

Vaadin TestBench User's Manual

Vaadin TestBench
2.0



Vaadin TestBench is an environment used for automated user interface regression testing of Vaadin applications on multiple platforms and browsers.

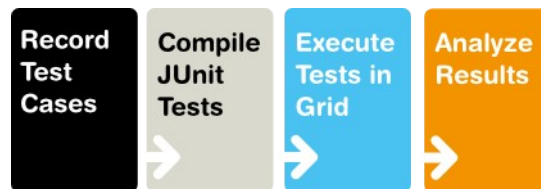
Vaadin TestBench allows you to run tests on the UI after a build and catch problems created by changes to the business logic. This helps you catch problems that affect the UI functionality early on, before they become a real problem.

1. Introduction to Vaadin TestBench

1.1. Overview

Quality assurance is one of the cornerstones of modern software development. Extending throughout the entire development process, quality assurance is what binds the end product to the requirements. In iterative development processes, with ever shorter release cycles and continuous integration, the role of regression testing is central. The special nature of web applications creates many unique requirements for regression testing.

Vaadin TestBench makes it possible to automate the regression testing of web applications that use Vaadin. You record test cases by interacting with your application. After recording, you can compile the test as JUnit tests and run them for as many times as you want, on multiple platforms and browsers. Test results are stored in a centralized test server for later analysis and quality assurance.



The main features are:

- Recording and playing back test cases using a recorder in browser
- Validating UI state by assertion points and screen capture comparison
- Screen capture comparison with difference highlighting
- Execution of tests through **JUnit**
- Distributed test grid for running tests
- Integration with unit testing

Execution of tests are distributed over a grid of test nodes, which speeds up testing. The grid nodes can run different operating systems and have different browsers installed. In a minimal setup, such as for recording the tests, you can use Vaadin TestBench on just a single computer.

Vaadin TestBench is based on the **Selenium** testing framework and **Selenium Grid** for distributed testing. Selenium is augmented with Vaadin-specific extensions, such as the screen capture feature.

1.2. TestBench Components

The main components of Vaadin TestBench are:

- Vaadin TestBench Recorder
- Vaadin TestBench Library
- Vaadin TestBench Grid Hub
- Vaadin TestBench Grid Remote Control

The components and a basic setup are illustrated in Figure 1.

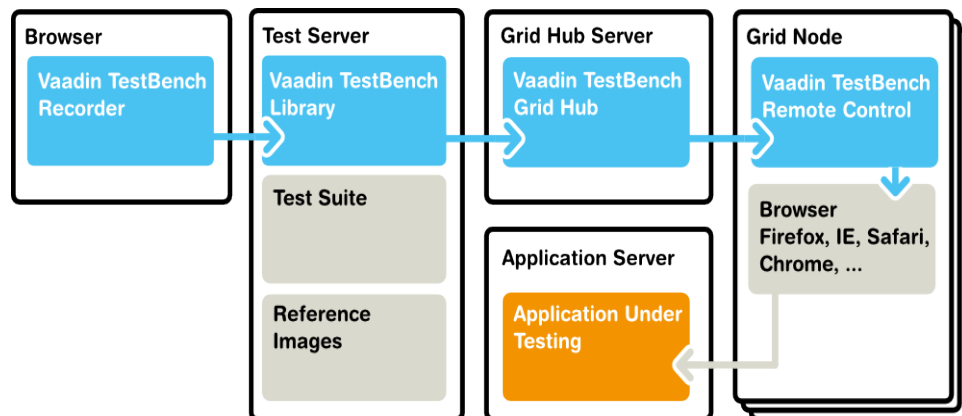


Figure 1: Vaadin TestBench Architecture

Recording test cases requires **Vaadin TestBench Recorder**, which is a Mozilla Firefox extension that you install in your browser. It provides a control panel to record test cases and play them back. You can play test cases right in the recorder and later automatically by the Grid Node Remote Control, which opens a browser and starts the recorder in playback mode.

The test suite and results from test runs are stored in a test server. A test suite includes recorded test cases and possible reference images for similarity tests.

Vaadin TestBench Library provides the central control logic for:

- Converting tests from recordings to JUnit tests
- Executing tests in the Vaadin TestBench Grid Hub
- Collecting test results
- Comparing screen captures with reference images

Vaadin TestBench Grid Hub is a service that distributes test tasks to nodes in the test grid. It contacts the Grid Node Controller in a node and asks it to open a specific browser and run specific tests in it. The Grid Hub runs in a server which can be dedicated, one of the grid nodes, or the test server.

Vaadin TestBench Grid Node Controller is a service that runs in each grid node. It is able to open any of the browsers installed in a node and start the recorder in the playback mode to execute the tests. It receives requests to execute tests from the grid hub and reports the results back to it.

A basic setup for a Vaadin TestBench environment consists of:

- A workstation with Firefox and the TestBench Recorder for recording tests
- A build/test server used to build, launch, and test the web application
- A server running the Grid Hub
- One or more servers running Grid Remote Controls

The workstation and servers can be separate computers or one computer can work in multiple roles.

1.3. Requirements

1.3.1. Requirements for Vaadin TestBench Recorder

For recording and playback with Vaadin TestBench Recorder:

- Mozilla Firefox 3 or newer

1.3.2. Requirements for Automated Testing

For running tests:

- Java JDK 1.5 or newer
- Browsers installed on test nodes
- Apache Ant or some other way to run Ant scripts (recommended)

1.3.3. Continuous Integration Compatibility

Vaadin TestBench is tested to work with TeamCity build management and continuous integration server and should work with other continuous integration systems as well.

1.3.4. Known Compatibility Problems

Firebug should be disabled

Firebug injects a `<div id="_firebugConsole">` element under the `<body>` element in some cases (the element is invisible in the Firebug's HTML structure browser). This can disturb recording of test cases, especially when closing notifications. Firebug 1.6 should fix this issue, but we could still encourage TestBench users to disable Firebug when recording or playing test cases.

2. Installing Vaadin TestBench

2.1. Overview

The installation of Vaadin TestBench covers the following tasks:

- Download and unpack the Vaadin TestBench installation package
- Install Vaadin TestBench Recorder
- Install Vaadin TestBench Library and launch scripts
- Install Vaadin TestBench Grid Hub
- Install Vaadin TestBench Grid Remote Controls

You only need to install the Recorder first. It allows you to record tests and play them back in the browser. The rest of the installation tasks are for running automated tests from a test server using a grid.

Two basic installation types are covered in these instructions:

- Test development installation on a workstation
- Distributed grid installation

2.1.1. Test Development Installation

In a typical small test development setup, you will install all the components in a single workstation.

For a development installation with multiple users, the Testing Server is located in a server host different from workstation(s), so you need to configure the URL address of the Test Server.

See Section 2.4: *Quick Setup for Playback on a Workstation* for a quick test development setup.

2.1.2. A Distributed Test Environment

A Vaadin TestBench grid consists of two categories of components:

- Vaadin TestBench Grid Hub service
- Grid nodes running Vaadin TestBench Grid Remote Control

The hub is a service that handles communication between the JUnit test runner and the node controllers. The node controllers are services that can launch a browser and perform the actual execution of test commands in the browser.

The hub requires very little resources, so you would typically run it either in the test server or in one of the nodes.

In a full distributed setup, you will install the Vaadin TestBench components in separate hosts.

2.2. Downloading and Unpacking the Installation Package

First, download the installation package `Vaadin-TestBench-<version>.zip` and extract the installation package in some suitable folder.

Windows

In Windows, use the default ZIP decompression feature to extract the package into your chosen directory, for example, `C:\dev`.

Warning: The default decompression program in Windows XP and Vista as well as some versions of WinRAR cannot unpack the installation package properly in certain cases. Decompression can result in an error such as “*The system cannot find the file specified.*” This can happen because the default decompression program is unable to handle long file paths where the total length exceeds 256 characters. This occurs, for example, if you try to unpack the package under Desktop. You should unpack the package directly into `C:\dev` or some other short path or use another decompression program.

Linux, MacOS X, and other UNIX

In Linux, Mac OS X, and other UNIX-like systems, use Info-ZIP or other ZIP software with the command:

```
$ mkdir Vaadin-TestBench
$ cd Vaadin-TestBench
$ unzip vaadin-6.x.x.zip
```

The contents of the installation package will be extracted under `Vaadin-TestBench` installation directory in the chosen directory.

2.3. Installing the Recorder

An environment for developing tests requires the use of the Vaadin TestBench Recorder to record test cases and to play them back.

After extracting the files from the installation package, do the following:

1. Enter the `testbench-recorder` directory under the installation directory.
2. Open Mozilla Firefox
3. Either drag and drop the `Vaadin-TestBench-recorder-<version>.xpi` to an open Firefox window or open it from the **File** menu.
4. Firefox will ask if you want to install the TestBench Recorder extension. Click **Install Now**.

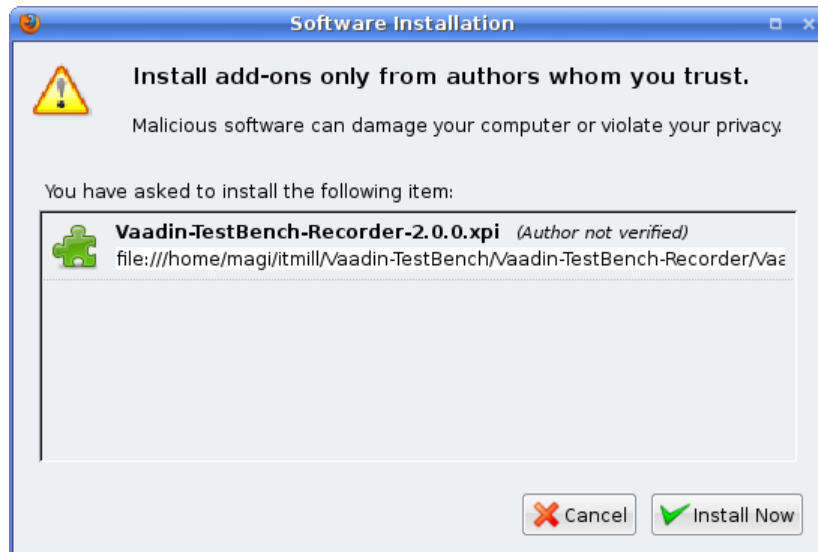


Figure 2: Installing Vaadin TestBench Recorder

5. After the installation of the add-on is finished, Firefox offers to restart. Click **Restart Now**.

The installation of a new version of Vaadin TestBench Recorder will overwrite a previous version.

After Firefox has restarted, navigate to a Vaadin application for which you want to record test cases, such as <http://demo.vaadin.com/colorpicker>.

2.4. Quick Setup for Playback on a Workstation

You can run the grid hub and a remote control on your workstation when developing tests. The configuration uses the local host as default, so you may not need to edit the configuration files at all.

Windows

1. Navigate to `grid/hub/` in the installation directory and run `hub.bat` to start the hub on port 4444.
2. Navigate to `grid/remote-control/` in the installation directory and run `rc.bat` to start the remote control on the local machine and have it connect to the hub.

Linux, Mac OS X, and other UNIX

1. Open a terminal window and change to `grid/hub/`
2. Run `hub.sh` to start the hub on port 4444.

```
$ cd vaadin-testbench-2.0.0/grid/hub
$ sh hub.sh
```

3. Open a second terminal window and change to `grid/node-controller/`

4. Run `rc.bat` to start the node controller on the local machine and have it connect to the hub.

```
$ cd vaadin-testbench-2.0.0/grid/remote-control
$ sh rc.sh
```

Check that the remote control registers correctly by opening <http://localhost:4444/console> in a web browser.

Next, change to the `example/` directory.

```
$ cd vaadin-testbench-2.0.0/example
```

Open the `build.xml` file in an editor and check that the list of target operating system – browser pairs matches your system and the installed browsers.

```
<property name="browsers"
          value="winxp-ie8,winxp-firefox35" />
```

The value must be a comma-separated list of target names. A complete list of predefined system—browser combinations is given in Section 2.5.1: *Configuring the Hub*.

If you think that the configuration is OK, run the script:

```
$ ant
```

The script will convert, compile, run the recorded test and give a brief output on test success or failure.

2.5. Setting Up a Grid Hub

The grid hub can be installed in the same server where the tests are run, in one of the grid nodes, or in a dedicated server. In a test development installation, you can install it on the same workstation or server with all the other components.

You can find the grid hub from the full installation package and from the `vaadin-testbench-grid` package.

2.5.1. Configuring the Hub

The hub is configured in a `grid_configuration.yml` file. You need to edit this file if the predefined list of targets does not cover all the targets, that is, operating system + browser combinations that you wish to test.

The first line contains tag `"hub:"`, after which should come the port definition. A hub uses port 4444 by default.. After the `"environments:"` tag comes a list of name-browser pairs that each define a target.

Parameter	Description
<code>name</code>	Corresponds to the Target value in a remote control in a

Parameter	Description
	node. The name can be defined as anything, but may not contain commas, which act as separator characters in the environment variable of the remote control configuration.
browser	Identifies a browser run by this target. The browser identifiers are prefixed with an asterisk.

The predefined targets are system-browser combinations mapped to a browser identifier. They are as follows:

Target Name	Browser
winxp-ie6	*iexplore
winxp-ie7	*iexplore
winxp-ie8	*iexplore
winxp-firefox35	*firefox
winxp-firefox36	*firefox
winxp-safari4	*safari
winxp-opera10	*opera
winxp-opera1010	*opera
winxp-opera1050	*opera
winxp-googlechrome4	*googlechrome
winxp-googlechrome4	*googlechrome
linux-firefox2	*firefox
linux-firefox3	*firefox
linux-opera10	*opera
osx-firefox35	*firefox
osx-safari4	*safari
osx-opera10	*opera

For example:

```

hub:
  port: 4444
  environments:
    - name: "winxp-ie7"
      browser: "*iexplore"
    - name: "winxp-firefox36"
      browser: "*firefox"
    - name: "winxp-opera1050"
      browser: "*opera"
    - name: "winxp-googlechrome41"
      browser: "*googlechrome"
    - name: "linux-firefox3"
      browser: "*firefox"

```

```
- name: "osx-firefox35"
  browser: "*firefox"
- name: "osx-safari4"
  browser: "*safari"
```

2.5.2. Starting the Hub

The grid hub is a service bound to a port, 4444 by default.

Windows

Navigate to `grid/hub/` in the installation directory and run `hub.bat` to start the hub on port 4444.

Closing the console window will stop the service.

Linux, Mac OS X, and other UNIX

1. Open a terminal window and change to `grid/hub/`
2. Run `hub.sh` to start the hub on port 4444.

```
$ cd vaadin-testbench-2.0.0/grid/hub
$ sh hub.sh
```

The hub service starts attached to the terminal and writes its log to standard output. Press **Ctrl+C** to stop the service.

2.6. Setting Up a Grid Node

A grid node is a server, a desktop computer, or a virtual machine running a windowed operating system. It needs to have:

- ➔ One or more web browsers installed
- ➔ Vaadin TestBench Grid Remote Control installed

A remote control is a service running in a grid node. It acts as a “remote control” for the web browsers installed in the node: it can launch and stop browser applications and run tests in them. A remote control is itself “remote controlled” by the grid hub, which delegates the tests to the available remote controls.

The installation of web browsers is not covered in this manual.

2.6.1. Configuring the Remote Control

A remote control must be configured before starting. The configuration is done by editing the `rc_configuration.xml` file. Values given in the XML configuration file override the default values defined in the run script.

Parameter	Description
port	The port this remote control listens to for commands from the hub.
hubURL	The address of the hub as a URL.

Parameter	Description
environment	An environment target (system-browser combination) that this remote control supports. The target must match one of the target names defined in the hub configuration. You can define multiple environments.
host	The host name or IP address of this node controller. If the host name is not defined, the hub determines it from the registration request.

2.6.2. Configuring the Run Script

A remote control can also be configured in the run scripts (`rc.sh` or `rc.bat`) by defining:

Environment Variable	Description
ENVIRONMENT	Environment targets (system-browser combinations) supported by this remote control. The targets in the comma-separated list must match one of the target names defined in the hub configuration.
USEREXTENSIONS	Where <code>user-extensions.js</code> is located.

In this case, you will also need to add the following parameters after `SelfRegisteringRemoteControlLauncher`.

Parameter	Description
hubUrl	Address of the hub.
port	The port that this node controller should listen to.

The `ENVIRONMENT` variable(s) should match one or more of the ones defined in the hub console.

2.6.3. Starting the Remote Control

A remote control is a service bound to a port, 5555 by default.

Windows

Navigate to `grid/remote-control/` in the installation directory and run `rc.bat` to start the hub on port 5555.

Closing the console window will stop the service.

Linux, Mac OS X, and other UNIX

1. Open a terminal window and change to `grid/remote-control/`
2. Run `rc.sh` to start the hub on port 5555.

```
$ cd vaadin-testbench-2.0.0/grid/remote-control
$ sh rc.sh
```

The remote control service starts attached to the terminal and writes its log to standard output. Press **Ctrl+C** to stop the service.

2.6.4. Running Tests Without a Grid Hub

Running test with only a standalone remote controller only requires changes to `browsers` property string in the ant script and that the `com.vaadin.testbench.testrunner.host` points to the remote control. No changes to tests or other settings are required.

To run tests with a standalone remote controller download Selenium RC from <http://seleniumhq.org/>

After unpacking copy `selenium-server.jar` from `selenium-server-1.x.x/`

Copy `user-extensions.js` from `grid/remote-control/` in the Vaadin TestBench package to where `selenium-server.jar` is located.

Start the Remote Control server with

```
java -jar selenium-server.jar -userExtensions user-extensions.js
```

In the Ant script in `example/` change the `browsers` property to use names defined in **Supported browser strings**.

(Note browser for *chrome is Firefox and not Google Chrome)

2.7. TODO:

Setting up a test node includes:

1. Install/confirm Java 1.5 JRE (or newer)
2. Install browsers to use on machine
3. Copy grid to test machine and edit remote-control run script
4. Set browser settings
5. Set OS settings

2.7.1. Edit remote control configuration file

After copying the grid package (or just the remote-control if another machine is running the hub) the configuration file needs to be edited.

If the hub is running on the same machine as the remote control then only the `<environment>` nodes need to be defined with proper targets for this remote controller as localhost.

For remote controllers running on other machines both the `<hubURL>` and `<environment>` nodes need to be checked and defined.

`<port>` - is the port this remote control listens to

<hubURL> - is the address to the hub

<environment> - defines one environment that this rc supports

2.7.2. Browser settings

Turn off pop-up blockers for all browsers.

[Internet Explorer - Tools → Pop-up Blocker → Turn Off Pop-up Blocker]

[Safari - Edit → Block Pop-up Windows]

Turn off default browser checks for all browsers.

2.7.3. Operating system settings

Windows:

Disable error reporting in case a browser crashes.

1. Open control panel → System
2. Select "Advanced" tab
3. Select "Error reporting"
4. Check that "Disable error reporting" is selected
5. Check that "But notify me when critical errors occur" is not selected

2.7.4. Notes for Using Screenshots

Disable the auto-hide function for the toolbar. (Windows)

Check that the toolbar is either locked or unlocked on all test machines.
(Windows)

Disable cursor blinking.

Windows: [Control panel → Keyboard → Cursor blink rate → none]

Turn on "Allow active content to run in files on My Computer" under Security settings on IE.

Use same resolution on all test machines and check that the maximized window is always the same size.

Configure browsers in the same manner on all machines (same toolbars visible etc.)

Turn off software that may suddenly popup a new window.

Disable cursor blinking for less false fails.

3. Using Vaadin TestBench Recorder

3.1. Overview

Tests are recorded using the Vaadin TestBench Recorder. You can play back recorded test cases and use the Recorder to make assertions and take screenshots for screen capture comparison.

The Recorder is available only for Mozilla Firefox. To run the recorded tests in other browsers, you need to compile them as JUnit tests and run them with JUnit, as described in Chapter 4: *Compiling and Executing JUnit Tests*. It also allows automating the testing.

3.2. Starting the Recorder

To start the Recorder:

1. Open the page with the application that you want to test.
2. Select **Tools → Vaadin TestBench Recorder** in Firefox.

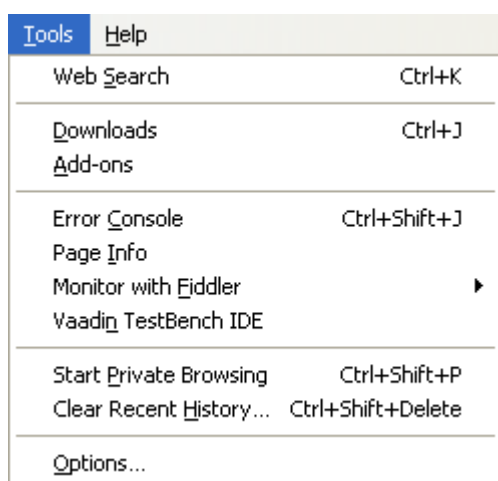


Figure 3: Starting Vaadin TestBench Recorder

The Vaadin TestBench Recorder window will open, as shown in Figure 4.

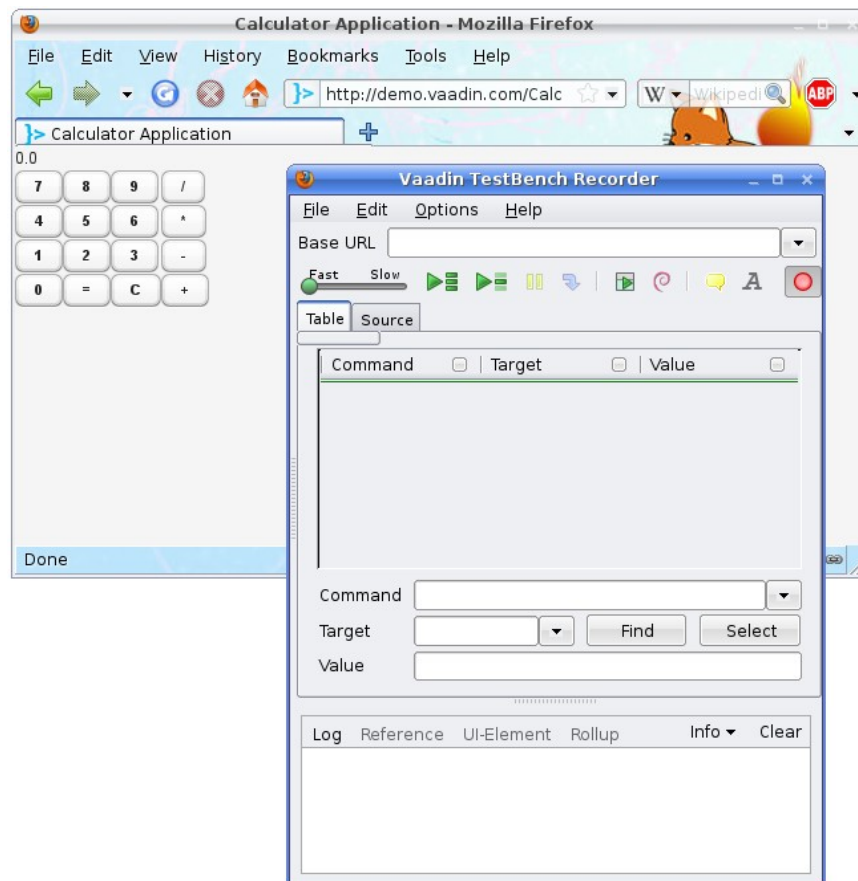



Figure 4: Vaadin TestBench Recorder running with the Calc demo

Recording will be automatically enabled. This is indicated by the pressed  **Record** button.

3.3. Recording

While recording, you can interact with the application in (almost) any way you like. The Recorder records the interaction as commands in a test script, which is shown in tabular format in the **Table** tab and as HTML source code in the **Source** tab.

Table	
Command	Target
open	/Calc
waitForVaadin	
click	vaadin=Calc::/VGridLayout[0...
waitForVaadin	
click	vaadin=Calc::/VGridLayout[0...
waitForVaadin	
click	vaadin=Calc::/VGridLayout[0...
waitForVaadin	

Figure 5: User interaction recorded as commands.


Please note the following:

- ➔ Changing browser tabs or opening a new browser window is not recommended, as any clicks and other actions will be recorded.
- ➔ Passwords are recorded in plain text as they do not differ from normal text input.

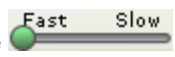
While recording, you can insert various commands such as assertions or take a screenshot by selecting the command from the **Command** list.

When you are finished, click the  **Record** button to stop recording.

3.4. Playing Back Tests

After you have stopped recording, reset the application to the initial state and press  **Play current test** to run the test again. You can use `&restartApplication` parameter for an application in the URL to restart it.

You can also play back saved tests by opening a target test in the Recorder with **File → Open**.

You can use the  slider to control the playback speed, click **Pause** to interrupt the execution and **Resume** to continue. While paused, you can click **Step** to execute the script step-by-step.

Check that the test works as intended and no unintended or invalid commands are found; a test should run without errors.

3.5. Editing Tests

You can insert various commands, such as assertions or taking a screenshot, in the test script during or after recording, by selecting the command from the **Command** list. The new command or comment will be added at the selected location, moving the selected location down.

Figure 6 shows adding an assertion after calculating “6*7=” with the Calc demo.

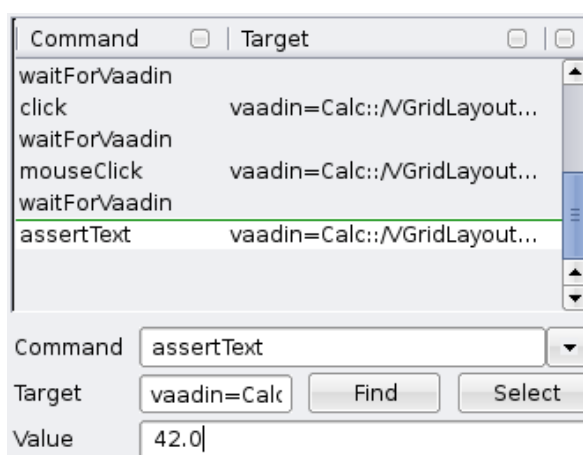


Figure 6: Inserting commands in a test script.

If the command requires a target element, click **Select** and then click an element in your application. A reference to the element is shown in the **Target** field and you can highlight the element by clicking **Find**. If the command expects some value, such as for comparing the element value, give it in the **Value** field.

You can insert the most important commands quickly by right-clicking an element in the application and selecting a command from the menu. This automatically selects the command and fills in the target and value parameters.

Commands in a test script can be changed by selecting a command and changing the command, target, or value.

3.6. Test Script Commands

Vaadin TestBench Recorder is based on the Selenium IDE, so the Selenium documentation provides a complete reference of all the commands available in the Selenium IDE and therefore in the Recorder.

Vaadin TestBench Recorder has the following special commands:

- `screenCapture`
- `showTooltip`
- `assertText`
- `includeTest`

These special commands are described next.

3.6.1. Comparing screen capture images: [`screenCapture`]

The `screenCapture` command orders the Remote Control to take a screen capture of the browser view and compare the result against a reference image, if one is available. If a reference image is not available, the command will save the screen capture and fail the test. You can then later copy the captured image as the reference image.


Observe that the screenshots are *not* taken by the Recorder, for example when you are playing back a test case in the Recorder. They are taken by the Remote Control only afterward, when you run the tests from the test server that controls the Remote Control (through the hub).

The **Value** field can be used to define an identifier string for the screenshot. If the value is empty, a running number starting from 1 will be used as the identifier. Using a given identifier is more reliable than the numbers if the test is changed and screenshots are added or removed.


The naming convention for screenshot file names is automated and has the following format:

```
NameOfTest_OperatingSystem_BrowserName_BrowserMajorNumber_Id
entifier.png
```

3.6.2. Recording ToolTips: [showTooltip]

Clicking  will enable recording of a tooltip. Recording is done by hovering over the target element until a tooltip appears and then moving the mouse away from the element. This will record a showTooltip command and disable the tooltip button.

3.6.3. Recording AssertText: [assertText]

Clicking  will enable recording of an assertText command on elements where context menu doesn't show. When assertText button is active the IDE will record an assertText on element command for the next mouse click. Assert stops bubbling of events for most Vaadin components.

3.6.4. Connecting tests together: [includeTest]

Vaadin TestBench has the capability to connect tests together. This is done by adding the command **includeTest** where Value is the path to the test to insert (from where the TestConverter is run or full path). Target test will be added in full at the position with **includeTest**. Inclusion only works when converting to JUnit tests with **com.vaadin.testbench.util.TestConverter**.

3.7. Saving Tests


You can save a test by selecting **File → Save Test**. If you are trying the Recorder, give `example/testscripts/` directory below the Vaadin TestBench installation directory as the target directory.

You can also save multiple tests as a test suite.

Vaadin TestBench stores the tests and test suites as HTML files. This makes it easy to review saved test scripts with a web browser and edit them manually.

3.8. Invalid Tests

Tests can become invalid due to intentional changes to the application. Normally it involves changes in elements so that the Vaadin TestBench element locator can't find the correct element. (refer to known problems)

If the problem is that an element can't be found press  to playback the test and find the problem position and check with the **Find** button that the element can't be found (eg. It's not a problem with timing that the element just isn't available yet). Re-selecting the element is simplest with the **Select** button that will update the target.

4. Compiling and Executing JUnit Tests

4.1. Overview

JUnit allows running tests remotely on a variety of different web browsers installed in grid nodes. The test scripts need to be first compiled from the HTML scripts saved with Recorder to Java classes. Vaadin TestBench Library includes a converter from the HTML format to Java source files, which you can then compile with a Java compiler.



The recommended way to compile and run JUnit tests is to use an Ant script, as described in Section 4.2: *Configuring the Ant Script*.

4.2. Configuring the Ant Script

The recommended way to compile and run JUnit tests is to use an Ant script. An example script is given in the `examples` directory in the installation package. It will convert, compile, and run the tests.

You can make the settings either by editing the Ant script or by giving the settings as property definitions from the command-line, for example:

```
$ ant -Dcom.vaadin.testbench.testrunner.host=localhost ...
```

4.2.1. Mandatory Settings

Define required system property values to the Java virtual machine:

`com.vaadin.testbench.testrunner.host`

Host name or IP address of the Vaadin TestBench Grid Hub. For example, "localhost". The port number should not be given in this parameter. If you have just a single test node and do not need the hub, you can give the address of the node.

`com.vaadin.testbench.deployment.url`

Base URL of the Vaadin application to be tested, for example, "http://demo.vaadin.com/".

`com.vaadin.testbench.screenshot.directory`

Base directory for screenshot reference and error images. The reference images are expected to be stored under the `reference` subdirectory. Error images are stored by JUnit under the `error` subdirectory. On the first

run, there are no reference images; you should copy “accepted” screenshots from the `error` directory to the `reference` directory.

`browsers`

A comma-separated list of target environments on which the tests should be run. The list entries must match targets defined in the configuration of remote controls in the grid nodes. They must also be listed in the configuration of the hub. The predefined browser names are system-browser pairs.

The default settings in the example test script are:

```
<!-- Host name or IP address of the host running -->
<!-- TestBench RemoteControl or TestBench Hub. -->
<property name="com.vaadin.testbench.testers.host"
  value="127.0.0.1" />

<!-- Base URL where the testable application is -->
<!-- deployed -->
<property name="com.vaadin.testbench.deployment.url"
  value="http://demo.vaadin.com/" />

<!-- Browsers to use for testing -->
<property name="browsers"
  value="winxp-ie8,winxp-firefox35" />

<!-- Base directory for screenshots. -->
<property name="com.vaadin.testbench.screenshot.directory"
  value="screenshots" />
```

4.2.2. Optional Settings

Optional property values that can be used:

`com.vaadin.testbench.screenshot.softfail`

If “true”, a test is allowed to continue even if a screenshot comparison fails.
If “false”, the test is interrupted and testing will continue with the next test.

`com.vaadin.testbench.screenshot.onfail`

If “true”, takes a screenshot when a test fails.

`com.vaadin.testbench.screenshot.cursor`

If “true”, makes a check if error is because of a cursor.

`com.vaadin.testbench.screenshot.block.error`

Sets the amount of difference that causes a screenshot comparison to fail. Comparison is done by 16×16 pixel blocks. The difference limit is given as a fraction ($0 < x \leq 1$) of how much a block can differ from the reference block. needed to cause a difference in screenshots to fail. how much different the 16x16 blocks may be. (default = 0.025 == 2.5%).

`com.vaadin.testbench.screenshot.reference.debug`

If “true” creates extra output for debugging purposes.

Tests can also be played back using the hub and running the test as a JUnit test.

Converting tests can be done in two ways.

One is to use `TestConverter` from the `vaadin-testbench` jar from the command line or with the `create-tests` target in the Ant script.

When compiling the JUnit tests you need to add the `Vaadin-TestBench@version@.jar` and `lib/junit-*.jar` into your build path. (These are also needed for running the test)

When running the JUnit test 3 system property values need to be specified.

```
-Dcom.vaadin.testbench.testrunner.host="localhost"
```

```
-Dcom.vaadin.testbench.testrunner.url="http://demo.vaadin.com/"
```

```
-Dcom.vaadin.testbench.testrunner.screenshot.directory="screenshot"
```

The given values are for the example test within the package, running it with Grid on the local machine.

Another is to export the test to a JUnit test from the IDE.

File → Export Test As → Vaadin Format – TestBench.

This will create a .java JUnit file out of the currently open test. (Some things need to be changed by hand in the .java file like screen shot names, test will use Firefox for the test and `includeTest` will not include the target test)

Note. Tests will not be able to run if no remote control has registered wanted environment.

4.3. Converting HTML Tests

The Test Converter in the Vaadin TestBench Library converts the HTML tests scripts to Java source files.

4.3.1. Command-Line Interface

```
$ java com.vaadin.testbench.util.TestConverter <OutputDir>
<Browsers> <HTMLTestFiles>
```

where:

Parameter	Description
<code>OutputDir</code>	Where the generated Java JUnit files should be saved.
<code>Browsers</code>	Target environments separated by a comma (environments need to be found in hub targets).
<code>HTMLTestFiles</code>	Target test files locations separated by spaces.

4.3.2. Ant Script

4.4. Compiling JUnit tests

Compile the java files created during the previous step.

5. Integrating Vaadin TestBench with a Build System

An example using Apache Ant can be found in the example directory. The example Ant script should be runnable if the hub and remote control have been started. The sample test will fail on the first run because there are no reference images.

5.1. Using Vaadin TestBench with a build system.

The build system needs to take the following steps after finishing the build to run tests.

1. Start server for testing (needs to be reachable by test machines)
2. Convert HTML test to Java Junit tests using TestConverter
3. Compile created Java Junit tests
4. Remove old error screens for screenshot directory
5. Run compiled Junit tests
6. Remove temporary files (source and compiled java files)

5.2. Starting the Server

(example assumes Jetty is used)

Create a WAR package of the application(s). Example of creating a war file using ant script:

```
<war destfile="example.war"
      webxml="./WebContent/WEB-INF/web.xml">
  <fileset dir="./src/possible_html_files">
  <lib dir="./thirdparty/libs">
    <exclude name="not_needed.jar"/>
  </lib>
  <classes dir="./build/" />
</war>
```

Copy created war file to \$JETTY_HOME/webapps/example.war

Start a server so that the program to be tested is accessible from the test machine(s).

```
$ java -jar start.jar
```


Application should now be deployed at http://BUILD_SYSTEM_ADDRESS:8080/example/ if example configuration has been used.

Note that the deployment url/ip and port may change, but context path needs to stay the same as the one received when recording tests. Else all locators will need to be edited.

5.3. Removing Old Error Screens

If there are any error screens from a previous run, remove these so that they don't mix with possible ones from this run.

5.4. Running Junit Tests

5.4.1. Remove temporary files

Clean away created files that are not needed anymore. (the .java and .class files)

5.4.2. Stop server

After tests have completed stop the server and remove application data from the webapps/ directory.

5.5. Notes for running tests

(Notes concern mostly people running tests on the local machine using Grid.)

When taking screenshots the mouse cursor will be moved to position 0,0 on screen to minimise problems with screenshot elements.

When using Opera, Safari or Google Chrome browsers, arrow key navigation is handled by the Java Robot that will send an actual key event to the system. This will not target the browser and target element, but the currently focused window and its selected element.

5.6. Handling of reference screenshots

For reference screenshots the best place to have them would be somewhere where both the user and build server have access or in the version control repository. This way users and the build server will have the latest references for their use. (note that both build server and users will need to have access to this location)

5.6.1. Updating references

Updating references on a users own machine only requires the user to copy over the correct/new screenshots from the [errors/](#) directory to the [references/](#) directory.

If using a repository, the images are copied the same way from [errors/](#) to [references/](#), but the new [references/](#) also needs to be committed to the repository.

5.7. Supported Browser Strings

Vaadin TestBench supports the following browser strings:

*firefoxproxy	*iehta
*firefox	*iexplore
*chrome	*opera
*firefoxchrome	*piiexplore
*firefox2	*pifirefox
*firefox3	*konqueror
*iexploreproxy	*mock
*safari	*googlechrome
*safariproxy	

In cases where multiple browsers are installed or the remote controller can't find the browser (not installed to a default location) the browser string can be extended with the absolute path to the browser executable. E.g. `*firefox /home/user/firefox3.5/firefox`

In these cases it should be noted that all remote controls using the target browser should have the same installation directory. (Target name used may be unique for this, but may cause mixups)

Note. Target `*firefox3` is defined as "Discovers a valid Firefox 2.x or 3.x installation on local system. Preference is given to 2.x installs". This will cause mix ups if both Firefox2 and Firefox3 are installed, as one would expect Firefox3 to be launched.

5.8. Limitations

Vaadin TestBench has currently the following limitations:

- Playback for modifier+arrow key only works in Firefox and InternetExplorer
- Tests with RichTextFields do not work
- Drag-and-drop functionality is not yet implemented. (Recording sliders is not functional)
- For LoginForm test will record a **selectFrame** when using LoginForm and **selectWindow** when going away from login. Remove these commands to get the test to work correctly.
- Screenshots don't always work as expected under Linux (OSX?) due to remote control resizing browser window wrong for `selenium.windowMaximize()`
- Theme changes don't record the required `AndWait`.
- URI fragment functionality is not reliable.
- Recording and playing back file upload is not supported.
- OSX firefox3.5 doesn't draw marking rectangle on element.

5.9. Jetty Configuration example

```
<?xml version="1.0"?>
<!DOCTYPE Configure PUBLIC "-//Mort Bay Consulting//DTD
Configure//EN" "http://jetty.mortbay.org/configure.dtd">
<Configure id="Server" class="org.mortbay.jetty.Server">
  <Call name="addConnector">
    <Arg>
      <New
class="org.mortbay.jetty.nio.SelectChannelConnector">
        <Set name="port">8080</Set>
      </New>
    </Arg>
  </Call>

  <Set name="handler">
    <New id="Handlers"
class="org.mortbay.jetty.handler.HandlerCollection">
      <Array type="org.mortbay.jetty.Handler">
        <Set name="handlers">
          <Item>
            <New id="Contexts"
class="org.mortbay.jetty.handler.ContextHandlerCollection"/>
          </Item>
        </Array>
      </Set>
    </New>
  </Set>

  <Call name="addLifeCycle">
    <Arg>
      <New
class="org.mortbay.jetty.deployer.WebAppDeployer">
        <Set name="contexts"><Ref id="Contexts"/></Set>
        <Set name="webAppDir"><SystemProperty
name="jetty.home" default="."/>/webapps</Set>
        <Set name="parentLoaderPriority">false</Set>
        <Set name="extract">true</Set>
        <Set name="allowDuplicates">false</Set>
      </New>
    </Arg>
  </Call>
</Configure>
```