

TestBench 2.0
User manual

Index

Definitions	p. 3
Overview	p. 4
What is TestBench	p. 4
Typical TestBench setup	p. 4
TestBench parts	p. 4
TestBench requirements	p. 4
How to install TestBench	p. 5
Simple setup for recording	p. 5
Simple setup for playback on local machine	p. 6
Setting up the grid	p. 6
Configuring the hub	p. 6
Configuring the remote control	p. 7
How to record test cases	p. 7
Recording	p. 7
Special commands	p. 8
Comparing screenshot images	p. 8
Recording ToolTips	p. 8
Connecting tests together	p. 8
IDE buttons	p. 8
Editing test cases	p. 9
Faulty test cases	p. 9
How to playback a test case	p. 9
From the IDE	p. 9
As JUnit test	p. 9
How to setup test machines	p. 10
Edit remote control run script	p. 10
Browser settings	p. 11
Operating system settings	p. 11
Windows	p. 11
Notes for using screenshots	p. 11
How to Integrate Tesbench with a build system	p. 11
Using TestBench with a build system	p. 11
Start server	p. 12
Convert HTML tests	p. 12
Compile JUnit tests	p. 12
Remove old error screens	p. 12
Run JUnit tests	p. 12
Remove temporary files	p. 13
Stop server	p. 13
Notes for running tests	p. 13
Handling of reference screenshots	p. 13
Updating references	p. 13
Supported browser strings	p. 14
Running tests with only a Remote Control	p. 14
Known issues	p. 15
Jetty Configuration example	p. 16

Definitions

@build@

Build identifier of format YYYYMMDDHHmm

@version@

Version identifier with version and build information eg. 2.0.0.development.@build@

Console

Console is used for the Hub Console that shows information about the Hub and Remote Controller statuses. The console is found at `http://{HUB_ADDR}:4444/console`

Grid

Grid consists of a Hub and one or more Remote Controls.

Hub

The hub controls Remote Control resource allocation for tests.

IDE

See. TestBench IDE

Remote Control

Remote Control executes the test commands and controls browsers during tests.

TestBench IDE

The TestBench IDE is the Firefox extension used for recording and editing test cases.

Overview

What is TestBench

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

TestBench enables 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.

Typical TestBench setup

The typical setup for a TestBench environment consists of

1. Build server used to build, launch and test the application.
2. Machine with Firefox + TestBench IDE for creating tests.
3. One machine running Grid Hub.
4. One or more machines running Grid Remote Controls.

Machines for Hub, Remote Controls and test creation may be the same machine.

TestBench parts.

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

TestBench requirements

For recording and playback using IDE

- Firefox 3 or newer

For running tests

- Java JDK 1.5 or newer
- Installed browsers
- Apache Ant or a way to run Ant script is recommended

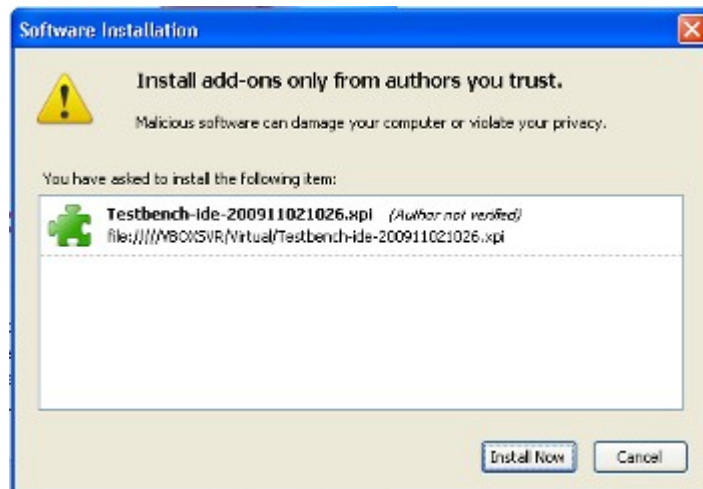
How to install TestBench

Simple setup for recording

After extracting the files from vaadin-testbench-@version@.zip, enter the [testbench-ide/](#) directory.

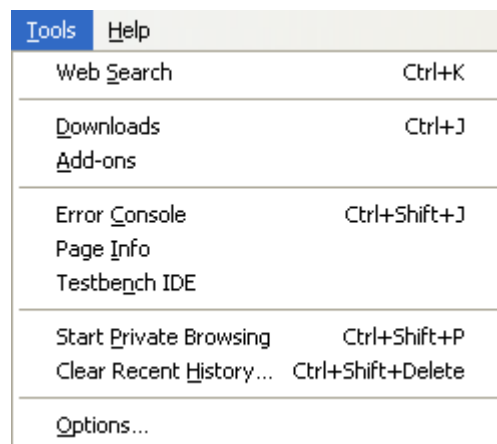
Open Firefox and either drag the Testbench-ide-@build@.xpi to an open Firefox window or open it from the [File] menu. This will ask if you want to install the TestBench IDE extension.

Note that because TestBench IDE is built on the Selenium IDE extension installation will overwrite an installed Selenium IDE extension and any old TestBench IDE.



After Firefox has restarted navigate to a Vaadin application eg. <http://demo.vaadin.com/colorpicker>

Open [Testbench IDE] from the [Tools] menu and record a small test.



Open an empty tab or window and press ► to see that the test works.

Save the test as a .html file with [Save Test Case] from the [File] menu to [example/testscripts/](#) (TestBench uses html notation for the test files)

Simple setup for playback on local machine

(see also Running tests with only a Remote Control p.14)

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.sh and rc.sh under Linux).

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

Then go to `grid/remote-control/` and run rc.bat to start the remote control on the local machine and have it connect to the hub.

Check that the remote control registered correctly by opening <http://localhost:4444/console>

Testbench Grid Hub

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
------	------	-------------

Next run the ant script in the `example/` directory. The script will convert, compile, run the recorded test and give a brief output on test success or failure.

Setting up the grid

The grid consists of 2 parts.

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

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 values are defined in the file `rc_configuration.xml`. Values given in the xml will override the values given in the run scripts.

- <port> - is the port this remote control listens to
- <hubURL> - is the address to the hub
- <environment> - defines one environment that this rc supports

The remote control can also be configured from the run scripts (rc.sh or rc.bat) by defining

- ENVIRONMENT - defines environments/browsers supported by this remote control
- USEREXTENSIONS - defines where user-extensions.js is located

and adding following after `SelfRegisteringRemoteControlLauncher`


- hubUrl [hubUrl] - is the address to the hub
- port [PORT] - is the port that this remote control listens to

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

How to record test cases

Test cases are recorded using the TestBench IDE.

Open the page with the application that you want to test and launch the IDE from the [Tools] menu.

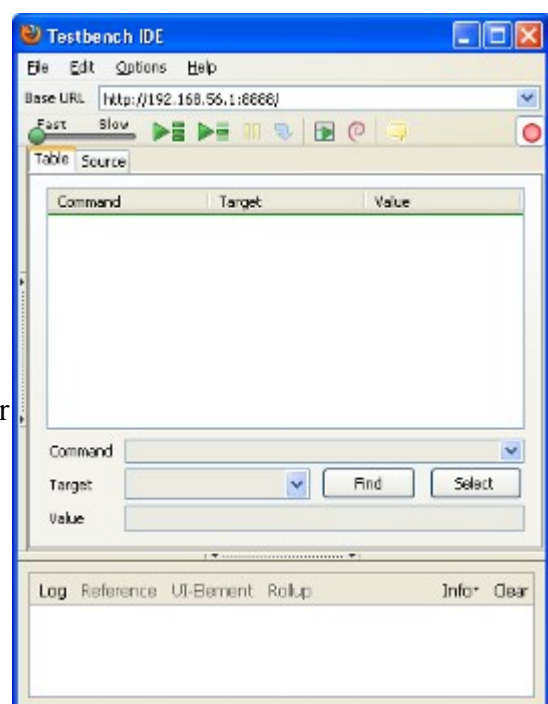
Recording will be automatically enabled when the program launches. This is marked by the  button.


Recording:

TestBench will record appropriate commands as the user navigates the application.

(note. Passwords are recorded in plain text as they do not differ from normal text input)

Changing tabs or open windows is not recommended while recording is enabled, as any clicks made will be recorded.



After the test has been recorded, it is advisable to playback the test using  to see that it works as planned and no faulty/wrong commands are found (test runs without errors).

When the test works correctly it should be saved as a .html file with [Save Test Case] from the [File] menu. HTML file format is used because the IDE saves the test case in XHTML format. (Also the default files that are searched when running tests as JUnit tests with the ant script is *.html)

Special commands

Comparing screenshot images: [screenCapture]


From the right click menu, **screenCapture** will record an event that takes a screenshot of the desktop, crops out the canvas and compares the result against a reference image (if one is available). If no reference image is available it will save this screenshot and fail the test at the end. (**note** that screenshots are not taken and compared with the IDE, only when using a remote control)

For the **screenCapture** command 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. Using an ID for the screenshot is more reliable than the numbers if the test changes and screenshots are added or removed.

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

NameOfTest_OperatingSystem_BrowserName_BrowserMajorNumber_Identifier.png

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.

Connecting tests together: [includeTest]

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`.

IDE buttons:

The **Find** button will highlight the target element on the page where as the **Select** button will allow changing the target for the currently selected command. **Select** will change the target according to the next element that is clicked. **Select** will stop bubbling of events for most Vaadin components, so selection of new target will not cause an actual click for most Vaadin components.

Editing test cases

To edit a test case open the test case in the IDE.


Command fields can be changed by selecting target command and then changing the wanted field. (Command, Target or Value)

New commands can be inserted by right clicking and selecting [Insert New Command] and comments with [Insert New Comment].

Note that new command and new comment will be added on the place that is selected and move the selected command down.

Faulty test cases:


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

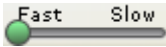
If the problem is that an element can't be found press  to playback the test and find the problem position and check with the **Find** button that the element can't be found (eg. It's not a problem with timing that the element just isn't available yet). Re-selecting the element is simplest with the **Select**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

How to playback a test case

From the IDE

Open target test in the IDE with [Open...] from the [E]file menu (if not testing an already open test).

Push the  button to playback the test. Playback 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

The recommended way to run JUnit tests is to edit/use the Ant Script in the examples directory. This will convert, compile and run the test(s). **Note** the property values defined in script.

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.

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.testster.host="localhost"

-Dcom.vaadin.testbench.deployment.url="<http://demo.vaadin.com/>"

-Dcom.vaadin.testbench.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 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, 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.

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

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

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.

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

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.

How to integrate 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.

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. (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 recieved when recording tests. Else all locators will need to be edited.

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 prevoius 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.

- Dcom.vaadin.testbench.screenshot.softfail
= if "true" lets the test run to finish even if errors are found for screenshots
- Dcom.vaadin.testbench.screenshot.onfail
= if "true" takes a screenshot when a test fails.
- Dcom.vaadin.testbench.screenshot.cursor
= if "true" makes a check if error is because of a cursor.
- Dcom.vaadin.testbench.screenshot.block.error
= sets how much different the 16x16 blocks may be $0 < x \leq 1$. (default = 0.025 == 2.5%)
- 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)

Stop server

After tests have completed stop the server and remove application data from the [webapps/](#) directory.

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.

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)

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.

Supported browser strings

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

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.

Running tests with only a Remote Control

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.testers.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 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)

Known issues

~~waitForVaadin doesn't always wait on the first command if the command is run before testbench hooks to the application are set on load. [This should be fixed now]~~

At the moment playback for modifier+arrow key only works in Firefox and InternetExplorer.

Tests with RichTextFields do not work.

Drag-n-drop functionality is not yet implemented.

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.

~~Testing of datefield popup in IE is erratic. This is due to execution of mousedown event halting for awhile sometimes. [Fixed since 200912100700]~~

Screenshots don't always work as expected under Linux due to remote control resizing browser window wrong for selenium.windowMaximize()

TestBench IDE will fail to record events if TestBench user- and ide-extensions are set under options.

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>
```