holmium.core Documentation

Release 0.8.5-dirty

Ali-Akber Saifee

September 06, 2016

Contents

CHAPTER 1

Introduction

holmium.core provides utilities for simplifying the creation and maintenance of tests that rely on Selenium.

Nothing beats an example. Conventionally automated tests integrating with python-selenium are written similarly to the following code block (using seleniumhq.org).

```
import selenium.webdriver
import unittest


class SeleniumHQTest(unittest.TestCase):
    def setUp(self):
        self.driver = selenium.webdriver.Firefox()
        self.url = "http://seleniumhq.org"

    def test_header_links(self):
        self.driver.get(self.url)
        elements = self.driver.find_elements_by_css_selector("div#header ul>li")
        self.assertTrue(len(elements) > 0)
        for element in elements:
            self.assertTrue(element.is_displayed())
        expected_link_list = ["Projects", "Download", "Documentation",
                              "Support", "About"]
        actual_link_list = [el.text for el in elements]
        self.assertEquals(sorted(expected_link_list), sorted(actual_link_list))

    def test_about_selenium_heading(self):
        self.driver.get(self.url)
        about_link = self.driver.find_element_by_css_selector(
            "div#header ul>li#menu_about>a"
        )
        about_link.click()
        heading = self.driver.find_element_by_css_selector("#mainContent>h2")
        self.assertEquals(heading.text, "About Selenium")

    def tearDown(self):
        if self.driver:
            self.driver.quit()


if __name__ == "__main__":
    unittest.main()
```

The above example does what most selenium tests do:

- initialize a webdriver upon setUp
- query for one or more web elements using either class name, id, css_selector or xpath
- assert on the number of occurrences / value of certain elements.
- tear down the webdriver after each test case

It suffers from the typical web development problem of coupling the test case with the HTML plumbing of the page its testing rather than the functionality its meant to exercise. The concept of PageObjects reduces this coupling and allow for test authors to separate the layout of the page under test and the functional behavior being tested. This separation also results in more maintainable test code (*i.e. if an element name changes - all tests don't have to be updated, just the PageObject*).

Lets take the above test case for a spin with holmium. Take note of the following:

- The initialization and reset of the webdriver is delegated to the TestCase base class (*alternatively the class could subclass unittest.TestCase and be run with the holmium nose plugin*).
- the page elements are accessed in the test only via Element & ElementMap.

```
from holmium.core import TestCase, Page, Element, Locators, ElementMap
import unittest


class SeleniumHQPage(Page):
    nav_links = ElementMap(Locators.CSS_SELECTOR
        , "div#header ul>li"
        , key=lambda element: element.find_element_by_tag_name("a").text
        , value=lambda element: element.find_element_by_tag_name("a")
    )

    header_text = Element(Locators.CSS_SELECTOR, "#mainContent>h2")


class SeleniumHQTest(TestCase):
    def setUp(self):
        self.page = SeleniumHQPage(self.driver, "http://seleniumhq.org")

    def test_header_links(self):
        self.assertTrue(len(self.page.nav_links) > 0)
        self.assertElementsDisplayed(self.page.nav_links)
        self.assertEquals(
            sorted(
                ["Projects", "Download", "Documentation", "Support", "About"]
            )
            , sorted(self.page.nav_links.keys()))

    def test_about_selenium_heading(self):
        self.page.nav_links["About"].click()
        self.assertElementTextEqual(self.page.header_text, "About Selenium")


if __name__ == "__main__":
    unittest.main()
```

Which can then be executed in a few different ways as shown below.

```
# if using TestCase as the base class run as:
HO_BROWSER=firefox nosetests test_selenium_hq.py
# or..
```

```
HO_BROWSER=firefox python test_selenium_hq.py
# if using unittest.TestCase as the base class run as:
nosetests test_selenium_hq.py --with-holmium --holmium-browser=firefox
```

CHAPTER 2

Feature Summary

- Automatic provisioning and configuration of webdriver instances based either on environment variables or nosetest arguments. (Unit test integration)
- Shorthand assertions for web pages (TestCase)
- Declarative model for defining pages, sections, page elements and element collections (Page Objects)
- Built in cucumber step definitions for accessing and navigating pages (Cucumber Features)

2.1 Building PageObjects

2.1.1 Overview

A typical PageObject built with *holmium.core* has the following composition:

- ***Page***
  - *Element*
  - *Elements*
  - *ElementMap*
  - ***Section***
    - *Element*
    - *Elements*
    - *ElementMap*
  - ***Sections***
    - *Element*
    - *Elements*
    - *ElementMap*

A Page is initalized with a `selenium.webdriver.remote.webdriver.WebDriver` instance and can take some optional arguments.

```
class MyPage(Page):
    pass

driver = selenium.webdriver.Firefox()
```

```
p = MyPage(driver)
p = MyPage(driver, url = "http://www.google.com")
p = MyPage(driver, url = "http://www.google.com", iframe = "#frame")
```

Providing the `url` argument will result in the driver navigating to the `url` when the *Page* is initialized. The `iframe` argument forces an invocation of `selenium.webdriver.remote.webdriver.WebDriver.switch_to_frame()` everytime an element in the *Page* is accessed.

The `webdriver` that is supplied to a *Page* is used when looking up any *Element*, *Elements* or *ElementMap* that is declared as a static member.

To understand the wiring between a *Page* and its elements try out the example below in a python repl.

```
from holmium.core import Page, Element, Elements, ElementMap, Locators
import selenium.webdriver
driver = selenium.webdriver.Firefox()
class GooglePage(Page):
    search_box = Element( Locators.NAME, "q", timeout = 1)
    google_footer = ElementMap ( Locators.CSS_SELECTOR, "#fsl>a" , timeout = 1 )

g = GooglePage(driver, url="http://www.google.ca")
g.search_box
# <selenium.webdriver.remote.webelement.WebElement object at 0x10b50e450>
g.google_footer
# OrderedDict([(u'Advertising', <selenium.webdriver.remote.webelement.WebElement object at 0x10b35f2
g.google_footer["About"]
# <selenium.webdriver.remote.webelement.WebElement object at 0x10b35f450>
g.google_footer["About"].get_attribute("href")
# u'http://www.google.ca/intl/en/about.html?fg=1'
driver.get("http://www.google.co.tz")
g.google_footer["Kuhusu Google"].get_attribute("href")
# u'https://www.google.co.tz/intl/sw/about.html?fg=1'
```

Both the element `search_box` and the collection of footer links `google_footer` are looked up using the driver that was passed into the `GooglePage` instance.

2.1.2 Sections

Section objects can be used to further encapsulate blocks of page logic that may either be reusable between different pages or accessed from within different parts of the page in a similar manner. Examples of such usecases are menus, footers and collections that may not follow a standard list or map formation.

Take for example a page with the following structure.

```
from holmium.core import Page, Section, Element, Elements, ElementMap, Locators
import selenium.webdriver

headlines_snippet = """
<html>
    <body>
        <div class='header'>
            <h1>Headlines</h1>
            <h2>Breaking news!!</h2>
        </div>
        <div class='news_section'>
            <ul>
                <li>
```

```
                    <div class='heading'>Big News!!!</div>
                    <div class='content'>Just kidding</div>
                </li>
                <li>
                    <div class='heading'>Other Big News!!!</div>
                    <div class='content'>Again, just kidding</div>
                </li>
            </ul>
        </div>
    </body>
</html>"""
sports_snippet = """
<html>
    <body>
        <div class='header'>
            <h1>Sports news</h1>
            <h2>Breaking news!!</h2>
        </div>
        <table class="events">
            <tr>
                <td class='sport'>Soccer</td>
                <td class='status'>World cup</td>
            </tr>
            <tr>
                <td class='sport'>Cricket</td>
                <td class='status'>League matches</td>
            </tr>
        </table>
        <div class='news_section'>
            <ul>
                <li>
                    <div class='heading'>Soccer worldcup finals!!!</div>
                    <div class='content'>I'm running out of meaningful snippets</div>
                </li>
                <li>
                    <div class='heading'>Cricket league matches</div>
                    <div class='content'>I'm definitely out.</div>
                </li>
            </ul>
        </div>
    </body>
</html>"""

class Heading(Section):
    main = Element( Locators.CSS_SELECTOR, "h1")
    sub = Element( Locators.CSS_SELECTOR, "h2")

class NewsSection(Section):
    articles = ElementMap( Locators.CSS_SELECTOR, "ul>li"
                           , key=lambda el: el.find_element_by_class_name('heading').text
                           , value=lambda el: el.find_element_by_class_name('content').text
                           )

class SportsEventsSection(Section):
    events = ElementMap( Locators.CSS_SELECTOR, "tr"
                           , key=lambda el: el.find_element_by_class_name('sport').text
                           , value=lambda el: el.find_element_by_class_name('status').text
                           )
```

```
class NewsPage(Page):
    heading = Heading(Locators.CLASS_NAME, "header")
    news_section = NewsSection(Locators.CLASS_NAME, "news_section")

class HeadlinePage(NewsPage):
    pass

class SportsPage(NewsPage):
    sports_events = SportsEventsSection(Locators.CLASS_NAME, "events")

driver = selenium.webdriver.Firefox()
open("/var/tmp/headlines.html","w").write(headlines_snippet)
open("/var/tmp/sports.html","w").write(sports_snippet)

headlines = HeadlinePage(driver, "file:///var/tmp/headlines.html")
print headlines.news_section.articles["Big News!!!"]
print headlines.heading.main.text

sports = SportsPage(driver, "file:///var/tmp/sports.html")
print sports.heading.main.text
print sports.news_section.articles["Soccer worldcup finals!!!"]
print sports.sports_events.events["Cricket"]
```

Though there are two different pages being accessed, they follow a similar structure and the `news_section` and `header` parts can be encapsulated into a common *`Section`*. Though the `events` section in the sports page isn't used anywhere else - it still makes it clearer to define it as a *`Section`* to separate its logic from the main `SportsPage`.

There may be other usecases where *`Section`* objects may be used to represent complex objects within a page that appear repeatedly in a list like manner. To reduce the duplication of specifying *`Section`* objects repeatedly in a *`Page`* a *`Sections`* object may be used to obtain an iterable view of all matched *`Section`* objects.

Warning: Though one could be inclined to treat *`Sections`* as any other collection please only use them as an iterable or do indexed access directly on the *`Sections`* object. Trying to cast a *`Sections`* property into a list is not supported.

```
from holmium.core import Page, Sections, Element, Elements, ElementMap, Locators
import selenium.webdriver

page_snippet = """
<html>
    <body>
        <div class='thought'>
            <div class='author'>
                <span class='user'>John</span>
                <span class='reputation'>1000</span>
            </div>
            <div class='details'>
                <div class='brief'>John's world view</div>
                <div class='full_text'>Sleeping is important</div>
            </div>
        </div>
        <div class='thought'>
            <div class='author'>
                <span class='user'>Jane</span>
                <span class='reputation'>100000000</span>
            </div>
```

```
            <div class='details'>
                <div class='brief'>Jane's world view</div>
                <div class='full_text'>John's world view is not important...</div>
            </div>
        </div>
    </body>
</html>"""

class ThoughtSections(Sections):
    author = Element(Locators.CLASS_NAME , "user")
    brief = Element(Locators.CSS_SELECTOR , "div.details div.brief")
    full_text = Element(Locators.CSS_SELECTOR , "div.details div.full_text")

class MainPage(Page):
    thoughts = ThoughtSections(Locators.CLASS_NAME, "thought")


driver = selenium.webdriver.Firefox()
open("/var/tmp/page.html","w").write(page_snippet)

main_page = MainPage(driver, "file:///var/tmp/page.html")
for thought in main_page.thoughts:
    print thought.author.text
    print thought.brief.text
    print thought.full_text.text
```

2.1.3 Collections

To keep the interaction with collections of elements in a Page readable and logically grouped - it is useful to represent and access such elements in a page the same way as one would a python list or dictionary. The *Elements* and *ElementMap* (which is used in the previous example) can be used to organize elements with either relationship.

Using the table defined in `snippet` below, a Page can be constructed that allows you to access the `value` or `title` of each row either as a list or a dictionary keyed by the `title`.

Take note of the differences in construction of `element_values` and `element_titles`. Since `element_values` does not provide a lookup function via the `value` argument, the element returned is a pure selenium `selenium.webdriver.remote.webelement.WebElement`. In the case of `element_titles` the lookup function extracts the text attribute of the element. The same type of lookup functions are used in `element_map` to create the key/value pairs.

```
snippet = """
<html>
<body>
<table>
    <tr>
        <td class='title'>title 1</td>
        <td class='value'>value one</td>
    </tr>
    <tr>
        <td class='title'>title 2</td>
        <td class='value'>value two</td>
    </tr>
</table>
<body>
</page>
"""
```

```
from holmium.core import Page,Elements,ElementMap,Locators
import selenium.webdriver

class Trivial(Page):
    element_values = Elements(Locators.CSS_SELECTOR
                              , "tr>td[class='value']" )
    element_titles = Elements(Locators.CSS_SELECTOR
                              , "tr"
                              , value=lambda el: el.find_element_by_css_selector("td[class='value']").t
    element_map = ElementMap(Locators.CSS_SELECTOR
                              , "tr"
                              , key=lambda el: el.find_element_by_css_selector("td[class='title']").tex
                              , value=lambda el: el.find_element_by_css_selector("td[class='value']").t

driver = selenium.webdriver.Firefox()
t = Trivial(driver)
open("/var/tmp/test.html","w").write(snippet)
driver.get("file:///var/tmp/test.html")
t.element_values[0].text
# u'one'
t.element_titles[0]
# u'1'
t.element_map.keys()
# [u'1', u'2']
t.element_map["1"]
# u'one'
```

2.1.4 Conditions

Changes to the dom, such as element visibility and content, are expected for any non-trivial web page. Page elements *Element*, *Elements*, and *ElementMap* accept a keyword argument `filter_by`: a callable that expects a `selenium.webdriver.remote.webelement.WebElement` and is expected to return `True/False`. In *Elements* and *ElementMap*, only elements that meet the `filter_by` condition will be included in the collection. If the `filter_by` condition is not met in *Element*, `None` will be returned. Some common conditions are provided:

class `holmium.core.conditions.`**VISIBLE**
checks if the element is visible

class `holmium.core.conditions.`**INVISIBLE**
checks if the element is invisible

class `holmium.core.conditions.`**MATCHES_TEXT**(*expr*)
checks if the element's text matches the provided regular expression.

These conditions can also be used to place explicit waits on elements expected to change asynchronously by pairing the `only_if` keyword argument with `timeout`. In *Element*, `only_if` is also a callable that expects a `selenium.webdriver.remote.webelement.WebElement` and is expected to return `True/False`.

Sample

```
from holmium.core import conditions, Page, Element, Locators

class MyPage(Page):
    required_element = Element(Locators.CLASS_NAME, "main_element",
```

```
                                only_if=conditions.VISIBLE(),
                                timeout = 5)
    delayed_element = Element(Locators.CLASS_NAME, "text_element",
                                only_if=conditions.MATCHES_TEXT('^ready.*'),
                                timeout = 5)
```

In the above example, `required_element` will return `None` unless it is displayed. The 5 second timeout will take effect every time the element is accessed. Similarly, `delayed_element` will return `None` until the text of the element matches a string that starts with `ready`.

`only_if` and `timeout` explicit waits can also be used with *Elements* and *ElementMap*. In these element collections, `only_if` expects a list of `selenium.webdriver.remote.webelement.WebElement` items and is also expected to return `True/False`. These common conditions for use with element collections are provided:

class holmium.core.conditions.**ANY**(*condition*)
checks if any of the elements in the collection match the condition.

class holmium.core.conditions.**ALL**(*condition*)
checks if all of the elements in the collection match the condition.

You can build your own condition objects by subclassing *conditions.BaseCondition* and implementing the *conditions.BaseCondition.evaluate()* method.

Context Managers

Conditions can also be used as context managers in cases where condition parameters are not known at page object declaration time. For example:

```
from holmium.core import Page, ElementMap
from holmium.core.conditions import ANY, MATCHES_TEXT

class MyPage(Page):
    dynamic_element_collection = ElementMap(Locators.CLASS_NAME, "dynamic", timeout = 5)

    def get_element(self, name):
        with ANY(MATCHES_TEXT(name)):
            return self.dynamic_element_collection[name]
```

2.1.5 Page Facets

Beyond elements maintained by a page, there are other characteristics that can define the behavior of a Page or Section. Holmium allows you to decorate a page with a *facets.Facet* which ensures evaluation of the facet before the first access on the Page or Section.

Builtin facets

class holmium.core.facets.**Title**(*required=True*, *debug=False*, ***kwargs*)
enforces the title of the current page.

Parameters

- **title** (*str*) – a regular expression to match the title.
- **debug** (*bool*) – if True a failure to evaluate will not result in an exception, only a log warning

- **required** (*bool*) – if False a failure to evaluate will be treated as a noop.

class holmium.core.facets.**Cookie**(*required=True*, *debug=False*, ***kwargs*)
enforces the existence (and optionally the value) of a cookie.

Parameters

- **name** (*str*) – name of the cookie
- **value** (*dict*) – (optional) dict (or callable) to validate the value of the cookie.
- **debug** (*bool*) – if True a failure to evaluate will not result in an exception, only a log warning
- **required** (*bool*) – if False a failure to evaluate will be treated as a noop.

class holmium.core.facets.**Strict**(*required=True*, *debug=False*, ***kwargs*)
enforces that every element declared in the `Page` or `Section` be present.

Parameters

- **debug** (*bool*) – if True a failure to evaluate will not result in an exception, only a log warning
- **required** (*bool*) – if False a failure to evaluate will be treated as a noop.

class holmium.core.facets.**Defer**(*required=True*, *debug=False*, ***kwargs*)

Parameters

- **page** (holmium.core.Page) – the page object that is expected to be deferred to
- **action** (*function*) – a callable that takes the page object instance as the first argument
- **action_arguments** (*dict*) – (optional) dictionary of arguments to pass to *action*
- **debug** (*bool*) – if True a failure to evaluate will not result in an exception, only a log warning
- **required** (*bool*) – if False a failure to evaluate will be treated as a noop.

For good measure, lowercased aliases are available for builtin facets:

holmium.core.facets.**title**
alias of *Title*

holmium.core.facets.**strict**
alias of *Strict*

holmium.core.facets.**cookie**
alias of *Cookie*

holmium.core.facets.**defer**
alias of *Defer*

Rolling your own

You can create your own facet decorator by subclassing *facets.Facet* and implementing the *facets.Facet.evaluate()* method. Any additional arguments that you want to access during evaluation should be declared as the following class members:

- required arguments as an **__ARGS__** list
- optional arguments as an **__OPTIONS__** dictionary.

You can also declare an **__ALLOW_MUTLIPLE__** property on your facet which will control the expectation from multiple decorations of the same facet type. If set to `False` the last facet decorator applied will be respected (for example as with the *facets.title* facet - for which it only makes sense to respect the last decorator applied).

The example facet below would require that *color* as an argument, and would optionally accept *image*. When the facet is evaluated it would assert on the *background-color* of the body element and optionally, the *background-image*.

```
class background(Facet):
    __ARGS__ = ["color"]
    __OPTIONS__ = {"image": None}
    def validate(self, driver):
        body = driver.find_element_by_tag_name("body")
        assert_equals( self.arguments["color"], body.value_of_css_property("background-color")
        if self.options["image"]:
            assert_equals( self.options["image"], body.value_of_css_property("background-image")
```

The decorater could then be applied as follows

```
@background(color="rgb(255, 255, 255)", image="none")
class Google(Page):
    google_button = Element(Locators.NAME, "btnK")
```

Additionally individual *Element*, *ElementMap* or *Elements* members of a Page or Section can be promoted to a facet by adding the *facet=True* keyword argument. This will ensure that the specified element is **required** at the time of the containers first access.

Sample

```
from holmium.core import facets, Page, Element, Section, Locators

class MySection(Section):
    required_element = Element(Locators.CLASS_NAME, "main_element", facet=True)
    optional_element = Element(Locators.CLASS_NAME, "secondary_element")

@facets.title(title='login page')
class LoginPage(Page):
    def do_login(self, username, password):
        .....

@facets.cookie(name="session")
@facets.defer(page=LoginPage, action=LoginPage.do_login, action_arguments= {"username":"ali", "passwo
class ContentPage(Page):
    section = MySection(Locators.ID, "main-section")
```

To understand how the facets are evaluated, consider the following code path

```
from selenium import webdriver

driver = webdriver.Firefox()

p = ContentPage(driver, "http://localhost/content")
assert p.section.optional_element != None
```

The chain of execution when calling *p.section.required_element* is as follows

- check defer to *LoginPage*
- check *title* of *LoginPage*
- perform *do_login*

- check *cookie* of *ContentPage*
- check *required* element exists in *MySection*
- return *optional_element*

2.1.6 Customizing page elements

To further customize domain / page specific behaviors of certain web elements, the *ElementEnhancer* base class can be extended to hijack `selenium.webdriver.remote.webelement.WebElement`. The located web element is made available to the subclass as `self.element`.

In the sample below, the `SelectEnhancer` enhancer will be used to hijack any web element that has the tag name *select*. All properties and methods exposed by the `selenium.webdriver.remote.webelement.WebElement` object will still be accessible, and extra methods/properties (such as 'options') will be added on. You can register your own *ElementEnhancer* via a call to *register_enhancer()* and subsequently reset them via a call to *reset_enhancers()*.

By default, holmium only installs a SelectEnhancer that shadows `selenium.webdriver.support.select.Select`.

```
class SelectEnhancer(ElementEnhancer):
    __TAG__ = "select"
    @property
    def options(self):
        return self.element.find_elements_by_tag_name("option")

    def has_option(self, option_name):
        return any([k.text == option_name for k in self.options])

holmium.core.register_enhancer(SelectEnhancer)
```

2.1.7 More Examples

google search

```
import unittest
import selenium.webdriver
from holmium.core import Page, Element, Elements, Locators, ElementMap


class GoogleMain(Page):
    search_box = Element(Locators.NAME, "q", timeout=2)
    google_buttons = ElementMap(
        Locators.CSS_SELECTOR, ".jsb input", timeout=2,
        key=lambda e: e.get_attribute("value")
    )
    search_results = Elements(
        Locators.CSS_SELECTOR, "div.g>div.rc", timeout=2,
        value = lambda el : {
            "link": el.find_element_by_css_selector("h3.r>a").get_attribute("href"),
            "title": el.find_element_by_css_selector("h3.r>a").text
        }
    )

    def search ( self, query ):
        self.google_buttons["Google Search"].click() # self.google_buttons behaves just like a dictio
```

```
        self.search_box.clear() # self.search_box is now evaluated directly to a WebElement
        self.search_box.send_keys(query)
        self.search_box.submit()


class TextSearchTest(unittest.TestCase):
    def setUp(self):
        self.driver = selenium.webdriver.Firefox()
        self.page = GoogleMain(self.driver, "http://www.google.com")

    def test_text_search(self):
        self.assertTrue(len(self.page.search("selenium").search_results) > 0)

    def test_text_search_first_result(self):
        self.page.search("selenium") # execute the page object method search
        self.assertEquals(
            self.page.search_results[0]["title"],
            u"Selenium - Web Browser Automation"
        )
        self.assertEquals(
            self.page.search_results[0]["link"],
            u"http://www.seleniumhq.org/"
        )

    def tearDown(self):
        self.driver.quit()
```

Wikipedia text search example

```
import unittest

import selenium.webdriver

from holmium.core import Page, Element, Locators, ElementMap


class WikiPedia(Page):
    languages = ElementMap(
        Locators.CLASS_NAME, "central-featured-lang"
        , key = lambda el:el.get_attribute("lang")
        , value = lambda el: el.find_element_by_tag_name("a")
    )
    search_box = Element(
        Locators.CSS_SELECTOR, "input#searchInput"
    )
    article_title = Element(
        Locators.CSS_SELECTOR, "h1#firstHeading"
    )
    search_results = ElementMap(
        Locators.CSS_SELECTOR, "div.mw-search-result-heading>a"
    )

    def search(self, query):
        self.search_box.clear()
        self.search_box.send_keys( query )
        self.search_box.submit()
```

```
class TextSearchArticle(unittest.TestCase):
    def setUp(self):
        self.driver = selenium.webdriver.Firefox()
        self.page = WikiPedia(self.driver, "http://wikipedia.org")

    def test_text_search_alllangs(self):
        for language in self.page.languages:
            self.page.go_home().languages[language].click()
            self.assertTrue(
                self.page.search("google").article_title.text.startswith("Google"),
                language
            )

    def tearDown(self):
        self.driver.quit()
```

2.2 Unit Test Integration

holmium.core

Holmium provides two utilities to ease integration with automated tests.

2.2.1 The TestCase base class

This base class extends `unittest.TestCase` and adds the following functionality:

- automatically provision any number of selenium webdrivers (`driver`/`drivers`) to the testcase based on the environment variable HO_BROWSER. The webdrivers are generated lazily upon access of the `driver` or `drivers` attributes.
- A remote selenium server can also be used by setting the value of HO_REMOTE to the fully qualified url to the selenium server (e.g. http://localhost:4444/wd/hub)
- clears the browser cookies between each test case
- quits the driver(s) at the end of the test class or at the end of the test run (depending on the configuration).
- extra assertion methods relevant to `selenium.webdriver.remote.webelement.WebElement` (refer to *`TestCase`*)

The following environment variables are respected by *`TestCase`* when `unittest.TestCase.setUpClass()` is executed.

Environment Variables

variable	description
`HO_BROWSER`	one of chrome,firefox,opera,safari, ie,phantomjs,android,iphone or ipad
`HO_REMOTE`	the full qualified url of the selenium server. If not provided the browsers will be attempted to be launched using the built in webdrivers.
`HO_USERAGENT`	useragent to use as an override. only works with firefox & chrome
`HO_IGNORE_SSL_ERRORS`	ignore ssl errors when accessing pages served under untrusted certificates (default False).
`HO_BROWSER_PER_TEST`	if the variable is set the browser is created/destroyed for each test class (default False).

Example test case

```
import unittest
import holmium.core

class SimpleTest(holmium.core.TestCase):
    def setUp(self):
        self.driver.get("http://www.google.com")

    def test_title(self):
        self.assertEquals(self.driver.title, "Google")

if __name__ == "__main__":
    unittest.main()
```

Execution

```
# against the builtin firefox driver
HO_BROWSER=firefox python test_simple.py
# against a firefox instance under a remote selenium server
HO_BROWSER=firefox HO_REMOTE=http://localhost:5555/wd/hub python test_simple.py
```

2.2.2 Plugin for nosetest

The plugin provides a way to configure holmium without the use of environment variables and the *TestCase* class as the base of your test class. adding the `--with-holmium` flag will result in the `driver` and `drivers` attributes to be injected into your test class and be made available during test execution.

Command line options

option	description
`--with-holmium`	to enable the use of the holmium plugin
`--holmium-browser`	one of chrome,firefox,safari,opera,ie,phantomjs,android,iphone or ipad
`--holmium-remote`	the full qualified url of the selenium server. If not provided the browsers will be attempted to be launched using the built in webdrivers.
`--holmium-useragent`	useragent to use as an override. only works with firefox & chrome
`--holmium-capabilities`	a json dictionary string or a fully qualified path to a json file of extra desired capabilities to pass to the webdriver.
`--holmium-ignore-ssl`	ignore ssl errors when accessing pages served under untrusted certificates (default False).
`--holmium-browser-per-test`	creates/destroys the browser for each test case (default False).

Note: *Environment Variables* may be used cooperatively when using the nose plugin, however, the options passed to nose will take precedence.

Example test case

```
import unittest

class SimpleTest(unittest.TestCase):
```

```
    def setUp(self):
        self.driver.get("http://www.google.com")

    def test_title(self):
        self.assertEquals(self.driver.title, "Google")
```

Execution

```
# against the builtin firefox driver
nosetest test_simple.py --with-holmium --holmium-browser=firefox
# against a firefox instance under a remote selenium server
nosetest test_simple.py --with-holmium --holmium-browser=firefox --holmium-remote=http://localhost:55
```

2.3 Cucumber style Features

holmium.core

If cucumber style feature definitions are your cup of tea, holmium provides integration with fresher. The following features are enabled if both nose plugins are in use at the same time.

- Automatic provisioning of webdriver
- Common *Step definitions* for accessing page objects and page elements
- Access to holmium config (`holmium.core.Config`) variables via jinja2 templates in scenario steps.

2.3.1 Setup

Before reading this section, I would strongly recommend browsing through the documentation for fresher. If you're already familiar - read on..

Holmium+Fresher features require the same structure as regular fresher features, i.e.

```
my_feature/
|- config.py
|- awesome.feature
|- lame.feature
|- steps.py
```

steps.py

To enable the scenario step expressions that are bundled with holmium, your *steps* module will require registration of these steps via the following snippet

```
from holmium.core.cucumber import init_steps
init_steps()
```

Subsequently, you will also have to ensure that the `Page` objects that you want to drive in the scenarios have been imported in the scope of your *steps* module. Once that is done, you can move on to writing your features.

2.3.2 Step definitions

Browser control

```
When I access the {{url}}
When I go back
And I go forward
Then I should see the title {{title}}
```

Page access

```
When I access the {{PageObject}} at {{url}}
```

Element visibility/content

```
Then the element {{Element}} should be visible
Then the element {{Element}} should have text {{text}}
Then the {{key/attr/index}} in {{Elements/ElementMap/Section}} should be visible
Then the {{key/attr/index}} in {{Elements/ElementMap/Section}} should have text {{text}}
Then the {{key/attr/index}} item for the {{key/index}} item in {{Sections}} should be visible
Then the {{key/attr/index}} item for the {{index}} item in {{Sections}} should have text {{text}}
```

Element interaction

```
When I type {{text}} in element {{Element}}
When I type {{text}} in the {{key/attr/index}} item in {{Elements/ElementMap/Section}}
When I press enter in element {{Element}}
When I press enter in the {{key/attr/index} item in element {{Element}}
When I click element {{Element}}
When I click the {{key/attr/index}} item in {{Elements/ElementMap/Section}}
When I wait for {{seconds}} second(s)
```

Page method execution

```
When I perform {{method_name}} on the page {{PageObject}}
When I perform {{method_name}} on the page {{PageObject}} with arguments {{comma_separated_args}}
```

2.3.3 Full example

With the following directory structure and content

```
google_feature
|- page.py
|- search.feature
|- steps.py
```

page.py

```
from holmium.core import Page, Sections, Locators, Element
from selenium.webdriver.common.keys import Keys

class Results(Sections):
    link = Element(Locators.CSS_SELECTOR, "h3.r")
    description = Element(Locators.CLASS_NAME, "st")

class GooglePage(Page):
    search_box = Element(Locators.NAME, "q")
    search_results = Results(Locators.CSS_SELECTOR, "li.g", timeout=5)
    next = Element(Locators.LINK_TEXT, "Next")
    def search(self, text):
        self.search_box.send_keys(text)
        self.search_box.send_keys(Keys.ENTER)
```

steps.py

```
from holmium.core.cucumber import init_steps
from page import GooglePage
init_steps()
```

google.feature

```
Feature: Google search
    Scenario: Search for selenium
        When I access the page GooglePage at url http://www.google.com
        Then I should see the title Google
        When I perform search with arguments selenium python documentation
        Then the element search_results should be visible
        And the element search_results should have 10 items
        And the link of the first item in search_results should have the text Selenium with Python --
        And the element next should have the text Next
        When I click the element next
        And I go back
        Then the link of the first item in search_results should have the text Selenium with Python -
```

Execution

All the nose plugin options defined at *Plugin for nosetest* can be used here.

```
nosetests --with-fresher --with-holmium --holmium-browser=firefox google_feature
```

2.4 Holmium API documentation

2.4.1 Public classes

Page Objects & Friends

class holmium.core.**Page**(*driver*, *url=None*, *iframe=None*)

Base class for all page objects to extend from. void Instance methods implemented by subclasses are provisioned with fluent wrappers to facilitate with writing code such as:

```
class Google(Page):
    def query(self):
        ....

    def submit(self):
        ....

    def get_results(self):
        ....

assert len(Google().query("page objects").submit().get_results()) > 0
```

class holmium.core.**Section**(*locator_type*, *query_string*, *iframe=None*, *timeout=0*)

Base class to encapsulate reusable page sections:

```
class MySection(Section):
    things = Elements( .... )

class MyPage(Page):
    section_1 =  MySection(Locators.CLASS_NAME, "section")
    section_2 =  MySection(Locators.ID, "unique_section")
```

class holmium.core.**Sections**(*locator_type*, *query_string*, *iframe=None*, *timeout=0*)

Base class for an Iterable view of a collection of *holmium.core.Section* objects.

class holmium.core.**Element**(*locator_type*, *query_string*, *base_element=None*, *timeout=0*, *value=<function <lambda>>*, *only_if=<function <lambda>>*, *facet=False*, *filter_by=<function <lambda>>*)

Utility to get a selenium.webdriver.remote.webelement.WebElement by querying via one of *holmium.core.Locators*

Parameters

- **locator_type** (holmium.core.Locators) – selenium locator to use when locating the element
- **query_string** (*str*) – the value to pass to the locator
- **base_element** (holmium.core.Element) – a reference to another element under which to locate this element.
- **timeout** (*int*) – time to implicitely wait for the element
- **value** (*lambda*) – transform function for the value of the element. The located selenium.webdriver.remote.webelement.WebElement instance is passed as the only argument to the function.
- **only_if** (*function*) – extra validation function that is called repeatedly until timeout elapses. If not provided the default function used checks that the element is present.

The located selenium.webdriver.remote.webelement.WebElement instance is passed as the only argument to the function.

- **facet** (*bool*) – flag to treat this element as a facet.
- **filter_by** (*function*) – condition function that determines if the located selenium.webdriver.remote.webelement.WebElement, the only argument passed to the function, should be returned. If not provided, the default function used checks that the element is present.

class holmium.core.**Elements**(*locator_type*, *query_string=None*, *base_element=None*, *timeout=0*, *value=<function <lambda>>*, *only_if=<function <lambda>>*, *facet=False*, *filter_by=<function <lambda>>*)

Utility to get a collection of selenium.webdriver.remote.webelement.WebElement objects by querying via one of *holmium.core.Locators*

Parameters

- **locator_type** (holmium.core.Locators) – selenium locator to use when locating the element
- **query_string** (*str*) – the value to pass to the locator
- **base_element** (holmium.core.Element) – a reference to another element under which to locate this element.
- **timeout** (*int*) – time to implicitely wait for the element
- **value** (*lambda*) – transform function for each element in the collection. The located selenium.webdriver.remote.webelement.WebElement instance is passed as the only argument to the function.
- **only_if** (*function*) – extra validation function that is called repeatedly until timeout elapses. If not provided the default function used checks that the element collection is not empty. The list of located selenium.webdriver.remote.webelement.WebElement instances is passed as the only argument to the function.
- **facet** (*bool*) – flag to treat this element as a facet.
- **filter_by** (*function*) – condition function determines which elements are included in the collection. If not provided the default function used includes all elements identified by query_string. A selenium.webdriver.remote.webelement.WebElement instance is passed as the only argument to the function.

class holmium.core.**ElementMap**(*locator_type*, *query_string=None*, *base_element=None*, *timeout=0*, *key=<function <lambda>>*, *value=<function <lambda>>*, *only_if=<function <lambda>>*, *facet=False*, *filter_by=<function <lambda>>*)

Used to create dynamic dictionaries based on an element locator specified by one of *holmium.core.Locators*.

The wrapped dictionary is an collections.OrderedDict instance.

Parameters

- **locator_type** (holmium.core.Locators) – selenium locator to use when locating the element
- **query_string** (*str*) – the value to pass to the locator

- **base_element** (holmium.core.Element) – a reference to another element under which to locate this element.
- **timeout** (*int*) – time to implicitely wait for the element
- **facet** (*bool*) – flag to treat this element as a facet.
- **key** (*lambda*) – transform function for mapping a key to a WebElement in the collection. The located `selenium.webdriver.remote.webelement.WebElement` instance is passed as the only argument to the function.
- **value** (*lambda*) – transform function for the value when accessed via the key. The located `selenium.webdriver.remote.webelement.WebElement` instance is passed as the only argument to the function.
- **only_if** (*function*) – extra validation function that is called repeatedly until `timeout` elapses. If not provided the default function used checks that the element collection is not empty. The list of located `selenium.webdriver.remote.webelement.WebElement` instances is passed as the only argument to the function.
- **filter_by** (*function*) – condition function determines which elements are included in the collection. If not provided the default function used includes all elements identified by `query_string`. A `selenium.webdriver.remote.webelement.WebElement` instance is passed as the only argument to the function.

class holmium.core.**Locators**

proxy class to access locator types

CLASS_NAME = 'class name'

CSS_SELECTOR = 'css selector'

ID = 'id'

LINK_TEXT = 'link text'

NAME = 'name'

PARTIAL_LINK_TEXT = 'partial link text'

TAG_NAME = 'tag name'

XPATH = 'xpath'

Deprecated Classes

Deprecated since version 0.3.

Warning: Earlier versions of *holmium.core* used rather verbose names for Page objects and elements. They have been removed as of version 0.4 and accessing them will raise a `exceptions.SyntaxError`.

class holmium.core.**PageObject**(*driver*, *url=None*, *iframe=None*)

Deprecated alias for *Page*

class holmium.core.**PageElement**(*locator_type*, *query_string*, *base_element=None*, *timeout=0*, *value=<function <lambda>>*, *only_if=<function <lambda>>*, *facet=False*, *filter_by=<function <lambda>>*)

Deprecated alias for *Element*

class holmium.core.**PageElements**(*locator_type*, *query_string=None*, *base_element=None*, *timeout=0*, *value=<function <lambda>>*, *only_if=<function <lambda>>*, *facet=False*, *filter_by=<function <lambda>>*)
Deprecated alias for *Elements*

class holmium.core.**PageElementMap**(*locator_type*, *query_string=None*, *base_element=None*, *timeout=0*, *key=<function <lambda>>*, *value=<function <lambda>>*, *only_if=<function <lambda>>*, *facet=False*, *filter_by=<function <lambda>>*)
Deprecated alias for *ElementMap*

class holmium.core.**HolmiumTestCase**(*methodName='runTest'*)
Deprecated alias for *TestCase*

Utilities

class holmium.core.**TestCase**(*methodName='runTest'*)
Base class for creating test classes for writing holmium driven test cases. More details can be found at *The TestCase base class*

assertConditionWithWait(*driver*, *condition*, *timeout=0*, *msg=None*, *ignored_exceptions=None*)
Fail if the condition specified does not hold for the element within the specified timeout

Parameters

- **driver** – the selenium driver
- **condition** – an instance of selenium.webdriver.support.expected_conditions
- **msg** – the failure message when timeout, could be a string or a callable without arguments that returns a string
- **timeout** – to be passed to *selenium.webdriver.support.wait.WebDriverWait*
- **ignored_exceptions** – to be passed to *selenium.webdriver.support.wait.WebDriverWait*

assertElementAttributeEqual(*element*, *key*, *value*, *msg=None*)
Fail if the element does not have the specified attribute value

assertElementCSS(*element*, *css_property*, *value*, *msg=None*)
Fail if the element does not exhibit the correct css property value. The value of the elements css property is the one returned by selenium.webdriver.remote.webelement.WebElement.value_of_css_property()

assertElementDisplayed(*element*, *msg=None*)
Fail if the element is not visible

assertElementSize(*element*, *width*, *height*, *msg=None*)
Fail if the element size does not match the provided values

assertElementTextEqual(*element*, *text*, *msg=None*)
Fail if the text attribute of the element does not match

assertElementsDisplayed(*elements*, *msg=None*)
Fail if any of the elements in the element collection are not visible

classmethod setUpClass()
prepare the driver initialization based on the environment variables that have been set. The driver is not actually initialized until the test itself actually refers to it via *self.driver*.

tearDown()
clear the cookies on the driver after each test

classmethod tearDownClass()
quit the driver after the test run (or after all the test methods in the class have finished if HO_BROWSER_PER_TEST is set).

class holmium.core.**Config**(*dct*, *environment={'holmium': {'environment': 'development'}}*)

Dictionary like helper class for maintaining test data configurations per environment.

holmium.core.TestCase and *holmium.core.HolmiumNose* both look for either a config.json or config.py file in the same directory as the test file, and will make a config object available to the test case instance.

The *holmium.core.Config* object is aware of the environment (specified with --holmium-env when using nose or HO_ENV as an environment variable) and will return the config variable from that environment or from the *default* key.

Values in the config file can use jinja2.Template templates to access either values from itself, environment variables or a select magic holmium variables: holmium.environment, holmium.browser, holmium.user_agent and holmium.remote.

Example config structure (which uses a magic variable holmium.environment and an environment variable $PATH).

JSON

```
{
        'default': {  'path':"{{PATH}}"
                    , 'login_url': '{{url}}/{{holmium.environment}}/login'}
                    , 'username' : '{{holmium.environment}}user'}
        ,'production': {'url':'http://prod.com'
                        ,'password': 'sekret'}
        ,'development': {'url':'http://dev.com'
                        ,'password': 'password'}
}
```

Python

```
config = {
    {
        'default': { 'path':"{{PATH}}"
                    , 'login_url': '{{url}}/{{holmium.environment}}/login'}
                    , 'username' : '{{holmium.environment}}user'}
        ,'production': {'url':'http://prod.com'
                        , 'password': 'sekret'}
        ,'development': {'url':'http://dev.com'
                        , 'password': 'password'}
    }
}
```

When accessing self.config within a test, due to the default:

- **self.config['path'] will always return the value of the environment** variable *PATH*,
- self.config['password'] will always return 'sekret'

if HO_ENV or --holmium-env are production:

- self.config['username'] will return productionuser
- self.config['password'] will return sekret

- `self.config['login_url']` will return `http://prod.com/production/login`

if `HO_ENV` or `--holmium-env` are `development`:

- `self.config['username']` will return `developmentuser`
- `self.config['password']` will return `password`
- `self.config['login_url']` will return `http://dev.com/development/login`

class holmium.core.**ElementEnhancer**(*element*)
base class for implementing custom element enhancers to add functionality to located webelements based on the element type (tag name)

classmethod matches(*element*)
class method to verify that this enhancer is appropriate for the provided webelement

holmium.core.**register_enhancer**(*enhancer*)
registers a *ElementEnhancer* with the internal lookup

holmium.core.**reset_enhancers**()
resets the state so that any *ElementEnhancer* that was registered via a call to *register_enhancer()* is removed.

2.4.2 Internal Classes

Page Object Helpers

class holmium.core.pageobject.**ElementDict**(*instance*, **args*, ***kwargs*)
proxy to a standard dict which would be stored in a *holmium.core.Page*.

class holmium.core.pageobject.**ElementList**(*instance*, **args*, ***kwargs*)
proxy to a standard list which would be stored in a *holmium.core.Page*.

class holmium.core.pageobject.**ElementGetter**(*locator_type*, *query_string*, *base_element=None*, *timeout=0*, *value=<function <lambda>>*, *only_if=<function <lambda>>*, *facet=False*, *filter_by=<function <lambda>>*)
internal class to encapsulate the logic used by *holmium.core.Element* & *holmium.core.Elements*

classmethod enhance(*element*)
incase a higher level abstraction for a WebElement is available we will use that in Pages. (e.g. a select element is converted into `selenium.webdriver.support.ui.Select`)

root
returns the root webelement

class holmium.core.facets.**FacetCollection**(**a*)
utility collection class for pageobjects to encapsulate facets

append(*item*)
overridden add method to pop the last item if its type does not support multiple facets on the same object.

evaluate_all(*driver*)
iterate over all registered *Facet* objects and validate them

Parameters **driver** (*selenium.webdriver.remote.webdriver.WebDriver*) – the webdriver

type_map
view on the list to help with figuring out if a facet of the same type already exists

class holmium.core.facets.**Facet**(*required=True*, *debug=False*, ***kwargs*)
base class to implement an attribute of a page

evaluate(*driver*)
evaluate whether this facet holds true. Raise an Exception if not.

Parameters **driver** (*selenium.webdriver.remote.webdriver.WebDriver*) – the webdriver

get_name()
returns the class name of the facet

get_parent_name()
returns the class name of the parent

parent_class
returns the parent class

register(*obj*)
registers a *Facet* on an object

Parameters **obj** (holmium.core.facets.Faceted) – the object to register the facet on.

class holmium.core.facets.**Faceted**
mixin for objects that want to have facets registered on them.

evaluate()
evaluates all registered facets (class & instance)

classmethod **get_class_facets**()
returns the facets registered on the class (presumably via a decorator)

get_instance_facets()
returns the facets registered on the instance

exception holmium.core.facets.**FacetError**(*facet*, *exc=None*)
exception raised when a facet has an error or can't complete

Parameters

- **facet** (holmium.core.facets.Facet) – the facet that failed to evaluate
- **exc** (*exceptions.Exception*) – the inner exception that caused the failure

class holmium.core.conditions.**BaseCondition**
base class to implement conditions passed to the only_if parameter of *holmium.core.pageobject.ElementGetter* subclasses.

evaluate(*element*)
abstract method to be implemented by derived classes.

Test configuration / execution

class holmium.core.**HolmiumNose**
nose plugin to allow bootstrapping testcases with a selenium driver

afterTest(*_*)
run after the test completes. kill the browser or just clear cookies depending on config

beforeTest(*test*)
setup the initialization for the driver, and load the config

static **is_freshen_test**(*test*)
checks for tell tale signs of being a fresher test

options(*parser*, *env*)
Register command line options

startTest(*test*)
before each test set the self.driver/self.drivers attribute for the test.

class holmium.core.config.**HolmiumConfig**(*browser*, *remote*, *capabilities*, *user_agent*, *environment*, *ignore_ssl*, *fresh_instance*)
utility class for storing holmium configuration options strictly. The class behaves like a dictionary after construction with the additional behavior that any attributes set on it are available as keys in the dictionary and vice versa.

2.5 Development

2.5.1 Contributors

- , Testing and initial integration and evaluation.
- , original Holmium logo
- , Section(s) design.
- , Evaluation and help designing Facets
- Nathan Jones, Bug fixes
- Lata Suresh, Bug fixes
- Jeffry Pincus, *filter_by* conditions
- Chow Loong Jin, Bug Fixes

2.5.2 Project Resources

Continuous Integration The project is being continuously built with travis against python 2.6 & 2.7 & 3.3.

Code The code is hosted on github.

Bugs, Feature Requests Tracked at the issue tracker.

Questions freenode irc channel

2.5.3 Installation

Note: Holmium is tested and supported on pythons version 2.6, 2.7 & 3.3.

The stable version can be installed either via `pip` or `easy_install`.

```
pip install holmium.core
# or
easy_install holmium.core
```

To use holmium.core directly from source the preferred method is to use the develop mode. This will make *holmium.core* available on your *PATH*, but will point to the checkout. Any updates made in the checkout will be available in the *installed* version.

```
git clone git@github.com:alisaifee/holmium.core
cd holmium.core
sudo python setup.py develop
```

2.5.4 Tests

holmium.core uses `nosetests` for running its tests. You will also need `phantomjs` installed to run certain tests that make more sense without mocking. For instructions on installing `phantomjs` go to the phantomjs download page.

```
cd holmium.core
nosetests --with-coverage --cover-html --cover-erase --cover-package=holmium.core
```

2.6 History

2.6.1 0.8.5 2016-09-06

- Extra options for assertConditionWithWait #42

2.6.2 0.8.4 2016-09-01

- Bug fix: assertConditionWithWait #40

2.6.3 0.8.3 2016-08-12

- Bug fix: Fix for IE with remote #38
- Bug fix: StaleElementReferenceException handling #33

2.6.4 0.8.2 2015-12-22

- New filter_by argument that accepts conditions

2.6.5 0.8.1 2015-10-30

- Bug fix: Fix setup requirements for python 3.x #30

2.6.6 0.8 2015-06-07

- No functional Change

2.6.7 0.7.9 2015-05-30

- Bug fix: Support for phantom 1.9.x #29

2.6.8 0.7.8 2014-11-02

- Bug fix: AttributeError when comparing with None #26
- Bug fix: Negative indexing in sections #27

2.6.9 0.7.7 2014-09-05

- Bug fix: IE Driver initialization #24

2.6.10 0.7.6 2014-07-14

- Hot fix: broken installation due to missing requirements

2.6.11 0.7.5 2014-07-14

- Bug fix for `StaleElementReferenceException` in WebDriverWait
- Support for using `holmium.core.coniditions` objects as context managers
- Additional conditions `ANY` and `ALL` for element collections.

2.6.12 0.7.4 2014-04-24

- Bug fix: Sections weren't working for index > 1 #22

2.6.13 0.7.3 2014-03-14

- Add missing timeout from Section

2.6.14 0.7.2 2014-02-22

- exclude packaging tests

2.6.15 0.7.1 2014-02-18

- Fix packaging problem with versioneer

2.6.16 0.7 2014-02-10

- Built-in conditions for explicit waits
- New assertion `assertConditionWithWait`
- Change behavior of `only_if` to not check `is_displayed` by default.
- Tweaks
- Allow passing a filename for nose argument `--holmium-capabilities`
- Change versioning to use versioneer
- Explicit py3k support with six
- Make primitive lists and maps wrapped in pageobjects behave.

2.6.17 0.6.2 2014-01-15

- Bug fix issue 19

2.6.18 0.6.1 2013-12-23

- Bug fix issue 18 for facet clobbering when page inheritance was involved
- Bug fix issue 17 for case of no browser specified
- new assertion for TestCase class : `assertElementAttributeEqual`

2.6.19 0.6 2013-12-14

- Lazy driver initialization. The webdriver is created when the test first accesses it.
- Support for using multiple browsers (drivers) in test cases. The original self.driver is still available along with a self.drivers list which lazily initializes new drivers as they are accessed via index. drivers[0] == driver.
- New environment variable / nose option to force browser(s) to be shutdown and restarted between tests. (it is disabled by default, but cookies are still always cleared between tests)
- New assertions added to the TestCase base class
- Documentation cleanups
- Bug fixes for default timeout/only_if arugment for Element/Elements/ElementMap

2.6.20 0.5.2 2013-12-09

- PyPy support
- Allow customization of WebElements by exposing ElementEnhancer

2.6.21 0.5.1 2013-12-01

- Re-added python 2.6 support

2.6.22 0.5.0 2013-12-01

- Python 3.3 now supported and tested.

2.6.23 0.4.2 2013-12-01

- New parameter **only_if** (callable that accepts the webelement that was found) accepted by Element, Elements, ElementMap that allows for waiting for an element to become valid according to the response of **only_if**. The callable will be checked uptil the timeout parameter set on the Element.

2.6.24 0.4.1 2013-11-29

- Bug fix for config module being reused between test runs.

2.6.25 0.4 2013-11-28

- Old style class names removed (Deprecated old class names)
- Introduced Facets
- Introduced Cucumber Features integration with fresher.
- General refactoring and code cleanup.

2.6.26 0.3.4 2013-11-21

- Added support to ignore ssl certificate errors on chrome, firefox & phantomjs
- code cleanup
- improved test coverage

2.6.27 0.3.3 2013-10-29

- Improved back reference access in Config object by allowing variable references without requiring a prefix of *default* or the environment name. The resolution order is current environment and then default.

  For example, the following config will resolve *login_url* as **http://mysite.com/login** and *profile_url* as **http://mysite.com/profile/prod_user** respectively, when *holmium.environment* is set to **production**

```
config = { "default" : {
                "login_url" : "{{url}}/login"
                , "profile_url":"{{url}}/profiles/{{username}}"}
           , "production": {
                "url": "http://mysite.com"
                , "username":"prod_user"}
         }
```

2.6.28 0.3.2 2013-10-10

- Fluent response from page objects only when page method returns None

2.6.29 0.3.1 2013-09-17

- Allow indexing of Sections objects

2.6.30 0.3 2013-09-16

- Bug Fix for instantiating multiple instances of the same the Page object (https://github.com/alisaifee/holmium.core/issues/4)
- Section object introduced

2.6.31 0.2 2013-09-11

- Deprecated old class names (PageObject, PageElement, PageElements, PageElementMap & HolmiumTestCase)
- Added more tests for holmium.core.TestCase
- New Config object.

2.6.32 0.1.8.4 2013-09-04

- Bug Fix : installation via pip was failing due to missing HISTORY.rst file.

2.6.33 0.1.8.3 2013-08-12

- Bug fix
  - improved error handling and logging for missing/malformed config file.

2.6.34 0.1.8 2013-03-18

- Added iphone/android/phantomjs to supported browsers
- Bug fix
  - fixed phantomjs build in travis

2.7 License

h

A

B

C

D

E

F

G

H

I

L

M

N

O

P

R

S

T

V

X