

Content

- **How to install Testbench**
- **How to record test cases**
- **How to playback a test case**
- **How to setup test machines**
- **How to integrate Testbench with a build system**

Testbench parts.

- Firefox extension Testbench-ide-@build@.xpi
- vaadin-testbench-@version@.jar
- Testbench Grid (includes Hub and Remote Control)

Testbench requirements

Firefox 1.5-3.5.* (Testbench IDE)

Java SDK 1.5

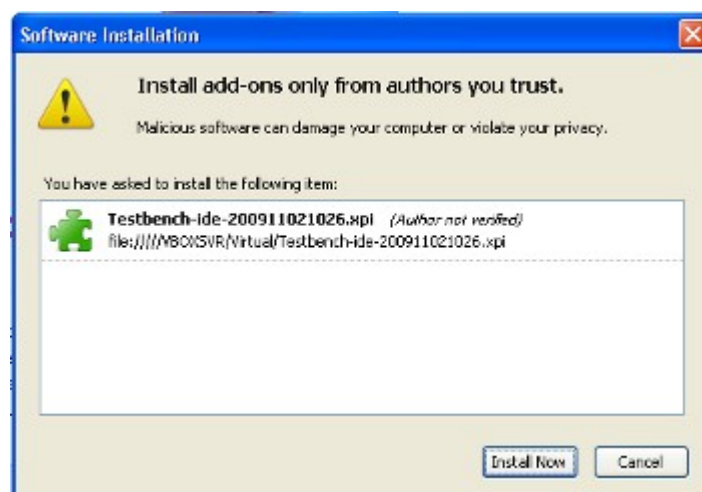
How to install Testbench

1. Extract files from vaadin-testbench-@version@.zip
2. Install Testbench IDE
3. Setup the grid
4. Setup test machines

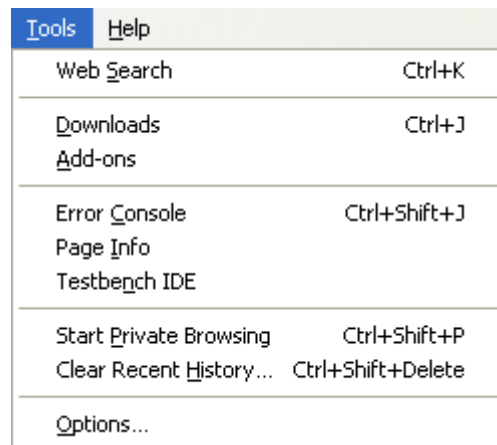
1. Extract vaadin-testbench-@version@.zip to a directory of your choice.

2. Install Testbench IDE

The Testbench IDE is a Firefox extension that is built on the Selenium IDE extension and will therefore overwrite an installed Selenium IDE extension and any old Testbench IDE.



After installation and restart the extension Testbench IDE will be found under the Tools menu. The installed Testbench IDE version can be checked from Extensions in the Add-ons menu.



3. Setup the grid

The grid consists of 2 parts both of which are modifications of Selenium Grid 1.0.4.

1. Hub (Handles communication between remote controls and test application)
2. Remote-Control (Actual execution of test commands)

The hub and remote control are setup so that they can be run on the local machine just by first starting the hub with `/grid/hub/hub.bat` and then the remote control with `/grid/remote-control/rc.bat` (hub and rc under linux).

After the hub has been started a console can be found on <http://localhost:4444/console> and after starting the remote control Available Remote Controls should be populated.

TestBench - Selenium Grid Hub

[Documentation](#) | [FAQ](#)

Configured Environments

Target	Browser
winxp-ie7	*iexplore
winxp-ie8	*iexplore
winxp-ie6	*iexplore
osx-firefox35	*firefox
linux-firefox2	*firefox
linux-firefox3	*firefox
linux-opera10	*opera
winxp-safari4	*safari
winxp-firefox35	*firefox
osx-opera10	*opera
winxp-opera10	*opera
osx-safari4	*safari
winxp-googlechrome3	*googlechrome

Available Remote Controls

Host	Port	Environment
127.0.0.1	5555	winxp-ie8
127.0.0.1	5555	winxp-firefox35
127.0.0.1	5555	winxp-safari4
127.0.0.1	5555	winxp-opera10
127.0.0.1	5555	winxp-googlechrome3

Active Remote Controls

Host	Port	Environment
------	------	-------------

Configuring the hub

For the Hub port and environments can be easily defined in [grid_configuration.yml](#)

- name: "" represents the Target value in the console and
- browser: "" is the browser run by this target.

The name can be defined as anything, but may not contain a [,] as the remote control environment string is parsed by [,].

The supported browser strings are defined at the end of this document.

Configuring the remote control

The remote control is configured in the script (rc or rc.bat).

- HOST - should be the address of this remote control
- HUBURL - is the address to the hub
- ENVIRONMENT - defines environments/browsers supported by this remote control
- PORT - is the port that this remote control listens to
- USEREXTENSIONS - defines where user-extensions.js is located

The environment variable should match one or more of the ones defined in the hub console.


4. Setting up test machines

Setup of the test machines require installation of programs (Java and browsers), setting of environment and copying of grid to test machine. This part is processed in more detail under **How to setup test machines**.

How to record test cases

Test cases are recorded using the Testbench IDE.

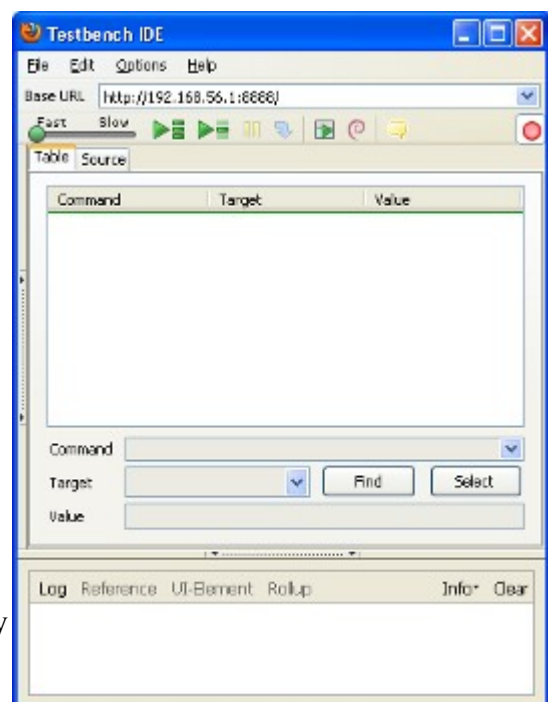
Open the page for which you want to make the test and launch the IDE from the Tools menu.

Recording will be automatically enabled when the program launches. This is marked by the  button.

Recording:

Testbench IDE will record all clicks made inside <div> elements and also such events as selections, key navigation and character input.

A mouseClicked is also recorded automatically when a notification opens so that the notification will close during the test run (as it's not always possible to actually click on a notification to record an event).



Arrow keys with possible modifying keys (shift, alt, ctrl) are also recorded as they happen. (note that at the moment playback for modifier+arrow key only work in Firefox and InternetExplorer)
Other automatic event recordings include scrolling and Menubar navigation.

ImageComparison:


In the right click menu, option screenCapture will record an event that will take a screenshot of the desktop, crop out the canvas and compare the result to a reference image if one is available. If no referense image is available it will save this screenshot and fail the test at the end.
(note that screenshots are not taken and compared in the IDE, only when using a remote control)

For a screenCapture the Value field can be used to define a identifier string for the screenshot. If left empty the identifier will be a running number starting from 1 for each test.

The naming convention of screenshots is automated and has the following format.

`NameOfTest_OperatingSystem_BrowserName_BrowserMajorNumber_Identifier.png`

Recording ToolTips:

Selecting  will enable recording of a tooltip by hovering until a tooltip appears and then moving the mouse away from the element. This will record a showTooltip command and disable the tooltip button.


Test editing functions:

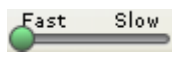
The Find button next to Target: will highlight the target on the page where as the Select button will allow changing the target of selected command to the element marked by the next mouse click made. Select button will stop bubbling of events for Vaadin components so selection of new target will not cause an actual click for most Vaadin components.

Testbench has the capability to connect tests together. This is done by adding a command appendToTest where the Value is the path to the test to insert. Target test will be added in full at this position and the rest of the appending test is then added after.

How to playback a test case

From the IDE

Playing back a test in the IDE works by pressing the  button that will disable recording and run the currently open/selected test.

The command execution speed on playback can be controlled with the  slider.
Slowing down the execution speed on IDE playback is helpful when confirming the correctness of a test sequence.

As a JUnit test

Tests can also be played back using the hub and running the test as a JUnit test. One way to create a JUnit test is to export the test to vaadin format. From the IDE select:

File → Export Test Case 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 and will use Firefox for the test)

Another way to create .java JUnit tests is by using `TestConverter` from the `vaadin-testbench` jar

com.vaadin.testbench.util.TestConverter [OUTPUTDIR BROWSERS HTMLTestFiles]

OutputDir defines where the generated Java JUnit files should be saved

Browsers defines target environments separated by a comma
(environments need to be found in hub targets)

HTMLTestFiles target test files locations separated by a space

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 case 3 system property values need to be specified.

```
-Dcom.vaadin.testbench.testrunner.host="localhost"
-Dcom.vaadin.testbench.deployment.url="http://demo.vaadin.com/"
-Dcom.vaadin.testbench.screenshot.directory="screenshot"
```

Values given are for the example within the package using the grid on local machine.

A simpler way is to use the Ant Script in the examples directory. This will convert, compile and run the test(s).

Note. Browser tests will not be able to run if no remote control has registered target environment.

How to setup test machines

Setting up a test machine consists of

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

3. Edit remote-control run script

After copying the grid package (or just the remote-control if another machine is running the hub) the run script needs to be changed.

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

For remote controllers running on other machines the `HOST`, `HUBURL` and `ENVIRONMENT` variables need to be defined.

`HOST` - The address of this remote control
`HUBURL` - The address to the hub
`ENVIRONMENT` - Environments/browsers supported by this remote control

4. Browser settings

Turn off popup 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.

5. OS 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

Disable the auto-hide function for the toolbar.

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

Disable cursor blinking [Control panel → Keyboard → Cursor blink rate → none]

Notes for using screenshots

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

Configure the browser in the same manner (same toolbars visible etc.)

Turn off software that may suddenly popup a new window.

Disable cursor blinking for less false fails.

How to integrate Testbench with a build system

Using 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)

Start server.

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

Convert HTML tests

Run `com.vaadin.testbench.util.TestConverter [OUTPUTDIR BROWSERS HTMLTestFiles]`

OutputDir	defines where the generated Java JUnit files should be saved
Browsers	defines target environments separated by a comma
HTMLTestFiles	target test files locations separated by a space

Compile Junit tests

Compile the java files created during the previous step.

Remove old error screens

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

Run Junit tests

Define required system property values to the java virtual machine.

- Dcom.vaadin.testbench.testers.host
= Hub address without the port definition eg. "localhost"
- Dcom.vaadin.testbench.deployment.url
= Base url of test application eg. "<http://demo.vaadin.com/>"
- Dcom.vaadin.testbench.screenshot.directory
= Base directory for reference and error images

Optional system property values that can be used are.

- Dcom.vaadin.testbench.screenshot.softfail
= if "true" lets the test run to finish even if errors are found for screenshots
- Dcom.vaadin.testbench.screenshot.reference.debug
= if "true" creates extra output for debugging purposes

Remove temporary files

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

Examples

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.

Supported Browser strings

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