

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

French Translation: Olivier Renault

Review and authorship on parts of this tutorial. German translation.: 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	19
5.1. Installing Jenkins	19
5.2. Installing and configuring Jenkins TestLink Plug-in	19
5.3. Creating a job in Jenkins	21
TestLink Configuration	22
Test Execution	23
Result Seeking Strategy	24
5.4. Explaining the Job configuration parameters	25
TestLink Version	25
Test Project Name	25
Test Plan name	25
Build Name	25
Single Build Steps	26
Iterative Test Build Steps	26
Test Result Seeking Strategies	26
Include pattern	26
Key Custom Field	26
Attach TestNG XML	26
Mark skipped test as Blocked	26
Environment variables	26
6. Executing Automated Test Cases	28
7. Result Seeking Strategies	31
7.1. JUnit	31
JUnit case name	31
JUnit class name	31
JUnit method name	31
JUnit suite name	31
7.2. TestNG	31
TestNG class name	32
TestNG method name	32
TestNG suite name	32
7.3. TAP - Test Anything Protocol	32
TAP file name	32
8. Advantages of using Jenkins TestLink Plug-in	33

9. Appendix	34
9.1. Adding attachments to your test results	34
9.2. Using Platforms	34
9.3. Plug-in behavior in distributed environments	35
9.4. How to use the plug-in with SSL or basic authentication?	35
Bibliography	36

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>]:

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 we suggest you to use the LTS (long term support) version of Jenkins. At the moment that this guide is being written, the current version is *1.424.6*. 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 is licensed under the MIT License and its code is hosted at GitHub - <http://github.com/jenkinsci> [<http://github.com/jenkinsci>].

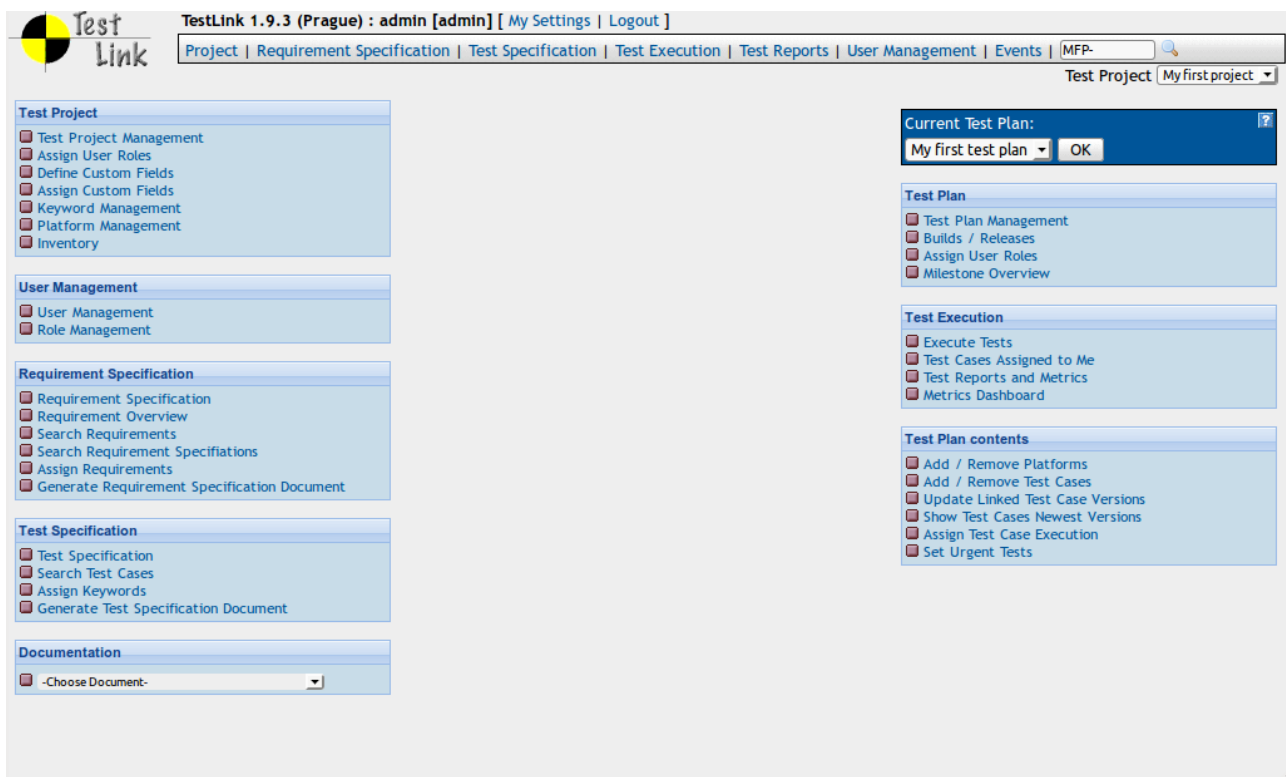
2. An Introduction to TestLink

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 such 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 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/tupilabs/jenkins-testlink-plugin-tutorial> [<https://github.com/tupilabs/jenkins-testlink-plugin-tutorial>] (as well as the DocBook source for this book).

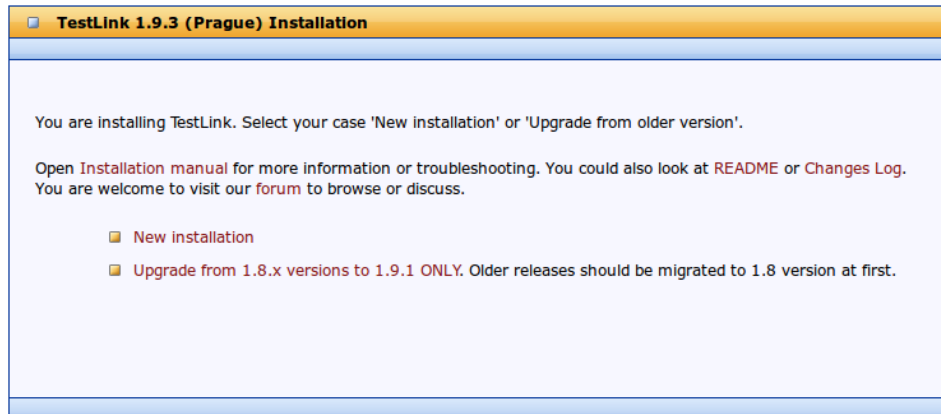
The current version of Jenkins TestLink Plug-in while this book is being written is 3.0.2.

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.



TestLink is developed and shared under GPL license. You are welcome to share your changes with community. Please, confirm your understanding below.

The GNU General Public License is a Free Software license.

Like any Free Software license, it grants to you the four following freedoms:

- *The freedom to run the program for any purpose.*
- *The freedom to study how the program works and adapt it to your needs.*
- *The freedom to redistribute copies so you can help your neighbor.*
- *The freedom to improve the program and release your improvements to the public, so that the whole community benefits.*

You may exercise the freedoms specified here provided that you comply with the express conditions of this license. The principal conditions are:

- *You must conspicuously and appropriately publish on each copy distributed an appropriate copyright notice and disclaimer of warranty and keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy*

☒ I agree to the terms set out in this license.

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

Read/write permissions

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

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:

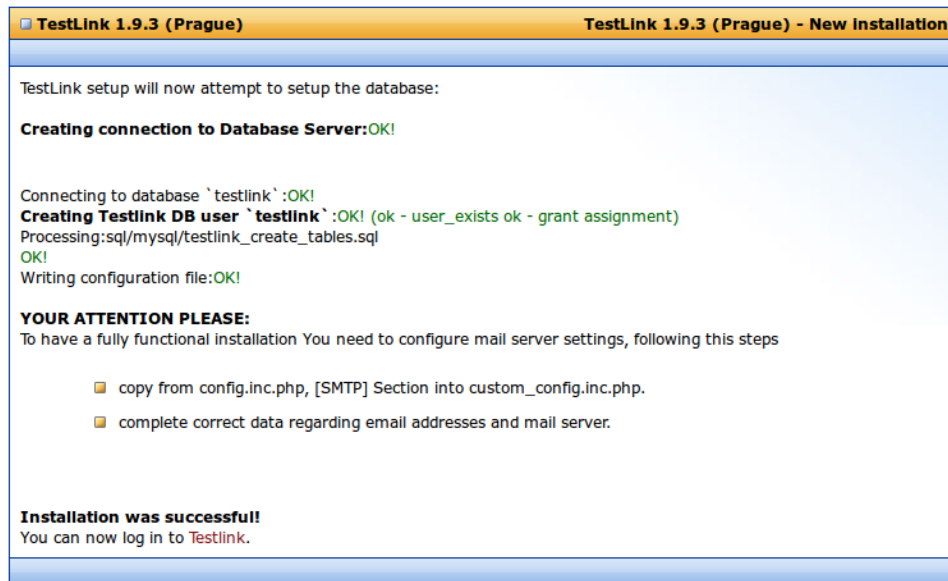
TestLink DB login
TestLink DB password

*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.

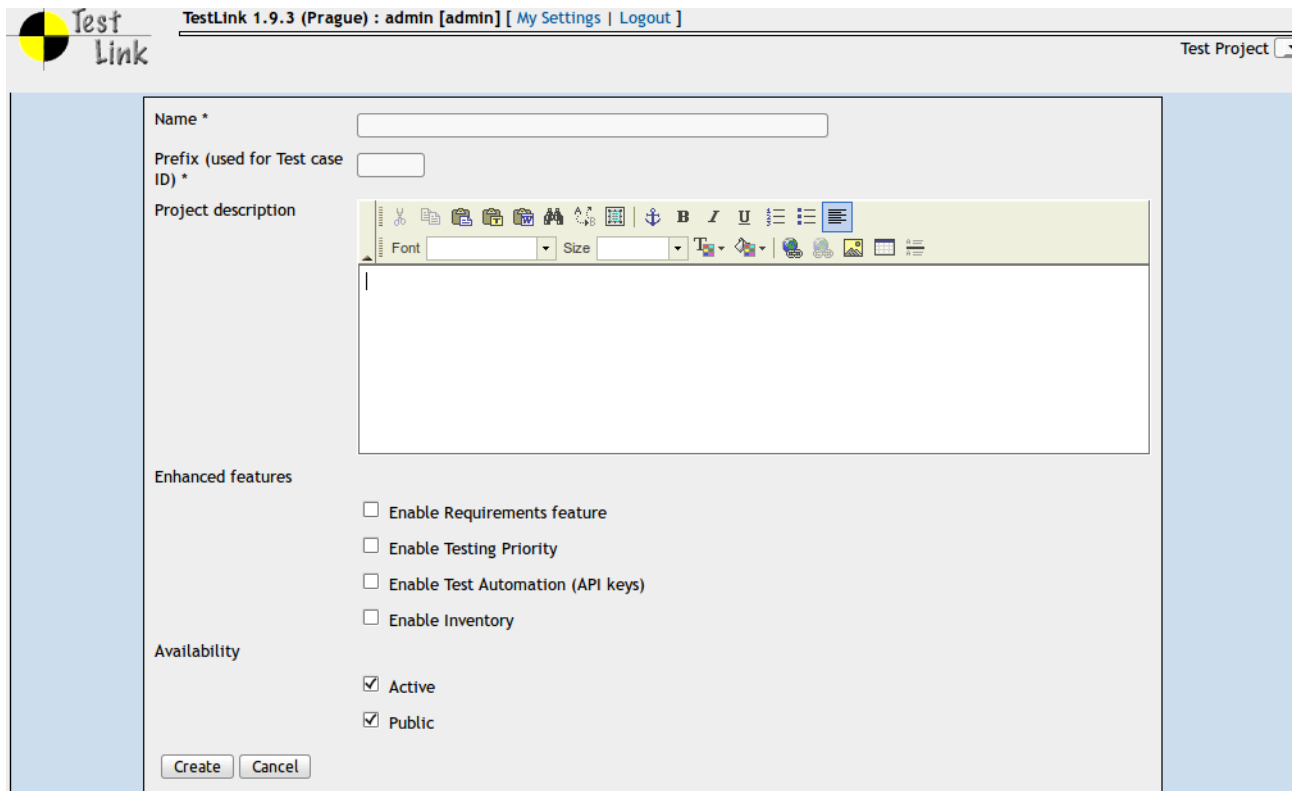
```
$tlCfg->api->enabled = TRUE;
```

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.

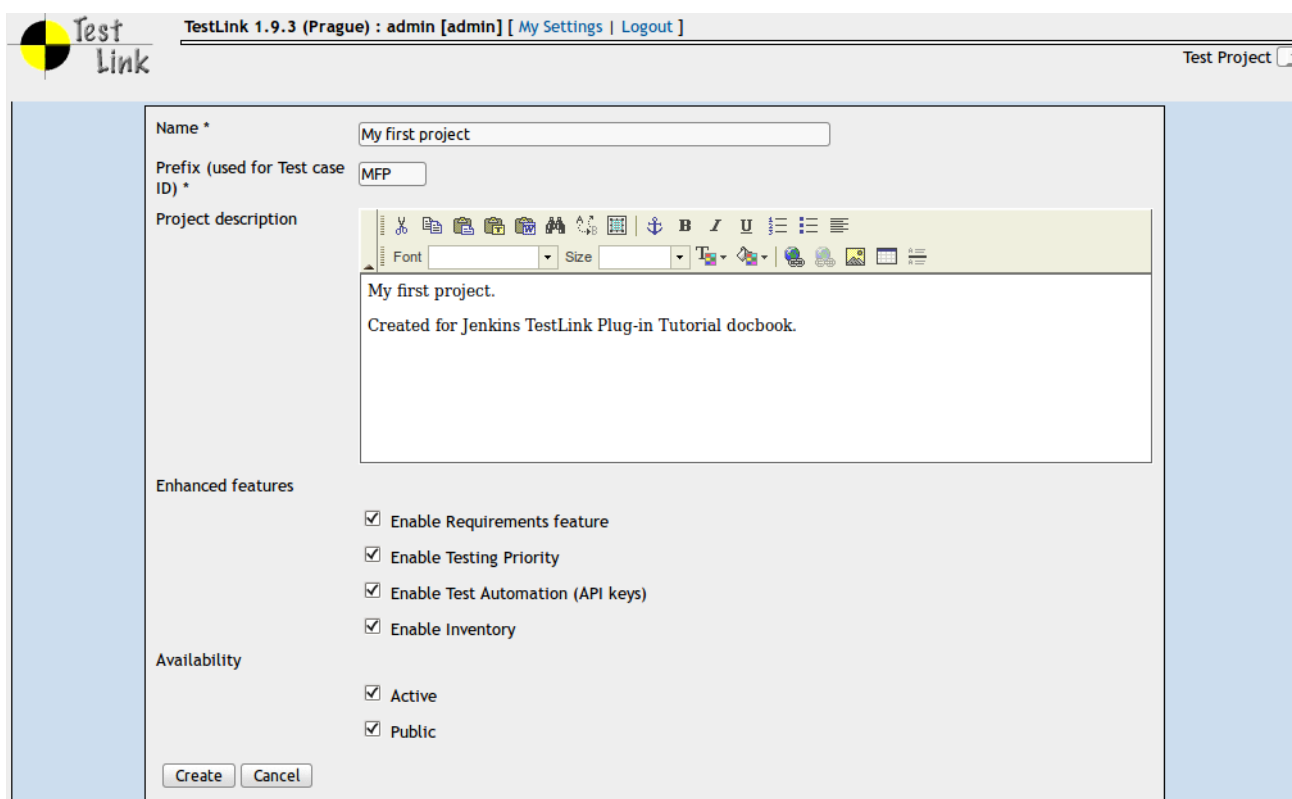


The screenshot shows the TestLink 1.9.3 (Prague) admin interface. The top navigation bar includes the TestLink logo, the version and environment information 'TestLink 1.9.3 (Prague) : admin [admin] [My Settings | Logout]', and a 'Test Project' dropdown menu. The main form is titled 'Create Test Project' and contains the following fields and options:

- Name ***: An empty text input field.
- Prefix (used for Test case ID) ***: An empty text input field.
- Project description**: A rich text editor with a toolbar and a large text area.
- Enhanced features**: A section with four checkboxes:
 - ☐ Enable Requirements feature
 - ☐ Enable Testing Priority
 - ☐ Enable Test Automation (API keys)
 - ☐ Enable Inventory
- Availability**: A section with two checkboxes:
 - ☒ Active
 - ☒ Public

At the bottom of the form are two buttons: 'Create' and 'Cancel'.

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.

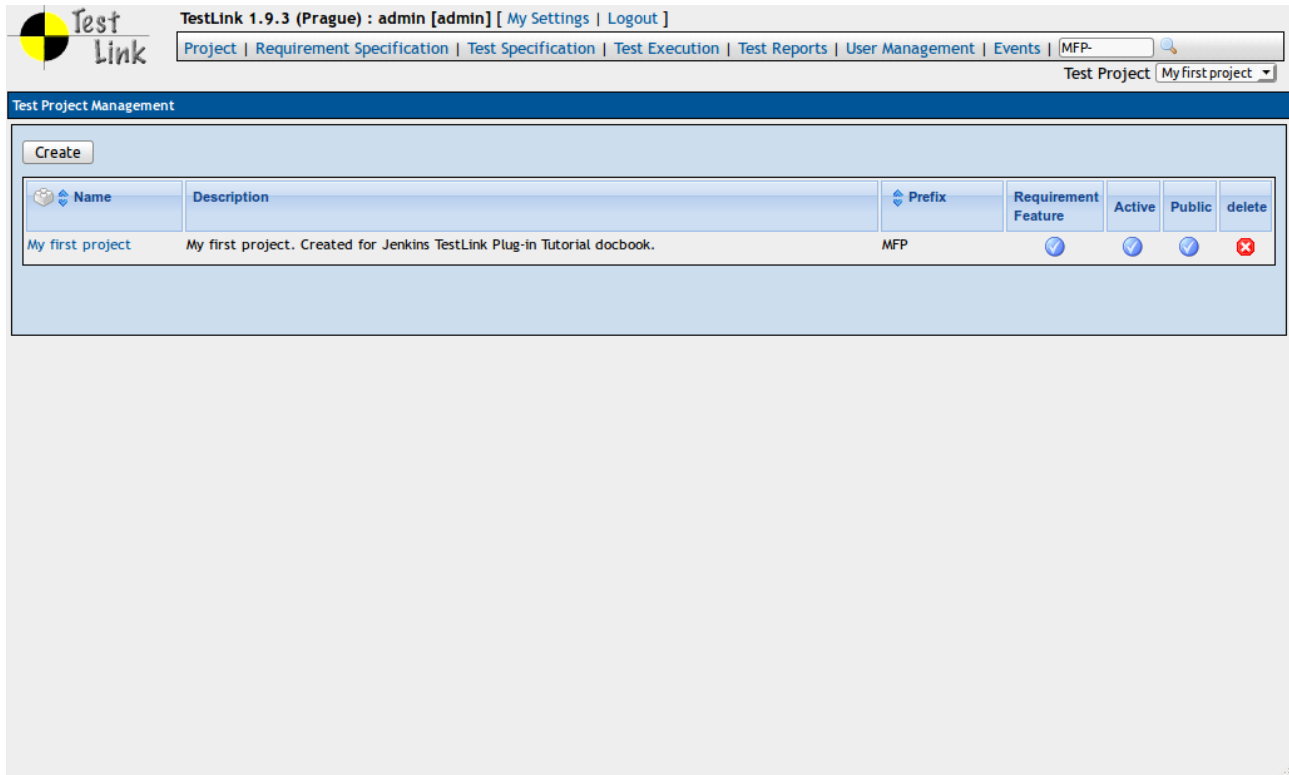


The screenshot shows the TestLink 1.9.3 (Prague) admin interface with the 'Create Test Project' form filled out. The top navigation bar is the same as in the previous screenshot. The form fields are now populated:

- Name ***: 'My first project'
- Prefix (used for Test case ID) ***: 'MFP'
- Project description**: The rich text editor contains the text 'My first project.' and 'Created for Jenkins TestLink Plug-in Tutorial docbook.'
- Enhanced features**: All four checkboxes are checked:
 - ☒ Enable Requirements feature
 - ☒ Enable Testing Priority
 - ☒ Enable Test Automation (API keys)
 - ☒ Enable Inventory
- Availability**: Both checkboxes are checked:
 - ☒ Active
 - ☒ Public

The 'Create' and 'Cancel' buttons remain at the bottom of the form.

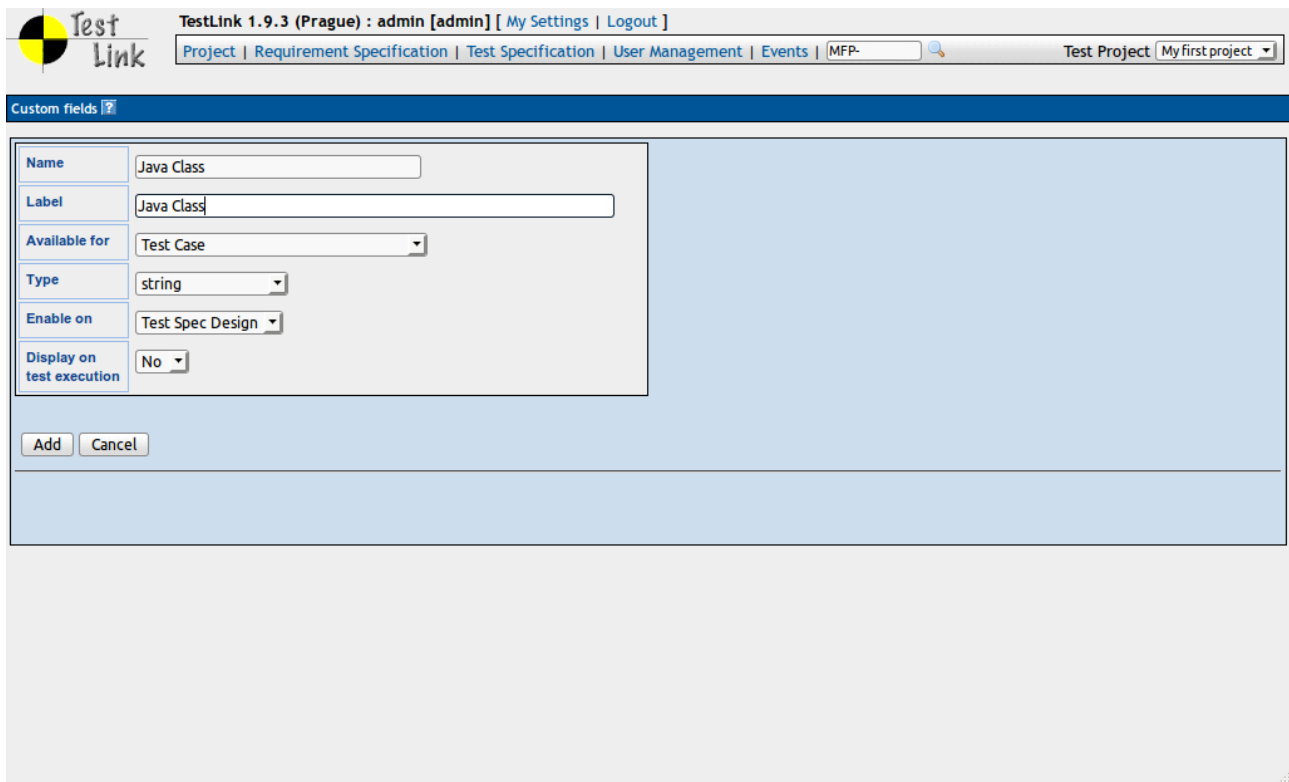
If the following screen is not displayed, review your previous steps or consult TestLink documentation for further assistance.



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

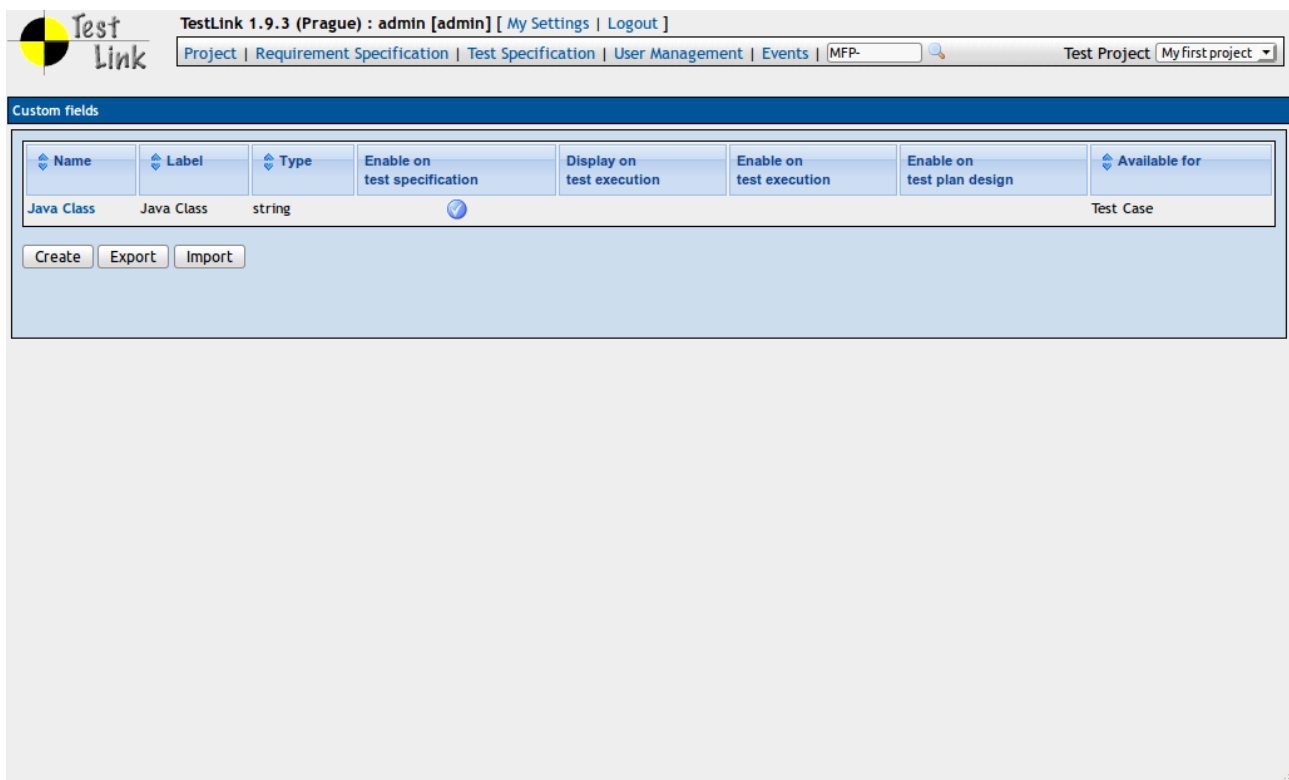
Available for: Test Case

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.



The screenshot shows the Jenkins TestLink 1.9.3 (Prague) interface. The top navigation bar includes links for Project, Requirement Specification, Test Specification, User Management, Events, and a search bar. The 'Test Project' dropdown is set to 'My first project'. The main section is titled 'Assign custom fields - Test Project : My first project'. Below this, there is a table of 'Assigned custom fields'.

<input checked="" type="checkbox"/>	Name	Label	Type	Available for	Display order	Location	Active
<input type="checkbox"/>	Java Class	Java Class	string	Test Case	1	standard	<input checked="" type="checkbox"/>

Below the table are three buttons: 'Unassign', 'Update active status', and 'Save display order and location'.

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 [v]
 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 [v]
 Expand tree Collapse tree

My first project (0)

Test Project : My first project

Test Suite Operations

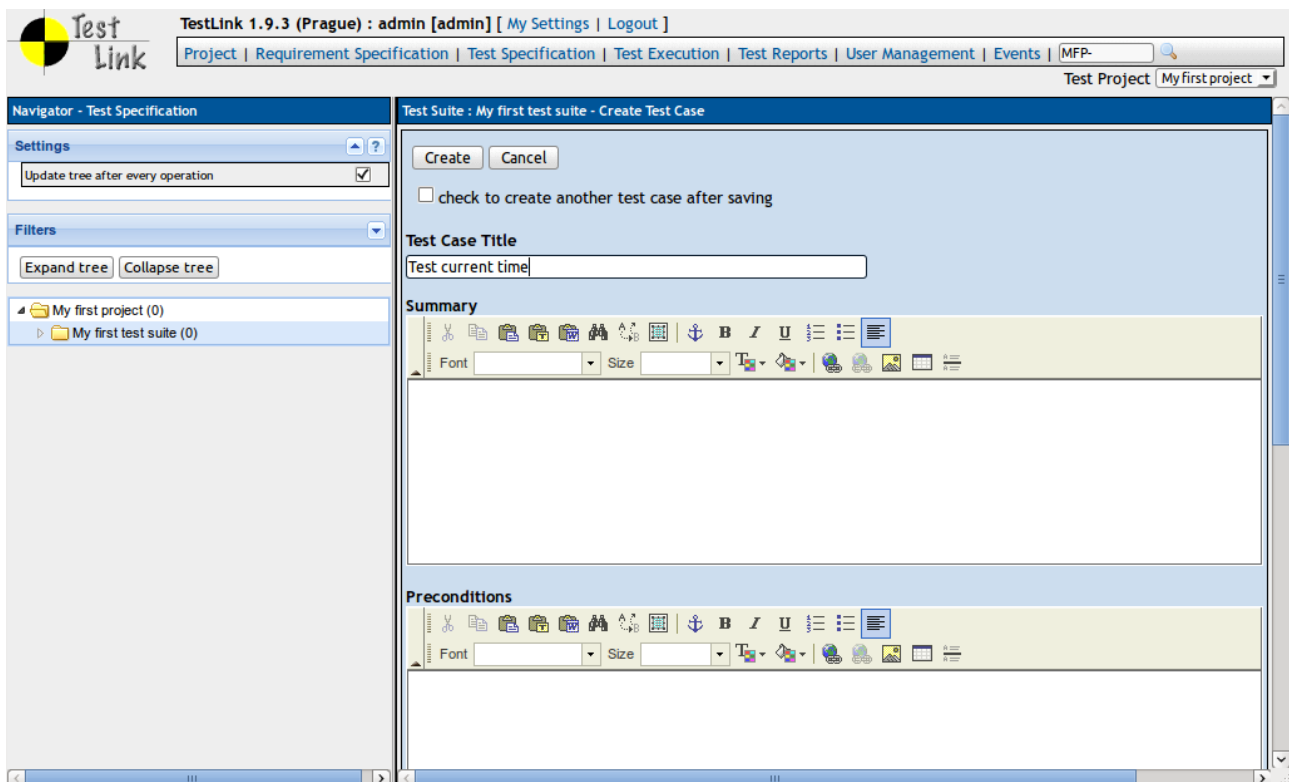
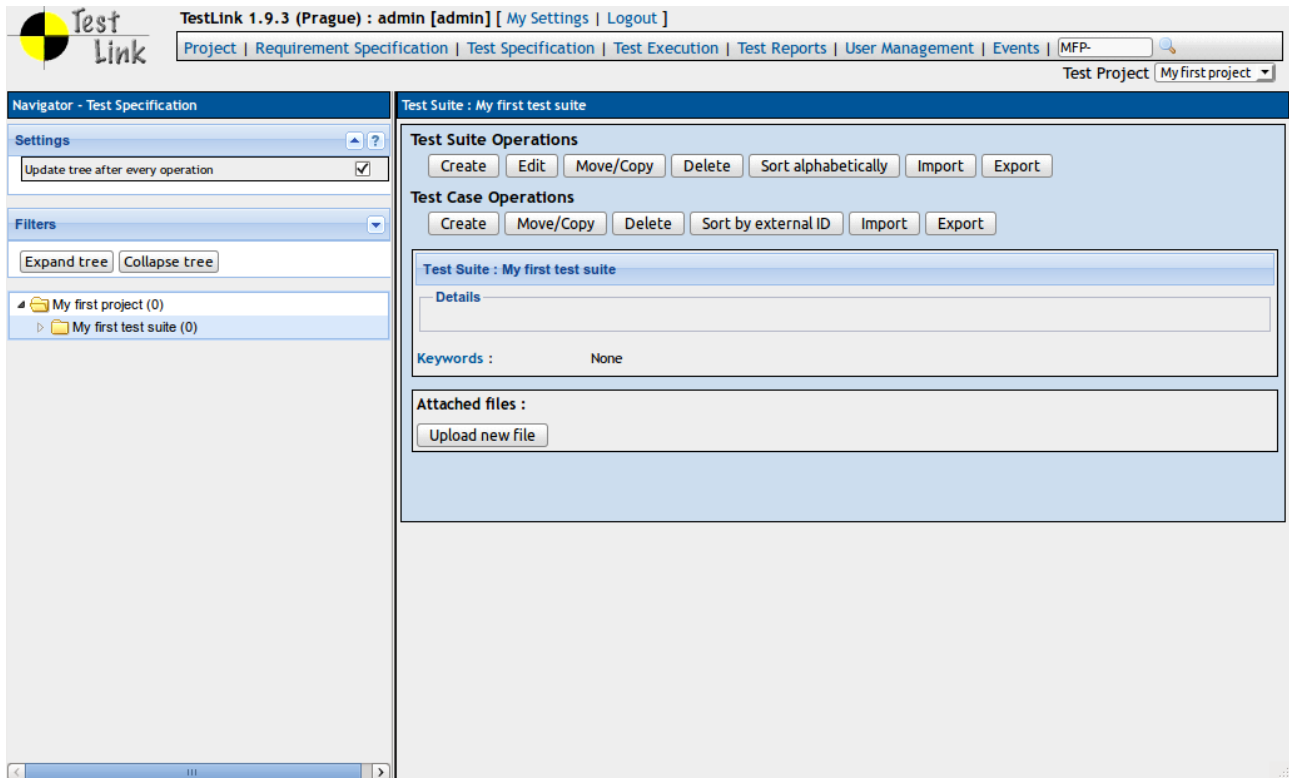
Create Sort alphabetically Import

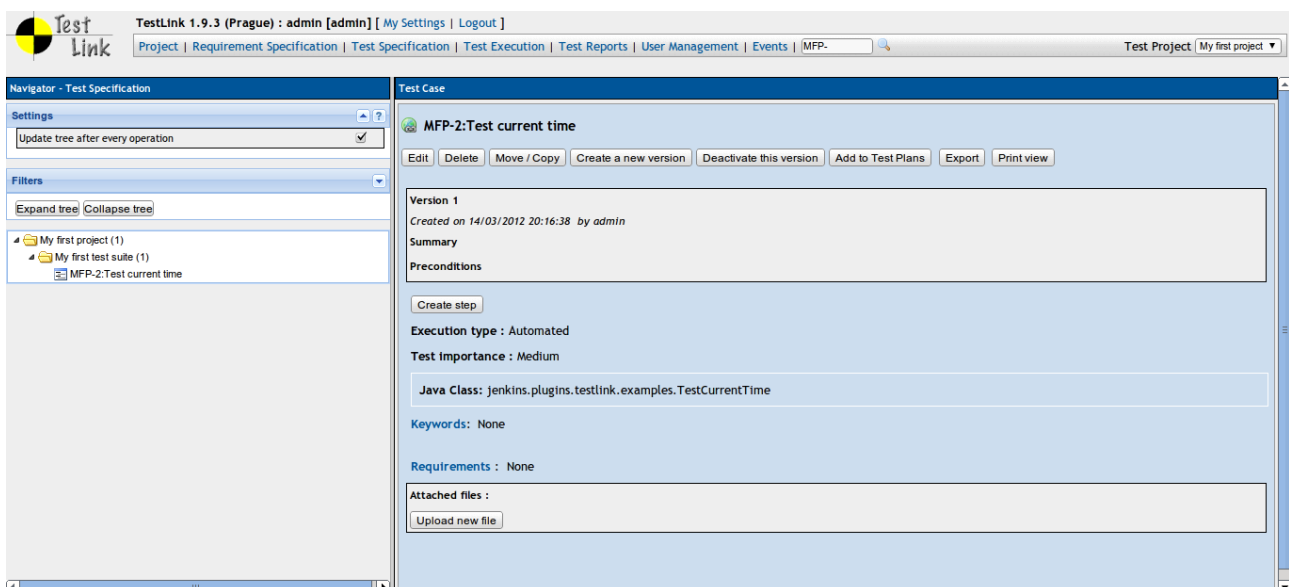
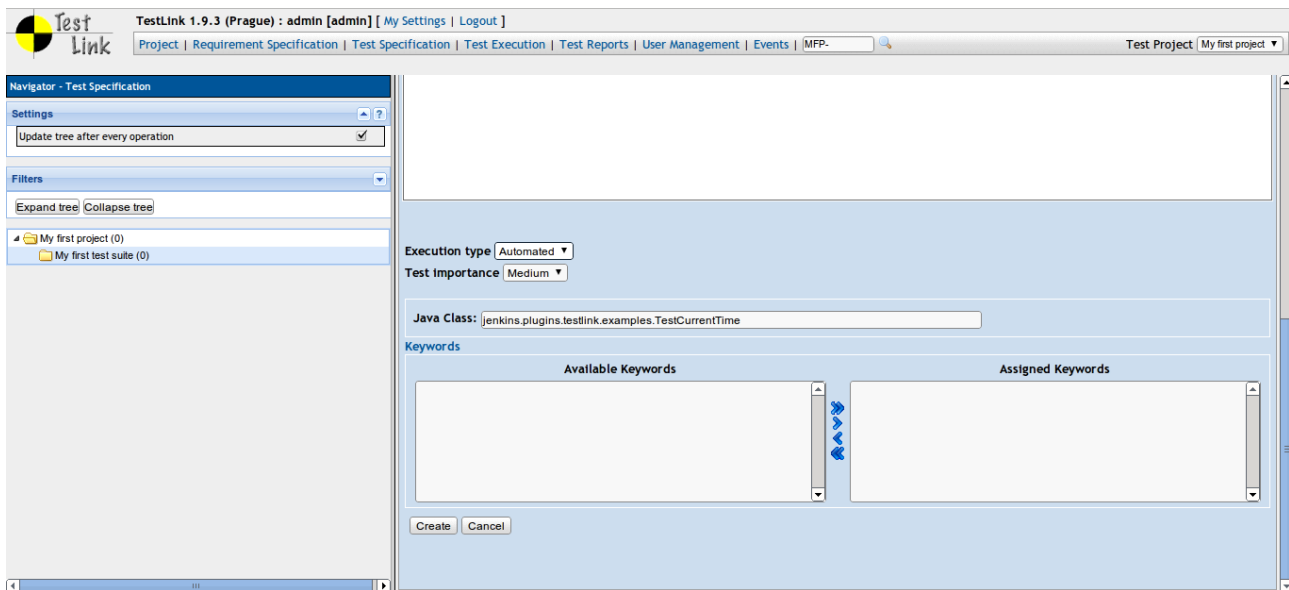
Test Project Name
My first project

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 `jenkins.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.

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

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

Test Plan Management - Test Project My first project

Name: My first test plan

Description: [Rich text editor]

Active: ☒ Public: ☒

Create Cancel

Test plans should encompass a (set of) clearly defined tasks with a timeframe and content. They can be created for everything from simple change requests to new product versions. It is recommended that the description field be used to document links to project plans and related documentation, lists of features to be tested, risks, etc. You can create a new test plan from an existing one. The items that are copied include: builds, test cases, priorities, milestones, and user roles. Test plans can be deactivated (i.e., editing and changing of results change are not allowed). Deactivated test plans are visible only via 'Reporting' and this page.

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 Management - Test Project My first project

Name	Description	Active	Public	Delete	Export	Import
My first test plan		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

Create

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.

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

Settings

Test Plan: My first test plan

Update tree after every operation: ☒

Filters

Expand tree | Collapse tree

My first project

My first test suite (1)

Test Plan : My first test plan - Add/Remove Test Cases to/from Test Plan

Check/uncheck all Test cases for: adding | removal | Add / Remove selected | Save order

My first test suite

Test Case	Version	Importance			
MFP-1: Test current time	1	Medium	1000	<input type="checkbox"/>	09/10/2011

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.

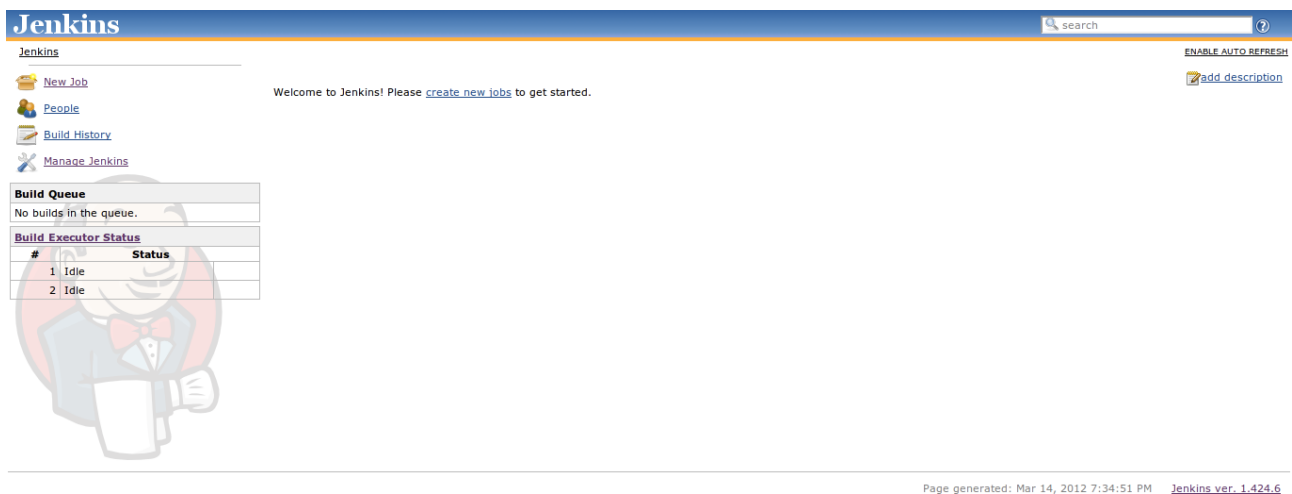
We are done with TestLink for now, the next step is to configure Jenkins.

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.



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"/>	The Schmant plugin enables Hudson to run Schmant build scripts.	1.1.4
<input type="checkbox"/>	SCons Plugin This plugin allows Hudson to invoke SCons build script as the main build step.	0.4
<input type="checkbox"/>	SCTMExecutor This plugin will let users use Borland's SilkCentral Test Manager 2008 R2 or later.	1.5.1
<input type="checkbox"/>	Selenium AES Plugin This plugin is for continuous regression test by Selenium Auto Exec Server (AES) .	0.5
<input type="checkbox"/>	Seleniumhq Plugin This plugin allows you to run and load HTML Selenese suite result generate by Selenium Server from Seleniumhq . Jenkins will generate the trend report of test result. The Seleniumhq plug in can be downloaded here .	0.4
<input type="checkbox"/>	SICCI for Xcode Plugin This plugin integrates support for Xcode projects.	0.0.8
<input type="checkbox"/>	STAF - STAX Plugin This plugin allows Hudson to invoke a STAF command or launch a STAX job as a build step.	0.1
<input type="checkbox"/>	Template Project Plugin This plugin lets you use builders, publishers and SCM settings from another project.	1.3
<input checked="" type="checkbox"/>	TestLink Plugin This plug-in integrates Jenkins and TestLink 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).	3.1.1
<input type="checkbox"/>	Unicorn Validation Plugin 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"/>	WAS Builder Plugin This plugin allows Jenkins to invoke IBM WebSphere Application Server's *wsadmin* as a build step.	1.6.1
<input type="checkbox"/>	Xcode Plugin This plugin adds the ability to call Xcode command line tools to automate build and packaging iOS applications (iPhone, iPad, ...).	1.3
<input type="checkbox"/>	XShell Plugin This plugin defines a new build type to execute a shell command in a cross-platform environment.	0.7
Build Triggers		
<input type="checkbox"/>	BuildResultTrigger Plugin BuildResultTrigger makes it possible to monitor the build results of other jobs.	0.5
<input type="checkbox"/>	DOS Trigger This plugin allows to trigger a build with a DOS script.	1.23
	Downstream-Ext Plugin	

Jenkins

Jenkins » Update center

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



Installing Plugins/Upgrades

Preparation

- Checking internet connectivity
- Checking update center connectivity
- Success

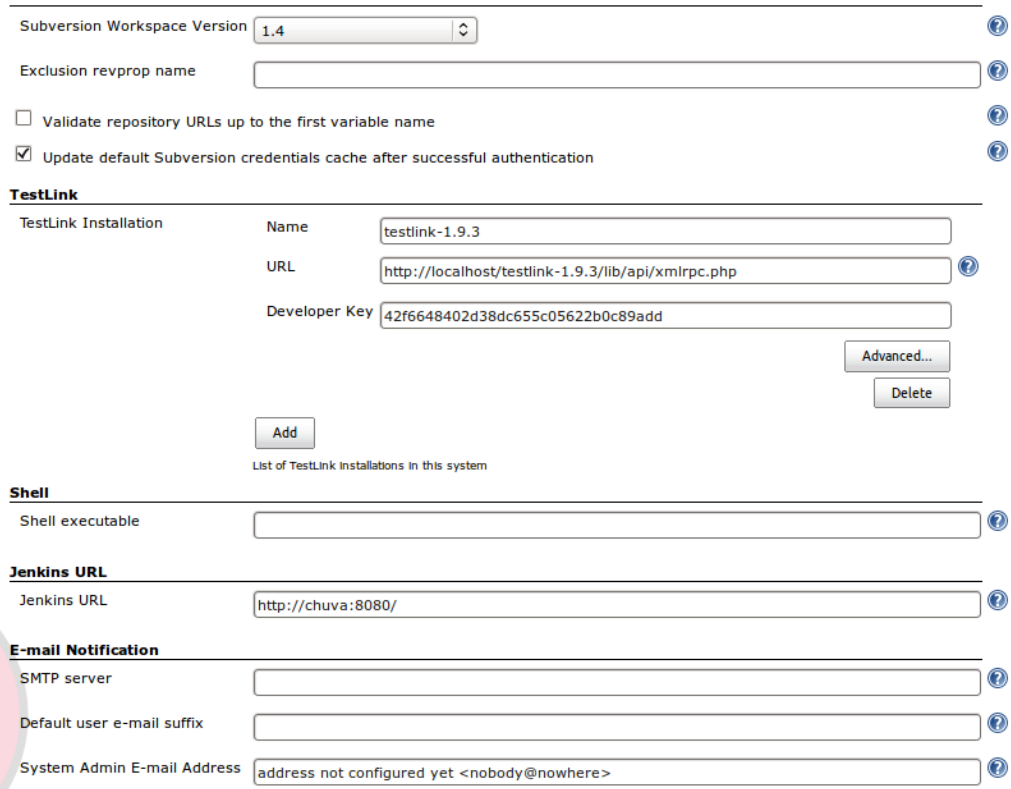
TestLink Plugin Installing

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

ENABLE AUTO REFRESH

Page generated: Mar 14, 2012 7:35:41 PM [Jenkins ver. 1.424.6](#)

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

URL

Developer Key

List of TestLink installations in this system

Shell

Shell executable

Jenkins URL

Jenkins URL

E-mail Notification

SMTP server

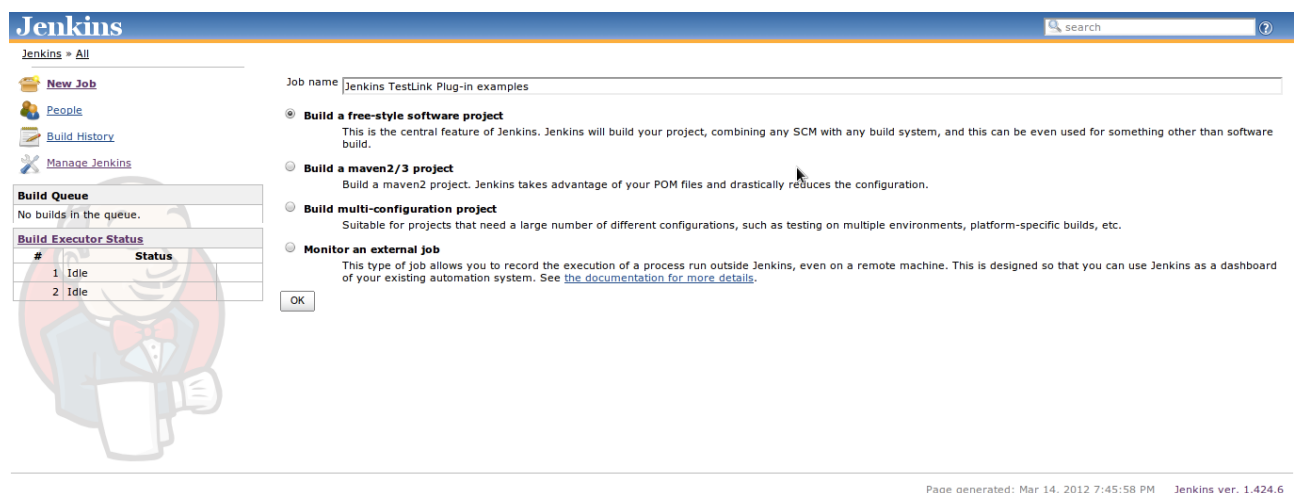
Default user e-mail suffix

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*.



Jenkins

Jenkins » All

Build Queue

No builds in the queue.

Build Executor Status

#	Status
1	Idle
2	Idle

Job name

☒ **Build a free-style software project**
This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.

☐ **Build a maven2/3 project**
Build a maven2 project. Jenkins takes advantage of your POM files and drastically reduces the configuration.

☐ **Build multi-configuration project**
Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.

☐ **Monitor an external job**
This type of job allows you to record the execution of a process run outside Jenkins, even on a remote machine. This is designed so that you can use Jenkins as a dashboard of your existing automation system. See [the documentation for more details](#).

Page generated: Mar 14, 2012 7:45:58 PM [Jenkins ver. 1.424.6](#)

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.

The screenshot shows the Jenkins configuration interface. On the left, there's a sidebar with links like 'Build Now', 'Delete Project', 'Configure', 'Build History', and 'BSS for all' / 'BSS for failures'. The main area is titled 'Advanced Project Options' and includes sections for 'Source Code Management' and 'Build Triggers'. In the 'Source Code Management' section, 'Git' is selected as the SCM, and the 'Repository URL' is set to 'git@github.com:kinow/jenkins-testlink-plugin-tutorial.git'. There are also buttons for 'Add', 'Delete Repository', and 'Delete Branch'. The 'Build Triggers' section is partially visible at the bottom.

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. The plug-in configuration screen contains three sections: *TestLink Configuration*, *Test Execution* and *Result Seeking Strategy*. To fill this form, we will use the configuration created in TestLink throughout Chapter 4, *TestLink Configuration*.

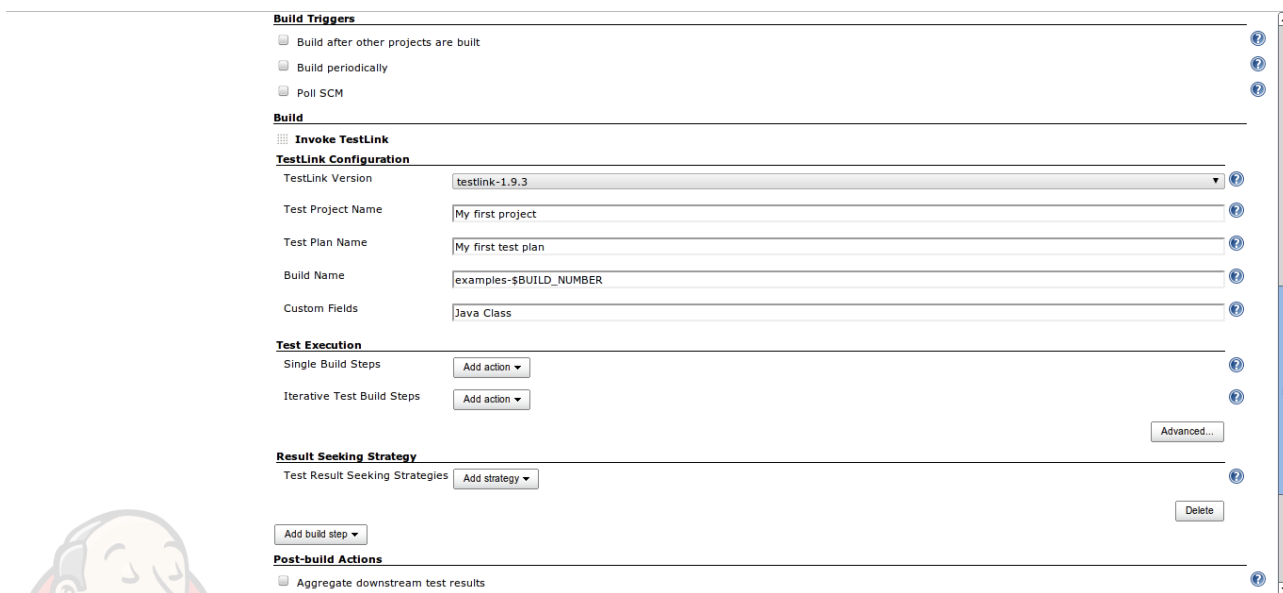
TestLink Configuration

In this section, you are asked to provide the configuration that the plug-in will use to connect to TestLink and retrieve the data for your automated tests.

Use the following settings for this section:

Table 5.1. TestLink Configuration 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



The image shows the Jenkins TestLink configuration page. It is divided into several sections: **Build Triggers** (with checkboxes for 'Build after other projects are built', 'Build periodically', and 'Poll SCM'), **Build** (containing **Invoke TestLink** and **TestLink Configuration**), **Test Execution**, **Result Seeking Strategy**, and **Post-build Actions**. The **TestLink Configuration** section includes fields for 'TestLink Version' (testlink-1.9.3), 'Test Project Name' (My first project), 'Test Plan Name' (My first test plan), 'Build Name' (examples-\$BUILD_NUMBER), and 'Custom Fields' (Java Class). The **Test Execution** section has 'Single Build Steps' and 'Iterative Test Build Steps' with 'Add action' buttons. The **Result Seeking Strategy** section has a 'Test Result Seeking Strategies' dropdown and an 'Add strategy' button. The **Post-build Actions** section has an 'Aggregate downstream test results' checkbox. There are also 'Advanced...' and 'Delete' buttons on the right side of the page.

Test Execution

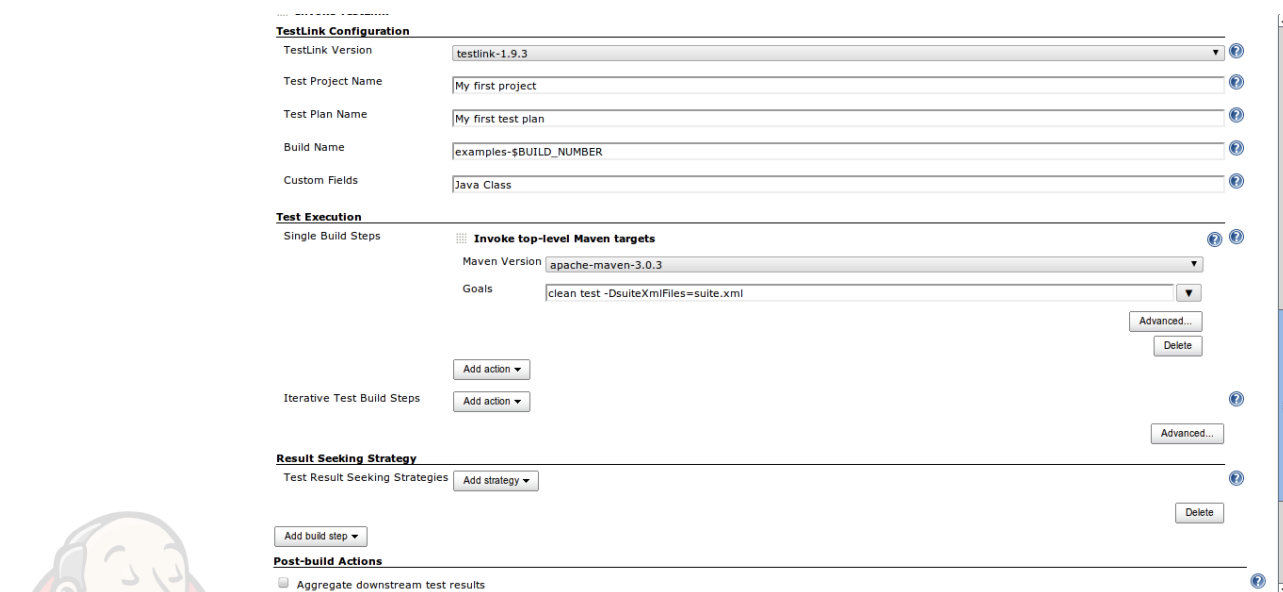
This section contains the configuration to execute your automated tests. You can add *Single Build Steps*, which are executed once per build, or *Iterative Build Steps*, which are executed once per automated test found in TestLink.

There is an advanced section, that is hidden by default, with extra hooks.

Use the following settings for this section:

Table 5.2. Test Execution settings explanation

Single Build Steps	Add an <i>Invoke top-level Maven targets</i> build step, and as goals use test -DsuiteXmlFiles=suite.xml
Iterative Test Build Steps	Leave it the way it is



The image shows the Jenkins TestLink configuration page with the **Test Execution** section expanded. The **TestLink Configuration** section is the same as in the previous image. The **Test Execution** section now shows the **Invoke top-level Maven targets** build step selected for 'Single Build Steps'. This step has a 'Maven Version' dropdown set to 'apache-maven-3.0.3' and a 'Goals' field containing 'clean test -DsuiteXmlFiles=suite.xml'. There are 'Add action', 'Advanced...', and 'Delete' buttons for this step. The 'Iterative Test Build Steps' section still has an 'Add action' button. The 'Result Seeking Strategy' and 'Post-build Actions' sections are also visible.

Result Seeking Strategy

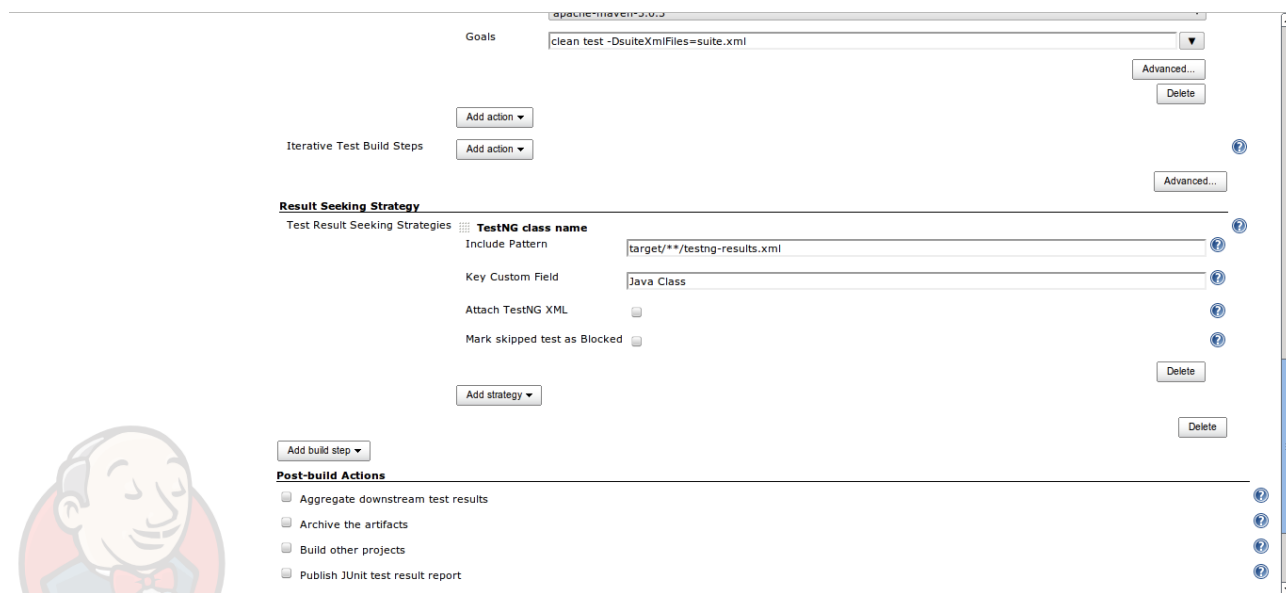
Finally, here you can configure the *Result Seeking Strategy*. The plug-in implements the Strategy pattern for seeking test results. What means that depending on your configuration, the plug-in might use certain algorithm for parsing and updating test results in TestLink.

Each strategy can have different settings. You can use more than one strategy, but you will have to take care to not have the same test in two different strategies.

Use the following settings for this section:

Table 5.3. Result Seeking Strategy settings explanation

Test Result Seeking Strategies	Add the <i>TestNG class name</i> strategy.
Include pattern	target/**/testng-results.xml
Key Custom Field	Java Class
Attach TestNG XML	Leave it the way it is
Mark skipped test as Blocked	Leave it the way it is



Save your job.

5.4 Explaining the Job configuration parameters

TestLink Configuration

TestLink Version: testlink-1.9.3

Test Project Name: My first project

Test Plan Name: My first test plan

Build Name: examples-\$BUILD_NUMBER

Custom Fields: Java Class

Test Execution

Single Build Steps: **Invoke top-level Maven targets**

Maven Version: apache-maven-3.0.3

Goals: clean test -DsuiteXmlFiles=suite.xml

Advanced... Delete

Iterative Test Build Steps: Add action

Advanced...

Result Seeking Strategy

Test Result Seeking Strategies: **TestNG class name**

Include Pattern: target/**/testng-results.xml

Key Custom Field: Java Class

Attach TestNG XML: ☐

Mark skipped test as Blocked: ☐

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.

On setting the build name you should wisely choose and review the underlying concept and workflow inside your testing process. Mind that in previous chapter it is mentioned that creating 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.

Unfortunately TestLink is been reported to slow down enormously and to be unresponsive on a project's TestLink own web interface on high amount of TestLink builds being in status open.

Thus if using environment variables in this build name field of Jenkins that might change their value often (like Jenkins' build number, current time or date) an increasing number of TestLink Builds will be created with correlated frequency accordingly inside TestLink then.

As a first recommendation your process should assure that the amount of open builds in TestLink is limited in an acceptable range. Even if this might mean to manually close builds in TestLink. Mind that especially in

huge testing projects this means that this type of strategy does not scale. Another recommendation could be to simply avoid or at least reduce any usage of frequently changing environment variables in that place thus avoiding all or most new creation of open builds.

Single Build Steps

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

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.

Test Result Seeking Strategies

This is the Result Seeking Strategy chosen to seek and update test results in TestLink.

Include pattern

The plug-in uses this pattern to find test results of your tests execution (single test command or iterative). It supports Ant-like expressions like `TEST*.xml` or `target/**/*.testng*.xml`.

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.

Attach TestNG XML

If this option is checked the plug-in will attach the TestNG XML file to TestLink execution. This way you can open the XML in TestLink.

Mark skipped test as Blocked

By default, in TestNG, skipped tests are not updated in TestLink. It means that they appear as Not Run in TestLink. If this option is checked, the plug-in updates the test case in TestLink as Blocked.

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

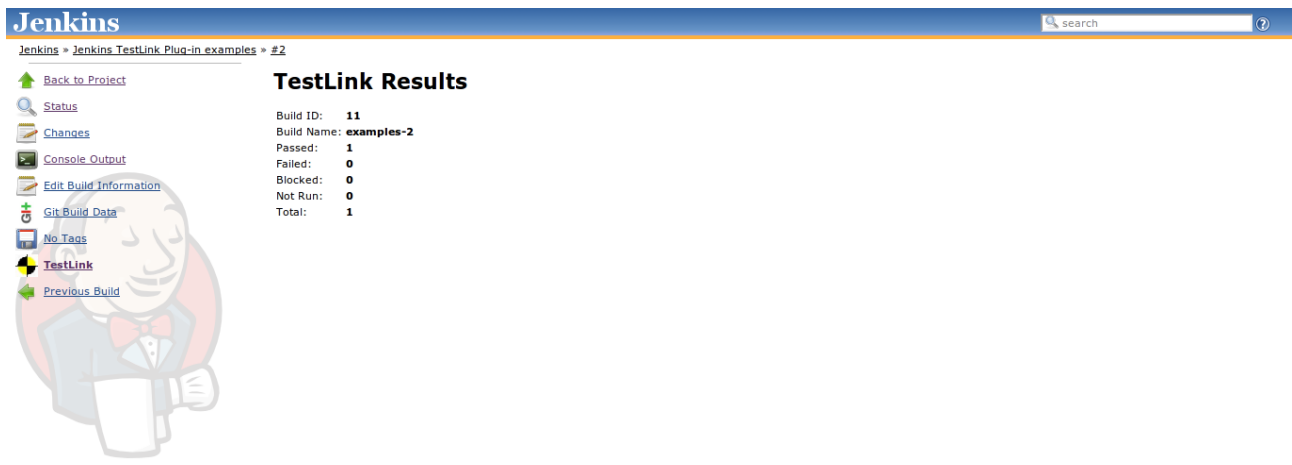
Now that everything is configured it is time to execute our automated test cases. Let's go back to the main screen in Jenkins. Click on your job and trigger your job execution by clicking on *Build Now*.

The screenshot shows the Jenkins dashboard for the 'Project Jenkins TestLink Plug-in examples' job. The sidebar on the left contains links: 'Back to Dashboard', 'Status', 'Changes', 'Workspace', 'Build Now', 'Delete Project', 'Configure', and 'TestLink results'. The main content area has 'Workspace' and 'Recent Changes' links. Below these are 'Permalinks' for 'Last build (#2)', 'Last stable build (#2)', and 'Last successful build (#2)'. A 'Build History' table shows two builds: #2 (Mar 14, 2012 8:40:19 PM) and #1 (Mar 14, 2012 8:37:04 PM). A 'TestLink results' graph is partially visible on the right, showing a 'TestLink Tests Count' on the y-axis and a legend for 'Blocked', 'Failed', 'Not Run', and 'Passed'.

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.

The screenshot shows the Jenkins dashboard for the 'Project Jenkins TestLink Plug-in examples' job. The sidebar on the left contains links: 'Back to Dashboard', 'Status', 'Changes', 'Workspace', 'Build Now', 'Delete Project', 'Configure', and 'TestLink results'. The main content area has 'Workspace' and 'Recent Changes' links. Below these are 'Permalinks' for 'Last build (#2)', 'Last stable build (#2)', and 'Last successful build (#2)'. A 'Build History' table shows two builds: #2 (Mar 14, 2012 8:40:19 PM) and #1 (Mar 14, 2012 8:37:04 PM). A 'TestLink results' graph is fully visible on the right, showing a 'TestLink Tests Count' on the y-axis and a legend for 'Blocked', 'Failed', 'Not Run', and 'Passed'.

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](#)
- [No Tags](#)
- TestLink**
- [Previous Build](#)

TestLink Results

Build ID: **11**
 Build Name: **examples-2**
 Passed: **1**
 Failed: **0**
 Blocked: **0**
 Not Run: **0**
 Total: **1**

Page generated: Mar 14, 2012 8:44:06 PM [Jenkins ver. 1.424.6](#)



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](#)
- [No Tags](#)
- TestLink**
- [Previous Build](#)

Build #2 (Mar 14, 2012 8:40:19 PM)

[Delete this build](#)

[add description](#) Started 3 min 59 sec ago Took 2 min 29 sec

No changes.

Started by user [anonymous](#)

Revision: 9c56c39d17b2ba34a2c7d669865fdb5e0d9deaf1

- origin/HEAD
- origin/master

TestLink build ID: 11

TestLink build name: examples-2

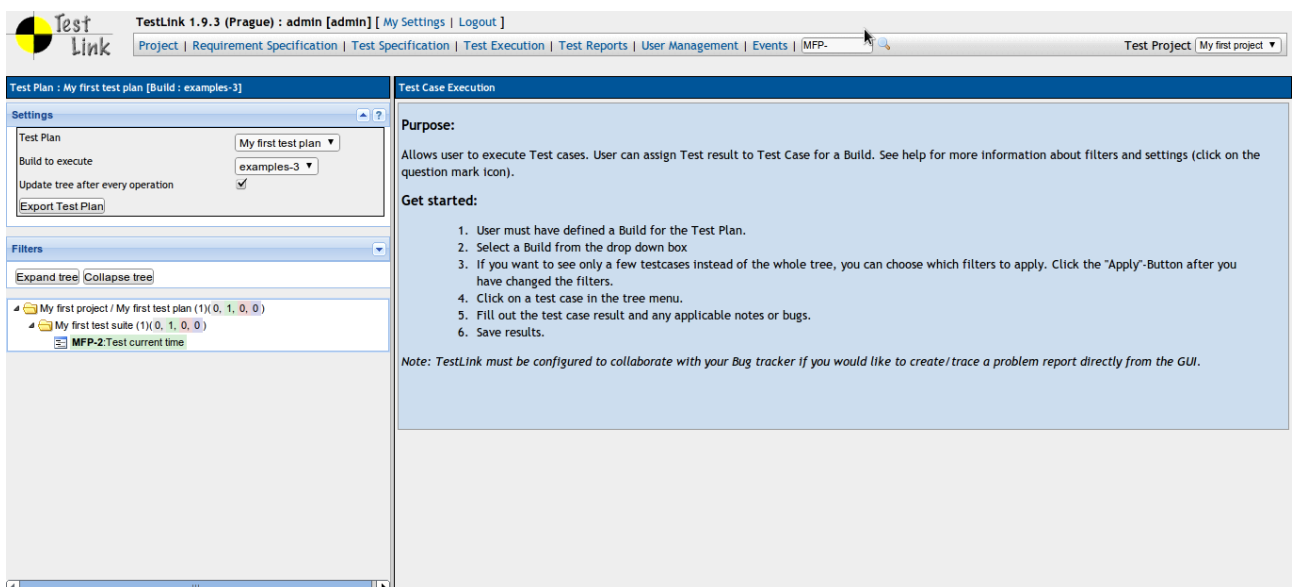
Total of 1 tests. Where 1 passed, 0 failed, 0 were blocked and 0 were not executed.

List of test cases and execution result status

Test case ID	Version	Name	Test project ID	Execution status
38	1	Test current time	33	Passed

Page generated: Mar 14, 2012 8:44:19 PM [Jenkins ver. 1.424.6](#)

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-3]

Settings

Test Plan: My first test plan

Build to execute: examples-3

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-2: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.

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-3]

Settings

Test Plan: My first test plan

Build to execute: examples-3

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-2:Test current time

Test Results on Build examples-3

Test plan notes

Build description

Print Show complete execution history Import XML Results

Test Suite: My first test suite/

Test Case ID MFP-2 :: Version : 1

Test current time

No tester assigned

Last execution (any build) - Build : examples-2

Date : 14/03/2012 20:42:48 - Tested by : admin - Build : examples-2 - Status : Passed

Last execution (current build) - Build : examples-2

Date	Build	Tested by	Status	Test Case Version	attachments	Run mode
14/03/2012 20:35:06	examples-3	admin	Passed	1		

Summary

Preconditions

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

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

Reports and Metrics

Report Format: HTML Print

Test Plan: My first test plan

Show inactive test plans

- Test Plan Report
- Test Report
- General Test Plan Metrics
- Results by Tester per Build
- Test Case Assignment Overview
- Query Metrics
- Test result matrix
- Failed Test Cases
- Blocked Test Cases
- Not run Test Cases
- Test Cases without Tester Assignment
- Charts
- Requirements based Report
- Test Cases with Custom Fields set on Execution
- Test Plan with Custom Field Info
- Test Cases not assigned to Any Test Plan

General Test Plan Metrics

Test Project : My first project

Test Plan : 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

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.

Test results according to test priorities

Priority	Total	Not Run	[%]	Passed	[%]	Failed	[%]	Blocked	[%]	Completed [%]
High	0	0	N/A	0	N/A	0	N/A	0	N/A	0
Medium	1	0	0.00	1	100.00	0	0.00	0	0.00	100.00

7. Result Seeking Strategies

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 supports three different formats of results: *JUnit*, *TestNG* and *TAP*.

7.1 JUnit

```
<?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>
```

JUnit case name

This strategy matches the case name with the key custom field value.

JUnit class name

This strategy matches the case class name with the key custom field value.

JUnit method name

This strategy matches `<package>.<classname>#<methodname>` with the key custom field value (e.g.: `com.example.MyClass#testSomething`)

JUnit suite name

This strategy matches the suite name with the key custom field value.

7.2 TestNG

```
<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>
```

TestNG class name

This strategy matches the class name with the key custom field value.

TestNG method name

This strategy matches `<package>.<classname>#<methodname>` with the key custom field value (e.g.: `com.example.MyClass#testSomething`)

TestNG suite name

This strategy matches the suite name with the key custom field value.

7.3 TAP - Test Anything Protocol

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*.

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

TAP file name

This strategy matches the TAP Stream file name with the key custom field value.

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 plug-in team members are contributors of TestLink and maintainers of the TestLink Java API. We try 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/95RfjjkmFmwgVXrvvffeelPBjtKbCoJ0qaIodqMYe8EWwRolatQkRmOixkTFAkYFvu8f\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]. Copyright © 2011 Wikipedia.
- [YAMLOrg] *YAML.org* [<http://www.yaml.org>]. Copyright © 2011 YAML.org.