



Talend Software

Development Life Cycle

Best Practices Guide

6.4.1

Contents

Copyright.....	3
What is Software Development Life Cycle?.....	4
Software Development Life Cycle main phases.....	4
Talend Development life cycle.....	6
Specifications.....	9
Developing and Testing.....	10
Executing Tests.....	18
Continuous Integration: Deploying to QA and Production environments.....	33
Maintaining.....	37

Copyright

Adapted for 6.4.1. Supersedes previous releases.

Publication date: June 29th, 2017

Copyright © 2017 Talend. All rights reserved.

Notices

Talend is a trademark of Talend, Inc.

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

End User License Agreement

The software described in this documentation is provided under **Talend's** End User License Agreement (EULA) for commercial products. By using the software, you are considered to have fully understood and unconditionally accepted all the terms and conditions of the EULA.

To read the EULA now, visit <http://www.talend.com/legal-terms/us-eula>.

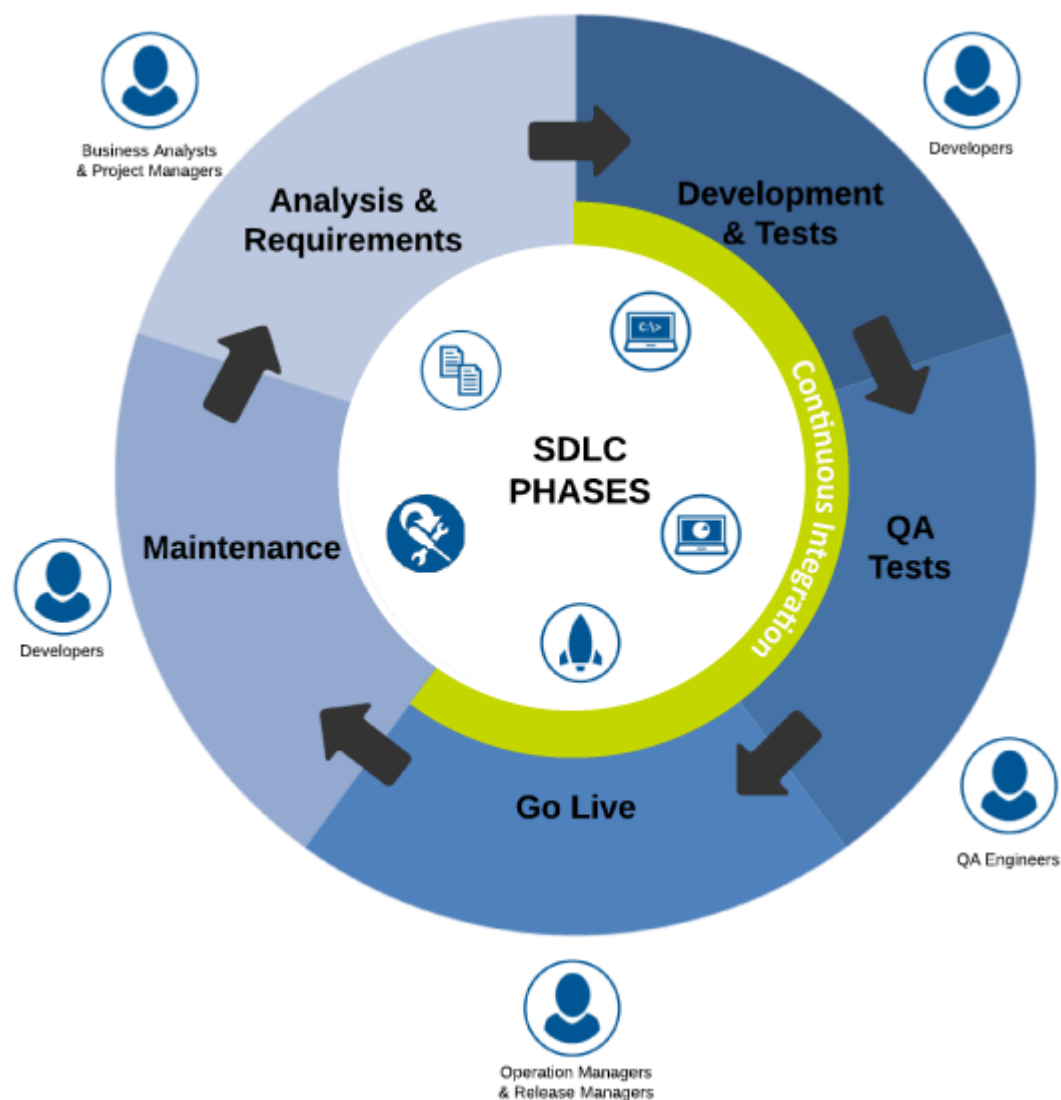
What is Software Development Life Cycle?

Software Development Life Cycle (SDLC) is the process of designing, developing and testing an application or piece of software in order to ensure its quality, efficiency and sustainability.

Software Development Life Cycle main phases

The following diagram shows the main phases of the Software Development Life Cycle : analysis and requirements definition, development of Jobs and tests, setting up of an automated build, Quality Assurance tests as well as product go-live and maintenance.

For more information on the people involved in these phases, see [Profiles](#) on page 5.



Development life cycle of a product

The development life cycle of a product or a feature is a wide process that includes specifications, development and testing, deployment into QA and Production environments, maintenance as well as migration.

- Specifications, which is the conceptual part of the process (technical requirements, task assignments and so on.). For more information, see [Specifications](#) on page 9.
- Development and testing, which is the designing part of the process. For more information, see [Developing and Testing](#) on page 10 and [Executing Tests](#) on page 18.

- Deployment into QA and Production environments. For more information, see [Continuous Integration: Deploying to QA and Production environments](#) on page 33.
- Maintenance which is mainly achieved through SVN or GIT branching and tags. For more information, see [Maintaining](#) on page 37.
- Migration.

SDLC related concepts

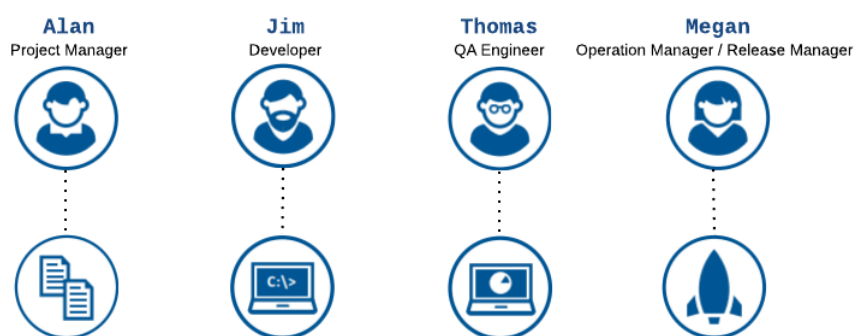
a list of the main concepts associated with Software Development Life Cycle (SDLC).

- Continuous Integration (CI) is a development practice where members of a team integrate their work frequently, each integration being verified by an automated build to detect integration errors as quickly as possible.
- Build automation is a best practice used during the software development life cycle to compile and package source code with an automation build script. For example, Talend users can export the sources of a Job they created in the Studio as a zip file using Maven and they can re-use this generated archive file in Talend Administration Center to schedule the next executions of this Job.
- Version control and Source Code Management (SCM) allow you to manage and track the changes made to the software by assigning revisions to these changes. Talend achieves this thanks to its shared Repository and branching system based on Git or Subversion. For example, Talend users can use the Repository that is shared between the Studio and other applications, as well as its branching system to version changes.

To summarize, SDLC aims at designing a regular and continuous build and deployment followed by automated end-to-end testing to verify the integrity of the current code base. These requirements also apply to the development of the software new features.

Profiles

Example set of the main persona that are involved in the phases of a product life cycle.



- Alan the Project Manager (Analysis and Requirements / Specifications Phase): Alan drives the project architecture and picks technologies to be used. He is in charge of designing the specifications along with business managers and developers;
- Jim the developer (Development and Tests + Maintenance Phases): Jim designs and implements Jobs and components in the Development environment. He also creates and run tests for the Jobs he designed, then passes these Jobs to Thomas. If needed, he can handle maintenance by fixing errors or improving the software to meet new technical requirements;
- Thomas the Quality Assurance engineer (QA Tests Phase): Thomas creates and executes test suites on features and products using manual (based on user scenarios) and automated build tests (using Test Cases) in the QA environment. He also sends regular result reports to Megan;
- Megan the Operation/Release Manager (Go Live Phase): Megan approves the release candidate build (RC) and deploys it to Production when she is sure all tests have passed.

Talend Development life cycle

What are the development environments and how the Development life cycle is implemented in these environments with the Talend products.

Development environments

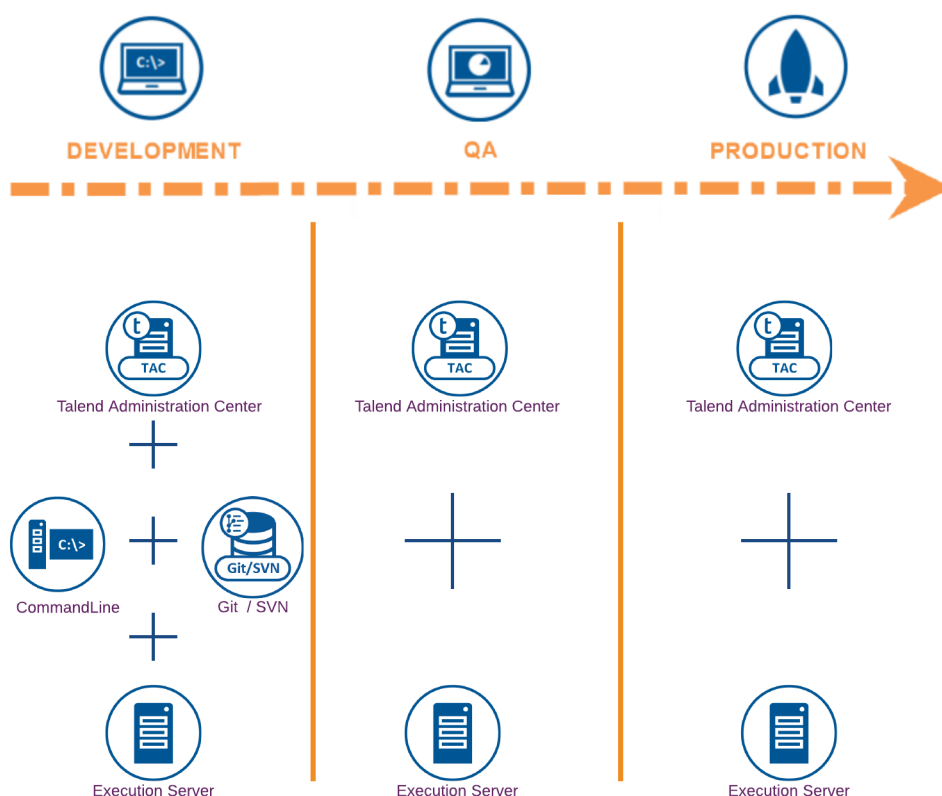
The following diagram shows the three environments in which the product or feature is created, integrated, tested and pushed to production.



The Development environment is a dedicated environment where the product or feature is designed and tested. For more information, see [Developing and Testing](#) on page 10.

The Quality Assurance (QA) environment allows you to validate the development processes created in the development environment without shutting down development work. Once the processes are validated, they are propagated to the Production environment. In this environment, the SVN and or Git Repository is no longer required and the Talend CommandLine application is optional as the Job sources can be imported from Job archives and builds.

The Production environment is a live environment where your product or feature is deployed into production. In this environment, the Talend CommandLine application as well as the Subversion/Git server are no longer required. For more information, see [Continuous Integration: Deploying to QA and Production environments](#) on page 33.

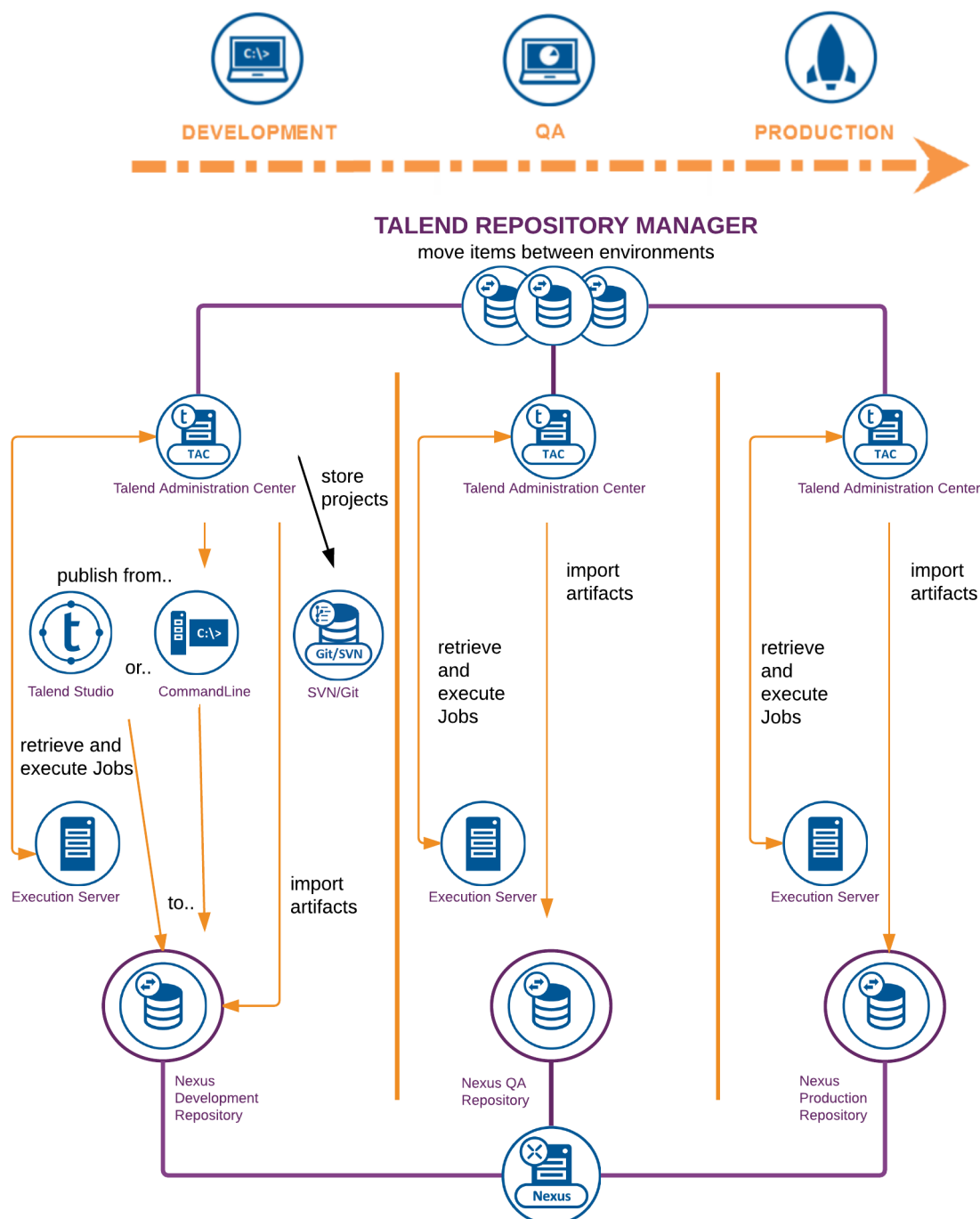


To understand how these three environments are linked together, see [Implementation with Talend](#) on page 7.

Implementation with Talend

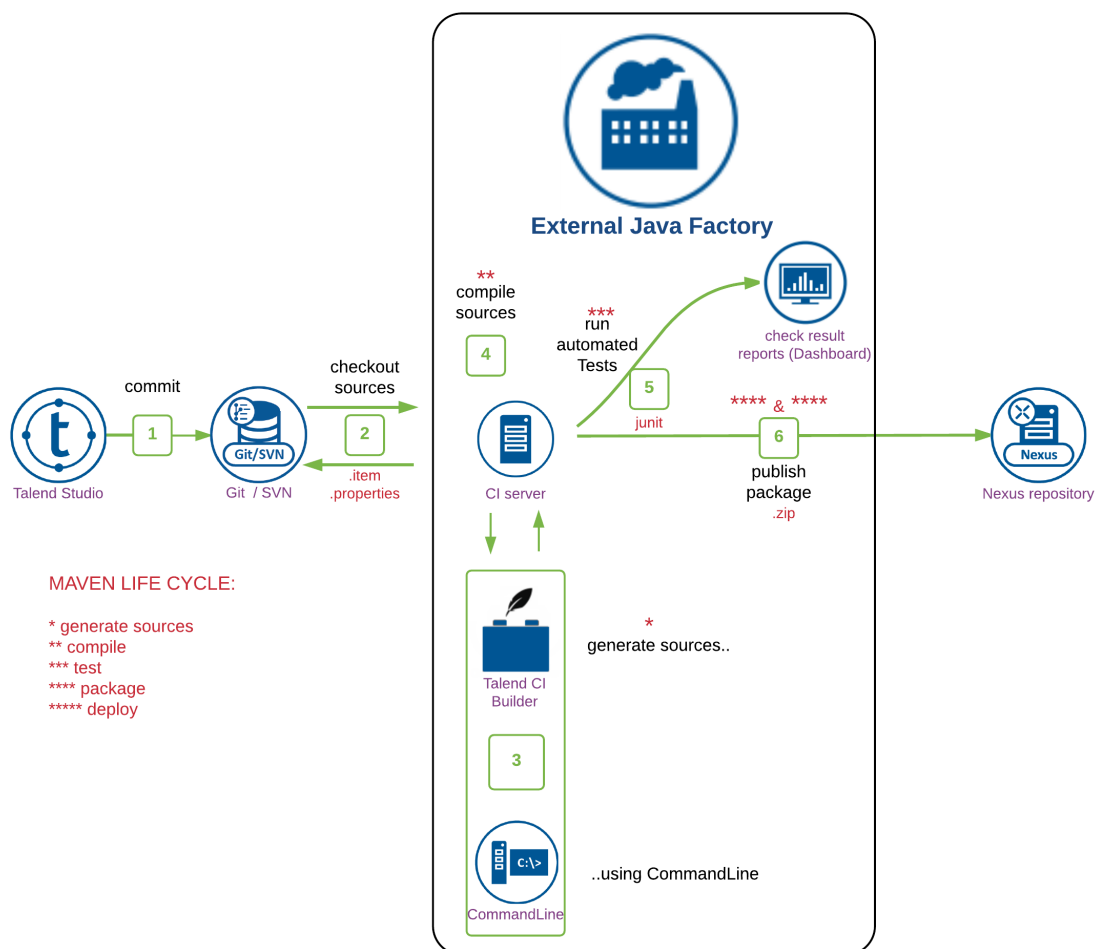
To pass items between environments, it is recommended to use both the Nexus Artifact Repository and Talend Repository Manager.

- Nexus stores and deploys the Job artifacts to the Nexus repositories of your choice.
- Talend Repository Manager moves project items once their quality is ensured. In order not to use SVN or Git in the development and production environment, "no-storage" projects holding the previously generated Job archives need to be created in these environments.



Implementation in your environment

The following diagram shows how Talend tools can be used and integrated in your own Java fabric, ensuring quick integration and quality of your projects from the beginning to the end of your software life cycle.



The main phases of the Continuous Integration process that are presented in this diagram are the following:

- **1 and 2 (Git or Subversion): Version and Revision Control**

Committing: Developers design Jobs and Tests in Talend Studio and commit them to Git or Subversion.

Checking out sources: SVN and or Git are linked to the Continuous Integration server that checks out the Jobs and Tests sources (in the form of .item and .properties files).

- **3 to 6 (in external Java factory): Maven Build, Continuous Integration and Deployment**

Generating sources: The Talend CI Builder and Talend CommandLine tools generate the SVN/Git sources and pass them to the Continuous Integration server that is used (Jenkins for example).

Compiling sources : An automated build is launched on the server to compile sources (transformed to Java classes).

Testing: Automated builds are launched on the server to execute Tests, and the server dashboard allows you to monitor and audit code quality before packaging.

Packaging and publishing: Once the Tests are executed and the bugs are fixed, items are packaged and published in the Nexus Artifact repository (in the form of .zip files). The versioned release candidate is then deployed to Production.

Continuous Integration ensures a quick, effective, automated and safe deployment to Production.

Specifications

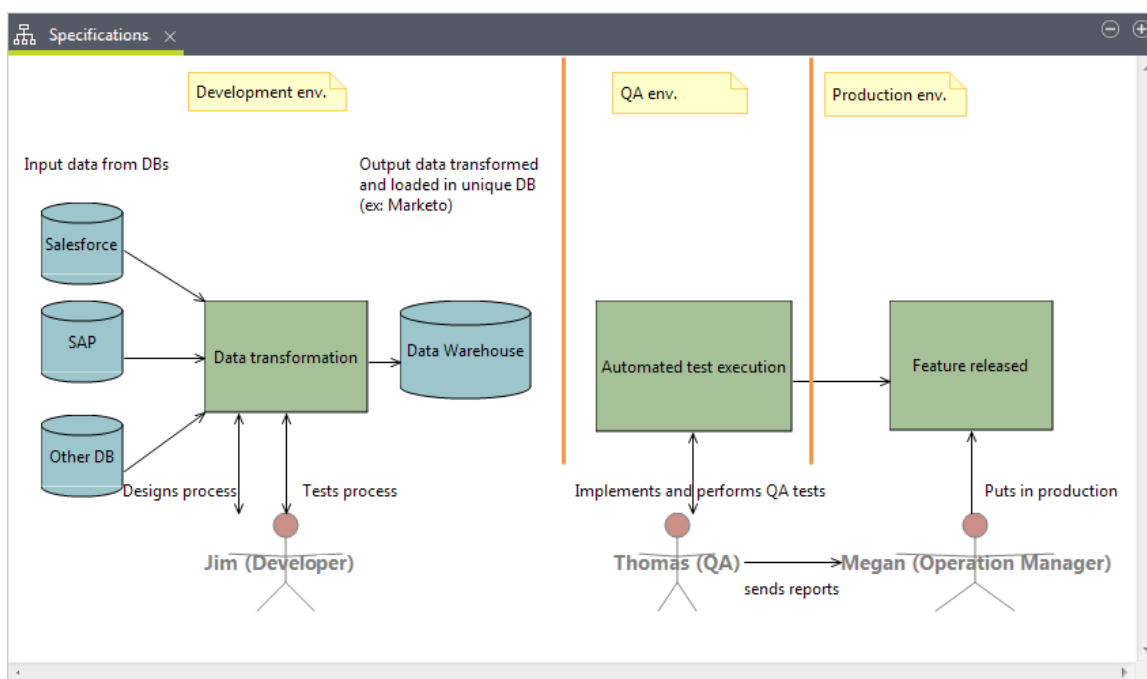
Before designing processes, business managers, decision makers or developers need to write down their specifications, this is one of the foundations of the test-driven development.

This step is essential to facilitate the designing and development of the product.

To address this issue, Talend offers a tool that helps you design Business Models, which are non technical views of a business workflow need. This tool helps you to graphically represent your needs regardless of the technical implementation requirements.

Business Models are both available in the Integration perspective of the Talend Studio and from the **Business Model Designer** page in Talend Administration Center.

With this tool, you can model your flow management needs at a macro level and assign tasks to specific actors.



You can then open, edit or update your business model on the **Repository Browser** page of Talend Administration Center by selecting the corresponding item in the Repository.



Once you have conceptualized your model, you need to put in place the management processes you have defined.

Developing and Testing

In a test-driven approach, process development and tests are closely related to each other and thus the task of the developer is only complete once the process has been both designed and tested. Once the Business Model is defined, the development team can start designing the processes which are called Jobs.

In a Development environment, developers need:

- the Talend Studio and/or Talend CommandLine (same as the Studio without GUI), where they design and generate the Jobs,
- a Git or Subversion server, in order to store their items (Jobs, metadata, etc.) into a shared repository,
- the Talend Administration Center application to schedule the execution of the Jobs,
- an execution server, to deploy and execute their Jobs.

Designing Jobs

At this stage, the conceptualization part is done and each team has been assigned some tasks. The development team designs Jobs in the Talend Studio, which are the development unit in Talend. Jobs allow you to put in place up and running dataflow management processes.

Best practices: To ensure continuous integration during development and to help developers design and build consistent, efficient and optimised Jobs, here are some best practices we recommend you to follow:

Concept	Best practice example
Naming standards	<p>In the Studio, define a naming convention for Jobs and folders and follow it.</p> <p>In this document, the naming convention is the following, but feel free to adapt it to your requirements: <code>job</code> prefix for Job names, <code>test</code> prefix for Test Case names, <code>pub</code> prefix for publishing task names and <code>task</code> prefix for execution task names.</p> <p>For example, name your folder <code>xxx</code>. Folders should be used to group Jobs of a similar type. Then create a Job named <code>job_xxx_description</code> and its Test case named <code>test_xxx_description</code>.</p> <p>At a more granular level, components should also have a meaningful name.</p>
Version control	<p>Use SVN/GIT branches and tags as well as the Studio to handle Job versions.</p> <p>For more information on how to change the version of your Jobs centrally at once to publish them with the version of your choice, see How to change the deployment version of each Job or Route at once on page 11.</p>
Metadata	Use schema metadata in your Jobs to share database connections between several Jobs and help designing source/target components.

Concept	Best practice example
Contexts	<p>Use contexts in order to reuse variables (context parameters locally for Jobs, group contexts globally for projects) such as database connectivity, host names, ports, etc. If values need to be changed or are used in multiple places, then they should not be hard coded and it is recommended to use contexts.</p> <p>These contexts are also useful to switch between environments (Development context then QA context then Production context).</p>
Standard Job layout	<p>Use a standard Job layout to ensure its readability, it is particularly useful for collaborative work.</p> <p>Some examples include: putting data flows from left to right, top-to-bottom layout to show the process flow between subJobs, target components on the right, etc.</p>
Complexity	<p>Jobs should follow a logic and be split in steps, called subJobs, when necessary. It is also recommended to use parent Jobs to run one or several child Jobs in order to create a process flow and even though there is no limit, you should avoid using more than 20 components in a Job.</p>

Once the Job is designed in a remote project from the Studio or the CommandLine (via the exportJob command), it can be published, deployed and executed in Talend Administration Center. Exporting the Job as an artifact will also help to perform Quality assurance tests on the same exact Jobs than those created in the Development environment. For more information, see [Continuous Integration: Deploying to QA and Production environments](#) on page 33.

How to change the deployment version of each Job or Route at once

How to edit the version of your Jobs or Routes centrally at once to deploy and publish them with the version of your choice. This feature allows you to release your whole project with the same fixed version. To do that, you need to edit some Maven parameters from the Talend Studio.

You have created several Jobs and or Routes in your project.

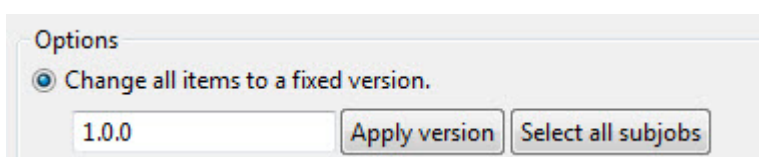
1. From the Talend Studio toolbar, click the **Project Settings** icon to open the corresponding window.
2. Open the menu **Build > Maven > Deployment Versioning**.
3. Change the version of your items according to your needs:

- to apply the same version to all your Jobs and Routes at once:

Select the items you want to edit in the repository view then click **Change all items to a fixed version**.

In the text field, edit the version value then click **Apply version**.








Ex : The version 1.0.0-SNAPSHOT is changed to 1.0.0 at release time.



- to apply a different version to each Job and Route at once:

Select the Jobs and Routes you want to edit in the repository view then click **Update the version of each item**.

Edit the versions in the **New version** column of the table.

Items	Version	New version
 Continuuou...	0.1.0	1.0.0
 Copy_Cont...	0.1.0	0.1.1
 MemoryRun	0.1.0	1.0.0
 Standard_L...	0.1.0	1.0.0
 job_Get_De...	0.1.0	1.0.0
 memory_run	0.1.0	1.0.0
 Route_test	0.1.0	1.0.0-SNAPSHOT

Tip:

You can also edit the versions of your items individually, as well as the GroupID used to deploy them, from the **Deployment** tab in the Job or Route settings.

- to apply the version that is used in the Job or Route:

Select the Jobs and Routes you want to edit in the repository view then click **Use Job versions** to use the version of the item as the artifact version that will be deployed.

4. Click **OK** to save your changes and close the window.

When you will republish your project items on your Continuous Integration server, at release time for example, these items will be published with the version you have defined centrally from the Studio project settings.

For more information on how to build Jobs or Routes from the Studio, see sections [How to build Jobs](#) and [How to build Routes](#) in the Talend Studio User Guide.

Fore more information on how to publish and execute Jobs via Talend Administration Center or an external Continuous Integration server, see [Executing Tests](#) on page 18.

Designing Tests

While designing the processes (Jobs and Routes), developers need to think of how to test them: Talend recommends you to use the Test Case feature: it automatically creates a Test Case with a skeleton in a Test Instance.

A Test Case is an executable test that consists of an immutable part extracted from the initial Job or Route, along with other editable components that form the skeleton of the Test Case.

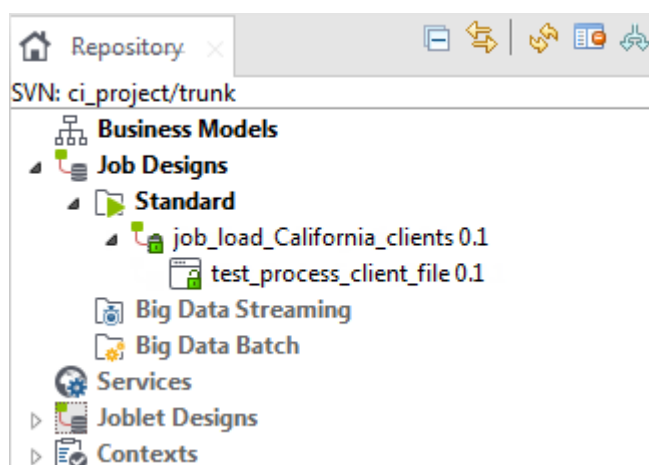
A Test Instance is a set of data that allows you to run the Test Case with different parameters that you define (input, reference files, etc.).

Best practices:

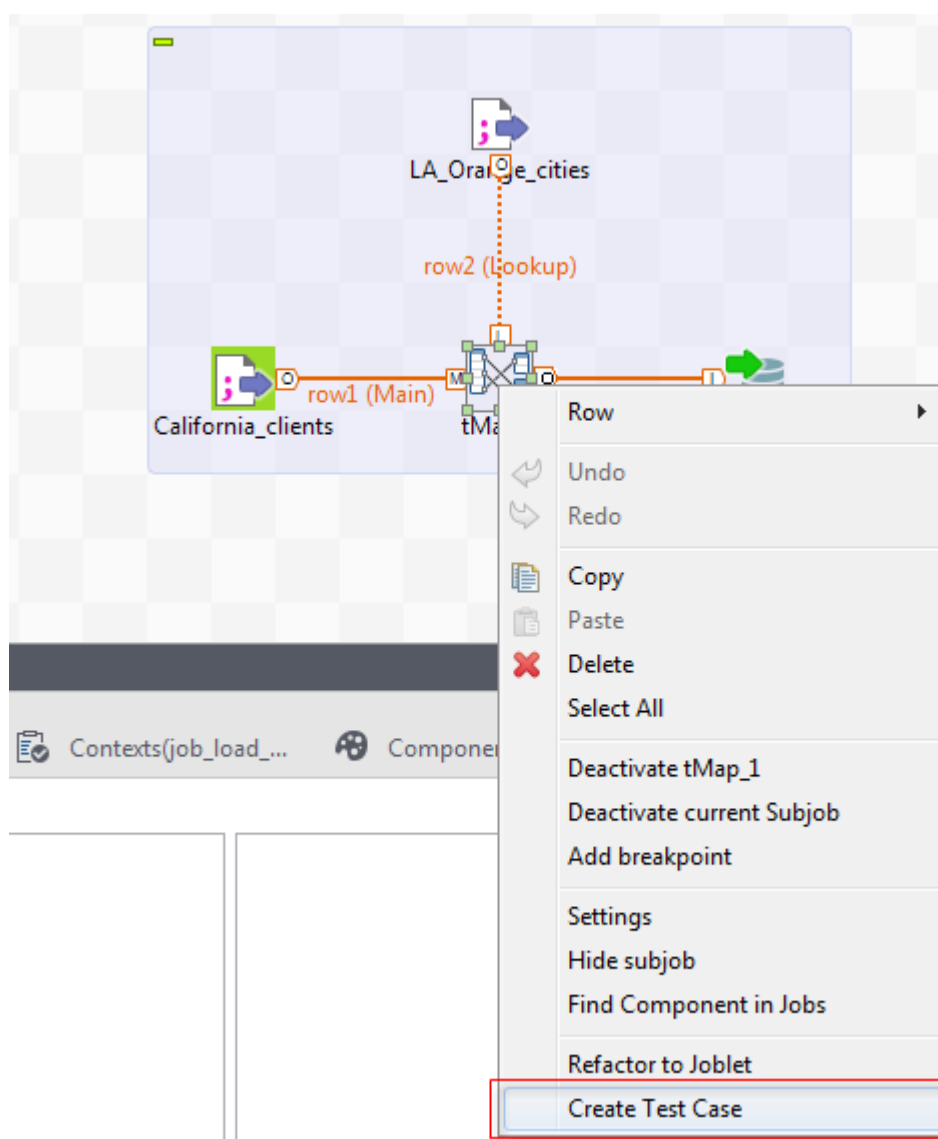
- It is recommended to create and use a context adapted to your environment (a Test context to execute test Jobs with the metadata of this environment, and a Production context to execute Jobs in the Production environment).
- When the feature is designed and tested, it is recommended to use Talend Repository Manager to pass items to the QA environment. See [Continuous Integration: Deploying to QA and Production environments](#) on page 33 for more information.

Example of Test case based on a Job

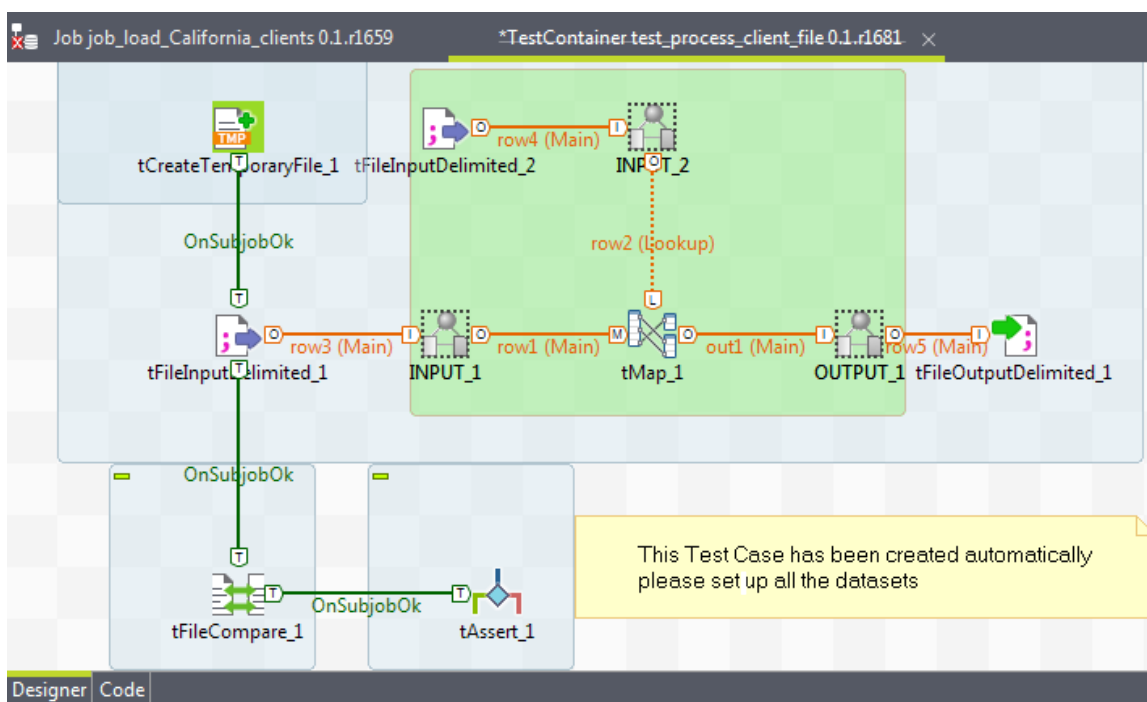
A Job called `job_load_California_clients` is created in a project called `ci_project`. The Job aims at reading a `.csv` file containing a list of clients living in California, matching these clients with those coming from the Orange county of Los Angeles using a **tMap** component, before uploading the result into a MySQL database.



The processing part (**tMap**) is used to create a Test case called `test_process_client_file` and will allow developers to test, filter and map any types of input and output files.

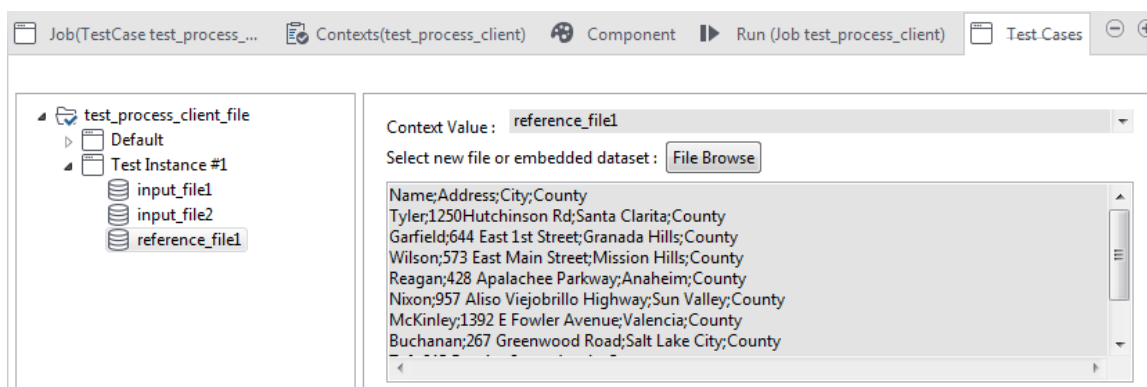


Note that the generated skeleton depends on the component(s) selected in the Job to create the Test.



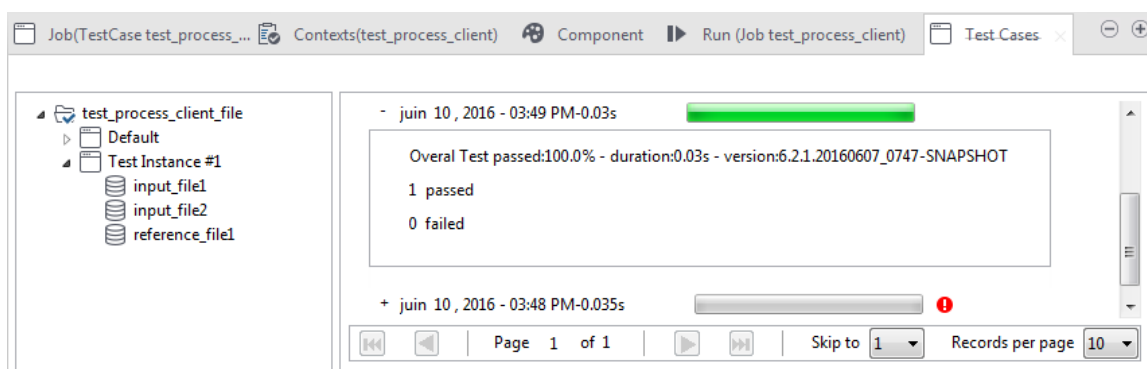
Here, the Test case aims at:

- reading input data files (**tFileInputDelimited** components),
- transforming data with an immutable set of components (**INPUT** and **OUTPUT** items) based on the initial Job,
- writing the output data (to a **tFileOutputDelimited** component),
- comparing the temporary output file (**tCreateTemporaryFile** component) to a reference file you need to define, using a **tFileCompare** component,
- generating the Test execution status (**OK** if it succeeds, **Fail** if it fails) using a **tAssert** component.



Note that you can add as many instances of tests as you need, which means you can run the same test with different input and reference files.

The Test Case is ready to be executed once the data set has been defined in the **Test Case** view, and a specific context group (called **Test**) has been defined in the **Context** view. The data set consists of data files that you define as input and reference files to test your data.

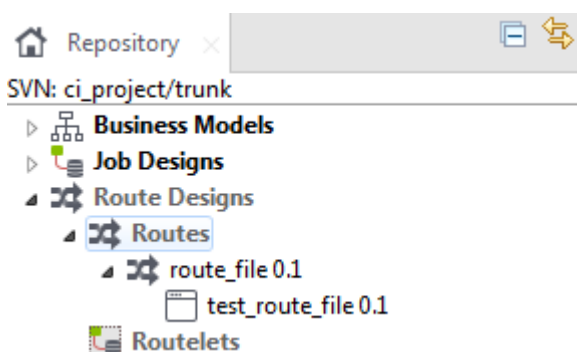


The Test Case was successfully executed on the Test Instance and the input and reference files are identical.

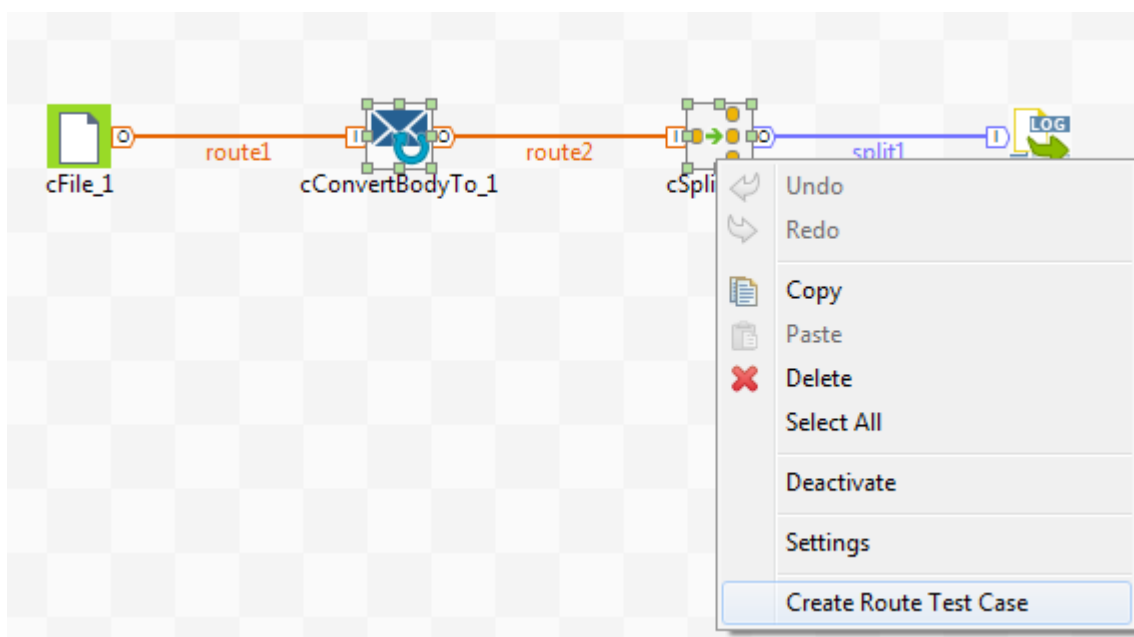
Once developers have designed the integration tests locally in the Studio, these tests need to be automated with continuous integration tools such as build systems. For more information, see [Executing Tests](#) on page 18.

Example of Test case based on a Route

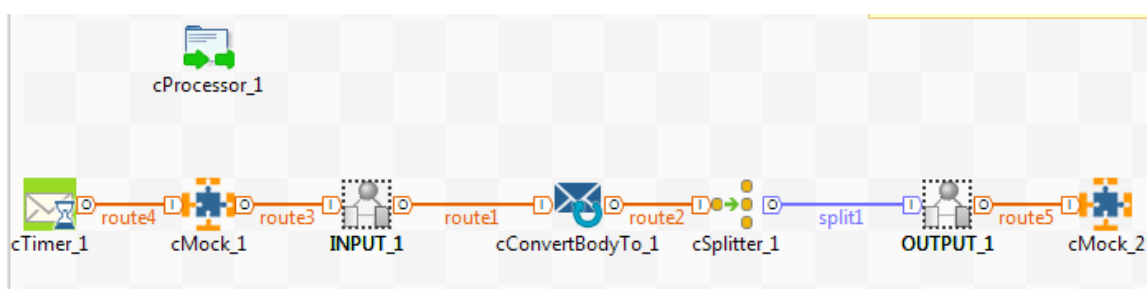
A Route called `route_file` is created in a project called `ci_project`. The Route aims at reading a file before converting its content to String rows, splitting it line by line and outputting the result in a log file.



The processing part (**cConvertBodyTo** and **cSplit**) is used to create a Route Test case called `test_route_file` and **cMock** components are used to simulate message generation and message endpoints, allowing developers to test and map any types of input and output messages.



Note that the generated skeleton depends on the component(s) selected in the Route to create the Route Test.

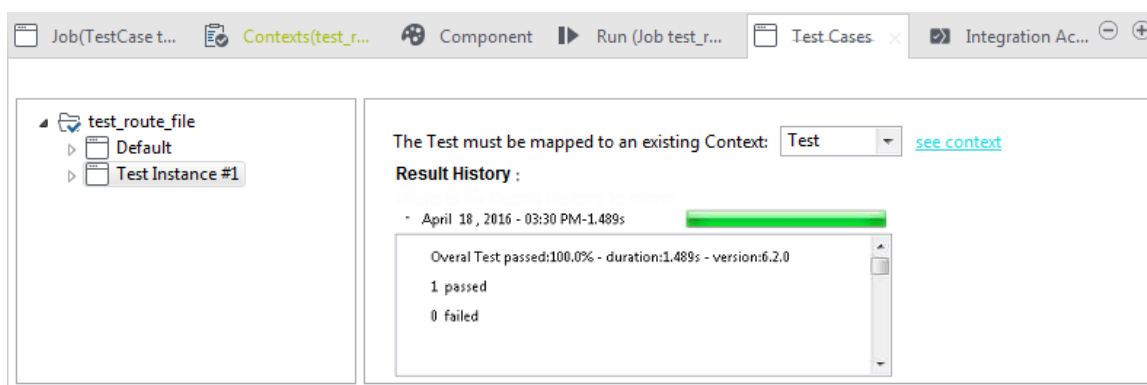


Here, the Test case aims at:

- generating test message exchanges and reading input test data (**cTimer** and **cMock_1** components),
- transforming data with an immutable set of components (**INPUT** and **OUTPUT** items) based on the initial Route,
- checking that messages have been routed as expected and validating the Test output result (message content, count, header, etc.) using the **cMock_2** component.

Note that you can add as many instances of tests as you need, which means you can run the same test with different input and reference files.

The Test Case is ready to be executed once the data set has been defined in the **Test Cases** view. The data set consists of data files that you define as input and reference files to test your data.



The Route Test Case was successfully executed on the Test Instance and the input and reference files are identical.

Once developers have designed the integration tests locally in the Studio, these tests need to be automated with continuous integration tools such as build systems. For more information, see [Executing Tests](#) on page 18.

Executing Tests

In a continuous integration environment, it is common practice to launch tests at every commit. By default, a new commit is made every time you save your Job (Automatic SVN or GIT commit mode). The **Unlocked items** commit mode allows you to commit the changes made on items only when those items are unlocked.

Talend offers you several ways to publish and schedule the test executions, and allows you to choose the one that suits best your needs:

- Option 1: use the **Publisher** and **Conductor** pages of Talend Administration Center. For more information, see [Option 1: Publishing and scheduling test execution in Talend Administration Center](#) on page 18.
- Option 2: use an external Continuous Integration server. For more information, see [Option 2: Scheduling automatic test executions in your build environment using Talend CI Builder](#) on page 21.
- Alternative: use the Talend Studio **Publish** option or Talend CommandLine.

Option 1: Publishing and scheduling test execution in Talend Administration Center

You can compile your Tests, execute them or schedule their executions as well as publishing them from the Talend Administration Center web application with no dependencies on third-party software.

- [Publishing Jobs and executing Tests at the same time](#) on page 18
- [Scheduling test executions](#) on page 20

Publishing Jobs and executing Tests at the same time

Talend Administration Center allows you to Publish your Jobs to a Nexus repository via the **Publisher** page.

From this page, you can:

- create a publishing task on a Job or on all the Jobs designed in the Studio and set a threshold (percentage) of maximum failed tests allowed for these Jobs:

Publish Task



Publish Task

Label:

Description:

Active: ☒

Project:

Branch:

All Services: ☐

All Routes: ☐

All Batch Jobs: ☒

All Runtime Jobs: ☐

Individual:

Name:

Version:

Snapshot: ☒

Repository:

Group ID:

Artifact:

Publish Version:

☒ Max failed test %



- add a time-based trigger to publish the task to the selected Nexus repository, or publish it manually from the top toolbar:

PUBLISHER												
Refresh Add Duplicate Delete Publish Open Artifact repository												
Status	Error status	Label	Active	Trigger s...	Time left ...	Project	Branch	Publish scope	Name	Version	Sn...	Repository
Ready to publish	No Error	pub_load_California_clients	<input checked="" type="checkbox"/>		4h 55min ~	ci_project	trunk	Job - Batch	job_load_California...	Latest	<input checked="" type="checkbox"/>	snapshots

- check the publication status (here, the publication went well which means no tests

PUBLISHER							
Refresh Add Duplicate Delete Publish Open Ar							
Status	Error status	Label	...	Trigg...	Time left b...	Last run	Project
Published	No Error	pub_all_jobs	<input checked="" type="checkbox"/>			2016-06-...	ci_project

failed):

Scheduling test executions

From Talend Administration Center, you can also create execution tasks on the **Job Conductor** page that will allow you to schedule the execution of your Jobs and Tests.

These tasks can be either:

- selected from the **Publisher**, provided that you have created the corresponding publishing task on the **Publisher** page.

Item from Publisher ✕

Label	Repository	Group ID	Artifact	Publish Version	Project	Branch	Name	Ver
pub_load_California_clients	snapshots	org.example	job_load_California...	Latest	ci_project	trunk	job_loa...	Lat

- or retrieved directly from Nexus at execution time if they have been published previously via the **Publisher** page.

Select artifact from Nexus repository ✕

Repository: ▼ Filter external repositories ☒

Browse

[-]

[+]

org

example

JobForLogs

0.1.0-SNAPSHOT

job_load_California_clients

0.1.0-SNAPSHOT

job_load_California_clients-0.1.0-SNAPSHOT.zip

Once the execution tasks have been created, you can add a time-based trigger on them to schedule their execution according to your needs.

JOB CONDUCTOR

Status	Error status	Label	Trigger s...	Actions	Time left ...	Project	Branch	Vers...	Context	Server
Project: ci_project (1 item)										
Ready to dep...		task_load_california_...			1h 54min ~	ci_project	trunk	(0.1)	(Default)	ci_server

Option 2: Scheduling automatic test executions in your build environment using Talend CI Builder

In case you want to use your own Continuous Integration server to schedule test executions, Talend allows you to do so by providing you the Talend CI Builder.

Talend CI Builder is a Maven Plugin delivered by Talend that transforms the Talend Job sources to Java classes using the Talend CommandLine application, allowing you to execute your tests in your own company Java factory.

The following sections introduce one way to automate the test executions using the Talend CommandLine and Talend CI Builder that generate the sources, as well as a Jenkins server that creates builds automating the execution of all Test cases and deploys them on the Nexus artifact repository. The Jenkins server is also used to monitor these executions.

- [Before configuring the execution of your Jobs](#) on page 21
- [Installing Talend CI Builder](#) on page 22
- [Configuring Jenkins](#) on page 22
- [Executing the Tests](#) on page 32

Before configuring the execution of your Jobs

There are several prerequisites you should ensure before you start scheduling automatic test executions with the Talend CI Builder.

- A version superior or equal to Apache Maven 3 is installed in your server. For more information, see [the Apache Maven website](#).
- The Jenkins Continuous Integration server is properly installed and set up. For more information, see [the Jenkins website](#).
- You have a Talend CommandLine application that will only be used for Continuous Integration purposes (cannot be the same as the Talend CommandLine used for generic actions such as publishing items from Talend Studio to Talend Administration Center, generating execution tasks in Talend Administration Center, etc.).
- You have previously created Test Cases in your SVN or Git project that you want to execute automatically. For more information, see [Designing Tests](#) on page 12.
- The external libraries stored in Nexus that are needed to execute your Jobs have been properly installed in this project and your Nexus instance is started. .
- If you are using Talend Administration Center, you have configured it to retrieve the external modules from Nexus.
- Otherwise, you need to configure the CommandLine to connect to the Nexus repository where these external modules are stored by editing the <StudioPath>/configuration/config.ini file and adding the following at the end of the file:

```
nexus.url=<http://localhost:8081/nexus>
nexus.user=<Nexus_username>
nexus.password=<Nexus_password>
nexus.lib.repo=talend-custom-libs-release
nexus.lib.repo.snapshot=talend-custom-libs-snapshot
```

where <http://localhost:8081/nexus> corresponds to your Nexus location URL, and <Nexus_username>/<Nexus_password> corresponds to your Nexus credentials.

By default, only the nexus.url value is mandatory, as nexus.user and nexus.password values are not needed to download the modules.

- In order for the deployment to Nexus to work, you need to add the following in the `/ .m2 / settings.xml` Maven configuration file:

```
<servers>
    <server>
        <id>tac</id>
        <username><Nexus_username></username>
        <password><Nexus_password></password>
    </server>
</servers>
```

For more information on this file, see [the Apache Maven documentation](#).

To summarize, you need the following applications: Talend CommandLine, a Continuous Integration server (here, Jenkins), Maven, Nexus and Talend CI Builder.

Installing Talend CI Builder

1. Unzip the `Talend-CI-Builder-V6.4.1.zip` archive file in the directory of your choice.
2. Browse to this installation directory and execute the following command: `mvn install:install-file -Dfile=ci.builder-6.4.1.jar -DpomFile=ci.builder-6.4.1.pom`
3. In your Nexus web application, click **Repositories** then select the **thirdparty** repository.
4. Click the **Artifact Upload** tab and upload the CI Builder POM and JAR files you have just installed on your machine.

This Maven Plugin is now available for anyone and can be incorporated in your builds. For more information on the Nexus **thirdparty** repository, see [the Nexus documentation](#).

Configuring Jenkins

How to configure Jenkins to generate the sources of your Jobs and Tests, to run all these Tests as well as to deploy Jobs on the Nexus artifact repository.

Before creating your build projects, check in the Jenkins configuration that:

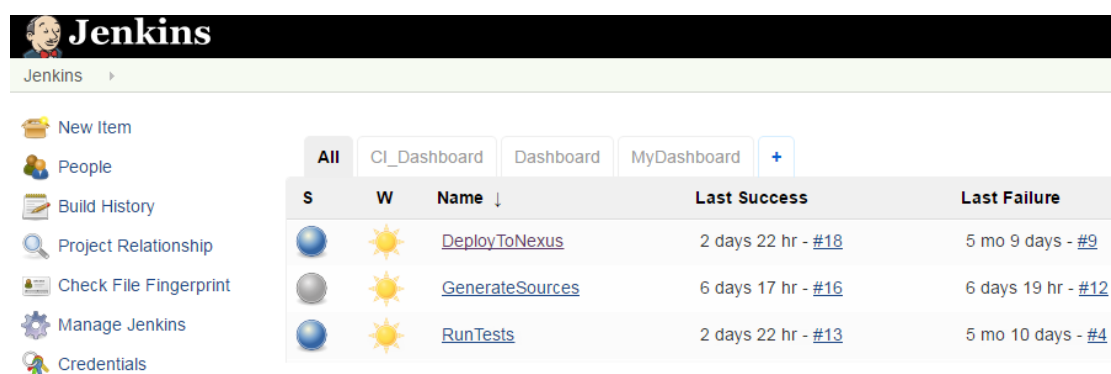
- the paths to your Jenkins, JDK (java 8 is needed), Git and Maven installation directories are properly set.
- the credentials of your Subversion and or Git accounts are set and valid.

Creating the Jenkins projects

How to create the build projects on the Jenkins server.

1. Launch the Jenkins server.
2. Create three Jenkins build projects (Maven projects):

Project name (example)	Goal
GenerateSources	Gets the sources (potentially at each commit) and converts them to Java classes with the Talend CI Builder. This project calls the Talend CommandLine application that generates all the sources of the Jobs and Tests you have created in your project. See Configuring Jenkins to generate the sources on page 23.
RunTests	Project triggered at the end of the build project GenerateSources that runs all Test Cases at once. See Configuring Jenkins to run all tests on page 31.
DeployToNexus	Project triggered at the end of the build project RunTests that deploys the Jobs on the Nexus repository of your choice. See Configuring Jenkins to deploy Jobs on Nexus on page 31.



Configuring Jenkins to generate the sources

To generate the sources of your Jobs and Tests on the Jenkins server, you need to:

- link the GenerateSources Jenkins project to the SVN/Git project where your sources are stored. See [Link the Jenkins project to the SVN/Git project](#) on page 23.
- decide how to generate your sources:
 - Recommended: locally (script mode): generate the sources of your Jobs and Tests locally without having to keep the Talend CommandLine up and running during the build execution, as the Talend CommandLine is called, used and shut down. It allows you to run several builds at the same time based on the same Talend CommandLine installation, however you need to use a different workspace for each build.
 - Alternative: remotely (server mode): generate the sources of your Jobs and Tests remotely using the Talend CommandLine which needs to be launched (often installed as a service).

See [Create the POM file to generate sources](#) on page 24.

- specify the parameters to generate sources in the GenerateSources Jenkins project. See [Specify the parameters to generate the sources](#) on page 28.

Link the Jenkins project to the SVN/Git project

1. Link the Jenkins GenerateSources project to the SVN or Git project in which you created your Test Cases. Note that several projects can be added.
2. SVN only: Fill in the branch you want to retrieve (in this example, `CI_PROJECT/trunk`) and the base folder where you saved all your projects (here, `projects`) followed by a folder with the same name as the original technical name of the project (here, `CI_PROJECT`). In this way, the existing workspace folder will be completely replicated and the Talend CommandLine application will be able to connect directly to the defined folder (`projects`) to find the projects.

For an SVN project

Source Code Management

☐ None
☐ CVS
☐ CVS Projectset
☐ Git
☒ Subversion

Modules Repository URL ?
 Local module directory (optional) ?
 Repository depth ?
 Ignore externals ☐ ?

[Add more locations...](#)

For a Git project

Source Code Management

☐ None
☐ CVS
☐ CVS Projectset
☒ Git

Repositories Repository URL ?
 Credentials ▼

Create the POM file to generate sources

1. Create a POM file (in XML format), called `BuildJob_pom.xml` for example, holding information on Maven project and its configuration and the instructions to generate your sources. It could look as follows (note that the parts in *italics* need to be modified):

- Recommended: **Local generation** (script mode)

When you choose to generate sources in script mode (`local-generate`), you do not need to launch the Talend CommandLine as it will be launched and shut down automatically, but you do need to specify in this POM file:

- properties linked to the Talend CommandLine used to generate the sources:
 - `<product.path>`: path to Talend CommandLine
 - `<commandline.workspace>`: path to the workspace which is the base folder where all the projects were checked out before
 - `<projectsTargetDirectory>`: folder where all the sources of your project will be generated

- `<deploy.version>`: defines the version of the items to be published on Nexus and applies `-SNAPSHOT` tags to these items (can be Jobs, Routines, Test cases, etc.). This parameter affects the output result of the generation.

Example of snapshot version: `<deploy.version>1.0.0-SNAPSHOT</deploy.version>`

Example of release version: `<deploy.version>1.0.0</deploy.version>`

Tip: The only requirement is to store all projects in the workspace folder, this means you have the possibility create a Jenkins task that clones the projects and copy them (folders only) to the workspace.

- the configuration parameters that allow you to customize the execution:
 - `<itemFilter>`: to filter generation on project items. This parameter allows you to build only specific Jobs. For examples of filtering, see [How to filter the execution of your project on a single Job or specific Jobs](#) on page 30.
 - `<commandlineUser>`: to define who is the Talend CommandLine user
 - `<jvmArguments>`: to define the memory that will be allocated to the generation
- the Nexus repository where the CI Builder plugin is stored: the parameter `<pluginRepository>` allows you to specify the URL of the Nexus **thirdparty** repository in which you have uploaded the CI Builder files.

The Maven documentation of the CI Builder plugin is embedded in the CI Builder archive file, see the files in the `<CIBuilder_home>/help` folder for more information.

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://
www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/
xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>org.talend</groupId>
    <artifactId>buildsources</artifactId>
    <version>A.B.C</version>

    <properties>
        <!-- Required. Commandline application workspace and Studio path, only for
local(script) mode -->
        <commandline.workspace>commandline_workspace_path</commandline.workspace>

        <product.path>studio_path</product.path>
        <!-- Optional. Specify target directory where generated sources will be stored -->
        <projectsTargetDirectory>${basedir}/projectSources</projectsTargetDirectory>
        <!-- Optional. Specify version for the artifact to be built. Can be set for each Job
independently -->
        <deploy.version>A.B.C-SNAPSHOT</deploy.version>
```

```
</properties>
```

```
<build>
```

```
  <plugins>
```

```
    <plugin>
```

```
      <groupId>org.talend</groupId>
```

```
      <artifactId>ci.builder</artifactId>
```

```
      <version>A.B.C</version>
```

```
      <executions>
```

```
        <execution>
```

```
          <phase>generate-sources</phase>
```

```
          <goals>
```

```
            <!-- local(script) mode -->
```

```
            <goal>local-generate</goal>
```

```
          </goals>
```

```
          <configuration>
```

```
            <!-- Optional. Specify CommandLine user -->
```

```
            <commandlineUser>jobbuilder@talend.com</commandlineUser>
```

```
            <!-- Optional. Jvm Parameters for local(script) mode -->
```

```
            <!-- <jvmArguments>-ea -Xms512m -Xmx1300m -XX:
```

```
+CMSCClassUnloadingEnabled -XX:MinHeapFreeRatio=10 -XX:MaxHeapFreeRatio=20
```

```
-XX:+HeapDumpOnOutOfMemoryError</jvmArguments> -->
```

```
            <!-- Optional. Parameter used to filter on specific Job status (TEST, DEV,
PROD, etc) -->
```

```
            <!-- <itemFilter>(status=TEST)or(status=PROD)or(status="")</
```

```
itemFilter> -->
```

```
          </configuration>
```

```
        </execution>
```

```
      </executions>
```

```
    </plugin>
```

```
  </plugins>
```

```
</build>
```

```
<pluginRepositories>
```

```
  <!-- everything through Maven Central Proxy -->
```

```
    <pluginRepository>
```

```

        <id>Central</id>
        <name>Central</name>
        <url>http://<host>:<port>/nexus/content/repositories/central</url>
    </pluginRepository>
    <pluginRepository>
        <id>thirdparty</id>
        <name>thirdparty</name>
        <url>http://<host>:<port>/nexus/content/repositories/thirdparty</url>
    </pluginRepository>
</pluginRepositories>

</project>

```

- Alternative: **Remote generation** (server mode)

When you choose to generate sources in server mode (**generate**), you need the Talend CommandLine to be launched (the best practice would be to have it installed as a service) and you then need to specify its access parameters in the `GenerateSources` Jenkins project.

```

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://
www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/
xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>org.talend</groupId>
    <artifactId>buildsources</artifactId>
    <version>A.B.C</version>
    <packaging>pom</packaging>

    <build>
        <plugins>
            <plugin>
                <groupId>org.talend</groupId>
                <artifactId>ci.builder</artifactId>
                <version>A.B.C</version>
                <executions>
                    <execution>
                        <phase>generate-sources</phase>
                        <goals>

```

```

        <!-- server mode -->
        <goal>generate</goal>
    </goals>
    <configuration>
    </configuration>
</execution>
</executions>
</plugin>
</plugins>
</build>

<pluginRepositories>
    <!-- everything through Maven Central Proxy -->
    <pluginRepository>
        <id>Central</id>
        <name>Central</name>
        <url>http://<host>:<port>/nexus/content/repositories/central</url>
    </pluginRepository>
    <pluginRepository>
        <id>thirdparty</id>
        <name>thirdparty</name>
        <url>http://<host>:<port>/nexus/content/repositories/thirdparty</url>
    </pluginRepository>
</pluginRepositories>

</project>

```

For more information on Maven life cycle and goals, see [the Apache Maven documentation](#).

2. You also have the possibility to customize some of the information that are generated by default for your project from Talend Studio, through the menu **Project Settings > Build > Maven > Default > Project**: from here you can edit the default <groupId> that will be used when publishing on Nexus, the <version> as well as other default information.

Specify the parameters to generate the sources

How to configure the GenerateSources project on Jenkins to generate the sources of your Jobs and Tests.

- You have linked your GenerateSources Jenkins project to your Git/SVN project, see [Link the Jenkins project to the SVN/Git project](#) on page 23.

- You have created the `BuildJob_pom.xml` file holding the instructions to generate sources, see [Create the POM file to generate sources](#) on page 24.

- In the **GenerateSources** project configuration, set the path to the `BuildJob_pom.xml` file holding the instructions to generate sources in the **Root POM** field of the **Build** area.
- In the **Goals and options** field, enter the Maven instruction to generate sources.
`org.talend:ci.builder:6.4.1:<local->generate`

Use `local-generate` to generate sources in script mode (recommended), or use `generate` to generate sources remotely.

- (Optional) If you want to perform a non-local generation (not recommended), set the other necessary Maven parameters in the **MAVEN_OPTS** field of the **Advanced** part of the **Build** area (plugin execution information as well as parameters needed to generate the sources).

Tip: Note that the Maven documentation of the CI Builder plugin is embedded in the CI Builder archive file, see the files in the `<CIBuilder_home>/help` folder for more information.

Field	Parameters
MAVEN_OPTS	<code>-Dcommandline.workspace=<jenkins_path>/workspace/GenerateSources/projects</code> <code>-Dcommandline.host=localhost</code> <code>-Dcommandline.port=8002</code> <code>-Dcommandline.user=xxx@talend.com</code> <code>-DprojectsTargetDirectory=<jenkins_path>/workspace/RunTests/projectSources</code> <code>-DitemFilter=(type=process)and(path=TEST)</code>

For more information on these parameters, see [Create the POM file to generate sources](#) on page 24.

For examples of filters you can apply to the execution of your project items, see [How to filter the execution of your project on a single Job or specific Jobs](#) on page 30.

Tip: The only requirement is to store all projects in the workspace folder, this means you have the possibility create a Jenkins task that clones the projects and copy them (folders only) to the workspace.

- If your project is on Git, you need to specify the folder in which your projects will be checked out (**Use custom workspace**) in the **Advanced** part of the **Build** area. This path needs to be the same as the one you will use in your Talend CommandLine (see [Executing the Tests](#) on page 32).

`<jenkins_path>/workspace/GenerateSources/projects`

You also have the possibility to check out one specific project in your repository by entering its name in **Additional Behaviours > Sparse Checkout** paths.

Example:

`CI_PROJECT`

- In the **Advanced** part of the **Build** area, select **Settings file in filesystem** in the **Settings file** list and point to the Maven settings file of the Studio with all the dependencies needed to generate the sources.

<studio_path>/configuration/maven_user_settings.xml

6. (Optional) Add a post-build action to get it to run the next project, the `RunTests` project, that will run all the tests.

How to filter the execution of your project on a single Job or specific Jobs

How to filter the execution of your project items to build only the Job(s) you want. To do that, you need to declare the filter in the Maven parameters entered when configuring the build project which generates your project sources on the Continuous Integration server.

You have created the build project to generate your sources (`GenerateSources`) on your Continuous Integration server, Jenkins for example. See [Creating the Jenkins projects](#) on page 22.

1. Open the configuration page of the `GenerateSources` build project.
2. in the **MAVEN_OPTS** field of the **Advanced** part of the **Build** area (plugin execution information as well as parameters needed to generate the sources)., define the `<itemfilter>` parameter value according to your needs:

- **Note:**

There should be no space between the filters and the operators (or and and).

filter on Job types:

`-DitemFilter=(type=process)` to filter on all Standard Jobs of the project

`-DitemFilter=(type=process_mr)` to filter on all Big Data Map/Reduce and Spark Batch Jobs of the project

`-DitemFilter=(type=process_storm)` to filter on all Big Data Storm and Spark Streaming Jobs of the project

`-DitemFilter=(type=route)` to filter on all Routes of the project

- filter on Job labels:

`-DitemFilter=(type=process_mr)and(label=job_ProcessWeatherData)` to filter on one specific Big Data Map/Reduce Job named `job_ProcessWeatherData`

`-DitemFilter=(type=process)and(label%job_dev*)` to filter on Jobs which names start with `job_dev`

- filter on Job paths:

`-DitemFilter=(type=process)and(path=Integration)` to filter on Jobs located in subfolder called `Integration`

`-DitemFilter=(type=process)and(path%Integration*)` to filter on Jobs located in subfolders with a name starting with `Integration`

- filter on the person who created the Job:

`-DitemFilter=(type=process_storm)and(author=rbunch@talend.com)` to filter on Big Data Storm Streaming Jobs whose author ID is `rbunch@talend.com`

- exclusion filter:

`-DitemFilter=(!path=sandbox)and(type=process)and(label%job_Export*)or(label%job_Monitor*)` to filter on Jobs with a name starting with `job_Export` or `job_Monitor`, but that are not in the `sandbox` folder.

`-DitemFilter=(!path%MainProcess/Import*)and(type=process)and(label%job_Export*)` to filter on Jobs with a name starting with `job_Export`, but that are not in the subfolders with a name starting with `Import` under the `MainProcess` folder.

Example of filter applied in order to execute Big Data Spark Batch Jobs located in the subfolders with a name starting with `Export` under the `MainProcess` folder, and with the exception of the Job named `job_batch_feature22`.

```
-DitemFilter=(type=process_mr)and(!label=job_batch_feature22)and(path
%MainProcess/Export*)
```

3. Save your changes and close the file.

When you will execute your project on your Continuous Integration server, the filter will be applied and only the Jobs you have filtered will be generated and executed.

Configuring Jenkins to run all tests

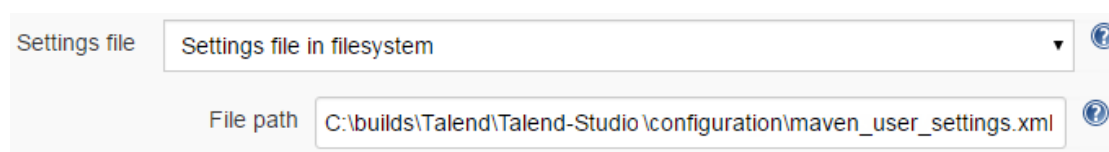
How to configure the `RunTests` project on Jenkins to execute all Tests available in your project.

You have created and configured the `GenerateSources` Jenkins project to retrieve the sources of your Jobs and Tests, see [Creating the Jenkins projects](#) on page 22 and [Configuring Jenkins to generate the sources](#) on page 23.

1. In the `RunTests` project configuration, set the path to the `<jenkins_workspace>/RunTests/projectSources/pom.xml` file generated previously in the **Root POM** field of the **Build** area.
2. In the **Goals and options** field, enter the Maven instruction run all Tests.
`test -fn -e`

This Jenkins project retrieves the POM file generated during source generation and use it to run all available Test Cases.

3. In the **Advanced** part of the **Build** area, select **Settings file in filesystem** in the **Settings file** list and point to the Maven settings file of the Studio with all the dependencies needed to run all Tests.



4. (Optional) Add a post-build action to trigger the `DeployToNexus` build. Note that, if you add this step, artifacts will be deployed automatically to Nexus after Test executions following default instructions (default groupID, default version, etc.) held in the source POM file. These default values can be edited through the Studio project settings before generating the sources and their corresponding POM file.

Configuring Jenkins to deploy Jobs on Nexus

How to configure the `DeployToNexus` project on Jenkins to deploy all Jobs available in your project to the Nexus artifact repository.

You have generated your sources and have run all corresponding Tests, see [Configuring Jenkins to generate the sources](#) on page 23 and [Configuring Jenkins to run all tests](#) on page 31.

1. In the `DeployToNexus` project configuration, set the path to the `<jenkins_workspace>/RunTests/projectSources/pom.xml` file generated previously in the **Root POM** field of the **Build** area.

2. In the **Goals and options** field, enter the Maven instruction to deploy all the Jobs available into the Nexus artifact repository of your choice.

```
deploy -fn -e
```

This project deploys artifacts on Nexus by using the instructions held in the POM file.

Executing the Tests

How to execute the Test cases you created in Talend Studio on Jenkins.

1. Make sure your Talend CommandLine application points to the Jenkins workspace where your project sources are stored:
 - If you installed Talend CommandLine as a service on Windows, you need to go to the service installation directory and open the file `<service_path>/conf/wrapper.conf` to edit the `wrapper.app.parameter.7` parameter value and point to the Jenkins workspace where your project sources are stored.
 - If you installed Talend CommandLine as a service on Unix, you need to edit the file `start_cmdline.sh` that calls the service script.
 - If you did not install Talend CommandLine as a service, edit the `.bat` or `.sh` file of Talend CommandLine to change the data parameter value.

```
Talend-Studio-win-x86_64.exe nosplash -application org.talend.commandline.CommandLine -consoleLog -data C:\Users\lgaudens.TALEND\.jenkins\workspace\BuildSources\projects startServer -p 8002
```

2. Start the Talend CommandLine service.
3. Launch the parent Jenkins task `GenerateSources` to generate the sources using the Talend CommandLine, to trigger the execution of all Test Cases created in the Studio and to deploy the tested Jobs in Nexus at once.

The tests are executed and you can see their results displayed:

- in Jenkins: the detail of your results can be found in the **Console Output** view of the specific builds.

Example in `RunTests` project (here you can see one failure on the execution of the test named `test_process_client_file`):

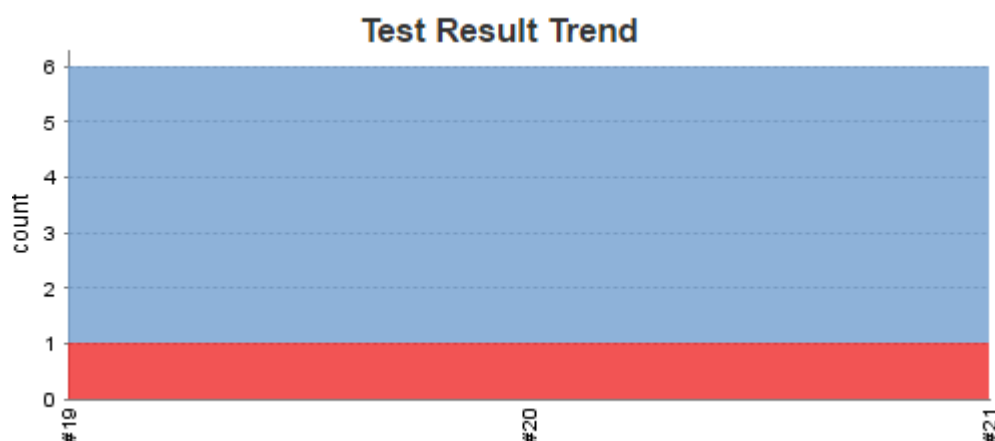
```
Tests run: 2, Failures: 1, Errors: 0, Skipped: 0, Time elapsed: 0.068 sec <<< FAILURE!
Results :
Failed tests:  ci_project.job_load_california_clients_0_1.test_process_client_file_0_1.test_process_client_fileTest.testDefault(): Failure=1

Tests run: 2, Failures: 1, Errors: 0, Skipped: 0
[ERROR] There are test failures.
```

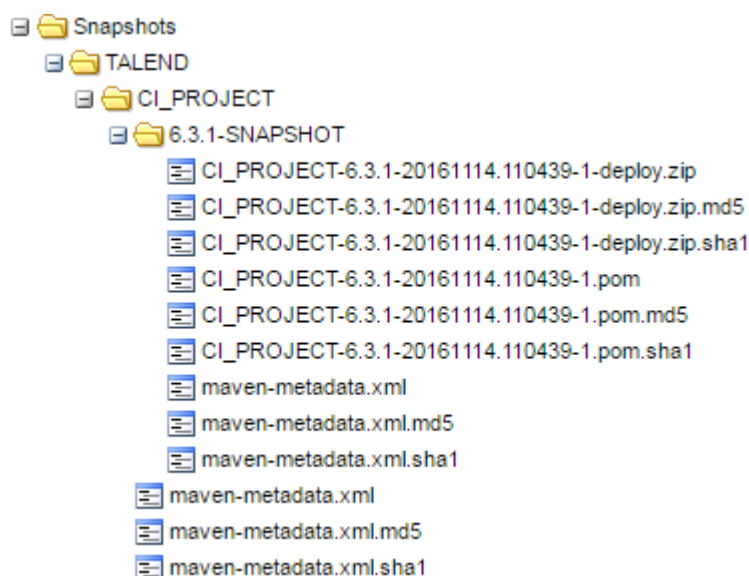
Example in `DeployToNexus` project (here 6.3.1-SNAPSHOT artifacts are being deployed to the TALEND group in the repository):


```
[INFO] Using alternate deployment repository tac::default::http://localhost:8081/nexus/content/repositories/snapshots/
Downloading: http://localhost:8081/nexus/content/repositories/snapshots/TALEND/CI_PROJECT/6.3.1-SNAPSHOT/maven-metadata.xml
Uploading: http://localhost:8081/nexus/content/repositories/snapshots/TALEND/CI_PROJECT/6.3.1-SNAPSHOT/CI_PROJECT-6.3.1-20161114.110439-1.pom
Uploading: http://localhost:8081/nexus/content/repositories/snapshots/TALEND/CI_PROJECT/6.3.1-SNAPSHOT/CI_PROJECT-6.3.1-20161114.110439-1.pom
KB/sec)
Downloading: http://localhost:8081/nexus/content/repositories/snapshots/TALEND/CI_PROJECT/maven-metadata.xml
Uploading: http://localhost:8081/nexus/content/repositories/snapshots/TALEND/CI_PROJECT/6.3.1-SNAPSHOT/maven-metadata.xml
Uploading: http://localhost:8081/nexus/content/repositories/snapshots/TALEND/CI_PROJECT/maven-metadata.xml
Uploading: http://localhost:8081/nexus/content/repositories/snapshots/TALEND/CI_PROJECT/maven-metadata.xml (276 B at 6.7 KB/sec)
Uploading: http://localhost:8081/nexus/content/repositories/snapshots/TALEND/CI_PROJECT/6.3.1-SNAPSHOT/CI_PROJECT-6.3.1-20161114.110439-1-deploy.zip
Uploading: http://localhost:8081/nexus/content/repositories/snapshots/TALEND/CI_PROJECT/6.3.1-SNAPSHOT/CI_PROJECT-6.3.1-20161114.110439-1-deploy.zip
at 29.1 KB/sec)
Uploading: http://localhost:8081/nexus/content/repositories/snapshots/TALEND/CI_PROJECT/6.3.1-SNAPSHOT/maven-metadata.xml
Uploading: http://localhost:8081/nexus/content/repositories/snapshots/TALEND/CI_PROJECT/6.3.1-SNAPSHOT/maven-metadata.xml (806 B at 15.7 KB/sec)
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
```

- in Jenkins: a diagram that represents graphically the status of the last executions in the RunTests projects (here you can see one failure out of a total of six tests):



- in your Nexus web application: you can find the artifacts deployed in the defined repository:



Continuous Integration: Deploying to QA and Production environments

In a Production environment, the only tools needed are:

- the Talend Administration Center application to schedule the execution of the Jobs,

- an execution server, to deploy and execute the Jobs,
- Talend Repository Manager to pass items to the QA and Production environments.

The Job generation step is indeed skipped as Talend Administration Center is able to retrieve pre-generated Jobs from archive files. The Talend CommandLine application is thus no longer needed.

Likewise, the Production environment does not require to be linked to SVN or GIT as the Jobs that were pre-generated in the previous environments can be imported and executed in a "no-storage" project.

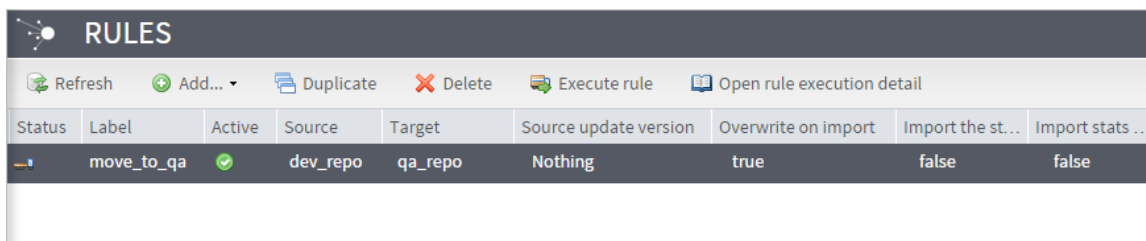
Best practice: Once a feature is released or a bug is fixed, the best practice is to use SVN or GIT branches and tags to record this change of status. They are used to identify a specific version of a project at a given time, for example the feature that has been developed, tested and validated, and to identify the environment used to make the feature work (CommandLine/Job versions, project properties etc.).

Passing items to the QA environment

Talend Repository Manager is a Talend web tool that allows you to move generated Jobs without recompiling them from development to production. This means that the generation of Jobs and thus the use of the Talend CommandLine application will no longer be required.

The archive files holding the Job sources, called standalone Jobs, are therefore sufficient to perform validation and regression tests on them. A benefit of this is that the Talend CommandLine version does not need to be maintained to the same version level as the Talend CommandLine application used in the Development environment, which simplifies the process. For more information on development environments, see [Talend Development life cycle](#) on page 6.

Note that, from the Talend Repository Manager application, the migration from one environment to the other can be scheduled to automate the execution of migration rules.



RULES								
Refresh Add... Duplicate Delete Execute rule Open rule execution detail								
Status	Label	Active	Source	Target	Source update version	Overwrite on import	Import the st...	Import stats ...
	move_to_qa		dev_repo	qa_repo	Nothing	true	false	false

For example, you can set a rule in Talend Repository Manager to select project items to be migrated (like Jobs, Contexts, Metadata, etc.) between environments.

Global

Type: Rule


Label:


Description:

Active: ☒

Source Repositories

Source: ▼

Filter: 

Projects: 

Source update version: ▼

Source change status:

Export reference projects: ☒

Export dependencies: ☒

Last item version only: ☐

Target Repositories

Target: ▼

Overwrite on import: ☒

 Save  Cancel

Once rules are set, you can automate and schedule their execution. For more information, see [Passing items to the Production environment](#) on page 35.

Passing items to the Production environment

In the same way that developed items are moved to the Quality Assurance environment, they are moved to the Production environment by an Operation Manager via the Talend Repository Manager application once the features are tested, the reports are sent and the errors are fixed.

Note that, from the Talend Repository Manager application, the migration from one environment to the other can be scheduled to automate the execution of migration rules and that the application supports the "no storage" project mode.

RULES								
Refresh Add... Duplicate Delete Execute rule Open rule execution detail								
Status	Label	Active	Source	Target	Source update version	Overwrite on import	Import the st...	Import stats ...
	move_to_prod		qa_repo	prod_repo	Nothing	true	false	false

For example, if you have set some rules in Talend Repository Manager to select project items to be migrated, you can define, from the **Scheduling** page, at what time they will be executed.

RULES								
Refresh Add... Duplicate Delete Execute rule Open rule execution detail								
Status	Label	Active	Source	Target	Source update version	Overwrite on import	Import the st...	Import stats ...
	move_to_qa		dev_repo	qa_repo	Nothing	true	false	false

Retrieving generated and validated Jobs

In the Production environment, Jobs are pre-generated and validated.

The user has to connect to the corresponding project in Talend Administration Center web application, which is linked to the Studio, in order to retrieve all items previously published to the Nexus artifact repository after they have been validated in the Quality Assurance environment and to execute the desired Jobs.

Execution task

Execution task

Label:




ta_load_clients_to_mysql

Description:

Active:

☒

Job:

Project:

di_project

Branch:

trunk

Name:

California1

Version:

0.1

Context:

Production

Apply context to children:

☐

Regenerate job on change:

☐

Log4j Level:

Info

Execution server:

ci_server

Statistic:

enabled

On unavailable JobServer:

Wait

Timeout(s):

120

Pause triggers on error:

☒

Save

Cancel

Once you have used the **Select Job from Nexus** option (recommended), you can see that the main Job properties are complete and read-only. You only need to select the context (if several), the log level to be applied and the execution server you have previously started in order to create an execution task on this Job.

Note that the selected execution server needs to have access to the project (access is defined via the **Server Project Authorizations** page of the web application).

Maintaining

Once a product or feature is live, developers might still need to update it in order to improve its performances, meet new requirements or correct some errors discovered in the product. Editing a product or a feature after delivery is possible thanks to version control and Git/Subversion software capabilities.

Best Practices: in addition to version control and SVN/GIT capabilities, to maintain a software it is recommended to have an easy-to-read source code, a documentation that describes the feature or product and to perform regression tests.

Versioning, branching and tagging

In the Talend products, project sources management is mainly handled by Subversion (SVN) and or GIT, which are complex version control systems that enable users to have different copies of the same project in different branches and tags.

SVN and GIT allow the collaboration between team members and the management of versions and changes made on the Talend projects files and directories.

SCM Concepts

Subversion and GIT allow multiple developers to work on the same project by committing/pushing and retrieving their changes to/from the server.

- Branching allows developers to isolate code and work independently without disturbing the main development line (the 'trunk' on SVN, the 'master' on GIT).

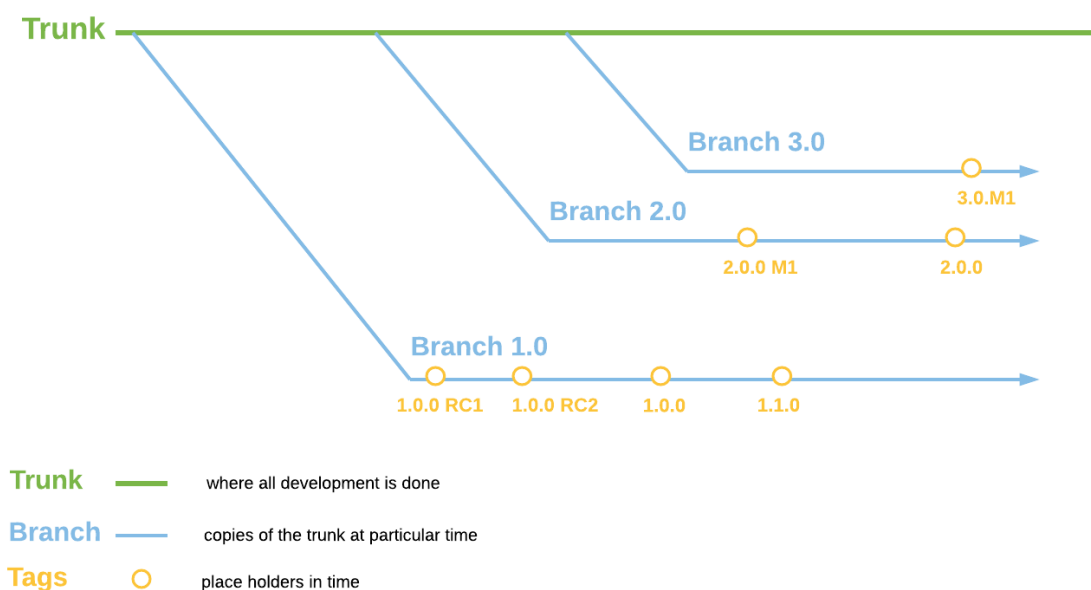
A **Branch** is a copy of the project taken at a specific point in time, for example when preparing a new release for promotion to another environment. The copy can be taken from the main development line ('trunk' or 'master'), from another Branch or from a Tag. A Branch is editable and can therefore "fork" from the original source. In this situation, reconciliation between the original source (branch or trunk) and the forked Branch must be done manually.

- Tagging allows developers to mark a particular revision as important in the development process.

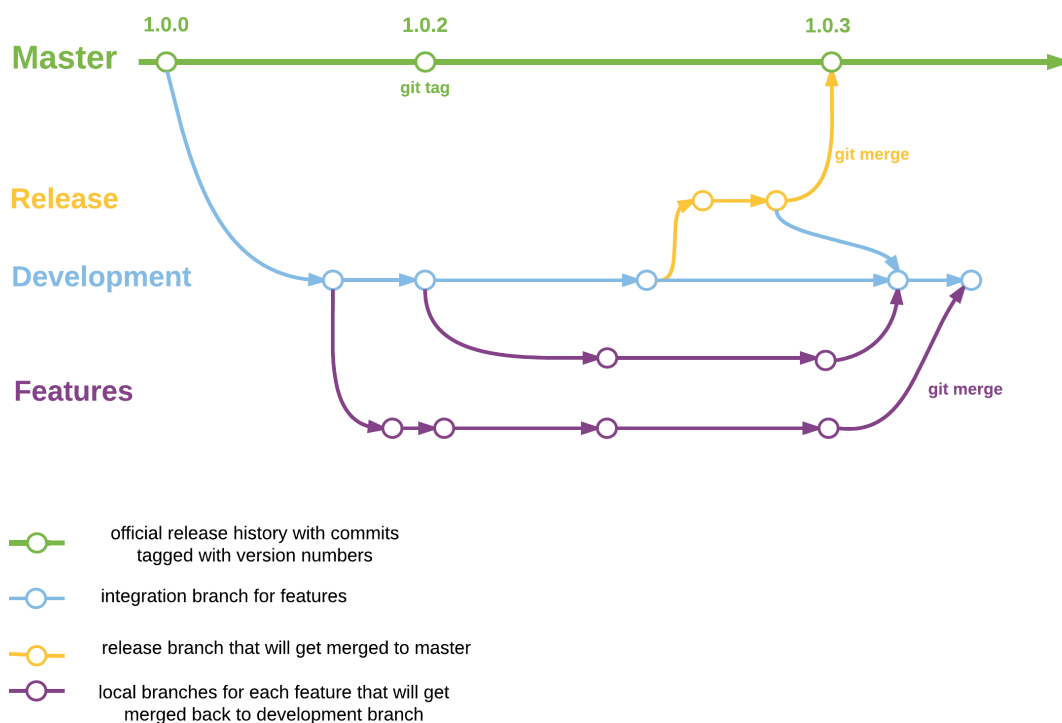
A **Tag** is similar to a Branch, but is a read-only snapshot of a Trunk/Master or Branch. Once created it cannot be edited in any way. However it is possible to create a new Branch (which is editable) from a Tag.

Development teams are expected to define the workflow they want to use. For more information on the differences between SVN and Git, see this [GitHub documentation article](#).

The following diagram shows the generic process of SVN branching and tagging.



The following diagram shows the generic process of GIT branching and tagging.



For more information on Git flows, see this [GitHub visual tutorial](#).

Branching and tagging in Talend

Best Practices and tips for using Git and SVN with the Talend products.

Best practices for Git:

- All developers should work on Jobs in the 'master' of the Development environment.
- For each development (bug, new features), a local feature branch can be created and then pushed to the remote repository when the developer wants his or her work to be reviewed.

On a Git remote project, Branches are either remote (**Remote Mode**, default mode) or local (**Local Mode**) branches. When a developer works in local or offline mode on a Git project, he/she is working on the local branch associated with the branch he/she last worked on, and changes are automatically committed to the local Git repository. Once the feature is ready, the developer needs to push the commits to make it a remote development branch before merging this remote branch to the 'master' when the feature is tested.

Unlike SVN, in Git a remote branch is created on the whole repository and thus is available in all projects of this repository.

- Each time developers reach a Milestone (for releases, features, sprints, etc.), a Tag should be used. A new Tag should be created when the feature is ready for delivery (Production environment). If the tagged version needs bug fixing, a branch can be created from the Tag and the fix can then be included in the 'trunk'/'master'.
- It is recommended to define patches as minor versions and full releases as major versions.
- When working in a specific branch, it is recommended to filter the project on this branch using the Git Branches whitelist option in order to reduce the use of disk resources and improve performances.

Best practices for SVN:

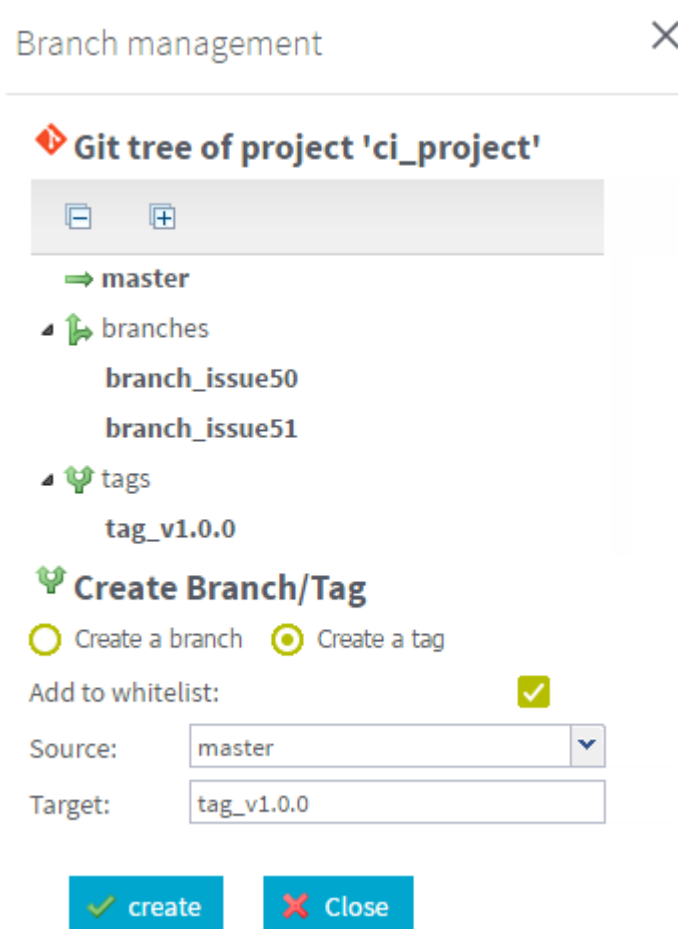
- All developers should work on Jobs in the 'trunk' of the Development environment.

- When a new intermediate release is required, the 'trunk' needs to be copied in its entirety to the new Branch or Tag.
- Each time developers reach a Milestone (for releases, features, sprints, etc.), a Tag should be used.
- A new Tag should be created when the feature is ready for delivery (Production environment). If the tagged version needs bug fixing, a branch can be created from the Tag and the fix can then be included in the 'trunk'.
- It is recommended to define patches as minor versions and full releases as major versions.
- When working in a specific branch, it is recommended to filter the project on this branch using the SVN Branches whitelist option in order to reduce the use of disk resources and improve performances.

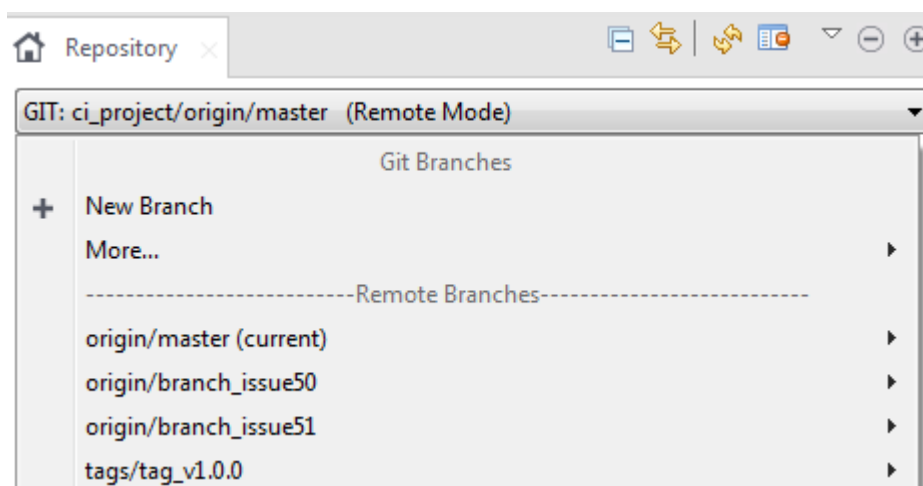
Talend provides a number of tools for branches and tags management, which are covered below.

You can create and edit branches and tags:

- on the **Projects** page of Talend Administration Center with the **Branch Management** option,



- using the MetaServlet with the `createBranch` and `createTag` commands,
- using the Repository Branch management menu in the Talend Studio Repository.



Once the branch or tag has been created:

From Talend Administration Center, you can create an execution task on a Job located in this specific branch or tag via the **Job Conductor** page.

From Talend Studio, you can also switch between branches and tags.

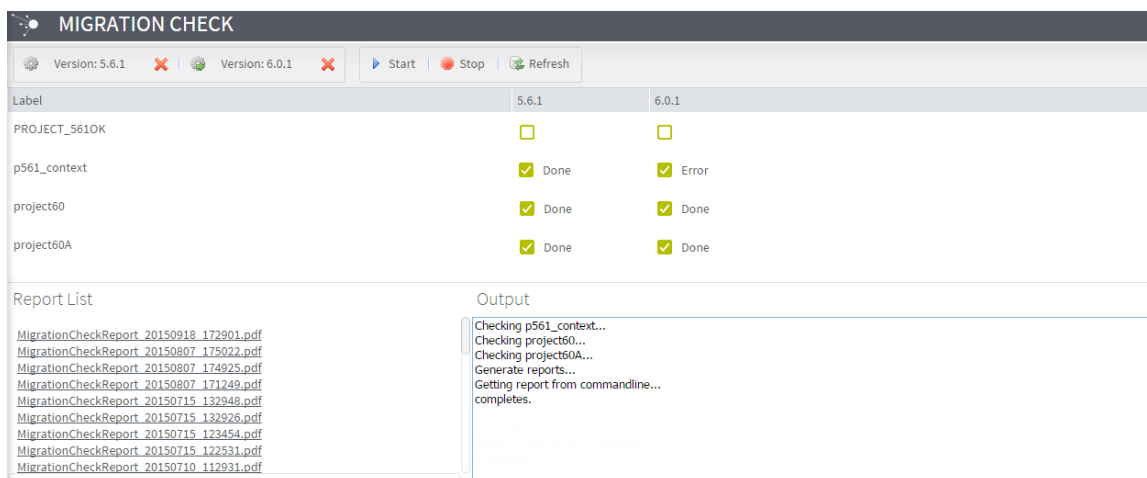
From Talend Repository Manager, you can move your projects from one branch to another. For more information, see [Passing items to the Production environment](#) on page 35.

Tags also allow you to fix errors on the exact same version as the one used to deploy the Jobs during the previous development phases.

Migrating projects

If you migrated to a newer version of Talend Administration Center and want to retrieve your existing projects, you may want to migrate these projects. Talend Administration Center allows you to select individually the projects to be migrated and to generate corresponding .pdf reports using the Talend CommandLine applications from the **Migration Check** page.

From the **Migration Check** page, click the CommandLine buttons on the top toolbar to configure the connections to both source (old) and target (new) Talend CommandLine applications, then enter the paths to the local directories where the database and report will be stored. The migration starts and reports are generated.



This migration phase requires you to re-install the Talend solutions.

Note that, by default, the Migration Check reports give you details about the compilation status of the Jobs generated in the latest version of the product. Talend offers you the possibility to optimize the migration reports by installing patches in the Talend CommandLine in order to know more precisely whether the Jobs were generating successfully in the previous versions. For more information, see the article [How to improve the migration check report](#).