

# **Jenkins TestLink Plug-in Tutorial**

**Bruno P. Kinoshita**

**César Fernandes de Almeida**

---

# Jenkins TestLink Plug-in Tutorial

Bruno P. Kinoshita

César Fernandes de Almeida

French Translation.: Flóreal Toumikian, Olivier Renault

Review and suggestions on how explain some topics of this tutorial.: Oliver Merkel

## Abstract

The Jenkins TestLink Plug-in Tutorial is intended to provide a better understanding on how to use the plug-in to integrate Jenkins and TestLink. It is an Open Source project, so contributions and suggestions are welcome.

---

# Table of Contents

1. An Introduction to Jenkins .....	1
2. An Introduction to TestLink .....	3
3. Jenkins and TestLink Integration .....	5
4. TestLink Configuration .....	6
4.1. Installing TestLink .....	6
4.2. Creating a Test Project .....	9
4.3. Creating and assigning a Custom Field .....	11
4.4. Specifying Test Suite and Test Cases .....	13
4.5. Create a Test Plan and add the Test Cases .....	16
5. Jenkins Configuration .....	20
5.1. Installing Jenkins .....	20
5.2. Installing and configuring Jenkins TestLink Plug-in .....	20
5.3. Creating a job in Jenkins .....	22
5.4. Explaining the Job configuration parameters .....	25
TestLink Version .....	26
Test Project Name .....	26
Test Plan name .....	26
Build Name .....	26
Custom fields .....	26
Key Custom Field .....	26
Single Build Steps .....	26
Before iterating all test cases Build Steps .....	26
Iterative Test Build Steps .....	26
After iterating all test cases Build Steps .....	26
JUnit, TestNG and TAP report files patterns. ....	26
Transactional execution? .....	27
Failed tests mark build as failure .....	27
Environment variables .....	27
6. Executing Automated Test Cases .....	28
7. Strategies to find Test Results .....	32
7.1. JUnit .....	32
7.2. TestNG .....	32
7.3. TAP - Test Anything Protocol .....	33
8. Advantages of using Jenkins TestLink Plug-in .....	34
9. Appendix .....	35
9.1. Adding attachments to your test results .....	35
9.2. Using Platforms .....	35
9.3. Plug-in behavior in distributed environments .....	36
9.4. How to use the plug-in with SSL or basic authentication? .....	36
Bibliography .....	37

# 1. An Introduction to Jenkins

Jenkins (née Hudson) is a continuous integration server written in Java. It can be downloaded from <http://www.jenkins-ci.org> [<http://www.jenkins-ci.org>] as an executable war. It means that you can run it with **java -jar jenkins.war**. Jenkins comes with an embedded Servlet Container (Winstone) but you also have the option to deploy the war to an application server like Tomcat, Jetty, JBoss, etc. Jenkins does not use any database to store its configuration. Jenkins uses XStream to save data as XML files.



# Jenkins

With Jenkins you can create, monitor and schedule jobs. There are plug-ins for almost anything that you may think about, from different SCMs (git, mercury, SVN, CVS) to plug-ins for integrating your Jenkins with Selenium, Gerrit, TestLink and other tools.

Here is a summarized list of Jenkins features that is available in [Jenkins Wiki](http://wiki.jenkins-ci.org/display/JENKINS/TestLink+Plugin) [<http://wiki.jenkins-ci.org/display/JENKINS/TestLink+Plugin>]:

1. Easy installation
2. Easy configuration
3. Change set support
4. Permanent links
5. RSS/E-mail/IM Integration
6. After-the-fact-tagging
7. JUnit/TestNG test reporting
8. Distributed builds
9. File fingerprinting
10. Plugin Support

For this guide you will need to download the latest and greatest version of Jenkins. At the moment that this guide is being written, the current version is *1.433*. However it is very likely that what you will learn here will work with newer versions of Jenkins. Jenkins development team does a great job not only developing cool features, but also keeping backward compatibility between versions.

**Jenkins** search ?

[New Job](#) [People](#) [Build History](#) [Manage Jenkins](#)

Welcome to Jenkins! Please [create new jobs](#) to get started. [add description](#) [ENABLE AUTO REFRESH](#)

**Build Queue**  
No builds in the queue.

**Build Executor Status**

#	Status
1	Idle
2	Idle

[Help us localize this page](#) Page generated: Oct 9, 2011 11:23:55 PM [Jenkins ver. 1.433](#)

Jenkins is licensed under the MIT License and its code is hosted at GitHub - <http://github.com/jenkinsci> [<http://github.com/jenkinsci>].

## 2. An Introduction to TestLink

TestLink is an open source test management tool written in PHP. In order to install TestLink you will need a HTTP server with PHP 5 and a database. The databases currently supported by TestLink out of the box are MySQL, Postgre SQL and MS SQL, although there are users that managed to use Oracle too. You can download it from <http://www.teamst.org> [<http://www.teamst.org>].



In TestLink you have, among other features, Test Plans, Requirement management, Baselines, Custom Fields, Test Suite with Test Cases and Reporting. For external access, there is an XML-RPC API available (it is not enabled by default). Other two nice features in TestLink are the versioning of the some entities, like Test Case and Requirements, and the ability to import and export data in different formats.

There are bindings for the TestLink XML-RPC API available for Java, Perl, PHP, Ruby and other programming languages. The integration between Jenkins and TestLink is done using TestLink Java API - <http://testlinkjavaapi.sourceforge.net> [<http://testlinkjavaapi.sourceforge.net>].

At the moment that this guide is being written the latest version of TestLink is 1.9.3. While the Jenkins version is not a big issue for this integration, the same cannot be said about TestLink. This guide is intended for Testlink 1.9.3 XML-RPC API. In case there are other versions available at the moment that you are reading

this guide, please consult the `plug-in Wiki` [<http://wiki.jenkins-ci.org/display/JENKINS/TestLink+Plugin>] for further information on this.

TestLink is licensed under the GPL License and its code is hosted at gitorious - <http://www.gitorious.org/testlink-ga/testlink-code> [<http://www.gitorious.org/testlink-ga/testlink-code>].

## 3. Jenkins and TestLink Integration

The integration between Jenkins and TestLink can be achieved with Jenkins TestLink Plug-in. The plug-in resides in Jenkins, being able to use some of its nice features like executing multiple jobs, scheduling jobs, distributed execution and a plethora of plug-ins. To retrieve the data from TestLink for tests, as well as to report the test case execution status to TestLink, the plug-in uses TestLink XML-RPC API.

This way, while Jenkins handles the execution of jobs and tasks such as downloading source code from an SCM, TestLink maintains the structure of your tests, as well as other assets as Test Plan, Custom Fields, Reports and Baselines.

We will see how to configure our environment for this integration throughout the next chapters. While it is good to have a good written explanation on how this integration works, probably it is a lot easier to understand how it works with a hands-on. The rest of this guide explains how to configure Jenkins and TestLink to run automated tests from a sample test project. This test project is a Java project that uses Maven and TestNG and you can download the source code from <https://github.com/kinow/jenkins-testlink-plugin-tutorial> [<https://github.com/kinow/jenkins-testlink-plugin-tutorial>] (as well as the DocBook source for this book).

The current version of Jenkins TestLink Plug-in while this book is being written is 3.0.2.

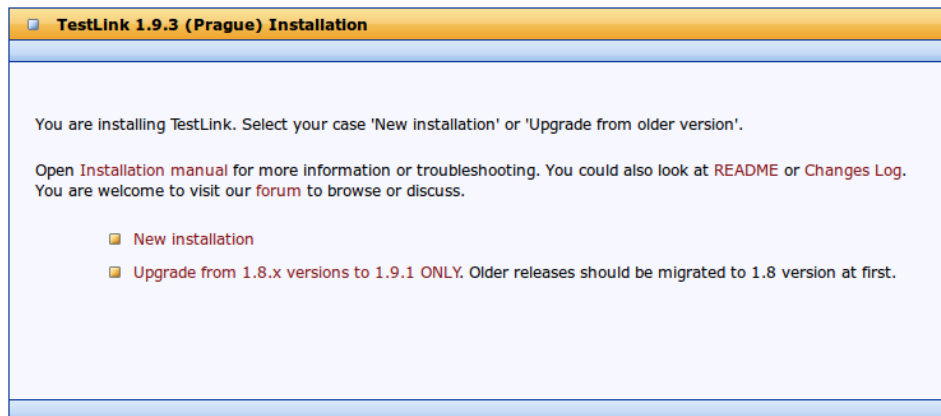
Jenkins TestLink Plug-in is licensed under the MIT License and its code is stored in github - <http://github.com/jenkinsci/testlink-plugin> [<http://github.com/jenkinsci/testlink-plugin>].

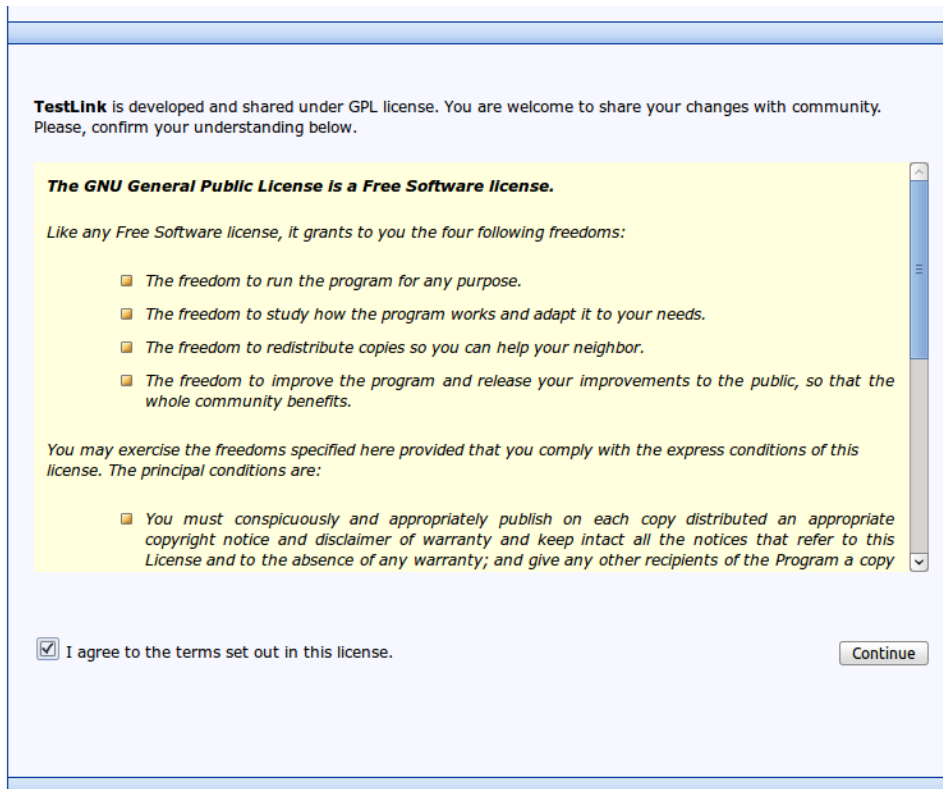


## 4. TestLink Configuration

### 4.1 Installing TestLink

In this part of the tutorial we will show how to install and configure TestLink. Let's start by downloading *testlink-1.9.3.tar.gz* from <http://www.teamst.org>. Decompress it with **tar -zxvf testlink-1.9.3.tar.gz**. Move the directory created to your HTTP server root directory and open <http://localhost/testlink-1.9.3> [<http://localhost/testlink-1.9.3>] in your browser.





Now the installation wizard will guide you through the rest of the installation. But before going on, we need to create a database in MySQL.

```
mysql> create database testlink;
```

The next step is to create a user that TestLink will use to access the database.

```
mysql> grant all privileges on testlink.* to 'testlink' identified by 'testlink';
```

```
mysql> flush privileges;
```

max_execution_time)	<b>hundred of test cases (edit php.ini)</b>
Checking maximal allowed memory (Parameter memory_limit)	<b>OK (128 MegaBytes)</b>
Checking if Register Globals is disabled	<b>OK</b>
Checking MySQL Database	<b>OK</b>
Checking Postgres Database	<b>Failed! Postgres Database cannot be used.</b>
Checking GD Graphic library	<b>OK</b>
Checking LDAP library	<b>Failed! LDAP library not enabled. LDAP authentication cannot be used. (default internal authentication will works).</b>
Checking JSON library	<b>OK</b>

**Read/write permissions**

Checking if /var/www/testlink-1.9.3/gui/templates_c directory exists	<b>OK</b>
Checking if /var/www/testlink-1.9.3/gui/templates_c directory is writable	<b>OK</b>
Checking if /var/www/testlink-1.9.3/logs directory exists	<b>OK</b>
Checking if /var/www/testlink-1.9.3/logs directory is writable	<b>OK</b>
Checking if /var/www/testlink-1.9.3/upload_area directory exists	<b>OK</b>
Checking if /var/www/testlink-1.9.3/upload_area directory is writable	<b>OK</b>

Your system is prepared for TestLink configuration (no fatal problem found).

[Continue](#)

Set an existing database user with administrative rights (root):

**Database admin login**

**Database admin password**

*This user requires permission to create databases and users on the Database Server. These values are used only for this installation procedures, and is not saved.*

Define database User for Testlink access:

**TestLink DB login**

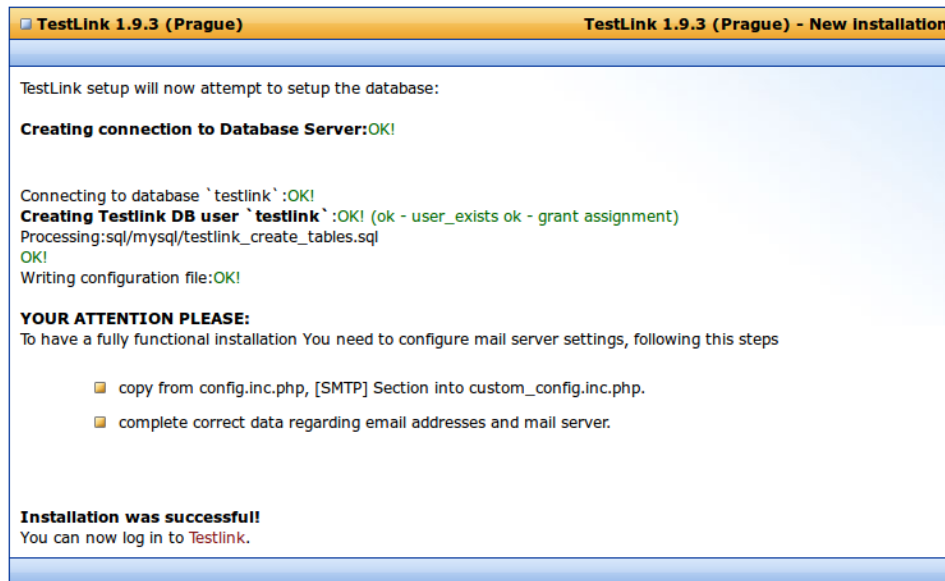
**TestLink DB password**

*This user will have permission only to work on TestLink database and will be stored in TestLink configuration. All TestLink requests to the Database will be done with this user.*

After successfull installation You will have the following login for TestLink Administrator:

login name: admin  
password : admin

[Process TestLink Setup!](#)



If everything worked out correctly you should be asked to log in with user admin and password admin. The examples in this tutorial require you to have a user with administrator rights in TestLink.

By default, the XML-RPC comes disabled in TestLink. Let's enable it by editing config.inc.php, located in TestLink root folder.

```
$tlCfg->api->enabled = TRUE;
```

Finally, let's make sure that the attachments retrieved from the database are ordered by its ID. This way, the order of attachments will be preserved in TestLink. We could use the date that the attachment was inserted in database, however the precision of the date\_added column is in seconds, what could lead to inconsistencies in the way that attachments are displayed in TestLink.

```
$g_attachments->order_by = " ORDER BY id ASC ";
```

## 4.2 Creating a Test Project

When you log in by the first time in TestLink it is showed to you a form to create a Test Project. After creating a test project you will be able to create test plans, requirements, specify and execute your tests.

TestLink 1.9.3 (Prague) : admin [admin] [ My Settings | Logout ]

Test Project

Name \*

Prefix (used for Test case ID) \*

Project description

Enhanced features

- Enable Requirements feature
- Enable Testing Priority
- Enable Test Automation (API keys)
- Enable Inventory

Availability

- Active
- Public

Create a test project with name My first project, prefix MFP and make sure the following options are checked: Enable Requirements feature, Enable Testing Priority, Enable Test Automation (API keys), Enable Inventory, Active and Public. Click on Create button.

TestLink 1.9.3 (Prague) : admin [admin] [ My Settings | Logout ]

Test Project

Name \*

Prefix (used for Test case ID) \*

Project description

My first project.  
Created for Jenkins TestLink Plug-in Tutorial docbook.

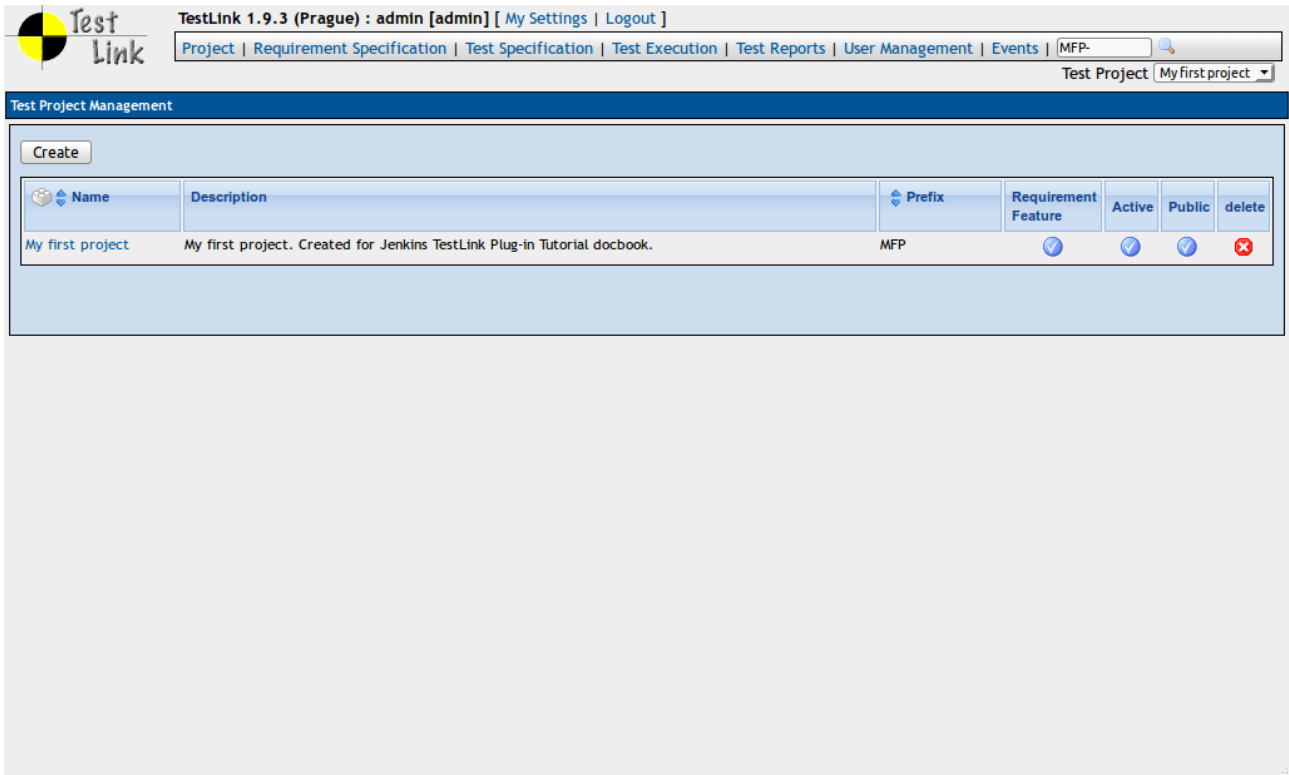
Enhanced features

- Enable Requirements feature
- Enable Testing Priority
- Enable Test Automation (API keys)
- Enable Inventory

Availability

- Active
- Public

If the following screen is not displayed, review your previous steps or consult TestLink documentation for further assistance.



### 4.3 Creating and assigning a Custom Field

Click on the Project item of the top menu to be redirected to the main screen. Now we will create the custom field used for automation. The plug-in uses this custom field's value to link a test case in TestLink with a test result from your Jenkins build.

Click on Define Custom Fields under the Test Project options box. Now create a custom field using the name Java Class, label Java Class, available for Test Case, type string, enable on Test Spec Design and display on test execution No. The plug-in retrieves the custom fields by its name and not by the value in its label.

TestLink 1.9.3 (Prague) : admin [admin] [ My Settings | Logout ]

Project | Requirement Specification | Test Specification | User Management | Events | MFP- Test Project My first project

Custom fields

Name	Java Class
Label	Java Class
Available for	Test Case
Type	string
Enable on	Test Spec Design
Display on test execution	No

Add Cancel

TestLink 1.9.3 (Prague) : admin [admin] [ My Settings | Logout ]

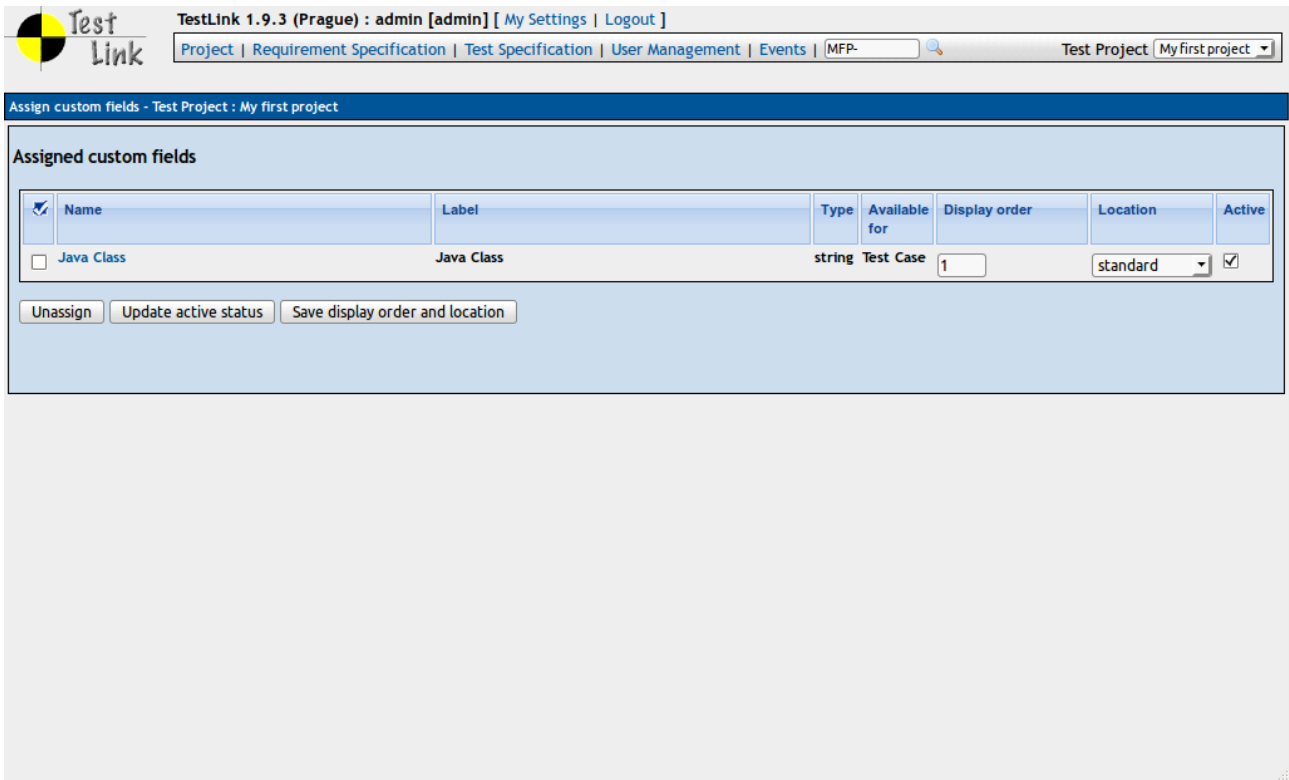
Project | Requirement Specification | Test Specification | User Management | Events | MFP- Test Project My first project

Custom fields

Name	Label	Type	Enable on test specification	Display on test execution	Enable on test execution	Enable on test plan design	Available for
Java Class	Java Class	string	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Test Case

Create Export Import

The last step now is to assign this custom field to be used in our test project. Go back to the main screen and click on Assign Custom Fields. It will take you to a screen with the list of the available custom fields. Select the Java Class custom field and click on the Assign button.



## 4.4 Specifying Test Suite and Test Cases

For those who work with tests this part may be slightly easier than the previous sections. What we are going to do now is to create a test suite and some test cases. In TestLink your test cases are kept under a test suite which is, by its turn, under a test project.

Back to the main screen, in the top menu you will see the option Test Specification. Click on this option. The test specification screen is quite simple. On the left you have the navigator, with the settings, filter and the tree of test suites and test cases. Start clicking on your test project to see the test suite operations available.



TestLink 1.9.3 (Prague) : admin [admin] [ My Settings | Logout ]

Project | Requirement Specification | Test Specification | Test Execution | Test Reports | User Management | Events | MFP- [search]

Test Project | My first project

**Navigator - Test Specification**

**Settings**

Update tree after every operation

**Filters**

Expand tree Collapse tree

My first project (0)

**Test Specification**

The *Test Specification* allows users to view and edit all of the existing *Test Suites* and *Test Cases*. Test Cases are versioned and all of the previous versions are available and can be viewed and managed here.

**Getting Started:**

1. Select your *Test Project* in the navigation tree (the root node). *Please note: You can always change the active Test Project by selecting a different one from the drop-down list in the top-right corner.*
2. Create a new Test Suite by clicking on **Create** (Test Suite Operations). Test Suites can bring structure to your test documents according to your conventions (functional/non-functional tests, product components or features, change requests, etc.). The description of a Test Suite could hold the scope of the included test cases, default configuration, links to relevant documents, limitations and other useful information. In general, all annotations that are common to the Child Test Cases. Test Suites follow the "folder" metaphor, thus users can move and copy Test Suites within the Test project. Also, they can be imported or exported (including the contained Test cases).
3. Test Suites are scalable folders. Users can move or copy Test Suites within the Test project. Test Suites can be imported or exported (include Test Cases).
4. Select your newly created Test Suite in the navigation tree and create a new Test Case by clicking on **Create** (Test Case Operations). A Test Case specifies a particular testing scenario, expected results and custom fields defined in the Test Project (refer to the user manual for more information). It is also possible to assign **keywords** for improved traceability.
5. Navigate via the tree view on the left side and edit data. Each Test case stores own history.
6. Assign your created Test Specification to a [Test Plan](#) when your Test cases are ready.

With TestLink you can organize Test Cases into Test Suites. Test Suites can be nested within other test suites, enabling you to create hierarchies of Test Suites. You can then print this information together with the Test Cases.

Create a test suite with any name that you want. This field is not important for this tutorial. Once created, your test suite will be displayed in the tree on the left. Now click on the test suite in the tree to see the available test case operations.

TestLink 1.9.3 (Prague) : admin [admin] [ My Settings | Logout ]

Project | Requirement Specification | Test Specification | Test Execution | Test Reports | User Management | Events | MFP- [search]

Test Project | My first project

**Navigator - Test Specification**

**Settings**

Update tree after every operation

**Filters**

Expand tree Collapse tree

My first project (0)

**Test Project : My first project**

**Test Suite Operations**

Create Sort alphabetically Import

**Test Project Name**

My first project

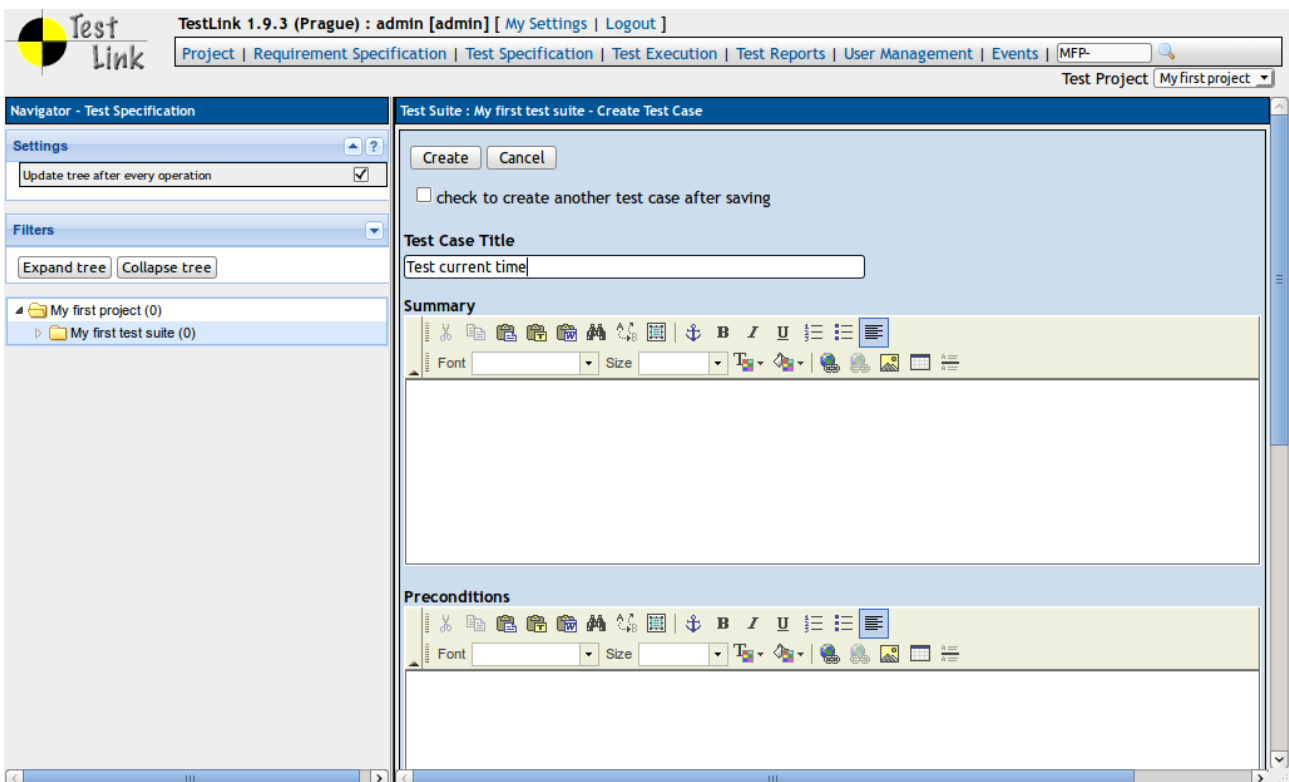
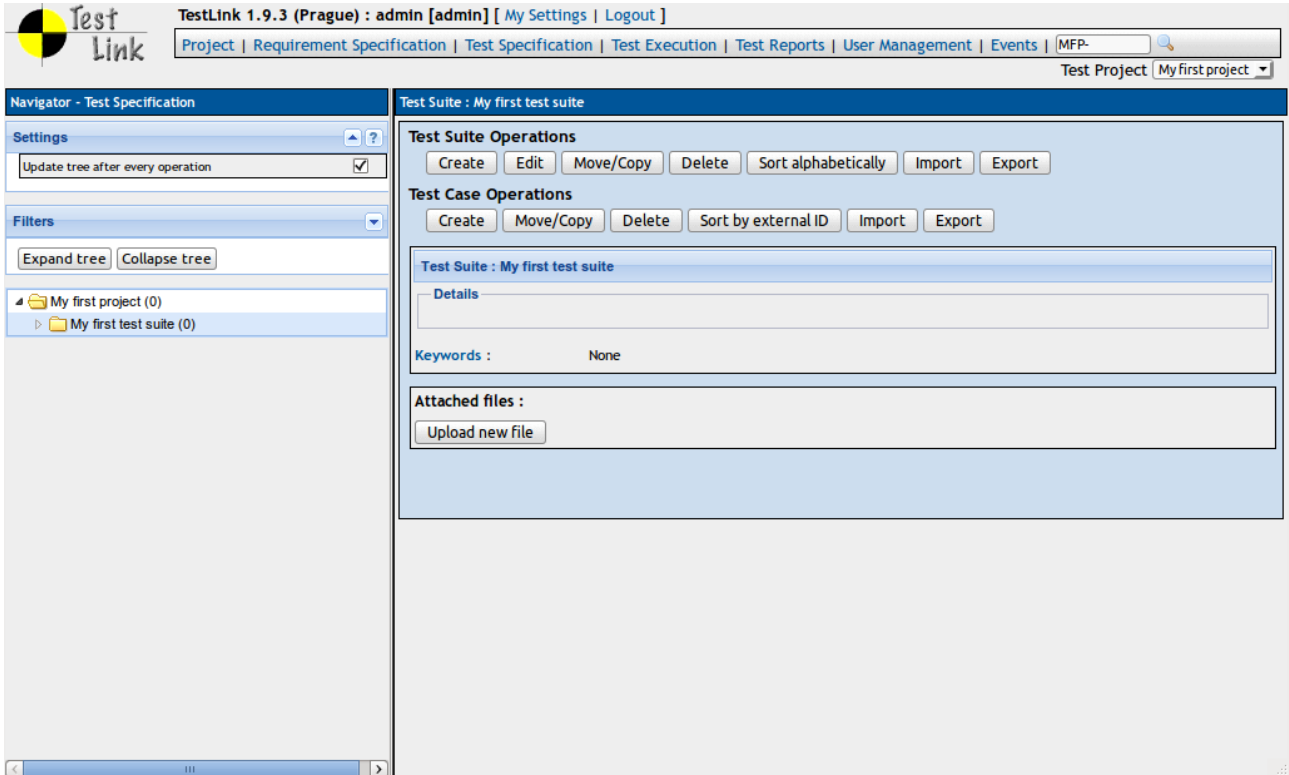
**Description**

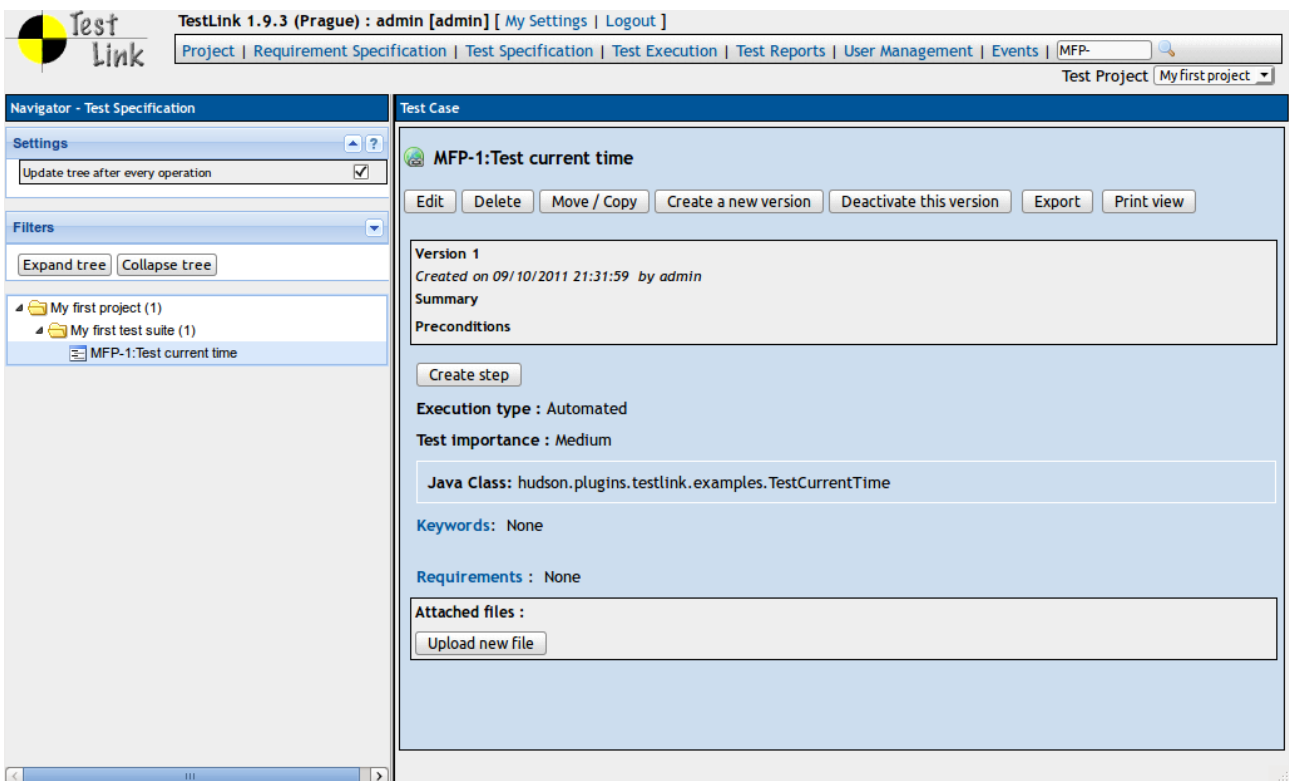
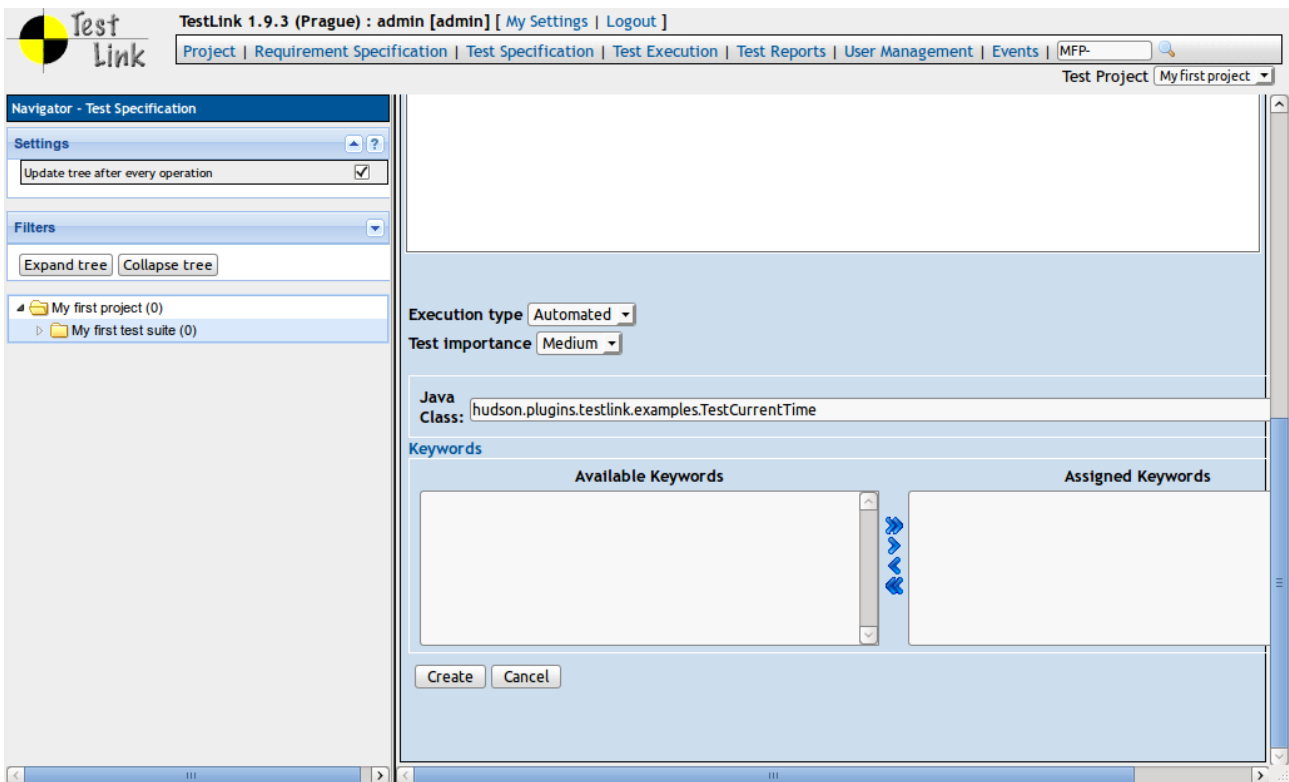
My first project.  
Created for Jenkins TestLink Plug-in Tutorial docbook.

**Attached files :**

Upload new file

Create a test case with any title or summary. The important information on this screen for the automation are the execution type and the Java class (custom field created in the previous step). In execution type select Automated and for Java class fill with the java class `hudson.plugins.testlink.examples.TestCurrentTime` (present in the example project).





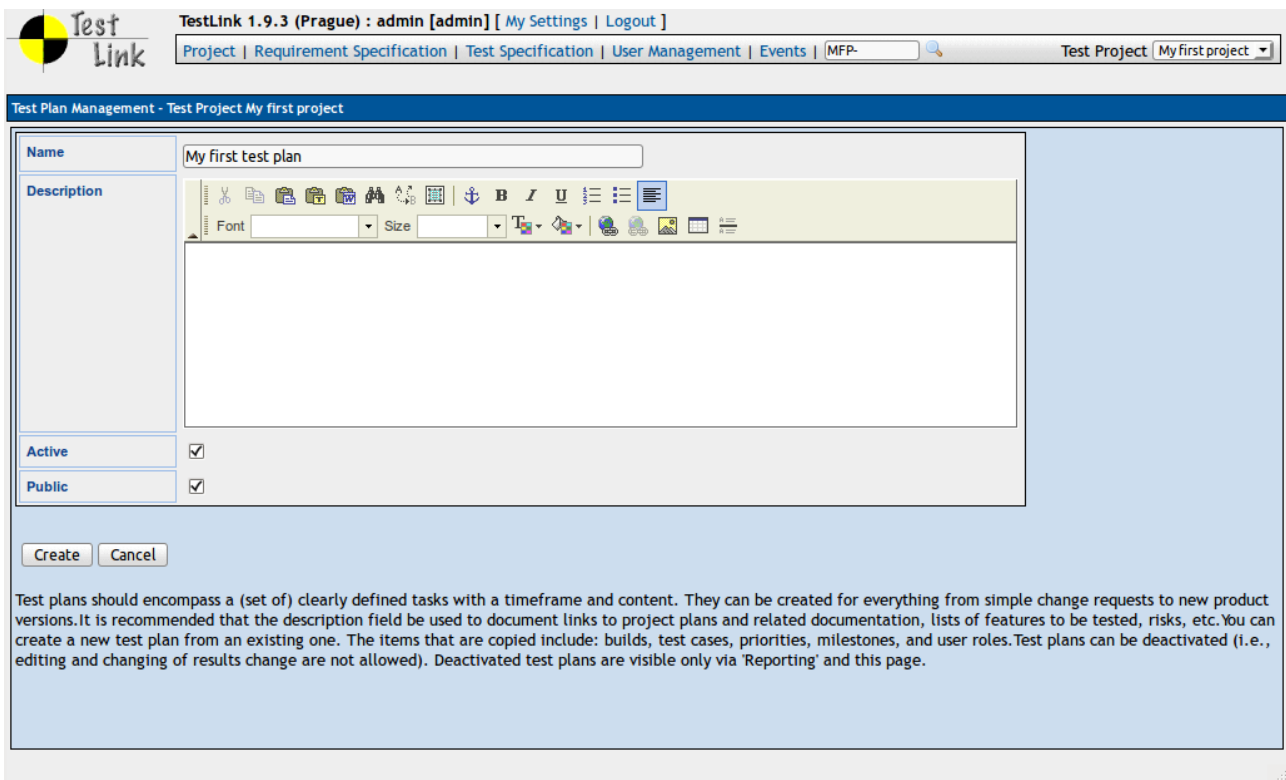
## 4.5 Create a Test Plan and add the Test Cases

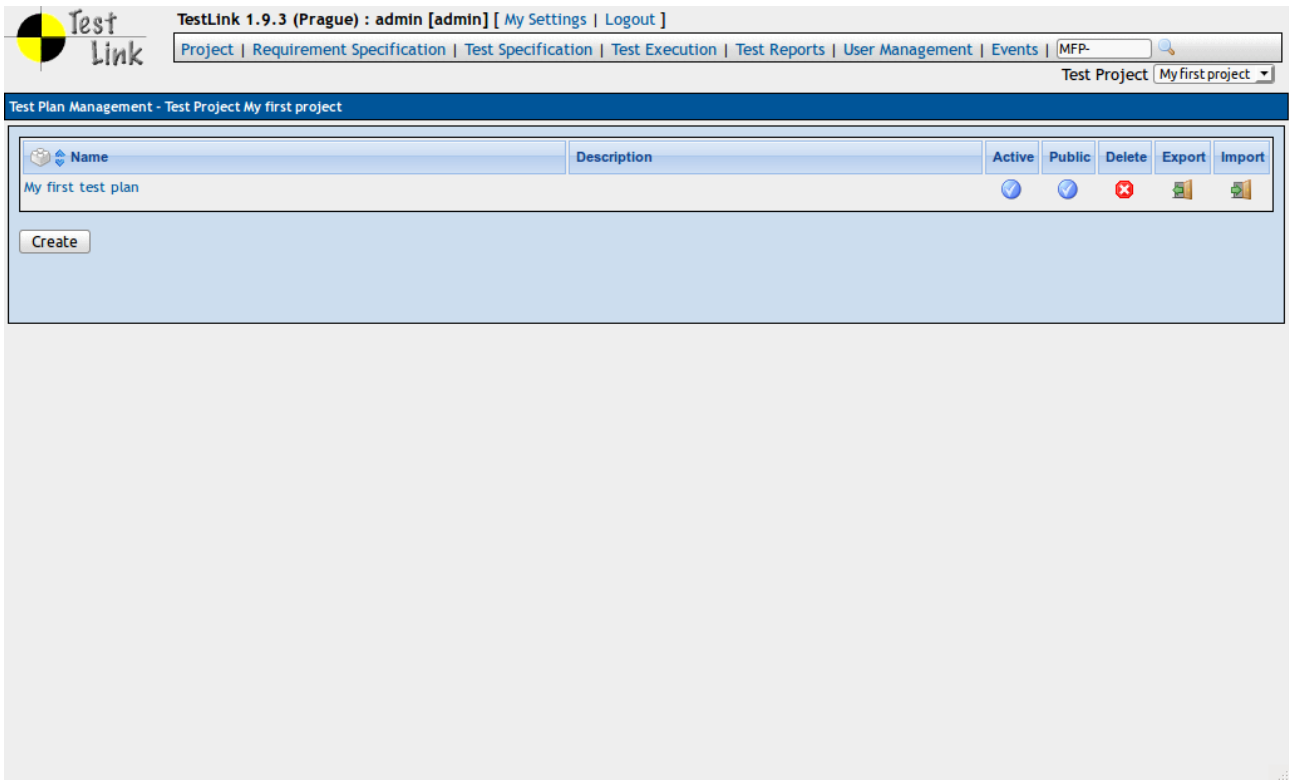
This is the last step for the TestLink configuration. However before we start this step, there is an important concept in TestLink: Builds.

In TestLink, you create a test plan outlining how you will test your application under test. Once you have a test plan you can start to add test cases to your test plan, and then execute the test plan.

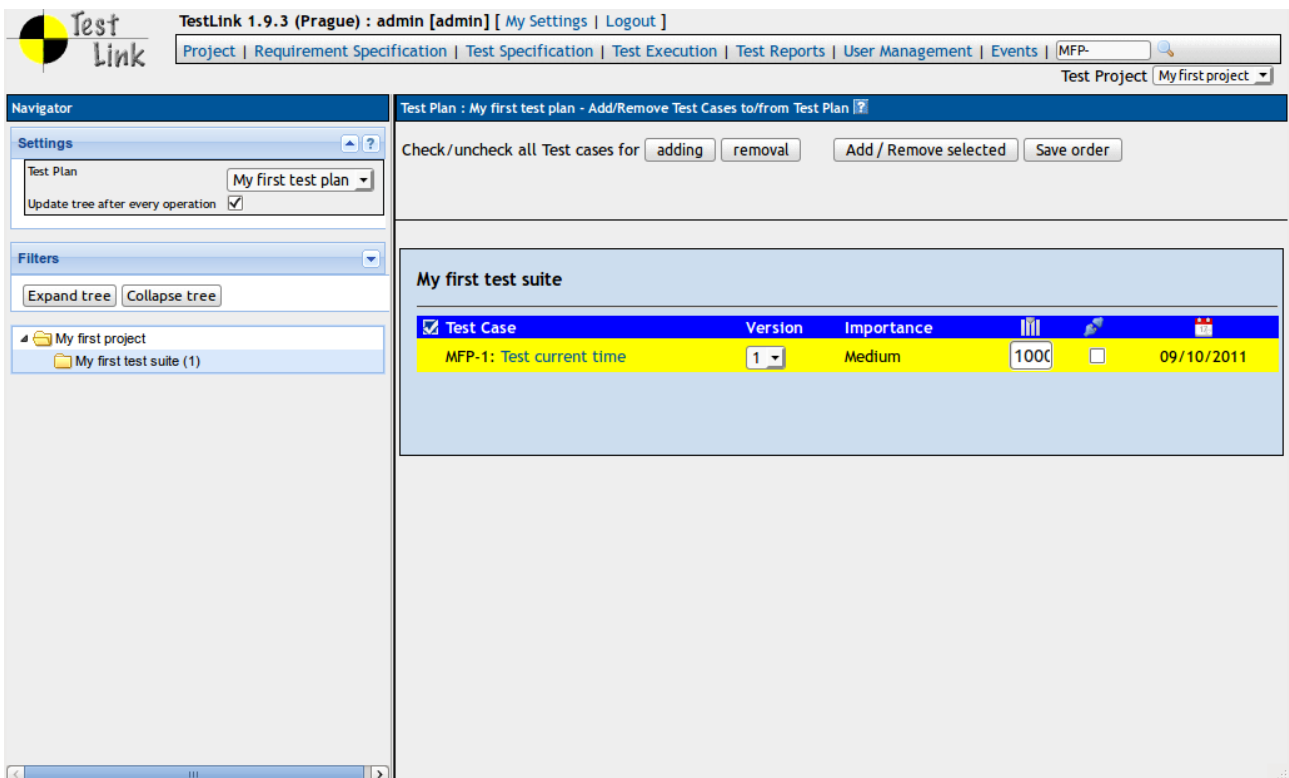
A build in TestLink can be seen as the execution of a test plan. Once the test plan is executed you are not allowed to edit the test cases of this test plan (it wouldn't be right to change the scope or exit criteria of a test case after it had already been executed).

Go back to the main screen and click on the Test Plan Management option available under the Test Plan box on the right of the screen. Create a test plan with the name My first test plan, any description and make sure that Active and Public are checked.





In the last box on the right of the screen, click on Add / Remove Test Cases and add the test case that you created to your test plan.



Create a Build in TestLink is optional, as the plug-in automatically creates a new build if there is none with the name that you provided in the Jenkins job configuration page. When you go back to your test plan, you should

see more options available in the Test Plan box and two other boxes: Test Execution and Test Plan contents, as well as other options available in the top menu.

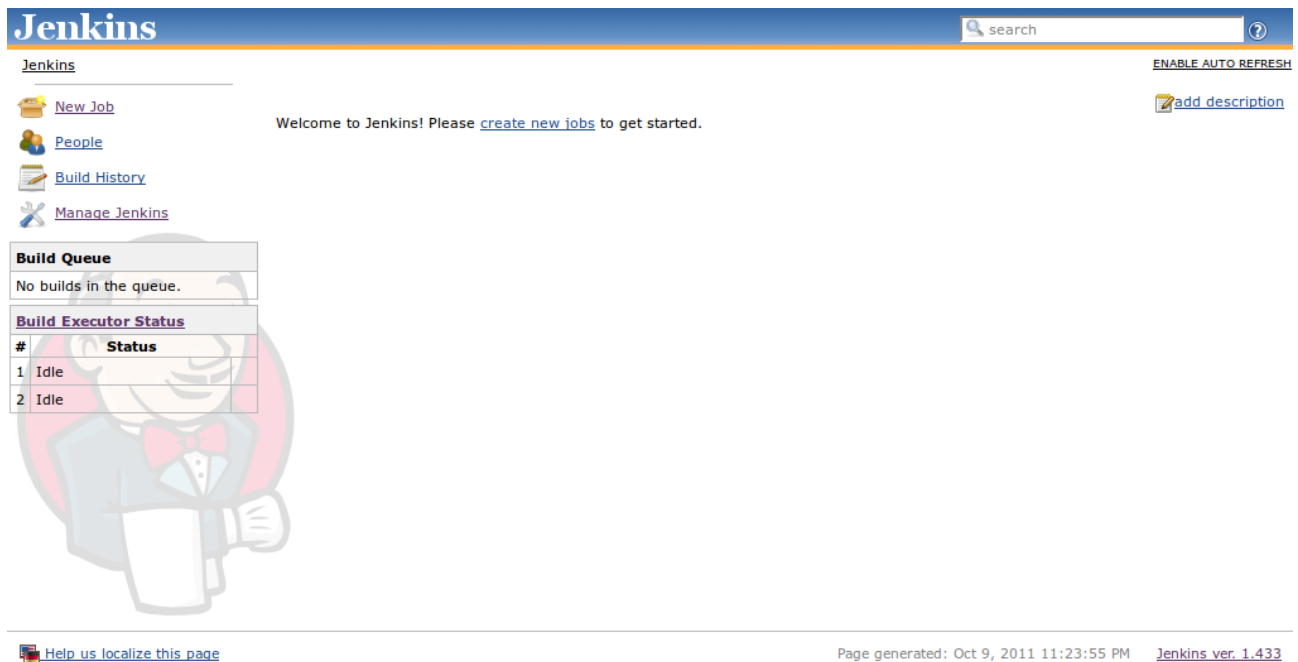
We are done with TestLink for now, the next step is to configure Jenkins.

## 5. Jenkins Configuration

### 5.1 Installing Jenkins

Download *jenkins.war* from <http://www.jenkins-ci.org>. Now open a terminal and execute **java -jar jenkins.war**. This will initiate Jenkins in port 8080 by default (if you need to change that port, use `--httpPort=9999`).

Jenkins creates a default workspace for you at `~/.jenkins` or it uses the folder specified in `JENKINS_HOME` environment variable.



The screenshot shows the Jenkins web interface. At the top, there is a blue header with the Jenkins logo and a search bar. Below the header, there is a navigation menu with links for "New Job", "People", "Build History", and "Manage Jenkins". A central message says "Welcome to Jenkins! Please [create new jobs](#) to get started." To the right, there is a link for "add description" and a toggle for "ENABLE AUTO REFRESH". Below the navigation menu, there is a "Build Queue" section showing "No builds in the queue." and a "Build Executor Status" table with two rows, both showing "Idle" status. A large, semi-transparent watermark of the Jenkins mascot is visible in the background.

#	Status
1	Idle
2	Idle

Go to <http://localhost:8080> to check if your installation is working. We will call this page as main screen from now on. The examples in this tutorial require you to have administrator rights in Jenkins.

### 5.2 Installing and configuring Jenkins TestLink Plug-in

The plug-ins in Jenkins are distributed from a central update site. Select the option *Manage Jenkins* in the left menu and look for the *Manage Plugins* option. Clicking on *Available* will bring you the list of plug-ins ready to be installed in your Jenkins installation.

Just check the check box besides the plug-in name on the list and click on *Install* to install the plug-in. Jenkins will download and install the plug-in automatically for you. Restart Jenkins to enable your plug-in.

<input type="checkbox"/>	<a href="#">Selenium</a> Hudson will generate the html report of test result. The Selenium plug-in can be <a href="#">downloaded here</a> .	
<input type="checkbox"/>	<a href="#">ShiningPanda Plugin</a> This plugin adds Python support to Jenkins with some useful builders (standard Python builder, <a href="#">virtualenv</a> builder, ...) and the ability to use a Python axis in multi-configuration projects (for testing on multiple versions of Python).	0.3
<input type="checkbox"/>	<a href="#">SICCI for Xcode Plugin</a> This plugin integrates support for Xcode projects.	0.0.8
<input type="checkbox"/>	<a href="#">STAF - STAX Plugin</a> This plugin allows Hudson to invoke a STAF command or launch a STAX job as a build step.	0.1
<input type="checkbox"/>	<a href="#">Template Project Plugin</a> This plugin lets you use builders, publishers and SCM settings from another project.	1.3
<input checked="" type="checkbox"/>	<a href="#">TestLink Plugin</a> This plug-in integrates Jenkins and <a href="#">TestLink</a> and generates reports on automated test execution. ith this plug-in you can manage your tests in TestLink, schedule and control in Jenkins, and execute using your favorite test execution tool (TestPartner, Selenium, TestNG, Perl modules, PHPUnit, among others). Nice uhn?	3.0.2
<input type="checkbox"/>	<a href="#">Unicorn Validation Plugin</a> This plugin uses W3C's Unified Validator, which helps improve the quality of Web pages by performing a variety of checks.	0.1.1
<input type="checkbox"/>	<a href="#">WAS Builder Plugin</a> This plugin allows Jenkins to invoke IBM WebSphere Application Server's *wsadmin* as a build step.	1.6
<input type="checkbox"/>	<a href="#">XShell Plugin</a> This plugin defines a new build type to execute a shell command in a cross-platform environment.	0.6
<b>Build Triggers</b>		
<input type="checkbox"/>	<a href="#">BuildResultTrigger Plugin</a> BuildResultTrigger makes it possible to monitor the build results of other jobs.	0.4
<input type="checkbox"/>	<a href="#">DOS Trigger</a> This plugin allows to trigger a build with a DOS script.	1.23
<input type="checkbox"/>	<a href="#">Downstream-Ext Plugin</a> This plugin supports extended configuration for triggering downstream builds: * trigger build only if downstream job has SCM changes * trigger build if upstream build result is better/equal/worse than any given result (SUCCESS, UNSTABLE, FAILURE, ABORTED) * for Matrix (alias multi-configuration) jobs you can	1.6

Jenkins
search

Jenkins » Update center
ENABLE AUTO REFRESH

- [Back to Dashboard](#)
- [Manage Jenkins](#)
- [Manage Plugins](#)


## Installing Plugins/Upgrades

Preparation

- Checking internet connectivity
- Checking update center connectivity
- Success

TestLink Plugin  Installing


➔  **Restart Jenkins when installation is complete and no jobs are running**



[Help us localize this page](#)
Page generated: Oct 9, 2011 11:26:33 PM [Jenkins ver. 1.433](#)

After restarting Jenkins, go to *Manage Jenkins* again, this time click on *Configure System* option and look for the TestLink section. Fill the TestLink configuration form with a name for your TestLink installation, the URL of the XML-RPC API and your *devKey*.





Subversion Workspace Version

Exclusion revprop name

Validate repository URLs up to the first variable name

Update default Subversion credentials cache after successful authentication

**TestLink**

TestLink Installation

Name	<input type="text" value="testlink-1.9.3"/>
URL	<input type="text" value="http://localhost/testlink-1.9.3/lib/api/xmlrpc.php"/>
Developer Key	<input type="text" value="42f6648402d38dc655c05622b0c89add"/>

List of TestLink installations in this system

**Shell**

Shell executable

**Jenkins URL**

Jenkins URL

**E-mail Notification**

SMTP server

Default user e-mail suffix

System Admin E-mail Address

By default the TestLink XML-RPC API URL is `http://<host>:<port>/lib/api/xmlrpc.php`. The `devKey` can be obtained by entering TestLink, clicking on *My Settings* (top menu) and generating a new `devKey`, if there is none yet. If you cannot see the API interface section in *My Settings* page, then it is very likely that you didn't enable it yet. Go back to Chapter 4, *TestLink Configuration* to review your work.

## 5.3 Creating a job in Jenkins

In order to create a new job all that you need to do is click on *New Job* and give it a name. Choose the option to create a *Free-style project*.

Our job will use git to retrieve the test automation project. This is the project mentioned in Chapter 3, *Jenkins and TestLink Integration*, and can be found at <https://github.com/kinow/jenkins-testlink-plugin-tutorial> [<https://github.com/kinow/jenkins-testlink-plugin-tutorial>]. In a real world project you probably will use a SCM (SVN, Git, CVS, etc) to store your test automation project.

Execute concurrent builds if necessary

---

**Advanced Project Options**

[Advanced...](#)

---

**Source Code Management**

CVS  
 Git

Repositories

URL of repository  [?](#)

[Advanced...](#)  
[Delete Repository](#)

[Add](#)

Branches to build

Branch Specifier (blank for default):  [?](#)

[Delete Branch](#)

[Add](#)

[Advanced...](#)

Repository browser  [?](#)

None  
 Subversion

---

**Build Triggers**

Build after other projects are built [?](#)  
 Build periodically [?](#)  
 Poll SCM [?](#)

.....

Click on *Add build step* button to expand its options and then click on *Invoke TestLink*. It will show a new form with options to integrate your Jenkins job with TestLink. Fill this form with the following information created in TestLink throughout Chapter 4, *TestLink Configuration*.

Configure your job with the following settings.

Table 5.1. Jenkins TestLink Plug-in settings explanation

TestLink Version	Select the version that you created in the global configuration
Test Project Name	My first project
Test Plan Name	My first test plan
Build Name	examples-\$BUILD_NUMBER
Custom Fields	Java Class
Key Custom Field	Java Class
Single Build Steps	Add an <i>Invoke top-level Maven targets</i> build step, and as goals use <b>test -DsuiteXmlFiles=suite.xml</b>
Before iterating all test cases Build Steps	Leave it the way it is
Iterative Test Build Steps	Leave it the way it is
After iterating all test cases Build Steps	Leave it the way it is
JUnit Report Files Pattern	Leave it blank
TestNG Report Files Pattern	target/**/testng-results.xml
TAP Report Files Pattern	Leave it blank

TestLink Version	<input type="text" value="testlink-1.9.3"/>	
Test Project Name	<input type="text" value="My first project"/>	
Test Plan Name	<input type="text" value="My first test plan"/>	
Build Name	<input type="text" value="examples-\$BUILD_NUMBER"/>	
Custom Fields	<input type="text" value="Java Class"/>	
Key Custom Field	<input type="text" value="Java Class"/>	
Single Build Steps	<div style="border: 1px solid #ccc; padding: 5px;"> <p><b>Invoke top-level Maven targets</b> </p> <p>Maven Version <input type="text" value="(Default)"/></p> <p>Goals <input type="text" value="test -DsuiteXmlFiles=suite.xml"/></p> <p style="text-align: right;"><input type="button" value="Advanced..."/> <input type="button" value="Delete"/></p> </div>	
	<input type="button" value="Add action"/>	
Before iterating all test cases Build Steps	<input type="button" value="Add action"/>	
Iterative Test Build Steps	<input type="button" value="Add action"/>	
After iterating all test cases Build Steps	<input type="button" value="Add action"/>	
JUnit Report Files Pattern	<input type="text"/>	
TestNG Report Files Pattern	<input type="text" value="target/**/testng-results.xml"/>	
TAP Report Files Pattern	<input type="text"/>	

Save your job.

## 5.4 Explaining the Job configuration parameters

TestLink Version	<input type="text" value="testlink-1.9.3"/>	
Test Project Name	<input type="text" value="My first project"/>	
Test Plan Name	<input type="text" value="My first test plan"/>	
Build Name	<input type="text" value="examples-\$BUILD_NUMBER"/>	
Custom Fields	<input type="text" value="Java Class"/>	
Key Custom Field	<input type="text" value="Java Class"/>	
Single Build Steps	<div style="border: 1px solid #ccc; padding: 5px;"> <p><b>Invoke top-level Maven targets</b> </p> <p>Maven Version <input type="text" value="(Default)"/></p> <p>Goals <input type="text" value="test -DsuiteXmlFiles=suite.xml"/></p> <p style="text-align: right;"><input type="button" value="Advanced..."/> <input type="button" value="Delete"/></p> </div>	
	<input type="button" value="Add action"/>	
Before iterating all test cases Build Steps	<input type="button" value="Add action"/>	
Iterative Test Build Steps	<input type="button" value="Add action"/>	
After iterating all test cases Build Steps	<input type="button" value="Add action"/>	
JUnit Report Files Pattern	<input type="text"/>	
TestNG Report Files Pattern	<input type="text" value="target/**/testng-results.xml"/>	
TAP Report Files Pattern	<input type="text"/>	

## TestLink Version

This is the name of your TestLink installation in Jenkins global configuration.

## Test Project Name

This is the name of your *Test Project* in TestLink. You can use environment variables in this field.

## Test Plan name

This is the name of your *Test Plan* in TestLink. You can use environment variables in this field.

## Build Name

This is the name of your *Build* in TestLink. You can use environment variables in this field.

## Custom fields

This is a comma separated list of *Custom Fields* in TestLink. Only the custom fields present here are retrieved from TestLink.

## Key Custom Field

This is the custom field used by the plug-in to link a test case in TestLink with your test results. This custom field must exist in the list of custom fields.

## Single Build Steps

These Build Steps are executed only once in the job execution.

## Before iterating all test cases Build Steps

These Build Steps are executed before iterating the automated test cases retrieved from TestLink.

## Iterative Test Build Steps

These Build Steps are executed iteratively for each test case retrieved from TestLink. For these Build Steps, a set of environment variables are available. We will discuss more about them soon.

## After iterating all test cases Build Steps

These Build Steps are executed after iterating the automated test cases retrieved from TestLink.

## JUnit, TestNG and TAP report files patterns.

The plug-in uses these patterns to find test results of your tests execution (single test command or iterative). You can use multiple patterns at once, but it may lead to inconsistencies in your test results in TestLink.

## Transactional execution?

If this option is marked and any test fails, all remaining tests will be marked as *Blocked* in TestLink.

## Failed tests mark build as failure

If this option is marked and any failed test found by the plug-in, your Build in Jenkins will be marked as failure.

## Environment variables

The plug-in retrieves all the information from TestLink for your Test Project, Test Plan, Build and automated Test Cases. You can then use any of this information to execute your tests. Jenkins itself provides the *Environment Variables*, plus *Build Environment Variables* (such as *BUILD\_ID*, which holds the date time of your job).

The plug-in injects the information retrieved from TestLink as extra environment variables. This way you can use the value of the *Java class* custom field value that you created in Chapter 4, *TestLink Configuration* in any of your iterative Build Steps. Below you can find an example of how to execute a single test with Maven and one of these environment variables.

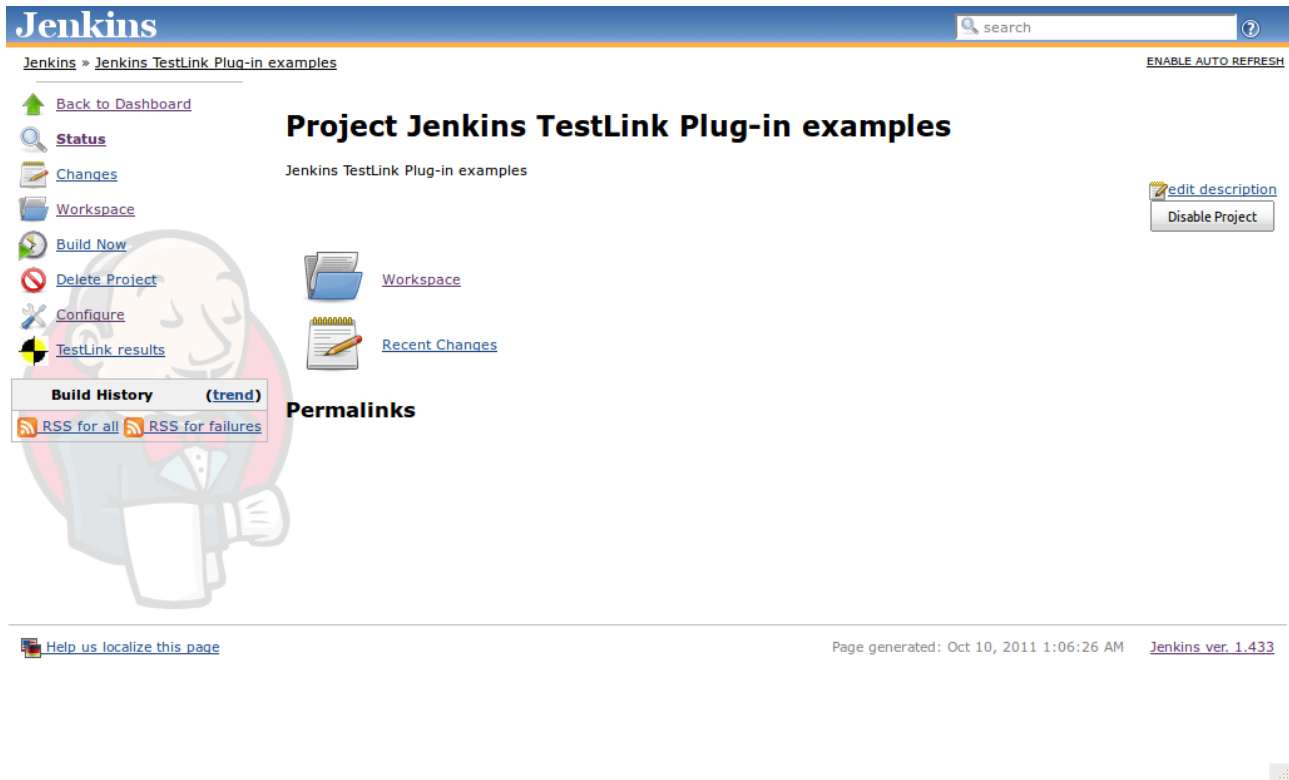
```
mvn test -Dtest=$TESTLINK_TESTCASE_JAVA_CLASS
```

As you can see, our test command uses the *Java class* custom field value to specify the name of the test to Maven (Maven Surefire Plug-in, actually). Below you will find a list with the information that the plug-in makes available for your job configuration. As custom fields names may vary, the strategy used is capitalize the custom field name, replace spaces with `_` and append it to `TESTLINK_TESTCASE_`, which represents information of a Test Case in TestLink.

- TESTLINK\_TESTCASE\_ID
- TESTLINK\_TESTCASE\_NAME
- TESTLINK\_TESTCASE\_TESTPROJECTID
- TESTLINK\_TESTCASE\_AUTHOR
- TESTLINK\_TESTCASE\_SUMMARY
- TESTLINK\_BUILD\_NAME
- TESTLINK\_TESTPLAN\_NAME
- TESTLINK\_TESTCASE\_\$CUSTOM\_FIELD\_NAME

## 6. Executing Automated Test Cases

Now that everything is configured it is time to execute our automated test cases. Let's go back to the main screen in Jenkins. Click on your job and trigger your job execution by clicking on *Build Now* .



The screenshot shows the Jenkins web interface for a project named "Project Jenkins TestLink Plug-in examples". The page has a blue header with the "Jenkins" logo and a search bar. Below the header, there is a navigation menu on the left with links for "Back to Dashboard", "Status", "Changes", "Workspace", "Build Now", "Delete Project", "Configure", and "TestLink results". The main content area displays the project name and a "Jenkins TestLink Plug-in examples" subtitle. There are several action buttons: "edit description" and "Disable Project" on the right, and "Workspace" and "Recent Changes" in the center. A "Build History" section is visible at the bottom left, with a "(trend)" link and two RSS feeds: "RSS for all" and "RSS for failures". The footer contains a link to "Help us localize this page", the page generation date "Page generated: Oct 10, 2011 1:06:26 AM", and the version "Jenkins ver. 1.433". A large, semi-transparent watermark of the Jenkins mascot is overlaid on the page.

Once your job execution is completed, refresh your current screen either by pressing F5 or by clicking over the build name. The following screen must appear, showing a graph with the test results.

**Jenkins** search ?

Jenkins » Jenkins TestLink Plug-in examples ENABLE AUTO REFRESH

[Back to Dashboard](#)  
[Status](#)  
[Changes](#)  
[Workspace](#)  
[Build Now](#)  
[Delete Project](#)  
[Configure](#)  
[TestLink results](#)

## Project Jenkins TestLink Plug-in examples

Jenkins TestLink Plug-in examples

[Workspace](#)  
[Recent Changes](#)

[edit description](#)  
[Disable Project](#)

**Build History** [\(trend\)](#)

#2	Oct 10, 2011 1:14:14 AM
#1	Oct 10, 2011 1:13:59 AM

[RSS for all](#) [RSS for failures](#)

**Permalinks**

- Last build (#2), 7.3 sec ago
- Last stable build (#2), 7.3 sec ago
- Last successful build (#2), 7.3 sec ago

**TestLink Tests Count**

TestLink Tests Count	1
Failed	0
Blocked	0
Not Run	0
Passed	1

[Help us localize this page](#) Page generated: Oct 10, 2011 1:14:22 AM [Jenkins ver. 1.433](#)

You can find more information on your tests by clicking on *Test Results* or on your build under *Build History*.

**Jenkins** search ?

Jenkins » Jenkins TestLink Plug-in examples » #2

[Back to Project](#)  
[Status](#)  
[Changes](#)  
[Console Output](#)  
[Edit Build Information](#)  
[Git Build Data](#)  
[TestLink](#)  
[Previous Build](#)

## TestLink Results

Build ID: **2**  
 Build Name: **examples-2**  
 Passed: **1**  
 Failed: **0**  
 Blocked: **0**  
 Not Run: **0**  
 Total: **1**

[Help us localize this page](#) Page generated: Oct 10, 2011 1:16:49 AM [Jenkins ver. 1.433](#)



**Jenkins** search ?

Jenkins » Jenkins TestLink Plug-in examples » #2 ENABLE AUTO REFRESH

- [Back to Project](#)
- [Status](#)
- [Changes](#)
- [Console Output](#)
- [Edit Build Information](#)
- [Git Build Data](#)
- [TestLink](#)
- [Previous Build](#)

## Build #2 (Oct 10, 2011 1:14:14 AM)

No changes.

Started by anonymous user

**Revision:** 587740d5bc5c017bccd2357864507d01bcfe7e85

- origin/HEAD
- origin/master

**TestLink build ID:** 2

**TestLink build name:** examples-2

Total of 1 tests. Where 1 passed, 0 failed, 0 were blocked and 0 were not executed.

List of test cases and execution result status

Test case ID	Version	Name	Test project ID	Execution status
3	1	Test current time	1	Passed

[Keep this build forever](#)

[Delete this build](#)

[add description](#) Started 3 min 19 sec ago  
Took [5.1 sec](#)

Help us localize this page Page generated: Oct 10, 2011 1:17:33 AM Jenkins ver. 1.433

Everything seems to be right in Jenkins, let's go back to TestLink and click on *Test execution* option in the top menu. It will bring a screen showing the result of your test execution.

**TestLink 1.9.3 (Prague)** : admin [admin] [ My Settings | Logout ]

Project | Requirement Specification | Test Specification | **Test Execution** | Test Reports | User Management | Events | MFP- Test Project My first project

**Test Plan : My first test plan [Build : examples-9]**

**Settings**

Test Plan: My first test plan

Build to execute: examples-9

Update tree after every operation:

[Export Test Plan](#)

---

**Filters**

[Expand tree](#) [Collapse tree](#)

- My first project / My first test plan (1)(0, 1, 0, 0)
- My first test suite (1)(0, 1, 0, 0)
- MFP-1:Test current time

**Test Case Execution**

**Purpose:**  
Allows user to execute Test cases. User can assign Test result to Test Case for a Build. See help for more information about filters and settings (click on the question mark icon).

**Get started:**

1. User must have defined a Build for the Test Plan.
2. Select a Build from the drop down box
3. If you want to see only a few testcases instead of the whole tree, you can choose which filters to apply. Click the "Apply"-Button after you have changed the filters.
4. Click on a test case in the tree menu.
5. Fill out the test case result and any applicable notes or bugs.
6. Save results.

*Note: TestLink must be configured to collaborate with your Bug tracker if you would like to create/trace a problem report directly from the GUI.*

You can see more information about the execution of your tests by clicking over it on the left tree. Notice that by default the plug-in uploads the test result file. In this case, as we are using TestNG, it uploaded *testng-results.xml*. It is useful as you can see what happened in your tests, e.g.: an exception stack trace.

The screenshot shows the TestLink interface for 'My first test plan'. The left sidebar contains 'Settings' and 'Filters'. The main area displays 'Last execution (any build) - Build : examples-2' with a status of 'Passed' on 10/10/2011. Below this is a table of test executions:

Date	Build	Tested by	Status	Test Case Version	attachments	Run mode
10/10/2011 01:09:35	examples-9	admin	Passed	1		

The 'Notes' section provides details for the test case:

```

name: Command line suite
duration in ms: 17
started at: 2011-10-10T01:09:35Z
finished at: 2011-10-10T01:09:35Z
number of tests: 1
-----
class name: hudson.plugins.testlink.examples.TestCurrentTime
number of methods: 1
name: testCurrentTime
config?: null
signature: testCurrentTime()[pri:0,
    
```

At the bottom, there is a link to the test result file: [testng-results.xml - testng-results.xml \(810 bytes, text/xml\) 10/10/2011](#).

Explore the reports available in TestLink to see the results of your tests by different perspectives.

The screenshot shows the 'Reports and Metrics' section of TestLink. The left sidebar lists various report options, and the main area displays 'General Test Plan Metrics' for 'My first project' and 'My first test plan'.

**Overall Build Status**

Build	Assigned	Not Run	[%]	Passed	[%]	Failed	[%]	Blocked	[%]	Completed	[%]
examples-1	0	0	0	0	0	0	0	0	0	N/A	
examples-2	0	0	0	0	0	0	0	0	0	N/A	
examples-3	0	0	0	0	0	0	0	0	0	N/A	
examples-4	0	0	0	0	0	0	0	0	0	N/A	
examples-5	0	0	0	0	0	0	0	0	0	N/A	
examples-6	0	0	0	0	0	0	0	0	0	N/A	
examples-7	0	0	0	0	0	0	0	0	0	N/A	
examples-8	0	0	0	0	0	0	0	0	0	N/A	
examples-9	0	0	0	0	0	0	0	0	0	N/A	

*Overall Build Status data is computed using only test cases that have tester assignment on build*

**Results by top level Test Suites**

Test Suite	Total	Not Run	[%]	Passed	[%]	Failed	[%]	Blocked	[%]	Completed	[%]
My first test suite	1	0	0.00	1	100.00	0	0.00	0	0.00	100.00	

*This report shows results for each top level test suite. Results for subordinated test suites are count in for the corresponding top level test suite.*

## 7. Strategies to find Test Results

After your automated tests are executed the plug-in has to look for *Test Results*. This way it will know whether an automated test was executed correctly or not and then it can update the test execution status in TestLink.

Jenkins TestLink Plug-in uses three strategies to find Test Results: *TAP*, *TestNG* and *JUnit*. All these strategies use a key custom field to associate a test result in Jenkins with an automated test case in TestLink.

### 7.1 JUnit

TestLink plug-in scans the JUnit test results twice. The first time it matches the key custom field value against the JUnit suite name, and in its second turn it matches the key custom field value against the JUnit *class name* or *test name*.

Every time the plug-in finds a pair that matches (be it in a JUnit suite name or a JUnit class name or test name) it looks for the test result status. In case of a test suite, if it has any errors or failures, then the status is set to *Failed*, otherwise the status is set to *Passed*.

The reason why TestLink plug-in needs to use either JUnit XML element `<testcase>`'s attribute *classname* or the attribute *testname* is that the JUnit format is used by many different tools, without having a de facto DTD or XML Schema document. First the plug-in tries to use the JUnit XML element `<testcase>`'s attribute *classname*. If it is empty, JUnit XML element `<testcase>`'s attribute *testname*.

```
<?xml version="1.0" encoding="UTF-8" ?>
<testsuite failures="1" time="0.078" errors="0" skipped="0" tests="1" name="A Suite">
  <testcase time="0" classname="tcA" name="testVoid">
    <failure type="junit.framework.AssertionFailedError" message="null">junit....</failure>
  </testcase>
  <testcase time="0" classname="tcB" name="testVoid" />
  <testcase time="0" classname="tcC" name="testVoid" />
  <testcase time="0" name="nameA" />
  <testcase time="0" name="nameB" />
</testsuite>
```

In the example above, we have a test suite which failed, and five test cases, where only one failed. Let's suppose you created this test suite to execute an automated test case from TestLink. This would mean that you have a test case in TestLink with a custom field which value is *A Suite* (we assume you are using this custom field as key in the job configuration). The plug-in, after scanning the test results, it will update your test execution status to *Failed* in TestLink, because the test suite has failed.

Now, suppose you created many test cases for your automated test cases from TestLink, and *tcB* is the value of your key custom field in TestLink. Then the plug-in will update the test case execution status to *Passed*, as this test case hasn't failed.

### 7.2 TestNG

The strategy used for TestNG is quite similar to the JUnit one. The plug-in scans the test results twice, also matching the key custom field value against the TestNG suite name and then against each TestNG class name.

Every time the plug-in finds a pair that matches (be it in a TestNG suite name or a TestNG class name or test name) it looks for the test result status. In case of a test suite, if its has any failures, then the status is set to *Failed*, otherwise the status is set to *Passed*.

```
<testng-results>
  <reporter-output>
</reporter-output>
  <suite name="Command line suite" duration-ms="0" started-at="2010-11-17T13:31:41Z" finished-at="2010-11-17T13:31:41Z">
    <groups>
</groups>
    <test name="Command line test" duration-ms="0" started-at="2010-11-17T13:31:41Z" finished-at="2010-11-17T13:31:41Z">
      <class name="br.eti.kinoshita.Test">
        <test-method status="FAIL" signature="testVoid()" name="testVoid" duration-ms="0" started-at="2010-11-17T13:31:41Z" finished-at="2010-11-17T13:31:41Z">
</test-method>
      </class>
    </test>
  </suite>
</testng-results>
```

In the above example, we have a test suite named *Command line suite*, and a class which name is *br.eti.kinoshita.Test*. You could have a key custom field in TestLink which value was the test suite name or the class name. In this case, both situations would lead to the test execution status in TestLink being updated to *Failed*. The TestNG test suite status is defined by its tests status, while the class status is defined by its status attribute.

## 7.3 TAP - Test Anything Protocol

Lastly, we have TAP. Here the plug-in simply utilizes the TAP file name as option to match it against the key custom field value. If the TAP file contains any *Bail out!*, *not ok* or *TODO directive*, then the test execution status is updated to *Failed* in TestLink. If the TAP file contains any *SKIP directive* then the plug-in updates the test execution status to *Blocked*. Otherwise, the plug-in considers that the test must be marked as *Passed*.

```
1..3
ok 1 testOk
ok 2
not ok 3
```

The example above shows a *TAP Stream* (that was stored in a file), which has three test results, and one of the failed. Hence the test execution status will be updated as *Failed* in TestLink.

Using TAP, you can define the Platform (defined in TestLink) that you used for your tests or extra attachments to be uploaded to TestLink. These topics are covered in Chapter 9, *Appendix*.

## 8. Advantages of using Jenkins TestLink Plug-in

The plug-in is not intended to be a test automation solution, though it helps you to automate your tests. It is not intended to manage your automated tests either, as this can be done in TestLink. The purpose of the plug-in is integrate Jenkins, a continuous integration server, and TestLink, a test management tool.

There are three clear advantages in using this approach. First, if you are already using Jenkins as continuous integration server and TestLink as test management tool, then you won't have another tool to worry about. Simply install the plug-in and then DevOps will keep working in Jenkins, testers will keep working in TestLink and your boss will be more than happy to know that he won't need to buy another tool and training for an automated tests management tool.

Secondly, it is language independent. You can run tests in PHP, Perl, Python, Java, Lua and even in C or C++. The only limitation here is that you have to output your test results either in JUnit, TestNG or TAP.

And lastly, it is free. The team of the plug-in are contributors of TestLink, maintainers of the TestLink Java API, and keep trying to fix the issues in Jenkins' JIRA [<http://issues.jenkins-ci.org>] as fast as they can. So if you use the plug-in, send us your feedback, write in the plug-in Wiki about your case or buy us a beer.

## 9. Appendix

### 9.1 Adding attachments to your test results

Each test result file is automatically attached to its automated test case in TestLink. That is the default behavior in the plug-in. You can extend it, only if you are using TAP. You will have to create a *YAMLish* following the example below (it is important that you encode the file content in Base64).

```
1..2
ok 1
not ok 2 - br.eti.kinoshita.selenium.TestListVeterinarians#testGoogle
---
extensions:
  Files:
    /tmp/screenshot3562328890173159732.png:
      File-Location: /tmp/screenshot3562328890173159732.png
      File-Title: screenshot3562328890173159732.png
      File-Description: Main page
      File-Size: 114542
      File-Name: screenshot3562328890173159732.png
      File-Content: "iVBORw0KGgoAAAANSUhEUgAAA+IAAAJqCAYAAACvjvpKAAAgAELEQVR4nOy9d1RU5/r3ff543+dZ\r\
        \n613v8zy/95RfjjkmFmwgVXrvvf fee1PBjtKbCoJ0qaIodqMYe8EWwRolatQkRmOixkTFAkYFvu8f\r\
        ...
        ...
      File-Type: image/png
```

An issue that wasn't solved yet is that if you are running your job in a distributed environment, then the plug-in won't be able to find the attachments for your test result.

### 9.2 Using Platforms

TestLink has a very nice feature, Platforms. Useful specially if you have tests that run in some specific environment and you use some logic (like pairwise) to choose what is going to be tested.

As well as attachments, the only way to use Platforms while integrating Jenkins and TestLink is using TAP. You have to create a *YAMLish* entry like the following.

```
1..3
ok 1
ok 2
---
extensions:
  TestLink:
    Platform: EC1
    ...
ok 3
```

As you can see above, the TAP Stream has the platform EC1. The plug-in scans a TAP Stream looking for this entry structure in each test result and in the test plan.

## 9.3 Plug-in behavior in distributed environments

The calls to TestLink are executed in the master, as well as the graph generating and some Jelly back end code. The rest, what includes executing Build Steps and looking for and parsing test results is executed in the slave, if present.

## 9.4 How to use the plug-in with SSL or basic authentication?

In the global configuration of the plug-in, there is a *Advanced* button. Click on it and an extra option will be shown, *TestLink Java API Properties*.

Click on the help icon on the right to see which options you can enable. They are used by `testlink-java-api` [<http://testlinkjavaapi.sf.net>] to set properties in the connection.

Among the properties, you may have to set `xmlrpc.basicUsername` and `xmlrpc.basicPassword` with the credentials used for basic authentication.

For SSL, there are guides available in the Internet showing how to add a certificate to the Java key store and proceed with a SSL connection.

# Bibliography

[JenkinsTestLink] *Jenkins TestLink Plug-in* [<http://wiki.jenkins-ci.org/display/JENKINS/TestLink+Plugin>].  
Copyright © 2011 The Jenkins TestLink Plug-in team.

[JenkinsCI] *Jenkins-CI* [<http://www.jenkins-ci.org/>]. Copyright © 2011 Jenkins-CI.

[TestLink] *TestLink* [<http://www.teamst.org/>]. Copyright © 2011 TestLink Community.

[TestNG] *TestNG* [<http://www.testng.org/>]. Copyright © 2011 TestNG.

[JUnit] *JUnit* [<http://www.junit.org/>]. Copyright © 2011 JUnit.

[TAP] *Test Anything Protocol Website* [<http://testanything.org/>]. Copyright © 2011 Test Anything Protocol Community.

[TAPWikipedia] *Test Anything Protocol article in Wikipedia* [[http://en.wikipedia.org/wiki/Test\\_Anything\\_Protocol](http://en.wikipedia.org/wiki/Test_Anything_Protocol)]. Copyright © 2011 Wikipedia.

[YAMLOrg] *YAML.org* [<http://www.yaml.org/>]. Copyright © 2011 YAML.org.