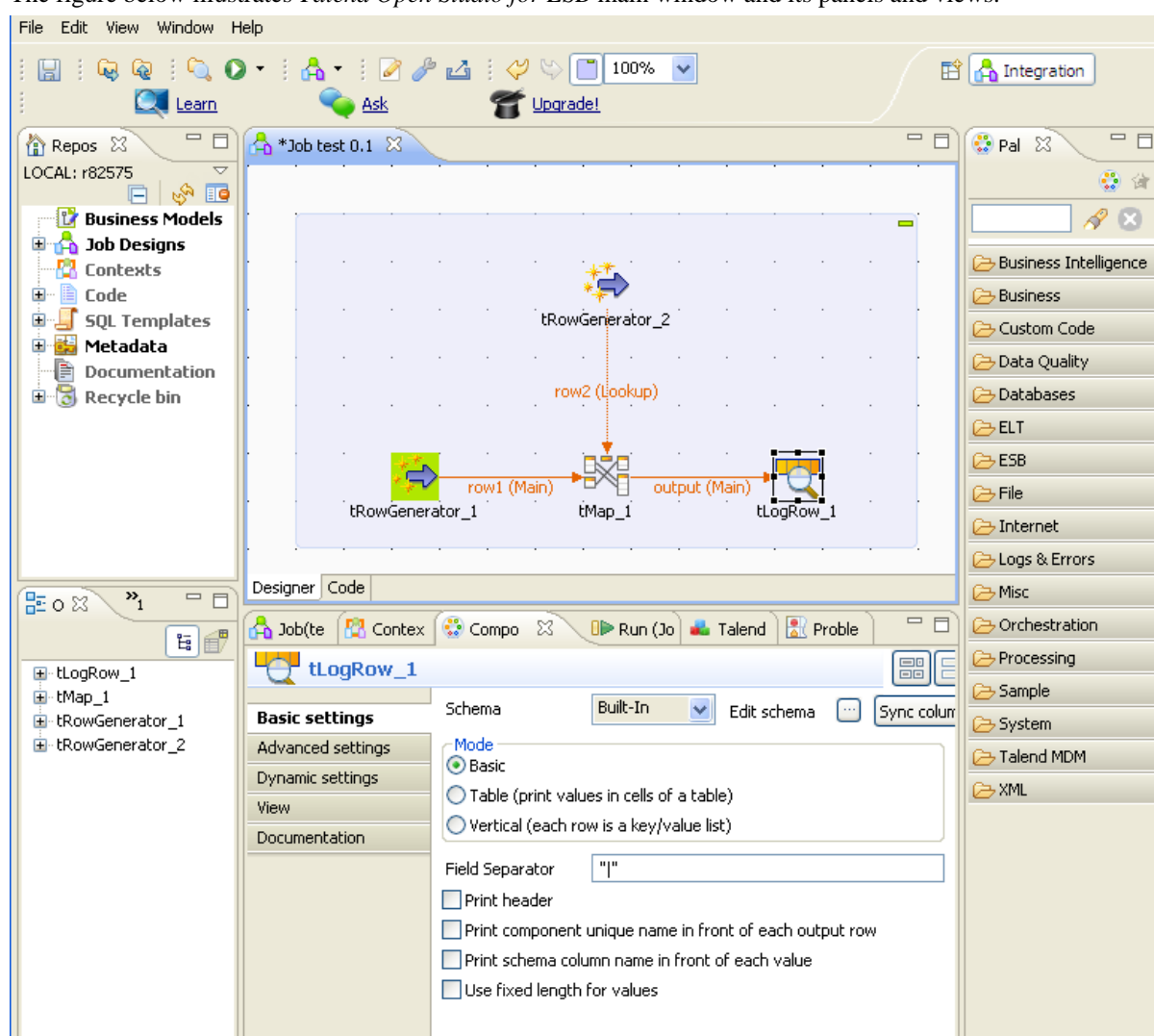
A.1. Main window

Talend Open Studio for ESB main window is the interface from which you manage all types of data integration processes.

The *Talend Open Studio for ESB* multi-panel window is divided into:

- menu bar,
- toolbar,
- **Repository** tree view,
- design workspace,
- Palette,
- various configuration views in a tab system, for any of the elements in the data integration Job designed in the workspace,
- **Outline view** and **Code Viewer**.

The figure below illustrates *Talend Open Studio for ESB* main window and its panels and views.



The various panels and their respective features are detailed hereafter.



All the panels, tabs, and views described in this documentation are specific to *Talend Open Studio for ESB*. Some views listed in the [Show View] dialog box are Eclipse specific and are not subjects of this documentation. For information on such views, check Eclipse online documentation at <http://www.eclipse.org/documentation/>.

A.2. Menu bar and Toolbar

At the top of the *Talend Open Studio for ESB* main window, various menus and a quick access toolbar gather **Talend** commonly features along with some Eclipse functions.

A.2.1. Menu bar of *Talend Open Studio for ESB*

Talend Open Studio for ESB's menus include:

- some standard functions, such as **Save**, **Print**, **Exit**, which are to be used at the application level.
- some Eclipse native features to be used mainly at the design workspace level as well as specific *Talend Open Studio for ESB* functions.

The table below describes menus and menu items available to you on the menu bar of *Talend Open Studio for ESB*.



The menus on the menu bar differ slightly according to what you are working with: a Business Model or a Job.

Menu	Menu item	Description
File	Close	Closes the current open view on the Studio design workspace.
	Close All	Closes all open views on the Studio design workspace.
	Save	Saves any changes done in the current open view.
	Save as	Saves any changes done without changing the current open view. For more information, see Section 3.6, "Saving a Business Model"
	Save All	Saves any changes done in all open views.
	Print	Unavailable option.
	Switch project	Closes the current session and launches another one to enable you to open a different project in the Studio.
	Edit project properties	Opens a dialog box where you can customize the settings of the current project. For more information, see Section 2.6, "Customizing project settings"
	Import	Opens a wizard that helps you to import different types of resources (files, items, preferences, XML catalogs, etc.) from different sources.
	Export	Opens a wizard that helps you to export different types of resources (files, items, preferences, breakpoints, XML catalogs, etc.) to different destinations.
Edit	Exit	Closes The Studio main window.
	Open File	Opens a file from within the Studio.
	Undo	Undoes the last action done in the Studio design workspace.
	Redo	Redoes the last action done in the Studio design workspace.
	Cut	Cuts selected object in the Studio design workspace.

Menu	Menu item	Description
	Copy	Copies the selected object in the Studio design workspace.
	Paste	Pastes the previously copied object in the Studio design workspace.
	Delete	Deletes the selected object in the Studio design workspace.
	Select All	Selects all components present in the Studio design workspace.
View	Zoom In	Obtains a larger image of the open Job.
	Zoom Out	Obtains a smaller image of the open Job.
	Grid	Displays grid in the design workspace. All items in the open Job are snapped to it.
	Snap to Geometry	Enables the Snap to Geometry feature.
Window	Perspective	Opens different perspectives corresponding to the different items in the list.
	Show View...	Opens the [Show View] dialog box which enables you to display different views on the Studio.
	Maximize Active View or Editor...	Maximizes the current perspective.
	Preferences	Opens the [Preferences] dialog box which enables you to set your preferences. For more information about preferences, see Section 2.5, “Setting Talend Open Studio for ESB preferences”
Help	Welcome	Opens a welcoming page which has links to <i>Talend Open Studio for ESB</i> documentation and Talend practical sites.
	Help Contents	Opens the Eclipse help system documentation.
	About Talend Open Studio for ESB	Displays: -the software version you are using, -detailed information on your software configuration that may be useful if there is a problem, -detailed information about plug-in(s), -detailed information about <i>Talend Open Studio for ESB</i> features.
	Export logs	Opens a wizard that helps you to export all logs generated in the Studio and system configuration information to an archived file.
	Software Updates	Find and Install...: Opens the [Install/Update] wizard that helps searching for updates for the currently installed features, or searching for new features to install.
		Manage Configuration...: Opens the [Product Configuration] dialog box where you can manage <i>Talend Open Studio for ESB</i> configuration.

A.2.2. Toolbar of Talend Open Studio for ESB

The toolbar contains icons that provide you with quick access to the commonly used operations you can perform from *Talend Open Studio for ESB* main window.



The icons on the toolbar differ slightly according to what you are working with: a Business Model or a Job.

The table below describes the toolbar icons and their functions.

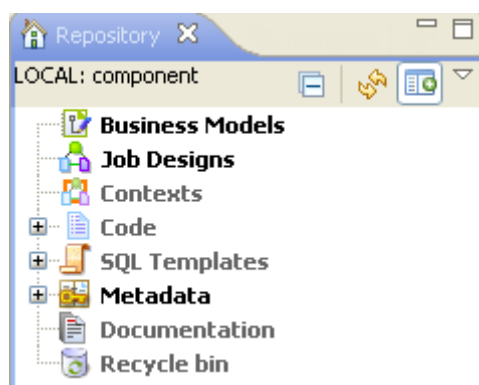
Name	Icon	Description
Save		Saves current job design.
Save as		Saves as another new Job.
Export items		Exports repository items to an archive file, for deploying outside <i>Talend Open Studio for ESB</i> . Instead if you intend to import the exported element into a newer version of <i>Talend Open Studio for ESB</i> or of another workstation, make sure the source files are included in the archive.
Import items		Imports repository items from an archive file into your current <i>Talend Open Studio for ESB</i> . For more information regarding the import/export items feature, see Section 7.2.1, “How to import items” .
Find a specific job		Displays the relevant dialog box that enables you to open any Job listed in the Repository tree view.
Run job		Executes the Job currently shown on the design space. For more information about job execution, see Section 4.2.7, “How to run a Job” .
Create		Launches the relevant creation wizard. Through this menu, you can create any repository item including Business models, Job Designs, contexts, routines and metadata entries.
Project settings		Launches the [Project Settings] dialog box. From this dialog box, you can add a description to the current Project and customize the Palette display. For more information, see Section 2.6, “Customizing project settings” .
Detect and update all jobs		Searches for all updates available for your Jobs.
Export Talend projects		Launches the [Export Talend projects] wizard. For more information about project export, see Section 2.4.6, “How to export a project” .

A.3. Repository tree view


The **Repository** tree view gathers all the technical items that can be used either to describe business models or to design Jobs. It gives access to any item including **Business Models**, **Job Designs**, as well as reusable routines or documentation.

The **Repository** centralizes and stores all necessary elements for any Job design and business modeling contained in a project.

The figure below illustrates the elements stored in the **Repository**.





The **Refresh** button  allows you to update the tree view with the last changes made.

The **Activate filter** button  allows you to open the filter settings view so as to configure the display of the **Repository** view.

The **Repository** tree view stores all your data (Business, Jobs) and metadata (Routines, DB/File connections, any meaningful Documentation and so on).

The table below describes the nodes in the **Repository** tree view.

Node	Description
Business Models	Under the Business Models folder, are grouped all business models of the project. Double-click the name of the model to open it on the design workspace. For more information, see Chapter 3, Designing a Business Model .
Job Designs	The Job Designs folder shows the tree view of the designed Jobs for the current project. Double-click the name of the Job to open it on the design workspace. For more information, see Chapter 4, Designing a Job .
Contexts	The Context folder groups files holding the contextual variables that you want to reuse in various Jobs, such as filepaths or DB connection details. For more information, see Section 4.4.2, “How to centralize contexts and variables” .
Code	The Code folder is a library that groups the routines available for this project and other pieces of code that could be reused in the project. Click the relevant tree entry to expand the appropriate code piece. For more information, see Chapter 4, Designing a Job .
SQL Templates	The SQL Templates folder groups all system SQL templates and gives the possibility to create user-defined SQL templates. For more information, see Section 4.4.3, “How to use the SQL Templates” .
Metadata	The Metadata folder bundles files holding redundant information you want to reuse in various Jobs, such as schemas and property data. For more information, see Chapter 4, Designing a Job .
Documentation	The Documentation folder gathers all types of documents, of any format. This could be, for example, specification documents or a description of technical format of a file. Double-click to open the document in the relevant application. For more information, see Section 7.6.1, “How to generate HTML documentation” .
Recycle bin	The Recycle bin groups all elements deleted from any folder in the Repository tree view.  The deleted elements are still present on your file system, in the recycle bin, until you right-click the recycle bin icon and select Empty Recycle bin .

Node	Description
	 Expand the recycle bin to view any folders, subfolders or elements held within. You can action an element directly from the recycle bin, restore it or delete it forever by clicking right and selecting the desired action from the list.

A.4. Design workspace

In the *Talend Open Studio for ESB*'s design workspace, both Business Models and Job Designs can be laid out.

For more information, see [Section 3.2, “Opening or creating a Business Model”](#) and [Section 4.2.1, “How to create a Job”](#).

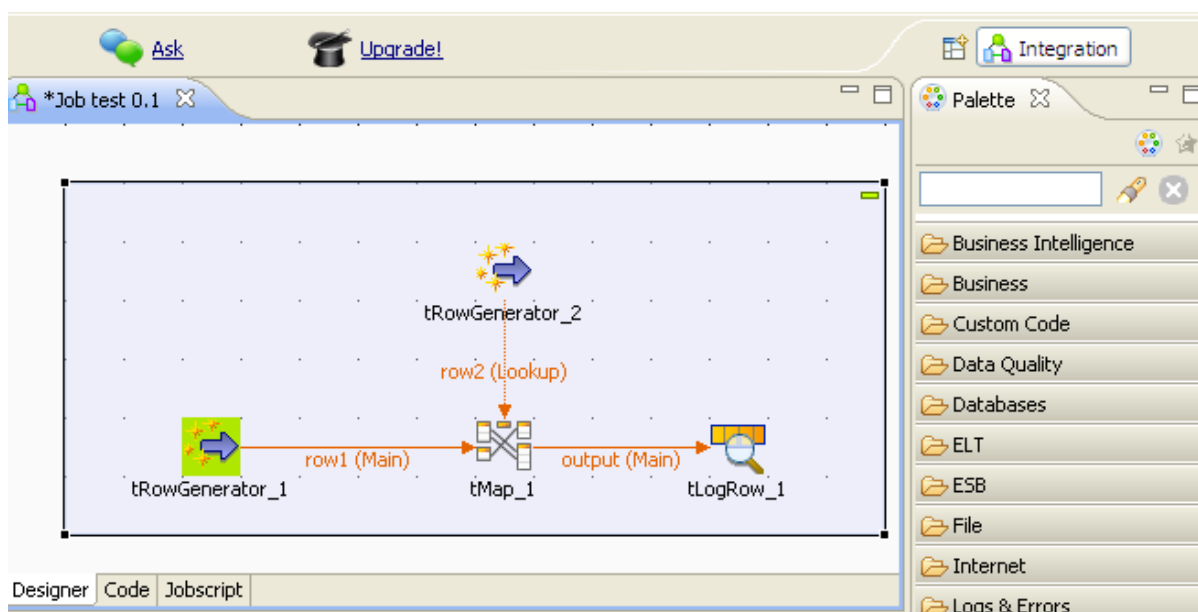
For both Business Models and Job Designs: active designs display in a easily accessible tab system above this workspace.

For Job Designs only: Under this workspace, you can access several other tabs:

- the **Designer** tab: opens by default when creating a Job. It displays the Job in a graphical mode.
- the **Code** tab: enables you to visualize the code and highlights the possible language errors.



Warnings are indicated in yellow whereas errors are indicated in red.



A **Palette** is docked at the top of the design workspace to help you draw the model corresponding to your workflow needs.

A.5. Palette

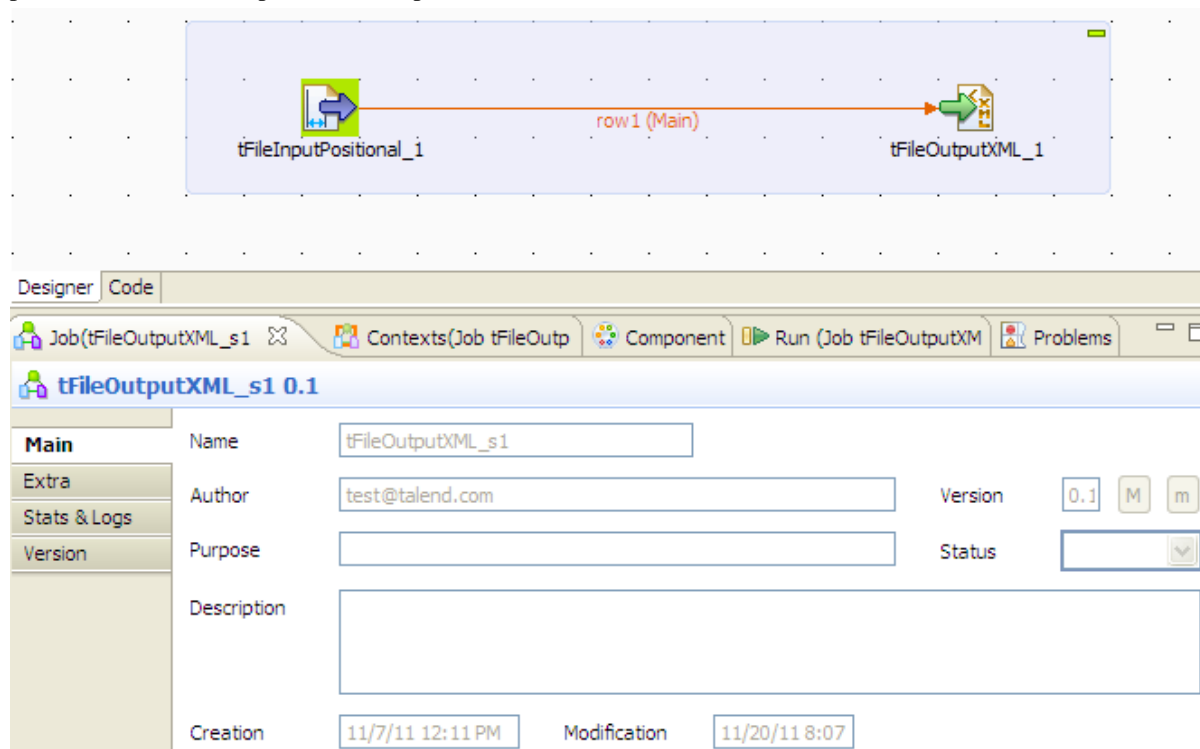
From the **Palette**, depending on whether you are designing a Job or modeling a Business Model, you can drop technical components or shapes, branches and notes to the design workspace. You can define and format them using the various tools offered in the **Business Model** view for the Business Models and in the **Component** view for the Job.

Related topics:

- [Chapter 3, *Designing a Business Model*](#).
- [Chapter 4, *Designing a Job*](#).
- [Section 4.2.8.1, “How to change the Palette layout and settings”](#).

A.6. Configuration tabs

The configuration tabs are located in the lower half of the design workspace. Each tab opens a view that displays the properties of the selected element in the design workspace. These properties can be edited to change or set the parameters related to a particular component or to the Job as a whole.



The **Component**, **Run Jobs**, **Problems** and **Error Log** views gather all information relative to the graphical elements selected in the design workspace or the actual execution of the open Job.


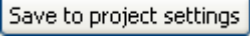
The **Modules** and **Scheduler** tabs are located in the same tab system as the **Component**, **Logs** and **Run Job** tabs. Both views are independent from the active or inactive Jobs open on the design workspace.



You can show more tabs in this tab system and directly open the corresponding view if you select **Window** > **Show view** and then, in the open dialog box, expand any node and select the element you want to display.

The sections below describe the view of each of the configuration tabs.

View	Description
Component	<p>This view details the parameters specific to each component of the Palette. To build a Job that will function, you are required to fill out the necessary fields of this Component view for each component forming your Job.</p> <p>For more information about the Component view, see Section 4.2.6, “How to define component properties”.</p>
Run Job	<p>This view obviously shows the current job execution. It becomes a log console at the end of an execution.</p>

View	Description
	For details about job execution, see Section 4.2.7, “How to run a Job” .
Error Log	<p>This view is mainly used for Job execution errors. It shows the history of warnings or errors occurring during job executions.</p> <p> The log tab has also an informative function for a Java component operating progress, for example.</p> <p>Error Log tab is hidden by default. As for any other view, go to Window > Show views, then expand PDE Runtime node and select Error Log to display it on the tab system.</p>
Modules	<p>This view shows if a module is necessary and required for the use of a referenced component. Checking the Modules view helps to verify what modules you have or should have to run smoothly your Jobs.</p> <p>For more information, see Section 4.5.4, “How to install external modules”.</p>
Scheduler	<p>This view enables you to schedule a task that will launch periodically the Job you select via the crontab program.</p> <p>For more information, see Section 4.5.5, “How to launch a Job periodically”.</p>
Job view	<p>The Job view displays various information related to the open Job on the design workspace. This view has the following tabs:</p> <p>Main tab</p> <p>This tab displays basic information about the Job opened on the design workspace, i.e. its name, author, version number, etc. The information is read-only. To edit it you have to close your Job, right-click its label on the Repository tree view and click Edit properties on the drop-down list.</p> <p>Extra tab</p> <p>This tab displays extra parameters including multi thread and implicit context loading features. For more information, see Section 4.6.7.2, “How to use the features in the Extra tab”</p> <p>Stats/Log tab</p> <p>This tab allows you to enable/disable the statistics and logs for the whole Job.</p> <p>You can already enable these features for every single component of your Job by simply using and setting the relevant components: tFlowMeterCatcher, tStatCatcher, tLogCatcher.</p> <p>For more information about these components, see <i>Talend Open Studio Components Reference Guide</i>.</p> <p>In addition, you can now set these features for the whole active Job (i.e. all components of your Job) in one go, without using the Catcher components mentioned above. This way, all components get tracked and logged in the File or Database table according to your setting.</p> <p>You can also save the current setting to Project Settings by clicking the  button.</p> <p>For more details about the Stats & Logs automation, see Section 4.6.7.1, “How to automate the use of statistics & logs”.</p> <p>Version tab</p> <p>This tab displays the different versions of the Job opened on the design workspace and their creation and modification dates.</p>

View	Description
Problems	<p>This view displays the messages linked to the icons docked at a components in case of problem, for example when part of its setting is missing. Three types of icons/messages exist: Error, Warning and Infos.</p> <p>For more information, see Section 4.6.3.1, “Warnings and error icons on components”.</p>
Job Hierarchy	<p>This view displays a tree folder showing the child Job(s) of the parent Job selected. To show this view, right-click the parent Job in the Repository tree view and select Open Job Hierarchy on the drop-down list.</p> <p>You can also show this view in the Window > Show view... combination where you can select Talend > Job Hierarchy.</p> <p>You can see Job Hierarchy only if you create a parent Job and one or more child Job(s) via the tRunJob component. For more information about tRunJob, see <i>Talend Open Studio Components Reference Guide</i>.</p>
Properties	<p>When inserting a shape in the design workspace, the Properties view offers a range of formatting tools to help you customizing your business model and improve its readability.</p>

A.7. Outline and code summary panel

This panel is located below the **Repository** tree view. It displays detailed information about the open Job or Business Model in the design workspace.

The Information panel is composed of two tabs, **Outline** and **Code Viewer**, which provide information regarding the displayed diagram (either Job or Business Model) and also the generated code.

For more information, see [Section 4.6.5, “How to display the code or the outline of your Job”](#).

A.8. Shortcuts and aliases

Below is a table gathering all keyboard shortcuts currently in use:

Shortcut	Operation	Context
F2	Shows Component settings view.	Global application
F4	Shows Run Job view.	Global application
F6	Runs current Job or shows Run Job view if no Job is open.	Global application
Ctrl + F2	Shows Module view.	Global application
Ctrl + F3	Shows Problems view.	Global application
Ctrl + H	Shows the Designer view of the current Job.	Global application
Ctrl + G	Shows the Code view of the current Job.	Global application
Ctrl + R	Restores the initial Repository view.	From Repository view
Ctrl + Shift + F3	Synchronizes components javajet components.	Global application
Ctrl + Shift + J	Opens a Job.	Global application (In Windows)
F7	Switches to Debug mode.	From Run Job view
F5	Refreshes the Repository view.	From Repository view

Shortcut	Operation	Context
F8	Kills current Job.	From Run Job view
F5	Refreshes Modules install status.	From Modules view
Ctrl+L	Execute SQL queries.	Talend commands (in Windows)
Ctrl+Space bar	Access global and user-defined variables. It can be error messages or line number for example, depending on the component selected.	From any component field in Job or Component views



Appendix B. Theory into practice: Data service and routing examples

This chapter aims at users of *Talend Open Studio for ESB* who seek a real-life use case to help them take full control over the product. This chapter comes as a complement of the *Talend Open Studio Components Reference Guide* and *Talend Open Studio for ESB Mediation Components Reference Guide*.

B.1. Data service example

This section provides a real-life use case of how to build a data service, deploy the service into Talend Runtime, use the service in a Route, and test the service with a soapUI project.

B.1.1. How to build a data service

To build a data service using *Talend Open Studio for ESB*, you have to design a data service Job that addresses all of the different sources and targets required for data integration processes and combines them with Web services.

Talend Open Studio for ESB provides the **Services** item to help build a data service Job from a given WSDL defined in the **Repository** tree view.

The following sections present a scenario to illustrate how to create the data service Job using a given WSDL.

B.1.1.1. Discovering the scenario

To illustrate the way *Talend Open Studio for ESB* combines data integration with Web services, find below a real-life example scenario. In this scenario, you will define an airport Web service using the corresponding WSDL in the **Repository** tree view, send a request to this Web service to get country codes and retrieve the response from the Web service for a further use. To do this, you need to build two data service Jobs:

- one Job that will give access to the Web service via a WSDL, to send a request and retrieve the response - the data service provider.
- one Job that will actually send data to request the Web service - the data service consumer.

B.1.1.2. Defining the Web service

From the **Services** item of the **Repository** tree view, you are able to define the Web service of interest by editing a WSDL file or by importing an existing WSDL file.

In this scenario, you import an existing WSDL used to access a given airport Web service. For further information about how to create a WSDL file from scratch, see [Section 6.2.1, “How to create a Service ”](#) and [Section 6.2.2, “How to edit a WSDL file”](#).

To define the airport Web service, proceed as follows:

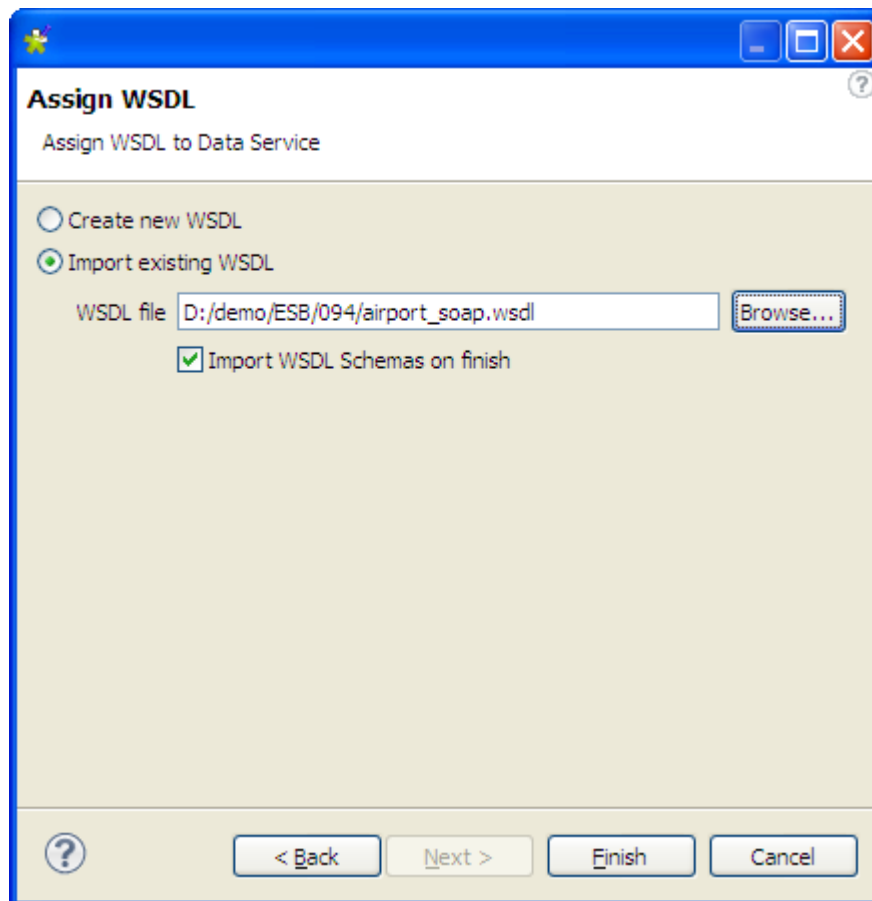
1. On the **Repository** tree view, right-click the **Services** node and from the contextual menu, select **Create Service**.

The image shows a 'title' dialog box with the following fields and values:

- Name:** airport
- Purpose:** (empty)
- Description:** (empty text area)
- Author:** test@talend.com
- Locker:** (empty)
- Version:** 0.1, with radio buttons for 'M' and 'm' (both unselected)
- Status:** (empty dropdown menu)
- Path:** (empty text field with a 'Select' button)

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

2. In the pop-up wizard, enter the information corresponding to the fields you need to complete. In this scenario, enter *airport* in the **Name** field.
3. Click **Next** to go to the next step.
4. Select **Import existing WSDL** and click **Browse...** to navigate to the WSDL file corresponding to the airport Web service of interest. In this example, this file is *airport_soap.wsdl*. By default, the **Import WSDL Schemas on finish** check box is selected. You can also import the schema from the WSDL file at any time after the Service is created. For further information, see [Section 6.3.1, “How to import WSDL schemas”](#). Keep the **Import WSDL Schemas on finish** check box selected in this scenario.



If you need to create a WSDL file from scratch, you have to select the **create new WSDL file** check box and follow the explanations of the wizard. For further information, see [Section 6.2.1, “How to create a Service”](#).

- Click **Finish** to validate this creation. Then the newly defined Web service with exclamation icon displays under the **Services** node of the **Repository** view. The exclamation icon means that this defined Web service is not yet used.



You can edit a Web service already defined in the **Repository** view via a given WSDL editor. For further information, see [Section 6.2.2, “How to edit a WSDL file”](#)..

B.1.1.3. Building data service provider

In this scenario, the data service provider uses the **tESBProviderRequest** and the **tESBProviderResponse** components to create the access to the airport Web service and uses the **tXMLMap** component to join the airport data provided by a given MySQL database into the request-response main flow for publication. The database data is loaded by the **tMysqlInput** component.

To create this data service provider, proceed as follows:

- Under the **Services** node of the **Repository** tree view, right-click the newly defined Web service and from the contextual menu, select **Create New Job**. In this scenario, this Web service is *getAirportInformationByISOCountryCode*.

2. In the opened **[New Job]** wizard, the Job to be created is already named automatically, so simply click **Finish**.

A draft Job is opened on the workspace.

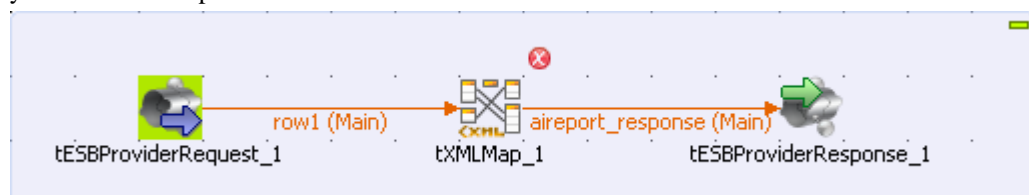
Dropping and linking the components

In the draft Job, a **tESBProviderRequest** and a **tESBProviderResponse** are already selected and configured. **tESBProviderRequest** will send a request to the specified Web service and **tESBProviderResponse** will send back the response corresponding to the request. These two components can be found in the **ESB** group of the **Palette**.

To build the data service provider Job, proceed as follows:

1. Right-click **tESBProviderRequest**, hold and drag to **tESBProviderResponse** to link these two components.
2. Drop a **tXMLMap** in the middle of the row link from the **Palette** and in the pop-up window, name the output link as, for example, *airport_response*. This will also be used as the name of the output table in the map editor of **tXMLMap**. For further information, see [Chapter 4, Designing a Job](#).

Then your data service provider Job should look like:



The red cross icon prompts you to configure the **tXMLMap** component.

- From the **Db Connections** node of the **Repository** tree view, drop the connection to the airport data, the *airport* database table in this example, onto the workspace. Then the **[Components]** wizard is opened.

For further information about how to create a database connection in the **Repository**, see [Section 9.2, “Setting up a DB connection”](#).

- Double click **tMySQLInput** in this wizard to create the corresponding component on the workspace and link it to **tXMLMap**.

In this scenario, the airport data is composed of airport names and the corresponding country codes. The following figure presents the database table in use.

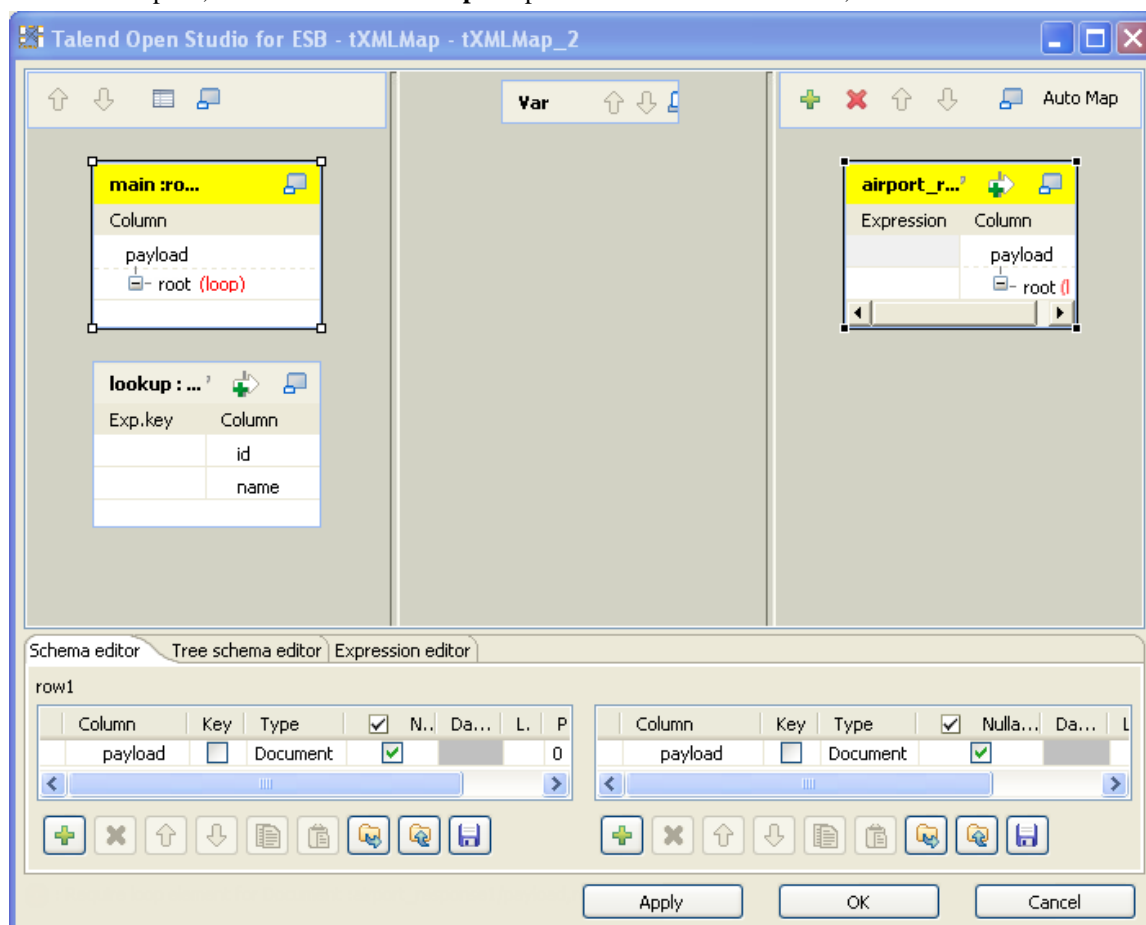
id	name
FR	Roissy
FR	Orly
FR	Marignan
CN	Beijing Capital
CN	Shanghai Hongqiao

Till now, you need only to configure **tXMLMap** as the other components are already configured automatically.

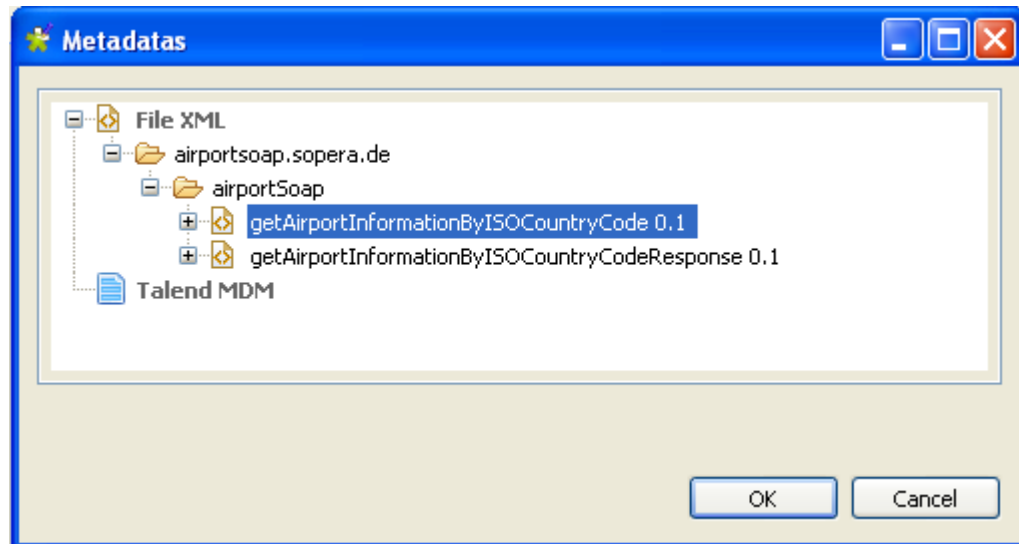
Configuring tXMLMap

To do this, perform the following operations:

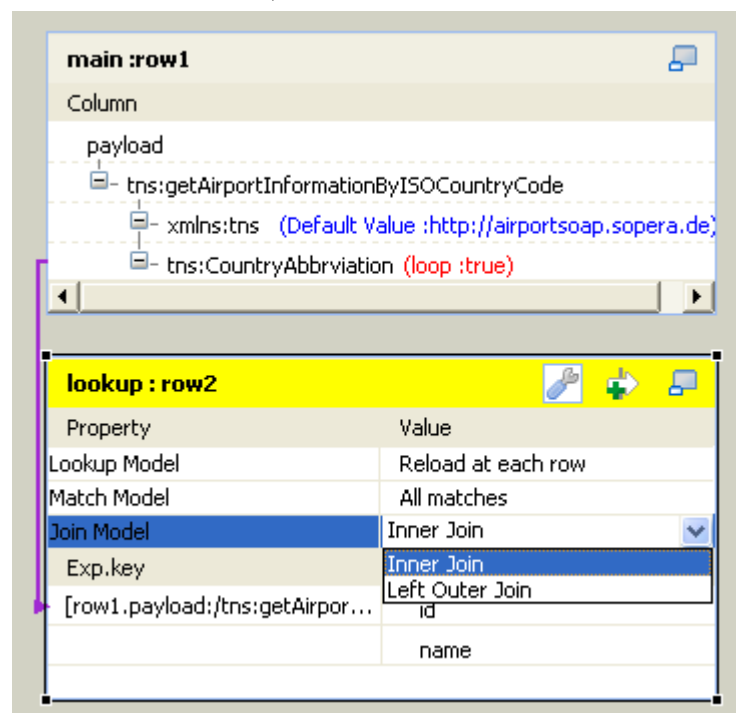
- On the workspace, double click **tXMLMap** to open its editor. At this moment, the editor should look like:



2. In the main row table of the input flow side (left), right-click the column name *payload* and from the contextual menu, select **Import from Repository**. Then the [Metadatas] wizard is opened. For further information, see [Section 8.3.1, “Using the document type to create the XML tree”](#).
3. Expand the **File XML** node in this wizard, select the schema of the request side and click **OK** to validate this selection. In this example, the schema is *getAirportInformationByISOCountryCode*.



4. Do the same to import the hierarchical schema for the response side (right). In this example, this schema is *getAirportInformationByISOCountryCodeResponse*.
5. Then to create the join to the lookup data, click the *CountryAbbreviation* node in the main row of the input side (left), hold and drop it onto the **Exp.key** column of the lookup flow, corresponding to the *id* row.
6. On the table representing the lookup flow, click the wrench icon on the up-right corner to open the setting panel.
7. Set **Lookup Model** as **Reload at each row**, **Match Model** as **All matches** and **Join Model** as **Inner join**.



For further information about **Lookup Model**, see [Section 8.2.7, “Handling Lookups”](#).

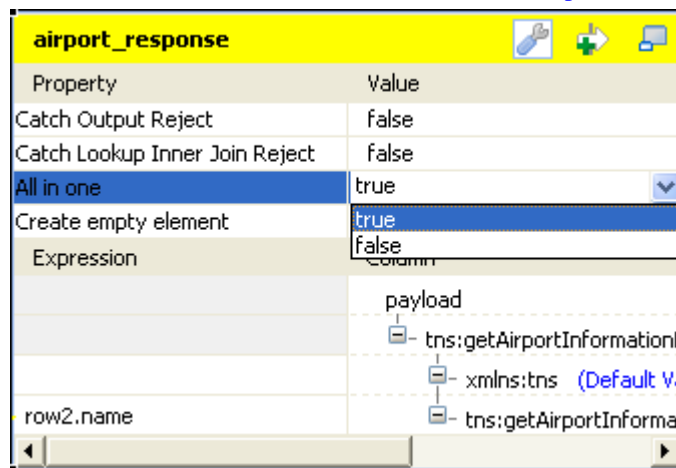
For further information about **Match Model**, see [Section 8.2.1.2, “How to use Explicit Join”](#).

For further information about **Join Model**, see [Section 8.2.1.3, “How to use Inner Join”](#).

A step-by-step tutorial related to this Join topic is available on the Talend Technical Community Site. For further information, see <http://talendforge.org/tutorials/tutorial.php?language=english&idTuto=101>.

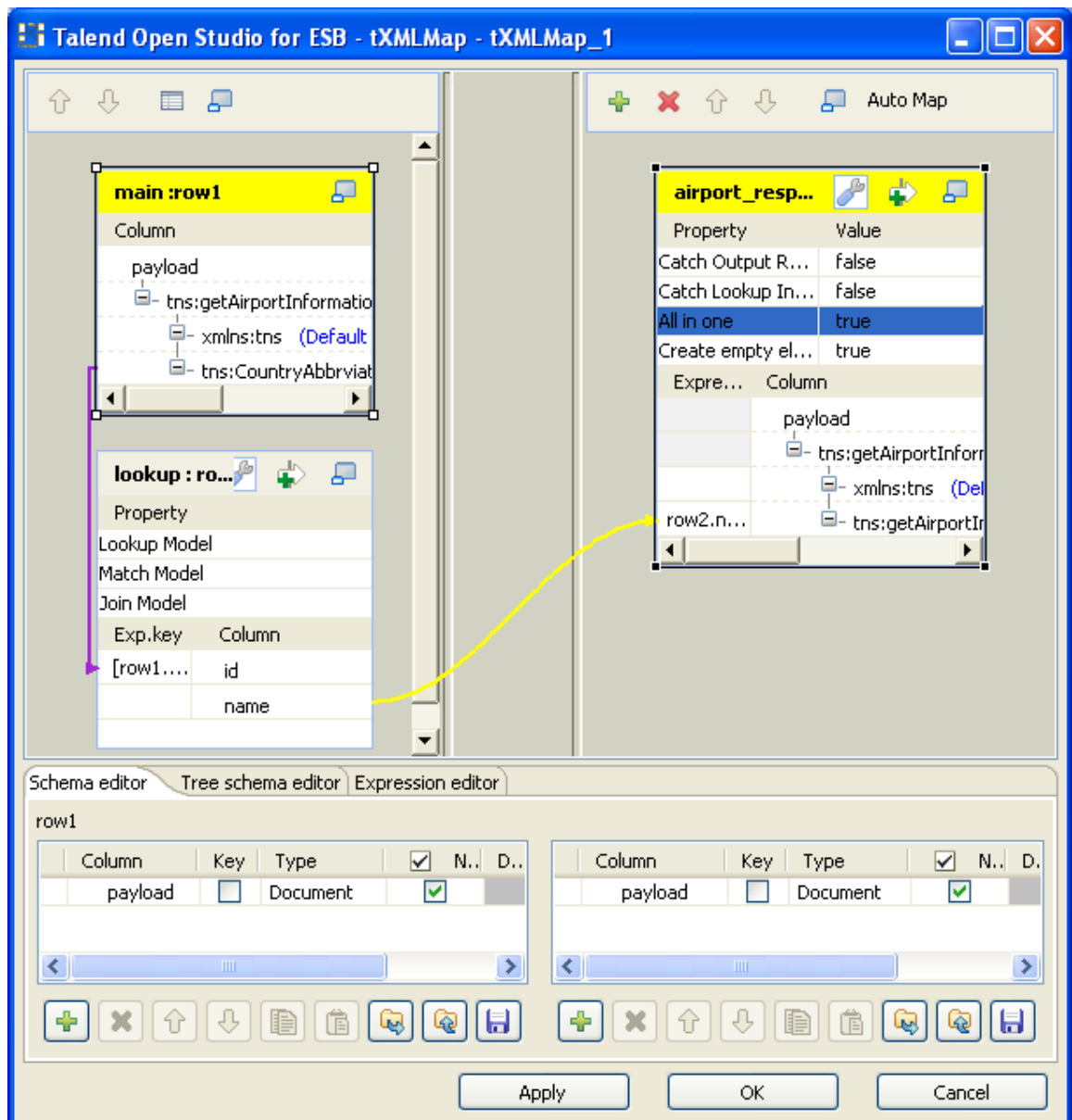
8. Do the same to open the setting panel on the output side (right) and set the **All in one** option as **true**. This ensures that only one response is returned every time when one request is sent, as, otherwise, the airport data from the given database may provide several airports, thus several responses, to each country code that you send as one request.

For further information about **All in one**, see [Section 8.3.2.1, “How to output elements into one document”](#)



9. Click the name row in the lookup flow (left), hold and drop it onto the **Expression** column corresponding to the *tns:getAirportInformationByISOCountryCodeResult* node in the XML tree view of the output flow (*airport_response* in this example).

Then your **tXMLMap** editor should look like:

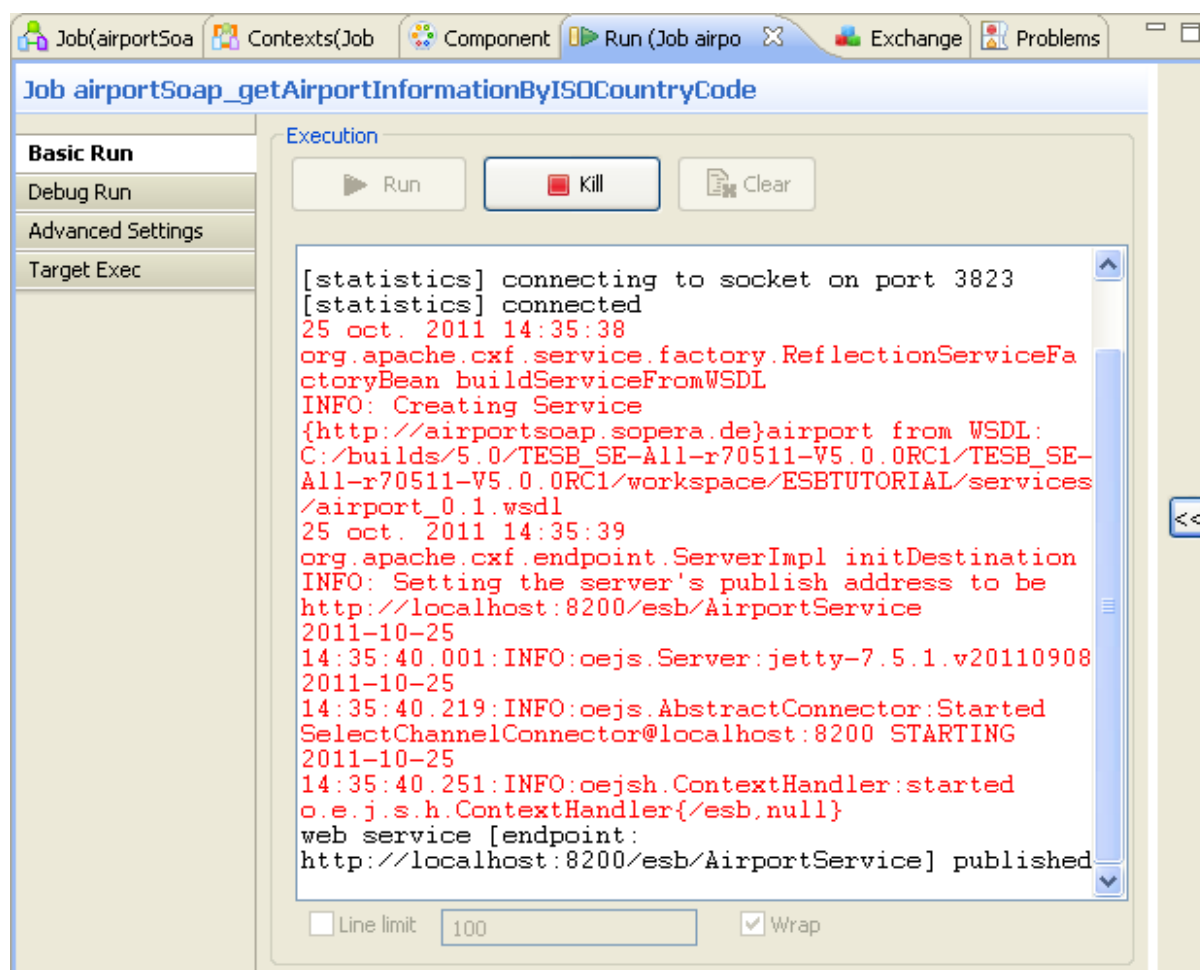


In the real-world practice, you can as well add hierarchical data for lookup. For further information, see *Talend Open Studio Components Reference Guide*

- Click **OK** to close the editor and validate this configuration.

Executing the Job

Press **F6** to run this Job and once launched, the **Run** view is opened for you to read the execution result.



The data service provider Job is executed, and will listen to all requests sent to the Web service until you click the **Kill** button to stop it as by default, the **Keep listening** option in the **Basic settings** view of **tESBProviderRequest** is selected automatically.

Now, you have to configure the consumer Job that will send actual requests to the data service provider Job you just created.

B.1.1.4. Building data service consumer

To build your consumer, you need at least these components: an input component allowing to read a data flow, a **tXMLMap** component that will map this flat data to a hierarchical document, the format expected by **ESB** components, the **tESBConsumer** components that will request the corresponding Web service and read its result and the **tLogRow** component that displays the Job execution result. For this specific scenario, you will use a **tFixedFlowInput** as input component to send a country code request to the **tESBConsumer** component.

To create the data service consumer Job that will actually send data to request the Web service, proceed as follows:

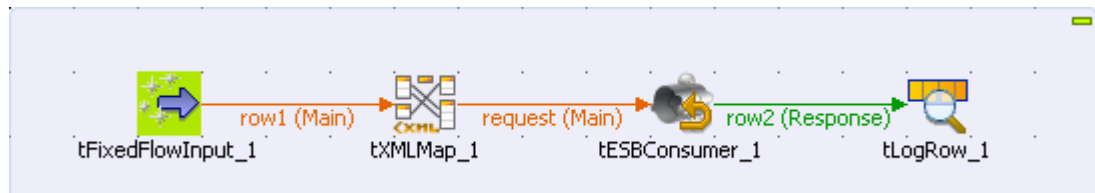
1. Right-click **Job Designs** in the **Repository** tree view and select **Create Job**.
2. In the dialog box displaying then, only the first field (**Name**) is required. Type in **WSDLConsumer** and click **Finish**.

Then an empty Job then opens on the main window and you can continue to create the Job of interest.

Dropping and linking the components

1. Click the **tFixedFlowInput** component from the **Misc** component group of the **Palette**, then click to the left of the design workspace to place it on the design area.
2. Do the same to drop **tXMLMap**, **tESBConsumer** and a **tLogRow** component from their respective group on the **Palette**.
3. To link the input components to the mapper, simply right-click **tFixedFlowInput**, hold and drop it to **tXMLMap**.
4. To link **tXMLMap** to **tESBConsumer**, right-click **tXMLMap**, hold and drag to **tESBConsumer**. A popup window displays, type in the name you want to give to the output row link: *request*, for example and then accept the propagation that prompts you to get the schema from **tESBConsumer**.
5. Link the **tESBConsumer** component to it with a **Response** row link.

Then the data service consumer Job should look like:



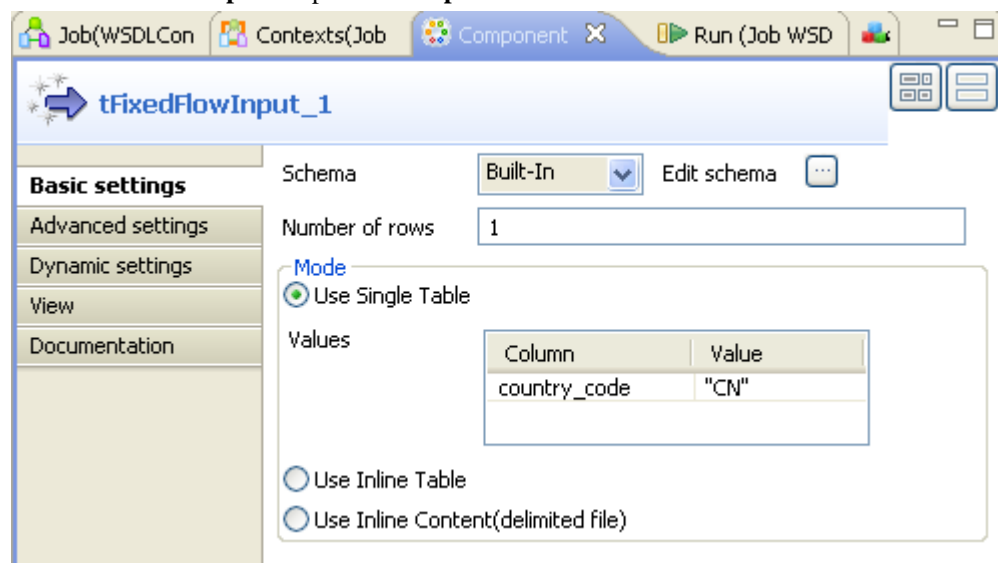
Then you need to configure each of these components.

Configuring the components

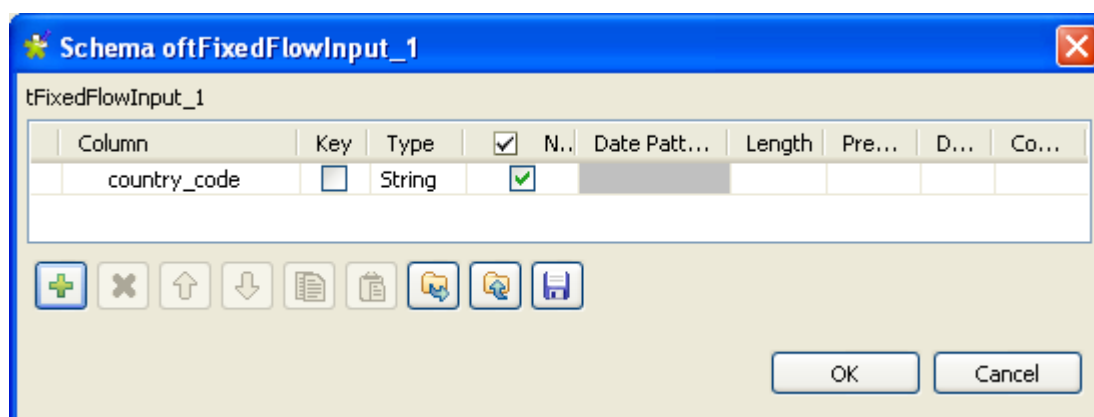
To do this, perform the following operations:

Procedure B.1. Configuring the tFixedflowInput component

1. Double-click **tFixedflowInput** to open its **Component** view.



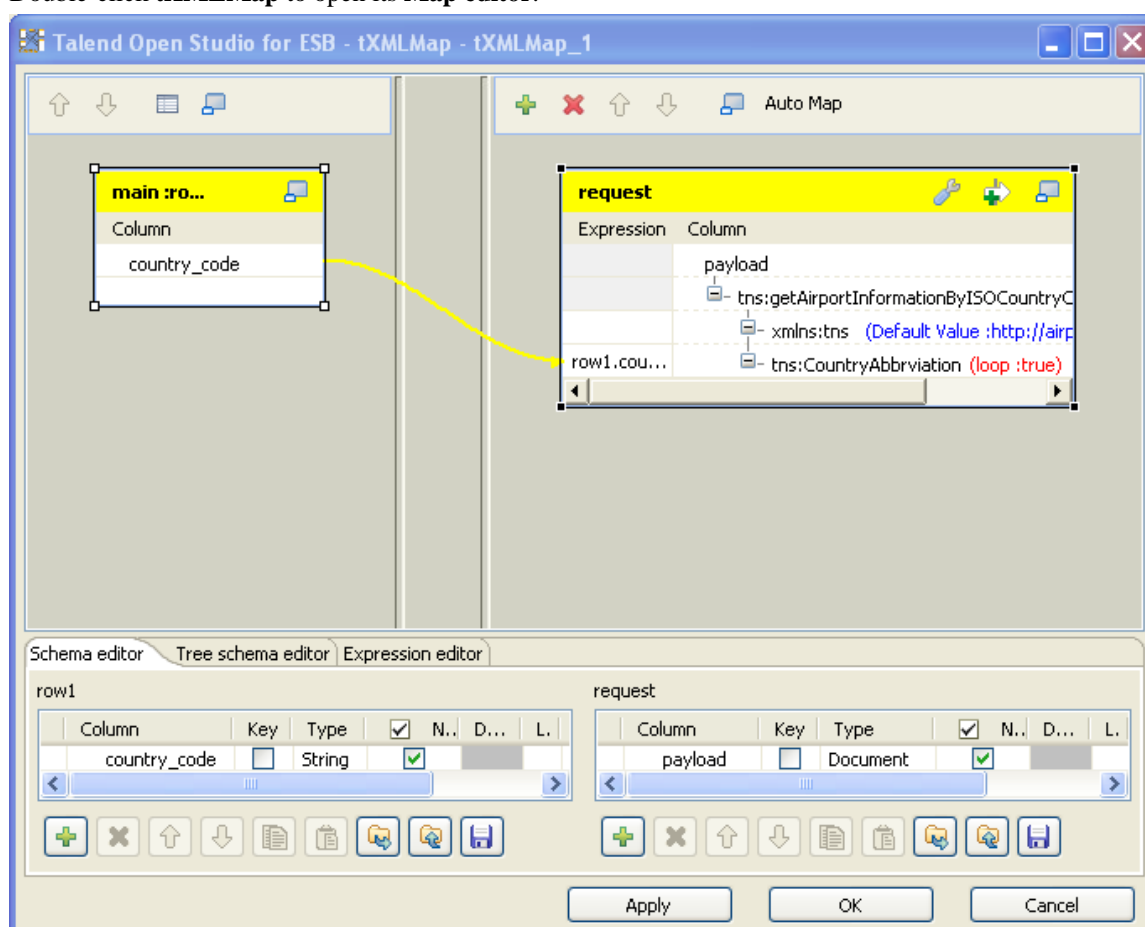
2. Click the three-dot button next to the **Edit schema** field to open the **[Schema]** window.



3. Click the plus button once to add one **Column** to the schema and name it *country_code*. Keep the **Type** field as *string*.
4. Click **OK** to validate this schema. Then in the **Mode** area of the **tFixedFlowInput** basic settings, the active option should be **Use Single Table** and the *country_code* row is already added automatically to the **Values** table
5. In the **Value** column of the **Values** table, type in CN within quotation marks.

Procedure B.2. Configuring the tXMLMap component

1. Double-click **tXMLMap** to open its **Map editor**.



2. To import the WSDL schema for the request, do the same as explained earlier when building up the data service provider. In this example, the request schema is *getAirportInformationByISOCountryCode*. For further information, see [the section called "Configuring tXMLMap"](#).

- Click *country_code* in the main flow table of the input side (left), hold **and** drop it to the **Expression** column corresponding to the *tns:CountryAbbreviation* node in the XML tree of the request table on the output side (right).

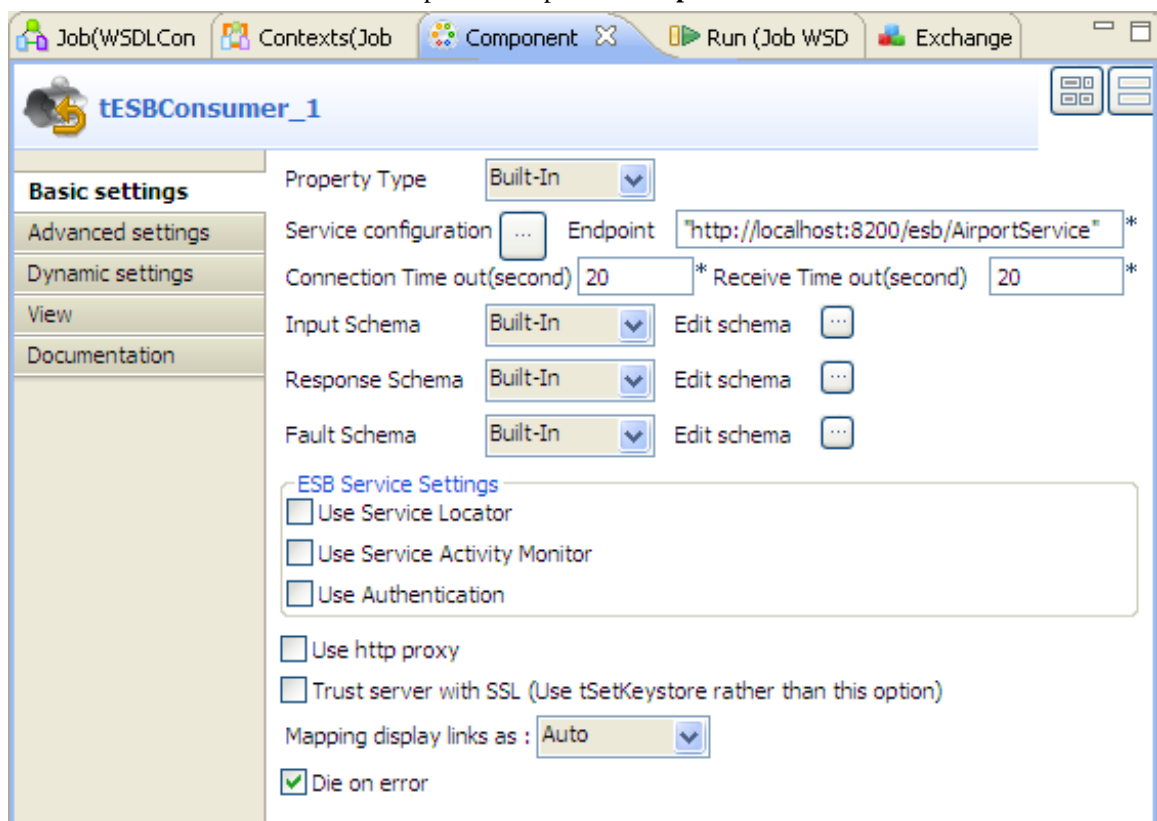


In the real-world operations, **tXMLMap** is able to handle the highly complex transformations of hierarchical data. For all of the available features of **tXMLMap**, see [Section 8.3, “tXMLMap operation”](#)

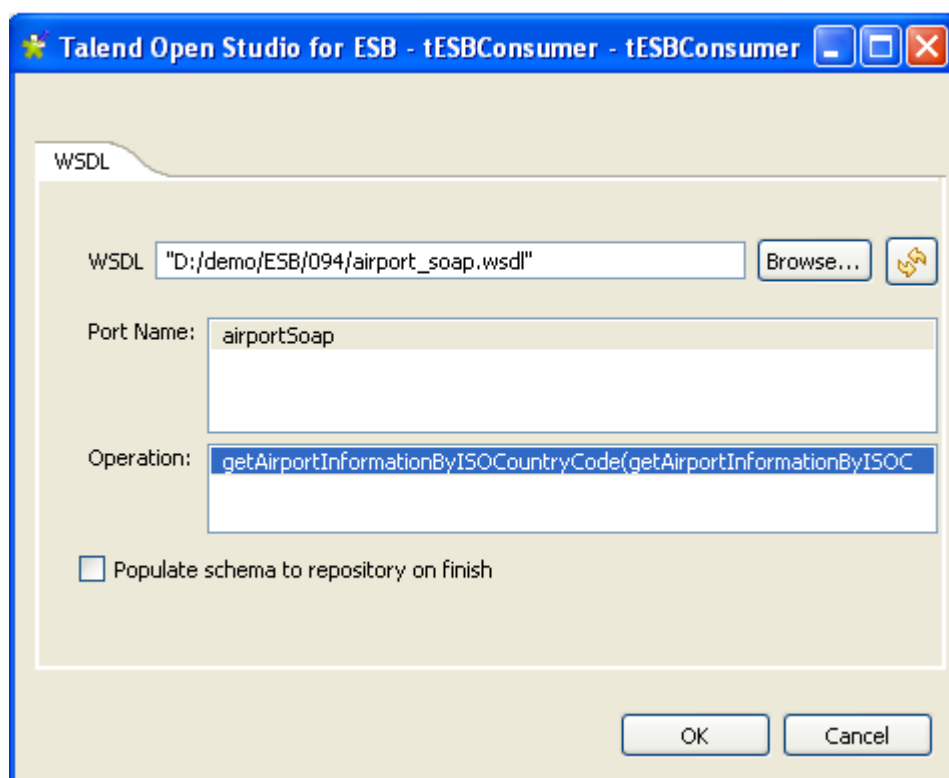
- Click **OK** to validate this configuration.

Procedure B.3. Configuring the tESBConsumer component

- Double-click the **tESBConsumer** component to open its **Component** view.



- Click the [...] button next to the **Service configuration** field to open the WSDL editor.
- In the editor, browse to the WSDL file provided, the **Port Name** and **Operation** fields are automatically filled in with the port(s) and method(s) available in the Web service.



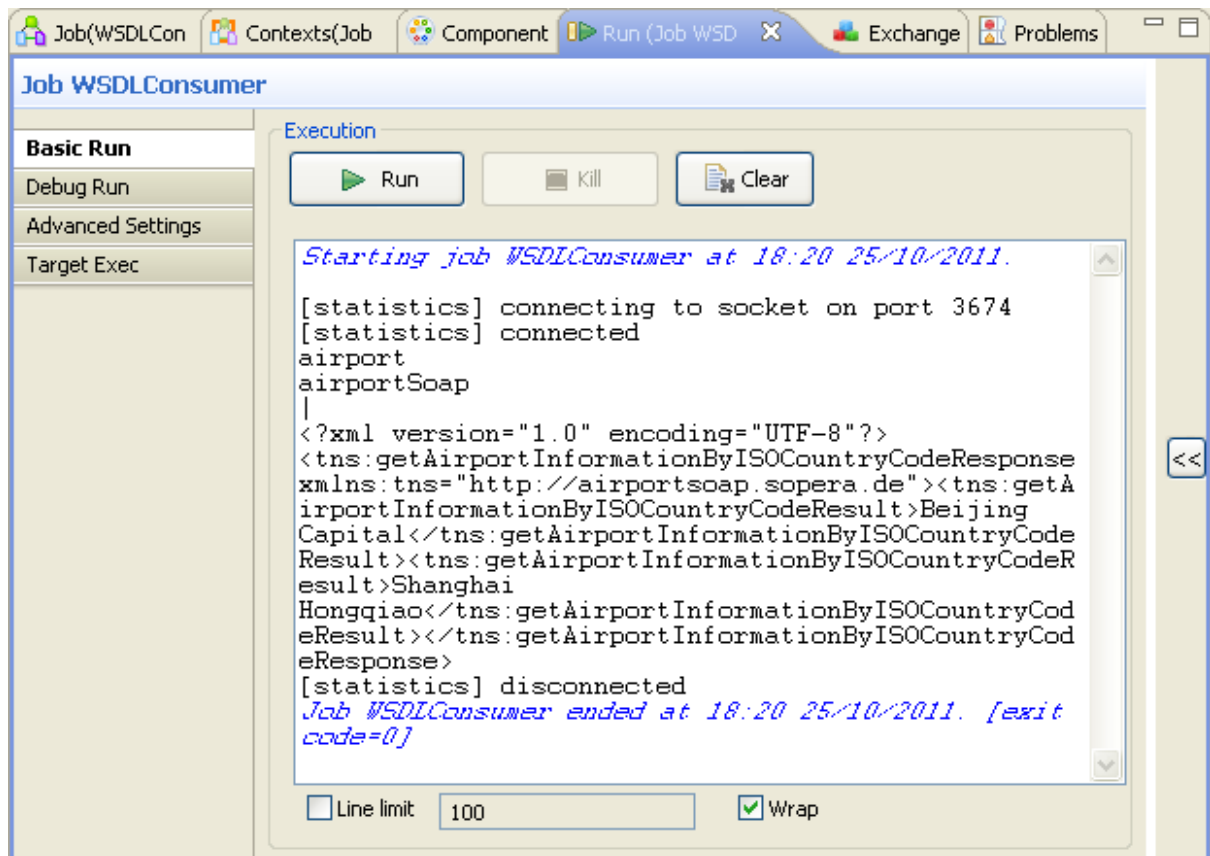
4. Select **airportSoap** in the **Port Name** field and select **getAirportInformationByISOCountryCode** in the **Operation** field.
5. Click **OK** to validate the configuration.

The **tLogRow** component will automatically retrieve the schema from the previous component. If not, double-click it and click the **Sync columns** button in its **Component** view.

Executing the Job

To execute this Job, press **F6**.

Once done, the **Run** view is opened automatically, where you can check the execution result.



This Job sends one country code request to the consumer that requests the Web service through the provider Job. The response is retrieved by the **tESBProviderResponse** and **tESBConsumer** components at the same time.

Note that although the provider Job received some request, it did not stop and is still listening to new requests.

For more scenario using **ESB** components, refer to *Talend Open Studio Components Reference Guide*.

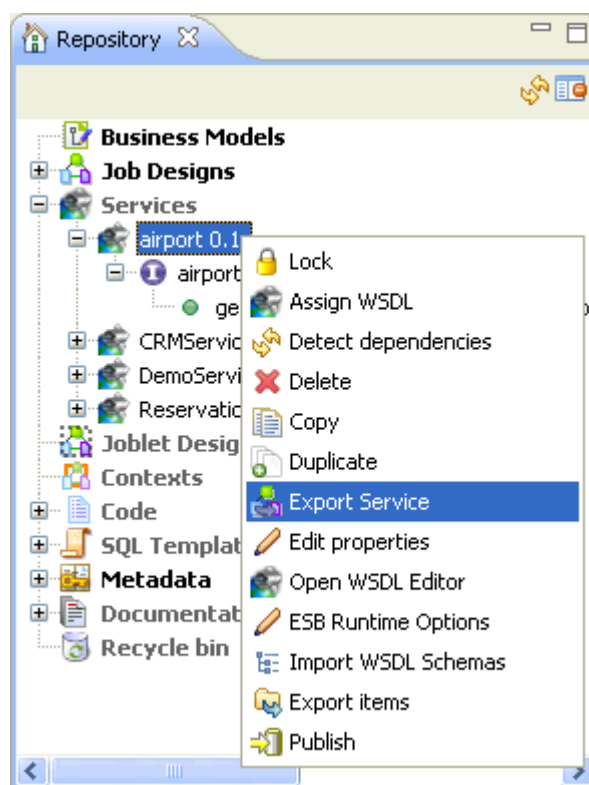
B.1.2. How to deploy a data service

In this section, we will export the data service we just created and deploy it in the Talend Runtime container.

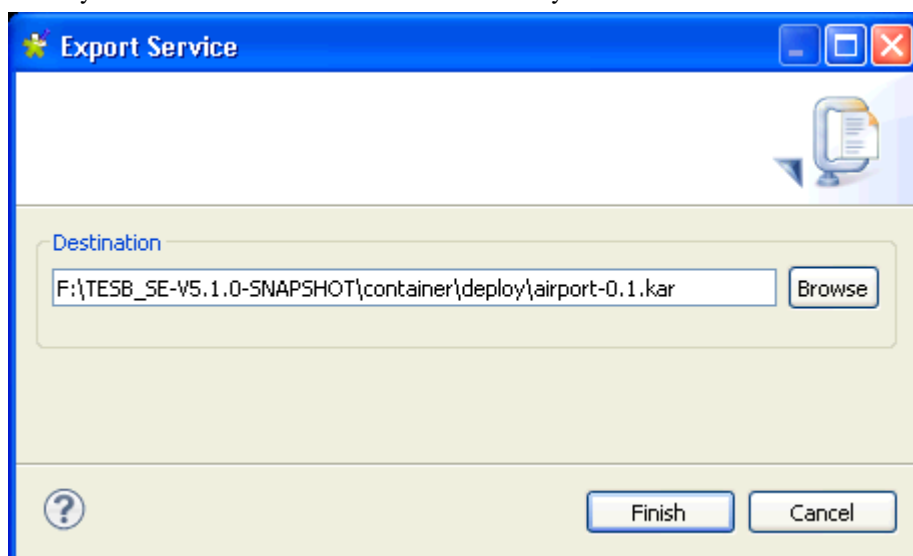
Before we start to export the service, start a Talend Runtime container first. For more information about how to install Talend ESB Runtime and how to run Talend Runtime container, see *Talend ESB Getting Started User Guide* and *Talend ESB Installation Guide*.

To export the data service for deployment, do the following:

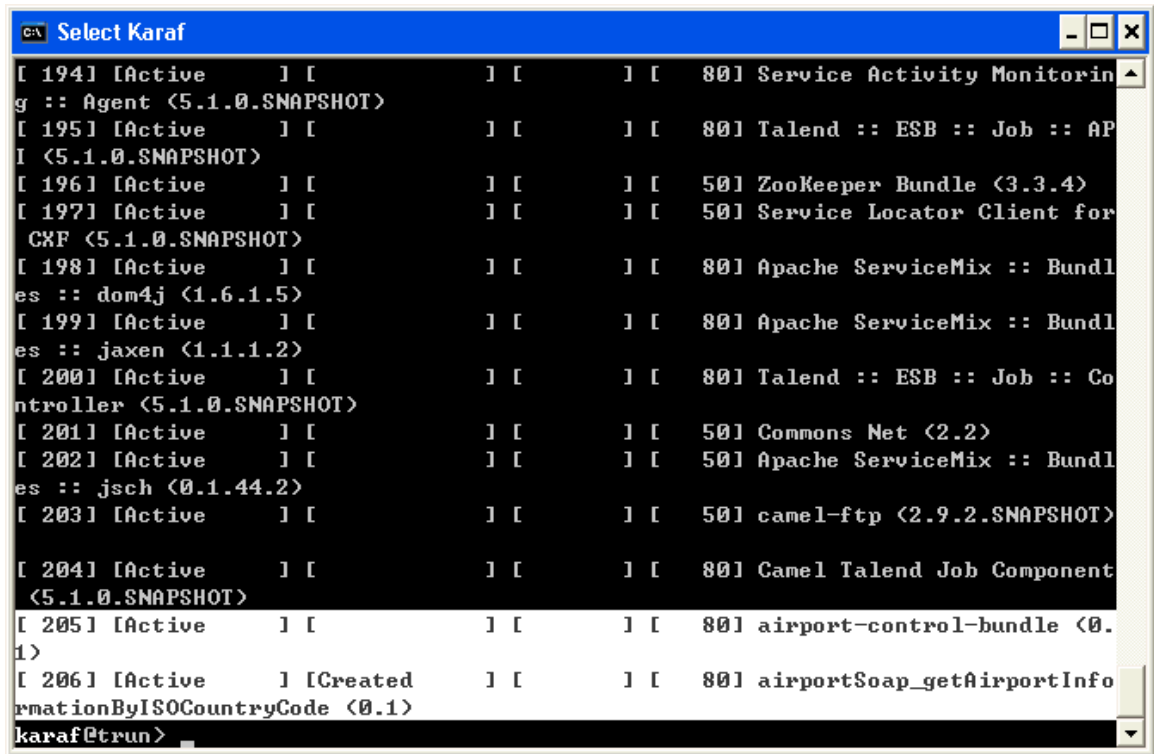
1. Under the **Services** node, right-click the **airport** service and select **Export service** in the contextual menu.



2. In the **Export Service** wizard, click **Browse** to navigate to the directory where you want to store the service and click **Finish**. In this use case, we export the service directly to the *deploy* folder within the Talend Runtime container directory so that the service starts to run immediately.



You can view this service in a container window by typing the `list` command.



B.1.3. How to build a Route using a data service

This section provides a scenario to illustrate how to build a Route in the Mediation perspective of *Talend Open Studio for ESB* using the data service we created in the previous section. For more information about how to access the Mediation perspective, see [Section 5.2, “Accessing the Mediation perspective”](#).

B.1.3.1. Discovering the scenario

In this scenario, we will define a Route using the Web service we just created, send a request to this Web service and retrieve the response from the Web service for a further use.

B.1.3.2. Starting the Apache ActiveMQ server

To send data to request the Web service, we use Apache ActiveMQ as the message broker. We need to launch the ActiveMQ server before executing the Route. For more information about installing and launching ActiveMQ server, see the Installation Guide available on the Talend ESB download page (<http://www.talend.com/download.php>) and the Apache Web site <http://activemq.apache.org/index.html>.

B.1.3.3. Building the Route

To build the Route, we need a **cJMSConnectionFactory** component to specify the connection factory for message handling, two **cJMS** components, one to read a JMS message from one queue and send the request to the Web service and the other to retrieve the response from the Web service and save it in another queue, and a **cCXF** component to connect to the Web service.

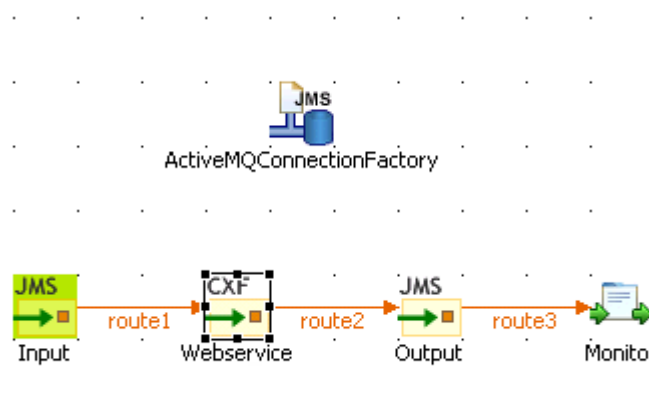
To create the Route, proceed as follows:

1. On the **Repository** tree view, right-click the **Route** node and from the contextual menu, select **Create Route**.

2. In the pop-up wizard, enter the information corresponding to the fields you need to complete. In this scenario, enter *airport* in the **Name** field and click **Finish**. The Route is opened in the design workspace.

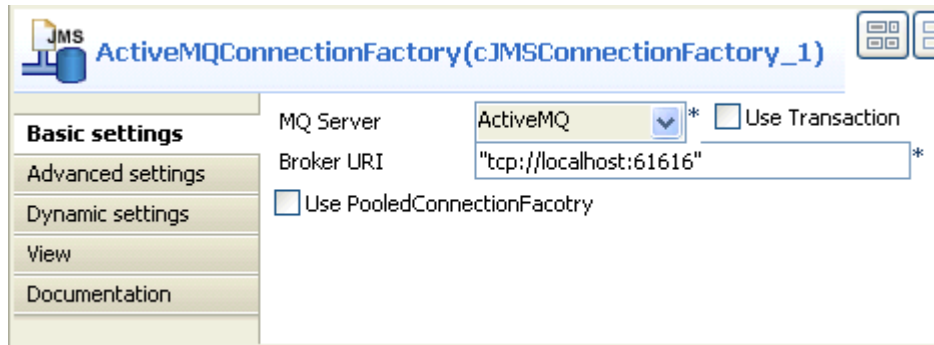
Dropping and linking the components

1. From the **Palette**, drop a **cJMSConnectionFactory**, a **cCXF**, a **cProcessor**, and two **cJMS** components on to the design workspace.
2. Label the components for better identification of their roles and link them using the **Row>Route** connection as shown below.



Configuring the components

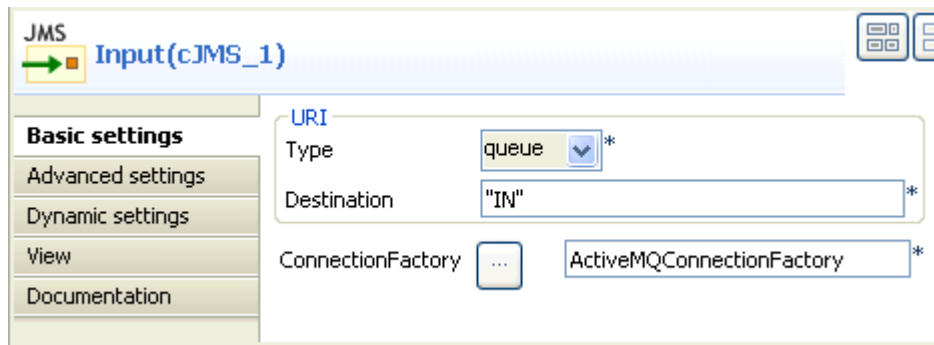
1. Double-click the **cJMSConnectionFactory** component to display its **Basic settings** view in the **Component** tab.



2. From the **MQ Server** list, select **ActiveMQ**.

In the **Broker URI** field, type in the URI of the local ActiveMQ server, "tcp://localhost:61616" in this use case.

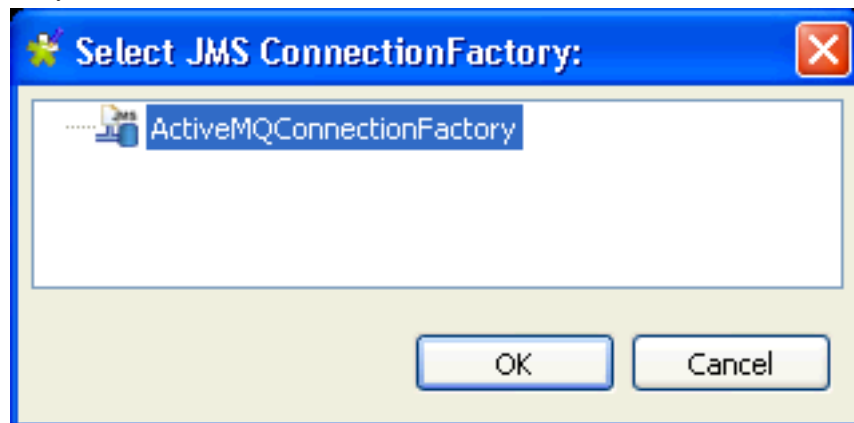
3. Double-click the **cJMS** component labeled *Input* to display its **Basic settings** view.



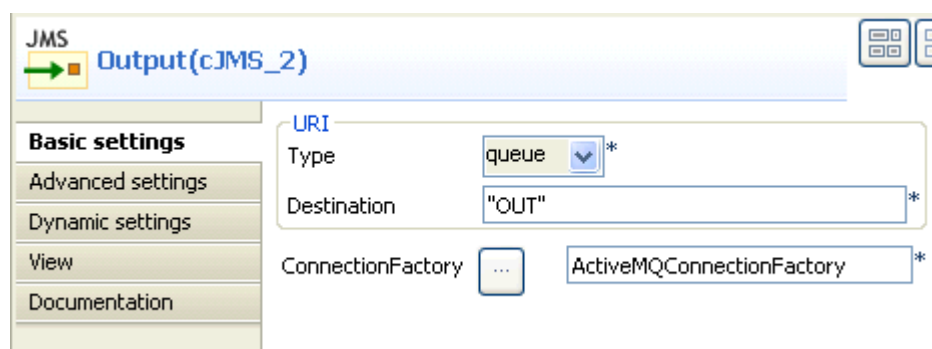
4. From the **Type** list, select **queue** to send the messages to a JMS queue.

In the **Destination** field, type in a name for the JMS queue, "IN" in this use case.

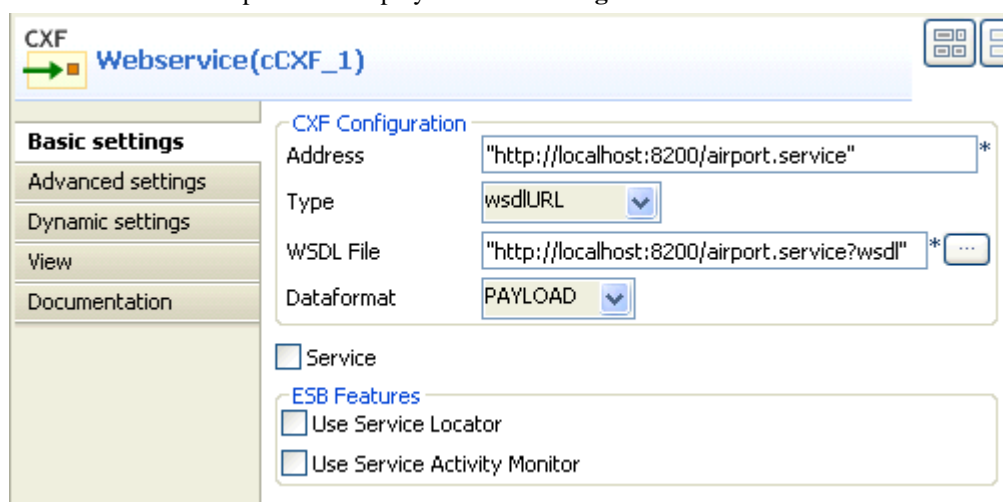
Double-click the [...] button next to **ConnectionFactory**. Select the JMS connection factory that you have just configured in the dialog box and click **OK**. You can also enter the name of the **cJMSConnectionFactory** component directly in the field.



5. Repeat this operation to configure the **cJMS** component labeled *Output* with the same **ConnectionFactory** and name the destination JMS queue *OUT*.



6. Double-click the **cCXF** component to display its **Basic settings** view.



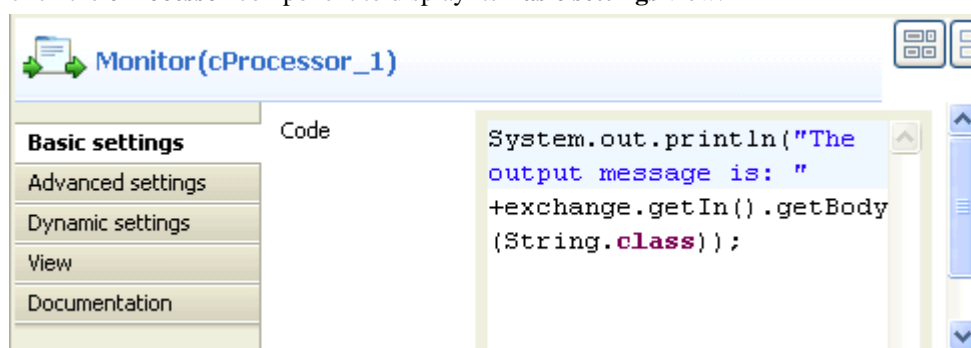
7. In the **Address** field, type in the service endpoint URL of the Web service that we have created, *http://localhost:8200/airport.service* in this example.

From the **Type** list, select **wsdlURL**.

In the **WSDL File** field, type in the path to the WSDL file, *http://localhost:8200/airport.service?wsdl* in this example.

From the **Dataformat** list, select **PAYLOAD**.

8. Double-click the **cProcessor** component to display its **Basic settings** view.



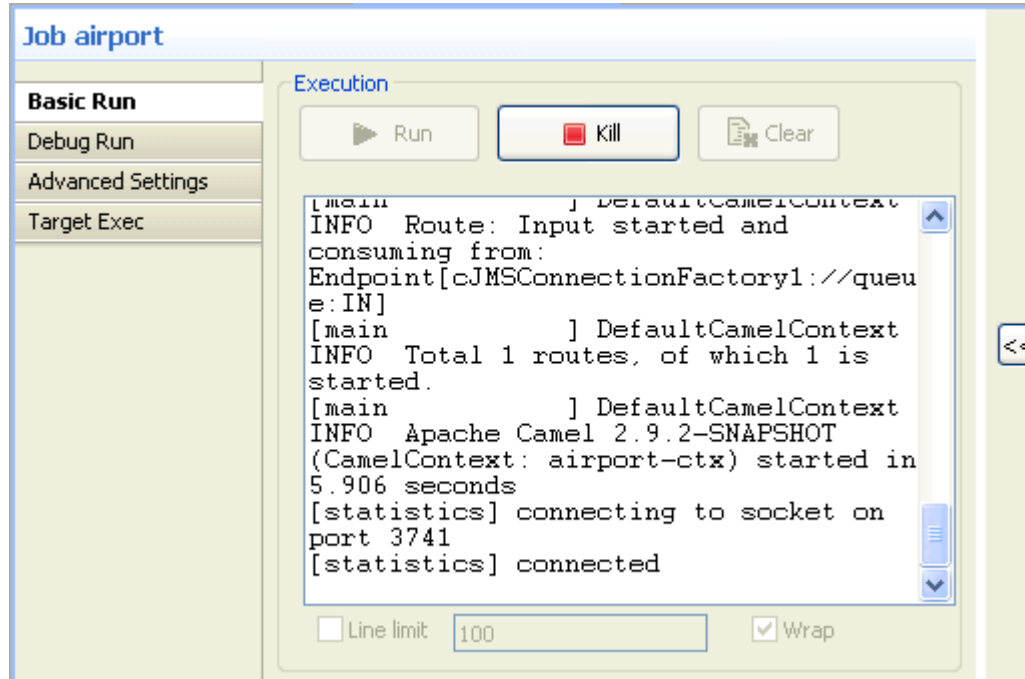
9. In the **Code** area, customize the code as shown below to display the message body on the **Run** console.

```
System.out.println("The output message is: " +
exchange.getIn().getBody(String.class));
```


Executing the Route

1. Click the **Run** view to display it and click the **Run** button to launch the execution of the Route. You can also press **F6** to execute it.

RESULT: The Route is successfully started.



2. Switch to the ActiveMQ Web console. The incoming queue is already created.

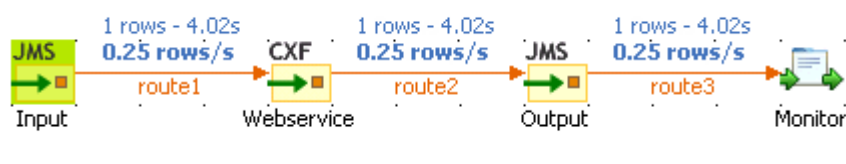


3. Enter the message body as shown below in the incoming queue and send the message.

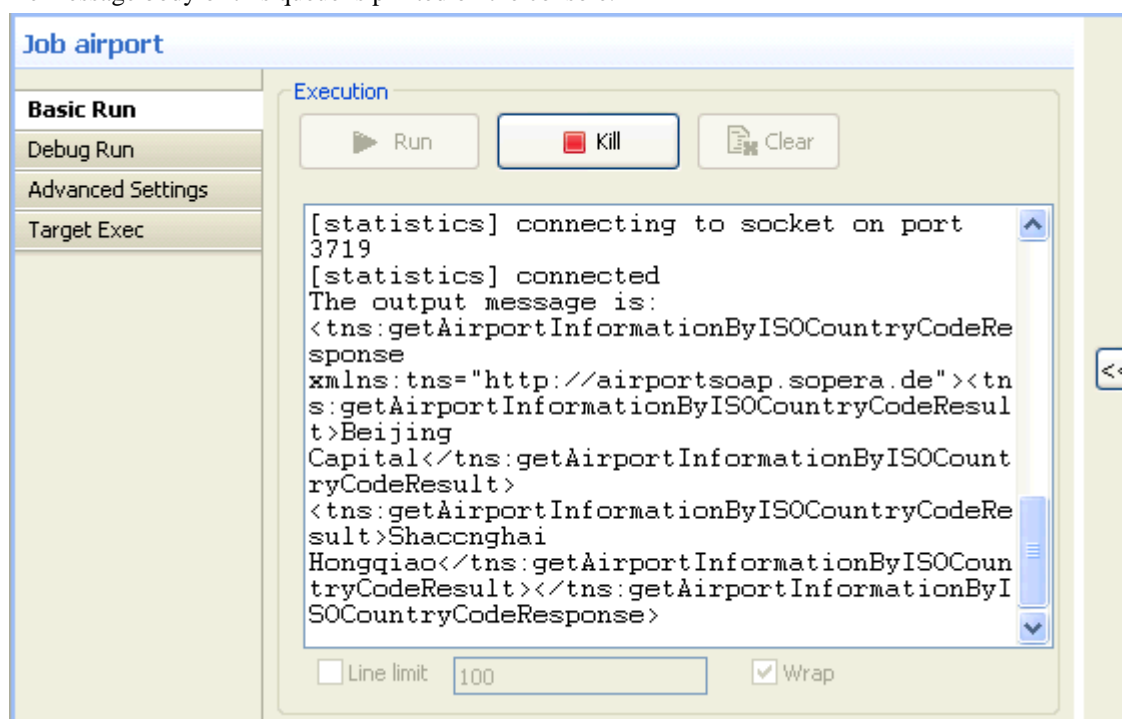
Message body

```
<air:getAirportInformationByISOCountryCode>
  <air:CountryAbbreviation>CN</air:CountryAbbreviation>
</air:getAirportInformationByISOCountryCode>
```

4. In the Route builder, we can see that the message payload is sent to the Web service. The Web service get called and the response is sent to another queue, *OUT* as we have configured.







The message body of this queue is printed on the console.



We can also view the *OUT* queue from the ActiveMQ Web console. It has one message in it, as shown below.

Queues

Name	Number Of Pending Messages	Number Of Consumers	Messages Enqueued	Messages Dequeued	Views	Operations
IN	0	1	1	1	Browse Active Consumers  	Send To Purge Delete
OUT	1	0	1	0	Browse Active Consumers  	Send To Purge Delete

B.2. Job and Route example

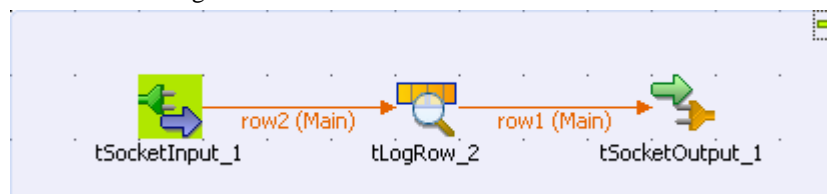
This section provides a scenario of how to use the Camel components not included in the palette in a Route and to use sockets for connecting a Job and a Route.

B.2.1. Discovering the scenario

In this scenario, we will build a DI Job in the **Integration** perspective that listens messages from one socket port and sends out the messages to another port. Then we will build a Route in the **Mediation** perspective that sends a test message to the listening port and logs message changes of the incoming and outgoing ports.

B.2.2. Building a DI Job

1. From the **Integration** perspective of the Studio, create a Job and open it in the design workspace. For more information on how to do it, see [Section 4.2.1, “How to create a Job ”](#).
2. From the palette, drag and drop a **tSocketInput**, a **tSocketOutput**, and a **tLogRow** component onto the design workspace. Link them together with the **Row > Main** connection.



3. Double-click the **tSocketInput** component to open its **Basic settings** view in the **Component** tab.

tSocketInput_1

☐ Bind host name (when more than 1 IP address in this sever side)

Basic settings

Port: 8900

Timeout: 30000 ☐ Uncompress ☐ Die on error

Row separator: "\\n" Field separator: ";"

Escape Char: "\"" Text Enclosure: "\""

Schema: Built-In Edit schema

Encoding: ISO-8859-15

- Define the listening Port number in the **Port** field, 8900 for example. By default, the local host server is used. Set the amount of time (in seconds) to 30000 in the **Timeout** field, after which the Job will time out.
- Double-click the **tSocketOutput** component to open its **Basic settings** view in the **Component** tab.

tSocketOutput_1

Host: "127.0.0.1" Port: 8901

☐ Compress Retry Times: 10 Timeout: 10000

Row Separator: "\\n" Field Separator: ";"

Escape char: "\"" Text enclosure: "\""

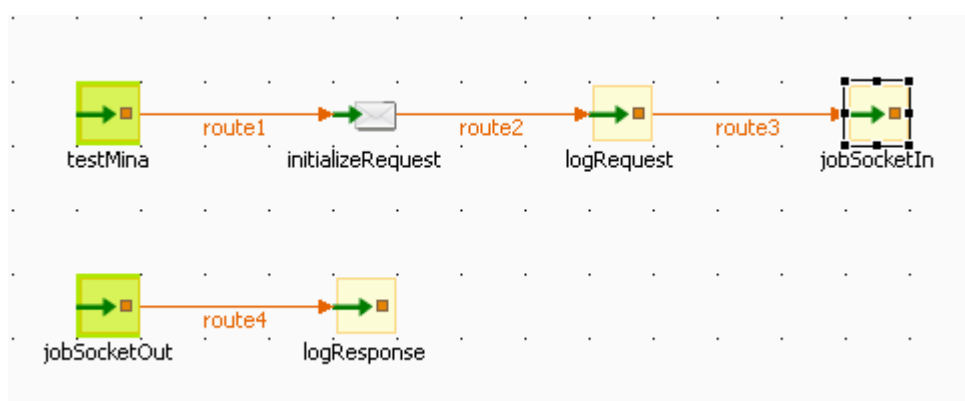
Schema: Built-In Edit schema Sync columns

Encoding: ISO-8859-15

- Define the **Host** IP address and the **Port** number where the data will be passed on to, as shown above.
- The **tLogRow** component is used to monitor data processed and does not need any configuration.
- Press **Ctrl+S** to save your Job.

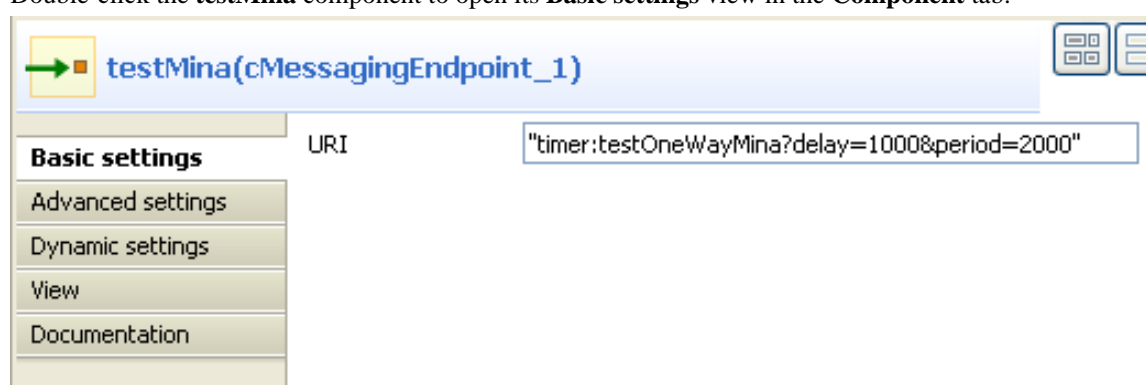
B.2.3. Building a Route

- Switch to the **Mediation** perspective. Create a Route and open it in the design workspace. For more information on how to do it, see [Section 5.3.1, "How to create a Route "](#).
- From the palette, drag and drop a **cSetBody**, and five **cMessagingEndpoint** components onto the design workspace. Label the components for better identification of their roles and link them together with the **Row** > **Main** connection.

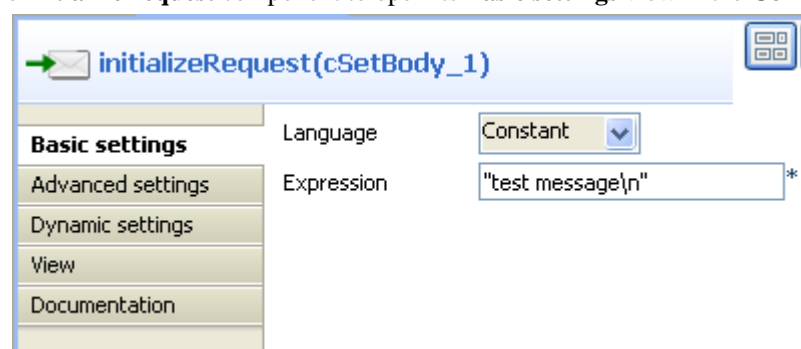


Procedure B.4. Configuring the incoming sub-route

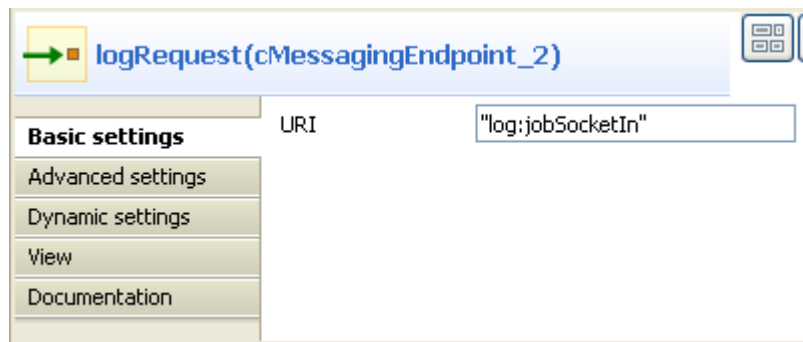
1. Double-click the **testMina** component to open its **Basic settings** view in the **Component** tab.



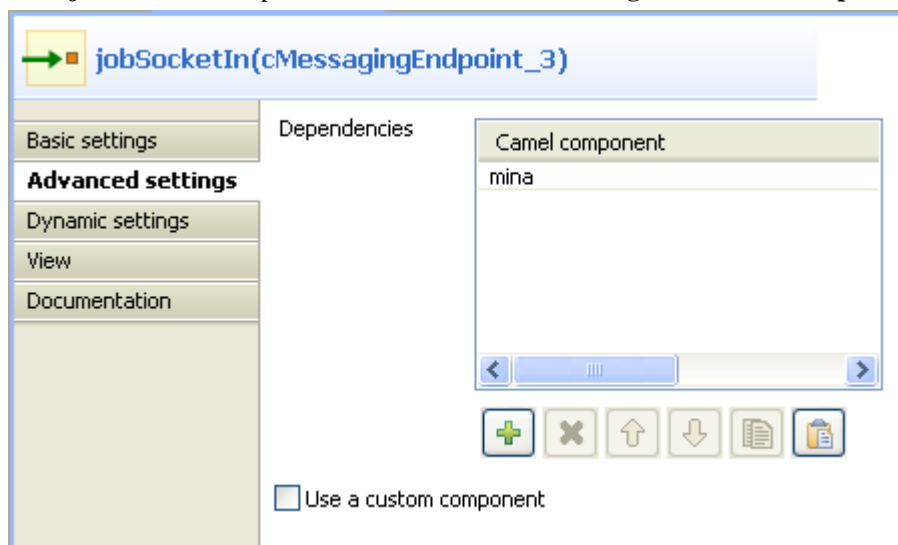
2. In the **URI** field, enter the code "timer:testOneWayMina?delay=1000&period=2000" to define a timer for starting message exchanges. In this use case, we want each message to be delivered after a 1-second delay at a period of 2 seconds.
3. Double-click the **initializeRequest** component to open its **Basic settings** view in the **Component** tab.





4. Select **Constant** from the **Language** list box and type in "test message\n" in the **Expression** field.
5. Double-click the **logRequest** component to open its **Basic settings** view in the **Component** tab.

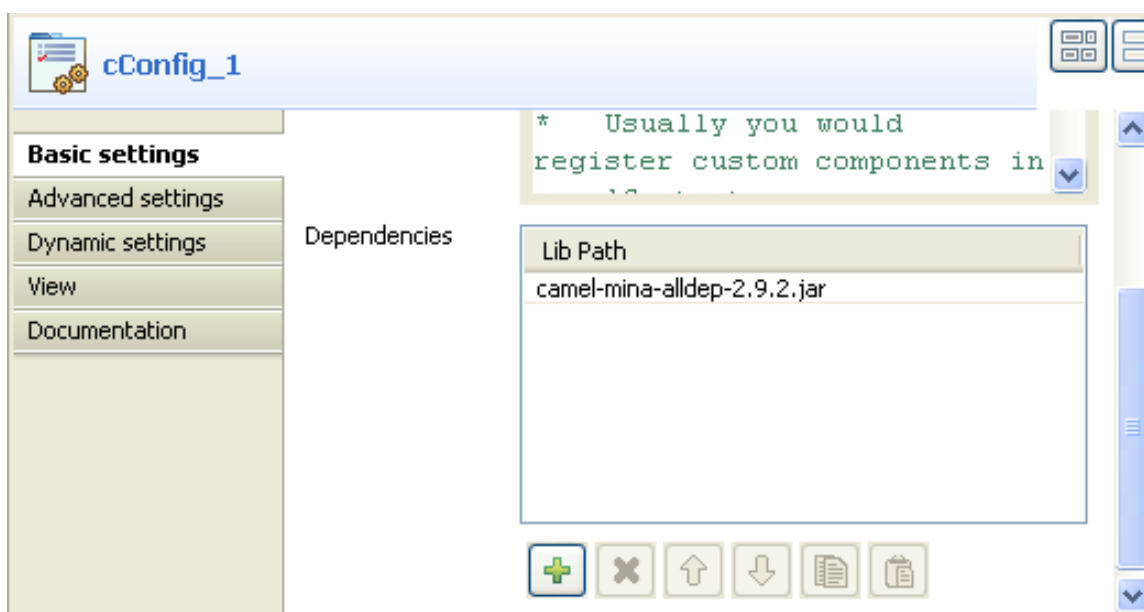


6. In the **URI** field, enter "log:jobSocketIn" where the incoming message exchanges are logged.
7. Double-click the **jobSocketIn** component and click **Advanced settings** view in the **Component** tab.

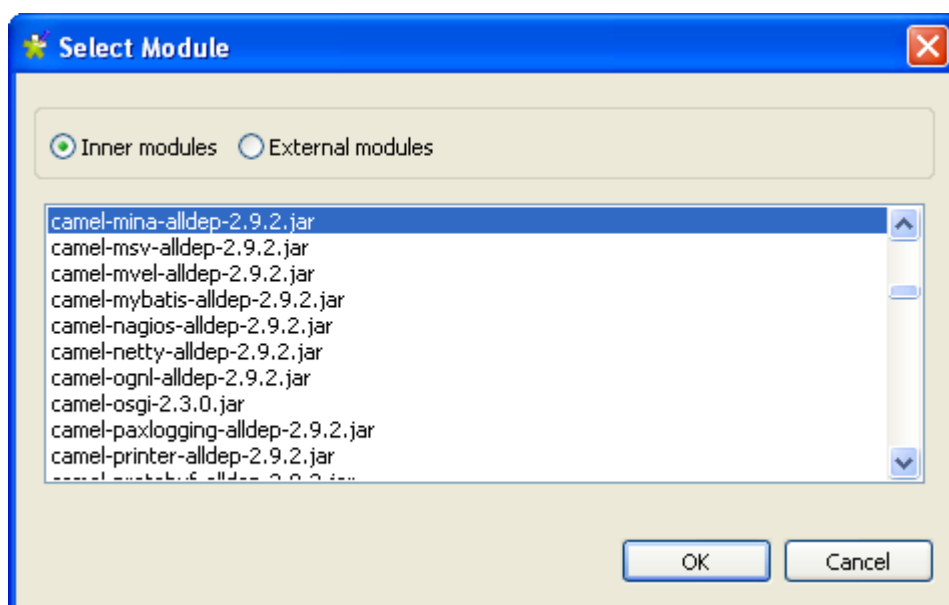


8. In this use case, we will use the Camel component camel-mina as the transport. To use this component, click  at the bottom of the **Dependencies** list to add a row and select `mina` from the drop-down list. For more information about Mina, see the site <http://camel.apache.org/mina.html>.

Alternatively, you can use a **cConfig** component and add the library of MINA to the **Dependencies** list of the **cConfig** component. To do so, click  at the bottom of the **Dependencies** list to add a row. Select this row and click the [...] button at the end to show the **Select Module** wizard.



Select camel-mina-alldep-2.9.2.jar from the inner modules and click **OK** to add it to the **Dependencies** list.



9. Click the **Basic settings** view in the **Component** tab of the **jobSocketIn** component. In the **URI** field, enter "mina:tcp://localhost:" + 8900 + "?textline=true&sync=false" to send the message to the Mina endpoint of a TCP service on port 8900 as a text line in the InOnly mode.

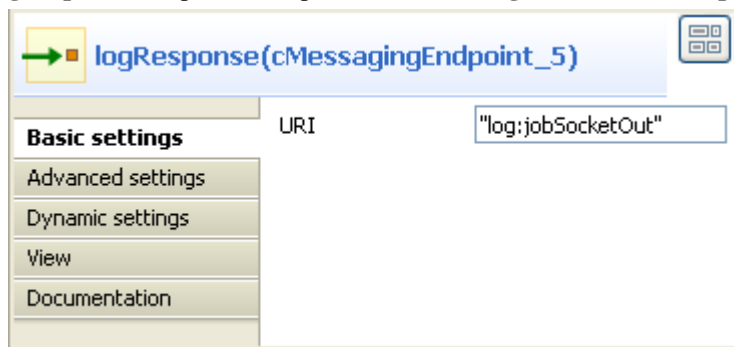


Procedure B.5. Configuring the outgoing sub-route

1. Double-click the **jobSocketOut** component to open its **Basic settings** view in the **Component** tab.



2. In the **URI** field, enter "mina:tcp://localhost:" + 8901 + "?textline=true&sync=false" of the outgoing socket port.
3. Double-click the **logResponse** component to open its **Basic settings** view in the **Component** tab.



4. In the **URI** field, enter "log:jobSocketOut" where the outgoing message exchanges are logged.

Press **Ctrl+S** to save your Route.

B.2.4. Viewing the code and executing the Route and the Job

1. Click the **Code** tab at the bottom of the design workspace to check the generated code.


```

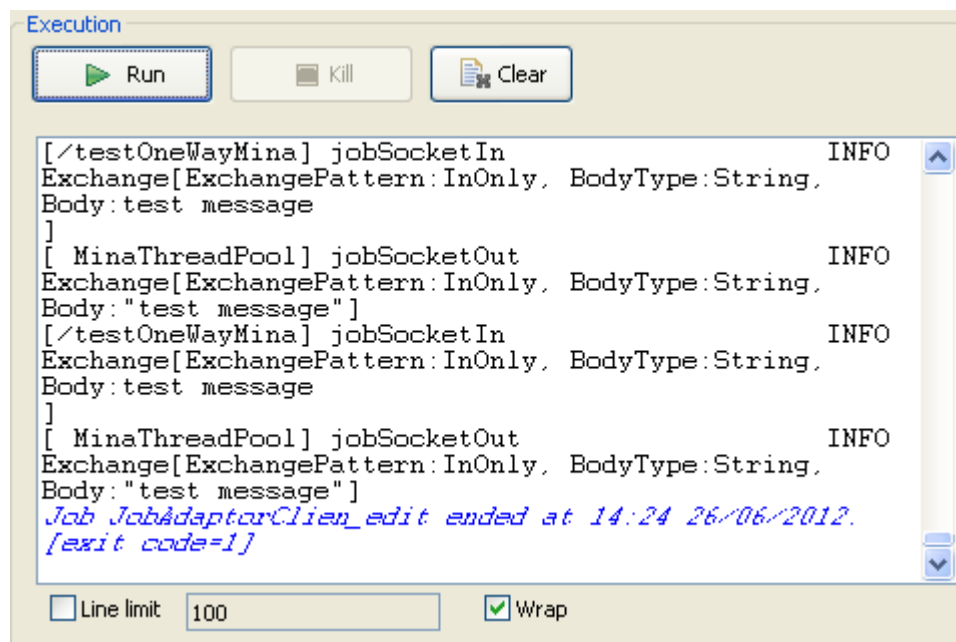
public void initRoute() throws Exception {
    routeBuilder = new org.apache.camel.builder.RouteBuilder() {
        public void configure() throws Exception {
            from(uriMap.get("testMina")).routeId("testMina")
                .setBody().constant("test message\n").id(
                    "cSetBody_1").to(
                    uriMap.get("logRequest")).id(
                    "cMessagingEndpoint_2").to(
                    uriMap.get("jobSocketIn")).id(
                    "cMessagingEndpoint_3");
            from(uriMap.get("jobSocketOut"))
                .routeId("jobSocketOut").to(
                    uriMap.get("logResponse")).id(
                    "cMessagingEndpoint_5");
        }
    };
    getCamelContexts().get(0).addRoutes(routeBuilder);
}

```

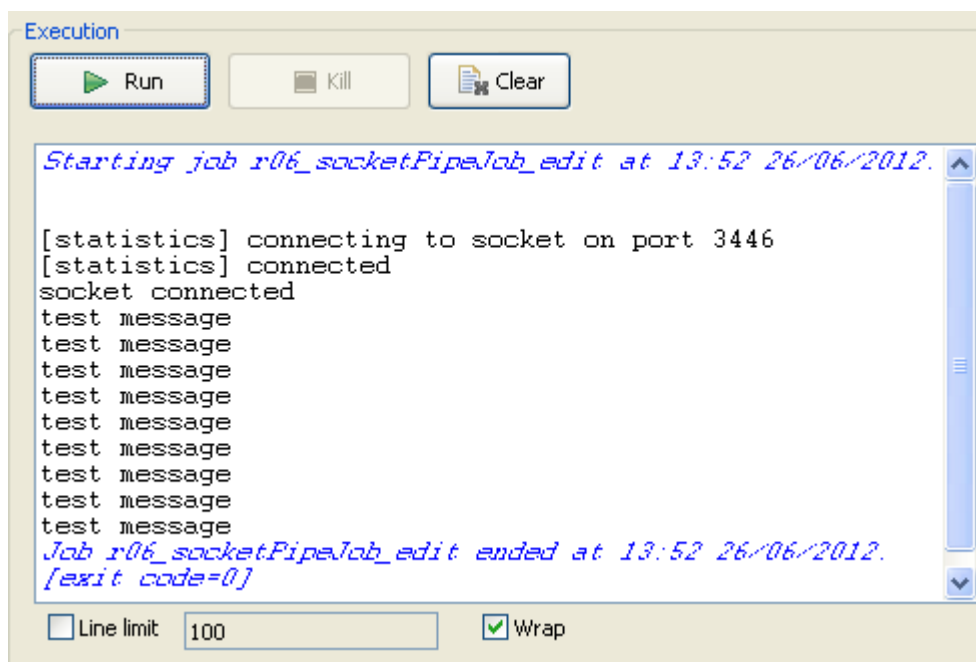
As shown above, the message flow from testMina is given a payload by cSetBody and then sent to logRequest and jobSocketIn. The other message flow is sent from jobSocketOut to logResponse.

2. Press **F6** to execute the Route, and then run the Job in the same way. The Route will keep trying to connect to the defined ports until the Job starts.

RESULT: The message change log of the incoming and outgoing ports is printed in the execution console of the Route.



On the Job side, the message processed is also displayed in console.





Appendix C. SQL template writing rules

This chapter describes the rules applied for the creation of SQL templates. It aims to help users of SQL templates in *Talend Open Studio for ESB* to understand and develop the SQL templates for more customized usage.

These rules provide details that you have to respect when writing the template statement, a comment line or the different relevant syntaxes.

These rules help to use the SQL code in specific use cases, such as to access the various parameters defined in components.

C.1. SQL statements

An SQL statement can be any valid SQL statement that the related JDBC is able to execute. The SQL template code is a group of SQL statements. The basic rules to write an SQL statement in the SQL template editor are:

- An SQL statement must end with ;.
- An SQL statement can span lines. In this case, no line should be ended with ; except the last one.

C.2. Comment lines

A comment line starts with # or --. Any line that starts with # or -- will be ignored in code generating.



There is no exception to the lines in the middle part of a SQL statement or within the `<% . . . %>` syntax.

C.3. The `<% . . . %>` syntax

This syntax can span lines. The following list points out what you can do with this syntax and what you should pay attention to.

- You can define new variables, use Java logical code like `if`, `for` and `while`, and also get parameter values.

For example, if you want to get the `FILE_Name` parameter, use the code as follows:

```
<%  
String filename = __FILE_NAME__;  
%>
```

- This syntax cannot be used within an SQL statement. In other words, it should be used between two separated SQL statements.

For example, the syntax in the following code is valid.

```
#sql sentence  
DROP TABLE temp_0;  
<%  
#loop  
for(int i=1; i<10; i++){  
%>  
#sql sentence  
DROP TABLE temp_<%=i %>;  
<%  
}  
%>  
#sql sentence  
DROP TABLE temp_10;
```

In this example, the syntax is used between two separated SQL templates: `DROP TABLE temp_0;` and `DROP TABLE temp_<%=i %>;`.

The SQL statements are intended to remove several tables beginning from `temp_0`. The code between `<%` and `%>` generate a sequence of number in loop to identify tables to be removed and close the loop after the number generation.

- Within this syntax, the <%= . . . %> or </ . . . /> syntax should not be used.

<%= . . . %> and </ . . . /> are also syntax intended for the SQL templates. The below sections describe related information.



Parameters that the SQL templates can access with this syntax are simple. They are often used for connection purpose and can be easily defined in components, such as *TABLE_NAME*, *DB_VERSION*, *SCHEMA_TYPE*, etc.

C.4. The <%= . . . %> syntax

This syntax cannot span lines and is used for SQL statement. The following list points out what you can do with this syntax and what you should pay attention to.

- This syntax can be used to generate any variable value, and also the value of any existing parameter.
- No space char is allowed after <%=.
- Inside this syntax, the <%...%> or </.../> syntax should not be used.

The statement written in the below example is a valid one.

```
#sql sentence
DROP TABLE temp_<%=__TABLE_NAME__ %>;
```

The code is used to remove the table defined through an associated component.

For more information about what components are associated with the SQL templates, see [Chapter 4, Designing a Job](#).

For more information on the <%= . . . %> syntax, see [Section C.3, “The <%= . . . %> syntax”](#).

For more information on the </ . . . /> syntax, see the following section.



Parameters that the SQL templates can access with this syntax are simple. They are often used for connection purpose and can be easily defined in components, such as *TABLE_NAME*, *DB_VERSION*, *SCHEMA_TYPE*, etc.

C.5. The </ . . . /> syntax

This syntax cannot span lines. The following list points out what you can do with this syntax and what you should pay attention to.

- It can be used to generate the value of any existing parameter. The generated value should not be enclosed by quotation marks.
- No space char is allowed after </ or before />.
- Inside this syntax, the <%...%> or <%=...%> syntax should not be used.

The statement written in the below example is a valid one.

```
#sql sentence
DROP TABLE temp_</TABLE_NAME/>;
```

The statement identifies the *TABLE_NAME* parameter and then removes the corresponding table.

For more information on the `<%...%>` syntax, see [Section C.3, “The `<% . . %>` syntax”](#).

For more information on the `<%=...%>` syntax, see [Section C.4, “The `<%= . . %>` syntax”](#).

The following sections present more specific code used to access more complicated parameters.



Parameters that the SQL templates can access with this syntax are simple. They are often used for connection purpose and can be easily defined in components, such as `TABLE_NAME`, `DB_VERSION`, `SCHEMA_TYPE`, etc.

C.6. Code to access the component schema elements

Component schema elements are presented on a schema column name list (delimited by a dot “.”). These elements are created and defined in components by users.

The below code composes an example to access some elements included in a component schema. In the following example, the `ELT_METADATA_SCHEMA` variable name is used to get the component schema.

```
<%  
String query = "select "  
SCHEMA(__ELT_METADATA_SCHEMA__);  
for (int i=0; i < __ELT_METADATA_SCHEMA__.length ; i++) {  
query += (__ELT_METADATA_SCHEMA__[i].name + ",");  
}  
query += " from " + __TABLE_NAME__;  
%>  
<%=query %>;
```

In this example, and according to what you want to do, the `__ELT_METADATA_SCHEMA__[i].name` code can be replaced by `__ELT_METADATA_SCHEMA__[i].dbType`, `__ELT_METADATA_SCHEMA__[i].isKey`, `__ELT_METADATA_SCHEMA__[i].length` or `__ELT_METADATA_SCHEMA__[i].nullable` to access the other fields of the schema column.

The `extract` statement is `SCHEMA(__ELT_METADATA_SCHEMA__);`. In this statement, `ELT_METADATA_SCHEMA` is the variable name representing the schema parameter to be extracted. The variable name used in the code is just an example. You can change it to another variable name to represent the schema parameter you already defined.



Make sure that the name you give to the schema parameter does not conflict with any name of other parameters.

For more information on component schema, see [Section 4.2.6.1, “Basic Settings tab”](#).

C.7. Code to access the component matrix properties

The component matrix properties are created and changed by users according to various data transformation purposes. These properties are defined by tabular parameters, for example, the *operation* parameters or *groupby* parameters that users can define through the **tSQLTemplateAggregate** component.

To access these tabular parameters that are naturally more flexible and complicated, two approaches are available:

- The `</.../>` approach:

`</.../>` is one of the syntax used by the SQL templates. This approach often needs hard coding for every parameter to be extracted.

For example, a new parameter is created by user and is given the name `NEW_PROPERTY`. If you want to access it by using `</NEW_PROPERTY/>`, the below code is needed.

```
else if (paramName.equals("NEW_PROPERTY")) {
    List<Map<String, String>> newPropertyTableValue = (List<Map<String, String>>)
    ElementParameterParser.getObjectValue(node, "__NEW_PROPERTY__");
    for (int ii = 0; ii < newPropertyTableValue.size(); ii++) {
        Map<String, String> newPropertyMap = newPropertyTableValue.get(ii);
        realValue += ...; //append generated codes
        .....
    }
}
```

- The `EXTRACT(__GROUPBY__)` ; approach:

The below code shows the second way to access the tabular parameter (`GROUPBY`).

```
<%
String query = "insert into " + __TABLE_NAME__ + "(id, name, date_birth) select sum(id), name, date_birth
from cust_teradata group by";
EXTRACT(__GROUPBY__);
for (int i=0; i < __GROUPBY_LENGTH__ ; i++) {
    query += (__GROUPBY_INPUT_COLUMN__[i] + " ");
}
%>
<%=query %>;
```

When coding the statements, respect the rules as follows:

- The extract statement must use `EXTRACT(__GROUPBY__)`; . Upcase should be used and no space char is allowed. This statement should be used between `<%` and `%>`.
- Use `__GROUPBY_LENGTH__`, in which the parameter name is followed by `_LENGTH`, to get the line number of the tabular `GROUPBY` parameters you define in the **Groupby** area on a **Component** view. It can be used between `<%` and `%>` or `<%=` and `%>`.
- Use code like `__GROUPBY_INPUT_COLUMN__[i]` to extract the parameter values. This can be used between `<%` and `%>` or between `<%=` and `%>`.
- In order to access the parameter correctly, do not use the identical name prefix for several parameters. For example in the component, avoid to define two parameters with the names `PARAMETER_NAME` and `PARAMETER_NAME_2`, as the same prefix in the names causes erroneous code generation.

Appendix D. System routines

This appendix gives you an overview of the most commonly used routines, along with use cases. In this Appendix, routines follow the order in which they display in the **Repository**. They are grouped according to their types. Each type is detailed in a different section.

For more information on how to define routines, to access to system routines or to manage system or user routines, see [Chapter 10, *Managing routines*](#).

Before starting any data integration processes, you need to be familiar with *Talend Open Studio for ESB* Graphical User Interface (GUI). For more information, see [Appendix A, *GUI*](#).

D.1. Numeric Routines

Numeric routines allow you to return whole or decimal numbers in order to use them as settings in one or more Job components. To add numeric IDs, for instance.

To access these routines, double click on the **Numeric** category, in the **system** folder. The Numeric category contains several routines, notably sequence, random and decimal (convertImpliedDecimalFormat):

Routine	Description	Syntax
<i>sequence</i>	Returns an incremental numeric ID.	Numeric.sequence("Parameter name", start value, increment value)
<i>resetSequence</i>	Creates a sequence if it doesn't exist and attributes a new start value.	Numeric.resetSequence (Sequence Identifier, start value)
<i>removeSequence</i>	Removes a sequence.	Numeric.RemoveSequence (Sequence Identifier)
<i>random</i>	Returns a random whole number between the maximum and minimum values.	Numeric.random(minimum start value, maximum end value)
<i>convertImpliedDecimalFormat</i>	Returns a decimal with the help of an implicit decimal model.	Numeric. convertImpliedDecimalFormat ("Target Format", value to be converted)

D.1.1. How to create a Sequence

The **sequence** routine allows you to create automatically incremented IDs, using a **tJava** component:

```
System.out.println(Numeric.sequence("s1",1,1));
System.out.println(Numeric.sequence("s1",1,1));
```

The routine generates and increments the ID automatically:

```
[statistics] connecting to socket on port 3360
[statistics] connected
1
2
```

D.1.2. How to convert an Implied Decimal

It is easy to use the **convertImpliedDecimalFormat** routine, along with a **tJava** component, for example:

```
System.out.println(Numeric.convertImpliedDecimalFormat("9V99","123"));
```

The routine automatically converts the value entered as a parameter according to the format of the implied decimal provided:

```
1.23
[statistics] disconnected
Job test_routine ended at 14:12 04/02/2010. [exit code=0]
```

D.2. Relational Routines

Relational routines allow you to check affirmations based on booleans.

To access these routines, double click on the **Relational** class under the **system** folder. The Relational class contains several routines, notably:

Routine	Description	Syntax
<i>ISNULL</i>	Checks if the variable provided is a null value.	<code>Relational.ISNULL(variable to be checked)</code>

To check a Relational Routine, you can use the **ISNULL** routine, along with a **tJava** component, for example:

```
System.out.println(Relational.ISNULL(null));
```

In this example, the test result is displayed in the **Run** view:

```
Starting job test_routine at 14:14 04/02/2010.
[statistics] connecting to socket on port 3375
[statistics] connected
true
[statistics] disconnected
Job test_routine ended at 14:14 04/02/2010. [exit code=0]
```

D.3. StringHandling Routines

The **StringHandling** routines allow you to carry out various kinds of operations and tests on alphanumeric expressions, based on Java methods.

To access these routines, doubleclick on **StringHandling** under the system folder. The **StringHandling** class includes the following routines:

Routine	Description	Syntax
<i>ALPHA</i>	checks whether the expression is arranged in alphabetical order. Returns the true or false boolean accordingly.	<code>StringHandling.ALPHA("string to be checked")</code>
<i>IS_ALPHA</i>	checks whether the expression contains alphabetical characters only, or otherwise. Returns the true or false boolean accordingly.	<code>StringHandling.IS_ALPHA("string to be checked")</code>
<i>CHANGE</i>	replaces an element of a string with a defined replacement element and returns the new string.	<code>StringHandling.CHANGE("string to be checked", "string to be replaced", "replacement string")</code>
<i>COUNT</i>	Returns the number of times a substring occurs within a string.	<code>StringHandling.COUNT("string to be checked", "substring to be counted")</code>
<i>DOWNCASE</i>	converts all uppercase letters in an expression into lowercase and returns the new string.	<code>StringHandling.DOWNCASE("string to be converted")</code>

Routine	Description	Syntax
<i>UPCASE</i>	converts all lowercase letters in an expression into uppercase and returns the new string.	<code>StringHandling.UPCASE("string to be converted")</code>
<i>DQUOTE</i>	encloses an expression in double quotation marks.	<code>StringHandling.DQUOTE("string to be enclosed in double quotation marks")</code>
<i>INDEX</i>	returns the position of the first character in a specified substring, within a whole string. If the substring specified does not exist in the whole string, the value - 1 is returned.	<code>StringHandling.INDEX("string to be checked", "substring specified")</code>
<i>LEFT</i>	specifies a substring which corresponds to the first n characters in a string.	<code>StringHandling.LEFT("string to be checked", number of characters)</code>
<i>RIGHT</i>	specifies a substring which corresponds to the last n characters in a string.	<code>StringHandling.RIGHT("chaîne à vérifier", number of characters)</code>
<i>LEN</i>	calculates the length of a string.	<code>StringHandling.LEN("string to check")</code>
<i>SPACE</i>	generates a string consisting of a specified number of blank spaces.	<code>StringHandling.SPACE(number of blank spaces to be generated)</code>
<i>SQUOTE</i>	encloses an expression in single quotation marks.	<code>StringHandling.SQUOTE("string to be enclosed in single quotation marks")</code>
<i>STR</i>	generates a particular character a the number of times specified.	<code>StringHandling.STR('character to be generated', number of times)</code>
<i>TRIM</i>	deletes the spaces and tabs before the first non-blank character in a string and after the last non-blank character, then returns the new string.	<code>StringHandling.TRIM("string to be checked")</code>
<i>BTRIM</i>	deletes all the spaces and tabs after the last non-blank character in a string and returns the new string.	<code>StringHandling.BTRIM("string to be checked")</code>
<i>FTRIM</i>	deletes all the spaces and tabs preceding the first non-blank character in a string.	<code>StringHandling.FTRIM("string to be checked")</code>

D.3.1. How to store a string in alphabetical order

It is easy to use the ALPHA routine along with a **tJava** component, to check whether a string is in alphabetical order:

```
System.out.println(StringHandling.ALPHA("abcdefg"));
```

The check returns a boolean value.

```
Starting job test_routine at 14:29 04/02/2010.
[statistics] connecting to socket on port 3469
[statistics] connected
true
```

D.3.2. How to check whether a string is alphabetical

It is easy to use the **IS_ALPHA** routine along with a **tJava** component, to check whether the string is alphabetical:

```
System.out.println(StringHandling.IS_ALPHA("ab33cd"));
```

The check returns a boolean value.

```
Starting job routine1 at 11:45 23/02/2010.
[statistics] connecting to socket on port 3892
[statistics] connected
false
[statistics] disconnected
Job routine1 ended at 11:45 23/02/2010. [exit code=0]
```

D.3.3. How to replace an element in a string

It is easy use the **CHANGE** routine along with a **tJava** component, to replace one element in a string with another:

```
System.out.println(StringHandling.CHANGE("hello
world!", "world", "guy"));
```

The routine replaces the old element with the new element specified.

```
hello guy!
```

D.3.4. How to check the position of a specific character or substring, within a string

The **INDEX** routine is easy to use along with a **tJava** component, to check whether a string contains a specified character or substring:

```
System.out.println(StringHandling.INDEX("hello world!", "hello"));
System.out.println(StringHandling.INDEX("hello world!", "world"));
System.out.println(StringHandling.INDEX("hello world!", "!"));
System.out.println(StringHandling.INDEX("hello world!", "?"));
```

The routine returns a whole number which indicates the position of the first character specified, or indeed the first character of the substring specified. Otherwise, - 1 is returned if no occurrences are found.

```
Starting job routine1 at 15:47 24/02/2010.
[statistics] connecting to socket on port 4027
[statistics] connected
0
6
11
-1
[statistics] disconnected
Job routine1 ended at 15:47 24/02/2010. [exit code=0]
```

D.3.5. How to calculate the length of a string

The **LEN** routine is easy to use, along with a **tJava** component, to check the length of a string:

```
System.out.println(StringHandling.LEN("hello world!"));
```

The check returns a whole number which indicates the length of the chain, including spaces and blank characters.

```
12
```

D.3.6. How to delete blank characters

The **FTRIM** routine is easy to use, along with a **tJava** component, to delete blank characters from the start of a chain:

```
System.out.println(StringHandling.FTRIM(" Hello world !"));
```

The routine returns the string with the blank characters removed from the beginning.

```
Starting job routine1 at 16:14 24/02/2010.
[statistics] connecting to socket on port 3790
[statistics] connected
Hello world !
[statistics] disconnected
Job routine1 ended at 16:14 24/02/2010. [exit code=0]
```

D.4. TalendDataGenerator Routines

The **TalendDataGenerator** routines are functions which allow you to generate sets of test data. They are based on fictitious lists of first names, second names, addresses, towns and States provided by **Talend**. These routines are generally used when developing Jobs, using a **tRowGenerator**, for example, to avoid using production or company data.

To access the routines, double click on **TalendDataGenerator** under the **system** folder:

Routine	Description	Syntax
<i>getFirstName</i>	returns a first name taken randomly from a fictitious list.	<code>TalendDataGenerator.getFirstName()</code>
<i>getLastName</i>	returns a random surname from a fictitious list.	<code>TalendDataGenerator.getLastName()</code>
<i>getUsStreet</i>	returns an address taken randomly from a list of common American street names.	<code>TalendDataGenerator.getUsStreet()</code>
<i>getUsCity</i>	returns the name of a town taken randomly from a list of American towns.	<code>TalendDataGenerator.getUsCity()</code>
<i>getUsState</i>	returns the name of a State taken randomly from a list of American States.	<code>TalendDataGenerator.getUsState()</code>
<i>getUsStateId</i>	returns an ID randomly taken from a list of IDs attributed to American States.	<code>TalendDataGenerator.getUsStateId()</code>



No entry parameter is required as **Talend** provides the list of fictitious data.

You can customize the fictitious data by modifying the **TalendGeneratorRoutines**. For further information on how to customize routines, see [Section 10.3, “Customizing the system routines”](#).

D.4.1. How to generate fictitious data

It is easy to use the different functions to generate data randomly. Using a **tJava** component, you can, for example, create a list of fictitious client data using functions such as **getFirstName**, **getLastName**, **getUSCity**:

```
System.out.println(TalendDataGenerator.getFirstName());
System.out.println(TalendDataGenerator.getLastName());
System.out.println(TalendDataGenerator.getUSCity());
System.out.println(TalendDataGenerator.getUSState());
System.out.println(TalendDataGenerator.getUSStateId());
System.out.println(TalendDataGenerator.getUSStreet());
```

The set of data taken randomly from the list of fictitious data is displayed in the **Run** view:

```
Starting job test_routine at 14:44 04/02/2010.
[statistics] connecting to socket on port 3907
[statistics] connected
Jimmy
Arthur
Des Moines
Wyoming
UT
Milpas Street
[statistics] disconnected
Job test_routine ended at 14:44 04/02/2010. [exit code=0]
```

D.5. TalendDate Routines

The TalendDate routines allow you to carry out different kinds of operations and checks concerning the format of Date expressions.

To access these routines, double click on TalendDate under the **system** folder:

Routine	Description	Syntax
<i>addDate</i>	adds n days, n months, n hours, n minutes or n seconds to a Java date and returns the new date. The Date format is: "yyyy", "MM", "dd", "HH", "mm", "ss" or "SSS".	TalendDate.addDate("String date initiale", "format Date - eg.: yyyy/MM/dd", whole n, "format of the part of the date to which n is to be added - eg.: yyyy").
<i>compareDate</i>	compares all or part of two dates according to the format specified. Returns 0 if the dates are identical, 1 if the first date is older than the second and -1 if it is more recent than the second.	TalendDate.compareDate(Date date1, Date date2, "format to be compared - eg.: yyyy-MM-dd")
<i>diffDate</i>	returns the difference between two dates in terms of days, months or years according to the comparison parameter specified.	TalendDate.diffDate(Date1(), Date2(), "format of the part of the date to be compared - eg.: yyyy")
<i>diffDateFloor</i>	returns the difference between two dates by floor in terms of years, months, days, hours, minutes, seconds or milliseconds	TalendDate.diffDateFloor(Date1(), Date2(), "format of the part of the date to be compared - eg.: MM")

Routine	Description	Syntax
	according to the comparison parameter specified.	
<i>formatDate</i>	returns a date string which corresponds to the format specified.	<code>TalendDate.formatDate("date format - eg.: yyyy-MM-dd HH:mm:ss", Date())</code> to be formatted
<i>formatDateLocale</i>	changes a date into a date/hour string according to the format used in the target country.	<code>TalendDate.formatDateLocale("format target", java.util.Date date, "language or country code")</code>
<i>getCurrentDate</i>	returns the current date. No entry parameter is required.	<code>TalendDate.getCurrentDate()</code>
<i>getDate</i>	returns the current date and hour in the format specified (optional). This string can contain fixed character strings or variables linked to the date. By default, the string is returned in the format, DD/MM/CCYY.	<code>TalendDate.getDate("Format of the string - ex: CCYY-MM-DD")</code>
<i>getFirstDayOfMonth</i>	changes the date of an event to the first day of the current month and returns the new date.	<code>TalendDate.getFirstDayMonth(Date)</code>
<i>getLastDayOfMonth</i>	changes the date of an event to the last day of the current month and returns the new date.	<code>TalendDate.getLastDayMonth(Date)</code>
<i>getPartOfDate</i>	returns part of a date according to the format specified. This string can contain fixed character strings or variables linked to the date.	<code>TalendDate.getPartOfDate("String indicating the part of the date to be retrieved, "String in the format of the date to be parsed")</code>
<i>getRandomDate</i>	returns a random date, in the ISO format.	<code>TalendDate.getRandomDate("format date of the character string", String minDate, String maxDate)</code>
<i>isDate</i>	checks whether the date string corresponds to the format specified. Returns the boolean value true or false according to the outcome.	<code>TalendDate.isDate(Date() to be checked, "format of the date to be checked - eg.: yyyy-MM-dd HH:mm:ss")</code>
<i>parseDate</i>	changes a string into a Date. Returns a date in the standard format.	<code>TalendDate.parseDate("format date of the string to be parsed", "string in the format of the date to be parsed")</code>
<i>parseDateLocale</i>	parses a .string according to a specified format and extracts the date. Returns the date according to the local format specified.	<code>TalendDate.parseDateLocale("date format of the string to be parsed", "String in the format of the date to be parsed", "code corresponding to the country or language")</code>
<i>setDate</i>	modifies part of a date according to the part and value of the date specified and the format specified.	<code>TalendDate.setDate(Date, whole n, "format of the part of the date to be modified - eg.:yyyy")</code>

D.5.1. How to format a Date

The **formatDate** routine is easy to use, along with a **tJava** component:


```
System.out.println(TalendDate.formatDate("dd-MM-yyyy", new Date()));
```

The current date is initialized according to the pattern specified by the `new date()` Java function and is displayed in the **Run** view:

```
Starting job routine1 at 17:28 25/02/2010.
2010-02-25 17:28:07
Job routine1 ended at 17:28 25/02/2010. [exit code=0]
```

D.5.2. How to check a Date

It is easy to use the **isDate** routine, along with a **tJava** component to check if a date expression is in the format specified:

```
System.out.println(TalendDate.isDate("2010-02-09 00:00:00", "yyyy-MM-dd
HH:mm:ss"));
```

A boolean is returned in the **Run** view:

```
Starting job routine1 at 17:36 25/02/2010.
true
Job routine1 ended at 17:36 25/02/2010. [exit code=0]
```

D.5.3. How to compare Dates

It is easy to use the **formatDate** routine, along with a **tJava** component to check if the current date is more recent than a specific date, according to the format specified.

```
System.out.println(TalendDate.compareDate(new Date(),
TalendDate.parseDate("yyyy-MM-dd", "2010/11/24"), "yyyy-MM-dd"));
```

The current date is initialized by the Java function `new date()` and the value `-1` is displayed in the **Run** view to indicate that the current date precedes the reference date.

```
Starting job routine1 at 18:09 25/02/2010.
-1
Job routine1 ended at 18:09 25/02/2010. [exit code=0]
```

D.5.4. How to configure a Date

It is easy to use the **setDate** routine, along with a **tJava** component to change the year of the current date, for example:

```
System.out.println(TalendDate.formatDate("yyyy/MM/dd HH:mm:ss", new
Date()));
System.out.println(TalendDate.setDate(new Date(), 2011, "yyyy"));
```

The current date, followed by the new date are displayed in the **Run** view:

```
Starting job routine1 at 18:03 26/02/2010.
2010/02/26 18:03:14
Sat Feb 26 18:03:14 CET 2011
Job routine1 ended at 18:03 26/02/2010. [exit code=0]
```

D.5.5. How to parse a Date

It is easy to use the **parseDate** routine, along with a **tJava** component to change a date string from one format into another Date format, for example:

```
System.out.println(TalendDate.parseDate("yyyy-MM-dd HH:mm:ss",  
"1979-10-20 19:00:59"));
```

The string is changed and returned in the Date format:

```
Starting job routine1 at 11:58 01/03/2010.  
Sat Oct 20 19:00:59 CET 1979  
Job routine1 ended at 11:58 01/03/2010. [exit code=0]
```

D.5.6. How to retrieve part of a Date

It is easy to use the **getPartOfDate** routine, along with a **tJava** component to retrieve part of a date, for example:

```
Date D=TalendDate.parseDate("dd-MM-yyyy HH:mm:ss", "13-10-2010 12:23:45");  
  
System.out.println(D.toString());  
System.out.println(TalendDate.getPartOfDate("DAY_OF_MONTH", D));  
System.out.println(TalendDate.getPartOfDate("MONTH", D));  
System.out.println(TalendDate.getPartOfDate("YEAR", D));  
System.out.println(TalendDate.getPartOfDate("DAY_OF_YEAR", D));  
System.out.println(TalendDate.getPartOfDate("DAY_OF_WEEK", D));
```

In this example, the day of month (DAY_OF_MONTH), the month (MONTH), the year (YEAR), the day number of the year (DAY_OF_YEAR) and the day number of the week (DAY_OF_WEEK) are returned in the **Run** view. All the returned data are numeric data types.

```
Starting job routine at 10:52 17/12/2010.  
  
[statistics] connecting to socket on port 3565  
[statistics] connected  
Wed Oct 13 12:23:45 CEST 2010  
13  
9  
2010  
286  
4  
[statistics] disconnected  
Job routine ended at 10:52 17/12/2010. [exit code=0]
```



In the **Run** view, the date string referring to the months (MONTH) starts with 0 and ends with 11: 0 corresponds to January, 11 corresponds to December.

D.5.7. How to format the Current Date

It is easy to use the **getDate** routine, along with a **tJava** component, to retrieve and format the current date according to a specified format, for example:

```
System.out.println(TalendDate.getDate("CCYY-MM-DD"));
```

The current date is returned in the specified format (optional):

Starting job routine1 at 10:58 02/03/2010.

2010-03-02

Job routine1 ended at 10:58 02/03/2010. [exit code=0]

D.6. TalendString Routines

The **TalendString** routines allow you to carry out various operations on alphanumeric expressions.

To access these routines, double click on **TalendString** under the **system** folder. The **TalendString** class contains the following routines:

Routine	Description	Syntax
<i>replaceSpecialCharForXML</i>	Returns a string from which the special characters (eg.: <, >, &...) have been replaced by equivalent XML characters.	TalendString.replaceSpecialCharForXML("string containing the special characters - eg.: Thelma & Louise")
<i>checkCDATAForXML</i>	identifies characters starting with <![CDATA[and ending with]]> as pertaining to XML and returns them without modification. Transforms the strings not identified as XML in a form which is compatible with XML and returns them.	TalendString.checkCDATAForXML("string to be parsed")
<i>talendTrim</i>	parses the entry string and removes the filler characters from the start and end of the string according to the alignment value specified: -1 for the filler characters at the end of the string, 1 for those at the start of the string and 0 for both. Returns the trimmed string.	TalendString.talendTrim("string to be parsed", "filler character to be removed", character position)
<i>removeAccents</i>	removes accents from a string and returns the string without the accents.	TalendString.removeAccents("String")
<i>getAsciiRandomString</i>	generates a random string with a specific number of characters.	TalendString.getAsciiRandomString(whole number indicating the length of the string)

D.6.1. How to format an XML string

It is easy to run the **replaceSpecialCharForXML** routine along with a **tJava** component, to format a string for XML:

```
System.out.println(TalendString.replaceSpecialCharForXML("Thelma & Louise"));
```

In this example, the "&" character is replaced in order to make the string XML compatible:

```
Starting job routine1 at 15:48 02/03/2010.  
Thelma & Louise  
Job routine1 ended at 15:48 02/03/2010. [exit code=0]
```

D.6.2. How to trim a string

It is easy to use the **talendTrim** routine, along with a **tJava** component to remove the string padding characters from the start and end of the string:

```
System.out.println(TalendString.talendTrim("***talend open studio****",  
'*', -1));  
System.out.println(TalendString.talendTrim("***talend open studio****",  
'*', 1));  
System.out.println(TalendString.talendTrim("***talend open studio****",  
'*', 0));
```

The star characters are removed from the start, then the end of the string and then finally from both ends:

```
Starting job routine1 at 14:19 02/03/2010.  
  
**talend open studio  
talend open studio****  
talend open studio  
Job routine1 ended at 14:19 02/03/2010. [exit code=0]
```

D.6.3. How to remove accents from a string

It is easy to use the **removeAccents** routine, along with a **tJava** component, to replace the accented characters, for example:

```
System.out.println(TalendString.removeAccents("sâcrebleü!"));
```

The accented characters are replaced with non-accented characters:

```
Starting job routine1 at 16:02 02/03/2010.  
  
sacrebleu!  
Job routine1 ended at 16:02 02/03/2010. [exit code=0]
```