

---

# Javascript Unittesting with js-test-driver

Jakob Westhoff <jakob@php.net>

PHP Usergroup Köln/Bonn  
November 6, 2009

# About Me

---

- Jakob Westhoff
  - PHP developer for more than 6 years
  - Computer science student at the TU Dortmund
  - Co-Founder of the PHP Usergroup Dortmund
  - Active in different Open Source projects

# Who is actively using TDD/Unittesting?

---

- Who uses unittesting?

# Who is actively using TDD/Unittesting?

---

- Who uses unittesting?
- Who uses TDD?

# Who is actively using TDD/Unittesting?

---

- Who uses unittesting?
- Who uses TDD?
- Who unittests **javascript** code?

# Goals of this session

---

- 1 Getting to know `js-test-driver`

# Goals of this session

---

- 1 Getting to know js-test-driver
- 2 Unittesting javascript

# Goals of this session

---

- 1 Getting to know js-test-driver
- 2 Unittesting javascript
  - Synchronous tests

# Goals of this session

---

- 1 Getting to know js-test-driver
- 2 Unittesting javascript
  - Synchronous tests
  - Asynchronous tests

# Goals of this session

---

- 1 Getting to know js-test-driver
- 2 Unittesting javascript
  - Synchronous tests
  - Asynchronous tests
- 3 Run these tests

# Goals of this session

---

- 1 Getting to know js-test-driver
- 2 Unittesting javascript
  - Synchronous tests
  - Asynchronous tests
- 3 Run these tests
  - From within a terminal

# Goals of this session

---

- 1 Getting to know js-test-driver
- 2 Unittesting javascript
  - Synchronous tests
  - Asynchronous tests
- 3 Run these tests
  - From within a terminal
  - From within Eclipse

# Goals of this session

---

- 1 Getting to know js-test-driver
- 2 Unittesting javascript
  - Synchronous tests
  - Asynchronous tests
- 3 Run these tests
  - From within a terminal
  - From within Eclipse
- 4 Create javascript code coverage reports

# What is js-test-driver

---

- Javascript testrunner and framework

# What is js-test-driver

---

- Javascript testrunner and framework
- Client/Server approach

# What is js-test-driver

---

- Javascript testrunner and framework
- Client/Server approach
  - Server: js-test-driver

# What is js-test-driver

---

- Javascript testrunner and framework
- Client/Server approach
  - Server: js-test-driver
  - Client: Browser

# What is js-test-driver

---

- Javascript testrunner and framework
- Client/Server approach
  - Server: js-test-driver
  - Client: Browser
  - Client: Testrunner

# What is js-test-driver

---

- Javascript testrunner and framework
- Client/Server approach
  - Server: js-test-driver
  - Client: Browser
  - Client: Testrunner
- Assertion framework

# What is js-test-driver

---

- Javascript testrunner and framework
- Client/Server approach
  - Server: js-test-driver
  - Client: Browser
  - Client: Testrunner
- Assertion framework
  - Mostly xUnit compatible

# What is js-test-driver

- Javascript testrunner and framework
- Client/Server approach
  - Server: js-test-driver
  - Client: Browser
  - Client: Testrunner
- Assertion framework
  - Mostly xUnit compatible
  - `assertEquals`
  - `assertSame`
  - `assertTrue`
  - ...

# What comes next?

---

## Architecture

## js-test-driver Server



# Architecture of js-test-driver

js-test-driver  
Server



Browser



# Architecture of js-test-driver

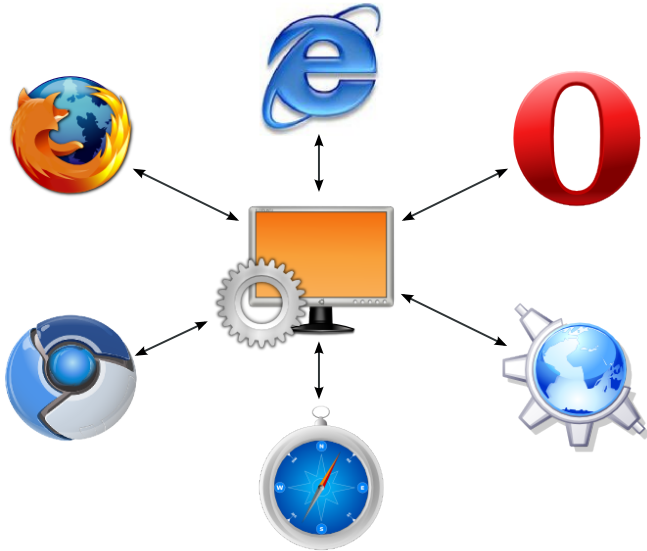
js-test-driver  
Server



Browser



# Architecture of js-test-driver



# Architecture of js-test-driver

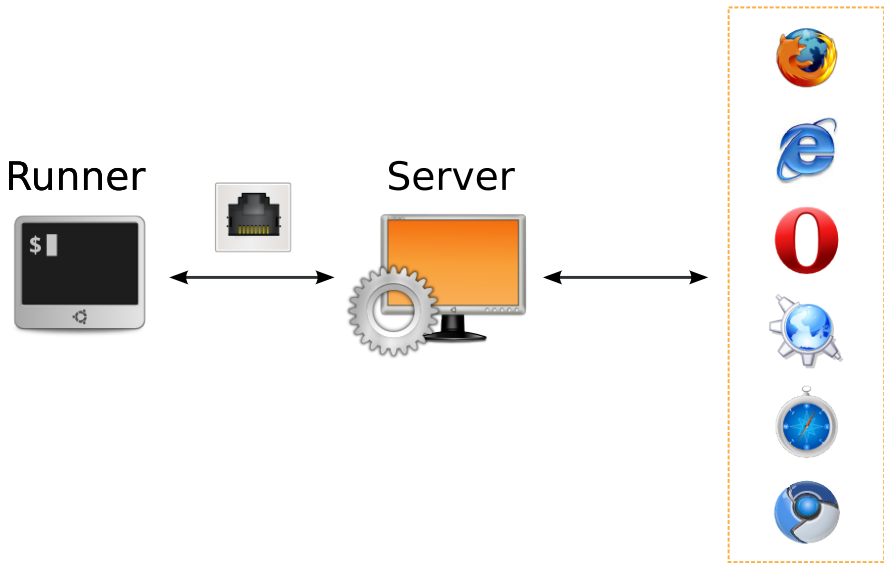
Runner



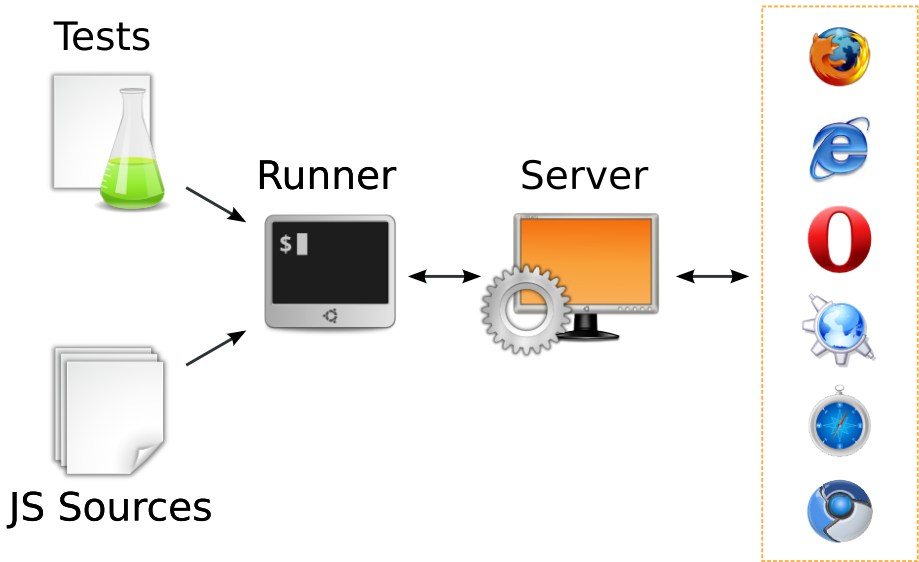
Server



# Architecture of js-test-driver



# Architecture of js-test-driver



# Advantages of this design

---

- Run from possibly any IDE

# Advantages of this design

---

- Run from possibly any IDE
  - Commandline

# Advantages of this design

---

- Run from possibly any IDE
  - Commandline
  - Eclipse

# Advantages of this design

---

- Run from possibly any IDE
  - Commandline
  - Eclipse
- **Parallel** execution of tests in **multiple** browsers

# Advantages of this design

---

- Run from possibly any IDE
  - Commandline
  - Eclipse
- **Parallel** execution of tests in **multiple** browsers
- Test execution on **different machines** with possibly different **operating systems**

# Advantages of this design

---

- Run from possibly any IDE
  - Commandline
  - Eclipse
- **Parallel** execution of tests in **multiple** browsers
- Test execution on **different machines** with possibly different **operating systems**
- Website DOM not used for any testrunner output

# Advantages of this design

---

- Run from possibly any IDE
  - Commandline
  - Eclipse
- **Parallel** execution of tests in **multiple** browsers
- Test execution on **different machines** with possibly different **operating systems**
- Website DOM not used for any testrunner output
- **Browser independent code coverage reports** (planned for a future release)

# What comes next?

---

## Example

# A simple example

---

- Javascript application/function

# A simple example

---

- Javascript application/function
- Testcase

# A simple example

---

- Javascript application/function
- Testcase
- js-test-driver configuration file

# A simple example - Javascript

- Really simple `hello_world` function:

```
1 myApp = {};  
2  
3 myApp.HelloWorld = function () {};  
4  
5 myApp.HelloWorld.prototype.sayHello = function () {  
6     return "Hello World!";  
7 }
```

# A simple example - Testcase

- Our first js-test-driver testcase:

```
1 HelloWorldTest = TestCase( 'HelloWorld' );
2
3 HelloWorldTest.prototype.setUp = function() {
4     this.helloFixture = new myApp.HelloWorld();
5 }
6
7 HelloWorldTest.prototype.testOutput = function() {
8     assertEquals(
9         "Hello World!",
10        this.helloFixture.sayHello()
11    );
12 }
```

# Digression about test lifecycle

---

The js-test-driver test lifecycle follows the JUnit lifecycle

# Digression about test lifecycle

---

The js-test-driver test lifecycle follows the JUnit lifecycle

- 1 Instantiate new `Testcase` implementation

# Digression about test lifecycle

---

The js-test-driver test lifecycle follows the JUnit lifecycle

- 1 Instantiate new `Testcase` implementation
- 2 Execute the `setUp()` method

# Digression about test lifecycle

---

The js-test-driver test lifecycle follows the JUnit lifecycle

- 1 Instantiate new `Testcase` implementation
- 2 Execute the `setUp()` method
- 3 Execute the next `testWhatever()` method

# Digression about test lifecycle

---

The js-test-driver test lifecycle follows the JUnit lifecycle

- 1 Instantiate new `Testcase` implementation
- 2 Execute the `setUp()` method
- 3 Execute the next `testWhatever()` method
- 4 Execute the `tearDown()` method

# Digression about test lifecycle

The js-test-driver test lifecycle follows the JUnit lifecycle

- 1 Instantiate new `Testcase` implementation
- 2 Execute the `setUp()` method
- 3 Execute the next `testWhatever()` method
- 4 Execute the `tearDown()` method
- 5 While tests left in testcase jump to Step 1

# A simple example - js-test-driver configuration

---

- Every project needs a js-test-driver configuration file.

# A simple example - js-test-driver configuration

---

- Every project needs a js-test-driver configuration file.
  - Written in YAML

# A simple example - js-test-driver configuration

---

- Every project needs a js-test-driver configuration file.
  - Written in YAML
  - Called `jsTestDriver.conf` by default

# A simple example - js-test-driver configuration

- Every project needs a js-test-driver configuration file.
  - Written in YAML
  - Called `jsTestDriver.conf` by default
- js-test-driver configuration for our example:

```
1 server: http://localhost:4224
2
3 load:
4   - src/hello_world.js
5   - tests/hello_world.js
```

# Run js-test-driver - Server

---

- Run the js-test-driver server

```
java -jar jsTestDriver.jar --port 4224
```

# Run js-test-driver - Browser

---

- Run the js-test-driver server

```
java -jar jsTestDriver.jar --port 4224
```

- Capture the all browsers

```
http://localhost:4224/capture
```

# Run js-test-driver - Testrunner

- Run the js-test-driver server

```
java -jar jsTestDriver.jar --port 4224
```

- Capture the all browsers

```
http://localhost:4224/capture
```

- Execute the testrunner

```
java -jar jsTestDriver.jar --tests all
```

# A simple example - Live Demo

---

Hello World example

Live Demo!

# Eclipse testrunner

---

- A **testrunner plugin for Eclipse** is available

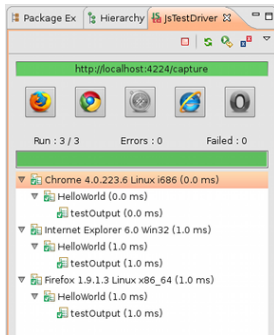
# Eclipse testrunner

---

- A **testrunner plugin for Eclipse** is available
- Available through eclipse plugin manager

# Eclipse testrunner

- A **testrunner plugin for Eclipse** is available
- Available through eclipse plugin manager



<http://code.google.com/p/js-test-driver/wiki/UsingTheEclipsePlugin>

# What comes next?

---

## Asynchronous tests

# Test asynchronous implementations

---

- Asynchronous **timeouts**
  - `setTimeout`, `setInterval`

# Test asynchronous implementations

---

- Asynchronous **timeouts**
  - `setTimeout`, `setInterval`
- Asynchronous **callbacks**
  - `XMLHttpRequest`, `onReady`

# Test asynchronous implementations

- Asynchronous **timeouts**
  - `setTimeout`, `setInterval`
- Asynchronous **callbacks**
  - `XMLHttpRequest`, `onReady`

"[...] JS test driver does not support asynchronous test. This was a conscious decision, as we want to make sure that the tests remain fast."

– Misko Hevery on Google groups

# Asynchronous timeouts

---

```
window.setTimeout(  
  function () { ... },  
  3000  
);
```

# Asynchronous timeouts

---

```
window.setTimeout(  
  function() { ... },  
  3000  
);
```

- Use `jsUnitMockTimeout.js`

# Asynchronous timeouts

---

```
window.setTimeout(  
  function() { ... },  
  3000  
);
```

- Use `jsUnitMockTimeout.js`
  - <http://jsunit.net>

# Asynchronous timeouts

```
window.setTimeout(  
  function() { ... },  
  3000  
);
```

- Use `jsUnitMockTimeout.js`
  - <http://jsunit.net>
- Mocks `setTimeout`, `setInterval`, `clearTimeout` and `clearInterval`

# Asynchronous timeouts

```
window.setTimeout(  
  function() { ... },  
  3000  
);
```

- Use `jsUnitMockTimeout.js`
  - <http://jsunit.net>
- Mocks `setTimeout`, `setInterval`, `clearTimeout` and `clearInterval`
- **Timeflow** can be controlled **manually** in tests

# jsUnitMockTimeout example - Stopwatch

- Simple stopwatch example:

```
1 Stopwatch.start = function () {
2     Stopwatch.clear();
3     Stopwatch.timer = setInterval(
4         Stopwatch.advance,
5         1000
6     );
7 }
8
9 Stopwatch.advance = function () {...}
10 Stopwatch.stop    = function () {...}
11 Stopwatch.clear   = function () {...}
12 Stopwatch.result  = function () {...}
```

# jsUnitMockTimeout example - Test

- Test using jsUnitMockTimeout:

```
1 StopwatchTest.prototype.testTiming = function () {
2     Clock.reset();
3     Stopwatch.start();
4     assertEquals(0, Stopwatch.result());
5
6     Clock.tick(5000);
7     assertEquals(5, Stopwatch.result());
8 }
```

# Asynchronous callbacks

---

- Generalized callback mocks do not exist

# Asynchronous callbacks

---

- Generalized callback mocks do not exist
- Mock the used function manually, calling the callback immediately

# Asynchronous callbacks

- Generalized callback mocks do not exist
- Mock the used function manually, calling the callback immediately

```
1 xhr.send = function(data) {
2     // Call onReady with appropriate results
      directly
3     this.onReady(someData);
4 }
```

# What comes next?

---

## Debugging

# Debugging

---

- Use whatever javascript debugging technique you like!

# Debugging

---

- Use whatever javascript debugging technique you like!
  - Firebug (Firefox)
  - Visual Studio (IE)
  - Web Inspector (Safari)
  - ...

# Debugging

---

- Use whatever javascript debugging technique you like!
  - Firebug (Firefox)
  - Visual Studio (IE)
  - Web Inspector (Safari)
  - ...

**1** Run the test you want to debug:

# Debugging

- Use whatever javascript debugging technique you like!
  - Firebug (Firefox)
  - Visual Studio (IE)
  - Web Inspector (Safari)
  - ...

1 Run the test you want to debug:

```
java -jar jsTestDriver.jar --tests HelloWorldTest.  
testOutput
```

# Debugging

- Use whatever javascript debugging technique you like!
  - Firebug (Firefox)
  - Visual Studio (IE)
  - Web Inspector (Safari)
  - ...

## 1 Run the test you want to debug:

```
java -jar jsTestDriver.jar --tests HelloWorldTest.  
testOutput
```

## 2 Set breakpoints

# Debugging

- Use whatever javascript debugging technique you like!
  - Firebug (Firefox)
  - Visual Studio (IE)
  - Web Inspector (Safari)
  - ...

1 Run the test you want to debug:

```
java -jar jsTestDriver.jar --tests HelloWorldTest.  
testOutput
```

2 Set breakpoints

3 Rerun the test using the command above.

# Debugging - Using the console

---

- A lot of javascript debuggers provide a `console` object

# Debugging - Using the console

---

- A lot of javascript debuggers provide a console object

```
console.log(" Hello World!");
```

# Debugging - Using the console

---

- A lot of javascript debuggers provide a console object

```
console.log(" Hello World!");
```

- Redirect this output to js-test-driver

# Debugging - Using the console

- A lot of javascript debuggers provide a console object

```
console.log("Hello World!");
```

- Redirect this output to js-test-driver

```
java -jar jsTestDriver.jar --tests all --  
captureConsole
```

- **Note:** Does not work in Firefox

# Debugging - Using the console

- A lot of javascript debuggers provide a console object

```
console.log(" Hello World!");
```

- Redirect this output to js-test-driver

```
java -jar jsTestDriver.jar --tests all --  
captureConsole
```

- Note: Does not work in Firefox
- Use `jstestdriver.console` directly

# Debugging - Using the console

- A lot of javascript debuggers provide a console object

```
console.log(" Hello World!");
```

- Redirect this output to js-test-driver

```
java -jar jsTestDriver.jar --tests all --  
captureConsole
```

- Note: Does not work in Firefox

- Use `jstestdriver.console` directly

```
jstestdriver.console.log(" Hello World!");
```

Debugging example

Live Demo!

# What comes next?

---

## Automation and CI

# Test automation and continuous integration

---

- `--testOutput` parameter generates **JUnit XML compatible** test logs

# Test automation and continuous integration

---

- `--testOutput` parameter generates **JUnit XML compatible** test logs
  - One XML for every used browser

# Test automation and continuous integration

- `--testOutput` parameter generates **JUnit XML compatible** test logs
  - One XML for every used browser

```
java -jar jsTestDriver.jar --testOutput someDirectory  
--tests all
```

# Test automation and continuous integration

- `--testOutput` parameter generates **JUnit XML compatible** test logs
  - One XML for every used browser

```
java -jar jsTestDriver.jar --testOutput someDirectory  
--tests all
```

- `--browser` parameter **runs** and **captures** browsers automatically

# Test automation and continuous integration

- `--testOutput` parameter generates **JUnit XML compatible** test logs
  - One XML for every used browser

```
java -jar jsTestDriver.jar --testOutput someDirectory  
--tests all
```

- `--browser` parameter **runs** and **captures** browsers automatically

```
java -jar jsTestDriver.jar --browser executable1 ,  
executable2 ,... --port 4224
```

# Test automation and continuous integration

---

- `--port` and `--tests` can be combined to automate server startup and `testrun`

# Test automation and continuous integration

---

- `--port` and `--tests` can be combined to automate server startup and `testrun`

```
java -jar jsTestDriver.jar --browser executable1 ,  
executable2 ,... --port 4224 --tests all
```

# What comes next?

---

## Code Coverage

# Code coverage

---

- Code coverage may be an **indicator** for **untested code paths**

# Code coverage

---

- Code coverage may be an **indicator** for **untested code paths**
- `js-test-driver` **plugin** allows to create code coverage reports

# Code coverage

---

- Code coverage may be an **indicator** for **untested code paths**
- js-test-driver **plugin** allows to create code coverage reports
- Possible output formats

# Code coverage

---

- Code coverage may be an **indicator** for **untested code paths**
- js-test-driver **plugin** allows to create code coverage reports
- Possible output formats
  - **Textual output** on terminal

# Code coverage

---

- Code coverage may be an **indicator** for **untested code paths**
- js-test-driver **plugin** allows to create code coverage reports
- Possible output formats
  - **Textual output** on terminal
  - **LCOV** compatible info file

# Code coverage - Plugin

---

- Download **jar-file** from js-test-driver homepage

# Code coverage - Plugin

---

- Download `jar-file` from `js-test-driver` homepage
- Put the `coverage.jar` file into some folder relative to your `js-text-driver.jar` (eg. `plugins/coverage.jar`)

# Code coverage - Plugin

- Download **jar-file** from js-test-driver homepage
- Put the **coverage.jar** file into some folder relative to your **js-text-driver.jar** (eg. `plugins/coverage.jar`)
- Add plugin to `jsTestDriver.conf`

```
1 plugin:  
2   - name: "coverage"  
3     jar: "plugins/coverage.jar"  
4     module: "com.google.jstestdriver.coverage.  
           CoverageModule"
```

# Using LCOV to generate HTML reports

---

- The `--testOutput` commandline parameter generates code coverage reports in **LCOV** compatible info file format


# Using LCOV to generate HTML reports

- The `--testOutput` commandline parameter generates code coverage reports in **LCOV** compatible info file format
- Use `genHtml` from the LCOV package to generate nice HTML results:

## LCOV - code coverage report

Current view: <a href="#">directory</a> - 03_coverage/src		Found	Hit	Coverage
Test: <code>jsTestDriver.conf-coverage.dat</code>	Lines:	6	5	83.3 %
Date: 2009-10-10	Functions:	0	0	-

Filename	Line Coverage ↕	Functions ↕
<a href="#">hello_world.js</a>	 83.3 % 5 / 6	- 0 / 0

Generated by: [LCOV version 1.7](#)

- <http://ltp.sourceforge.net>

Code coverage example

Live Demo!

Questions, comments or annotations?

Slides: <http://westhoffswelt.de/portfolio.htm>

Contact: Jakob Westhoff <[jakob@php.net](mailto:jakob@php.net)>

- Breathe Icon Set:

<https://launchpad.net/breathe-icon-set>, Licensed under Creative Common Attribution-ShareAlike 3.0 License (<http://creativecommons.org/licenses/by-sa/3.0>)