# django_seleniumhelpers Documentation

***Release 1.1***

**Espen Angell Kristiansen**

December 04, 2015

Contents

CHAPTER 1

# Code/issues

https://github.com/espenak/django_seleniumhelpers

CHAPTER 2

# Contents

## 2.1 Getting started

### 2.1.1 Features

- Skip selenium tests
- Select test browser
- Helper functions to simplify common use cases.

### 2.1.2 Issues/contribute

Report any issues at the github project page, and feel free to add your own guides/experiences to the wiki, and to contribute changes using pull requests.

### 2.1.3 Install

Requires **django>=1.4**.

```
$ pip install django_seleniumhelpers
```

### 2.1.4 Setup

Add `'seleniumhelpers'` to `INSTALLED_APPS`.

## 2.2 Usage

Subclass *`seleniumhelpers.SeleniumTestCase`* instead of `django.test.LiveServerTestCase` in your testcases.

### 2.2.1 Example

```
from seleniumhelpers import SeleniumTestCase

class TestFrontpage(SeleniumTestCase):
    def test_frontpageimage(self):
        # Use our shortcut instead of self.selenium.get(self.live_server_url + '/something')
        self.getPath('/something/')

        # Fail unless the expected image and text is available within 10 secons
        self.waitForCssSelector('img.frontpageimage')
        self.waitForText(u'This is the frontpage image text.')

        # Assert that we remembered to close the body. Mostly to show how
        # to get hold of the selenium WebDriver object
        # Note: The selenium attribute is actually set as an attribute on
        #       the class in SeleniumTestCase.setUpClass()
        self.assertTrue('</body>' in self.selenium.page_source)
```

### 2.2.2 Running tests

You can run the tests just like normal Django tests, however we provide the ability to override the browser used for the tests, and to completely skip all selenium tests.

Select selenium browser:

```
$ SELENIUM_BROWSER=Firefox python manage.py test
```

See available browsers:

```
$ python manage.py listseleniumbrowsers
```

Skip selenium tests:

```
$ SKIP_SELENIUMTESTS=1 python manage.py test
```

**Note:** Settings can also be set in `settings.py`. See settings.

### 2.2.3 Configure timout

You can configure the default timeout using `SELENIUM_DEFAULT_TIMEOUT`, in `settings.py` or as environment variables, just like `SKIP_SELENIUMTESTS` and `SELENIUM_BROWSER`. Example:

```
$ SELENIUM_BROWSER=Firefox SELENIUM_DEFAULT_TIMEOUT=10 python manage.py test
```

The default timeout is `4` seconds, which should be enough unless you are running on a very slow machine.

### 2.2.4 Use Selenium RC

Using Selenium RC is easy, it only requires you to run the RC-server, and use an additional setting:

1. Download *selenium-server-standalone-XXXX.jar* from http://code.google.com/p/selenium/downloads/list.
2. Run the RC-server:

```
    $ java -jar selenium-server-standalone-XXXX.jar
```

3. Run the tests with SELENIUM_USE_RC:

```
$ SELENIUM_USE_RC=true SELENIUM_BROWSER=Opera python manage.py test
```

### 2.2.5 Use Chrome or Chromium

Download and install chromedriver (webdriver for Chrome/Chromium) with one of these methods:

- Install with `brew install chromedriver` on Mac OSX if you use Homebrew.
- Download from http://chromedriver.storage.googleapis.com/index.html.

Run with `SELENIUM_BROWSER=Chrome`.

## 2.3 PhantomJS

We support PhantomJS via ghostdriver.

### 2.3.1 Requirements

You need to build PhantomJS as described on the ghostdriver website.

### 2.3.2 Running tests with PhantomJS and ghostdriver

Run PhantomJS with ghostdriver on port `8080` as described on the ghostdriver website:

```
$ phantomjs /path/to/ghostdriver/src/main.js 8080
```

Run the tests with `SELENIUM_BROWSER=phantomjs`:

```
SELENIUM_BROWSER=phantomjs python manage.py test
```

## 2.4 Settings

All the settings can be set in `settings.py`, and most settings can be set through using environment variables. If a setting is both in `settings.py` and as an environment variable, the environment variable is used.

### 2.4.1 List of settings

**SKIP_SELENIUMTESTS** Skip all seleniumtests.

**SELENIUM_BROWSER** The selenium browser to use. Defaults to `Chrome`.

**SELENIUM_USE_RC** If `bool(SELENIUM_USE_RC)` is `True`, we use the Selenium RC server instead of webdriver to run the tests. `SELENIUM_BROWSER` is forwarded to the RC-server as the browser.

**SELENIUM_DEFAULT_TIMEOUT** The default timeout, in seconds, of the `waitFor*` methods. Defaults to `4`.

## 2.5 API-docs

### 2.5.1 SeleniumTestCase API docs

**class** `seleniumhelpers.`**`SeleniumTestCase`**(*methodName='runTest'*)

Extends `django.test.LiveServerTestCase` to simplify selenium testing.

**`executeScript`**(*script*, *element*)

Shortcut for `self.selenium.executeScript(script, element)`.

**`failIfCssSelectorFound`**(*element*, *css_selector*, *msg='CSS selector, "{css_selector}" matches at least one element, when we expected it not to.'*)

Assert that `element.find_element_by_css_selector(css_selector)` does not raise `NoSuchElementException`.

**classmethod `getDriver`**(*browser*, *use_rc*)

Override this to create customize the `selenium`-attribute.

**Parameters**

- **`browser`** – The value of the `SELENIUM_BROWSER` setting.
- **`use_rc`** – The value of `bool(SELENIUM_USE_RC)`.

**`getInnerHtml`**(*element*)

Get `innerHTML` of the given element.

**`getPath`**(*path*)

Shortcut for `self.selenium.get(...)` with `path` prefixed by `live_server_url` as argument.

**classmethod `setUpClass`**()

Adds the `selenium` attribute to the class. The `selenium` attribute defaults to an instance of `selenium.webdriver.Chrome`, however this can be overridden using the `SELENIUM_BROWSER` django setting or environment variable. If both the django setting and and environment variable is set, the environment variable is used. This means that you can set the default value in `settings.py` and override it in an environment variable, typically when running the `test` command:

```
SELENIUM_BROWSER=Firefox python manage.py test
```

**`waitFor`**(*item*, *fn*, *timeout=4*, *msg=None*)

Wait for the `fn` function to return `True`. The `item` is forwarded as argument to `fn`.

Example (wait for text in an element):

```
waitFor(myelem, lambda myelem: len(myelem.text) > 0, msg='myelem is empty')
```

**`waitForAndFindElementByCssSelector`**(*cssselector*, *within=None*, *timeout=4*)

Use *`waitForCssSelector()`* to wait until `cssselector` is found, then use `self.selenium.find_element_by_css_selector` to locate and return the element.

**Parameters**

- **`within`** – The element to run `find_element_by_css_selector()` on. Defaults to `self.selenium`.
- **`timeout`** – Fail unless the `cssselector` is found before `timeout` seconds.

**`waitForCssSelector`**(*cssselector*, *timeout=4*, *within=None*, *msg='No elements match css selector "{cssselector}".'*)

Wait for the given `cssselector`.

**Parameters**

- **within** – The element to run `find_element_by_css_selector()` on. Defaults to `self.selenium`.
- **timeout** – Fail unless the `cssselector` is found before `timeout` seconds.

**waitForCssSelectorNotFound**(*cssselector*, *timeout=4*, *within=None*, *msg='CSS selector, "{cssselector}" matches at least one element, when we expected it not to.'*)
Wait for the given `cssselector` not to be found.

**Parameters**

- **within** – The element to run `find_elements_by_css_selector()` on. Defaults to `self.selenium`.
- **timeout** – Fail if the `cssselector` is still found after `timeout` seconds.

**waitForDisabled**(*element*, *timeout=4*, *msg='The element is not disabled.'*)
Wait for the given `element` to become disabled (`element.is_enabled() == False`).

**Parameters** **timeout** – Fail unless the `element` becomes disabled before `timeout` seconds. Defaults to `10`.

**waitForDisplayed**(*element*, *timeout=4*, *msg='The element is not displayed.'*)
Wait for the given element to be displayed.

**waitForEnabled**(*element*, *timeout=4*, *msg='The element is not enabled.'*)
Wait for the given `element` to become enabled (`element.is_enabled() == True`).

**Parameters** **timeout** – Fail unless the `element` becomes enabled before `timeout` seconds.

**waitForNotDisplayed**(*element*, *timeout=4*, *msg='The element is not hidden.'*)
Wait for the given element to be hidden.

**waitForText**(*text*, *timeout=4*, *msg='Could not find text "{text}"'*, *within=None*)
Wait for `text` to appear in `selenium.page_source` or from the text of an element.

**Parameters**

- **within** – The element to find text within (uses within.text). If this is not specified, we get text from `selenium.page_source`.
- **timeout** – Fail unless the `text` appears in `selenium.page_source` before `timeout` seconds has passed.

**waitForTitle**(*title*, *timeout=4*)
Wait until the page title (title-tag) equals the given `title`.

**waitForTitleContains**(*title*, *timeout=4*)
Wait until the page title (title-tag) contains the given `title`.

### 2.5.2 Helper-functions

`seleniumhelpers.`**get_setting_with_envfallback**(*setting*, *default=None*, *typecast=None*)
Get the given setting and fall back to the default of not found in `django.conf.settings` or `os.environ`.

**Parameters**

- **settings** – The setting as a string.
- **default** – The fallback if `setting` is not found.
- **typecast** – A function that converts the given value from string to another type. E.g.: Use `typecast=int` to convert the value to int before returning.

seleniumhelpers.**get_default_timeout**()
Get the default timeout. Uses *get_setting_with_envfallback()* to get SELENIUM_DEFAULT_TIMEOUT. Defaults to 4.

## 2.6 Customize the driver-object (cls.selenium)

The selenium-attribute of the seleniumhelpers.SeleniumTestCase` is created in the *seleniumhelpers.SeleniumTestCase.getDriver()* classmethod. You can override this method if you have more advanced needs than the simple customizations provided by the default implementation.

### 2.6.1 Example

This example makes it possible to run Firefox3.6 if SELENIUM_BROWSER=="Firefox3.6":

```
from seleniumhelpers import SeleniumTestCase
from selenium import webdriver

class CustomSeleniumTestCase(SeleniumTestCase):
    @classmethod
    def getDriver(cls, browser, use_rc):
        if browser == 'Firefox3.6':
            return webdriver.Remote('desired_capabilities': {'browser_name': browser,
                                                             'version': '3.6'})
        else:
            return super(CustomSeleniumTestCase, self).getDriver(browser, use_rc)
```

CHAPTER 3

# Indices and tables

- genindex
- modindex
- search

## E



## F



## G



## S



## W