webdriverplus

Release 0.1

Sep 27, 2017

Contents

CHAPTER 1

The most simple and powerful way to use Selenium with Python

WebDriver Plus is an extension to the Python bindings for Selenium WebDriver, which gives you a more concise and expressive API.

It helps you to quickly write readable, robust tests.

What's so great about WebDriver Plus?

- Browser instances that support pooling and automatic quit on exit.
- A concise API for finding elements, and a wide range of selectors.
- JQuery-style traversal and filtering for locating elements without using complex xpath expressions.
- Perform actions directly on elements without having to use ActionChains.
- Element highlighting makes working from the Python console a joy.
- Supports headless mode using HtmlUnit, Xvfb or Xvnc.

Help contribute on GitHub.

CHAPTER 2

Getting started

Install `webdriverplus` using `pip`.

```
pip install webdriverplus
```

Now fire up your Python console...

```
>>> from webdriverplus import WebDriver
>>> browser = WebDriver().get('http://www.google.com')
```

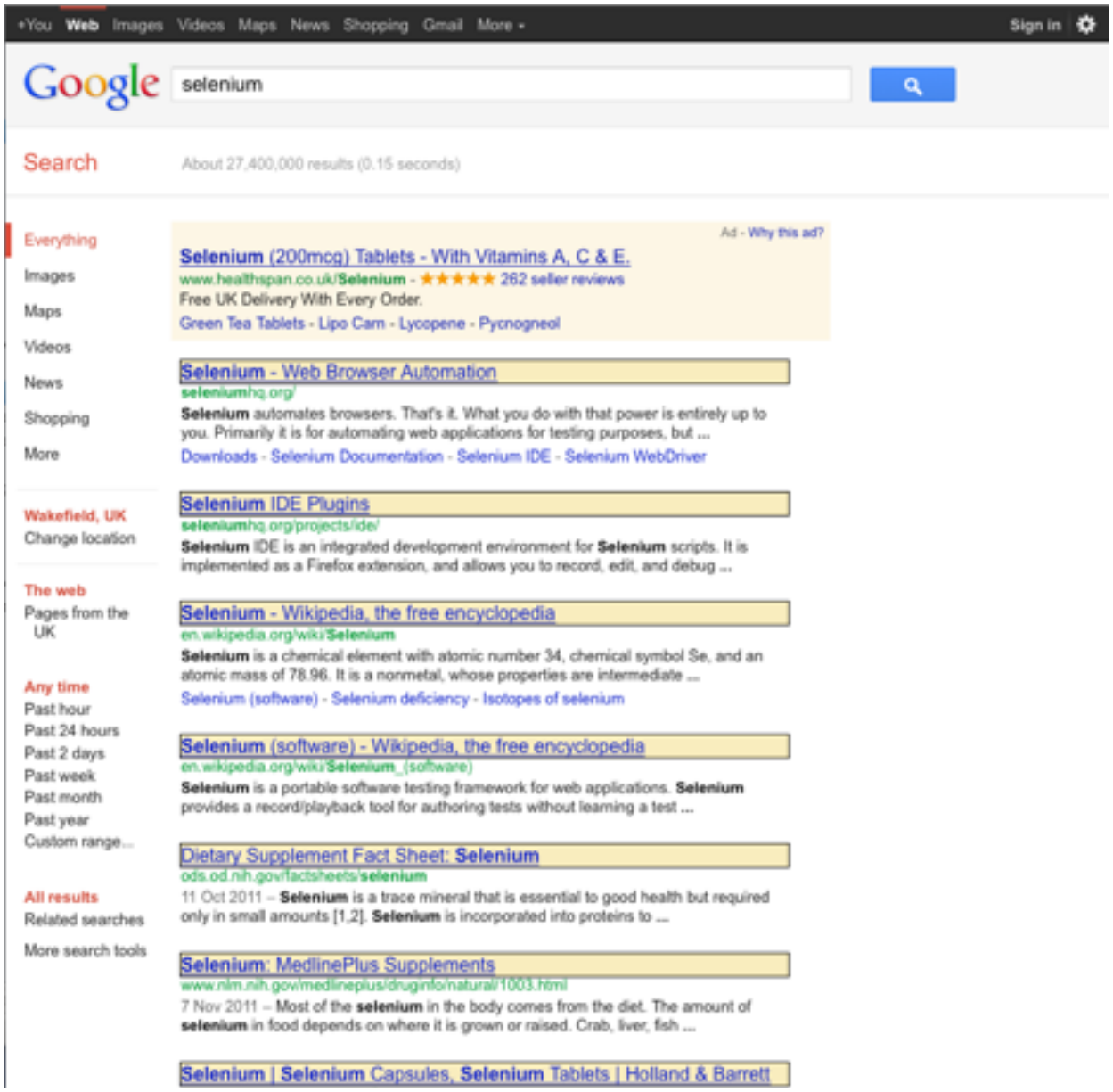
Ok, let's do a search.

```
>>> browser.find(name='q').send_keys('selenium\n')
```

Now let's get the headings for all the search results.

```
>>> browser.find(id='search').find('h3')
WebElementSet(
  <h3 class="r"><a href="http://seleniumhq.org/" class="l" onmousedown="retur...
  <h3 class="r"><a href="http://www.google.co.uk/url?sa=t&amp;rct=j&amp;q=sel...
  <h3 class="r"><a href="http://en.wikipedia.org/wiki/Selenium" class="l" onm...
  <h3 class="r"><a href="http://en.wikipedia.org/wiki/Selenium_%28software%29...
  <h3 class="r"><a href="http://ods.od.nih.gov/factsheets/selenium" class="l"...
  <h3 class="r"><a href="http://www.nlm.nih.gov/medlineplus/druginfo/natural/...
  <h3 class="r"><a href="http://www.hollandandbarrett.com/selenium-050" class...
  <h3 class="r"><a href="http://www.whfoods.com/genpage.php?dbid=95&amp;tname...
  <h3 class="r"><a href="http://www.patient.co.uk/doctor/Selenium.htm" class=...
  <h3 class="r"><a href="/search?q=selenium&amp;hl=en&amp;biw=940&amp;bih=938...
  <h3 class="r"><a href="http://www.umm.edu/altmed/articles/selenium-000325.h...
)
```

Notice that WebDriver Plus has highlighted the selected elements for you, which is super helpful for when you're writing tests and trying to make sure you're selecting the correct elements:


+You Web Images Videos Maps News Shopping Gmail More
Sign in
Google
selenium
Search
About 27,400,000 results (0.15 seconds)
Everything
Images
Maps
Videos
News
Shopping
More
Wakefield, UK
Change location
The web
Pages from the UK
Any time
Past hour
Past 24 hours
Past 2 days
Past week
Past month
Past year
Custom range...
All results
Related searches
More search tools
Ad - Why this ad?
Selenium (200mcg) Tablets - With Vitamins A, C & E.
www.healthspan.co.uk/Selenium - 262 seller reviews
Free UK Delivery With Every Order.
Green Tea Tablets - Lipo Cam - Lycopene - Pycnogneol
Selenium - Web Browser Automation
seleniumhq.org/
Selenium automates browsers. That's it. What you do with that power is entirely up to you. Primarily it is for automating web applications for testing purposes, but ...
Downloads - Selenium Documentation - Selenium IDE - Selenium WebDriver
Selenium IDE Plugins
seleniumhq.org/projects/ide/
Selenium IDE is an integrated development environment for Selenium scripts. It is implemented as a Firefox extension, and allows you to record, edit, and debug ...
Selenium - Wikipedia, the free encyclopedia
en.wikipedia.org/wiki/Selenium
Selenium is a chemical element with atomic number 34, chemical symbol Se, and an atomic mass of 78.96. It is a nonmetal, whose properties are intermediate ...
Selenium (software) - Isotopes of selenium
Selenium (software) - Wikipedia, the free encyclopedia
en.wikipedia.org/wiki/Selenium_(software)
Selenium is a portable software testing framework for web applications. Selenium provides a record/playback tool for authoring tests without learning a test ...
Dietary Supplement Fact Sheet: Selenium
ods.od.nih.gov/factsheets/selenium
11 Oct 2011 – Selenium is a trace mineral that is essential to good health but required only in small amounts [1,2]. Selenium is incorporated into proteins to ...
Selenium: MedlinePlus Supplements
www.nlm.nih.gov/medlineplus/druginfo/natural/1003.html
7 Nov 2011 – Most of the selenium in the body comes from the diet. The amount of selenium in food depends on where it is grown or raised. Crab, liver, fish ...
Selenium | Selenium Capsules, Selenium Tablets | Holland & Barrett

CHAPTER 3

Overview

Browser Instances

Creating browser instances

To create a WebDriver browser instance:

```
from webdriverplus import WebDriver
browser = WebDriver('firefox')
```

Currently supported browsers are `firefox`, `chrome`, `ie`, `htmlunit`, `remote` and `phantomjs`.

The default browser is firefox. If called without any arguments, `WebDriver()` will create a firefox instance.

```
browser = WebDriver()
```

reuse_browser

Because starting up a web browser instance on every test run can be a significant performance hit, WebDriverPlus provides an easy way to allows instances to be reused between test sessions.

Setting the `reuse_browser` flag ensures that when you call `driver.quit()` the browser will be returned to a browser pool, and reused when you create a new WebBrowserInstance:

```
browser = WebDriver('firefox', reuse_browser=True)
```

There are some important aspects to bear in mind about this behaviour:

- WebDriver Plus currently has no way of clearing browser history or cache. Be aware that this may affect the behaviour of tests.
- On quitting the browser and returning it to the pool, WebDriver Plus will clear any cookies for the browser for the current domain. It has no way of clearing all cookies for all domains. If you have test cases that access URLs from multiple domains, consider if you need to explicitly clear any cookies between sessions.

- WebDriver Plus will only retain one instance of each browser type (Eg Firefox, Chrome etc...) in the pool. Instances will only be reused if the arguments to the WebDriver() constructor have not changed since the previous instance was created.

quit_on_exit

By default WebDriverPlus will ensure that when a python process quits it will attempt to quit any remaining open WebDriver instances.

If you do not want this behaviour set quit_on_exit to False:

```
browser = WebDriver('firefox', quit_on_exit=False)
```

wait

By default WebDriverPlus will not wait for elements to become available. You can pass a `wait` argument to specify the number of seconds that `find()` should wait for elements to become available before timing out.

```
browser = WebDriver('firefox', wait=10)
```

This uses WebDriverWait under the covers, but is much less verbose. The idea behind setting a per-browser wait argument instead of forcing the programmer to use `WebDriverWait` around each piece of code that needs to wait for an element is to free the programmer from having to think about waiting, which we consider a low-level detail that the framework should deal with.

highlight

By default WebDriverPlus highlights elements it found. Setting highlight to falsy will disable this

Switching to an iframe

To switch to a specific iframe, call `switch_to_frame()`

Waiting for a specific condition

Right now `webdriverplus` supports expected condition style waiting with `wait_for(selector, displayed=True`. By default it will wait until the element with `selector` to be present AND visible. If `displayed` is set to False, it will only wait until element is present.

```
browser.wait_for('div')
```

Quitting browser instances

To quit a WebDriver browser instance, call `quit()`:

```
browser.quit()
```

force

Setting the `force` flag causes the browser instance to quit and ignore the value of the `reuse_browser` flag. The instance will be terminated and will not be returned to the browser pool:

```
browser.quit(force=True)
```

Supported browsers

- Firefox - Should run out-of-the-box.
- Chrome - Install the chrome driver first.
- IE - Install the IE driver first.
- HTMLUnit (headless browser) - should auto-install and run out-of-the-box.
- PhantomJS - Install PhantomJS first.

Headless mode using Xvfb or Xvnc

Using pyvirtualdisplay, you can run real browser instances in a virtual X framebuffer or VNC session. This enables you to run Firefox or Chrome tests in headless mode, without having to install HTMLUnit.

```
$ pip install pyvirtualdisplay
```

You need to install either Xvfb or Xvnc as a backend for *pyvirtualdisplay*.

To run the headless tests, use the `--headless` argument:

```
$ python runtests.py --headless
Running tests in headless mode.
.....................................................
----------------------------------------------------------------------
Ran 57 tests in 7.715s

OK
```

Selectors

Finding elements on the page

WebDriver Plus supports a wide variety of selectors to let you easily locate elements on the page.

To locate elements on the page use the `.find()` method. Calling `.find()` will return a set of all the elements that matched the selectors.

```
import webdriverplus
browser = webdriverplus.WebDriver()
browser.get('http://www.example.com/')
browser.find(id='login-form')
```

Different types of selector are used depending on the keyword arguments passed to `find()`.

```
browser.find(id='login-form')
browser.find(tag_name='a')
browser.find(css='#login-form input')
browser.find(link_text='Images')
```

The default selector for the `.find()` method is `css`. If you pass an unnamed argument to `.find()` it will be treated as a css selector.

```
browser.find('a')              # All 'a' tags
browser.find('a.selected')     # All 'a' tags with 'selected' class
browser.find('div#content a')  # All 'a' tags within the 'content' div
```

Multiple selectors can be used in a single expression. The resulting set will be the set of elements that match all the given selectors.

```
browser.find('input', type='checkbox', checked=True)
```

Note: When finding elements and traversing the DOM, WebDriver Plus follows the same style as JQuery.

You will always be working with sets of WebElements rather than indivdual elements. Actions on those sets, such as `.click()`, will applied only to the first element in the set.

Chaining selectors

The `.find()` method can be applied either to the browser instance, or to an existing set of elements. When applied to an existing set of elements `.find()` will return all children of any element in the set that match the given selector.

```
login = browser.find(id='login-form')  # The login form
inputs = login.find('input')           # All 'input' tags within the form
```

Selectors can also be chained in a single expression.

```
browser.find(id='login-form').find('input')
```

Supported selectors

css

id

name

tag_name

class_name

xpath

text

text_contains

link_text

link_text_contains

attribute

attribute_value

value

type

checked

selected

Actions

Actions in WebDriver Plus always operate on the first element in the `WebElementSet`. If you want to apply an action to each element in the set, you should iterate over the set:

```
for elem in browser.find('input', type='checkbox'):
    elem.check()
```

Actions return the original `WebElementSet`, which means they can be chained. For example:

```
elem = browser.find(id='username')
elem.send_keys('denvercoder9').submit()
```

Supported actions

Currently the following actions are supported.

Note: Many actions are not yet fully supported natively through WebDriver, and have to instead be simulated using javascript. As a result it's possible that some behavior may vary slightly between different web browsers.

.click()

Clicks an element.

.double_click()

Double-clicks an element.

.context_click()

Performs a context-click (right click) on an element.

.move_to(x, y)

Moves the mouse to an element with offset x and y.

.move_to_and_click(x, y)

Moves the mouse to an element with offset x and y, then click.

.click_and_hold()

Holds down on an element.

.release()

Releases a held click.

.check()

Clicks a checkbox if it isn't already checked.

.uncheck()

Clicks a checkbox to uncheck it if it's checked.

.submit()

If the element is a form, submit it. Otherwise search for a form enclosing the element, and submit that.

.clear()

Clears any user editable text from the element.

.send_keys(*text*)

Sends keys to an element.

.type_keys(*text*)

Bug in chrome driver that prevents send_keys to certain elements so click 1st, clear, then send_keys.

https://code.google.com/p/chromedriver/issues/detail?id=290

.select_option(value, text, index)

Trigger option select based on `value`, `text`, or `index` if the element is a `select` element

.deselect_option(value, text, index)

Trigger option deselect based on `value`, `text`, or `index` if the element is a `select` element

Traversing

WebDriver Plus supports a large set of JQuery style DOM traversal methods. These help you to you easily target the parts of the web page you're interested in.

Traversal methods in WebDriver Plus can be called without any arguments:

```
>>> elems.children()
```

Or they can be filtered by one or more selectors:

```
>>> elems.children('input', type='checkbox')
```

children(*selector*)

Get the children of each element in the set of matched elements, optionally filtered by a selector.

```
>>> from webdriverplus import WebDriver
>>> snippet = """
... <ul>
...     <li>1</li>
...     <li><strong>2</strong></li>
...     <li>3</li>
... </ul>"""
>>> WebDriver().open(snippet).find('ul').children()
WebElementSet(
  <li>1</li>
  <li><strong>2</strong></li>
  <li>3</li>
)
```

parent(*selector*)

Get the parent of each element in the current set of matched elements, optionally filtered by a selector.

```
>>> from webdriverplus import WebDriver
>>> snippet = """
... <ul>
...     <li>1</li>
...     <li><strong>2</strong></li>
...     <li>3</li>
... </ul>"""
>>> WebDriver().open(snippet).find('strong').parent()
WebElementSet(
  <li><strong>2</strong></li>
)
```

descendants()

Get the descendants of each element in the current set of matched elements.

```
>>> from webdriverplus import WebDriver
>>> snippet = """
... <ul>
...     <li>1</li>
...     <li><strong>2</strong></li>
...     <li>3</li>
... </ul>"""
>>> WebDriver().open(snippet).find('ul').descendants()
WebElementSet(
  <li>1</li>
  <li><strong>2</strong></li>
  <strong>2</strong>
  <li>3</li>
)
```

Note: Unlike the other traversal operations `.descendants()` cannot be filtered by a selector. You should use `.find()` instead, which is equivelent to filtering against all descendants.

ancestors(*selector*)

Get the ancestors of each element in the current set of matched elements, optionally filtered by a selector.

```
>>> from webdriverplus import WebDriver
>>> snippet = """
... <ul>
...     <li>1</li>
...     <li class="selected">2</li>
...     <li>3</li>
... </ul>"""
>>> WebDriver().open(snippet).find('.selected').parents()
WebElementSet(
  <html webdriver="true"><head></head><body><ul> <li>1</li> <li class="select...
  <body><ul> <li>1</li> <li class="selected">2</li> <li>3</li> </ul></body>
  <ul> <li>1</li> <li class="selected">2</li> <li>3</li> </ul>
)
```

next(*selector*)

Get the immediately following sibling of each element in the set of matched elements, optionally filtered by a selector.

```
>>> from webdriverplus import WebDriver
>>> snippet = """
... <ul>
...     <li>1</li>
...     <li>2</li>
...     <li class="selected">3</li>
...     <li>4</li>
...     <li>5</li>
... </ul>"""
```

```
>>> WebDriver().open(snippet).find('li.selected').next()
WebElementSet(
  <li>4</li>
)
```

prev(*selector*)

Get the immediately preceding sibling of each element in the set of matched elements, optionally filtered by a selector.

```
>>> from webdriverplus import WebDriver
>>> snippet = """
... <ul>
...     <li>1</li>
...     <li>2</li>
...     <li class="selected">3</li>
...     <li>4</li>
...     <li>5</li>
... </ul>"""
>>> WebDriver().open(snippet).find('li.selected').prev()
WebElementSet(
  <li>2</li>
)
```

next_all(*selector*)

Get all following siblings of each element in the set of matched elements, optionally filtered by a selector.

```
>>> from webdriverplus import WebDriver
>>> snippet = """
... <ul>
...     <li>1</li>
...     <li>2</li>
...     <li class="selected">3</li>
...     <li>4</li>
...     <li>5</li>
... </ul>"""
>>> WebDriver().open(snippet).find('li.selected').next_all()
WebElementSet(
  <li>4</li>
  <li>5</li>
)
```

prev_all(*selector*)

Get all preceding siblings of each element in the set of matched elements, optionally filtered by a selector.

```
>>> from webdriverplus import WebDriver
>>> snippet = """
... <ul>
...     <li>1</li>
...     <li>2</li>
...     <li class="selected">3</li>
...     <li>4</li>
```

```
...     <li>5</li>
... </ul>"""
>>> WebDriver().open(snippet).find('li.selected').prev_all()
WebElementSet(
  <li>1</li>
  <li>2</li>
)
```

siblings(*selector*)

Get the siblings of each element in the set of matched elements, optionally filtered by a selector.

```
>>> from webdriverplus import WebDriver
>>> snippet = """
... <ul>
...     <li>1</li>
...     <li>2</li>
...     <li class="selected">3</li>
...     <li>4</li>
...     <li>5</li>
... </ul>"""
>>> WebDriver().open(snippet).find('li.selected').siblings()
WebElementSet(
  <li>1</li>
  <li>2</li>
  <li>4</li>
  <li>5</li>
)
```

Filtering

The `filter()` and `exclude()` methods can be used to select a subset of elements from a `WebElementSet`.

filter(*selector*)

Filters a `WebElementSet` to only include elements that match the selector.

```
>>> from webdriverplus import WebDriver
>>> snippet = """
... <ul>
...     <li>1</li>
...     <li class="selected">2</li>
...     <li>3</li>
...     <li>4</li>
...     <li class="selected">5</li>
... </ul>"""
>>> WebDriver().open(snippet).find('li').filter('.selected')
WebElementSet(
  <li class="selected">2</li>
  <li class="selected">5</li>
)
```

exclude(*selector*)

Filters a `WebElementSet` to only include elements that do not match the selector.

```
>>> from webdriverplus import WebDriver
>>> snippet = """
... <ul>
...     <li>1</li>
...     <li class="selected">2</li>
...     <li>3</li>
...     <li>4</li>
...     <li class="selected">5</li>
... </ul>"""
>>> WebDriver().open(snippet).find('li').exclude('.selected')
WebElementSet(
  <li>1</li>
  <li>3</li>
  <li>4</li>
)
```

Inspection & Manipulation

Note: Some changes to be made here - slim down the number of methods, be more consistent.

.css(name, value)

A jQuery style shortcut to getting/setting `css` value of the element. Note that name is javascript-style name (so `backgroundColor` instead of `background-color`). Leaving `value` empty will simply return the value of that css property.

.style

Returns an object that lets you get and set the CSS style of an element.

```
>>> import webdriverplus
>>> driver = webdriverplus.Firefox()
>>> driver.get('http://www.google.com')
>>> for elem in driver.find('input'):
>>>     elem.style.background = 'green'
>>> driver.find('body').style.background = 'red'
```

.size

Returns the size of the element, as a `namedtuple`.

```
>>> elem.size
Size(width=300, height=25)
>>> width, height = elem.size
>>> width = elem.size.width
```

.location

Returns the location of the element in the canvas, as a `namedtuple`.

```
>>> elem.location
Location(x=10, y=50)
>>> x, y = elem.location
>>> x = elem.location.x
>>> y = elem.location.y
```

.value

.type

.name

.is_checked()

Returns `True` if the checkbox has a `checked` attribute, `False` otherwise.

```
>>> from webdriverplus import WebDriver
>>> snippet = """
... <input type="checkbox" name="vehicle" value="Walk" />I walk to work</input>
... <input type="checkbox" name="vehicle" value="Cycle" checked />I cycle to work</
↪input>
... <input type="checkbox" name="vehicle" value="Drive" />I drive to work</input>
... """
>>> driver = WebDriver().open(snippet)
>>> driver.find(value='Walk').is_checked
False
>>> driver.find(value='Cycle').is_checked
True
```

.is_selected()

Returns `True` if the option has a `selected` attribute, `False` otherwise.

```
>>> from webdriverplus import WebDriver
>>> snippet = """
... <select>
...     <option selected>Walk</option>
...     <option>Cycle</option>
...    <option>Driver</option>
... </select>
... """
>>> driver = WebDriver().open(snippet)
>>> driver.find(text='Walk').is_selected
True
>>> driver.find(text='Cycle').is_selected
False
```

.is_enabled()

.is_displayed()

.class

Returns the set of CSS classes applied on the element.

.has_class(cls)

Checks if class `cls` is in the list of the element's classes

.id

Returns the 'id' attribute of the element.

.tag_name

Returns the element's tag name. (Eg. 'h1', 'div', 'input')

.attr(attribute)

Returns the `attribute` from the element with corresponding name

.attributes

Returns a dictionary-like object representing all the DOM attributes on the element. Supports getting, setting, and deleting attributes.

```
>>> elem = driver.find(id='logo')
>>> elem.attributes
{u'width': u'50px', u'src': u'/static/images/logo.png', u'height': u'50px'}
>>> elem.attributes['src']
u'/static/images/logo.png'
>>> elem.attributes['src'] = '/static/images/other.png'
>>> del(elem.attributes['width'])
>>> del(elem.attributes['height'])
>>> elem.attributes
{u'src': u'/static/images/other.png'}
```

Note: The values returned by `.attributes` differ slightly from those returned by WebDriver's `.get_attribute()`.

Eg: When dealing with sizes, `.attribute['height']` returns a value like `50px` where `.getAttribute('height')` returns a value like `50`. When dealing with links, `.attribute['src']` returns the raw src value, where `.getAttribute('src')` returns an absolute URL.

Both styles are supported by WebDriver Plus.

CHAPTER 4

Topics

Migrating to WebDriver Plus

Where possible WebDriver Plus remains compatible with WebDriver, which means that migrating can often be as simple as importing `webdriverplus` in place of `webdriver`.

So you'd replace your existing import:

```
from selenium import webdriver
```

with this:

```
import webdriverplus as webdriver
```

Todo

List incompatible changes. Explain relation between base classes in each.

Note: If you're migrating from `webdriver` to `webdriverplus`, please consider making a note of any issues you come across, and letting me know.

Contributing

WebDriver Plus is hosted on GitHub, and is currently in pre-release.

Contributions and bug reports are always welcome.

Requirements

If you are using a git clone of the project, rather than a `pip install`, you'll need to make sure you've installed `selenium`.

```
pip install selenium
```

Running the Tests

```
./runtests.py
```

There are also some slower tests which are not run by default. To include these tests:

```
./runtests.py --all
```

Building the Docs

```
./makedocs.sh
```